Mantık ve İspatlar

Önermeler doğru veya yanlış sonuç bildiren ifadelerdir. p, q, r, t, s ile gösterilirler.

1+1=2 (önermedir)

Elazığ Türkiye'nin başkentidir. (önermedir)

Saat kaç? } önerme değildir.
x+1=2 }

p bir önerme olsun. p'nin tersi $\neg p$, $\bar{p}$, p' olarak gösterilir. p'nin değili diye okunur.

p: Bugün cumadır.

$\bar{p}$: Bugün cuma değildir.

p, q bir önermeyse $p \wedge q$'da bir önermedir p.q ile gösterilir.

p, q önerme olsun. $p \vee q$'da bir önermedir p+q ile gösterilir.

$p \to q$ p ise q

$p \to q \not\equiv q \to p$

$p \to q \equiv p' \vee q$

$p \to q \neq q \to p$

$p \to p \equiv 1$

$p \to 1 \equiv 1$

$p \to 0 \equiv p'$

$p \to p' \equiv p'$

Karşıt, Zıt-Pozitif ve Ters

$p \to q$ 'nun zıt pozitifi $\neg p \to \neg q$ dır.

$p \to q$ 'nun karşıtı $q \to p$ 'dir.

$p \to q$ 'nun karşıt tersi $q' \to p'$

ÖR: Aşağıdaki şartlı ifadenin karşıtı, zıt pozitifi ve tersi ne olur?

"Ev sahibi takım yağmur yağdığı zaman kazanır."

$p \to q$

Eğer yağmur yağarsa ev sahibi takım kazanır.

Zıt pozitifi:

"Ev sahibi takım kazanmadıysa yağmur yağmıyordur."

Karşıtı:

"Eğer ev sahibi takım kazandıysa yağmur yağıyordur."

Tersi:

"Eğer yağmıyorsa ev sahibi takım kazanamaz."

Çift Şartlı (Ancak ve Ancak) (Exnor) → Ya da değil

p	q	$p \leftrightarrow q$ (Exnor)	$p \oplus q$ (Exor)
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

Exor ya da (sadece ve sadece'nin tersi)

$p \leftrightarrow p \equiv 1$ $\quad p \leftrightarrow p' \equiv 0$

$p \leftrightarrow 1 \equiv p$ $\quad p \leftrightarrow 0 \equiv p'$

$p \leftrightarrow q \equiv q \leftrightarrow p$

$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$

Operatör Öncelikleri

1) $\neg$
2) $\wedge$
3) $\vee$
4) $\to$
5) $\leftrightarrow$

$p \oplus q \equiv q \oplus p$

$p \oplus p \equiv 0$

$p \oplus p' \equiv 1$

$p \oplus 1 \equiv p'$

$p \oplus 0 \equiv p$

Önerme Denklikleri

Kendisine düşürülen önermelerin doğruluk değerleri ne olursa olsun her zaman doğru olan bir bileşik önermeye her zaman doğru ya da tatoloji denir.

Sonucu her zaman yanlış olan bileşik önermeye çelişki denir.

tatoloji $\equiv 1$ çelişki $\equiv 0$

Ne tatoloji ne de çelişki olmayan bileşik önermeye belirsiz denir

De Morgan Kanunları

$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\quad \neg(p \vee q) \equiv \neg p \wedge \neg q$

$(p \vee q) \vee r \equiv p \vee (q \vee r)$
$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
} Birleşme Kanunları

$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
} Dağılma Kanunları

$p \vee (p \wedge q) \equiv (p \vee p) \wedge (p \vee q) \equiv p$
$p \wedge (p \vee q) \equiv p \wedge p \equiv p$
} Yutma Kanunları

Şart Cümlelerini Sağlayan Mantıksal Denklikler

$p \to q \equiv p' \vee q$ $\quad p \vee q \equiv p' \to q$

$p \to q \equiv q' \to p'$

$p \wedge q \equiv (p \to q') \to (\overline{p \wedge q}) \equiv (\overline{p' \vee q'}) \equiv (p \to q')'$

$(p \to q)' \equiv p \wedge q'$
($p \to q \equiv p' \vee q$)

$p' \vee q \equiv p \wedge q'$

$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \vee (q \to r) \equiv (p \wedge r) \to r$
$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$
$p \leftrightarrow q \equiv p' \leftrightarrow q'$

$p \leftrightarrow q \equiv (p \wedge q) \vee (p' \wedge q')$
$(p \leftrightarrow q)' \equiv p \leftrightarrow q$

ÖR: $P(x)$ "$x>3$" olsun. $P(2)$ ve $P(4)$ doğruluğunu kanıtlayınız.

$P(2) \equiv F$ — 2, 3'den küçük

$P(4) \equiv T$ — 4, 3'den büyük

Evrensel Niceleme

$P(x)$ evrensel nicelemesi tanım bölgesindeki tüm x değerleri için $P(x)$ demektir.

$\forall x P(x)$ gösterimi $P(x)$'in evrensel nicelemesini gösterir.

$\forall \rightarrow$ evresel niceleyici

Varoluşsal Niceleyici

Tanım bölgesinde $P(x)$ olacak şekilde bir x elemanı olmalıdır.

$\exists x P(x)$ gösterimi $P(x)$'in varolussol nicelemesidir.

ifade	doğru olduğunda	yanlış olduğunda
$\forall x P(x)$	$P(x)$ herbir x için geçerlidir.	$P(x)$'in yanlış olduğu bir x vardır.
$\exists x P(x)$	$P(x)$'in doğru olduğu bir x değeri vardır.	$P(x)$ her x için yanlıştır

$P(x)$, "$x+1>x$" ifadesi olsun. Tüm pozitif sayılar için $P(x) \equiv T$

$P(x)$ ifadesi "$x^2<10$" olduğunda tanım bölgesi 4'ü geçmeyen pozitif tam sayılar olduğunda $\forall x P(x)$ nedir?

$x = 1, 2, 3, 4$ — 4 için yanlıştır.

$P(1) \wedge P(2) \wedge P(3) \wedge P(4) \equiv F$

$\exists x P(x) \equiv P(1) \vee P(2) \vee P(3) \vee P(4) \equiv T$

$\forall$ ve $\exists$ niceleyicileri önermesel analitde bütün diğer mantıksal operatörlerden daha yüksek önceliğe sahiptir.

Niceleyicileri İçeren Mantıksal Denklikler

Kısıtlı Bölgeler

$\forall x < 0 \quad (x^2 > 0) \quad x \in R$

$\forall y \neq 0 \quad (y^3 \neq 0) \quad y \in R$

$\exists x \ (x + y = 1)$

$\forall x \ (P(x) \wedge Q(x)) \equiv \forall x \, P(x) \wedge \forall x \, Q(x)$

Niceleyicilerin Değilleri

Değil	Eşdeğer ifade	Değili ne zaman doğru?	Ne zaman yanlış?
$\neg \exists x \, P(x)$	$\forall x \, \neg P(x)$	Her x için $P(x)$ yanlış	$P(x)$ doğru olacak şekilde bir x vardır.
$\neg \forall x \, P(x)$	$\exists x \, \neg P(x)$	$P(x)$ yanlış olacak şekilde bir x vardır.	Her x için $P(x)$ doğrudur.

$\forall x P(x) \rightarrow$ her x için $P(x)$ doğrudur.

ÖR: $\neg \forall x \ (P(x) \rightarrow Q(x))$ değilini bulun.

$\exists x \ \neg (P(x) \rightarrow Q(x)) \equiv \exists x \ \neg (\overline{P(x)} \vee Q(x)) \equiv \exists x \ (P(x) \wedge \neg Q(x))$

Mantıksal Programlama

Prolog (Programming Logic) Önerme mantığı kullanılır.

Prolog gerçekleri ve prolog kuralları vardır.

Her dersin öğretim elemanının ve kayıtlı öğrencilerin hangi sınıfta olduğunu söyleyen bir program ele alalım.

Böyle bir program için;

egitmen(p, c)

kayıtlı (s, c)

prolog gerçekleri

egitmen(akman, mat273)

egitmen(ertemiz, ele222)

egitmen(demirli, bil302)

kayitli(lale, ele222)

kayitli(lale, bil302)

kayitli(kenan, mat273)

kayitli(batuhan, mat273)

kayitli(batuhan, bil302)

prolog kurali

ogretir(P,S) :- egitmen(P, C), kayitli(S, C)

? kayitli(kenan, mat273) → yes

? kayitli(x, mat273) → mat273 dersini alanları sırala

cikti → Kenan, Batuhan

? ogretir(x, lale) → Lale'ye kimler ders verir?

cikti → ertemiz, demirli

İç içe Niceleyiciler

$\forall x \forall y P(x,y)$ $\exists x \exists y P(x,y)$ → Sıra önemli değildir.

$\forall x \exists y P(x,y)$ $\exists x \forall y P(x,y)$ → Dıştan içe doğru

ÖR: $Q(x,y)$: $x+y=0$

1) $\forall x \exists y (Q(x,y)) \equiv T$ 2) $\exists x \forall y (Q(x,y)) \equiv F$

1) Tüm x'ler için bazı y'ler sağlar (T)

2) Bazı x'ler için tüm y'ler sağlar (F)

Çıkarım Kuralları

Geçerli şifreniz varsa iletişim ağına bağlanabilirsiniz.

p → geçerli bir şifreniz var
q → iletişim ağına bağlanabilirsiniz

$$\frac{p \to q \quad p}{\therefore q}$$

"iletişim ağına erişiminiz varsa, notunuzu değiştirebilirsiniz"
"iletişim ağına erişiminiz var"

$$\frac{}{\therefore \text{"notunuzu değiştirebilirsiniz"}}$$

p → q: "Ali piyangoyu kazanırsa araba alacak"
p: "Ali piyangoyu kazandı"
q: "Ali araba alacak"

Modus Ponens

$$\frac{p \to q \quad p}{\therefore q}$$

Totoloji: $(p \wedge (p \to q)) \to q$ (Tümevarım)

Modus Tollens

$$\frac{q' \quad p \to q}{\therefore p'}$$

(Tümdengelim)

$(p' \wedge (p \to q)) \to p'$

Varsayıma Dayalı Kıyas

$$\frac{p \to q \quad q \to r}{\therefore p \to r}$$

$((p \to q) \wedge (q \to r)) \to (p \to r)$

Ayırıcı Kıyas

$$\frac{p \vee q \quad p'}{\therefore q}$$

Ali'nin cüzdanı cebinde veya masasındadır.
$((p \vee q) \wedge p') \to q$

Toplama

$$\frac{p}{\therefore p \vee q} \qquad p \rightarrow (p \vee q)$$

Sadeleştirme

$$\frac{p \wedge q}{\therefore p} \qquad (p \wedge q) \rightarrow p$$

Birleşme

$$\frac{p \quad q}{\therefore p \wedge q} \qquad ((p) \wedge (q)) \rightarrow (p \wedge q)$$

Karar

$$\frac{p \vee q \quad p' \vee r}{\therefore q \vee r} \qquad ((p \vee q) \wedge (p' \vee r)) \rightarrow (q \vee r)$$

ÖR: "Bu öğleden sonra hava güneşli değil ve dünden daha soğuk"

"Sadece güneşli olursa yüzmeye gideceğiz"

"Yüzmeye gitmezsek, o zaman kano gezintisi yapacağız"

"Kano gezintisi yaparsak o zaman güneş batana kadar evde olacağız"

p: "Bu öğleden sonra hava güneşli"

q: "Dünden daha soğuk"

r: "Yüzmeye gideceğiz"

s: "Kano gezintisi yapacağız"

t: "Güneş batana kadar evde olacağız"

$p' \wedge q,\ r \rightarrow p,\ r' \rightarrow s,\ s \rightarrow t$

1) $p' \wedge q$	ön koşul
2) p'	1'i kullanarak sadeleştirme
3) $r \wedge p$	ön koşul
4) r	2 ve 3'ü kullanarak Modus Tollens
5) $r' \rightarrow s$	ön koşul
6) s	4 ve 5 kullanarak Modus ponens
7) $s \wedge t$	ön koşul
8) t	6 ve 7 kullanarak Modus ponens

ÖR: "Remzi çok çalışır"

"Remzi çok çalışırsa aptal bir çocuktur."

"Remzi aptal bir çocuksa ise giremeyecek"

p: "Remzi çok çalışır"

q: "Remzi aptal bir çocuktur."

r: "Remzi ise giremeyecek"

$$\frac{p \to q \quad q \to r}{\therefore p \to r}$$

Varsayıma dayalı kıyas

ÖR: "Bana e-posta gönderirseniz o Aman program yazmayı bitireceğim"

"Bana e-posta göndermezseniz o Aman erken uyumaya gideceğim"

"Erken uyumaya gidersem kendimi iyi hissederek uyanacağım"

p: "e-posta gönderme" q: "program yazmayı bitirme" $p \to q$

r: "erkenden uyumaya gideceğim" $p' \to r$

s: "kendimi iyi hissederek uyanacağım" $r \to s$

Teorem İspat Metotları

Doğrudan İspat

$p \to q$ ifadesinin doğrudan ispatında p'nin doğru olduğu kabul edilir. Çıkarım kuralları kullanılarak q'nun doğru olduğu ispatlanır.

ÖR: "Eğer n tek tam sayı ise n^2 tektir." teoremini ispatlayınız.

$\forall n (P(n)) \to Q(n)$

$n = 2k+1$ = tektir bütün k sayıları için

$n^2 = (2k+1)^2 = 4k^2 + 4k + 1$

Devirmeli İspat

$p \to q$ koşullu önermesinin devriği $q' \to p'$ önermesine denk olduğu gerçeği kullanılır. Buna devirmeli ispat denir.

ÖR: Eğer n bir tam sayı ve $3n+2$ tek sayı ise o zaman n'in tek olduğunu ispatlayınız.

$n=2k$ $\quad 3n+2=6k+2$

İçi Boş ve Asikar İspatlar

p'nin yanlış olduğu durumlarda $p \to q$ önermesi doğrudur. Sonra olarak p'nin yanlış olduğunu gösterebilirsek o zaman $p \to q$ içi boş ispat olarak adlandırılır.

ÖR: Tanım bölgesinin tam sayılar olduğu durumda P(n)'in "Eğer $n>1$ ise o zaman $n^2>n$" önermesi olduğu halde P(0)'ın doğru olduğunu gösteriniz.

$P(0): 0>1 \quad 0^2>0 \Rightarrow$ içi boş bir ispattır.

KÜMELER

Sıralı olmayan nesneler topluluğudur.

Küme içindeki nesnelere eleman ya da küme üyesi denir.

a nesnesinin A kümesinin elemanı olma durumu $a \in A \quad a \notin A$

$\{a, b, c, \ldots\}$ en yaygın listeleme kullanımı

$A = \{x \mid x<10$ tam sayıdır$\}$

$A = \{x \in \mathbb{Z}^+ \mid x$: tek sayıdır$\}$

N: Doğal sayılar kümesi

Z: Tam sayılar kümesi

$\mathbb{Z}^+$: Pozitif tam sayılar kümesi

$Q = \{p/q \mid p \in \mathbb{Z}, q \in \mathbb{Z}, q \neq 0\}$ rasyonel sayılar kümesi

R: reel sayılar kümesi

C: Kompleks sayılar kümesi

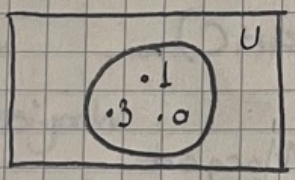
Tanım: İki küme sadece ve sadece aynı elemanlardan oluşuyorsa denktirler.

$\forall x (x \in A \leftrightarrow x \in B)$

$\{1, 3, 5\} \equiv \{5, 3, 1\}$

Boş küme: İçinde eleman olmayan kümeye denir. $\{\}$ ya da $\emptyset$ ile gösterilir.

Evrensel Küme:



Alt Küme: Bir A kümesi, sadece ve sadece A'nın tüm elemanları aynı zamanda B'nin elemanı ise B kümesinin alt kümesidir.

$A \subset B$ $A \not\subset B \rightarrow$ alt küme değildir.

Her S kümesi için, boş küme S'in alt kümesidir. Her S de S'in alt kümesidir.

$\emptyset \subset S$ $S \subset S$

Küme Büyüklüğü

Kümenin eleman sayısı. Mutlak değer ile gösterilir. $|S|$

Bir küme sonlu değilse sonsuz kümedir.

Kartezyen Çarpım

A ve B kümeler olsun. Kartezyen çarpımı $A \times B$ şeklinde gösterilir. $a \in A$ $b \in B$ olmak üzere bütün sıralı çiftler $A \times B$'nin TOM

$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$ elemanıdır

ÖR: $A = \{1, 2\}$ $B = \{a, b, c\}$

$A \times B = \{(1,a), (1,b), (1,c), (2,a), (2,b), (2,c)\}$

$A \times B \neq B \times A$

Özellikler

$A \cap U \equiv A$	$A \cup B \equiv B \cup A$
$A \cap \emptyset \equiv A$	$A \cap B \equiv B \cap A$
$A \cup U \equiv U$	$A \cup (B \cup C) \equiv (A \cup B) \cup C$
$A \cap \emptyset \equiv \emptyset$	$A \cap (B \cap C) \equiv (A \cap B) \cap C$
$A \cup A \equiv A$	$A \cup (B \cap C) \equiv (A \cup B) \cap (A \cup C)$
$A \cap A \equiv A$	$A \cap (B \cup C) \equiv (A \cap B) \cup (A \cap C)$
$A \cup \bar{A} \equiv U$	$\overline{A \cap B} \equiv \bar{A} \cup \bar{B}$ } De Morgan
$A \cap \bar{A} \equiv \emptyset$	$\overline{A \cup B} \equiv \bar{A} \cap \bar{B}$ } De Morgan

$A \cup (A \cap B) \equiv A$ } Yutma özelliği

$A \cap (A \cup B) \equiv A$ } Yutma özelliği

Tanım: $A_1, A_2, \ldots, A_n$ kümelerinin kartezyen çarpımı $A_1 \times A_2 \times \ldots \times A_n$ olarak gösterilir ve $(a_1, a_2, \ldots, a_n)$'den oluşan sıralı n'li demetten oluşur. Burada, a_i elemanı $i = 1, 2, \ldots, n$ olmak üzere A_i'den gelir. Başka bir deyişle

$$A_1 \times A_2 \times \ldots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i,\ i = 1, 2, \ldots, n \text{ için}\}$$

Küme İşlemleri

A ve B birer küme olsun.

$A \cup B = \{x \mid x \in A \vee x \in B\}$

$A \cap B \equiv \{x \mid x \in A \wedge x \in B\}$

NOT: İki kümenin kesişimi boş küme ise bu iki küme ayrık kümelerdir.

A bir küme olsun. A'nın tümleyeni $\bar{A}$ ile gösterilir. Ya da U evrensel küme olmak üzere $\bar{A} = U - A$ 'dır. ve $A - B = A \cap \bar{B}$ 'dir.

$$\bar{A} = \{ x \in U \mid x \notin A \}$$

$$\overline{A \cup (B \cap C)} \equiv \bar{A} \cap (\bar{B} \cup \bar{C})$$

$\bigcup_{i=1}^{n} A_i \quad A_1 \cup A_2 \cup A_3 \cup \ldots \cup A_n$

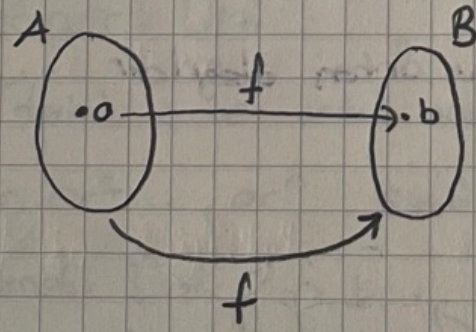
$\bigcap_{i=1}^{n} A_i \quad A_1 \cap A_2 \cap A_3 \cap \ldots \cap A_n$

Fonksiyonlar

A ve B boş olmayan kümeler olsun. A'dan B'ye bir fonksiyon f, A'nın her elemanının B'nin tam olarak bir elemanına atanmasıdır.

$f: A \to B$ şeklinde gösterilir.

$f(a) = b \quad a \in A \quad b \in B$



$b = f(a)$

f_1 ve f_2 A'dan B'ye fonksiyonlar olsun.

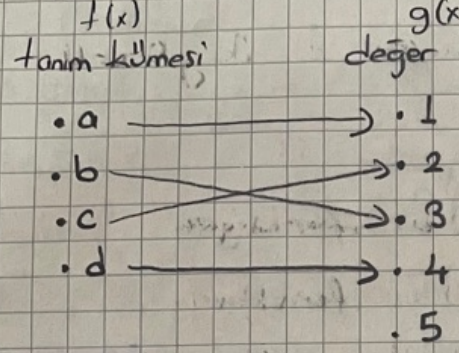
$(f_1 + f_2)(x) = f_1(x) + f_2(x)$

$(f_1 \cdot f_2)(x) = f_1(x) \cdot f_2(x)$

Birebir Fonksiyon

Bir f fonksiyonu ancak ve ancak her A ve B, f'in tanım kümesi eleman için $f(a)=f(b)$ eşitliği $a=b$ olmasını gerektiriyorsa birebir denir.

f birebir ise içinedir.



Birebirdir.

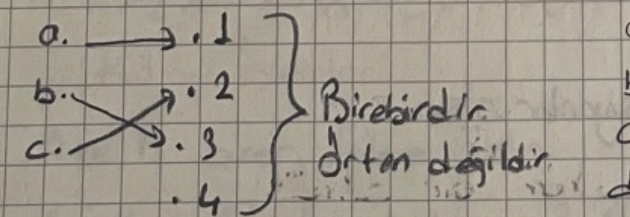
f(x)'deki her eleman g(x)'de bir elemana gidecek. f(x)'de boşta eleman kalmayacak. f(x)'deki iki eleman g(x)'e gidebilir (fonksiyon olma şartı)

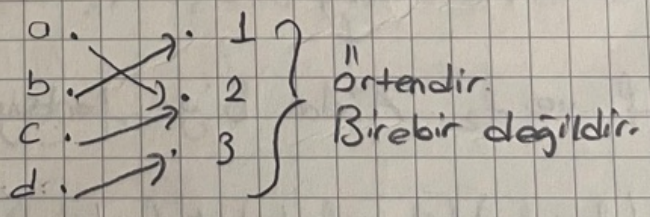
Örten Fonksiyon

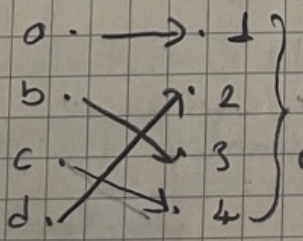
A'dan B'ye bir f fonksiyonu ancak ve ancak her $b \in B$ için $f(a)=b$ olacak şekilde bir $a \in A$ varsa örtendir denir. Bir f fonksiyonu örten ise üzerine olarak adlandırılır.

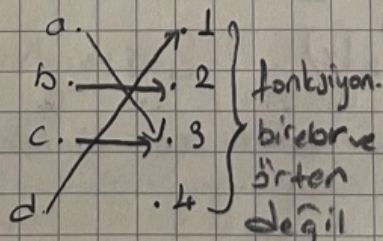
Tam sayılar kümesindeki x^2 fonksiyonu örten değildir.

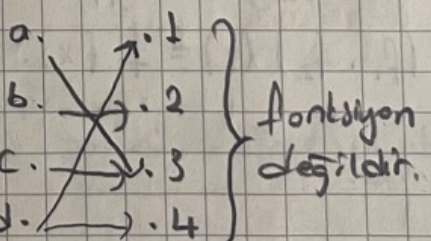
$f(x)=x+1$ tam sayılar için örtendir.











Ters Fonksiyonlar ve Bileşkeleri

f: A kümesinden B kümesine bir birebir eşleme olsun. f'in ters fonksiyonu, B'nin bir b elemanını A'nın $f(a)=b$ olacak şekildeki tek a elemanına götürür. f'in ters fonksiyonu f^{-1} ile gösterilir. Dolayısıyla $f(a)=b$ olduğunda $f^{-1}(b)=a$ olur.

g: A kümesinden B kümesine $(g: A \to B)$ bir fonksiyon.
f: B kümesinden C kümesine $(f: B \to C)$ bir fonksiyon.
f ve g'nin bileşkesi fog ile gösterilir. Her $a \in A$ için $fog(a)=f(g(a))$

Diziler

Bir dizi, tamsayılar kümesinin bir alt kümesinden bir S kümesine bir fonksiyondur. Bir n tamsayısının görüntüsü a_n ile gösterilir.

$a_n = a_1 \cdot r^{n-1}$

Ör: $a_n = \frac{1}{n}$ olmak üzere $\{a_n\}$ dizisi $a_1, a_2, a_3, \ldots, a_n$

$1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots, \frac{1}{n}$

Geometrik Dizi

İlk terimi a ve ortak oranı (çarpanı) r gerçel sayıları olan $a, ar, ar^2, \ldots, ar^n$ şeklinde bir dizidir.

Aritmetik Dizi

İlk terimi a, ve ortak farkı d gerçel sayıları olan $a, a+d, a+2d, \ldots, a+nd$ şeklinde bir dizidir.

Özyineleme

Bir $\{a_n\}$ dizisi için özyineleme ilişkisi a_n teriminin dizinin önceki terimlerinin bir ya da daha fazlası cinsinden ifadesidir.

Ör: fibonacci dizisi $1, 1, 2, 3, 5, 8, 13, 21, \ldots$

ÖR: Bir kişi bir bankadaki yıllık bileşik faizi %11 olan mevduat hesabına 10.000 TL yatırır. 30 yıl sonra hesapta kaç TL olur?

$P_n = P_{n-1} + 0.11 P_{n-1} = (1.11) P_{n-1}$

$P_{30} = 228.922,97$

Toplamlar

$\sum$ ile gösterilir.

$a_m + a_{m+1} + \dots + a_n$ toplamını $\{a_n\}$ dizisinden temsil etmek için

$\sum_{j=m}^{n} a_j$ gösterimi kullanılır.

a ve r gerçel sayılar ve $r \neq 0$ ise

$$\sum_{j=0}^{n} ar^j = \begin{cases} \frac{ar^{n+1} - a}{r-1} & \text{eğer } r \neq 1 \\ (n+1)a & \text{eğer } r = 1 \end{cases}$$

ÖR: $\sum_{i=1}^{4} \sum_{j=1}^{3} i \cdot j = \sum_{i=1}^{4} (i + 2i + 3i) = \sum_{i=1}^{4} 6i = 6 + 12 + 18 + 24 = 60$

ÖR: $1 + 2 + 8 + \dots + 1024 = ?$

$$\frac{a r^{n+1} - 0}{r-1} = \frac{2^{11} - 1}{2-1} = 2^{11} - 1 = 2047$$

ÖR: $\sum_{k=1}^{n} k^2 = 1 + 4 + 9 + 25 + \dots + n^2$

$$= \frac{n \cdot (n+1) \cdot (2k+1)}{6}$$

ÖR: $\sum_{k=1}^{n} k^3 = \frac{n^2 \cdot (n+1)^2}{4}$

MATRİSLER

Sayıların dikdörtgensel bir düzlemde gösterimidir. m satır ve n sütundan oluşan matrise bir m×n matris denir.

Aynı sayıda satır ve sütuna sahip matrislere kare matris denir.

Eğer iki matrisin satır ve sütun sayıları eşit ve aynı adresteki bileşenleri birbirine eşitse bu iki matrise eşit matrisler denir.

$$A=[a_{ij}] \qquad B=[b_{ij}]$$

$$A+B=[a_{ij}+b_{ij}]$$

Matris Çarpımı: A bir m×k matris, B bir k×n matris olsun. A ile B matrisinin AB çarpımının (i, j) bileşeni, B matrisinin j. sütunu ve A matrisinin i. satırına karşılık gelen elemanların çarpımının toplamı olarak tanımlanır.

$$c_{ij}=a_{i1}\cdot b_{1j}+b_{1j}\cdot a_{2i}+\cdots+a_{ik}\cdot b_{kj}$$

Birim Matris

n. mertebeden birim matris eğer $i=j$ ise $\delta_{ij}=1$, $i\neq j$ ise $\delta_{ij}=0$ olduğu matristir.

$$A=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$ Birim matris

Bütün elemanları 0 ya da 1 olan matrislere 01 matrisi denir. Bu tür matrisler üzerinde mantıksal operatörler kullanılabilir.

$$A=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \quad B=\begin{bmatrix}1 & 0\\ 1 & 0\end{bmatrix} \quad A\vee B=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$$

Matrislerin Boolean Çarpımı

$A \odot B$ şeklinde gösterilir.

$A \odot B \quad (a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee \dots \vee (a_{ik} \wedge b_{kj})$

ÖR:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$A \odot B = \begin{bmatrix} (1 \wedge 1) \vee (0 \wedge 0) & (1 \wedge 1) \vee (0 \wedge 1) & (1 \wedge 0) \vee (0 \wedge 1) \\ (0 \wedge 1) \vee (1 \wedge 0) & (0 \wedge 1) \vee (1 \wedge 1) & (0 \wedge 0) \vee (1 \wedge 1) \\ (1 \wedge 1) \vee (0 \wedge 0) & (1 \wedge 1) \vee (0 \wedge 1) & (1 \wedge 0) \vee (0 \wedge 1) \end{bmatrix}_{3 \times 3}$$

$$= \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

ALGORİTMALAR

Algoritma, bir hesaplama yapmak için veya bir problemi çözmek için gereken sonlu ve kesin emirler kümesidir.

Sonlu bir serideki en büyük elemanı bulma

procedure max ($a_1, a_2, \dots, a_n$: tam sayılar)

max = a_1

for i := 2 to n

if max < a_i then max := a_i

return max { max en büyük elemandır.

Arama Algoritmaları

Doğrusal arama algoritması

```
procedure linearSearch (x: tam sayı, a1, a2, ..., an: farklı tam sayılar)
i = 1
while( i ≤ n ve x ≠ ai)
    i := i+1
if i ≤ n then konum := i
else konum := 0
return konum { konum: değeri x'e eşit olan terimin indisidir veya 0'dır,
                                x bulunamamışsa }
```

İkili arama algoritması $O(\log n)$

```
procedure binarySearch(x: tam sayı, a1, a2, ..., an: artan tamsayılar)
i=1 { i arama aralığının sol son noktasıdır. }
j=n { j arama aralığının sağ son noktasıdır }
while i<j
    m := ⌊(i+j)/2⌋
    if x > am then i := m+1
    else j := m
if x = ai then konum := i
else konum = 0
return konum
```

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

1 2 3 5 6 7 8 10 | 12 13 15 16 18 19 20 22

12 13 15 16 | 18 19 20 22

18 19 | 20 22

18 | 19

19

Listenin terimleri artan sırada sıralanmış olmalıdır. Bulunacak elemanı listenin ortasındaki elemanla karşılaştırarak başlar. Liste daha sonra aynı boyuttaki 2 alt listeye bölünür. Bu listelerden birinin eleman sayısı diğerinden 1 eksiktir. Arama, aranan elemanın ortanca eleman ile karşılaştırılmasının sonucuna göre ilgili listeden devam eder.

Bubble Sort (Kabarcık Sıralama)

En kolay sıralama algoritmalarından birisidir fakat en verimlisi değildir ($O(n^2)$)

Bir listeyi ardışık olarak komşu elemanları karşılaştırarak ve eğer yanlış sırada ise yerlerini değiştirerek sıraya koyar.

Listenin elemanları bir sütun olarak düşünülürse kabarcık sıralamasında küçük elemanlar büyük elemanlarla yerleri değiştikçe yukarı doğru balonlanmaktadır. Büyük elemanlar ise dibe batar.

Insertion Sort (Yerleştirmeli Sıralama) $O(n^2)$

n elemanlı bir listeyi sıralamak için ikinci elemandan başlar. İkinci elemanı ilk elemanla karşılaştırır. İkinci eleman ilk elemandan küçük ise onu ilk elemandan önce ekler. Eğer ikinci eleman ilk elemandan büyükse ilk elemandan sonraya ekler. Bu noktada ilk iki eleman doğru sıradadır. Üçüncü eleman, ilk eleman ile karşılaştırılır. ve eğer ilk elemandan büyükse ikinci elemanla karşılaştırılır. İlk üç eleman içinde doğru konuma yerleştirilir.

Bu algoritma gerçek hayatta çalışır. ÖR: Kartların sıralanması

Recursive binary search algoritması (mazeret sınavı sorusu)

procedure binarySearch(l, r, x: tamsayılar, $a_1, a_2, \ldots, a_n$: artan tamsayılar)

if $r \geq l$ then $mid := l + (r-l)/2$

if $arr_{mid} = x$ then return mid

if $arr_{mid} > x$ then return binarySearch(l, mid-1, x, arr)

return binarySearch(mid+1, r, x, arr)

return 0

Sıralama Algoritmaları

Bubble sort algoritması $O(n^2)$

procedure bubbleSort($a_1, a_2, \ldots, a_n$: reel sayılar, $n \geq 2$

for i := 1 to n-1

  for j := 1 to n-1

    if $a_j > a_{j+1}$ then a_j ve a_{j+1}'i değiştirin.

{$a_1, \ldots, a_n$ artan sıradadır}

Insertion sort algoritması

procedure insertionSort($a_1, a_2, \ldots, a_n$: reel sayılar, $n \geq 2$)

for j := 2 to n

  i := 1

  while $a_j > a_i$

    i := i+1

  m := a_j

  for k := 0 to j-i-1

    $a_{j-k} := a_{j-k-1}$

  $a_i := m$

{$a_1, a_2, \ldots, a_n$ sıralanmıştır}

379 Greedy (Aç Gözlü) Algoritmalar

Optimizasyon algoritmalarıdır. Optimizasyon, sistemin en iyi çözümünü bulmaktır. En iyi çözümler her problem için çözülemez. Bu tip problemler NP (non-polynomial) problemlerdir. Bunun haricindeki problemler P (polynomial) problemlerdir. P problemleri polinom şeklindedir çözülebilirler. NP problemleri polinom değildir Greedy ile çözülür. En iyi çözümü garanti etmez. Felsefesi o anki çözümü en iyi kabul etmesidir.

Açgözlü para bozdurma algoritması

procedure greedy ($c_1, c_2, \ldots, c_r$: bozuk para değerleri, burada $c_1 > c_2 > \ldots > c_r$; n bir pozitif tam sayı)

for i := 1 to r

$d_i := 0$ { d_i kullanılan c_i büyüklüğünde bozuk paraları saymaktadır }

while $n \geq c_i$

$d_i := d_i + 1$

$n_i := n - c_i$

{ d_i; i = 1, 2, ..., r, çeşit c büyüklüğünde bozuk paralar }

$c_1 = 100$ TL

$c_2 = 50$ TL

$c_3 = 20$ TL

$c_4 = 10$ TL

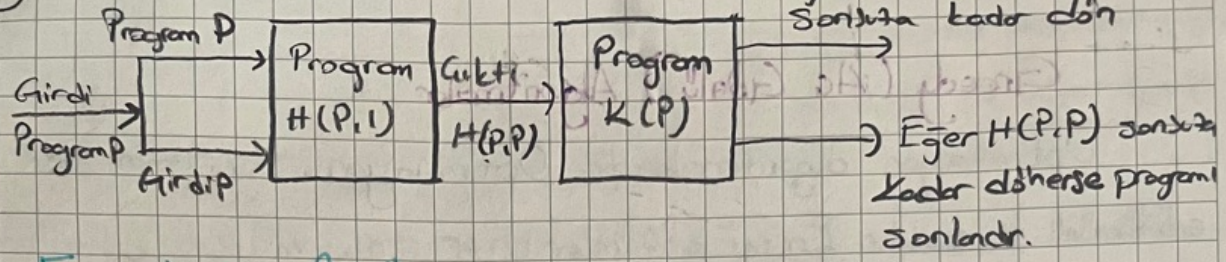
$c_5 = 5$ TL

$c_6 = 1$ TL

Halting Problem

Bir bilgisayar programı ve bu programın girdisini girdi olarak alan ve programın belli bir zaman sonunda sonlanıp sonlanmayacağının tespit edilmesi problemidir. (Hata kontrolü yaparken programın sonsuz bir döngüye girip girmediğinin test edilmesi)

...lting Problem çözümsüzdür:



Fonksiyonların Büyümesi

Big O Notasyonu

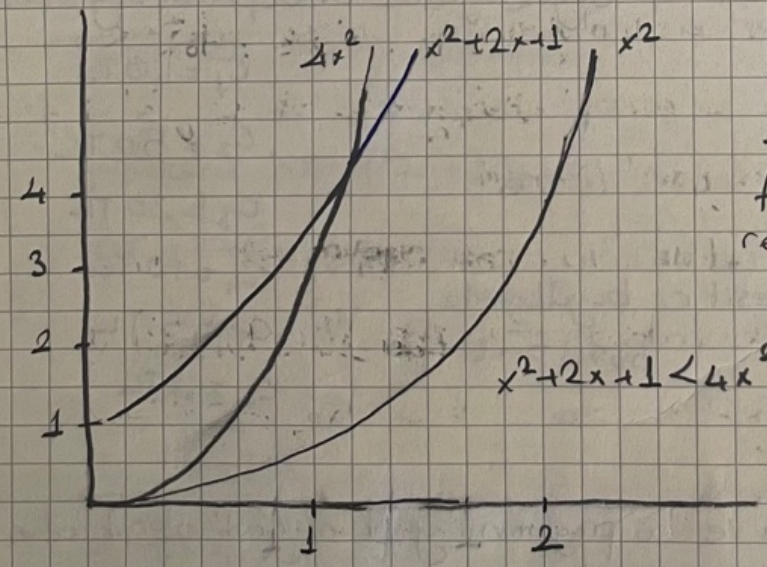
f ve g tamsayılar kümesinden veya reel sayılar kümesinden reel sayılar kümesine tanımlanmış iki küme olsun. Eğer $x > k$ olduğunda $|f(x)| \leq C|g(x)|$ oluyorsa bu eşitsizliği sağlayan C ve k gibi sabit sayılar varsa bu durumda $f(x) = O(g(x))$ olmaktadır.

ÖR: $f(x) = x^2 + 2x + 1$ 'in Big O'sunun $O(x^2)$ olduğunu gösteriniz

$x > 1$ olduğunda $x > x^2$ ve $1 > x^2$ 'dir.

$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$

$0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$ $C=3$ $k=2$ $O(x^2)$



$f(x) = x^2 + 2x + 1$ 'in grafiğinin $f(x) < 4x^2$ 'yi sağlayan kısmı renkli olarak gösterilmiştir.

NOT: Böl ve yönet algoritmaları logn'de çalışır.

Algoritmik paradigmalar → Brute force (kobakuvvet), devide and conquer (böl ve yönet), greedy (aç gözlü), dinamik programlama, doğrusal programlama, backtracking (geri izleme)

Ω Notasyonu (Alt Değer / En İyi Durum) (Omega)

f ve g tam veya reel sayılar kümesinden reel sayılara tanımlanmış fonksiyonlar. $x > k$ olduğunda $|f(x)| \geq C|g(x)|$ oluyorsa bu eşitliği sağlayan C ve k gibi sabit sayılar varsa bu durumda $f(x) = \Omega(g(x))$

Θ Notasyonu (Ortalama Değer / Ortalama Durum) (Teta)

f ve g, tam veya reel sayılar kümesinden reel sayılara tanımlanmış fonksiyonlar. $f(x)$, $O(g(x))$ ve $f(x)$, $\Omega(g(x))$ ise $f(x)$, $\Theta(g(x))$'tir deriz.

$x > k$ ise $C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$ oluyorsa ve bu eşitsizliği sağlayan pozitif C_1 ve C_2 reel sayıları ve bir pozitif k reel sayısı bulunabiliyorsa bu durumda $f(x)$'in $\Theta(g(x))$ olduğunu gösterebiliriz.

$f_1(x) = O(g_1(x))$ ve $f_2(x) = O(g_2(x))$ ise

$(f_1 + f_2)(x) = O(\max(|g_1(x)|, |g_2(x)|))$

$(f_1 \cdot f_2)(x) = O(g_1(x) \cdot g_2(x))$

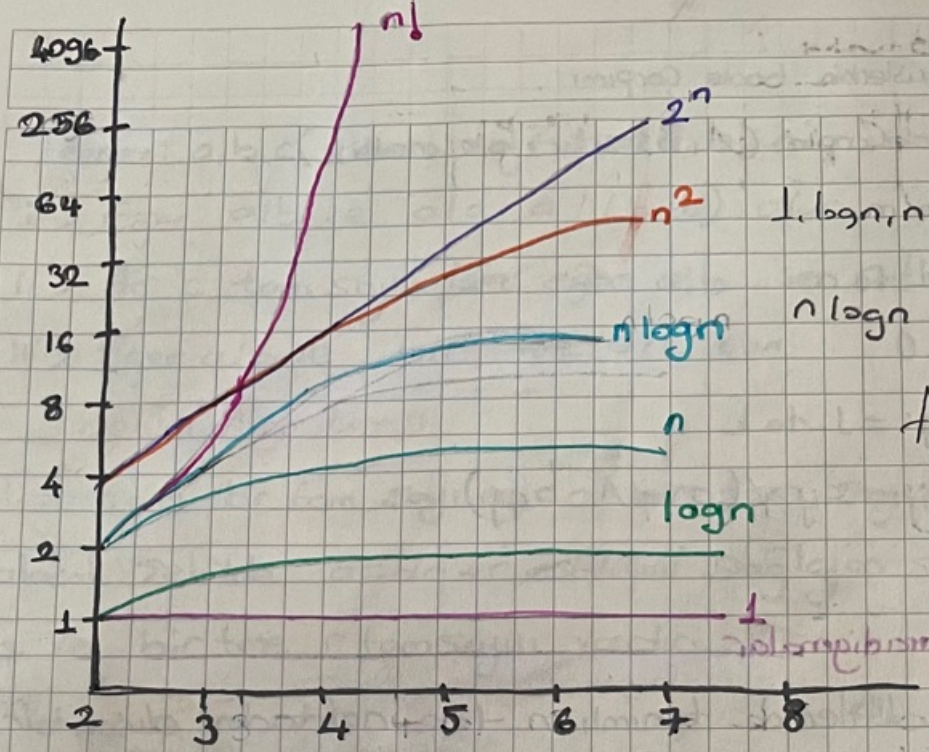
$(x+1) \cdot \log(x^2+1) + 3x^2 \Rightarrow x^2$

$(x+1)^2 \cdot \log(x^2+1) + 3x^2 \Rightarrow x^2 \log x$

Binary Search

$T(n) = T(n/2) + 1 = T(n/4) + 1 + 1 = (n/k^n) + k \rightarrow T(1) + k \Rightarrow 1 + \log n$

$= \log n$



1. $\log n, n, n\log n, n^2, n^3, 2^n, n!$

$n \log n \rightarrow x^2$

```
for(i=0; i<75; i++)
```

$f_1(x) = O(g_1(x))$; $f_2(x) = O(g_2(x))$ olsun. $(f_1+f_2)(x)$ Big-O'su $O(\max(g_1(x), g_2(x)))$'dir. $(f_1 \cdot f_2)(x)$ Big-O'su $O(g_1(x) \cdot g_2(x))$'dir.

Tanım: f ve g tamsayılar kümesinden veya reel sayılar kümesinden reel sayılar kümesine tanımlanmış iki fonksiyon olsun. $x > k$ olduğunda $|f(x)| \geq C \cdot |g(x)|$ oluyorsa bu eşitliği sağlayan C ve k gibi sabitler varsa bu durumda $f(x) = O(g(x))$'tir denir.

f ve g tamsayılar kümesinden veya reel sayılar kümesinden reel sayılar kümesine tanımlı iki fonksiyon olsun. Eğer $f(x) = O(g(x))$ ve $f(x) = \Omega(g(x))$ ise $f(x) = \Theta(g(x))$'tir. denir ve ortalama değeri belirtir.

Matris Çarpımı

```
Procedure matrisCarpma (A, B : matrisleri)
for i := 1 to m
    for j := 1 to n
        cij := 0
        for q := 1 to k
            cij = cij + aiq . baj
return C { C = {cij} matrisi A ve B çarpımıdır }
```

Strassen Algoritması
Matris çarpım algoritmasıdır.
$n^3 \rightarrow n^{2.78}$

Sıfır bir matrislerinin boole çarpımı

```
procedure booleCarpimi (A, B: sıfır-bir matrisi)
for i:=1 to m
    for j:=1 to n
        cij=0
        for q:=1 to k
            cij:= cij ∨ (aiq ∧ bqj)
return C
```

Algoritmik Paradigmalar

2 boyutlu bir düzlemde tanımlı n tane noktadan oluşan bir kümedeki en yakın nokta çiftini bulan kaba kuvvet algoritması

```
procedure yakinCiftler((x1, y1), (x2, y2), ..., (xn, yn): reel sayı çiftleri)
min=∞
for i:=2 to n
    for j:=1 to i-1
        if (xj - xi)^2 + (yj - yi)^2 < min then
            min:= (xj - xi)^2 + (yj - yi)^2
            enYakinCift:= ((xi, yi), (xj, yj))
return enYakinCift
```

Sayılar Teorisi

a ve b tamsayı $a \neq 0$ olmak üzere $b = a \cdot c$ şeklinde bir c sayısı varsa yani b/a tam sayı ise; a, b'yi böler.

a, b'nin bölenidir. b, a'nın katıdır.

$a \mid b$ → b bölünmüyorsa böyle ifade edilir.

3 | 7 4 | 12 $\frac{12}{4}$

Teorem: a, b, c tam sayı $a \neq 0$ olsun.

i) Eğer $a|b$ ve $a|c$ $a|(b+c)$ olur.

ii) Her c tamsayısı için eğer $a|b$ ise $a|bc$ sağlanır.

iii) Eğer $a|b$ ve $b|c$ ise $a|c$ olur.

Bölüm Algoritması

Teorem: a bir tam sayı ve d bir tam sayı olsun. $0 \leq r \leq d$ olacak şekilde $a = dq + r$ eşitliğini sağlayan sadece bir tane q ve bir tane r tamsayısı vardır.

Modüler Aritmetik

a ve b tamsayılar ve m bir pozitif tamsayı, m, $(a-b)$'e bölünüyorsa b sayısına denktir deriz.

$a \equiv b \mod m$

Teorem: m pozitif bir tamsayı, eğer $a \equiv b \mod m$ ise $c \equiv d \mod m$ ise; $a+c \equiv b+d \mod m$'dir.

$7 \equiv 2 \mod 5$ $\quad 11 \equiv 1 \mod 5$ $\quad 18 \equiv 3 \mod 5$

b tabanında açılım algoritması

procedure bTabanindaAcilim (n, b: b>1 koşulunu sağlayan pozitif tamsayılar)

q := n

k := 0

while $q \neq 0$

$a_k := q \mod b$

$q := q \text{ div } b$

$k := k+1$

return $(a_{k-1}, \ldots, a_1, a_0)$

Tamsayı toplamı algoritması

```
procedure topla (a, b: pozitif tamsayılar)
c := 0
for j = 0 to n-1
    d := [(aj + bj + c)/2]
    sj = aj + bj + c - 2d
    c := d
sn := c
return (s0, s1, ..., sn)
```

Çarpma algoritması

```
procedure carpma(a, b: pozitif tamsayılar)
for j = 0 to n-1
    if bj = 1 then cj := a sayısı j basamak kayar.
    else cj := 0
p := 0
for j := 0 to n-1
    p := p + cj
return p
```

Bölüm algoritması

```
procedure bolum (a: tam sayı, d: pozitif tamsayı)
q := 0
r := |a|
while r ≥ d
    r := r - d
    q := q + 1
if a < 0 and r > 0 then
    r := d - r
    q := -(q + 1)
return (q, r)
```

Öklid Algoritması

ebob(287, 91) Büyük olan küçük olana bölünür (sürekli devam eder)

$287 = 91 \cdot 3 + 14$

$91 = 14 \cdot 6 + 7$

$14 = (7) \cdot 2 + 0$ → ebob

ebob (252, 189)

$252 = 198 \cdot 1 + 54$

$198 = 54 \cdot 3 + 36$

$54 = 36 \cdot 1 + 18$

$36 = (18) \cdot 2 + 0$ → ebob

ebob(662, 414)

$662 = 414 \cdot 1 + 248$

$414 = 248 \cdot 1 + 166$

$248 = 166 \cdot 1 + 82$

$166 = 82 \cdot 2 + 4$

$82 = 4 \cdot 20 + 2$

$4 = (2) \cdot 2 + 0$ → ebob

Öklid algoritması

```
procedure ebob (a,b pozitif tamsayılar)
x := a
y := b
while y ≠ 0
    r := x mod y
    x := y
    y := r
return x
```

ebob(60,36)

$60 = 36 \times 1 + 24$

$36 = 24 \times 1 + 12$

$24 = 12 \times 2 + 0$

Recursive olarak kalan 0 olana kadar devam eder

ebob = 12

Fermat Teoremi: Eğer p asal ve a, p'ye bölünmeyen bir tamsayı ise o zaman $a^{p-1} \equiv \text{mod } p$ Ayrıca $a^p \equiv a \bmod p$

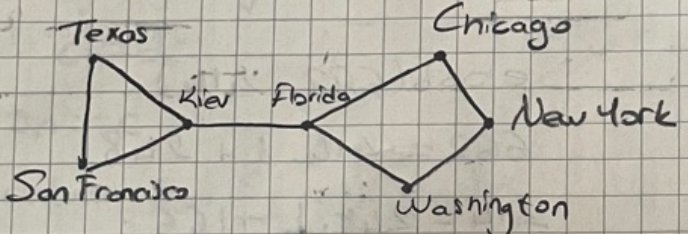
Graflar

$G = (V, E)$ grafı boş olmayan (V) düğümler kümesi ile yine boş olmayan kenarlar kümesi (E)'den oluşur.

Her kenarın bir veya iki köşesi vardır.

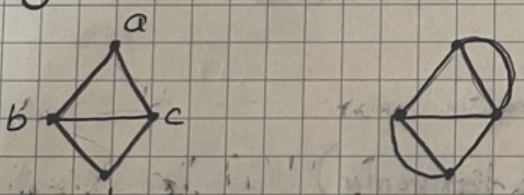
Düğümler veya kenarlar sayısı sonsuz ise sonsuz graf denir.

Sonlu ise sonlu graf denir.



İki düğüm arasında birden fazla ayrıt yoksa buna simple-graph (basit graf) denir.

İki düğüm arasında birden fazla ayrıt varsa buna multi-graph (çoklu graf) denir.

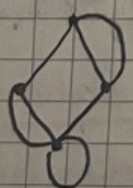


basit graf

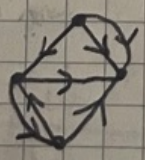
çoklu graf

Döngü içeren graflara pseudograph (sözde graf) denir.

Düğümler arası yön belirtilmişse buna directed graph (yönlü graf) denir. Düğümler arası yön belirtilmemişse buna undirected graph (yönsüz graf) denir



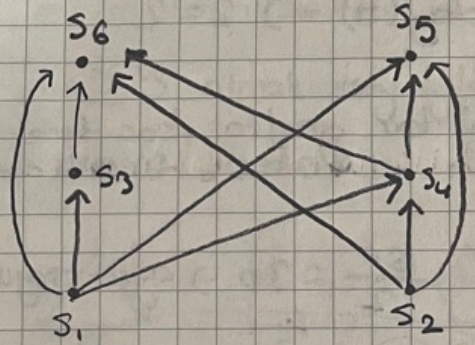
pseudograph



directed graph

Düğümler arası n tane ayrıt varsa buna n katlı graf denir.

Type Tip	Edges Kenar	Multiple Edges Allowed? Çoklu Kenar İzni	Loops Allowed? Döngü İzni
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple Directed Graph	Directed	No	No
Directed Multigraph	Directed	Yes	Yes
Mixed Graph	Directed and Undirected	Yes	Yes



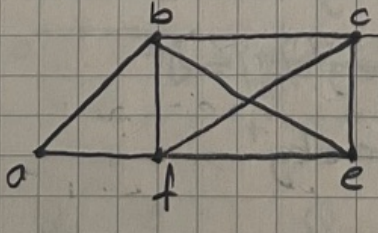
a precedence graph
(öncelik grafı)

S1 a := 0
S2 b := 1
S3 c := a+1
S4 d := b+a
S5 e := d+1
S6 e := c+d

G(V, E) grafının v köşesine komşu olan bütün köşelerinin kümesine v'nin komşuluğu denir. (adjacent) N(v) ile gösterilir.

Eğer e kenarı {u, v} birleştiriyorsa u ve v bağlıdır, bağlantılıdır denir. u ve v uç düğümler olarak adlandırılır.

Bir düğümün derecesi, düğüme bağlı ayrıtların sayısıdır. deg(v) ile gösterilir. (Eğer döngü varsa düğüm derecesine 2 ekler)

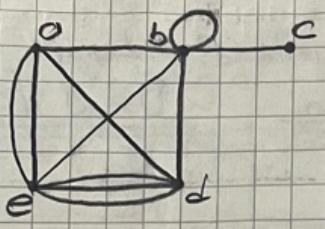


→ bağlantılı

→ izole edilmiş düğüm (ayrık, terk edilmiş)

deg(a) = 2 deg(e) = 3
deg(b) = 4 deg(f) = 4
deg(c) = 4 deg(g) = 0
deg(d) = 1

N(a) = {b, f} N(b) = {a, c, e, f} N(c) = {b, d, e, f}
N(d) = {c} N(e) = {b, c, f} N(f) = {a, b, c, e}
N(g) = ∅



$\deg(a)=4$ $\deg(e)=6$

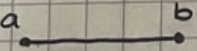
$\deg(b)=6$

$\deg(c)=1$

$\deg(d)=5$

Handshaking (El Sıkışma) Teoremi

$G(V,E)$ yönsüz bir graf olsun. Bu grafta m tane ayrıt varsa $2m=\sum_{v\in V}\deg(v)$



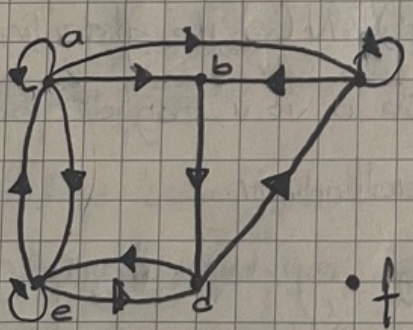
$2.1=\deg(a)+\deg(b) \Rightarrow 2=2$

ÖR: Herbirinin derecesi 6 olan 10 köşeli bir grafta kaç tane kenar vardır?

$6.10=60 \rightarrow$ dereceler toplamı, $\frac{60}{2}=30 \rightarrow$ ayrıt sayısı

$G(V,E)$ yönlü bir graf olsun. $2m=\sum \deg(v)$ $\quad 2m=60 \quad m=30$

$$\sum_{v\in V}\deg^-(v)=\sum_{v\in V}\deg^+(v)=|E|$$



girenler — çıkanlar

$\deg^-(a)=2$ $\deg^+(a)=4$

$\deg^+(c)=2$ $\deg^-(c)=3$

$\deg^-(d)=2$ $\deg^+(d)=2$

$\deg^-(b)=2$ $\deg^+(b)=1$

$\deg^-(e)=3$ $\deg^+(e)=3$

$\deg^-(f)=0$ $\deg^+(f)=1$

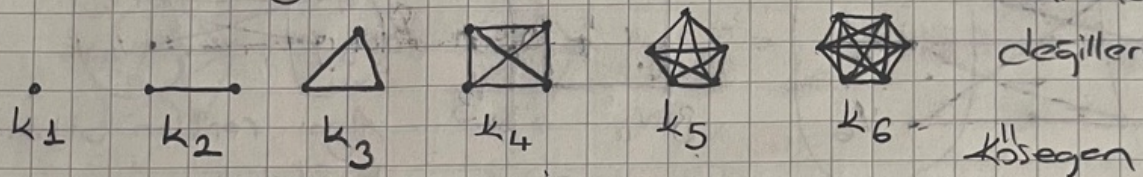
- Düğüme girenler $\deg^-$, düğümden çıkanlar $\deg^+$

★ Yönsüz bir grafta derecesi tek olan köşelerin sayısı çifttir

Bazı Özel Basit Graflar

Tam Graflar (Complete Graphs) İki kümeli (bipartite) değildirler

Her farklı köşe çifti arasında tam olarak 1 tane kenar bulunan basit graftır.

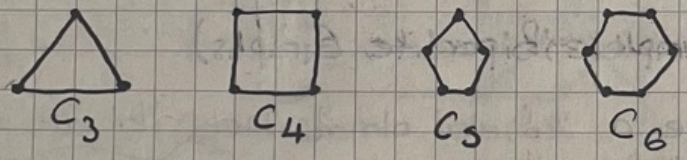


İki kümeli değillerdir.

köşegen

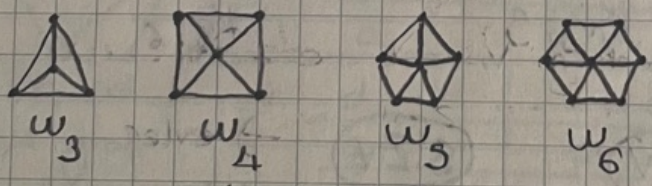
Çevrim Graflar (Cycle Graphs)

$n \geq 3$ olmak kaydıyla her bir düğüm sadece 2 düğüme bağlı olup devre içerir.

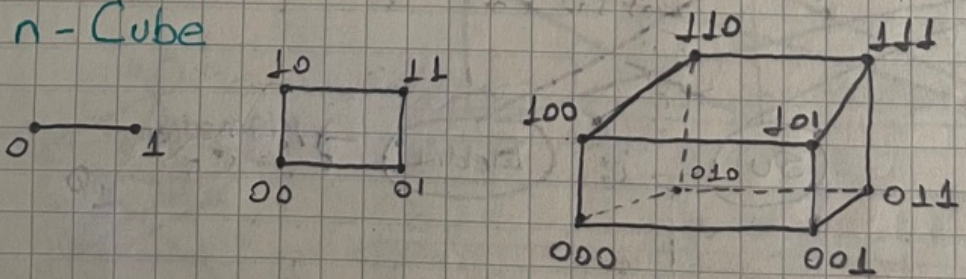


Çark Graflar (Wheels)

Çevrim graflara ilaveten bir merkez düğüm içerip bu düğümden diğer düğümlere ayrıtlar içerir.



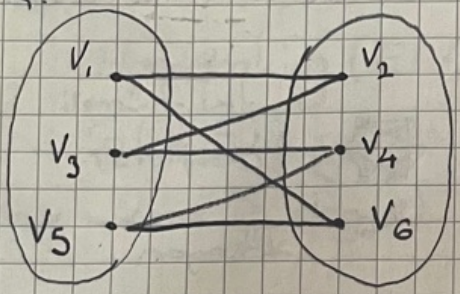
n-Cube

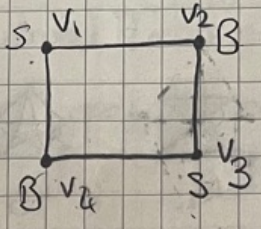


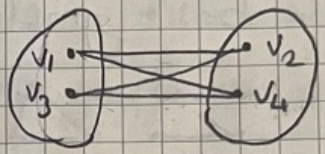
Q_1 Q_2 Q_3

Bipartite Graphs (İki Kümeli Graflar)

G grafındaki düğümler kümesi V'yi, V_1 ve V_2 alt kümeleri şeklinde ifade etmek ve V_1 ile V_2 düğümler kümesinin kendileri arasında ayrıt içermemesi durumunda G grafı iki kümeli bir graftır.





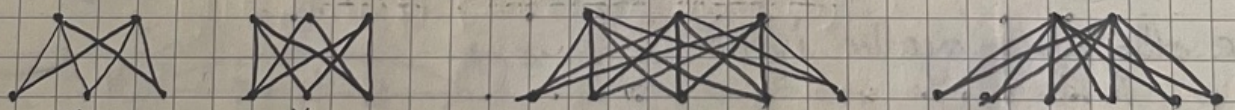


Aynı renkler birbirine bağlanmaz

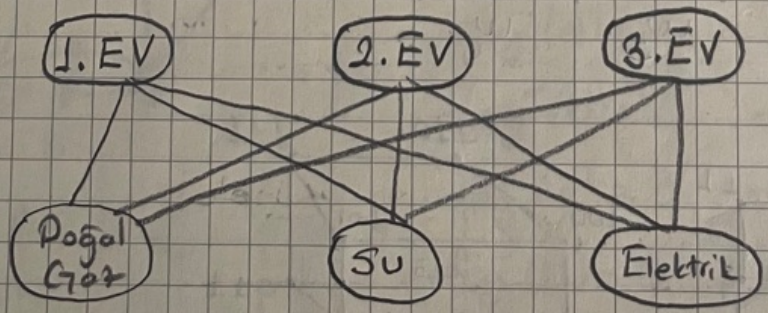
Teorem: Basit bir graf ancak ve ancak hiçbir komşu köşe aynı renge atanmayacak şekilde G'nin köşelerinden her birinin iki renkten birine atanması mümkünse iki kümeli olur. ★ K_n → iki kümeli değildir.

Tam İki Kümeli Graflar (Complete Bipartite Graphs)

Köşe kümesi sırasıyla m ve n köşeleri olmak üzere iki alt kümeye ayrılabilen ve iki köşe arasında bir kenarın bulunması ancak ve ancak bu köşeler farklı alt kümelerde ise mümkün olan graflardır.



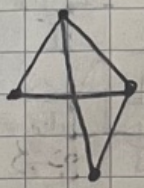
ÖR:



Eski Graflardan Yeni Graflar Elde Etme

$G=(V,E)$ grafının alt grafı $H=(W,F)$'dir. Burada $W \subseteq V$ ve $F \subseteq E$'dir. G'nin H alt grafı, G'nin özel bir grafıdır. $H \neq G$

K_5



K_5'in alt grafı

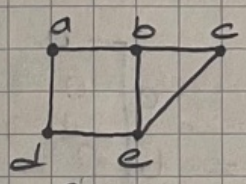
İki grafın birleşimi;

$G_1=(V_1, E_1)$

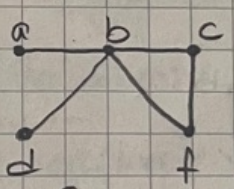
$G_2=(V_2, E_2)$

$V_1 \cup V_2$, $E_1 \cup E_2$

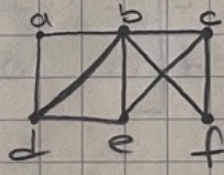
$G=G_1 \cup G_2$ oluşturur.



G_1

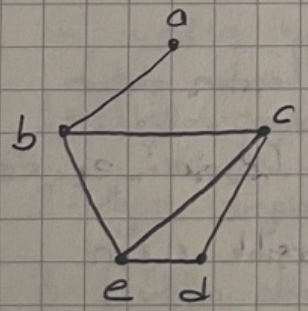


G_2



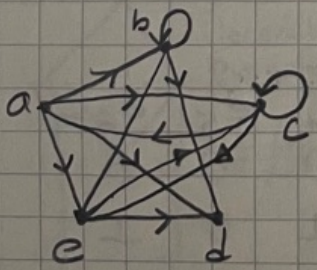
$G_1 \cup G_2$

Graf Gösterimi



Basit graf

Köşe	Komşu Köşeler
a	b, c, e
b	a
c	a, d, e
d	c, e
e	a, c, d

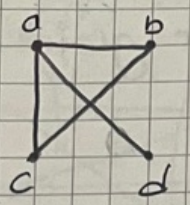


Köşe	Komşu Köşeler
a	b, c, d, e
b	b, d
c	a, c, e
d	
e	b, c, d

Komşuluk Matrisleri (Adjacency Matrices)

$A = [a_{ij}]$ $\quad a_{ij} = \begin{cases} 1 & (v_i, v_j) \text{ G'nin kenarı ise} \\ 0 & \text{değilse} \end{cases}$

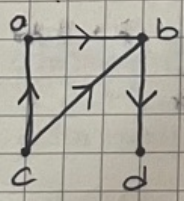
ÖR:



$$A_G = \begin{array}{c} \\ a \\ b \\ c \\ d \\ \\ \downarrow \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \\ \begin{array}{cccc} 3 & 2 & 2 & 1 \end{array} \end{array} \begin{array}{l} \\ \rightarrow 3 \\ \rightarrow 2 \\ \rightarrow 2 \\ \rightarrow 1 \\ \\ \end{array}$$

Yönsüz bir graf olduğu için sütun ve satırların toplamları, düğümlerin derecelerini verir.

ÖR:



$$A_G = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

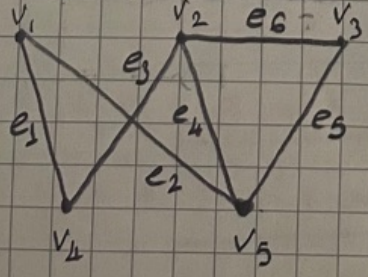
Yönlü graflarda satırlar, düğümden çıkanların derecesini (deg^+), sütunlar düğüme girenlerin derecesini (deg^-) verir.

Bağlılık Matrisleri (Incidence Matrices) 1 ve 0'lardan oluşur

$G(V, E)$ grafı olsun. $v_1, v_2, \ldots v_n$ G grafının köşeleri, $e_1, e_2, \ldots, e_m$ G grafının kenarları olsun. V ve E'ye karşılık $n \times m$ bağlılık matrisi $M = [m_{ij}]$'dir.

$$m_{ij} = \begin{cases} 1 & e_j \text{ kenarı } v_i \text{ köşesine bağlı ise} \\ 0 & \text{değilse} \end{cases}$$

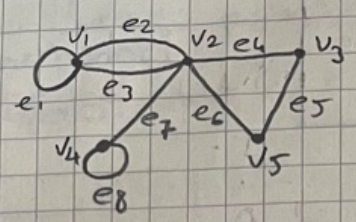
ÖR:



Yönsüz graf

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

ÖR:

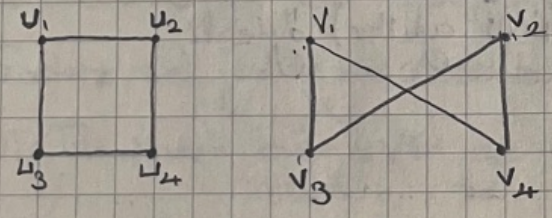


5 düğüm Graf

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_1	1	1	1	0	0	0	0	0
v_2	0	1	1	1	0	1	1	0
v_3	0	0	0	1	1	0	0	0
v_4	0	0	0	0	0	0	1	1
v_5	0	0	0	0	1	1	0	0

Isomorfizm (Isomorphism of Graphs) (Eş yapılık)

$G_1 = (V_1, E_1)$ $G_2 = (V_2, E_2)$. V_1'den V_2'ye birebir ve örten (one to one and onto) fonksiyon varsa; G_1'de komşu a ve b G_2'de $f(a), f(b)$ varsa G_1 ve G_2 isomorfizm graflardır.

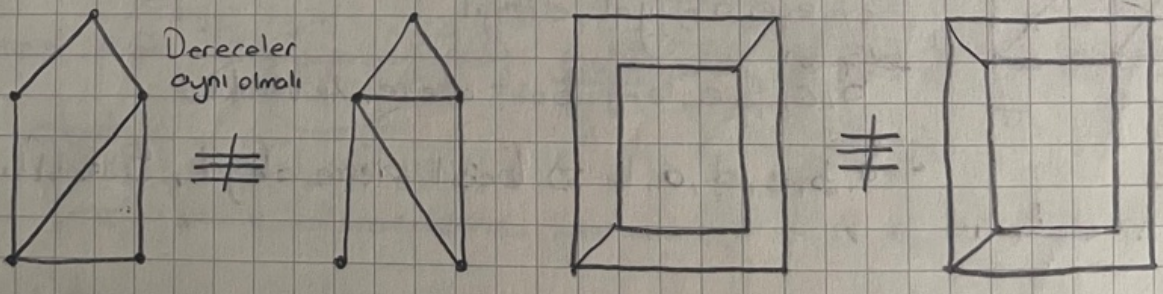


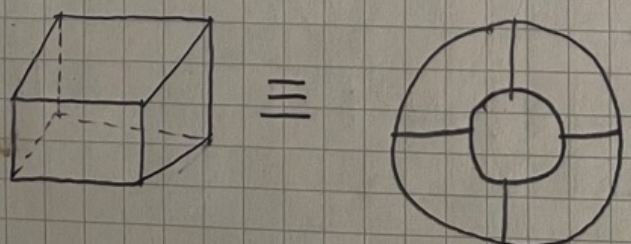
f fonksiyonu
$f(u_1) = v_1$
$f(u_2) = v_2$
$f(u_3) = v_3$
$f(u_4) = v_4$
} olup birebir ve örtendir.

İki grafın izomorfizm olup olmadıklarını anlamak için

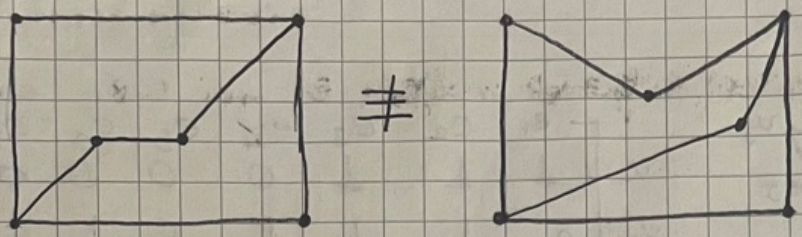
1- Düğüm sayıları
2- Ayrıt sayıları
3- Düğüm dereceleri ve komşu düğüm dereceleri
4- Yollar ve devreler

} Komşuluk matrisi çizilebilir. Kontrol edilir.

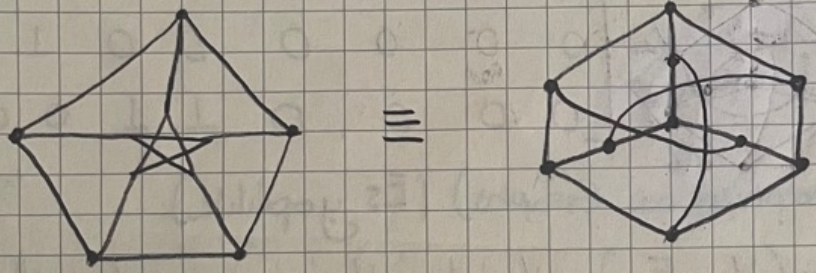




İsomorfizmdir.



İsomorfizm değildir.



İsomorfizmdir.

Yol ve Devre

n negatif olmayan tam sayı, G yönsüz graf. n uzunluklu yol, u'dan v'ye $e_1, e_2, e_3, \ldots, e_n$ dizisinde (G'deki ayrıtlar) öyle ki;

$e_1 = \{x_0, x_1\}$ birleştirir

$e_2 = \{x_1, x_2\}$ birleştirir

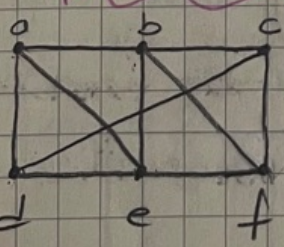
$e_n = \{x_{n-1}, x_n\}$ $x_0 = u$ (başlangıç) $x_n = v$ (son düğüm) dolayısıyla

yol $\rightarrow x_0, x_1, \ldots, x_n$ Eğer x_n, x_0'a bağlanabiliyorsa devre olur.

(Başlangıç noktasına geri dönülebiliyorsa)

★ Eğer başlangıç noktasında düğüm sonlanırsa devre olur.

★ Yol ve devre basitse aynı düğümden birden fazla kez geçilemez.



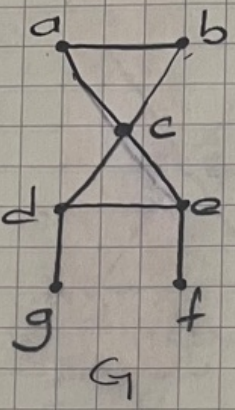
a, d, c, f → basit yol, 4 uzunluklu

d, e, c, a → yol değil

b, c, f, e, b → basit devre, 4 uzunluklu

a, b, e, d, a, b → basit devre değil, 5 uzunluklu

Tanım: Yönsüz bir grafta tüm düğümler arasında bir yol varsa bu grafa bağlı graf denir.

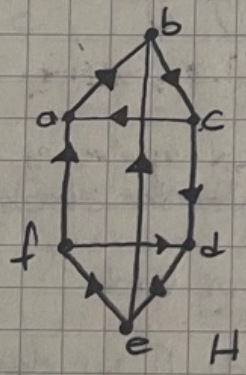


G

G grafı bağlı bir graftır.

Tanım: Yönlü bir grafta tüm düğüm çiftlerinin hem a'dan b'ye, hem de b'den a'ya bir yol varsa buna kuvvetli bağlı graf denir. Sadece tek yönlü yol varsa buna da zayıf bağlı graf denir.

NOT: Tüm kuvvetli bağlı graflar aynı zamanda zayıf bağlı graflardır.



H

H grafı kuvvetli bağlı graftır.

Euler Paths and Circuits Basit yol ve devre olmalı

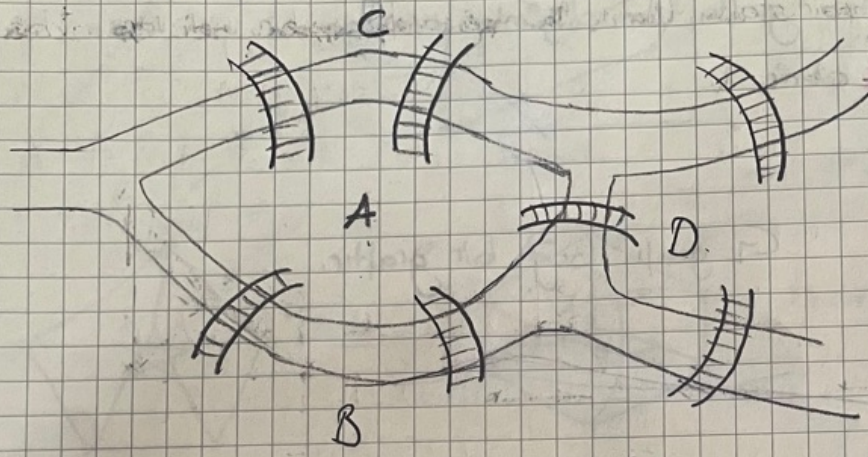
Bir G grafında tüm ayrıtları geçen basit bir devre varsa buna Euler devresi denir. Tüm ayrıtları geçen basit bir yol varsa buna da Euler yolu denir.

Grafın köşe derecelerine bakarak basitçe çözülebilir.

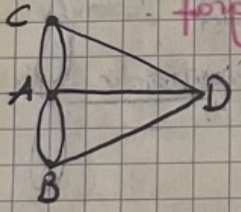
Hamilton zordur.

Euler devresi: G'nin tüm kenarlarını içeren basit devre
Euler yolu: G'nin tüm kenarlarını içeren basit yol

Königsberg Köprüsü



Herhangi bir bölgeden başlanılarak (A, B, C, D) tüm köprüleri bir kere geçmek şartıyla başlanıca bölgesine geri dönülebilir mi?



Königsberg köprüsü multigraf modeli
Her köprüye uğrayarak başladığımız noktaya geri dönmeyi. Gözümü yoktur.

★ G, en az iki düğüme sahip ve bütün düğümlerinin dereceleri çift olan bir multigraf ise bu durumda Euler devresi vardır.

★ En fazla 2 düğümün derecesi tek ise Euler yolu vardır. (Euler devresi yoktur)

procedure Euler (G: tüm düğümler çift dereceli multigraf)

devre := G'deki bir devre keyfi seçilen bir düğümden başlayıp ardışık kenarların eklenmesiyle bir yol oluşturulur ve başlanan köşeye geri döner.

H := Bu devrenin kenarlarıyla G'yi kaldırınız.

while H kenarlara sahipse

alt devre := H'daki bir düğümden başlayıp ayrıtlarında bu düğümde bitkin H'daki bir devre.

H := alt devre'nin ve tüm silinmiş düğümlerin kenarlarından oluşan H

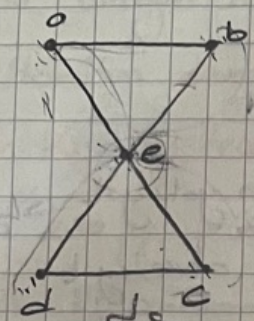
devre := uygun düğüme yerleştirilmiş alt devre'den oluşan devre

return devre {devre bir Euler devresidir}

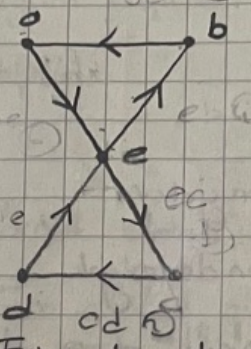
GIPTA

Teorem: Bir grafta tam olarak iki düğümün derecesi tek ise Euler yolu vardır, devresi yoktur.

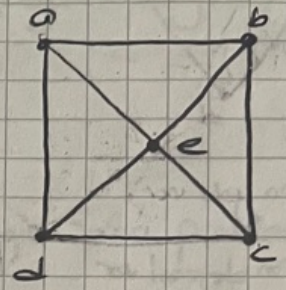
ÖR:



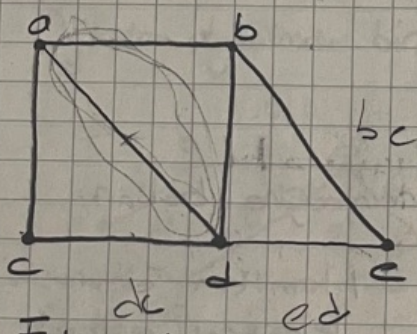
Euler yolu var
Euler devresi var



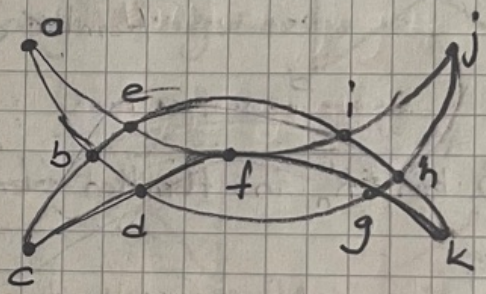
Euler yolu yok
Euler devresi var



Euler yolu yok
Euler devresi yok.



Euler yolu var
Euler devresi yok



Euler yolu var
Euler devresi var

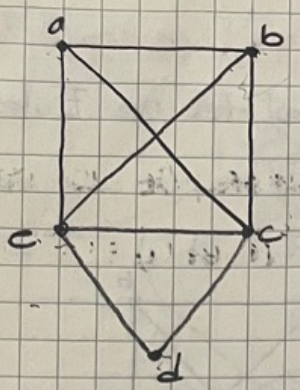
Hamilton Paths and Circuits

G gibi basit bir grafta her köşeden tam olarak bir kez geçilmesiyle oluşan basit yola Hamilton yolu, her köşeden tam olarak bir kez geçilmesiyle oluşan basit devreye Hamilton devresi denir.

$G=(V,E)$. $0 \le i < j \le n$ Eğer $V=\{x_0, x_1, \ldots, x_{n-1}, x_n\}$ $(x_i \neq x_j)$ ise G grafı Hamilton yolu içerir.

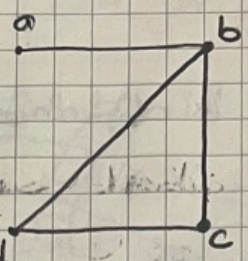
$n>0$ için eğer $x_0, x_1, \ldots, x_{n-1}, x_n, x_0$ basit devresi; Eğer $x_0, x_1, \ldots, x_{n-1}, x_n$ bir Hamilton yolu ise, Hamilton devreydir.

ÖR:



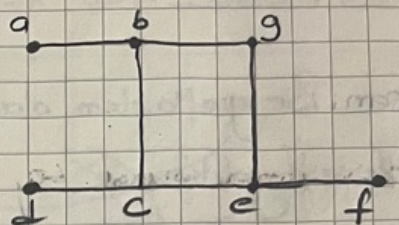
Hamilton yolu var

Hamilton devresi var



Hamilton yolu var

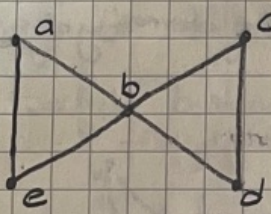
Hamilton devresi yok



Hamilton yolu yok

Hamilton devresi yok

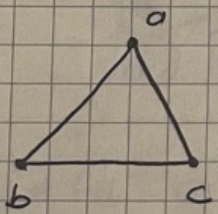
NOT: Hamilton devresinden daha küçük devre yoktur.



Hamilton yolu var

Hamilton devresi yok

ÖR: $n \geq 3$ için K_n 'in Hamilton devresi içerip içermediğine bakınız.



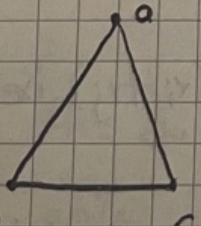
Hamilton yolu var

Hamilton devresi var.

Hamilton devresini kontrol etmek için
- Dirac $n/2$ → tüm düğümlerin derecesi
- Ore $\deg(u) + \deg(v) \geq n$

Dirac Teoremi

$n \geq 3$ olmak üzere, n köşeli basit bir G grafı, G'deki bütün düğümlerin derecesi en az $\frac{n}{2}$ ise G grafı Hamilton devresine sahiptir.



3 düğüm var

$3/2 = 1,5$

Hamilton devresi içerir.

Ore Teoremi

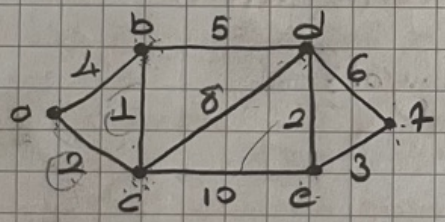
$n \geq 3$ olmak üzere G, n düğümlü basit bir graf ise G'deki komşu olmayan u ve v düğüm çiftleri için $(deg(u)+deg(v)) \geq n$ ise G Hamilton devresine sahiptir.

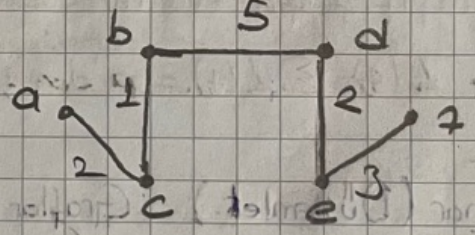
En Kısa Yol Problemi

En kısa yol problemlerine örnek olarak Gezgin Satıcı problemi örnek verilebilir. NP problemlerdir. Tam çözümü olmasa da Greedy algoritmaları ile çözülebilir. (NP - non-polynomial)

Dijkstra Algoritması $O(n^2)$

Dijkstra algoritması bir Greedy (açgözlü) algoritmadır.





↗ başlangıç düğümü

a	b	c	d	e	7
0	∞	∞	∞	∞	∞
0	4	2*	∞	∞	∞
0	3*	2	10	12	∞
0	3	2	8*	12	∞
0	3	2	8	10*	14
0	3	2	8	10	13

$O(n^2)$'ye göre çalışır.

procedure Dijkstra(G: Basit bir graf. tüm ayrıtlar pozitif)
{G grafında $a=v_0, v_1, \ldots, v_n$ e 4 köşeleri vardır. Eğer (v_i, v_j) kenarı G'de bulunmuyorsa $w(v_i, v_j)=\infty$ dur}

for i := 1 to n

$L(v_i) := \infty$

$L(a) := 0$

$S := \emptyset$

while $4 \notin S$

u := S'nin içinde olmayan ve L(u) değeri minimum olan köşe

$S := S \cup \{u\}$

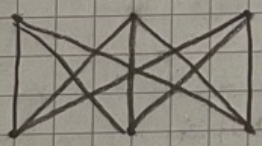
for S'de olmayan tüm köşeler için

if $L(u)+w(u,v) < L(v)$ then $L(v) := L(u)+w(u,v)$

return L(4) {L(4) = a'dan 4'ye en kısa yol}

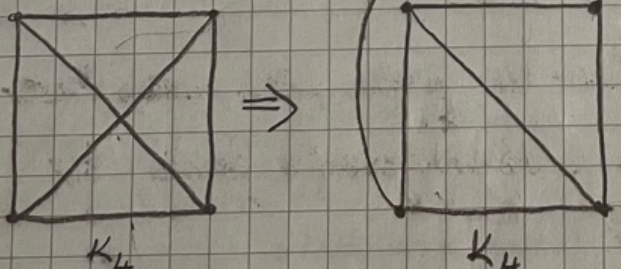
Planar (Düzlemsel) Graflar

Kenarları kesişmeksizin bir düzlemde çizilebilen grafa düzlemsel (planar) graf denir.



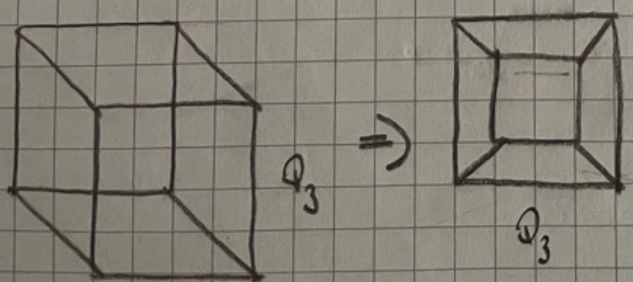
$K_{3,3}$

Düzlemsel değildir.



K_4 K_4

Planar (Düzlemsel) graftır.



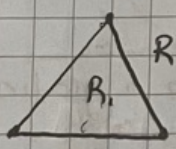
Q_3 Q_3

Planar graftır.

Euler Formülü

G; v köşeye, e kenara sahip bağlantılı basit bir graf olsun. r ise G'nin bir düzlemsel gösterimdeki bölge sayısı olsun. O halde $r = e - v + 2$ dir.

ÖR: Herbirinin derecesi 3 olan, 20 düğümden oluşan bir planar grafta kaç bölge vardır? (Handshaking)



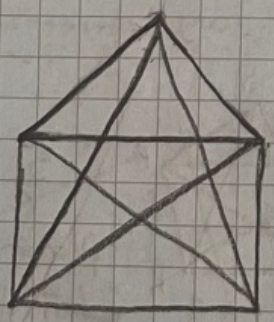
deg = 2 $\quad 3.2 = 6 \quad 6/2 = 3$ kenar(e) (El sıkışma teoremi)

düğüm 3 (v) $\quad r = 3 - 3 + 2 = 2$

Çözüm: $v = 20 \quad e = 20 \cdot 3/2 = 30 \quad r = 30 - 20 + 2 = 12$

SONUÇ1: G; v köşeli e kenarlı bağlantılı basit planar bir graf ise $v \geq 3$ olduğunda $e \leq 3v - 6$ olur.

SONUÇ2: G; bağlantılı basit bir planar graf ise G, 5'i aşmayan köşe derecesine sahiptir.



$10 \leq 3v - 6$

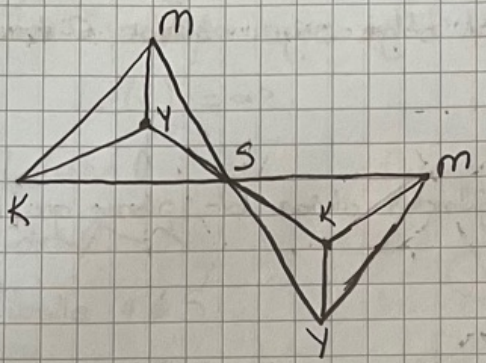
$10 \leq 15 - 6$

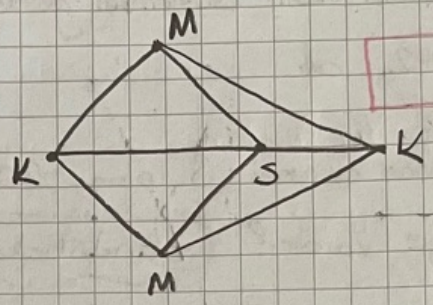
$10 \leq 9$ planar (düzlemsel) değildir.

SONUÇ3: Eğer bağlantılı düzlemsel basit bir grafın $v \geq 3$ olmak üzere v köşesi ve e kenarı mevcutsa ve 3 uzunluklu devresi yoksa bu durumda $e \leq 2v - 4$ olur.

Graf Boyama

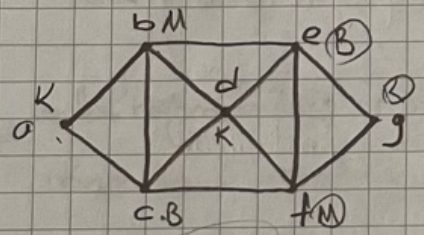
Herhangi iki komşu köşe aynı renge atanmayacak şekilde grafın her bir köşesine bir rengin atanmasına grafın renklendirilmesi denir.



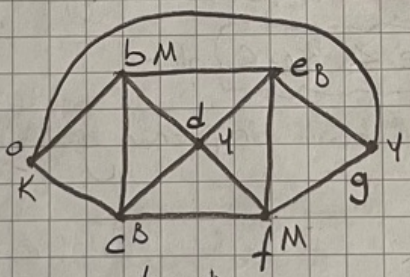


Bir grafın renklendirilmesinde gerekli en az renk sayısına o grafın renk sayısı veya kromatik sayısı denir.

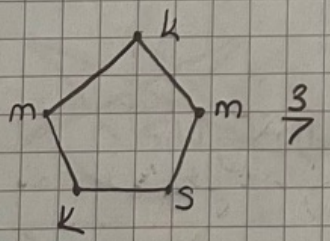
★★ Bir planar grafın kromatik sayısı 4'ten büyük olamaz.



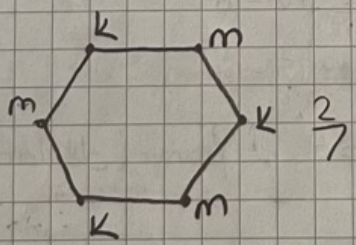
$k = 3$



$k = 4$

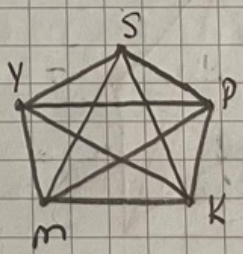


3 renk



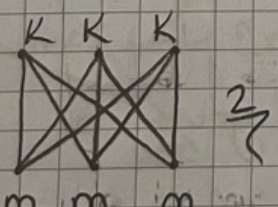
2 renk

2018-2019 Maaret sınav Sorusu



5 renk

planar graf değildir.



2 renk

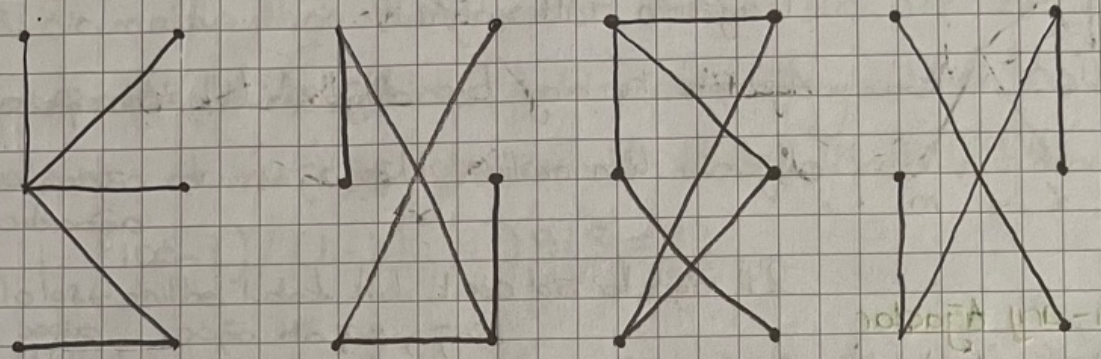
- K_n'in kromatik sayısı n'dir.
- $K_{m,n}$'in kromatik sayısı (m grubu aynı renk n grubu aynı renk) 2 olur.
- C_n'in kromatik sayısı;

  n çift ise 2
  n tek ise 3
  } olur.

AĞAÇLAR

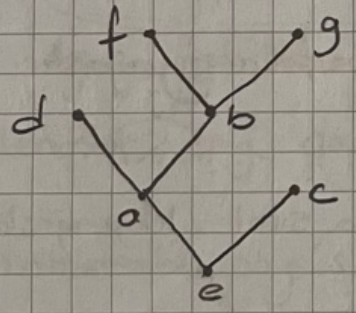
Devre içermeyen, bağlı yönsüz graflardır.

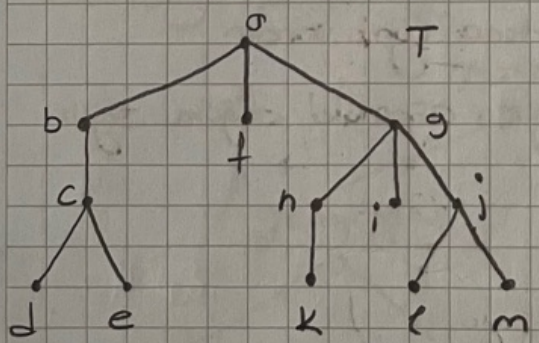
Herhangi iki düğüm arasında bir tek basit yol varsa bu graf ağaçtır.



ağaç ağaç ağaç değil ağaç değil

Bir ağaçtaki tüm düğümler bir düğüme bağlı kabul ediliyorsa bu düğüme kök düğüm (root) denir. Bu ağaca da köklü ağaç denir.





T grafında v düğümü kök hariç bir düğüm ise v'nin ebeveyni sadece u gibi tek bir düğümdür. (Parent)

v'nin ebeveyni u ise v u'nun çocuğudur. (child)

Aynı ebeveyne sahip olanlar kardeşlerdir. (siblings)

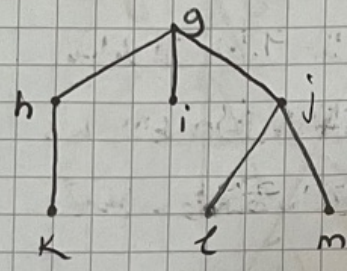
Bir düğümden köke ulaşmak için geçilen kök hariç her bir düğüm, o düğümün atasıdır. (ancestor)

Bir v düğümünün nesilleri (descendants), o düğümü ata kabul eden tüm düğümlerdir.

Bir düğümün çocuğu yoksa yaprak düğümdür. Çocuğu olanlar ise iç düğümdür.

Yukarıdaki T ağacında iç düğümler → a, b, c, g, h, j

yaprak düğümler → d, e, k, l, m, i, f



T ağacının alt ağacıdır.

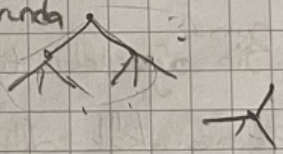
Ağaçtaki herhangi bir düğümü kök seçip o düğümün tüm neslinin olduğu ağacı ona ağacın alt ağacıdır.

2'li ağaçta sol çocuk kök kabul edilirse → sol alt ağaç
sağ → sağ alt ağaç

m-ary Ağaçlar

Eğer köklü bir ağaçta iç düğümlerin m'den fazla çocuğu yoksa buna m-ary ağaç denir.

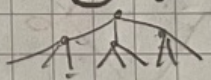
tam m sayıda olmak zorunda değil



$m=2$ için binary tree olarak adlandırılır.

Full m-ary Ağaçlar

Her iç düğümün tam olarak m tane çocuğu varsa bu full m-ary ağaçtır.



Sıralı Ağaç

Sıralı köklü bir ağaç, iç düğümlerin soldan sağa sıralı olduğu ağaçlardır.

Ağaçların Özellikleri

Bir ağaç n tane düğüme sahip ise $n-1$ tane ayrıt vardır.

Bir full m-ary ağacı i tane iç düğüm içeriyorsa ağaçtaki düğüm sayısı

$n = m \cdot i + 1$

Full m-ary ağacı için

n düğüm varsa iç düğüm sayısı $i = (n-1)/m$

i iç düğümlü ağaç $n = m \cdot i + 1$ düğüme sahipse yaprak sayısı

$l = [(m-1)n + 1]/m$

l yapraklı ağaç $n = (ml - 1)/(m-1)$ düğüme sahipse $i = (l-1)/(m-1)$

m-ary ağacının yüksekliği h ise en çok m^h yaprak vardır.

h yüksekliğindeki bir m-ary ağacı l yaprağa sahipse $h \geq \log_m l$

full m-ary ağacı için $h = \log_m l$ olur.

ÖR: Bir mektup zinciri oluşturulacaktır. Mektubu alan her kişi 4 kişiye gönderecektir. Kimse 1'den fazla mektup almayacaktır. 100 kişi bu mektubu okumuştur ve göndermemiştir. Kaç kişi mektubu göndermiştir?

$i = (100-1)/(4-1) = 99/3 = 33$

Köklü bir m-ary ağaçta tüm yapraklar h veya $h-1$ seviyesinde ise bu ağaca dengeli ağaç denir.

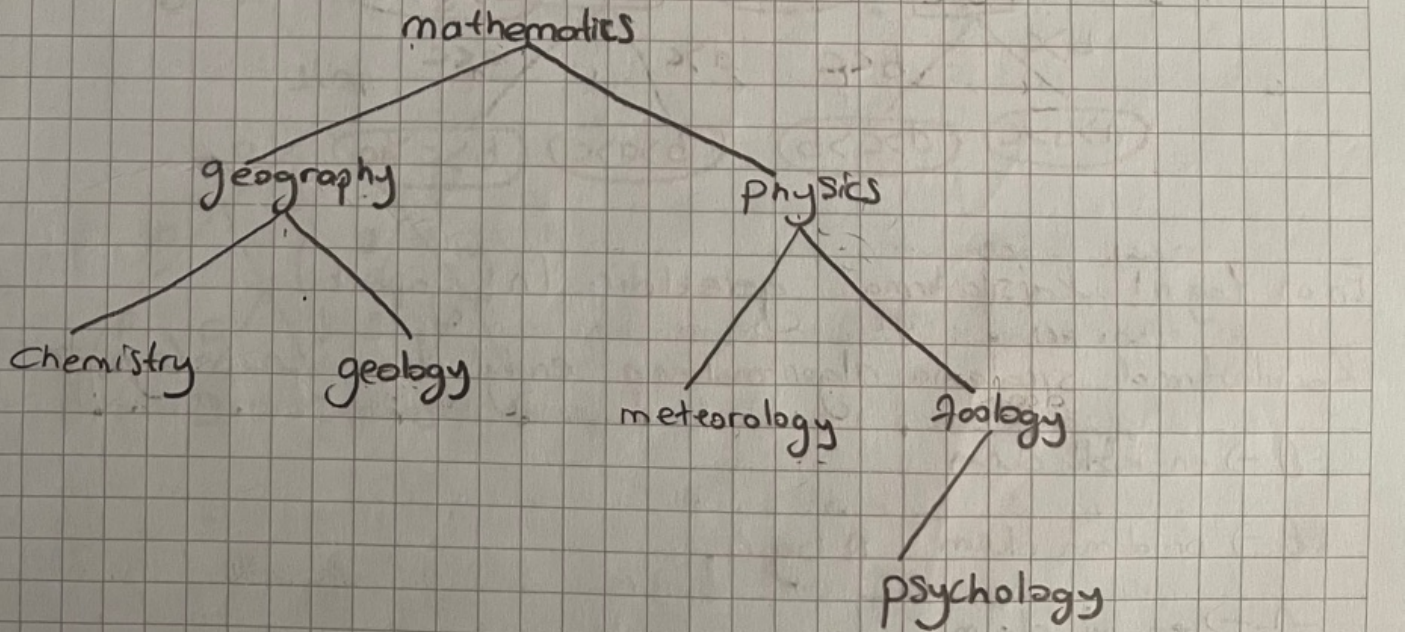
Seviye: Bir ağaçta kökten yaprağa giden en uzun yoldur.

Binary Search Tree

Ağaca yerleştirme yapılırken kökten küçük elemanlar sol çocuğa, büyük elemanlar sağ çocuğa yazılır.

mathematics, physics, geography, zoology, meteorology, geology, psychology, chemistry

a b c d e f g h i j k l m n o p q r s t u v w x y z



Binary Tree'ye eleman ekleme veya yerleştirme

procedure insertion(T: Binary Search Tree, x: eleman)

v := T ağacının "kök"ü

while v ≠ null and label(v) ≠ x

if x < label(v) then

if v'nin sol çocuğu ≠ null then v := v'nin sol çocuğu

else v'nin sol çocuğu olarak yeni bir düğüm ekle and v := null

else

if v'nin sağ çocuğu ≠ null then v := v'nin sağ çocuğu

else v'nin sağ çocuğu olarak yeni bir düğüm ekle and v := null

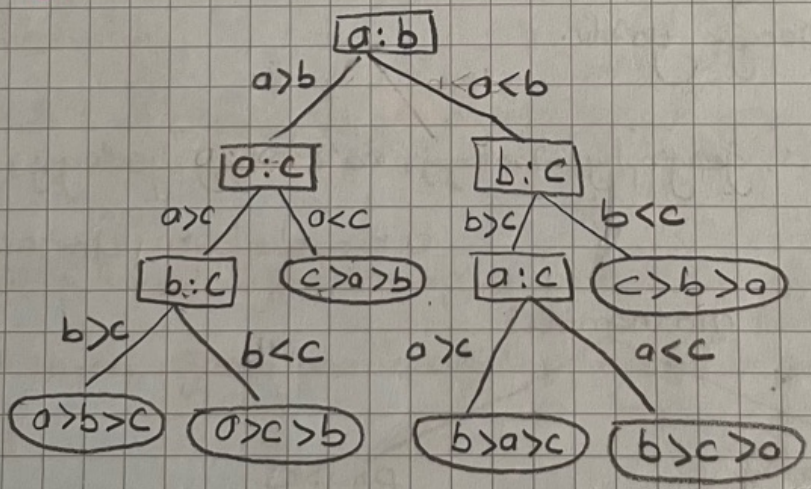
if T'nin "kök" = null then ağaca bir düğüm v ekle ve değerini x yap

else if v = null or label(v) ≠ x then x değerinde yeni düğüm oluştur v'yi bu düğüme ekle

return v {v = x'in konumu}

Karar Ağaçları (Decision Trees)

a b c



En az log n! karşılaştırma gereklidir. (n! yaprak)

Karşılaştırmalı sıralama algoritmalarının en iyi durumu Ω(n log n)dir.

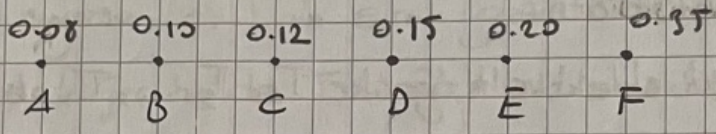
Ω → en kötü durum

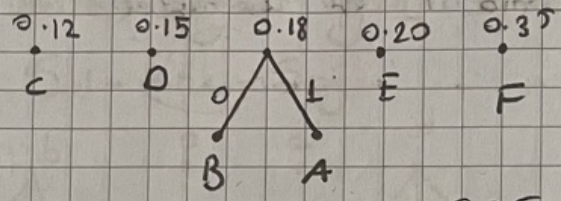
Θ → ortalama durum n log n

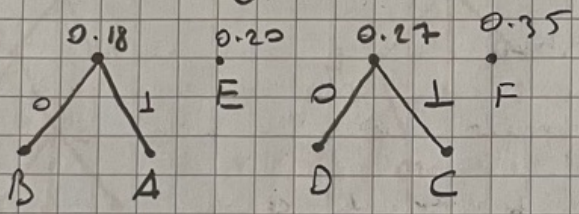
Ω → en iyi durum

Huffman Coding

Veri sıkıştırma algoritmasıdır.

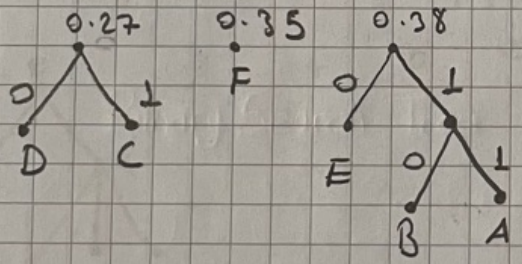


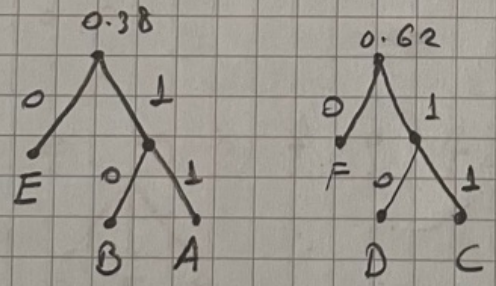


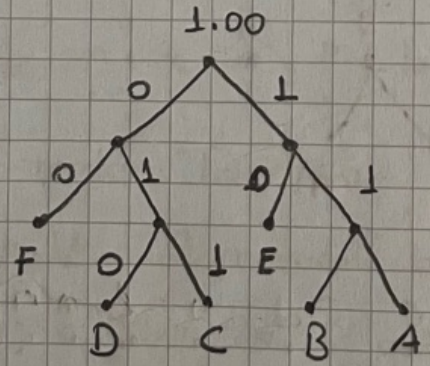


Huffman coding bir greedy algoritmadır.

NP







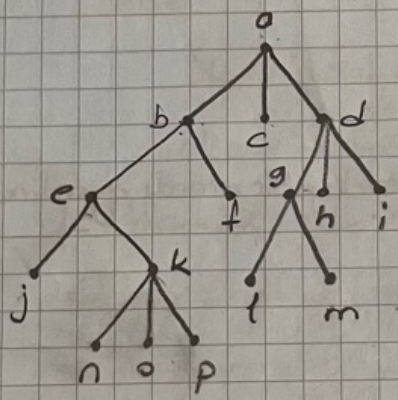
Ağaç Üzerinde Gezinme Algoritmaları

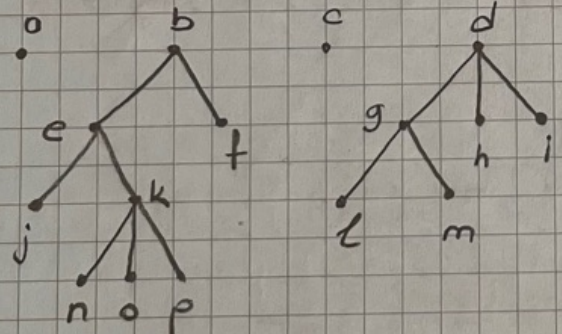
Preorder Traversal

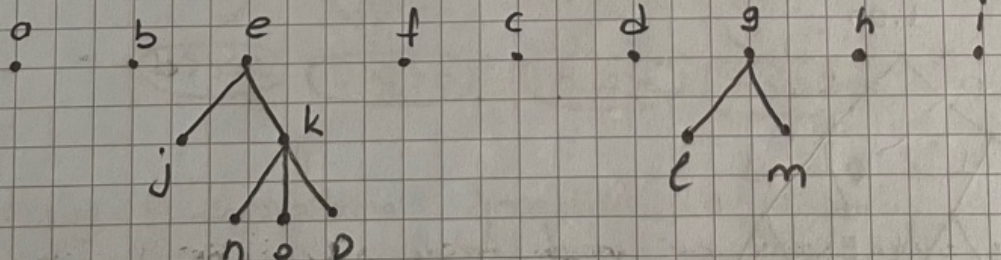
T, kökü r olan sıralı bir ağaç olsun. Eğer T sadece r'den oluşuyorsa; r, T'nin sıralı gezintisidir.

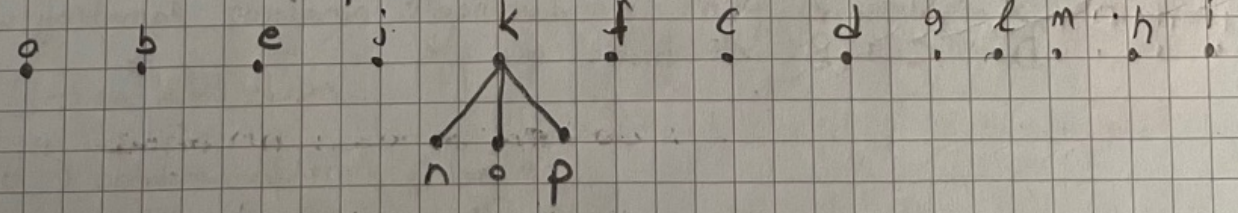
Diğer halde T'nin içindeki r'deki alt ağaçlar $T_1, T_2, \ldots, T_n$ olsun. Preorder gezinti r ile başlar. Daha sonra T_1 ve T_2 ziyaret edilir.

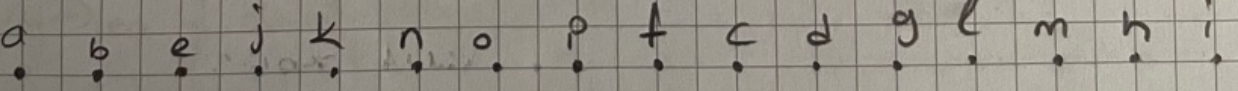
Kök - Sol - Sağ









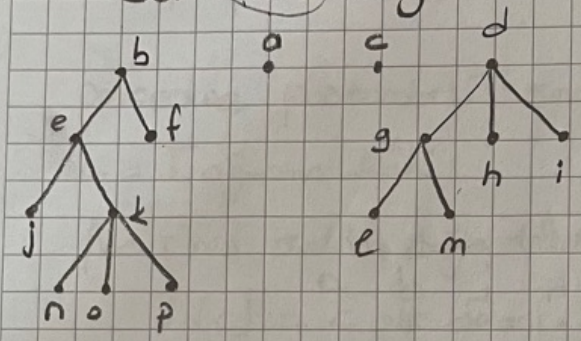


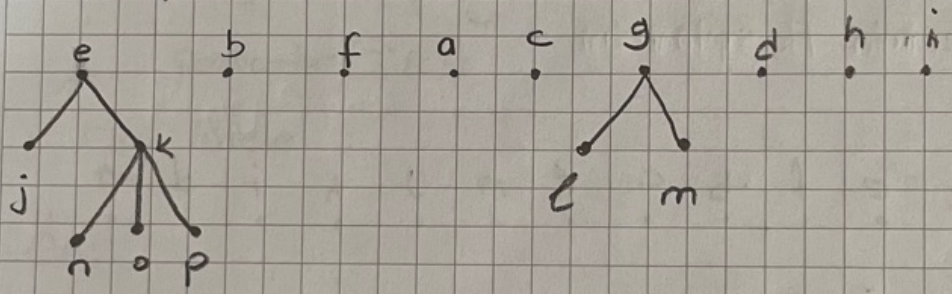
Inorder Traversal

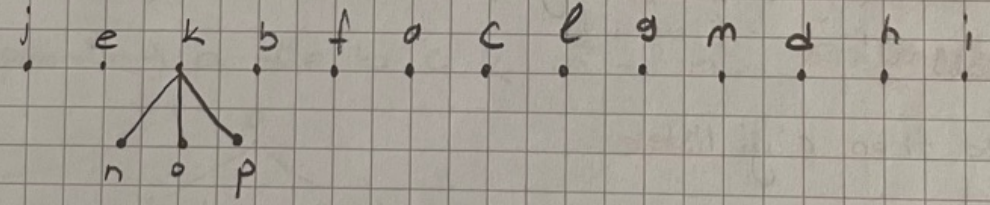
T, kökü r olan sıralı bir köklü ağaç olsun. Eğer T sadece r'den oluşuyorsa; r, T'nin sıralı gezintisidir.

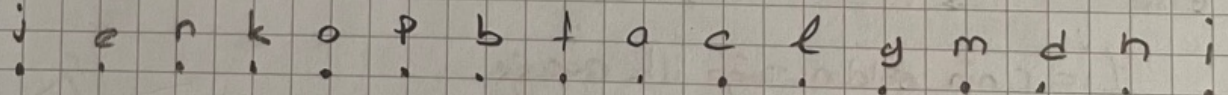
Diğer halde T'nin içindeki r'deki alt ağaçlar sıralı ziyaret T_1 ile başlar, sonra r ziyaret edilir. Daha sonra T_2 ziyaret edilir.

Sol - Kök - Sağ







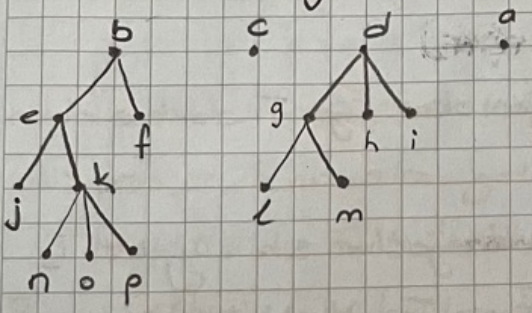


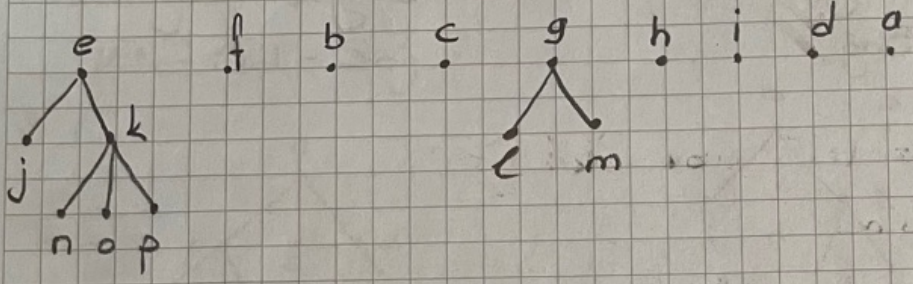
Postorder Traversal

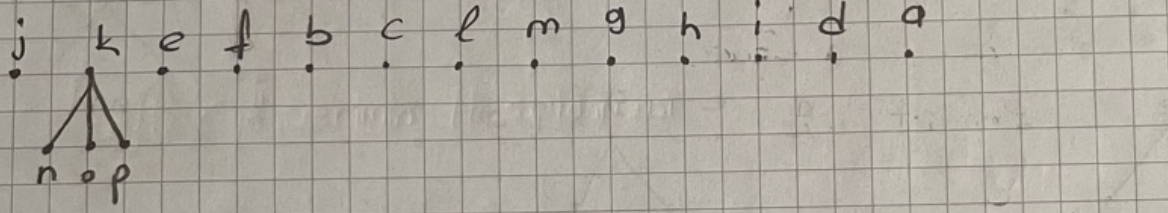
T, kökü r olan sıralı bir köklü ağaç olsun. Eğer T sadece r'den oluşuyorsa; r, T'nin sıralı gezintisidir.

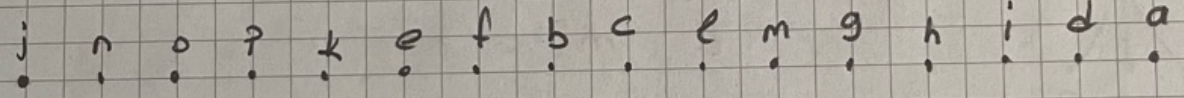
Diğer halde $T_1, T_2, \ldots, T_n$ r'deki soldan sağa alt ağaçlar olsun. Gezinti T_1 ile başlar daha sonra T_2'yi ziyaret eder. T_n'e kadar devam eder. En son r ziyaret edilir.

Sol - Sağ - Kök









procedure inorder(T: sıralı köklü ağaç)

r := T'nin kökü

if r yaprak ise then r'yi listele

else

l := r'nin soldan sağa ilk çocuğu

T(l) := l alt ağacı ∑ alt ağacın kökü

inorder(T(l))

r'yi listele

for r'nin l dışında tüm c çocukları

T(c) := c alt ağacı

inorder(T(c))

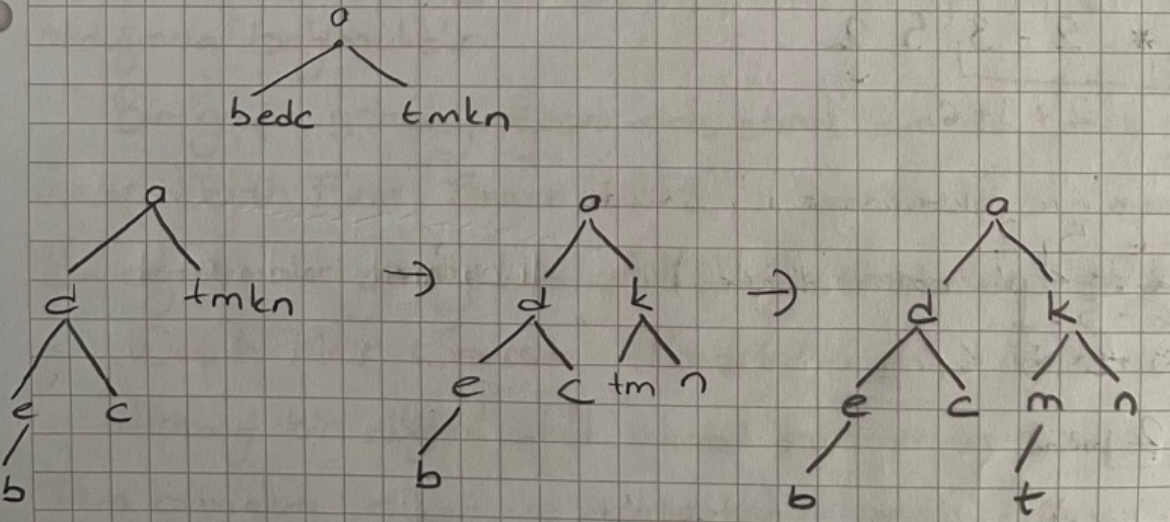
```
procedure preorder(T: sıralı kökllü ağaç)
r := T ağacının kökü
r'yi listele
for r'nin soldan sağa tüm c çocukları
    T(c) := c alt ağacı
    preorder(T(c))

procedure postorder(T: sıralı kökllü ağaç)
r := T ağacının kökü
for r'nin soldan sağa tüm c çocukları
    T(c) := c alt ağacı
    postorder(T(c))
r'yi listele
```

"
ÖR:

inorder → b e d c a t m k n ağacını çiziniz. postorder sıralamasını veriniz.

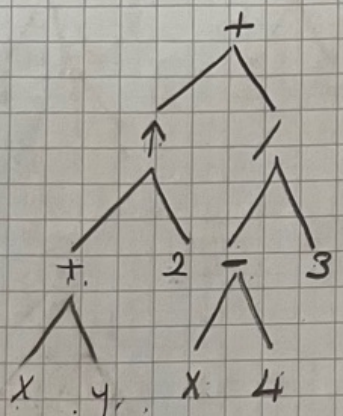
preorder → a d e b c k m t n



postorder → b e c d t m n k a

Infix, Postfix, Prefix

((x+y)↑2)+((x-4)/3) ?



(x+y↑2)+(x-4)/3

+↑+xy2/-x43

((x+y)↑2)+((x-4)/3) prefix formu → +↑+xy2/-x43

Prefix Gösterimin Hesaplanması

+ - * 2 3 5 / ↑ 2 3 4

$2^3 = 8$

+ - * 2 3 5 / 8 4

8/4 = 2

+ - * 2 3 5 2

2*3 = 6

+ - 6 5 2

6-5 = 1

+ 1 2

1+2 = 3

Postfix Gösterimin Hesaplanması

7 2 3 * - 4 ↑ 9 3 / +

2 * 3 = 6

7 6 - 4 ↑ 9 3 / +

7 - 6 = 1

1 4 ↑ 9 3 / +

$1^4 = 1$

1 9 3 / +

9 / 3 = 3

1 3 +

1 + 3 = 4

Spanning Trees (Kapsayan Ağaçlar)

G basit bir graf olsun. G'nin bir kapsayan ağacı G'nin bir alt grafı olan ve G'nin her düğümünü kapsayan bir ağaçtır

Teorem: Basit bir graf ancak ve ancak bir kapsayan ağaca sahip olduğunda bağlantılıdır.

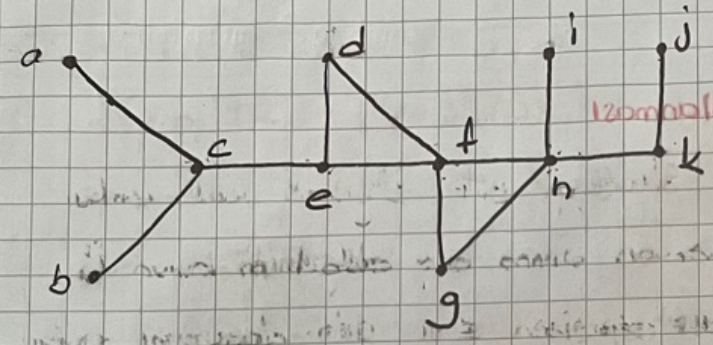
Bir grafın spanning ağacını elde etmek için iki farklı algoritma vardır. Depth First Search (DFS), Breadth First Search (BFS) Bu algoritmalar greedy (aç gözlü) algoritmalardır. (NP)

Depth First Search (Backtracking)

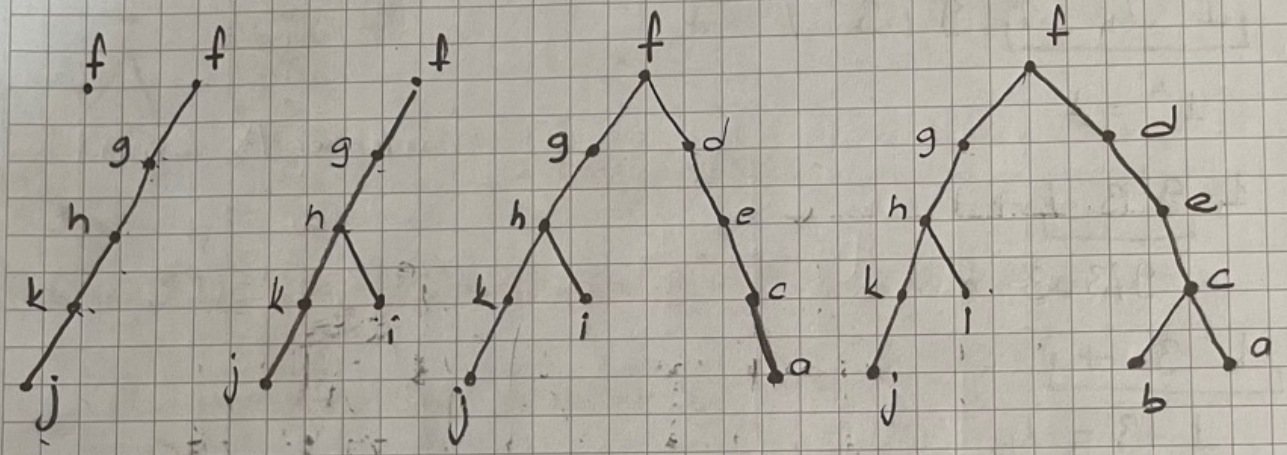
Herhangi bir düğümü kök kabul et ve en derine doğru ilerle. Tüm durumları gör ve birleştirme işlemini yap.

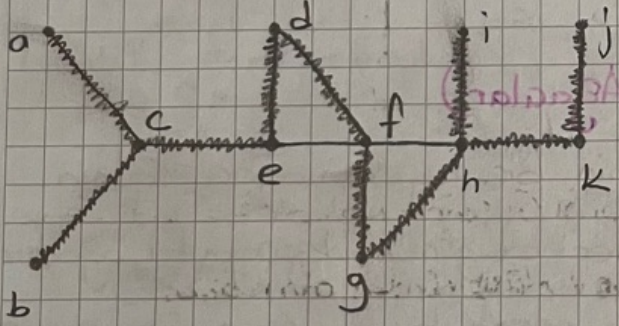
* Bu algoritma verimsiz çalışır.

Kökten uzaklaştığı için



f kök olsun



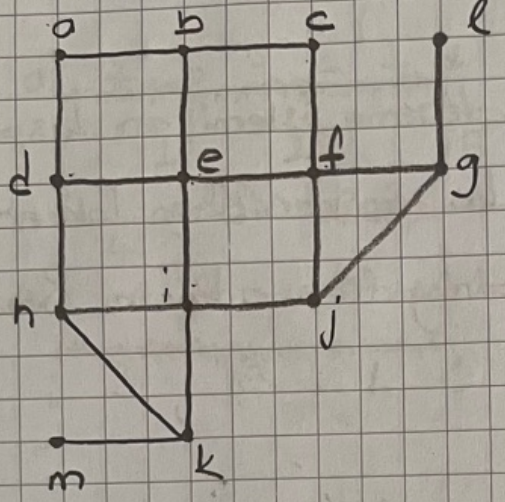


O(e) or O(n^2)
e: edges

procedure DFS (G: dugumler $v_1, v_2, \ldots, v_n$ baglantili graf)
T := sadece v_1 dügümünden olusan bir agac
visit(v_1)

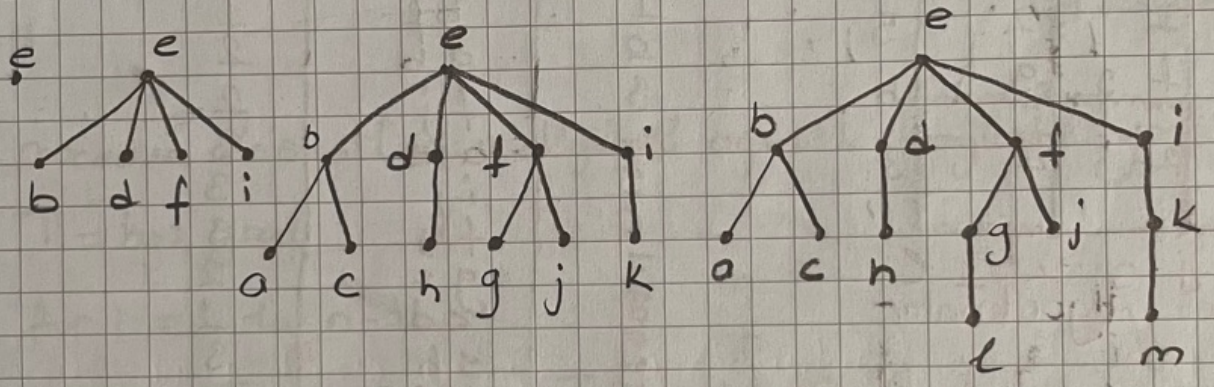
procedure visit(v: G'nin dügümü)
for v'ye komsu olan ve T'de henüz olmayan herbir w dügümü için
w dügümünü ve {v, w} kenarını T'ye ekle
visit(w)

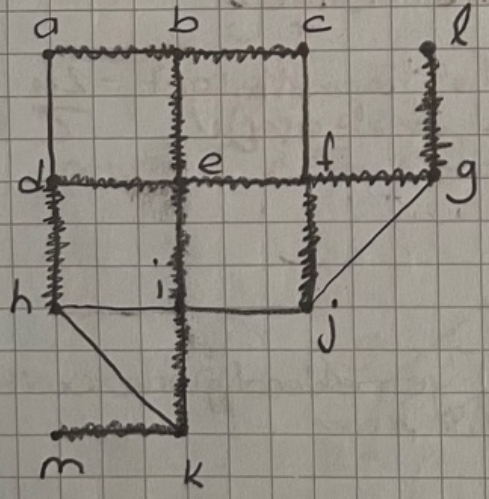
Breadth First Search



Herhangi bir düğümü kök kabul eder. Sonra alt düğümleri bulur. Bulduğu alt ağaçların da alt ağaçlarını bulur.

e kök kabul edilirse





procedure BFS(G: Bağlantılı graf, düğümler $v_1, v_2, \ldots, v_n$)

T := sadece v düğümünden oluşan ağaç

L := boş liste

v_1'i işleme alınmamış düğümlerden oluşan L'ye ekle

while L boş değilken

  ilk düğüm v'yi L'den çıkart

  for v'nin her bir w komşusu için

    w'yi L listesinin sonuna ekle

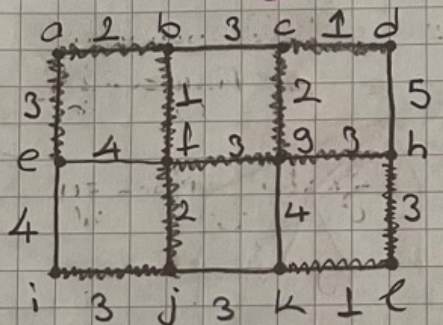
    w ve {v, w} kenarını T'ye ekle

Minimum Spanning Trees

Ayrıtlarının ağırlıkları toplamı mümkün olan en düşük değere sahip ağaçtır.

Bir binaya kablo döşenecektir. Kablo döşeme işlemi en kısa yola göre yapılır. Amaç en az kablayı kullanmaktır. Bunun için en kısa yolun ağacı alınır. Yani minimum spanning tree alınır. Bunun için iki yöntem vardır.

Prim's Algorithm



Tercih	Kenar	Ağırlık
1	bf	1
2	ab	2
3	fj	2
4	ae	3
5	ij	3
6	fg	3
7	gc	2
8	cd	1
9	gh	3
10	hℓ	3
11	kℓ	1
		+
		Maliyet = 24/2

procedure Prim(G: n düğümlü ağırlıklı bağlantılı yönsüz graf)

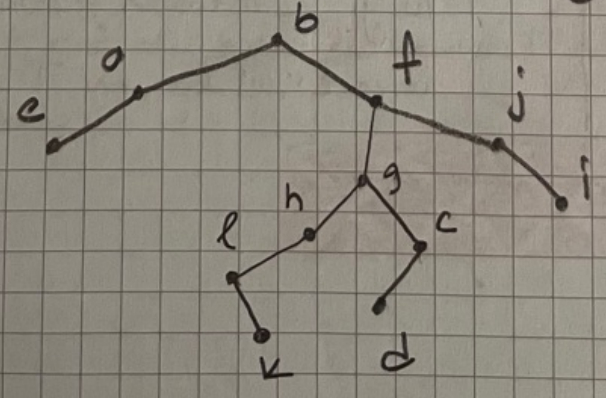
T := en az ağırlıklı kenar

for i := 1 to n-2

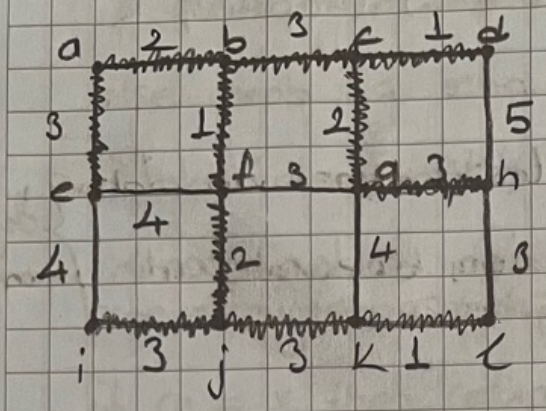
e := T'deki bir düğüme bağlı ve T'ye eklendiğinde basit devre oluşturmayacak minimum ağırlıklı kenar

T := e eklenmiş T

return T { T, G'nin minimum spanning ağacı }



Kruskal's Algorithm



Tercih	Kenar	Ağırlık
1	cd	1
2	kl	1
3	bf	1
4	cg	2
5	ab	2
6	fj	2
7	bc	3
8	jk	3
9	gh	3
10	ij	3
11	ae	+

Maliyet = 24

procedure Kruskal(G: n düğümlü ağırlıklı bağlantılı yönsüz graf)

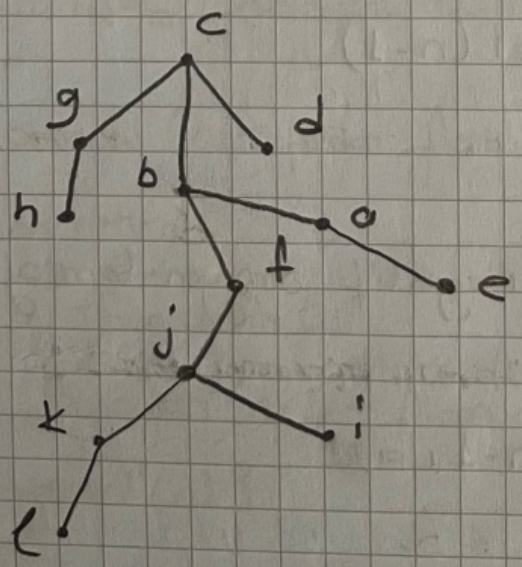
T := boş graf

for i := 1 to n-1

e := En az ağırlıklı ve T'ye eklendiğinde basit devre oluşturmayacak herhangi bir kenar.

T := e kenarı eklenmiş T

return T {T, G'nin minimum spanning ağacı}



Kuvvetli Tümevarım

Temel Basamak: P(1)'in doğru olduğunu göster.

Tümevarım Basamağı: $P(1) \wedge P(2) \dots P(k) \rightarrow P(k+1)$ tüm pozitif k tamsayıları için doğru olduğunu göster.

Özyinelemeli Basamak: Bir tamsayıdaki değeri tespit etmek için daha küçük tamsayılardaki değerlerden hesaplamak için bir kural verilir. } Özyinelemeli tanımlar

Temel basamak fonksiyonun sıfırdaki değerini belirtiyor.

ÖR: $f(0)=3$

$f(n+1)=2f(n)+3$, $f(1), f(2), f(3), f(4)=?$

$n=0$ için $f(1)=2f(0)+3 \rightarrow 2\cdot 3+3=9$

$n=1$ için $f(2)=2f(1)+3 \rightarrow 2\cdot 9+3=21$

$n=2$ için $f(3)=2f(2)+3 \rightarrow 2\cdot 21+3=45$

$n=3$ için $f(4)=2f(3)+3 \rightarrow 2\cdot 45+3=93$

Özyineli Algoritmalar

Bir problemi daha küçük girdilere sahip aynı problemlere indirgeyerek çözebilen algoritmaya "özyineli" algoritma denir.

```
procedure faktöriyel (n: negatif olmayan tamsayı)
if n=0 then return 1
else return n.faktöriyel(n-1)
{çıktı n! dir}

procedure kuvvet (n: negatif olmayan tamsayı, a: reel sayı)
if n=0 then return 1
else return a*kuvvet(n-1, a)
{çıktı a^n dir}
```

procedure ebob (a,b : negatif olmayan a<b şartı ile tamsayı)
if a=0 then return b
else ebob (a mod b, a)
{çıktı ebob(a,b)}

procedure linearsearch (i,j,x : $1 \leq i \leq j \leq n$ tamsayılar)
if $a_i = x$ then return i
else if i=j then return 0
else return linearSearch(i+1, j, x)
{eğer x $a_1, a_2, \ldots, a_n$ içindeyse bulundu değilse yok}

procedre binarySearch (i,j,x: $1 \leq i \leq j \leq n$ tamsayılar)
$m := \lfloor (i+j)/2 \rfloor$
if $a_m = x$ then return m
else if $x < a_m$ and $i < m$ then return binarySearch(i, m-1, x)
else if $x > a_m$ and $j > m$ then return binarySearch(m+1, j, x)
else return 0
{x; $a_1, a_2, \ldots, a_n$ içindeyse bulundu değilse yok}

procedure fibonacci (n: sıfırdan farklı pozitif tamsayı)
if n=0 then return 0
else if n=1 then return 1
else return fibonacci(n-1) + fibonacci(n-2)
{çıktı fibonacci(n)}

Merge Sort

8, 2, 4, 6, 9, 7, 10, 1, 5, 3

8, 2, 4, 6, 9 | 7, 10, 1, 5, 3

8, 2, 4 | 6, 9 | 7, 10, 1 | 5, 3

8, 2 | 4 | 6 | 9 | 7, 10 | 1 | 5 | 3

8 | 2

2 | 8 | 4 7, 10 | 1

2, 4, 8 | 6, 9 | 1, 7, 10 | 3, 5

2, 4, 6, 8, 9 | 1, 3, 5, 7, 10

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

procedure mergeSort($L = a_1, \dots, a_n$)

if $n > 1$ then

$m := \lfloor n/2 \rfloor$

$L_1 := a_1, a_2, \dots, a_m$

$L_2 := a_{m+1}, a_{m+2}, \dots, a_n$

$L :=$ merge(mergeSort(L_1), mergeSort(L_2))

{ L azalmayan sıradaki elemanlar olarak sıralanmıştır }

procedure merge(L_1, L_2 : sıralı listeler)

$L :=$ boş liste

while L_1 ve L_2 her ikisi de boş değilken

L_1 ve L_2'nin ilk elemanlarının en küçüğünü sil, bunu

L'nin sağ boş tarafına yerleştir.

if bu silme sonunda liste boş ise then öteki listeden tüm elemanları sil ve bu elemanları L'ye yerleştir.

return L

MATEMATİKSEL TÜMEVARIM

Matematiksel tümevarım genellikle tüm n pozitif tamsayıları için P(n) ifadesinin doğru olduğunu ispatlamak için kullanılır. Burada P(n) bir önerme fonksiyonudur.

Matematiksel tümevarımle ispat iki basamaktan oluşur. P(1) ifadesinin doğru olduğunu gösterdiğimiz temel basamak ve bütün pozitif k tamsayıları için P(k) doğru ise P(k+1)'in de doğru olduğunu gösterdiğimiz tümevarım basamağı

TEMEL BASAMAK ⇒ P(1) ifadesinin doğru olduğunu göstermek

TÜMEVARIM BASAMAĞI ⇒ P(k) → P(k+1) koşullu ifadesinin tüm pozitif k tamsayıları için doğru olduğunu göstermek.

ÖR: Eğer n pozitif bir tamsayıysa aşağıdaki eşitliğin doğru olduğunu gösteriniz

$$1+2+\dots+n=\frac{n\cdot(n+1)}{2}$$

Temel basamak: P(1) doğrudur. n=1 için $1=\frac{1\cdot(1+1)}{2}$

Tümevarım basamağı: P(k) ifadesinin rastgele seçilen bir pozitif k tamsayısı için doğru olduğu varsayılır. Yani

$1+2+\dots+k=\frac{k\cdot(k+1)}{2}$ eşitliğinin doğru olduğu varsayılır.

P(k+1)'in de doğru olduğunu göstermemiz gerekiyor. Eşitliğin her iki tarafına k+1 ekle

$$1+2+\dots+k+(k+1)=\frac{k\cdot(k+1)}{2}+k+1=\frac{k\cdot(k+1)+2(k+1)}{2}$$

$$=\frac{(k+1)(k+2)}{2}$$ → P(k)'nın doğru olduğu varsayımıyla P(k+1)'in de doğru old. gösterildi.

Eşitsizliklerin İspatlanması

Tüm pozitif n tamsayıları için $n < 2^n$ doğru old. matematiksel tümevarım ile ispatlayınız.

TEMEL BASAMAK → $P(1)$ doğrudur. Çünkü $1 < 2^1$

TÜMEVARIM BASAMAĞI → $P(k)$ ifadesinin doğru old. varsayımıyla $P(k+1)$'in de doğru olduğunu göster.

$P(k+1) = k+1 < 2^{k+1}$ Bu koşullu ifadenin k pozitif tamsayısı için doğru old. göstermek için öncelikle $k < 2^k$ eşitsizliğinin her iki tarafına da 1 ekle. ($1 \leq 2^k$ doğru old. biliyoruz)

$k+1 < 2^k + 1 \leq 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$

Bölünebilirlik Kurallarının İspatlanması

n negatif olmayan bir tamsayı olmak üzere $7^{n+2} + 8^{2n+1}$'in 57'ye tam bölündüğünü gösteriniz.

TEMEL BASAMAK → $P(0)$ için $7^{0+2} + 8^{2\cdot 0+1} = 7^2 + 8^1 = 57$ doğrudur.

TÜMEVARIM BASAMAĞI → $P(k)$: $7^{k+2} + 8^{2k+1}$ 57'ye tam bölünebildiğini varsayıyoruz.

$$7^{(k+1)+2} + 8^{2(k+1)+1} = 7^{k+3} + 8^{2k+3}$$

$$= 7 \cdot 7^{k+2} + 8^2 \cdot 8^{2k+1}$$

$$= 7 \cdot 7^{k+2} + 64 \cdot 8^{2k+1}$$

$$= 7(7^{k+2} + 8^{2k+1}) + 57 \cdot 8^{2k+1}$$

Sayma

İleri Sayma Teknikleri

Hanoi kuleleri

$H_n = 2H_{n-1} + 1 = 2(2H_{n-2}+1)+1 = 2(2(2H_{n-3}+1)+1)+1$

$= 2^3 H_{n-3} + 1 + 2 + 4$

$\vdots$

$2^{n-1} H_1 + 2^{n-2} + \dots + 4 + 2 + 1 = 2^{n-1} + 2^{n-2} + \dots + 4 + 2 + 1 = 2^n - 1$

Doğrusal Özyinelemeli

Sabit katsayılı k. dereceden doğrusal bir homojen özyineleme ilişkisi

$a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2} + \dots + c_k \cdot a_{n-k}$

$c_1, c_2, \dots, c_k$ reel sayılar ve $c_k \neq 0$

$P_n = 1.11 \cdot P_{n-1}$ → I. dereceden doğrusal homojen

$a_n = a_{n-1} + a_{n-2}^2$ → Doğrusal değildir (karesi var). Homojendir.

$H_n = 2H_{n-1} + 1$ → Homojen değildir (1'den dolayı)

$B_n = n \cdot B_n - 1$ → Sabit katsayılı değil (n)

$f_n = f_{n-1} + f_{n-2}$ → 2. dereceden homojen

Doğrusal Homojen Özyineli İlişkilerin Sabit Katsayılarla Çözümü

Teorem 1: c_1 ve c_2 gerçel sayılar $r^2 - c_1 r - c_2 = 0$ eşitliğinin iki ayrı kökleri r_1 ve r_2 olsun. Bu durumda ancak ve ancak α_1 ve α_2 sabitler olmak üzere $n = 0, 1, 2, \dots$ için $a_n = \alpha_1 \cdot r_1^n + \alpha_2 \cdot r_2^n$ eşitliğinin sağlandığı durumda a_n dizisi

$a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2}$ özyinelemeli ilişkinin çözümüdür.

ÖR: $a_0 = 2$, $a_1 = 7$ $a_n = a_{n-1} + 2a_{n-2}$ çözümünü bul.

$a_n = a_{n-1} + 2a_{n-2}$

$\downarrow \quad \downarrow \quad \downarrow$

$r^2 = r + 2$

ÖR: Çıkarım kurallarını kullanarak aşağıda verilen önermelerin sonucunu doğrulayınız.

$p \to r \equiv p' \vee r$

$q \to r \equiv q' \vee r$

$p \to r$

$q \to r$

$\neg(p \vee r) \to p' \wedge r'$ sadeleştirme $\to p'$

$\therefore \neg r$

ÖR: Aşağıda verilen prosedürde $n=4$ için geri dönen s değeri kaç olur? Prosedürün Big-O notasyonunu n cinsinden bulunuz.

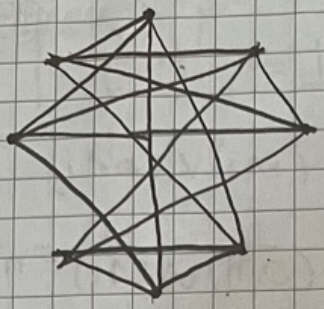
```
Procedure Sum(n: positive integer)
s := 0
for i := 1 to n
    for j := 1 to i
        s := s + j
return s
```

i	j	s
1	1	1
2	1	2
2	2	4
3	1	5
3	2	7
3	3	10
4	1	11
4	2	13
4	3	16
4	4	20

$s = 20$

$O(n^2)$?

ÖR:



Dirac'in teoremine göre aşağıda verilen grafın Hamilton devresi içerip içermediğini belirle.

Tüm düğümlerin derecesi en az $n/2$ olmalı.

8 düğüm vardır. $8/2 = 4$ dereceli olmalı.

Hamilton devresi içerir.

ÖR: İki tek sayının çarpımının tek olduğunu doğrudan İspat kullanarak ispatlayınız.

$\forall n (P(n)) \rightarrow Q(n)$

$n = 2k+1$

$(2k+1)(2k+3) = 4k^2 + \underbrace{6k + 2k}_{8k} + 3 = \underbrace{4k^2 + 8k + 3}$

k'nın bütün değerleri için tek olacaktır.

ÖR: $\underbrace{x'y'z' + x'y'z}_{x'y'(z'+z)} + \underbrace{x'yz' + xyz}_{y(x'z' + xz)}$

$= x'y'$

ÖR:

$$A=\begin{bmatrix}1 & 0\\ 0 & 1\\ 1 & 0\end{bmatrix} \quad B=\begin{bmatrix}1 & 1 & 1\\ 0 & 1 & 1\end{bmatrix}$$ boole carpimi?

$$=\begin{bmatrix}(1\wedge1)\vee(0\wedge0) & (1\wedge1)\vee(0\wedge1) & (1\wedge1)\vee(0\wedge1)\\ (0\wedge1)\vee(1\wedge0) & (0\wedge1)\vee(1\wedge1) & (0\wedge1)\vee(1\wedge1)\\ (1\wedge1)\vee(0\wedge0) & (1\wedge1)\vee(0\wedge1) & (1\wedge1)\vee(0\wedge1)\end{bmatrix}$$

$$=\begin{bmatrix}1 & 1 & 1\\ 0 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$2\cdot2\cdot2\cdot2\cdot2\cdot2\cdot2\cdot2\cdot2 = 2^{512}$ (128, 256)

ÖR:

$$\sum_{i=1}^{10} 3 = 3+3+3+3+3+3+3+3+3+3 = 30$$

$$\sum_{j=0}^{8}(2^{j+1}-2^{j}) = (2^1-2^0)+(2^2-2^1)+(2^3-2^2)\ldots \quad 8-4=4 \quad 64$$

$$\sum_{j=0}^{n} ar^j = \begin{cases}\frac{ar^{n+1}-a}{r-1} & \text{eğer } r\neq1\\ (n+1)a & \text{eğer } r=1\end{cases} \qquad \frac{1\cdot2^9-1}{2-1} = 511$$

$$1+(2^1+2^2+2^3+2^4+2^5+2^6+2^7+2^8) = (2+4+8+16+32+64$$

128

$$\sum_{i=0}^{2}\sum_{j=0}^{3}(2i+3j) = \sum_{i=0}^{2} 2i+(2i+3)+(2i+6)+(2i+9)$$

$$=(3+6+9)+(2+5+8+11)+(4+7+10+13) = 78$$

18 + 26 + (34) =

34
26

$$\prod_{i=1}^{100} (-1)^i = 1$$

ÖR $a_n = 2a_{n-1} - 1 \quad a_0 = 1$

n=1 için $a_1 = 1$ n=2 için $a_2 = 1$ n=3 için $a_3 = 1$

$a_n = 1$

ÖR: Bir bakteri kolonisinde bakterilerin sayısı her saatte üç katına çıkmaktadır. Bakterilerin sayısının n saat sonra kaç olacağını bulmak için gerekli tekrarlı bağıntıyı bulunuz. Başlangıçta 100 bakteri varsa 10 saat sonra bakteri sayısı kaç olur?

$a_n = 3a_{n-1} \quad 3a_{n-2} + 2a_{n-1} - a_n = 0 \quad 3r^2 + 2r - 1 = 0$

$\Delta = b^2 - 4ac = 2^2 - 4 \cdot 3(-1) = 16 \quad r_{1,2} = \frac{-2 \mp \sqrt{16}}{2 \cdot 3} = \frac{-2+4}{6} = \frac{1}{3} \quad \frac{-2-4}{6} = -1$

n=0 için $a_n = \alpha_1 (\frac{1}{3})^n + \alpha_2 (-1)^n$

n=1 için $a_0 = \alpha_1 + \alpha_2 \Rightarrow 1 = \alpha_1 + \alpha_2 \quad \frac{1 = \frac{\alpha_1}{3} - \alpha_2}{3 = \frac{4}{3}\alpha \Rightarrow \alpha_1 = \frac{9}{4}} \quad \alpha_2 = 1 - \frac{9}{4} = \frac{-5}{4}$

$a_1 = \frac{\alpha_1}{3} - \alpha_2$

$a_n = \frac{9}{4}(\frac{1}{3})^n - \frac{5}{4}(-1)^n$

ÖR: n elemanlı bir dizinin palindrom olup olmadığını bulan algoritma?

```
procedure palindromeCheck(a1, a2, a3, ... an : string)
    answer := true
    for i >= 1 to n/2
        if ai ≠ an-1-i then answer := false
    end
```

O(n)

$a_{10} = ?$

$3^{10} = 3 \cdot 10$

ÖR: n tamsayının minimumunu bulan recursive algoritma

```
procedure Smallest(a1, a2, ..., an : integers, k: integer)
    if n=1 then return a1
    return (min(Smallest(a1, a2, ..., an), k-1)
    end
```

ÖR: Negatif olmayan tamsayılar için n^5-n ifadesinin 5 ile bölünüp bölünmediğini matematiksel tümevarım ilkesini kullanarak gösteriniz.

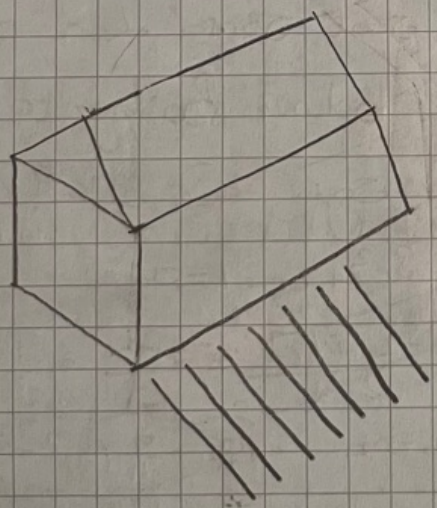
Temel basamak: P(0) Doğrudur çünkü 0^5-0, 5'e bölünebilir.

Tümevarım basamağı: P(k): k^5-k 5 ile bölünebilir mi?

$(5k)^5-5k$ $\quad 5^5 \cdot k^5-5k$

$5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot k^5-5k$ bölünebilir.

ÖR:



6 araçlık otopark. 5 farklı araç kaç şekilde park edebilir?

$6! = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 = 720$

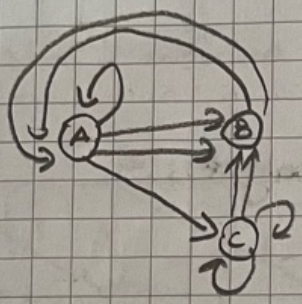
ÖR: BİLGİSAYAR kelimesindeki harflerden 2 ünlü ve 2 ünsüz harf kullanılarak 4 harfli kaç kelime oluşturulabilir?

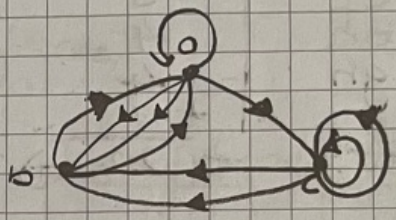
$C(4,2) * C(6,2) * 4! = 2160$

ÖR:

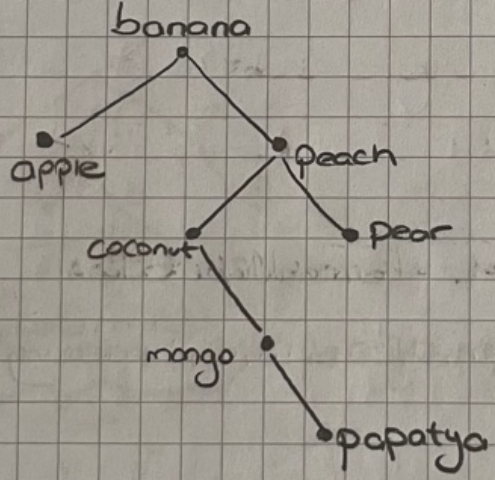
	a	b	c
a	1	2	1
b	2	0	0
c	0	2	2

Komşuluk matrisine karşılık gelen yönlü graf?

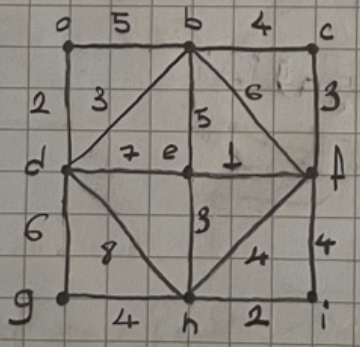




ÖR: Alfabetik sıralamaya göre "banana, peach, apple, pear, coconut, mango, papatya" kelimelerini ikili arama ağacında gösteriniz



ÖR:



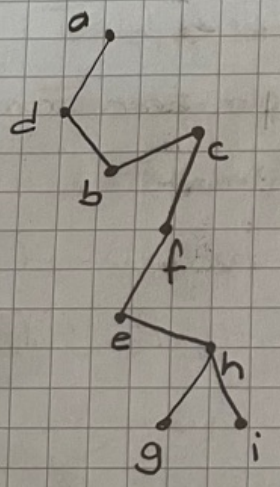
Minimum spanning tree?

Prim

Tercih	Kenar	Ağırlık
1	ef	1
2	eh	3
3	hi	2
4	fc	3
5	gh	4
6	cb	4
7	bd	3
8	da	2
		+
		22 → toplam maliyet

Kruskal

Tercih	Kenar	Ağırlık
1	ef	1
2	hi	2
3	ad	2
4	eh	3
5	bd	3
6	cf	5
7	bc	4
8	gh	4
		+
		22 → Toplam maliyet



ÖR: Binary search algoritmasını vererek karmaşıklığını bulunuz.

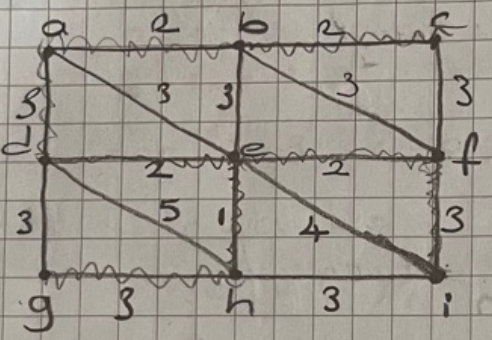
```
procedure binarySearch(x:integer, a1, a2, ..., an: increasing integers)
  i:=1
  j:=n
  while i<j
    m:=⌊(i+j)/2⌋
    if x > am then i:=m+1
    else j:=m
  if x:=ai then location:=i
  else location:=0
  return location
```

O(logn)

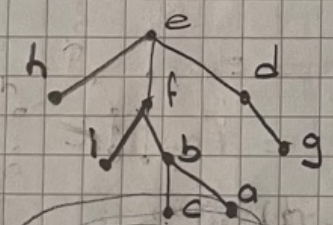
ÖR: Bir bölümde en az iki profesörün aynı ayda doğmuş olabilmesi için en az kaç profesör olmalıdır? Güvercin yuvası prensibiyle açıklayınız.

13

ÖR: Kruskal minimum spanning tree?



Tercih	Kenar	Ağırlık
1	eh	1
2	ef	2
3	ed	2
4	ab	2
5	bc	2
6	bf	3
7	fi	3
8	dg	+ 3
		18 → Maliyet



ÖR: (obeb(n,2))! fonksiyonunu hesaplayan recursive bir algoritma?

a^

```
if n==1
    return 1
else
    return n*f(n-1)
```

Üsttaki soru Prim:

Tercih	Kenar	Ağırlık
1	eh	1
2	ef	2
3	ed	2
4	fi	3
5	da	3
6	ab	2
7	bc	2
8	hg	3

ÖR: $a_n = 2a_{n-1} + 3a_{n-2}$ yineleme bağıntısının genel çözümünü elde ederek $a_0 = 1$ ve $a_1 = 2$ başlangıç durumları için bu genel çözümdeki katsayıları elde ediniz.

$n=2$ için

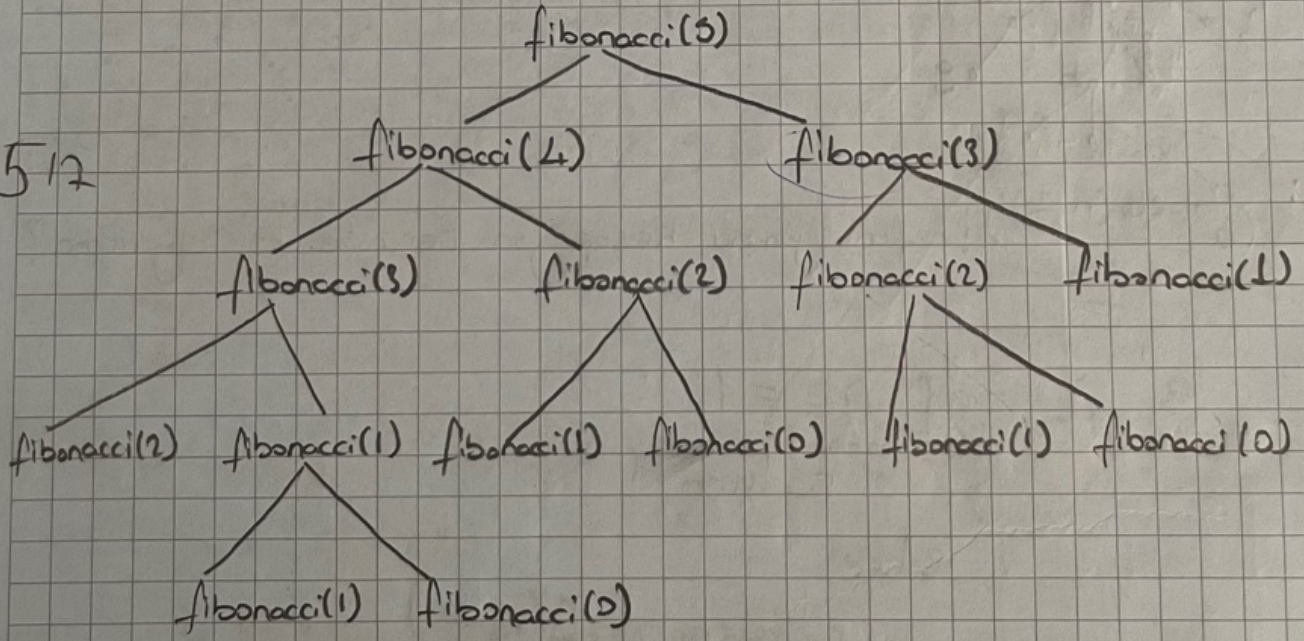
$a_2 = \underbrace{2 \cdot a_1}_{4} + \underbrace{3a_0}_{3} = 7$

$a_3 = \underbrace{2 \cdot a_2}_{14} + \underbrace{3a_1}_{6} = 20$

2. dereceden homojen

Recursive Fibonacci

```
procedure fibonacci(n: integer)
    if n ≤ 1 then return n
    else return fibonacci(n-1) + fibonacci(n-2)
```



5/7

$O(n)$ olmaz.

dynamic programming

n=32

ÖR:

```
i := 1
t := 0
while i ≤ n
    t := t + i
    i := 2i
```

i
1
2
4
16
32

O(log n)

a	b	c
1	1	2
3	5	8

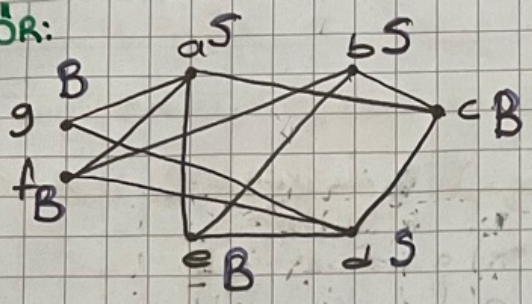
```
a = 1;
b = 1;
for ( i=3; i<=10; i++){
    c = a+b;
    sout (c);
    a = b;
    b = c;
}
```

O(n)

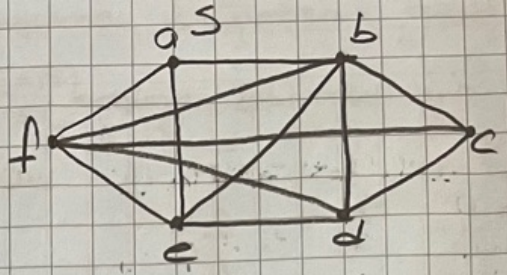
$$T(n) = T(n-1) + T(n-2)$$
$$= T(n-2) + T(n-2)$$
$$= 2\,T(n-2)$$
$$= 2^k$$

2^n

Ör:



G



H (bipartite)

Şekilde gösterilen G ve H grafları iki-kümeli midir?

G grafının birbirine komşu olmayacak düğümlerine sıyan ve boyası olmak üzere iki renk atanabileceğinden dolayı iki kümelidir.

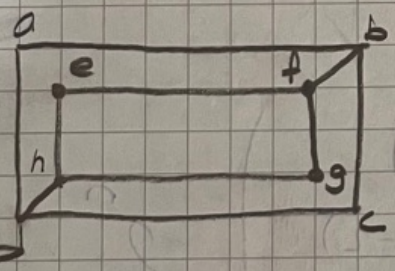
{a, b, d} {c, e, f, g}

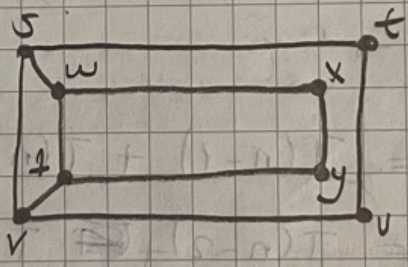
S B

Bipartite

H grafının birbirine komşu olmayacak düğümlerine atanacak renk sayısı 2'den fazla olacağı için bipartite değildir.

Ör:

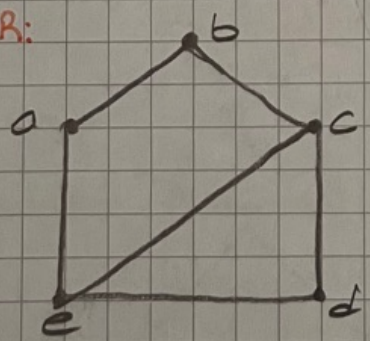


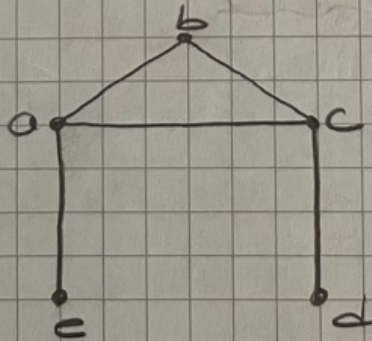


Eş yapılı değiller

(izomorfizm)

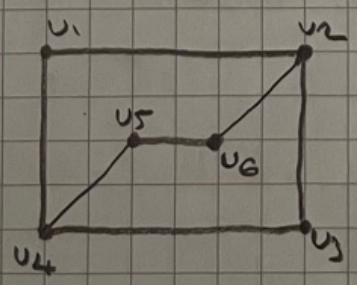
Ör:



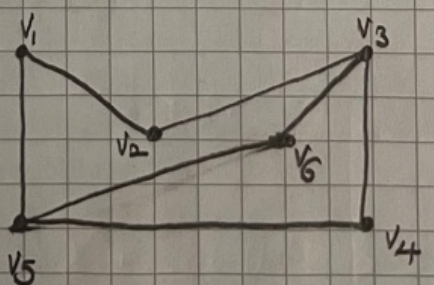


Eş yapılı değiller

Ör:



G



H

Komşuluk matrisi çizilir.

$$A_G = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{array} \begin{array}{c} \begin{array}{cccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$A_H = \begin{array}{c} \\ v_6 \\ v_5 \\ v_4 \\ v_3 \\ v_2 \\ v_1 \end{array} \begin{array}{c} \begin{array}{cccccc} v_6 & v_5 & v_4 & v_3 & v_2 & v_1 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Graflar Eş yapılıdır

İki grafın izomorfizm olup olmadığını belirlemek için

Düğüm sayıları

Ayrıt "

Düğüm dereceleri ve komşu düğüm dereceleri

Yollar ve devreler.

İleri Sayma Teknikleri

Doğrusal Özyinelemeli

Tanım: Katsayılı k. dereceden doğrusal homojen özyineleme ilişkisi

$$a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2} \dots c_k \cdot a_{n-k}$$

$c_1, c_2, \dots c_k$ reel sayılar $c_k \neq 0$ formundadır.

$P_n = 1.11 \times P_{n-1}$ → I. Dereceden doğrusal homojen

$a_n = a_{n-1} + a_{n-2}^2$ → Doğrusal değil (karesi var)

$H_n = 2H_{n-1} + 1$ → Homojen değil (1'den dolayı)

$B_n = n \cdot B_n - 1$ → Sabit katsayılı değil (n)

Doğrusal Homojen Özyineli İlişkilerin Sabit Katsayılar ile Çözümü

Teorem 1

c_1 ve c_2 gerçel sayılar;

$r^2 - c_1 r - c_2 r = 0$ eşitliğinin iki ayrı kökleri r_1 ve r_2 olsun.

Bu durumda ancak ve ancak α_1 ve α_2 sabitler olmak üzere

$n = 0, 1, 2, \dots$ için $a_n = \alpha_1 \cdot r_1^n + \alpha_1 \cdot r_2^n$ eşitliğini sağladığı durumda

a_n dizisi

$a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2}$ özyineli ilişkinin çözümüdür.

Ör: $a_0 = 2,\ a_1 = 7$

$a_n = a_{n-1} + 2a_{n-2}$ bu denklemin çözümü?

$r^2 = r + 2$

$r^2 - r - 2 = 0$ $\boxed{r_0 = 2}$ $\boxed{r_1 = -1}$

$a_n = \alpha_1 2^n + \alpha_2 (-1)^n$

$n = 0$ için $\rightarrow a_0 = \alpha_1 \cdot 2^0 + \alpha_2 (-1)^0 \rightarrow 2 = \alpha_1 + \alpha_2$

$n = 1$ için $\rightarrow a_1 = \alpha_1 \cdot 2^1 + \alpha_2 (-1)^1 \rightarrow 7 = 2\alpha_1 - \alpha_2$

$9 = 3\alpha_1$ $\boxed{\alpha_1 = 3}$ $\boxed{\alpha_2 = -1}$

$\boxed{a_n = 3 \cdot 2^n - (-1)^n}$

Teorem 2

c_1 ve c_2 gerçel sayılar $c_2 \neq 0$ olsun

$r^2 - c_1 r - c_2 = 0$ eşitliğinin iki tek kökü r_0 olsun.

$a_n = c_1 \cdot a_{n-1} + c_2 \cdot a_{n-2}$ özyinelemeli ilişkinin α_1 ve α_2'ler sabit olmak üzere;

$a_n = \alpha_1 \cdot r_0^n + \alpha_2 \cdot n \cdot r_0^n$ dir

ÖR: $a_0 = 1$, $a_1 = 6$

$a_n = 6a_{n-1} - 9a_{n-2}$ özyineli ilişkisinin çözümü?

$r^2 = 6r - 9$

$r^2 - 6r + 9 = 0$ $\quad r_0 = 3$

$a_n = \alpha_1 \cdot 3^n + \alpha_2 \cdot n \cdot 3^n$

$n = 0$ için $\rightarrow a_0 = \alpha_1 \cdot 3^0 + \alpha_2 \cdot 0 \cdot 3^0 \rightarrow 1 = \alpha_1 \cdot 3^0 \rightarrow \boxed{\alpha_1 = 1}$

$n = 1$ için $\rightarrow a_1 = \alpha_1 \cdot 3^1 + \alpha_2 \cdot 1 \cdot 3^1 \rightarrow 6 = 3\alpha_1 + 3\alpha_2 \rightarrow \boxed{\alpha_2 = 1}$

$\boxed{a_n = 3^n + n3^n}$

Teorem 3

$c_1, c_2, \ldots, c_k$ gerçel sayılar olsun.

$r^k - c_1 r^{k-1} - c_2 \cdot r^{k-2} - \ldots - c_k = 0$ k aynk $r_1, r_2, \ldots, r_k$ kökler olsun.

Bu durumda $\alpha_1, \alpha_2, \ldots, \alpha_k$ sabitleri ve $n = 0, 1, 2 \ldots$ için

$a_n = \alpha_1 \cdot r_1^n + \alpha_2 \cdot r_2^n + \ldots + \alpha_k \cdot r_k^n$ olur ve bu $r^k - c_1 r^k - \ldots - c_k = 0$ özyinelemenin çözümüdür.

ÖR: $a_0 = 2, a_1 = 5, a_2 = 15$

$a_n = 6a_{n-1} - 11a_{n-2} + 6a_{n-3}$ denkleminin bulunuz.

$r^3 - 6r^2 + 11r - 6 = 0$

$(r-1)(r-2)(r-3) = 0$ $\boxed{r_1 = 1}$ $\boxed{r_2 = 2}$ $\boxed{r_3 = 3}$

$a_n = \alpha_1 \cdot 1^n + \alpha_2 2^n + \alpha_3 \cdot 3^n$

$n = 0$ için $\rightarrow 2 = \alpha_1 + \alpha_2 + \alpha_3 \rightarrow \boxed{\alpha_1 = 1}$

$n = 1$ için $\rightarrow 5 = \alpha_1 + 2\alpha_2 + 3\alpha_3 \rightarrow \boxed{\alpha_2 = -1}$

$n = 2$ için $\rightarrow 15 = \alpha_1 + 4\alpha_2 + 9\alpha_3 \rightarrow \boxed{\alpha_3 = 2}$

$\boxed{a_n = 1 - 2^n + 2 \cdot 3^n}$

ALGORİTMA ANALİZİ

Introduction to Algorithms MIT Press

A bilgisayarı
Insertion Sort
1 milyon eleman
1 milyar komut/sn
En iyi programcı makine dilinde $2n^2$

$$\frac{2\cdot(10^6)^2 \text{ komut sayısı}}{10^9 \text{ komut/s}} = 2000 \text{sn}$$

B bilgisayarı
Merge sort
1 milyon eleman
10 milyon komut/sn
Ortalama bir programcı yüksek seviyeli dilde $50.n\log n$

$$\frac{50\cdot 10^6\cdot \log_2 10^6}{10\cdot 10^6} \cong 100 \text{sn}$$

merge sort sürekli ikiye bölerek gittiği için $\log_2$ olur.

1.000.000

$2^x = 10^6$

$x \cong 20$

ÖRNEK: Aynen kor. $2n^2$ işlemci hızı 2 gHz bir bilgisayarda 600.000 veri kaç dk'da işlenir?

$$\frac{2\cdot(6\cdot 10^5)^2}{2\cdot 10^9} = \frac{36\cdot 10^{10}}{10^9} = 36\cdot 10 = 360 \text{ sn} \qquad \frac{360}{60} = 6 \text{ dk}$$

2.000.000.000

ÖRNEK: $3n^3$ 3 Ghz 3 dk'da kaç veri işler?

$$\frac{3n^3}{3\cdot 10^9} = 3.60$$

$n^3 = 180.10^9$ $\quad n = \sqrt[3]{180.10^9}$ $\quad 5{,}5\times 10^3$ veri

Insertion Sort

		Cost	Times
1	for $j \leftarrow 2$ to length[A]	c_1	n
2	do key $\leftarrow A[j]$	c_2	$n-1$
3	▸ insert $A[j]$ to $A[1\ j+1]$	$c_3=0$	$n-1$
4	$j \leftarrow j-1$	c_4	$n-1$
5	while $i>0$ and $A[i]>key$	c_5	$\sum_{j=2}^{n} t_j$ $(n-1).n=2$
6	do $A[i+1] \leftarrow A[i]$	c_6	$\sum(t_j-1)$ $(n-1).(n-1) \rightarrow \frac{n}{2}-1$
7	$i \leftarrow i-1$	c_7	$\sum(t_j-1)$ $(n-1).(n-1) \rightarrow \frac{n}{2}-1$
8	$A[i+1] \leftarrow key$	c_8	$n-1$

En kötü durum analizi (worst case) → Big O()

En iyi " " (Best case) → Ω()

Ortalama " " (Average case) → Θ()

$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_4 \cdot (n-1) + c_5 \cdot (n-1) \cdot n + c_6 \cdot (n-1) \cdot (n-1)$
$+ c_7 \cdot (n-1) \cdot (n-1) + c_8 (n-1)$

$T(n) = a \cdot n^2 + bn + c \rightarrow O(n^2) \Rightarrow$ En kötü

$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_4 \cdot (n-1) + c_5 \cdot (n-1) + c_6 \cdot (n-1) + c_7 (n-1)$
$+ c_8 \cdot (n-1)$

$T(n) = a \cdot n + b \rightarrow \Omega(n) \rightarrow$ En iyi

$T(n) = c_1 \cdot n + c_2 (n-1) + c_4 (n-1) + c_5 (n-1) \cdot \frac{n}{2} + c_6 (n-1) \cdot (\frac{n}{2} - 1)$
$\cdot c_7 (n-1) \cdot (\frac{n}{2} - 1) + c_8 (n-1)$

$T(n) = a \cdot n^2 + bn + c \Rightarrow \Theta(n^2)$

Algoritma Tasarım Stratejileri

İteratif

Rekürsif

Böl yönet (Divide and conquer

Greedy

Dinamik Programlama

Kaba kuvvet (Brute force)

Branch and Bound

Heuristic

Backtracking

Decrase and Conquer

Transform and Conquer

Seçim Kriterleri

Problemin özellikleri

Problemin boyutu

Problem için kullanılabilir kaynaklar

Recursive Algoritmalar: Doğrudan veya dolaylı olarak kendini çağıran metotlardır.

- Problemleri daha basit alt problemlere bölerek çözme yöntemidir.
- Alt problemler de kendi içlerinde başka alt problemlere bölünür.
- Alt problemler çözülecek kadar küçülünce bölme işlemi durur.

Backtracking Algoritması: Amaca ulaşıncaya kadar doğru seçimleri değerlendirir.

- Bir labirentte sağ ve sol 2 seçerek varsa, sol taraf seçildikten sonra çıkışa ulaşılamıyorsa geri dönülüp sağ taraf seçilir.

Böl ve Yönet:

- Problemi küçük parçalara böl
- Her bir parçayı bağımsız şekilde çöz.
- Parçaları birleştirerek ana problemin çözümüne ulaş.

Dinamik Programlama: Böl ve yönet yöntemine benzer olarak alt problemlerin çözümlerini birleştirerek çözüme gitme mantığına sahip olup alt problem tekrarı varsa bunlardan biri tanesi çözülür ve bu çözüm diğer tekrarlarda kullanılır.

Greedy Yaklaşımı: Temel amaç, en iyi sonucu elde etmek için en iyi ara adım çözümlerini seçmektir.

Kaba Kuvvet Algoritması: En ilkel şekilde her yolu dener. En düşük performanslı algoritmalardan birisidir.

Sezgisel (Heuristic) Algoritmalar: Uygulanan yöntemin doğruluğunun ispatlanması gerekmez. Tek istenen karmaşık bir problemi daha basit hale getirmesi veya algoritmanın tatmin edici bir sonuç bulabilmesidir.

Branch and Bound: Optimizasyon problemini adımlamak için kullanılan bir ağaç budama tekniğidir. En iyi yolu bularak en fazla kâr sağlayan algoritmayı oluşturur.

- Her bir dal (branch) farklı bir adımı göstermektedir. Amaç her dal üzerinde analiz yaparak en iyi sonucu elde etmektir. Ağaç yapısı bu algoritmanın çalışma mantığını en iyi şekilde tanımlar.

Geometrik " Queue

Sayısal " Graph

Tree

Sets and dictionary

Böl Yönet Yaklaşımı:

- Böl
- Yön
- Birleştir

$$T(n) = \begin{cases} \Theta(1) & \text{eğer } n \le c \\ a \cdot T\left(\frac{n}{b}\right) + D(n) + C(n) & \text{diğer} \end{cases}$$

Yönet Böl Birleştir

```
MERGE(A, p, q, r)
    n1 = q - p + 1
    n2 = r - q
    L[1..n1+1] ve R[1..n2+1] yeni diziler olsun.
    for i = 1 to n1
```

Problem Tipleri

- Arama problemleri
- Sıralama "
- Dizi işleme "
- Graf "
- Kombinasyon "
- Geometrik "
- Sayısal "

Kullanılan Veri Yapıları

Listeler
- Diziler
- Bağlı Listeler
- Stringler

Stack

Queue

Graph

Tree

Sets and dictionary

Böl Yönet Yaklaşımı

- Böl
- Yön
- Birleştir

$$T(n) = \begin{cases} \Theta(1) & \text{eğer } n \le c \\ a \cdot T\left(\frac{n}{b}\right) + D(n) + C(n) & \text{diğer} \end{cases}$$

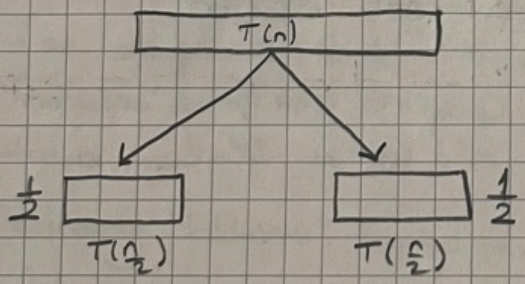
$a \cdot T\left(\frac{n}{b}\right)$: Yönet, $D(n)$: Böl, $C(n)$: Birleştir

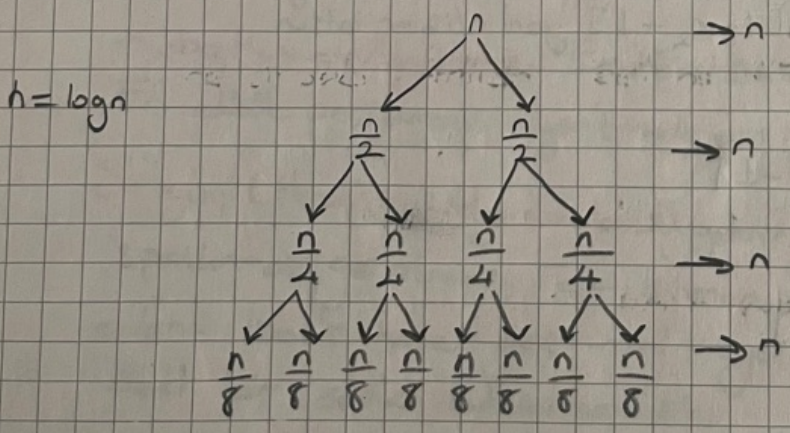
```
MERGE(A, p, q, r)
1.  n1 = q - p + 1
2.  n2 = r - q
3.  L[1..n1+1] ve R[1..n2+1] yeni diziler olsun.
4.  for i = 1 to n1
5.      L[i] = A[p+i-1]
6.  for j = 1 to n2
7.      R[j] = A[q+j]
8.  L[n1+1] = ∞
9.  R[n2+1] = ∞
10. i = 1
11. j = 1
```

```
12.  for k=p to r
13.      if L[i] ≤ R[j]
14.          A[k]=L[i]
15.          i=i+1
16.      else A[k]=R[j]
17.          j=j+1
```

```
MERGESORT(A, p, r)
1.  if p<r                          } Bol
2.      q=⌊(p+r)/2⌋.                }
3.      MERGESORT(A, p, q)          } Yönet
4.      MERGESORT(A, q+1, r)        }
5.      MERGE(A, p, q, r)           } Birleştir
```

$T(n) = 2T(\frac{n}{2}) + \Theta(1) + \Theta(n)$





$h = \log n$

$T(n) = n \cdot \log_2 n$

NOT: Merge Sort'ta en iyi durum, en kötü durum, ortalama durum yoktur. Her zaman aynı kodlar çalışacağı için her durumda nlogn olur.

Asimptotik Notasyonlar

Algoritmaların Analizi

Algoritma, bir hesaplama yapmak veya bir problemi çözmek için sonlu kesin talimatlar kümesidir.

Algoritmaları analiz etmenin amacı, onları çalışma zamanları ve diğer faktörler (bellek gereksinimi, programcının gösterdiği efor vb.) açısından karşılaştırmaktır.

Çalışma zamanı analizi yaparak problemin boyutu arttığında çalışma süresinin nasıl artacağı belirlenir.

Analiz Çeşitleri

En Kötü Durum

Çalışma zamanında bir üst sınır sağlar

Girişler ne olursa olsun algoritmanın daha uzun süre çalışmayacağının kesin bir garantisidir

En İyi Durum

Çalışma zamanında bir alt sınır sağlar

Giriş değeri, algoritmanın en hızlı çalıştığı değerlerdir

Ortalama Durum

Çalışma zamanıyla ilgili bir çıkarım sağlar

Girişin rastgele olduğunu varsayar

Asimptotik Analiz

İki algoritmayı çalışma zamanları $f(n)$ ve $g(n)$ açısından karşılaştırmak için, fonksiyonun ne kadar hızlı büyüdüğünü karakterize eden bir dile ihtiyaç duyarız.

Giriş boyutu n olan bir fonksiyonun çalışma süresi $f(n)$ ile ifade ediliyor.

Çalışma sürelerine karşılık gelen farklı fonksiyonları karşılaştır.

Böyle bir analiz makine zamanından, programlama stilinden vb. bağımsızdır.

Asimptotik notasyon fonksiyonların sınırdaki veya sınırlar olmadan davranışlarını tanımlamanın bir yoludur.

Notasyonlar, domainleri $N = \{0, 1, 2 \ldots\}$ doğal sayılar kümesi olan fonksiyonlar olarak tanımlanırlar.

Bu tür gösterimler, en kötü çalışma süresi fonksiyonu $T(n)$'in tanımlanması için uygundur.

Ayrıca bu gerçek (reel) sayılar domainine de genişletilebilir.

Örnek:

x, $x-1$ ile asimptotiktir.

$$\lim_{x \to \infty} f(x) = k$$

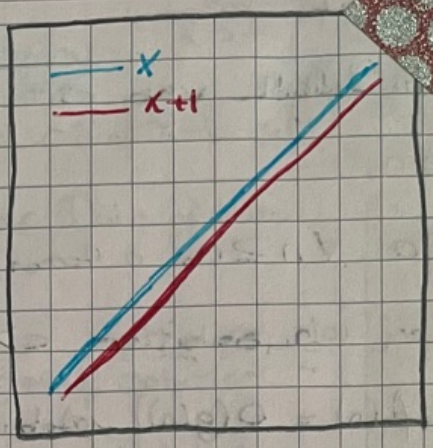
Kabaca şu şekilde tercüme edilebilir:

x, ∞'a yakınken ve $f(x)$ de k'ya yakınken; x, ∞'a yaklaşır, $f(x)$ de k'ya yaklaşır.

Analiz yapılırken sıklıkla iki limit kullanılır:

$$\lim_{x \to \infty} f(x) = \lim_{x \to \infty} f(x) \perp / x = 0$$

$$\lim_{x \to \infty} f(x) = \lim_{x \to \infty} f(x) \, c / x = \infty \qquad c > 0 \text{ için}$$



Asimptotik Büyüme Oranı

Big oh (O) notasyonu

Omega (Ω) notasyonu

Teta (Θ) notasyonu

Little oh (o) notasyonu

w notasyonu

O notasyonu: asimptotik "küçüktür" $f(n) \leq cg(n)$

Ω notasyonu: asimptotik "büyüktür" $f(n) \geq cg(n)$

Θ notasyonu: asimptotik "eşitlik" $c_1 g(n) \leq f(n) \leq c_2 g(n)$

Big O Notasyonu

Asymtotik upper bound

$$\lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| < \infty$$

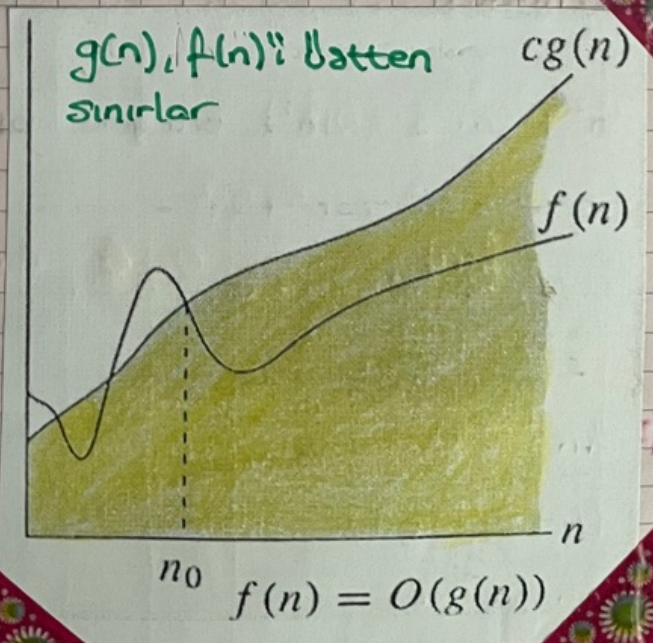
$f(n)$ için muhtemel asimptotik üst sınır. En kötü durumdur. (Daha kötü olmaz)

Fonk. üzerinde sabit bir çarpanı olan üst sınır vermek için kullanırız.

n problem boyutu,

$f(n) \in O(g(n))$,

$O(g(n)) = \{f(n)$: ∃ pozitif sabitler c, n_0, öyle ki $0 \leq f(n) \leq cg(n)$, $\forall n \geq n_0\}$



Bunun anlamı; $n \geq n_0$ 'ın bütün değerleri için $f(n)$ $g(n)$'in üstünde veya altındadır.

$O(g(n))$, $cg(n)$ değerinden küçük veya eşit olan $f(n)$ fonksiyonlarının bir kümesidir, $\forall n \geq n_0$

Eğer $f(n) \leq cg(n)$, $c>0$, $\forall n \geq n_0$ o zaman $f(n) \in O(g(n))$

Yani tüm $n \geq n_0$ değerleri için sabitler $c>0$, $n_0>0$ ise;

$0 \leq f(n) \leq cg(n)$ durumunda $f(n) = O(g(n))$ yazabiliriz.

$f(n)$ ve $g(n)$ verilen iki çalışma zamanı fonksiyonudur.

$f(n) \leq cg(n)$ ve $n \geq n_0$ koşullarını sağlayan c ve n_0 değerleri varsa $f(n)$ zaman karmaşıklığı $O(g(n))$'dir.

n sayısı yeteri kadar büyük olduğunda $f(n)$, $g(n)$ ile aynı büyüklüktedir.

ÖR: $2n^2 = O(n^3)$ için c ve n_0 değerlerini bulunuz.

$0 \leq f(n) \leq cg(n)$, $0 \leq 2n^2 \leq cn^3$

$c=2$ için $n_0=1$ şartı sağlar.

ÖR: $\frac{1}{2}n^2 + 3n$ için üst sınırın $O(n^2)$ olduğunu gösteriniz.

$c=1$ için;

$\frac{1}{2}n^2 + 3n \leq n^2 \Rightarrow 3n \leq \frac{1}{2}n^2 \Rightarrow 6 \leq n$

$n_0 = 6$ $\quad 6n \leq n^2 \quad 6 \leq n$ $\quad 0 \leq \frac{1}{2}n^2 + 3n \leq cn^2$

ÖR: $3n^2 + 2n + 5 = O(n^2)$ olduğunu gösteriniz.

$10n^2 = 3n^2 + 2n^2 + 5n^2$

$10n^2 \geq 3n^2 + 2n + 5$, $n \geq 1$

$c=10$, $n_0=1$

ÖR: $4n = O(n)$ eşitliği doğru mu?

$f(n) \leq c \cdot g(n)$ olmalı

$4n \leq c \cdot 1$

$c=4$ $\quad n \geq 1$ için sağlar.

Asymptotic lower bound

$\lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| > 0$

Ω Notasyonu

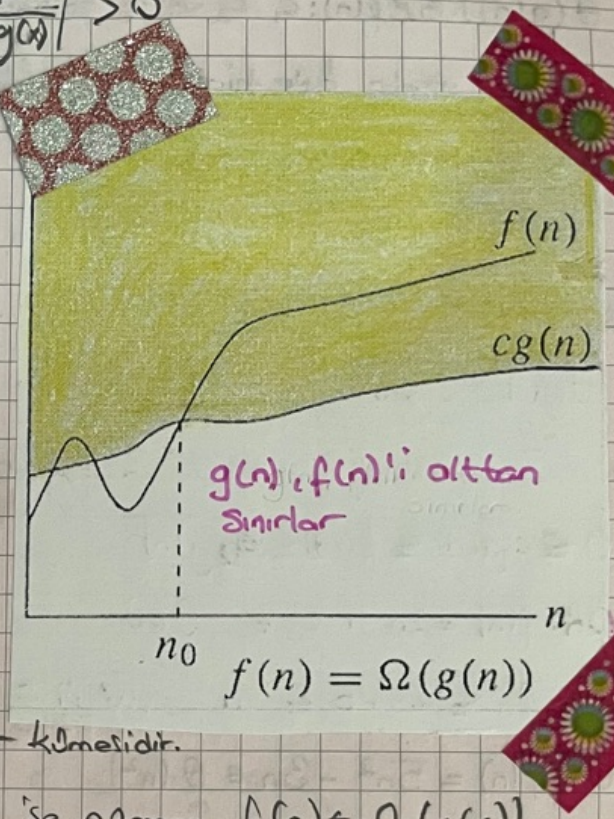
$f(n)$ için muhtemel asimptotik sıkı alt sınır. En iyi durumdur. (Daha iyi olamaz)

$f(n) \in \Omega(g(n))$,

$\Omega(g(n)) = \{f(n)$: ∃ pozitif sabitler $c > 0, n_0$ öyle ki $0 \leq cg(n) \leq f(n)$, $\forall n \geq n_0$

$n \geq n_0$ 'ın bütün değerleri için $f(n)$ $g(n)$'in yukarısında veya üstündedir.

$\Omega(g(n))$, $cg(n)$ değerinden büyük veya eşit olan $f(n)$ fonksiyonlarının bir kümesidir.



Eğer $cg(n) \leq f(n)$, $c > 0$ ve $\forall n \geq n_0$ ise o zaman $f(n) \in \Omega(g(n))$

$\Omega(g(n)) = \{f(n)$: tüm $n \geq n_0$ değerlerinde $c > 0$, $n_0 > 0$ ise,

$\{0 \leq cg(n) \leq f(n)\}$

Her durumda $f(n) \geq cg(n)$ ve $n \geq n_0$ koşullarını sağlayan n_0 değerleri bulunabiliyorsa $f(n) = \Omega(g(n))$'dir.

ÖR: $2n + 5 \in \Omega(n)$ olduğunu gösteriniz.

$n_0 \geq 0$, $2n + 5 \geq n \Rightarrow c = 1$ $n_0 = 0$

ÖR: $5n^2 - 3n \geq n^2 = \Omega(n^2)$ olduğunu gösteriniz.

$5n^2 - 3n \geq n^2 \Rightarrow c = 1$ $n_0 = 0$

$-3n \geq -4n^2$

$3n \leq 4n^2$

ÖR: $5n^2 = \Omega(n)$

$0 \leq cn \leq 5n^2 \Rightarrow cn \leq 5n^2 \Rightarrow c = 1$ $n_0 = 1$

ÖR: $100n + 5 \neq \Omega(n^2)$

$0 \leq cn^2 \leq 100n + 5$

$100n + 5 \leq 100n + 5n$ $(\forall n \geq 1) = 105n$

$cn^2 \leq 105n \Rightarrow n(cn - 105) \leq 0$

Θ Notasyonu

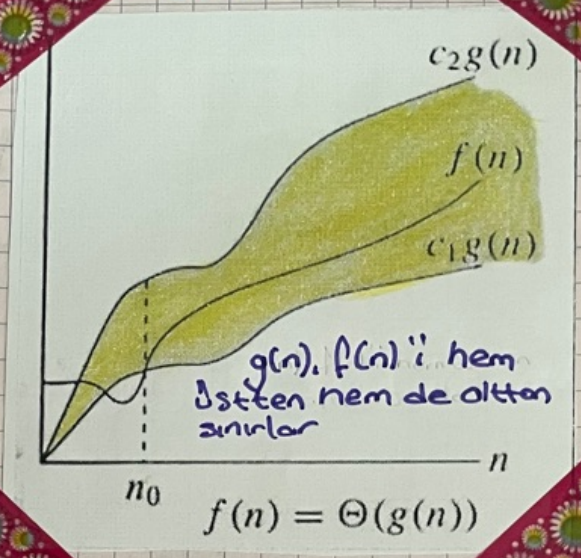
Asymtotically tight bound

$\Theta(g(n)) = \{f(n)$: c_1, c_2 ve n_0 pozitif sabitleri vardır öyle ki $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$, tüm $n \ge n_0$ için$\}$

$f(n) \in \Theta(g(n))$ için $f(n)$ asimptotik sıkı sınırdır.

$\Theta(g(n))$; $c_1 g(n)$ ve $c_2 g(n)$ arasındaki tüm $f(n)$ fonksiyonlarının bir kümesidir.

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)\}$



ÖR: $f(n) = 2n + 5 \in \Theta(n)$

$2n \le 2n+5 \le 3n \quad \forall n \ge 5$ için

$0 < \lim_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| < \infty$

ÖR: $f(n) = 5n^2 - 3n \in \Theta(n^2)$

$4n^2 \le 5n^2 - 3n \le 5n^2 \quad \forall n \ge 4$ için

I. ÖR:

$f(n)$	O	Ω	Θ	$g(n)$
$(n+3)\log n$	✓		✓	$n^3 + \log n$
$n^2 \log n$		✓	✓	$5n^2 + n\log n$
$n \log n$	✓		✓	$n\sqrt{n}$
$n^2 + 10\log n$	✓	✓	✓	$n\sqrt{n}$
$\log_3 n^3$	✓	✓	✓	$2\log_2 n$

II. ÖR:

$f(n)$	O	Θ	Ω	$g(n)$
$n^3 + n\log n$	✓	✓		$n^3 + n^2 \log n$
$\log\sqrt{n}$		✓	✓	$\sqrt{\log n}$
$n \log_3 n$		✓	✓	$n \log_4 n$
2^n				$2^{n/2}$
$\log(2^n)$	✓	✓		$\log(3^n)$

Eg: $f(n) = g(n)$

$(n+5)\log n = O(n^2+\log n)$ $\Rightarrow f(n)$'i üstten sınırlıyor

$(n+5)\log n \neq \Theta(n^2+\log n)$ $\Rightarrow$ hem üstten hem alttan sınırlaması gerek sadece üstten sınırlıyor

$(n+5)\log n \neq \Omega(n^2+\log n)$ $\Rightarrow f(n)$'i alttan sınırlamıyor

O, Ω ve Θ Notasyonları Arasındaki İlişkiler

Eğer $g(n) = \Omega(f(n))$ ise $f(n) = O(g(n))$

Eğer $f(n) = O(g(n))$ ve $f(n) = \Omega(g(n))$ ise $f(n) = \Theta(g(n))$

$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

n'in büyük olduğu ve sabitlerin elendiği durumlarda;

$O(g(n))$ için $f(n)$ ile eşit veya büyük, üstten sınır $f(n)$ ile aynı hızda veya hızlı büyür.

$\Omega(g(n))$ için $f(n)$ ile eşit veya küçük, alttan sınır $f(n)$ ile aynı hızda veya yavaş büyür.

$\Theta(g(n))$ için $f(n)$ ile eşit, alttan ve üstten sınırın büyüme oranları eşittir.

Recursive Algoritmalar

$$\left.\begin{array}{l} a(n) = 5n+4 \\ a(1) = 9 \end{array}\right\} a(n+1) = a(n)+5$$

$$\left.\begin{array}{l} a(n) = 2n! \\ a(1) = 2 \end{array}\right\} a(n+1) = a(n).a(n+1)$$

$$\left.\begin{array}{l} a(n) = \sum_{i=1}^{n} n \\ a(1) = 1 \end{array}\right\} a(n+1) = a(n)+n+1$$

Küçük o Notasyonu — Asymtotically negligible

$0 < f(n) < c \cdot g(n)$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

$2n = o(n^2)$, $2n^2 \neq o(n^2)$

w Notasyonu — Asymtotically dominant

$f(n) = w(g(n))$

$0 \leq c \cdot g(n) < f(n)$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

$\frac{n^2}{2} = w(n)$, $\frac{n^2}{2} \neq w(n^2)$

$2n \leq 2n+5 \leq 3n \quad \forall n \geq 5$ için

R: $f(n) = 5n^2 - 3n \in \Theta(n^2)$

$4n^2 \leq 5n^2 - 3n \leq 5n^2 \quad \forall n \geq 4$ için

:

$f(n)$	Θ	Ω	Θ	$g(n)$
$(n+3)\log n$	✓		✓	$n^3+\log n$
$n^2 \log n$		✓	✓	$5n^2+n\log n$
$n \log n$	✓		✓	$n\sqrt{n}$
$n^2+10\log n$	✓	✓	✓	$n\sqrt{n}$
$\log_3 n^3$	✓	✓	✓	$2\log_2 n$

Ör: $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

$f(n)$ $g(n)$

$0 \le c_1 \cdot g(n) \le f(n) \le c_2 g(n)$

$c_1 \cdot n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$

$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$

(her tarafı n^2'ye böl)

problemin boyutu

$n = 6$ olursa 0 olur. (+ olmalı)

bu durumda $n = 7$ olur.

$n = 7$ için $c_2 = \frac{1}{2}$

$c_1 = \frac{1}{14}$

$\lim_{n\to\infty} \frac{\frac{1}{2}n^2 - 3n}{n^2}$

Ör: $6n^3 = \Theta(n^2)$ doğru mu?

$c_1 g(n) \le f(n) \le c_2 g(n)$

$c_1 n^2 \le 6n^3 \le c_2 n^2$

$c_1 \le 6n \le c_2$

Sabit bir c_1 ve c_2 değerleri bulunamadığı için eşitlik sağlanmaz.

Θ notasyonu $6n^3$'ü ifade etmek için kullanılamaz.

Ör: $4n + 3 = \Omega(n)$

$0 \le cg(n) \le f(n)$

$0 \le cn \le 4n + 3$

$0 \le c \le 4 + \frac{3}{n}$

$c = 5 \quad n \ge 3$

Ör: $4n = O(n)$

$f(n) \le cg(n)$

$4n \le c \cdot 1$

$c = 4 \quad n \ge 1$ için sağlar

Ör: $4n + 3 = O(n)$

$f(n) \le cg(n)$

$4n + 3 \le cn$

$c = 7$ için $n \ge 1$ olmalı

yo da

$c = 5$ için $n \ge 3$ olmalı

Ör: $4n + 3 = \Theta(n)$

$c_1 \cdot n \le 4n + 3 \le c_2 n$

$c_1 \le 4 + \frac{3}{n} \le c_2$

$c_1 = 1 \quad c_2 = 5 \quad n \ge 3$ için sağlanır.

Ör: $g(n) = n^2$ 1 dk'da 1GHz hızla kaç veri işlenir?

1	10^9
60	n^2

$n^2 = 6 \cdot 10^8$

$n = \sqrt{6 \cdot 10^8}$

$n = \sqrt{6} \cdot 10^5$

$n = 2.4 \times 10^5$

$n \simeq 24 \cdot 10^4$

Ör: $g(n) = 2n^2$ 2GHz hızla 6000000 veriyi kaç dk'da işler?

1sn	2×10^9
x	$2(6 \cdot 10^5)^2$

$2 \cdot 10^9 x = 2 \cdot 36 \cdot 10^{10}$

$x = 360$ sn

$\Rightarrow$ 6 dk

Divide and Conquer

Tekrarlı Bağıntıların Çözümü

Master Teoremi Yöntemi

Yerine Koyma Yöntemi

Recursive Tree Yöntemi

1- Yerine Koyma Yöntemi

Bir sınır tahmini yapıp bu tahminin doğruluğunu göstermek için matematiksel tümevarım kullanacağız.

ÖR: $T(n) = T(n/2) + c \quad n \geq 2$ ve $T(1) = 1$

$T(2) = 1+c$, $T(4) = 1+2c$, $T(8) = 1+3c$ ...

$T(2^k) = 1+kc$ burada $n = 2^k$, $T(n) = 1 + c\log n$ $T(n) = T(\frac{n}{2}) + c$ için doğru

$T(n+1) = 1 + (k+1)c = (1+kc) + c$, $n = 2^{k+1}$

Proof:

$T(n+1) = T(2^{k+1}) = T(2^{k+1}/2) + c$

$T(n+1) = T(2^k \cdot 2/2 + c)$

$= T(2^k) + c = 1 + kc + c$ tümevarım adımı ispatlanmış olur.

$n = 2^k$ için $T(n) = 1 + c\log n$ ve $T(n) \in O(\log n)$'dir.

2- Master Teoremi

$T(n) = a\,T(\frac{n}{b}) + cn^i \quad a \geq 1, \quad b \geq 2$

$4\,T(\frac{n}{2}) + n^3 \log n$

$a = 4$

$b = 2$

$i = 3$

$$T(n) = \begin{cases} \theta(n^i \log_b n), & a = b^i, \; i = \log_b a \quad (1) \\ \theta(n^{\log_b a}), & a > b^i, \; i < \log_b a \quad (2) \\ \theta(n^i), & a < b^i, \; i > \log_b a \quad (3) \end{cases}$$

ÖR: $T(n) = T(n/2) + n$

$a = 1 \quad b = 2 \quad i = 1$

$a \leq b^i \quad 1 \leq 2$

Durum 3 $\Rightarrow \theta(n^i)$

$= \underline{\theta(n)}$

ÖR: $T(n) = 8\,T(n/2) + cn^2$

$a = 8 \quad b = 2 \quad i = 2$

$8 > 2^2 \quad a > b^i$

Durum 2 $\Rightarrow \theta(n^{\log_b a})$

$= \theta(n^{\log_2 8}) \Rightarrow \underline{\theta(n^3)}$

$3n/7 = n \cdot \frac{3}{7}$ → $(7/10)^1$

$= \frac{n}{\frac{7}{3}}$ → b

OR: $T(n) = T(3n/7) + 1$

$a=1 \quad b=7/3 \quad i=0$

$1 = (\frac{7}{3})^0 \Rightarrow$ Durum 1

$\Theta(n^i \log_b n) = \Theta(n^0 \log_{7/3} n)$

$\cong \Theta(\log n)$

OR: $T(n) = 9T(n/3) + n$

$a=9 \quad b=3 \quad i=1$

$9 > 3^1 \Rightarrow a > b^i$

Durum 2 $\Rightarrow \Theta(n^{\log_3^9}) = \underline{\Theta(n^2)}$

OR: $T(n) = 2T(n/2) + n^i \log^p n$

$a=2 \quad b=2 \quad i=1 \quad p=1$

$2 = 2^1 \Rightarrow$ Durum 1

$\Rightarrow \Theta(n \log_2 n)$

OR: $T(n) = 3T(\frac{n}{4}) + n \log n$

$a=3 \quad b=4$

$\log n^n = \log 10^{n^n} = 1$

$i=0 \quad 3 > 4^0 \Rightarrow a > b^i \geq 0.8$

Durum 2 $\Rightarrow \Theta(n^{\log_b a}) = \Theta(n^{\log_4^3})$

$\cong \Theta(n)$

OR: $T(n) = T(2n/3) + 1$

$a=1 \quad b=\frac{3}{2} \quad i=0$

$a = b^i$ Durum 1

$\Rightarrow \Theta(n^0 \log_{\frac{3}{2}} n) = \Theta(\log_{\frac{3}{2}} n)$

OR:

```
program(n) {                    → T(n)
  if n==1 return 1              → Θ(1)
  else x = program(n-1)         → T(n-1)
  return x                      → Θ(1)
}
```

$T(n) = T(n-1) + \Theta(1)$

OR:

$$n! = \begin{cases} 1 & \text{if } n=1 \\ n.(n-1)! & \text{if } n>1 \end{cases}$$

$$T(n) = \begin{cases} b & n=1 \text{ base case} \\ c + T(n-1) & \text{recursive step} \end{cases}$$

```
FACT(n)
if n=1 then
  return
else
  return n * FACT(n-1)
endif
```

OR:

GIPTA

OR:

```
f(n):
  if n=1 then
    return 2
  else
    return 3*f(n/2)+f(n/2)+5;
  endif
```

$T(1)=O(1)=\alpha$ (sbt)

$$T(n)=\begin{cases} T(1)=\alpha & \text{base case} \\ 2T(\frac{n}{2})+b & \text{recursive step} \end{cases}$$

OR:

```
Program(n): T(n)
  if n=1 return 1      O(1)
  a=n+1                O(1)
  b=n-1                O(1)
  for i=0 to n         ] O(n)
    c=a+b+i
  if (...) return Program(n/2)+Program(n/2) => T(n/2)+T(n/2) = 2T(n/2)
  else if (...) return Program(n-1) => T(n-1)
  else return Program(n/3)+Program(n/3)+Program(n/3) 3T(n/3)
```

$$T(n)=\begin{cases} 2T(\frac{n}{2})+n & \to \Theta(n\log n) \\ T(n-1)+n & \to \Theta(n^2) \\ 3T(\frac{n}{3})+n & \to \Theta(n\log_3 n) \end{cases} \Rightarrow \Theta(n^2) \text{ olur.}$$

OR:

```
Deneme(n): T(n)
  if n=1 return 0      O(1)
  else return Deneme(n/2)+1   T(n/2)
```

$T(n)=T(n/2)+O(1)$

$T(n-1)=T(\frac{n-1}{2})+O(1)$

3

Ör:

elma(n): T(n)

if n=1 return 1 O(1)

else return elma(n-1) + elma(n-1) 2T(n-1)

$T(n) = 2T(n-1) + O(1)$ $T(n-1) = 2T(n-2) + O(1)$

$T(n-2) = 2T(n-3) + O(1)$ $T(n-n) = 2^n T(n-n) + O(1) \cdot n$

$T(n) = 2^n T(0) + O(1) n = 2^n + n = \underline{\underline{O(2^n)}}$

Ör:

Power(x, n):

if n=0 return 1

if n=1 return x

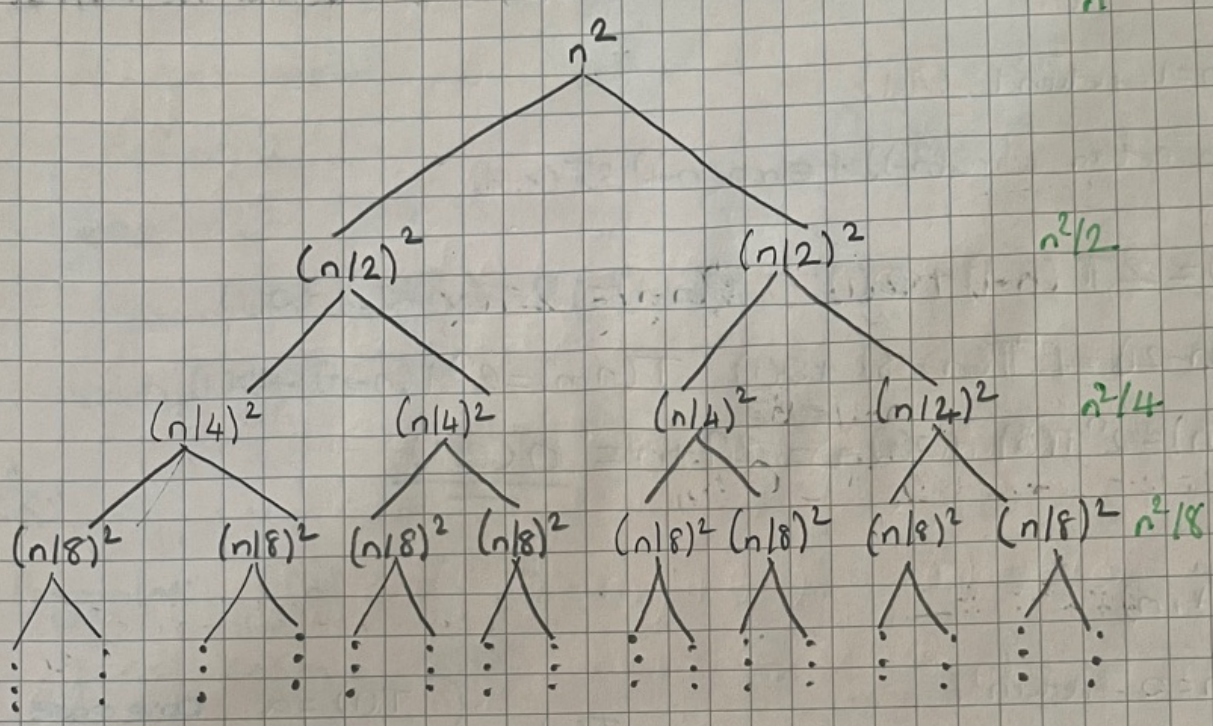
else return x * Power(x, n-1)

$$T(n) = \begin{cases} T(1) = a & \text{base case} \\ T(n-1) + b & \text{recursive step} \end{cases}$$

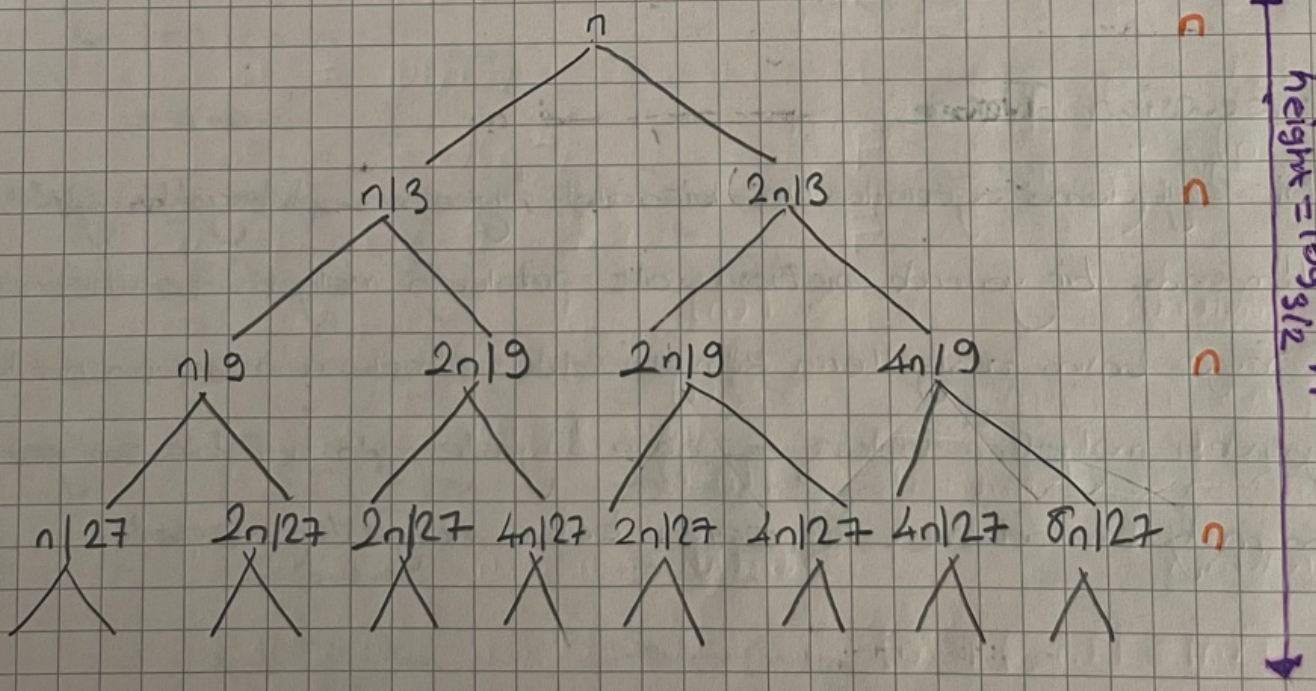
Recursion Trees

Bir özyineleme ağacında her bir dal özyineleme fonksiyonları yürütmeleri kümesinde bir yerlerde bir basit alt problemin maliyetini temsil etmektedir. Seviye başına maliyetlerin kümesini elde etmek için ağacın her bir seviyesindeki maliyetleri toplarız ve sonra özyinelemenin her bir seviyesinin toplam maliyetini belirlemek için seviye başına maliyetleri toplarız.

Omega $T(n) = 2T(n/2) + n^2$



$T(n) = T(n/3) + T(2n/3) + n$



ÖR: Aşağıdaki algoritmanın tekrarlı bağıntısını ve karmaşıklığını açıklayarak veriniz.

```
program(n)
    if n=1 return 1
    else    return program(n-1)+program(n-1)
```

$T(n) = T(n-1) + T(n-1) + \Theta(1)$

$\rightarrow 1+2+2^2+\dots+2^{n-1} = \Theta(n)$

ÖR: $2^{n+1} = O(n^2)$ ifadesinin doğru olup olmadığını matematiksel tanımdan yola çıkarak gösteriniz

$2^{n+1} = 2 \cdot 2^n$ olduğu için ifade doğrudur.

ÖR: $2^{2n} = O(2^n)$ ifadesi doğru mu?

$2^{2n} = 2^{n^2}$ olduğu için ifade doğru değildir.

ÖR: Zaman karmaşıklığı $g(n) = 3n^3$ olan bir algoritma ile işlemci hızı 500 MHz olan bilgisayarda 1000 veri kaç dk'da işlenebilir?

$1 / (500 \times 10^6) \cdot 3(1000)^3 = t \rightarrow t = 6\,sn \quad 6/60 = 0.1\,dk$

ÖR: Algoritmik karmaşıklığını açıklayarak veriniz.

```
for(i=1 ; i<=n ; i++){
    for(j=1; j<=n ; j+=i)
            x=x+1;
}
```

$n + n/2 + n/3 + \dots + 1 = n(1 + 1/2 + 1/3 + \dots + 1/n) = \Theta(n \log n)$

ÖR: Aşağıdaki algoritmanın tekrarlı bağıntısını ve karmaşıklığını açıklayınız.

```
MIN(A, ℓ, r)
    if ℓ=r return A[ℓ]
    else t1 ← MIN(A, ℓ, [(ℓ+r)/2])
         t2 ← MIN(A, [(ℓ+r)/2]+1, ℓ)
         if t1 ≤ t2 return t1
         else return t2
```

Master teoremi

$T(n) = T\left(\frac{n+1}{2}\right) + T\left(\frac{n+1}{2} - 1\right) + \Theta(1)$ $\quad \underline{\underline{T(n) = \Theta(n)}}$

ÖR:

(1) $\sum_{i=1}^{n} \sqrt{i} = O(n^{\frac{3}{2}})$ T

(2) $n^n = O(2^n)$ F

(3) $n! = O(n^n)$ T

(4) $n^2 \log n = \Theta(n^2)$ F

(5) $n^2 / \log n = \Theta(n^2)$ F

(6) $33n^3 + 4n^2 = \Omega(n^2)$ T

Big O için $\Rightarrow 0 \leq f(n) \leq c \cdot g(n)$

(2): $0 \leq n^n \leq 2^n$ $n=3$ için sağlanmaz.

(3): $0 \leq n! \leq n^n$ n'in bütün değerleri için sağlar

Θ için $\Rightarrow 0 \leq c \cdot g(n) \leq f(n)$

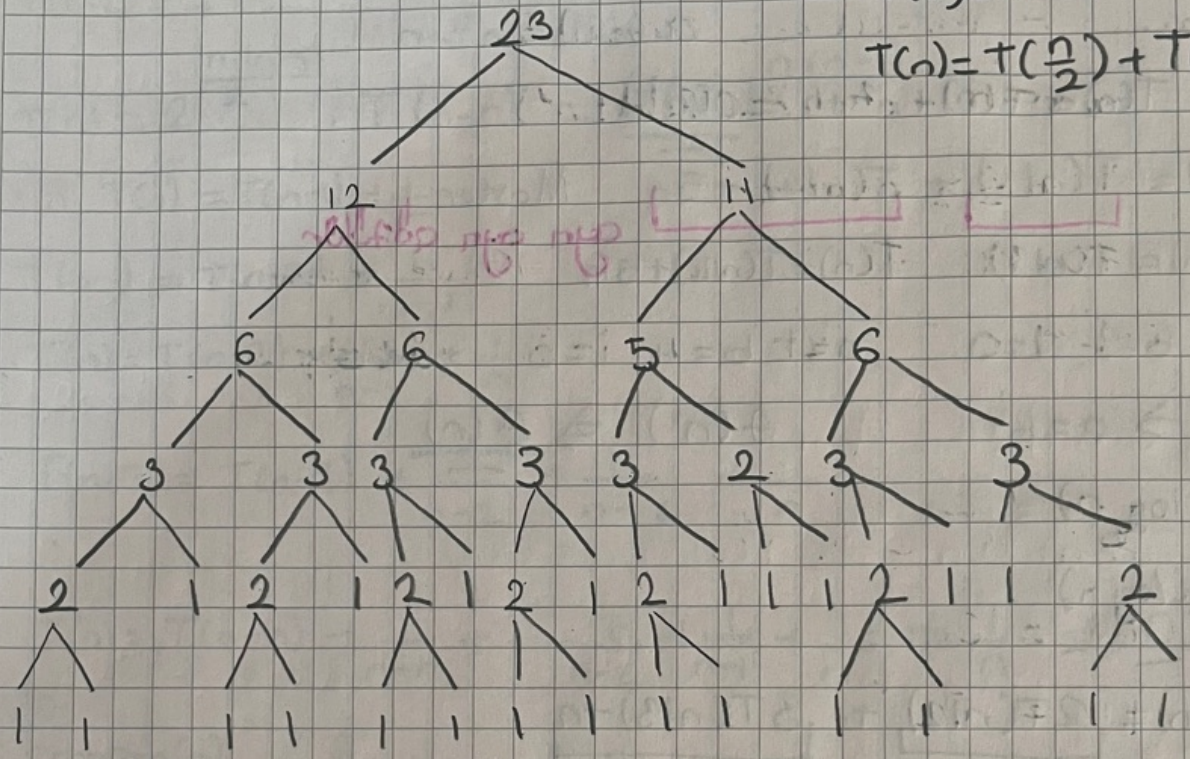
(4): $0 \leq n^2 \leq n^2 \log n$

(5): $0 \leq n^2 \leq n^2 / \log n$ $n > 10$ için sağlanmaz

(6): $0 \leq n^2 \leq 33n^3 + 4n^2$

ÖR: n elemanlı bir dizi'yi sıralamak için merge sort kullanılacaktır. n=23 için algoritmanın recursion ağacını çiziniz. Recursion ağacındaki seviye sayısı nedir? En kötü durumda her bir seviyedeki karşılaştırma sayısı nedir?

$$T(1)=0$$
$$T(n)=T(\tfrac{n}{2})+T(\tfrac{n}{2})+n-1$$



Seviye sayısı → 6

Karşılaştırma sayısı için 1 olmayan bütün elemanları say

1. seviye → 12 karşılaştırma 2. Seviye → 21 3. Seviye → 19

4. seviye → 15 5. seviye → 7 6. Seviye → 0 Toplam: 84

* n verilmemiş olsaydı, merge sort'ta sürekli 2'ye bölünerek gidildiği için ağacın seviyesi $\log_2 n$ olur. Karmaşıklık $n \log n$ olur

ÖR: $n^{1/\lg n}$, $(\sqrt{2})^{\lg n}$, $(n+1)!$, $2^{\sqrt{2\lg n}}$, $(\lg n)^{\lg n}$ fonksiyonları ile verilen karmaşıklıklara sahip olan algoritmaların karmaşıklıklarını büyükten küçüğe

$$(n+1)! > (\lg n)^{\lg n} > (\sqrt{2})^{\lg n} > 2^{\sqrt{2\lg n}} > n^{1/\lg n}$$

* Bu tür soruları çözmenin en kolay yolu n'e değer vermektir

Ör: $T(n) = T(n-2) + \log n$ Yerine koyma

Gidelim: $T(n-2) = T(n-4) + \log(n-2)$

$T(n) = T(n-4) + \log(n-2) + \log n$

$T(n) = T(n-n) + \log(n-4) + \log(n-2) + \log n$

$n.(n+1) = n^2+n$

$T(n) = T(0) + n^2 + n = \underline{O(n^2)}$

Ör: $T(n) = T(n/3) + T(n/4) + 5n$ Master ayrı ayrı çözülür

$T(n) = T(n/3)$ $T(n) = T(n/4) + 5n$

$a=1$ $b=3$ $i=0$ $a=1$ $b=4$ $i=0$ $a<b^i$

$1 = 3^0 \Rightarrow a = b^i$ $\Theta(n^i) \Rightarrow \underline{\Theta(n)}$

$\Theta(n^i \log_b n)$

$\Rightarrow \underline{\Theta(\log_3 n)}$

Ör: $2T(n) = 2T(n/2) + 3T(n/3) + n$

$T(n) = T(n/2)$ $T(n) = \frac{3}{2}T(n/3) + \frac{n}{2}$

$a=1$ $b=2$ $i=0$ $a=\frac{3}{2}$ $b=3$ $i=1$

$a = b^i \Rightarrow \underline{\Theta(\log_2 n)}$ $a < b^i \Rightarrow \underline{\Theta(n)}$

Ör: $T(n) = 4T(n-1) - 3T(n-2) + 1$

$T(n-1) = 4T(n-2) - 3T(n-3) + 1$

$T(n) = 4(4T(n-2) - 3T(n-3) + 1) - 3T(n-2) + 1$

$T(n) = 16T(n-2) - 12T(n-3) + 1 - 3T(n-2) + 1$

$T(n) = 13T(n-2) - 12T(n-3) + 4 + 1$

$\vdots$

$T(n) = 40T(n-3) - 39T(n-4) + 13 + 4 + 1$

$T(n) = 40T(n-n) - 39T(n-n) + 3^0 + 3^1 + 3^2 + \dots 3^n$

$T(n) = 3^n = \underline{O(3^n)}$

OR: $T(n) = 2^n T(n-1)$

$T(n-1) = 2^{n-1} T(n-2)$

$T(n) = 2^n \cdot 2^{n-1} T(n-2)$

$T(n-2) = 2^n 2^{n-1} 2^{n-2} T(n-3)$

$\vdots$

$T(n) = 2^{\frac{n\cdot(n-1)}{2}} \cdot T(n-n) = \Theta(\sqrt{2}^{n\cdot(n-1)})$

OR: $T(n) = T(n-1) + \frac{1}{n}$

$T(n-1) = T(n-2) + \frac{1}{n-1}$

$T(n) = T(n-2) + \frac{1}{n-1} + \frac{1}{n}$

$T(n-2) = T(n-3) + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$

$\vdots$

$T(n) = T(n-n) + \frac{1}{n-n} + \dots + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n} \neq \frac{1}{n} \Rightarrow O(\log n)$

OR:

```
program(n)   T(n)
  if n=1 return 1   Θ(1)
  else x=program(n-1)   T(n-1)
  return x   Θ(1)
```

$T(n) = T(n-1) + 2\Theta(1)$

$T(n-1) = T(n-2) + 2\Theta(1)$

$T(n-2) = T(n-3) + 2\Theta(1)$

$\vdots$

$T(n) = T(n-n) + n\Theta(1) = T(0) + n \cdot \Theta(1) = \underline{\underline{\Theta(n)}}$

OR: $T(n) = T(n/4) + \sqrt{n}$

$a=1 \quad b=4 \quad i = 1/2 \quad 1 < 4^{1/2} \quad a < b^i$

$\Theta(n^i) = \Theta(\sqrt{n})$

OR: $T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n-1$

$T(n) = T(n-2) + n-1 + n$

$T(n-2) = T(n-3) + n-2$

$T(n-3) = T(n-4) + n-3$

$\vdots$

$T(n) = T(n-n) + n \cdot n = T(0) + n^2 \Rightarrow T(n) = \underline{\underline{O(n^2)}}$

OR: $T(n) = T(n-1) + 1$

$T(n-1) = T(n-2) + 1$

$T(n-2) = T(n-3) + 1$

$T(n-3) = T(n-4) + 1$

$\vdots$

$T(n) = T(n-n) + 1 \cdot n \Rightarrow T(0) + 1 \cdot n = O(n)$

OR:

Arama ve Sıralama Algoritmaları

Algorithm	Best Time	Average Time	Worst Time	Worst Space
Linear Search	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Binary Search	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Bucket Sort	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$
Tim Sort	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Shell Sort	$O(n)$	$O((n \log n)^2)$	$O((n \log n)^2)$	$O(1)$

Hash Search

1. Get the key k to be searched
2. Set counter j=0
3. Compute hash function h[k] = k%0SIZE
4. If the key space at hashtable[h[k]] is occupied
   Compare the element at hashtable[h[k]] with the key k
   If they are equal
   The key is found at the bucket h[k]
   Stop
   Else
   The element might be placed at the next location given by the quadatic funct.
   Increment j

Set h[k]=(k+j*j)%SIZE so that we can probe the bucket at a new slot h[k]

Repeat Step4 till j is greater than SIZE of hashtable

The key was not found in the hashtable

Stop

Algorithm	Average	Worst	Space	Other Notes
Hash Search	O(1)	O(1)	O(n)	Suitable for large data
Interpolation Search	O(log(logn))	O(n)		
Exponential Search	O(n)	O(n)	O(n)	Suitable for large data
Ternary Search	O(logn)	O(n)		Suitable for large data set

Binary Search (İkili Arama)

Yapı olarak bölbul yöntet yaklaşımının uygulamasıdır.

Dizi sıralı olmalıdır

procedure binarySearch (x: tamsayı, $a_1, a_2, \ldots, a_n$: artan tam sayılar)

i=1 {i, arama aralığının sol son noktasıdır}

j=n {j, arama aralığının sağ son noktasıdır}

while i<j

m := (i+j)/2

if $x > a_m$ then i := m+1

else j=m

if $x := a_i$ then konum := i

else konum := 0

return konum

Arama yapılacak dizinin tam ortasına bak

Aranan eleman bulunduysa sonlan

Aranan değer ortadaki elemandan küçükse sol tarafı kontrol et

" " " " büyükse " " " "

Aranan eleman bulununcaya kadar devam et

Selection Sort

Her adımda en küçük değer başa getirilir

```
void SelectionSort(int[] arr){
    int n = arr.length
    for(int i=0; i<n-1; i++){
        int minIdx = i;
        for(int j=i+1; j<n; j++){
            if(arr[j]<arr[minIdx]){
                minIdx=j; }
    }
    int tmp=arr[minIdx];
    arr[minIdx] = arr[i];
    arr[i] = tmp;
}}
```

```
for i ← 0 to n-1 do
    min ← i
    for j ← i+1 to n
        if A[j] < A[min]
            min ← j
    swap(A[j], A[min])
```

En iyi durum: $O(n^2)$

Ortalama durum: $O(n^2)$

En kötü durum: $O(n^2)$

Quick Sort

Bir divide and conquer algoritmasıdır.

- Sayı dizisinden herhangi bir eleman pivot seçilir.
- Pivottan küçük olanlar pivotun önüne, büyük olanlar arkasına gelecek şekilde elemanlar sıralanır

```
int partition(int arr[], int low, int high) {
  int pivot = arr[high];
  int i = (low-1);
  for(int j = low; j < high; j++) {
    if(arr[j] < pivot) {
      i++;
      int tmp = arr[i];
      arr[i] = arr[j];
      arr[j] = tmp; } //end if //end for
  int tmp = arr[i+1];
  arr[i+1] = arr[high];
  arr[high] = tmp;
  return i+1; }
void sort(int arr[], int low, int high) {
  if (low < high) {
    int pi = partition(arr, low, high);
    sort(arr, low, pi-1);
    sort(arr, pi+1, high);
}}
```

```
quickSort(A, p, r)
  if p < r
    q ← partition(A, p, r)
    quickSort(A, p, q)
    quickSort(A, q+1, r)
```

En iyi durum: $O(n \log n)$

Ortalama durum: $O(n \log n)$

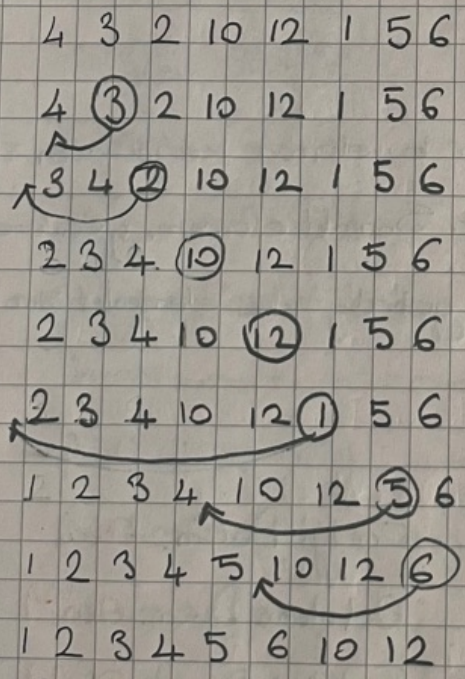
En kötü durum: $O(n^2)$

QuickSort için;

En iyi durum; seçilen pivotun solunda küçük, sağında büyük elemanların bulunması

En kötü durum; seçilen pivotun çok küçük veya çok büyük olması

Insertion Sort



En iyi durum: $O(n)$

Ortalama durum: $O(n^2)$

En kötü durum: $O(n^2)$

Insertion sort için;

En iyi durum: Distaki döngünün n, içteki döngünün 1 kez çalışmalıdır. Yani dizi sıralıdır. Kontrol amaçlı döngü çalışır...

En kötü durum; Dizinin tersten sıralı olmasıdır.

Distaki döngü → kontrol etmek için

İçteki döngü → yerdeğiştirmek için

```
void sort(int[] arr) {
  int n = arr.length;
  for(int i=1; i<n; ++i){
    int key= arr[i];
    int j = i-1;
    while( j>=0 && arr[j] >key){
      arr[j+1] = arr[j];
    } //end while
    arr[j+1] =key;
  }
}
```

Bubble Sort

Sıralanacak diziden ilk eleman seçilir. Eğer bu eleman kendisinden sonra gelen elemandan büyükse yerleri değiştirilir. Sonraki elemana geçilerek bu işlemlere devam edilir. Dizinin sonuna gelindiğinde elemanlar sıralanmış olur.

```
static int[] BubbleSort(int[] arr){
    int n = arr.length;
    for(int i=0; i<n-1 ; i++){
      for(int j=0 ; j<n-i-1 ; j++){
        if(arr[j] >arr[i+1]){
          int tmp= arr[j];
          arr[j] = arr[i+1];
          arr[i+1] = tmp;
  }}}
  return arr;
}
```

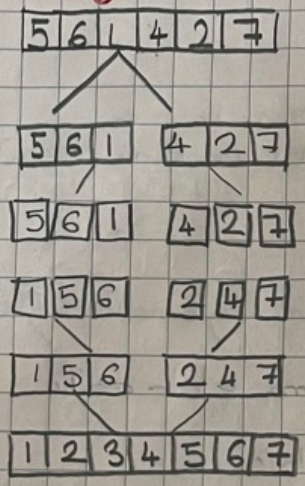
En İyi Durum: $O(n)$

Ortalama Durum: $O(n^2)$

En Kötü Durum: $O(n^2)$

```
procedure bubbleSort(A[])
  for i=1 to N
    for j=0 to N-1
      if a[j] > a[i+1] then
        swap(a[j], a[i+1])
```

Merge Sort



* Merge sort'ta en iyi, en kötü, ortalama durumlar yoktur. Her zaman aynı kodlar çalışacağı için her durumda nlogn olacaktır.

```
void merge(int[] arr, int l, int m, int r){
  int n1=m-l+1, n2=r-m, L[]=new int[n1], R[]=new int[n2];
  for(int i=0; i<n1; ++i){ L[i]=arr[l+i];}
  for(int j=0; j<n2; ++j){R[j]=arr[m+1+j];}
  int i=0, j=0, k=l;
  while(i<n1 && j<n2){ if(L[i] <= R[j]){
    arr[k]=L[i]; i++;}
  else{arr[k]=R[j]; j++;} k++;} // end of while
  while(i<n1){arr[k]=L[i]; i++; k++;}
  while(j<n2){arr[k]=R[j]; j++; k++;}
}
```

```
void sort(int[] arr, int l, int r){
    if(l<r){
        int m=(l+r)/2
        sort(arr, l, m);
        sort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

Greedy Algoritmalar

Her zaman o an için en iyi görünen seçimleri yapan ve bu mevcut alt çözüme eklegen optimizasyon problemidir.

Greedy algoritmalar her zaman optimal çözüm sağlamazlar. Fakat sağladıklarında genelde en basit ve verimli algoritma elde edilir.

En iyi çözüm her zaman doğru çözüm değildir.

Knapsack problemi → Fractional Knapsack $O(n \log n)$

Knapsack problemi → 0-1 Knapsack

```
KNAPO-1(v, w, n, W)
    for w ← 0 to W do
        c[0, w] ← 0
    for i ← 1 to n do
        c[i, 0] ← 0
    for i ← 1 to n do
        for w ← 1 to W do
            if wi < w then
                c[i, w] ← max{vi + c[i-1, w-wi], c[i-1, w]}
            else
                c[i, w] ← c[i-1, w]
    return c[n, W]
```

c is an (n+1) x (W+1) array;

c[0...n : 0...W]

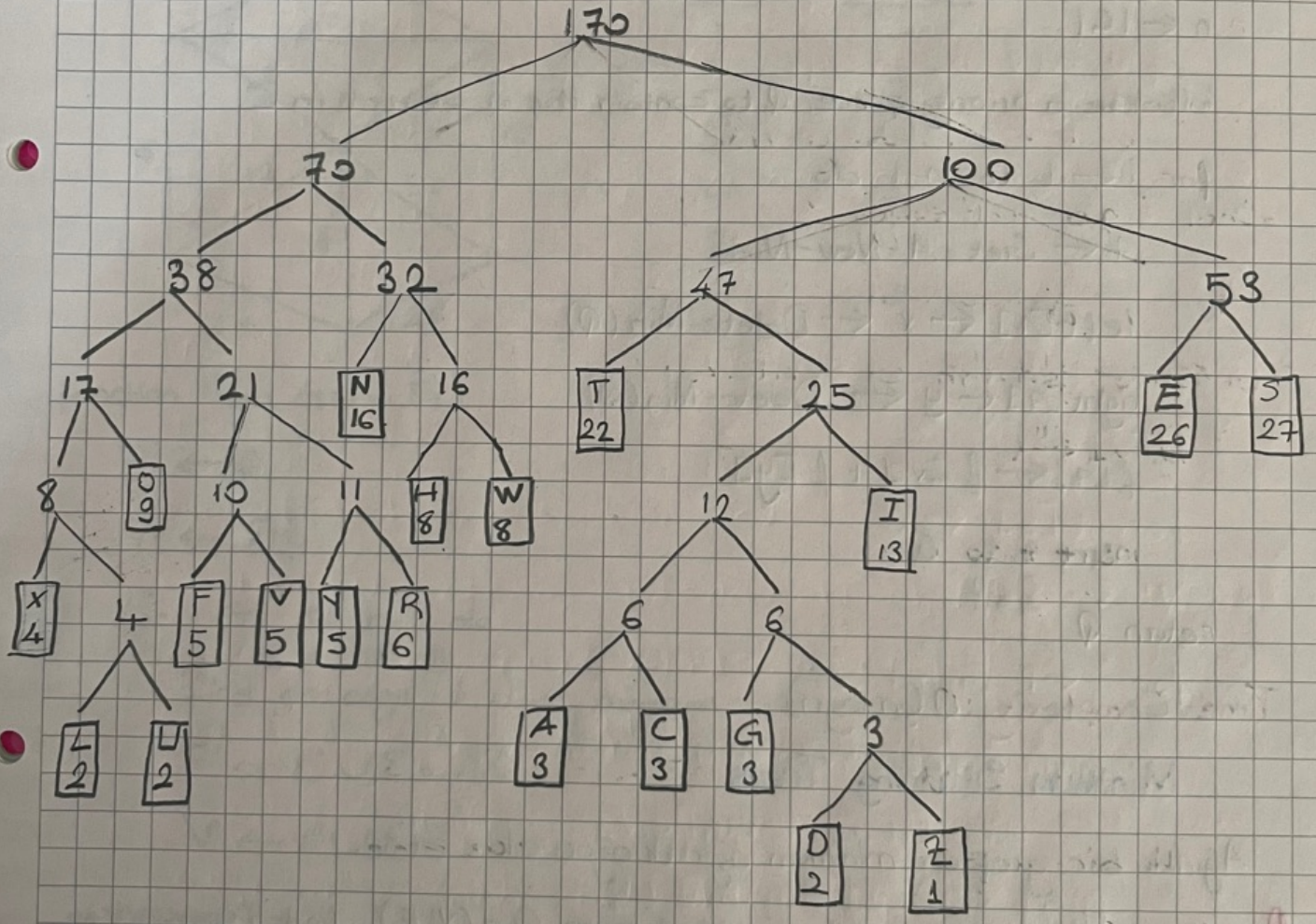
Run Time: $T(n) = \theta(n \cdot W)$

GIPTA

Huffman Code

THISSENTENCECONTAINSTHREEASTHRECSTWODSTWENTYSIXESFIVEFSTHREE
GSEIGHTHSTHIRTEENISTWOLSSIXTEENNSNINEOSTWENTYSEVENSSTWENTY
TWOTSTWOUSFIVEVSEIGHTWSFOURXSFIVEYSANDONLYONEZ

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1



100	0110	1011	111	111	110	010	100	110	010	101001	110	101001	0001	010	100	...
T	H	I	S	S	E	N	T	F	N	C	E	C	O	N	T	

Greedy Algorithm:

Regard C as a forest with |C| single-node trees

repeat

merge two trees with least frequencies

until it becomes a single tree

Implementation:

```
Huffman(C)
    n ← |C|
    initialize a priority queue, Q, to contain the n elements in C
    for i ← 1 to n-1 do
        t ← Get-A-New-Node
        left[t] ← x ← Delete-Min(Q)
        right[t] ← y ← Delete-Min(Q)
        f[t] ← f[x] + f[y]
        insert t to Q
    return Q
```

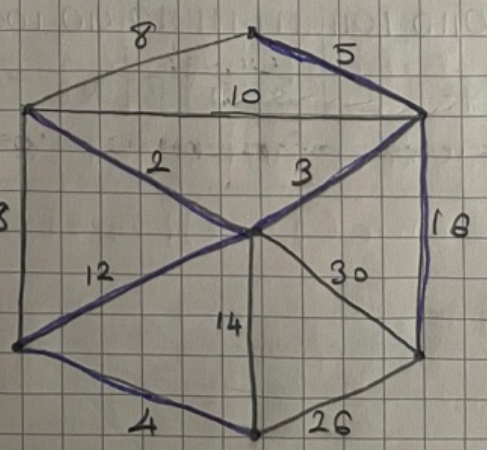
Time Complexity: $O(n \log n)$

Minimum Spanning Trees

Ağırlıklı bir grafton minimum ağırlıklı ağacı elde etme.

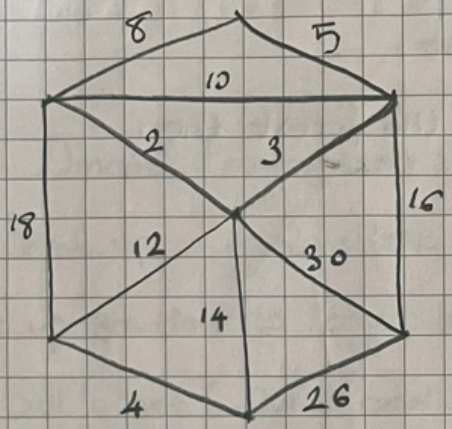
Problem: Given a connected weighted graph $G = (V, E)$, find a spanning tree of minimum cost.

Assume $V = \{1, 2, \ldots, n\}$.



Prim's Algorithm

Ağırlıklı grafta ilk olarak en düşük ağırlıklı ayrıtı seçer. Daha sonra, bu ayrıta bağlı olan ayrıtlara bakar. Bu ayrıtlar içerisinden de yine en düşük ağırlıklı ayrıtı seçerek bütün köşeleri alana kadar devam eder. Kapalı devre oluşmamalıdır. Kapalı devre oluşursa ağaç olmaz.



Sıra	Seçim
1	2
2	3
3	5
4	12
5	4
6	16

```
function Prim(G=(V, E))
    E' ← ∅
    V' ← {1}
    for i ← 1 to n-1 do
        find an edge (u,v) of minimum cost such that u ∈ V' and v ∉ V'
        E' ← E' ∪ {(u,v)}
        V' ← V' ∪ {v}
    return (T=(V', E'))
```

Complexity: $O(n^2)$

Başlangıç düğümü genelde rastgele olarak seçilir.

Kruskal's Algorithm

Önce ayrıtların ağırlıkları sıralanır. En düşük ağırlıklı ayrıttan başlanır.

Time Complexity: $O(e \log n)$

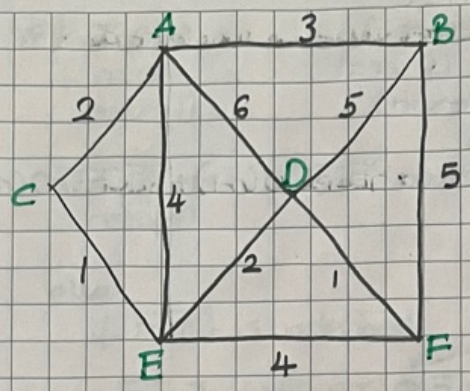
e: edges in the graph

```
Sort edges by increasing cost
T ← ∅
Let G' = (V, T), which is a forest
repeat
    (u, v) ← next edge of the sorted list
    if u and v are on different trees of the forest then
        T ← T ∪ {(u, v)}
until T has n-1 edges
```

Dijkstra's Algorithm

```
Dijkstra's Algorithm (G = (V, E), s)
    D[s] ← 0
    Parent[s] ← 0
    V' ← {s}
    for i ← 1 to n-1 do
        find an edge (u, v) such that u ∈ V', v ∉ V'
            and D[u] + length[u, v] is minimum;
        D[v] ← D[u] + length[u, v];
        Parent[v] ← u;
        V' ← V' ∪ {v};
    endfor
```

Complexity: $O(n^2)$



Step	Costs B	C	D	E	F	visited?
Init	3	2	6	4	-	C
1	3	2	6	3	-	B
2	3	2	6	3	8	E
3	3	2	5	3	7	D
4	3	2	5	3	6	F

Dinamik Programlama

Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.

Böl yönet (divide and conquer) ve dinamik programlamanın en önemli farkı;

Divide and conquer → Top-down (yukarıdan aşağı) (Merge sort)

Dynamic programming → Bottom-up (aşağıdan yukarı)

Alt problemlerin çözümü hesaplanılıp bu çözümler bir tablada tutulur.

Compute the solutions to the sub sub problems once and store the solutions in a table, so that they can be reused (repeatedly) later.

Dinamik programlamanın adımları;

1- Defining subproblems

2- Finding recurrences

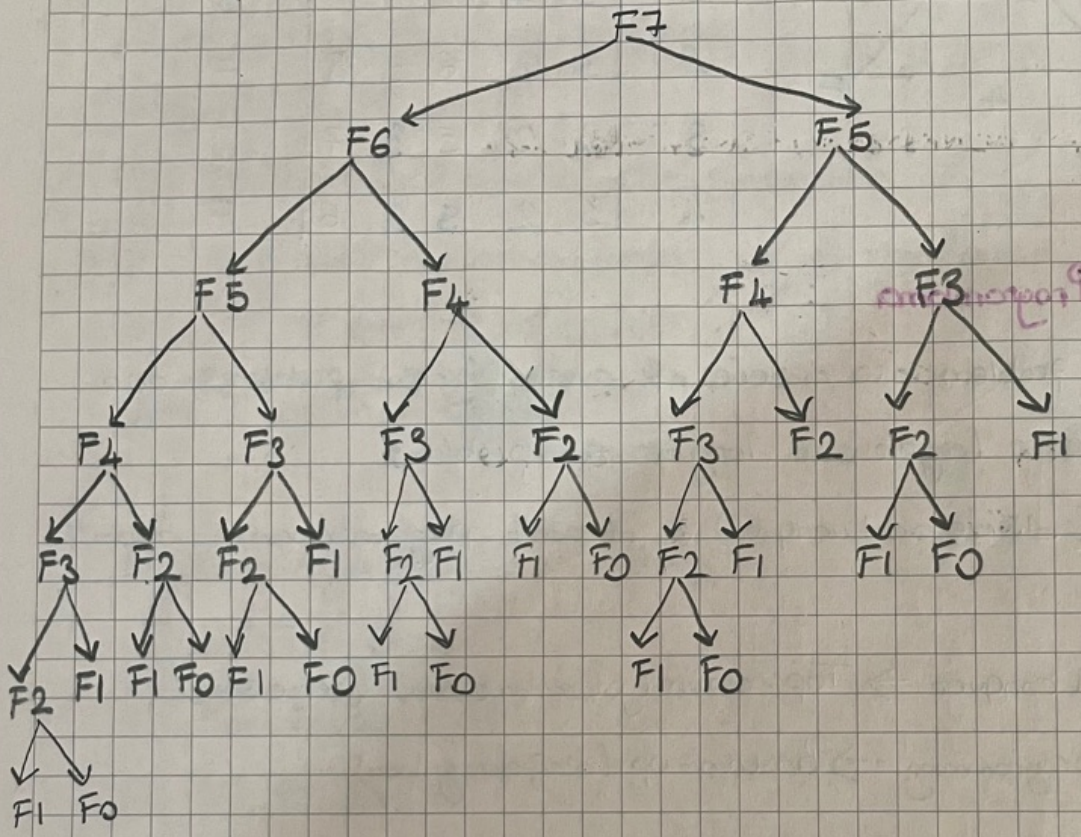
3- Solving the base cases

* Backtracking search algorithm with memoization

Fibonacci örneği:

$$F_n = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

```
RECFIBO(n):
  if n=0
    return 0
  else if n=1
    return 1
  else
    return RECFIBO(n-1)+RECFIBO(n-2)
```



Bu şekilde bir adetim 2^n karmaşıklığa sahip olacaktır. Bunun sebebi boşa hesaplamaların birden fazla kez yapılacak olmasıdır. Yukarıdaki fibonacci örneğinde 7. elemanın hesaplanabilmesi için 1. eleman 8 kez hesaplanmıştır. Eleman sayısı arttıkça exponensiyel olarak karmaşıklık artacaktır.

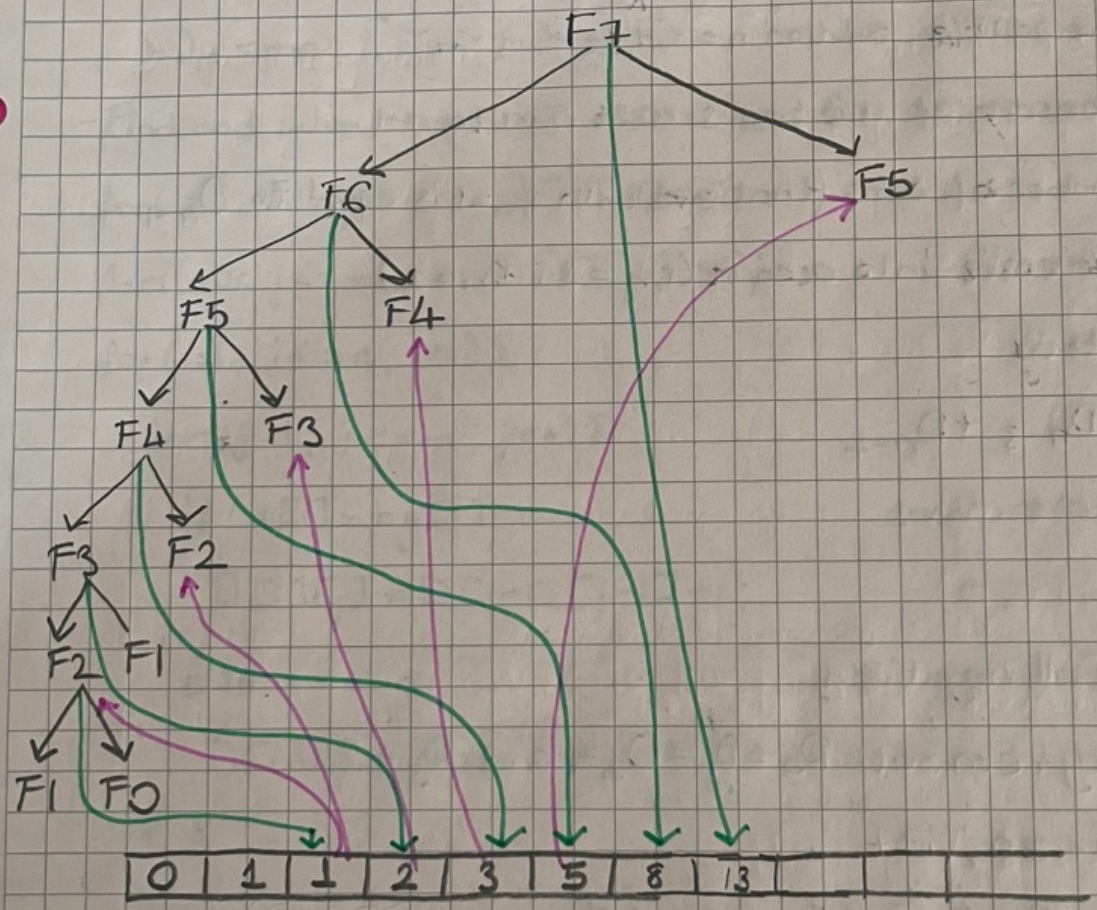
* Memoization: Memoization ensures that a method doesn't run for the same inputs more than once by keeping a record of the results for the given inputs (usually in a hashmap)

```
MEMFIBO(n):
  if n=0
    return 0

  else if n=1
    return 1

  else
    if F[n] is undefined
      F[n] ← MEMFIBO(n-1) + MEMFIBO(n-2)

    return F[n]
```




```
ITERFIBO(n):
  F[0] ← 0
  F[1] ← 1
  for i ← 2 to n
    F[i] ← F[i-1] + F[i-2]
  return F[n]
```

Bu çözümün karmaşıklığı O(n) olacaktır.

1-Dimensional Dynamic Programming

Problem: given n, find the number of different ways to write n as the sum of 1, 3, 4

Example: for n=5, the answer is 6

1+1+1+1+1 1+1+3 1+3+1 3+1+1 1+4 4+1

Define subproblems

- Let D_n be the number of ways to write n as the sum of 1, 3, 4

Find the recurrence

- Consider one possible solution $n = x_1 + x_2 + \cdots + x_m$
- If $x_m = 1$, the rest of the terms must sum to $n-1$
- Thus, the number of sums that end with $x_m = 1$ is equal to D_{n-1}
- Take other cases into account ($x_m = 3$, $x_m = 4$)

Recurrence is then

$$D_n = D_{n-1} + D_{n-3} + D_{n-4}$$

Solve the base cases

- $D_0 = 1$
- $D_n = 0$ for all negative n
- Alternatively, can set: $D_0 = D_1 = D_2 = 1$ and $D_3 = 2$

```
for(i=4; i<=n; i++
  D[i] = D[i-1] + D[i-3] + D[i-4];
```

2-Dimentional Dynamic Programming

Problem: Given two strings x and y, find the longest common subsequence (LCS) and print its length. Ayrik veya bitisik olabilir

Example:

- x: ABCBDAB
- y: BDCABC

} "BCAB" is the longest subsequence found in both sequences, so the answer is 4

Define subproblems

- Let D_{ij} be the length of the LCS of $x_{1\cdots i}$ and $y_{1\cdots j}$

Find the recurrence

- If $x_i = y_j$, they both contribute to the LCS
  - $D_{ij} = D_{i-1,j-1} + 1$
- Otherwise, either x_i or y_j does not contribute to the LCS, so one can be dropped
  - $D_{ij} = \max\{D_{i-1,j}, D_{i,j-1}\}$
- Find and solve the base cases: $D_{i0} = D_{0j} = 0$

```
for(i=0; i<=n; i++) D[i][0]=0;
for(j=0; j<=m; j++) D[0][j]=0;
for(i=1; i<=n; i++){
  for(j=1; j<=m; j++){
    if(x[i]==y[i])
      D[i][j]=D[i-1][j-1]+1;
    else
      D[i][j]=max(D[i-1][j], D[i][j-1]);
  }}
```

Graph coloring problem:

- First, we arbitrarily decide the root node r
- B_v: the optimal solution for a subtree having v as the root, where we color v black
- W_v: the optimal solution for a subtree having v as the root, where we don't color v
- Answer is $\max\{B_r, W_r\}$

Find the recurrence

- Crucial observation: once v's color is determined, subtrees can be solved independently

- If v is colored, its childern must not be colored

$$B_v = 1 + \sum_{u \in children(v)} W_u$$

- If v is not colored, its childern can have any color

$$W_v = 1 + \sum_{u \in children(v)} \max\{B_u, W_u\}$$

Base cases: leaf nodes

Knapsack

```
KnapSack(v, w, n, W) {
  for(w=0 to W) V[0,w] = 0;
  for(i=1 to n)
    for(w=0 to W)
      if(w[i] ≤ w)
        V[i,w] = max{V[i-1, w], v[i] + V[i-1, w - w[i]]};
      else
        V[i,w] = V[i-1, w];
  return V[n,W];
}
```

Time Complexity: $O(n \cdot W)$

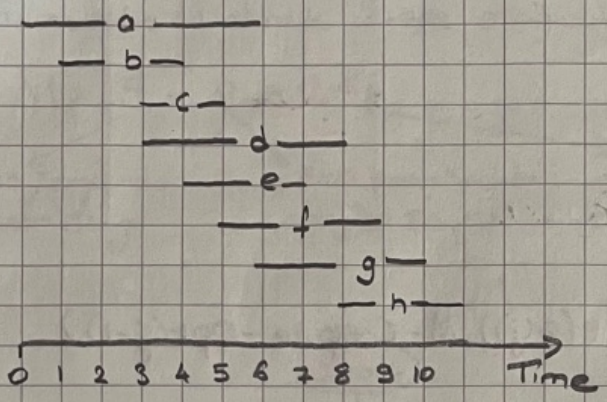
Dynamic programming saves us from having to recompute previously calculated subsolutions

Weighted Interval Scheduling Problem

Job j starts at s_j, finishes at f_j, and has weight or value v_j

Two jobs compatible if they don't overlap

Goal: find maximum weight subset of mutually compatible jobs



Brute force algorithm:

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

Sort jobs by finish times so that $f_1 \leq f_2 \leq \cdots \leq f_n$.

Compute $P(1), P(2), \ldots, P(n)$

```
Compute-Opt(j) {
  if (j=0)
    return 0
  else
    return max(vj + Compute-Opt(p(j)), Compute-Opt(j-1)) }
```

Recursive algorithm fails spectacularly because of redundant subproblems

→ exponential algorithms

Tekrarlı hesaplamalar olusacaktır.

Complexity: $O(2^n)$

Memoization: Store results of each subproblem in a cache; lookup as needed

Input: $n, s_1, \ldots, s_n, f_1, \ldots, f_n, v_1, \ldots, v_n$

Sort jobs by finish times so that $f_1 \leq f_2 \leq \ldots \leq f_n$

Compute $p(1), p(2), \ldots, p(n)$

```
for j=1 to n
  M[j]= empty      ← global array
M[0]=0
M-Compute-Opt(j){
  if (M[j] is empty)
    M[j]=max(v_j+M-Compute-Opt(p(j)),M-Compute-Opt(j-1))
  return M[j] }
```

Complexity: $O(n)$

Run M-Compute-Opt(n) $O(n)$

Run Find-Solution(n)

```
Find-Solution(j){
  if(j=0) output nothing
  else if(v_j + M[p(j)] > M[j-1]
    print j
    Find-Solution(p(j))
  else Find-Solution(j-1) }
```

Bottom-up dynamic programming Unwind recursion

```
Iterative-Compute-Opt{
  M[0]=0
  for j=1 to n
    M[j] = max(v_j + M[p(j)], M[j-1])
}
```

Cyclomatic Complexity

Graf teorisine dayalı bir karmaşıklık ölçütüdür.

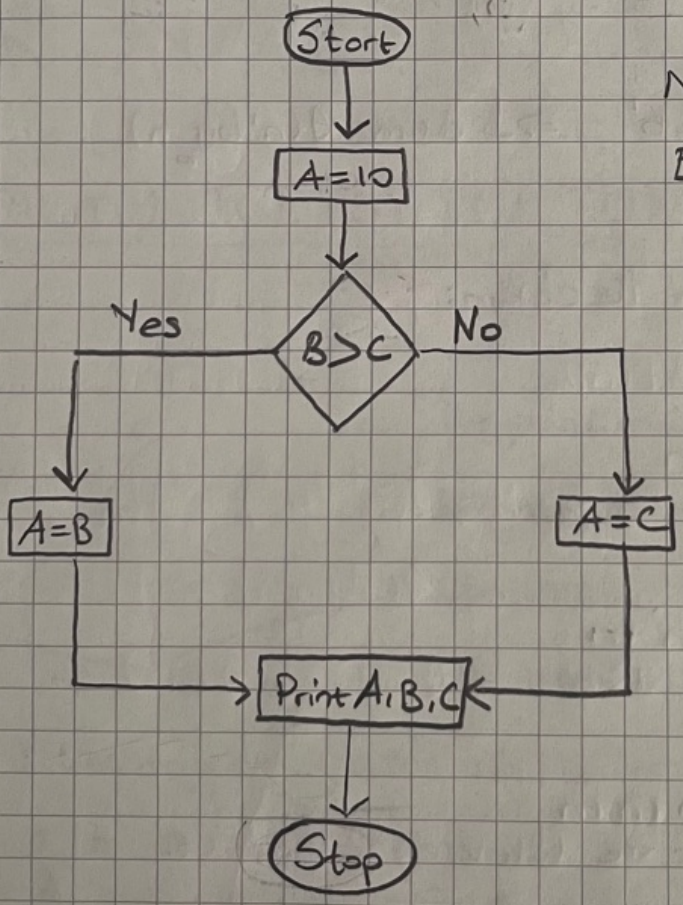
Akış şemasındaki bağımsız bölümler ve karar yapıları göz önünde bulundurur. Ne kadar çok karar yapısı bulunuyorsa kod karmaşası o kadar artar.

Karmaşıklık M olmak üzere aşağıdaki formülle hesaplanır:

$$M = E - N + 2*P$$

- E: Akış şemasındaki kenar sayısı
- N: Akış şemasındaki düğüm sayısı
- P: Akış şemasında bağlı olan elemanların sayısı

ÖR:



$M = 7 - 7 + 2*1 = 2$

$E = 7,\ N = 7,\ P = 1$

	Also called	n=100	n=10000	n=1000000
$O(1)$	Constant time	0.000001 sec	0.000001 sec	0.000001 sec
$O(\lg n)$	Logarithmic time	0.000007 sec	0.000013 sec	0.00002 sec
$O(n)$	Linear time	0.0001 sec	0.01 sec	1 sec
$O(n \lg n)$		0.00066 sec	0.13 sec	20 sec
$O(n^2)$	Quadratic time	0.01 sec	100 sec	278 hours
$O(n^3)$	Cubic time	1 sec	278 hours	317 centuries
$O(2^n)$	Exponential time	10^{14} centuries	10^{2995} centuries	10^{30087} centuries
$O(n!)$	Factorial time	10^{143} centuries	10^{35645} centuries	N/A

ÖR: $T(1) = 1 \quad T(n) = 2T(n/2) + n$ (Mergesort da bu şekildedir)

Master teoremi çözümü:

$a=2 \quad b=2 \quad i=1 \quad a=b^i \Rightarrow$ 1. durum $\Theta(n^i \log_b n)$

$\Rightarrow \Theta(n \log_2 n)$

Yerine koyma (iteratif) yöntemi ile çözüm:

$T(n) = 2T(n/2) + n$

$= 2(2T(n/4) + n/2) + n$

$= 4T(n/4) + n + n$

$= 4(2T(n/8) + n/4) + n + n$

$= 8T(n/8) + n + n + n$

$= nT(n/n) + n + n + \dots + n + n + n$

$T(1)$ olur ve tüketeceği süre sabittir.

$= n + n + n + \dots + n + n + n$

↳ Sürekli 2'ye bölünerek gittiği için bu toplamlar, $\log n$ tane n sayısının toplamına denk gelir.

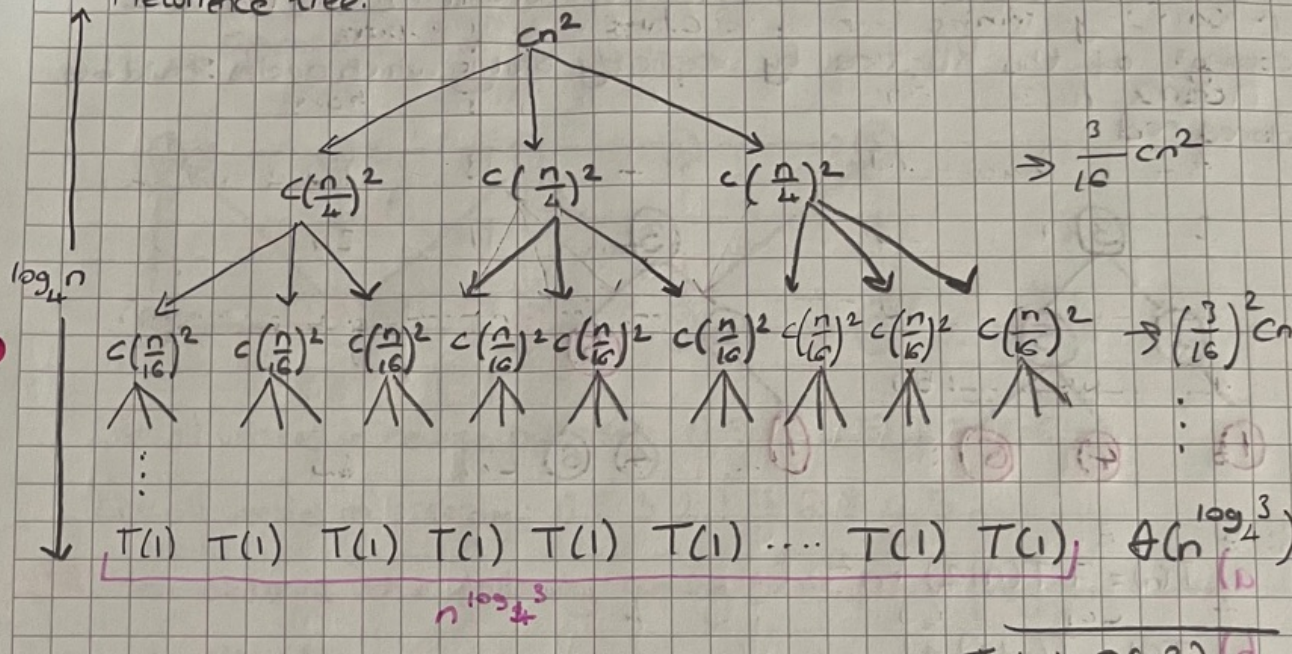
$= n \log n$

ÖR: $T(n) = 3T(n/4) + cn^2$

Master teoremine göre çözüm:

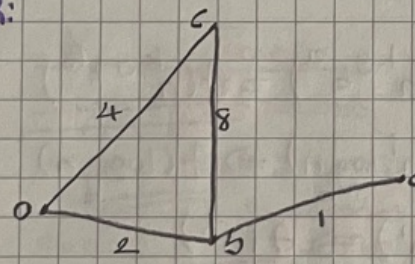
$a=3 \quad b=4 \quad i=2 \quad a < b^i$ ve $i > \log_b a \quad \Theta(n^i) \Rightarrow \Theta(n^2)$

Recurrence tree:



Total $\Theta(n^2)$

ÖR:



Weight of minimum spanning tree: 7

Weight of maximum weight matching: 8

Length of shortest path c-d : 7

Diameter of the graph : 7

What is the adjacency matrix of the graph considered above?

	a	b	c	d
a	0	1	1	0
b	1	0	1	1
c	1	1	0	0
d	0	1	0	0

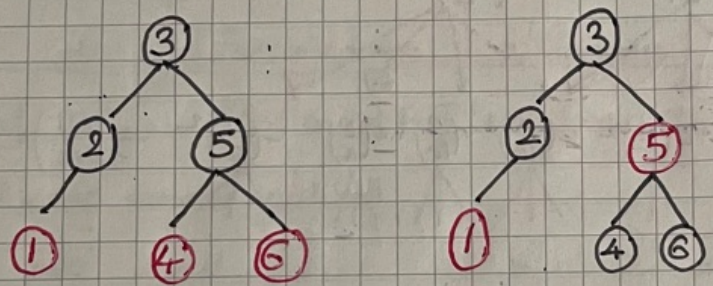
$$= \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

OR: $T(n-1)+n$ for $n>1$ $T(1)=1$ $T(n)=\Theta(n^x)$ $x=?$

$T(n)=n+(n-1)+(n-2)+\dots+1$

$=n(n-1)/2=\Theta(n^2)$ $x=2$

OR: Consider the following Red-Black tree with 6 (non-NIL) nodes whose keys are $\{1,2,3,4,5,6\}$ (Note that NILs are not drawn). Find two valid colorings of this RB-tree by explicitly stating which nodes should be colored red



OR:

a) $T(n)=7T(n/2)+n^2$

b) $T(n)=T(n/2)+1$

c) $T(n)=4T(n/2)+n^3$

a) $a=7$ $b=2$ $i=2$ $a>b^i \Rightarrow \Theta(n^{\log_b a}) = \underline{\Theta(n^{\log_7 2})}$

b) $a=1$ $b=2$ $i=0$ $a=b^i \Rightarrow \Theta(n^i \log_b n) \Rightarrow \underline{\Theta(\log_2 n)}$

c) $a=4$ $b=2$ $i=3$ $a<b^i \Rightarrow \Theta(n^i) \Rightarrow \underline{\Theta(n^3)}$

OR: int min = N;

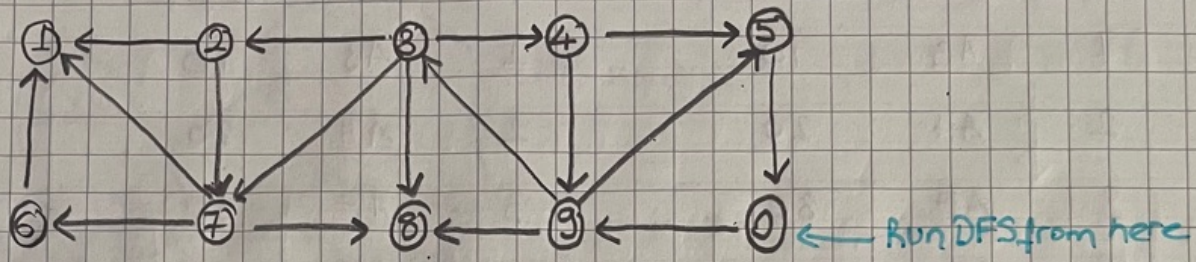
```
for(int i=0; i<N; i++){
  Selection.sort(a, a[i].BY_POLAR_ORDER);
  for(int j=0; j<N; j++){
    for(int k=j+1; k<N; k++){
      if(a[j].distanceTo(a[k]) <= 1.0){
        min = Math.min(min, k-j);
      }
    }
  }
}
```

GIPTA

Suppose that the code fragment takes 30 seconds when $N = 2000$. Estimate the running time (in seconds) as a function of the input size N. Use tilde notation to simply your answer

$$\frac{3}{800.000000} N^3$$

OR:



List all vertices in reverse postorder: 0 9 3 4 5 2 7 8 6 1

List all vertices in preorder: 0 9 3 2 1 7 6 8 4 5

OR: You observe the following running times for a program with on input of size N

N	Time
1000	0.1 seconds
2000	0.3 seconds
4000	2.5 seconds
8000	19.8 seconds
16000	160.1 seconds

Estimate the running time of the program (in seconds) on an input size $N = 80000$

20000 seconds.

The running time is cubic

OR:

```
int f1(int N){ N
  int x=0;
  for(int i=0; i<N; i++){
    x++;}
  return x;}
```

```
int f2(int N, int R){ R^2
  int x=0;
  for(int i=0; i<R; i++){
    x+=f1(i)}
  return x;}
```

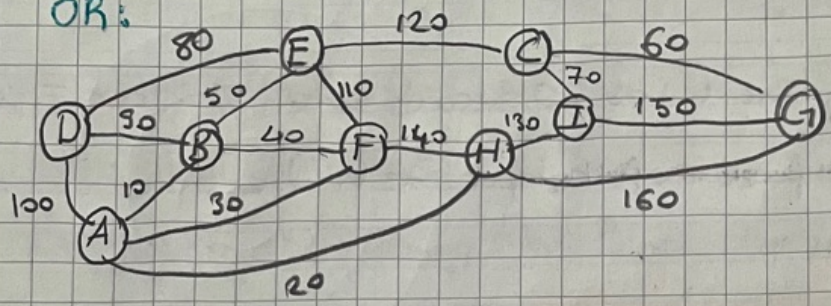
```
int f3(int N, int R){ RN^3
  int x=0;
  for(i=0; i<R; i++){
    for(j=0; j<N; j++){
      x+=f1(j);}}
  return x;
}
```

```
int f4(int N, int R){ NlogR
  int x=0;
  for(i=0; i<N; i++){
    for(j=1; j<=R; j+=j){
      x++;}}
  return x;
}
```

```
int f5(int N, int R){
  int x=0;
  for(i=0; i<N; i++){
    for(j=1; j<=R; j+=j){
      x+=f1(j);}}
  return x;
}
```

NRlogR

ÖR:



Prim's algorithm:

Tercih	Kenar	Ağırlık
1	AB	10
2	AH	20
3	AF	30
4	BE	50
5	ED	80
6	EC	120
7	CG	60
8	CI	70

Kruskal's algorithm:

Tercih	Kenar	Ağırlık
1	AB	10
2	AH	20
3	AF	30
4	BE	50
5	CG	60
6	CI	70
7	ED	80
8	EC	120

ÖR:

N	time
10000	1.2 sec
40000	2.1 sec
160000	3.9 sec
640000	7.9 sec
2 560 000	16 sec

$f(N) = \frac{1}{100} N^{1/2}$

N 4 katına çıktığında süre de 4 katına çıkıyor olsaydı karmaşıklık N olurdu

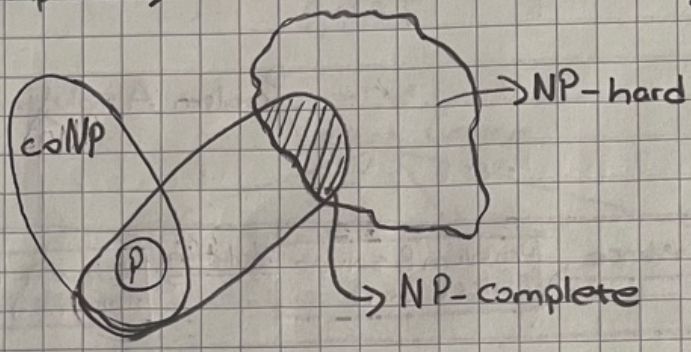
NP Problemler

P → Polinomal zamanda çözülebilen problemler.

NP → Polinomal zamanda doğrulanabilen problemler.

NP-hard → NP'deki en zor problemlerdir

NP-complete → NP ve NP-hard'ın kesişiminde yerallırlar



coNP → is the exact opposite of NP

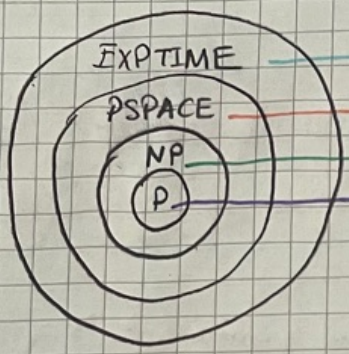
NP sınıfı non-deterministic turing machineler ile ifade edilirler

Hard Problems (NP-complete)
- 3SAT
- Traveling Salesman Problem
- Longest Path
- 3D Matching
- Knapsack
- Independent Set
- Integer Linear Programming
- Rudrata Path
- Balanced Cut
- Clique

Easy Problems (in P)
- 2SAT, HORN SAT
- Minimum Spanning Tree
- Shortest Path
- Bipartite Matching
- Unary Knapsack
- Independent Set on trees
- Linear Programming
- Euler Path
- Minimum Cut

P	NP-complete	NP-hard Problems
Shortest Path	Longest Path	Planar Circuit SAT
Eulerian Circuit	Hamiltonian Circuit	Not All Equal 3SAT
Edge Cover	Vertex Cover	Planar 3SAT
		Planar Not All Equal 3SAT
		Exact 3D Matching or X3M
		Partition
		Subset Sum
		3Partition
		Set Cover
		Hitting Set
		Longest Path
		Steiner Tree
		Tetris

Graph Coloring → NP-hard



EXPTIME → Chess, Checkers

PSPACE → Go, Othello, Sokoban, Generalized Geography, Hex

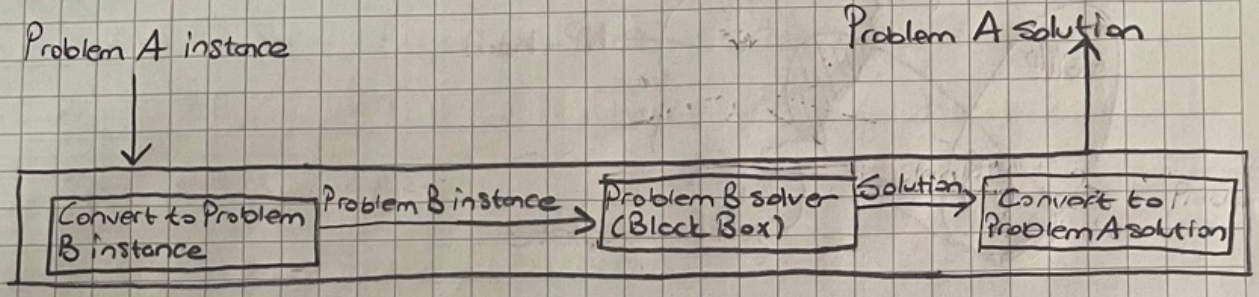
NP → Tetris, Mastermind, Minesweeper, Sliding Blocks, TSP

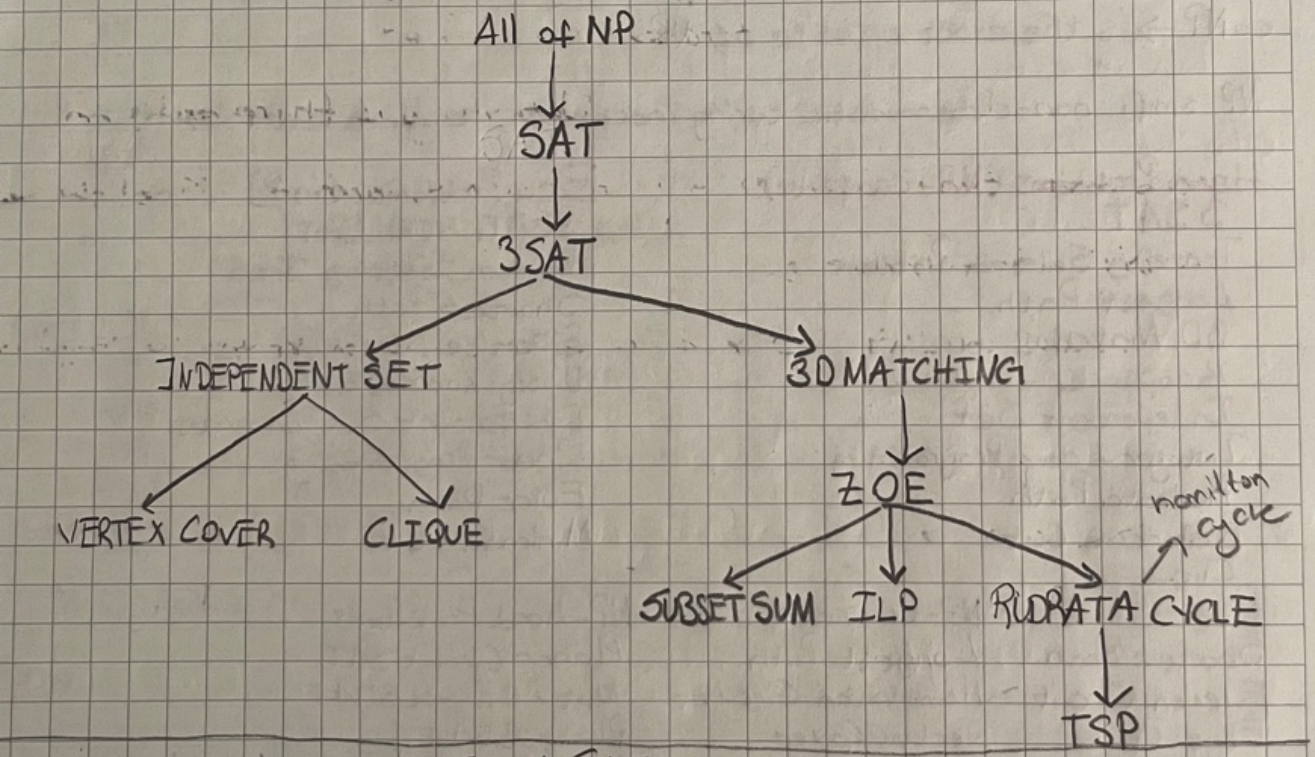
P → Jenga

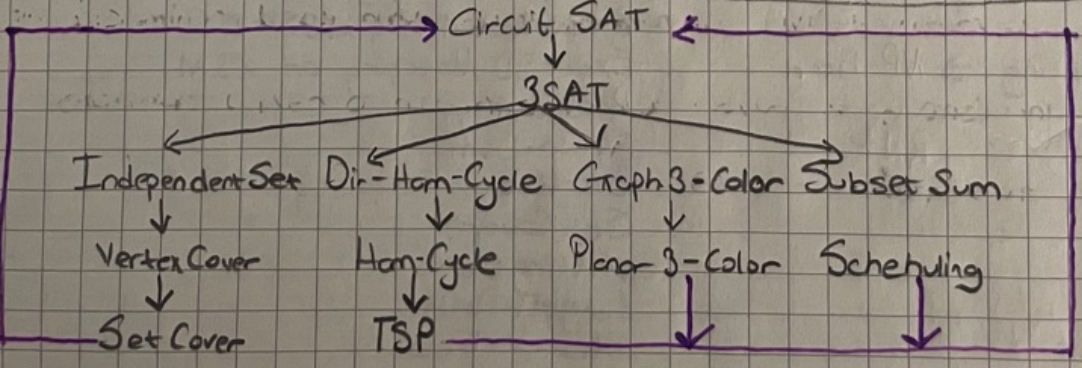
$P \neq EXPTIME$

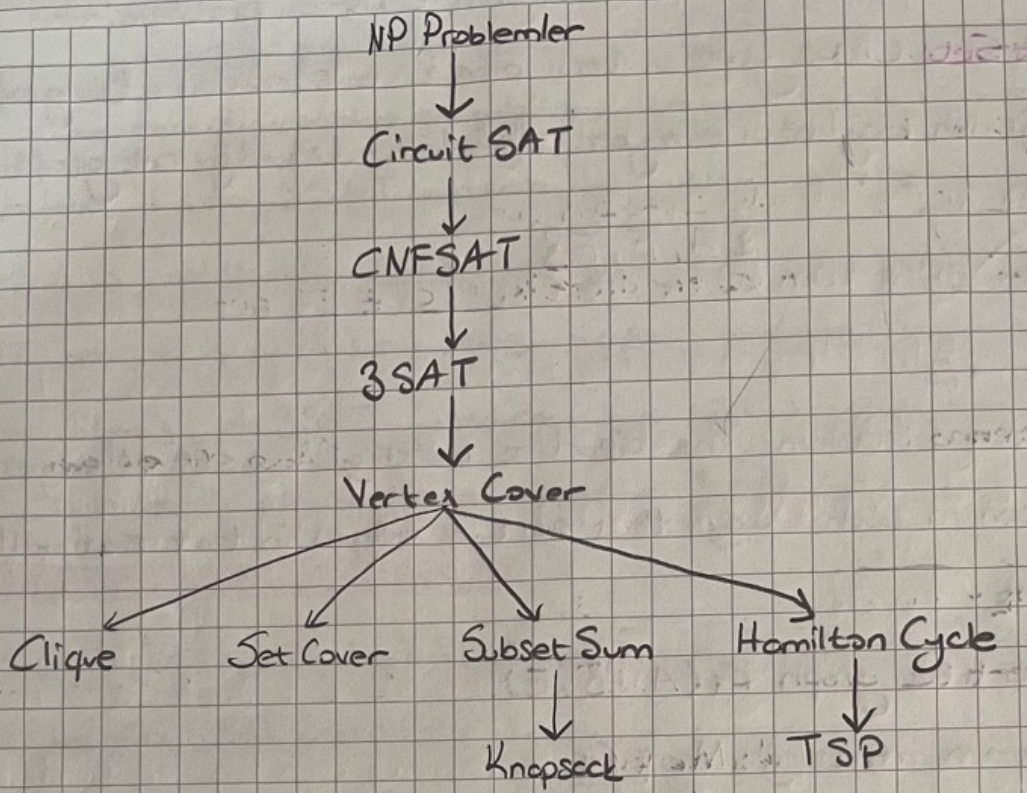
$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$

Reduction



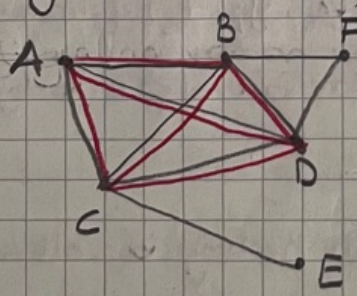






SAT (Boolean Satisfiability Problem): Determining if there exists an interpretation that satisfies a given boolean formula. (Bool denklemlerinin doğruluğunun sağlanması)

Clique: Graf teorisinde her iki düğümü birbirine bir kenar ile bağlanmış alt graflara verilen isimdir.

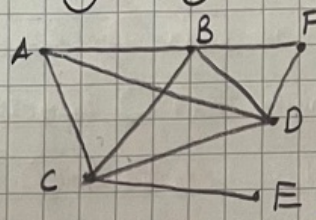


Burada bir clique kırmızı ile işaretlenmiştir. Buna göre {A, B, C, D} alt grafı bir cliquedir.

The clique problem is the computational problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph.

Vertex Cover: Bir graf içerisindeki tüm kenarların en az sayıda seçilebilecek düğüm ile kapsanabilir olup olmadığının bulunması problemidir.

Independent Set: Clique yapısının tersi olarak düşünülebilir. Bir grafta birbiriyle doğrudan bağlantısı olmayan düğümlerin oluşturduğu alt graftır.



{A, E, F}

Bipartite Matching Problem: The bipartite matching is a set of edges in a graph is chosen in such a way that no two edges in that set will share an endpoint

Input: A bipartite graph G(A∪B, E)

Output: A maximum matching M of G

```
BIPARTITE-MATCHING(G)
    M = ∅
    repeat
        P = (AUGMENTING-PATH(G, M))
        M = M ⊕ P
    until P = ∅
```

3D Matching: Üç ayrık küme X, Y, Z ve bir kısıtlamalar kümesi C verildiğinde, C'deki hiçbir kısıtlama ihlal edilmeden (x_i, y_j, z_k) formunun 3 sonlu sıralı listesinden oluşan n boyutlu bir S kümesinin olup olmadığının belirlenmesidir

Subset Sum: Verilen (+) ve (-) tamsayılar kümesinin herhangi bir alt kümesinin toplamının 0 olduğunu bulma problemi. (NP-complete)

3-Coloring: Given a graph G(V, E), return 1 if and only if there is a proper coloring of G using at most 3 colors. (NP-complete)

Reduction From SAT to 3SAT

The reduction takes an arbitrary SAT instance ϕ as input, and transforms it to a 3SAT instance ϕ', such that satisfiability is preserved, i.e., ϕ' is satisfiable if and only if ϕ is satisfiable. Recall that a SAT instance is an AND of some clauses, and each clause is OR of some literals. A 3SAT instance is a special type of SAT instance in which each clause has exactly 3 literals.

Example of a SAT instance:

$x_1 \wedge (x_1 \vee \overline{x_2}) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \overline{x_6} \vee \overline{x_7}) \wedge (x_1 \vee x_2 \vee \overline{x_3} \vee x_5 \vee x_7)$

Example of a 3SAT instance:

$(x_1 \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee x_3 \vee x_5) \wedge (x_1 \vee x_4 \vee \overline{x_6})$

The reduction replaces each clause in ϕ with a set of clauses, each having exactly three literals. Assume that ϕ involves n variables $x_1, \ldots, x_n$. The new formula ϕ' will have some new variables in addition to the x_i's.

How we replace each clause in ϕ? Let C is an arbitrary clause in ϕ.

case1: C has one literal: Let C consist of a single literal ℓ. ℓ is either x_i or $\overline{x_i}$ for some i. Let q_1 and q_2 be two new variables. We replace C by the following four clauses:

$(\ell \vee q_1 \vee q_2), (\ell \vee \overline{q_1} \vee q_2), (\ell \vee q_1 \vee \overline{q_2}), (\ell \vee \overline{q_1} \vee \overline{q_2})$

The logical AND of the above four clauses is equal to ℓ. Thus, the new formula obtained by replacing C by these four clauses computes the same Boolean function as the original formula. Hence, the new formula is satisfiable if and only if the old formula is satisfiable.

case2: C has two literals: Let $C = \ell_1 \vee \ell_2$. Each of ℓ_1 and ℓ_2 is either a variable x_i or a negated variable $\overline{x_i}$. Let z_1 be a new variable. Replace C by the following two clauses $(\ell_1 \vee \ell_2 \vee z_1), (\ell_1 \vee \ell_2 \vee \overline{z_1})$

The logical AND of the above two clauses is equal to $C = \ell_1 \vee \ell_2$. Thus, the new formula obtained by replacing C by these two clauses computes the same Boolean function as the original formula. Hence, the new formula is satisfiable if and only if the old formula is satisfiable.

case3: C has three literals: In this case, leave C unchanged.

case4: C has more than three literals: Let $k \geq 3$ and $C = \ell_1 \vee \ell_2 \vee \dots \vee \ell_k$, where each ℓ_i either a variable x_i or a negated variable $\overline{x_i}$. Let $z_1, z_2, \dots, z_{k-3}$ be $k-3$ new variables. We replace C by the following $k-3$ clauses: $(\ell_1 \vee \ell_2 \vee z_1), (\overline{z_1} \vee \ell_3 \vee z_2), (\overline{z_2} \vee \ell_4 \vee z_3), \dots, (\overline{z_{k-4}} \vee \ell_{k-2} \vee z_{k-3}), (\overline{z_{k-3}} \vee \ell_{k-1} \vee \ell_k)$.

Unlike cases 1 and 2, the logical AND of the above $k-2$ clauses is not same as C. However, the following two statements can be verified to be true. Let Ψ denote the AND of the above $k-3$ clauses.

1. Given an assignment to $x_1, \dots, x_n$ in which C is TRUE, there is a way of setting the new variables $z_1, \dots, z_{k-3}$ such that Ψ is TRUE.
2. Given an assignment to $x_1, \dots, x_n$ in which C is FALSE, there is no way of setting $z_1, \dots, z_{k-3}$ such that Ψ is TRUE.

ÖR:

a) public static int f1(int n) {
int x=0;
for(int i=0; i<n; i++)
x++;
return x; }

O(n)

b) public static int f2(int n) {
int x=0;
for(int i=0; i<n; i++)
for(int j=0; j<i*i; j++)
x++;
return x; }

$O(n^3)$

c) public static int f3(int n) {
if(n<=1) return 1;
return f3(n-1) + f3(n-1); }

$T(n) = 2T(n-1) + O(1)$
$T(n-1) = 2T(n-2) + O(1)$
$T(n-3) = 2T(n-3) + O(1)$
$T(n-n) = 2^n T(n-n) + O(1) \cdot n$
$T(n) = 2^n T(0) + O(1)n = 2^n + n = O(2^n)$

d) public static int f4(int n) {
if(n<=1) return 1;
return f4(n/2) + f4(n/2); }

e) public static int f5(int n) {
if(n<=1) return 1;
return f1(n) + f5(n/2) + f5(n/2); }

f) public static void f6(int n) {
// 1<<i is the same as 2^i.
// ignore integer overflow
// 1<<i takes constant time.
for(int i=0; i<n; i=1<<i); }

O(lgn)

d) $T(n) = 2T(n/2) + O(1)$
$T(n/2) = 2T(n/4) + O(1)$
$T(n/4) = 2T(n/8) + O(1)$
$T(n/n) = 2^n T(n/n) + O(1) \cdot n$
$T(n) = 2^n T(0) + O(1)n = 2^n + n = O(2^n)$

e) $T(n) = 2T(n/2) + O(n) + O(1)$
$T(n/2) = 2T(n/4) + 2O(n) + O(1)$
$T(n/4) = 2T(n/8) + 4O(n) + O(1)$
$T(n/n) = 2^n T(n/n) + nO(n) + O(1) \cdot n$
$T(n) = 2^n T(0) + nO(n) + nO(1)$
$= O(2^n)$.

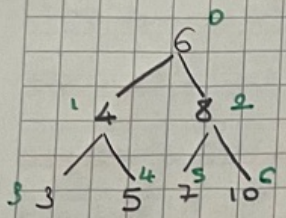
Heap Tree (Priority Queue)

2 çeşidi vardır: Min Heap, Max Heap

Min Heap → Küçükten büyüğe

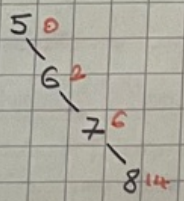
Max Heap → Büyükten küçüğe

★ Static olmalıdır. Bu yüzden dizi ile tanımlamak uygundur.



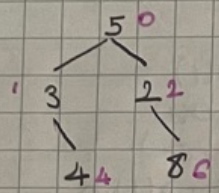
6	4	8	3	5	7	10

Left → 2x+1
Right → 2x+2
} x - parent



5	-	6	...	7	...	8

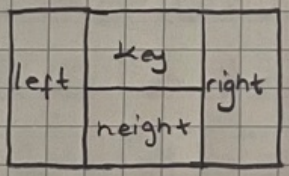
parent node index = $\frac{t-1}{2}$ → if root index begins with 0



5	3	2	-	4	-	8

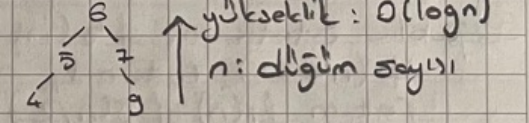
If index begins with 1, the parent node will be $\frac{t}{2}$
(t → index of node)

AVL Tree - binary search treelerdir



Balans faktörü X_L yüksekliği - X_R yüksekliği

Worst case O(logn)

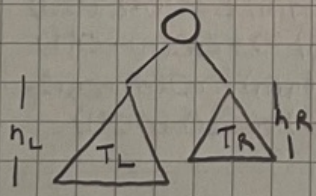


Sağalt ağaç ile sol alt ağacı arasındaki yükseklik farkı en fazla bir düğüm olmalıdır.

Boş ağaçlar AVL ağaçdır

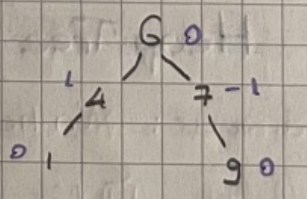
T ağacı boş değilse, AVL ağacı olabilmesi için

- T_L ve T_R AVL ağaçları
- $|h_L - h_R| \leq$ olmak zorundadır



Balans faktörü sadece -1, 0 veya 1 olabilir

sağ fazla ← eşit → sol fazla



örnekler