# Časovna zahtevnost

Omejena navzgor: $f(n) = O(g(n))$
Če $\exists c > 0$, da je $\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$
Omejena navzdol: $f(n) = \Omega(h(n))$
Če $\exists c > 0$, da je $c \leq \lim_{n\to\infty} \frac{f(n)}{h(n)}$
Enaka: f(n) = $\Theta(k(n))$
Če $\exists c_1, c_2 > 0$, da je $c_1 \leq \lim_{n\to\infty} \frac{f(n)}{k(n)} \leq c_2$

# Urejanje

## Notranje urejanje

Navadni algoritmi, imajo podobno zgradbo (urejeni-U in neurejeni-N del):
**Insertion sort** - prvi element N vrineš na ustrezno mesto v U
**Selection sort** - prvi N zamenjaš z najmanjšim N
**Bubble sort** - primerjaš soseda in jih po potrebi zamenjaš
Imajo **časovno zahtevnost** $O(n^2)$
**Heapsort** - levo poravnana kopica, koren je večji od sinov. Vzameš koren, gor prestaviš zadnji list, popraviš kopico, ponavljaš do konca. Implementacija: seznam, sinovi a[i] so v a[2i] in a[2i+1], oče v $a[\lfloor n/2 \rfloor]$
**Časovna zahtevnost:** $\Theta(n * log(n))$
Maksimalno število pogrezanj (na i-tem nivoju): $(n-i)2^{i-1}$, kjer je n število nivojev, i pa trenutni nivo.
**Quicksort** - izbereš pivot, levo manjši, desno večji, to rekurzivno delaš. Naš algoritem:
-Pivot maš nekako podan
-Greš z dvema pointerjema z obeh strani, primerjaš če je sprednji večji od pivota in zadnji manjši.
-Če je element na pravem mestu se s pointerjem prestaviš na naslednjega
-Ko najdeš na levi in na desni po enega napačnega ju zamenjaš.
**Časovna zahtevnost:** $\Theta(n * log(n))$ če dobro izbiraš pivote, drugače $\Theta(n^2)$

## Zunanje urejanje, i think

**Mergesort (z zlivanjem)** - razdeliš tabelo na podtabele dokler ni samo 0 ali 1 element, potem pa po 2 skupi združuješ tako da primerjaš najmanjši element.
**S štetjem** - narediš tabelo od 0 do največje številke in v njej beležiš pojavitve. Potem greš čez tabelo in prebereš.
**Korensko urejanje** - sortiraš po vsaki cifri, enice, desetice itd. To menda ponavadi s counting sortom.

# Metoda deli in vladaj

Ideja je da razdeliš nalogo na pod-naloge, in iz njihovih rešitev sestaviš celotno rešitev.

```
procedure A(N);
begin
    if n<2 then resi neposredno else
    begin
        R(N); // delitev N na podnaloge N_1,..,N_p
        A(N_1); // resevanje a podnalog
        ... ...
        A(N_a); ...
        S(N) // sestavljanje delnih resitev v koncno
    end
end
```

**Master theorem:**
-a je število podnalog,
-b nek konstatnten cas > 0
-c je kolikokrat manjše so podnaloge,
-d je zahtevnost delitve naloge na dele.

$$T(n) = \begin{cases} b; & n=1 \\ aT(n/c)+bn^d; & n>1 \end{cases}$$

$$T(n) = \begin{cases} \Theta(n^d); & a<c^d \\ \Theta(n^d log(n)); & a=c^d \\ \Theta(n^{log_c a}); & a>c^d \end{cases}$$

# Množenje števil

Naj bosta $a = a_{n-1}...a_1a_0$ in $b = b_{n-1}...b_1b_0$ dani n-mestni števili zapisani po številkah. Šolski algoritem jih zmnoži v času $\Theta(n^2)$

## Karatsubov algoritem

Dano nalogo $c = ab$ razdeli na tri dele, $z_1 = x_1y_1$, $z_2 = x_2y_2$, $z_3 = x_3y_3$. Te dele dobimo tako:
$a = a_LB^m + a_D$ kjer je $\underbrace{a_{n-1}...a_m}_{a_L}\underbrace{a_{m-1}...a_0}_{a_D}$, in enako $b = b_LB^m + b_D$.
Velja $a_D, b_D < B^m$. Potem je iskani produkt enak:
$ab = \underbrace{a_Lb_L}_{z_2}B^{2m} + \underbrace{(a_Lb_D + a_Db_L)}_{z_1}B^m + \underbrace{a_Db_D}_{z_0}$
Torej izračunamo:
$z_0 = a_Db_D$
$z_2 = a_Lb_L$
$z_1 = (a_L + a_D)(b_L + b_D) - z_2 - z_0$
**Časovna zahtevnost:** $\Theta(n^{log_2 3}) = \Theta(n^{1,58})$

# Matrično množenje

Direktno zahteva $n^3$ skalarnih množenj in $n^2 * (n-1)$, torej $\Theta(n^3)$
**Poskus z metodo deli in vladaj**: predpostavimo da so kvadratne matrike dimenzije $2^k$. Vsako od matrik A in B razdelimo na 4 enako velike. C = A*B = $\left(\begin{smallmatrix} A_{11}A_{12} \\ A_{21}A_{22} \end{smallmatrix}\right)\left(\begin{smallmatrix} B_{11}B_{12} \\ B_{21}B_{22} \end{smallmatrix}\right) = \left(\begin{smallmatrix} C_{11}C_{12} \\ C_{21}C_{22} \end{smallmatrix}\right)$
$C_{11} = A_{11}B_{11} + A_{12}B_{21}$
$C_{12} = A_{11}B_{12} + A_{12}B_{22}$
$C_{21} = A_{21}B_{11} + A_{22}B_{21}$
$C_{22} = A_{21}B_{12} + A_{22}B_{22}$
Na koncu ugotovimo da je vse brezveze ker je še vedno $\Theta(n^3)$.

## Strassenovo matrično množenje

Enako kot prej razdelimo matrike, ampak jih seštejemo tako, da ni treba poračunati vseh 8 pod-matrik:
$P_{11} = (A_{11} + A_{22})(B_{11} + B_{22})$
$P_{21} = (A_{22} + A_{11})(B_{22} + B_{11})$
$P_{12} = (A_{12} - A_{22})(B_{21} + B_{22})$
$P_{22} = (A_{21} - A_{11})(B_{12} + B_{11})$
$P_{13} = (A_{11} + A_{12})B_{22}$
$P_{23} = (A_{22} + A_{21})B_{11}$
$P_{14} = A_{22}(B_{11} - B_{21})$
$P_{24} = A_{11}(B_{22} - B_{12})$
$C_{11} = P_{11} + P_{12} - P_{13} - P_{14}$
$C_{12} = P_{13} - P_{24}$
$C_{21} = P_{23} - P_{14}$
$C_{22} = P_{21} + P_{22} - P_{23} - P_{24}$
Ker sta P11 in P21 enaka, lahko izračunamo samo 7 pod-matrik, zato pride časovna zahtevnost $\Theta(n^{2,80735})$

# k-ti največji element

Z metodo deli in vladaj: izberemo delilni element, razdelimo tabelo na t1 < m, t2 = m, t3 > m.
Če je k < |t1| iščemo v t1, če je k > |t1| + |t2| v t3, drugače pa v t2.
**Naivni algoritem:** $\Theta(n^2)$, ker je m lahko vedno največji element.
**Izboljšani algoritem:**
-tabelo razdelimo v peterke zaporednih elementov.
-v vsaki peterki poiščemo mediano.
-m je mediana vseh teh median.
S tem algoritmom ima problem časovno zahtevnost $\Theta(n)$.

# Diskretna Fourierova transformacija

$\hat{f}_k = \sum f_j e^{-i2\pi jk/n}$ -DFT
$\hat{f}_k = 1/n \sum f_j e^{i2\pi jk/n}$

DFT matrika = $\begin{bmatrix} \omega^{0*0} & \dots & \omega^{0*j} & \dots & \omega^{0*n} \\ \omega^{i*0} & \dots & \omega^{i*j} & \dots & \omega^{i*n} \\ \omega^{n*0} & \dots & \omega^{n*j} & \dots & \omega^{n*n} \end{bmatrix}$

inverzna DFT = $\frac{1}{N}\begin{bmatrix} \omega^{-0*0} & \dots & \omega^{-0*j} & \dots & \omega^{-0*n} \\ \omega^{-i*0} & \dots & \omega^{-i*j} & \dots & \omega^{-i*n} \\ \omega^{-n*0} & \dots & \omega^{-n*j} & \dots & \omega^{-n*n} \end{bmatrix}$

## Fast Fourier transform

**Razdelitev** problema na dva podproblema:
-polinom $p(x)$ razdelimo na sode in lihe koeficiente in iz njih definiramo $p_S(x)$ in $p_L(x)$
-$p(x)$ lahko potem izrazimo z drugima dvema:
$p(x) = p_S(x^2) + xp_L(x^2)$ -Torej lahko p(x) zračunamo v dveh korakih:
*Izračunamo $p_S(x^2)$ in $p_L$ v d točkah $x^2 = (\omega^0)^2, ..., (\omega^n)^2$
*S temi vrednostmi izračunamo $p(x)$ v d točkah $x = \omega^0, \omega^1, ..., \omega^n$
-Ugotovimo da je treba izračunati le pol toliko točk ker se ponavljajo?
**Sestavljanje** delnih rešitev v končno:
-računanje vrednosti v prvi polovici gre po enačbi
$p(\omega^k) = p_S(\omega^{2k}) + \omega^k p_L(\omega^{2k}) = p_S(\psi^k)(= A) + \omega^k p_L(\psi^k)(= B)$
A in B se bosta že prej izračunala, torej bomo samo enkrat množili in seštevali. Za izračun vseh r vrednosti $p(\omega^k)$ iz prve polovice bo potrebnih r množenj in seštevanj.
-računanje druge polovice $p(\omega^{r+k})$ je le odštevanje:

$p(\omega^{r+k}) = p_S(\psi^k) - \omega^k p_L(\psi^k) (= C)$.
Takrat bo C že izračunan (v prejšnjem koraku), tako da računanje vseh r vrednsti druge polovice zahteva le r odštevanj.
**Časovna zahtevnost FFT:** $T(n) = \Theta(nlog(n))$

# Največji pretok

Imamo usmerjen graf $G(V, A, c)$. V je množica vozlišč, $A \subseteq V \times V$ množica povezav, $c : A \to R_0^+$ funkcija ki povezavam priredi kapaciteto. Naivni algoritem preskočimo.

## Ford-Fulkersonov algoritem

Ideja je, da izbiramo nezasičene poti od začetka do konca in jih zasičimo. Če to naredimo za vse poti, dobimo maksimalni pretok.
**-Oznaka vozlišča.** Če je i označeno j pa neoznačeno vozlišče, potem j označimo z $(+i, \delta_j)$ če je povezava od i do j, in $(-i, \delta_j)$ če je povezava obrnjena.
**-Pomen oznake.** s prvim delom lahko rekonstruiramo pot. Drugi pa pove za koliko se lahko poveča pretok, in je definiran kot:
$\delta_j = \begin{pmatrix} min(\delta_i, c_{i,j} - v_{i,j}) & ;(i,j) \in A \\ min(\delta_i, v_{j,i}) & ;(j,i) \in A \end{pmatrix}$
**-Potek označevanja.** Če ima vozlišče vse sosede označene, je pregledano. Označevanje bo potekalo tako: izberi nepregledano vozlišče i, označi soseda j, ga razglasi kot označenega in nepregledanega, razglasi i za pregledanega če so vsi sosedi označeni.
**-Izsleditev poti.** Ko pridemo do ponora je njegova oznaka $(\pm i, \delta_n)$. Če je $\delta_n > 0$ potem lahko pretok povečamo za $\delta_n$ in po prvem delu oznake backtrackamo do izvira.
**-Zasičenje poti.** Ko odkrijemo pot moramo zasičenost vsake povezave popraviti.
**-Ponovitev.** Pobrišemo oznake in gremo še enkrat.
**Časovna zahtevnost:** ko računamo $v'$ (največji pretok) moramo vsaj $v'$ krat povečati pretok za 1. Vsaka pot je sestavljena iz največ A povezav, torej je časovna zahtevnost $O(v'A)$

# Metoda dinamičnega programiranja

Ideja je, da problem rešimo pri trivialni velikosti, potem pa rešitev večjega problema sestavimo iz rešitev manjših, torej "od spodaj navzgor".
Načelo optimalnosti: če je zaporedje odločitev optimalno (nas privede do optimalne rešitve) potem je tudi vsako podzaporedje optimalno.

# Nahrbtnik

Problem je NP-težek, zato ne bomo našli rešitve v polinomskem času.
Ideja z dinamičnim programiranjem:
-Imamo množico $N_i$. -Ko $i = 1, N_i = \{\emptyset, \{R_1\}\}$, torej ali vzamemo prvi predmet $R_1$ ali pa ne.
-Za vsak korak vsaki množici v $N_{i-1}$ dodamo ali pa ne dodamo predmet $R_i$
-Če je množica ko ji dodamo $R_i$ pretežka ali pa obstaja lažja z enako/večjo vrednostjo jo odstranimo
**Časovna zahtevnost:** $O(n2^d)$ ?? –> $d = \lceil log(V) \rceil$, V je velikost vhodnih podatkov

# Najcenejše poti iz izbranega izhodišča

Imamo graf $G(V, A, c)$, V so vozlišča, A usmerjene povezave, c pa povezavam priredi cene. Problem: za vsako vozlišče $i \in V$ poišči najcenejšo pot iz 1 v i. Predpostavimo da v G obstaja usmerjena pot do vsakega vozlišča in da G nima negativnih ciklov.

### Bellmanove enačbe:

$u_i = \begin{cases} 0 & ;i = 1 \\ \min\limits_{k,(k,i) \in A} (u_k + c_{k,i}) & ;i \geq 2 \end{cases}$
Torej $u_i$ je minimalna vsota poti do soseda in povezave od soseda do vozlišča i.

## Topološko urejanje grafov

Graf je topološko urejen, če velja $(i, j) \in A \Rightarrow \tau(i) < \tau(j)$, torej da povezave vedno kažejo od manjšega k večjemu vozlišču. Topološko uredimo lahko vse aciklične grafe.
Algoritem:

```
G' = G;
s = 0;
while (G' ima vozlišče z vhodno stopnjo 0){
    s++;
    vozlisce = vozlisce z vhodno stopnjo 0;
    vozlisce.poVrsti = s;   #ga preimenujemo
    G' = G'-vozlisce;     #odstranimo vozlisce in povezave
}
if (G'.prazen()) {
    print("G je acikličen, urejen");
} else {
    print("G je cikličen");}
```

**Časovna zahtevnost:** $O(|V|^2)$, kjer je $|V|$ število vozlišč.

## Najcenejše poti iz izhodišča v acikličnem grafu

Ko topološko uredimo graf lahko rešimo Bellmanove enačbe. Začnemo z $u_1 = 0, u_2 = u_1 + c_{1,2}, u_3 = min(u_1 + c_{1,3}, u_2 + c_{2,3})$ itd. **Časovna zahtevnost:** $O(|V|^2)$, $|V|$ je število vozlišč v grafu.

## Najcenejše poti iz izhodišča v grafu s pozitivnimi cenami (Dijkstra)

Spet graf G(V, A, c). Ideja algoritma:
-Cene $u_i$ so bodisi začasne bodisi dokončne. Na začetku vemo le $u_1 = 0$, ostale cene moramo izračunati.
-Cene hočemo narediti dokončne. Če vzamemo $u_k$ ki ima najmanjšo začasno ceno, jo lahko naredimo dokončno, ker ni več krajše poti do tega vozlišča (ker so cene povezav vedno pozitivne).
-Ko ceno $u_k$ naredimo dokončno, moramo posodobiti začasne cene.
**Časovna zahtevnost:** $O(|V|^2)$, $|V|$ je število vozlišč.

## Najcenejše poti iz izhodišča v splošnem grafu (Bellman-Ford)

Spet graf G(V, A, c). Še vedno ne sme biti negativnih ciklov.
Pomembne misli:
-Največ n-1 povezav.
-$u_i^{(p)}$ = cena najcenejše poti iz 1 v i, ki vsebuje kvečjemu p povezav.
-Najcenejša pot P iz 1 v i, ki vsebuje največ p povezav, vsebuje bodisi največ p-1 povezav, bodisi natanko p povezav.
-cena $u_i^{(p)}$ je enaka ali $u_i^{(p-1)}$, ali pa $\min(u_k^{(p-1)+c_{k,i}})$
Iz tega dobimo sistem enačb:

$$u_i^{(p)} = \begin{cases} 0 & ; \quad i = 1 \\ c_{1,i} & ; \quad i > 1, p = 1 \\ min(u_i^{(p-1)}, min(u_i^{(p-1)} + c_{k,i})) & ; \quad i > 1, p > 1 \end{cases}$$

**Časovna zahtevnost:** $O(|V|^3), |V|$ je število vozlišč.

# Najcenejše poti med vsemi pari

Razširjen prejšnji problem. Lahko bi torej gnali prejšnje algoritme za vsako vozlišče. Nočemo samo gnati prejšnjih algoritmov v vsakem vozlišču.

## Floyd-Warshallov algoritem

Ideja:
-Naj bo $u_{i,j}^{(m)}$ cena najcenejše poti od i do j, na kateri imajo vsa vozlišča oznake največ m.
-Velja $u_{i,j} = u_{i,j}^{(n)}$.
-Najcenejša pot P iz i v j z oznakami največ m ali ne gre čez vozlišče m ali pa gre.
-Cena $u_{i,j}^{(m)}$ je bodisi enaka $u_{i,j}^{(m-1)}$, ali pa $u_{i,m}^{(m-1)} + u_{m,j}^{(m-1)}$
-Izberemo manjšo od teh dveh in dobimo sistem enačb:

$$u_{i,j}^{(m)} = \begin{cases} c_{i,j} & ; \quad m = 1 \\ min(u_{i,j}^{(m-1)}, u_{i,m}^{(m-1)} + u_{m,j}^{(m-1)}) & ; \quad 1 \leq m \leq n \end{cases}$$

Torej m-to generacijo izračunamo i (m-1)-te. Gremo od spodaj navzgor.
**Časovna zahtevnost:** $O(|V|^3), |V|$ je število vozlišč.
Algoritem je mogoče izvesti na mestu, sam se mi ne da več brt.

# Srečno vsem!