Teorija složenosti algoritama

1. čas:

Za svaki problem ne postoji algoritam (precizno definisana procedura realizovana konačnim skupom naredbi koje izvršavaju određeni problem) ali ako postoji taj problem se zove odlučiv. Često za neki problem ima više algoritama koji ga rešavaju, i oni se razlikuju po vremenskoj i prostornoj složenosti.

Uobičajen način za izražavanje složenosti algoritma je O-notacija.

$f(n) = O(g(n)) \overset{def.}{\iff} (\exists c \in \mathbb{R})(\exists n_0 \in \mathbb{N})(\forall n \geq n_0): cg(n) \geq f(n)$. Tada je $\lim_{n\to\infty} f(n)/g(n) = c$ → ovako se nalazi gornja granica složenosti, jer je $f(n)$ ograničeno odozgo sa $g(n)$. ($f(n)$ ne raste brže od $g(n)$).

Preciznija oznaka $f(n) \in O(g(n))$.

Primjer: $n^2 = O(n)$ - nije tačno; $n = O(n^2)$ - tačno jer $n < \overset{1}{c} \cdot n^2$

- ova notacija zanemaruje konstantne članove koji su nekad bitni. Zbog ovoga se nekad određuje i donja granica složenosti - koristeći

Ω-notaciju: $f(n) = \Omega(g(n)) \overset{def.}{\iff} (\exists c > 0 \in \mathbb{R})(\exists n_0 \in \mathbb{N})(\forall n > n_0)\ f(n) \geq c \cdot g(n)$,

($f(n)$ - ograničeno odozdo sa $g(n)$). Precizније $f(n) \in \Omega(g(n))$

Primjer: $n^2 = \Omega(n)$ - tačno $\quad n = \Omega(n^2)$ - nije tačno

- Ukoliko neka f-ja $f(n)$ zadovoljava istovremeno relacije $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$ onda se kaže da je $f(n) = \Theta(g(n))$. ($g(n)$ i $f(n)$ istog reda)

$\overset{def.}{\iff} (\exists c_1, c_2 \in \mathbb{R})(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) \quad c_1 g(n) \leq f(n) \leq c_2 g(n)$

Primjer: $100n^2 + 5n - 30 = O(n^2)$ ✓
- " - $= O(n^3)$
- " - $= \Omega(n^2)$ ✓
- " - $= \Omega(n)$
- " - $= \Theta(n^2)$

Svaki računar može da $+, *, \leq$ → elementarne operacije. Da bi odredili složenost mi ćemo odrediti broj izvršenih elementarnih operacija. Ako želimo precizniju analizu - mi svakoj elem. operaciji dodijelimo koliko joj treba jedinica vremena za izvršenje.

Prebrajavanje elem. oper. nije uvijek jednostavno, mi uštimo razne aproksimacije i gledamo one djelove koji dominiraju, zato nam i trebaju ove asimptotske oznake.

Složenost izračunavamo i u zavisnosti od veličine ulaza.

Primjeri:

```
for i=1 to n do
begin
=========   } niz koraka koji zahtijevaju konstan. vrijeme - O(1)
end.
```
} $O(n)$

```
i=1 to n
  j=1 to m
  =========   } O(m)
```
} $O(n\,m)$

Kod algoritama sortiranja postoje dva kriterijuma za određivanje složenosti: 1° koliko je operacija ≤ (operacija koja dominira)

2° koliko uršimo zamjena (do izražaja kad su el. veliki - datoteke)

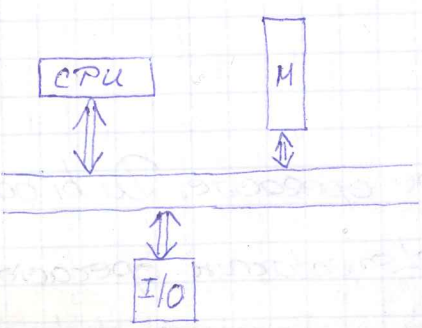
Prostorna složenost nekog algoritma - koliku memoriju zauzima algoritam. Za istu veličinu ulaza - algoritam zahtjeva različito vrijeme. U tom smislu se mogu razlikovati najbolji, prosječni i najgori slučaj (minimalna složenost)

Najređe se određuje složenost u najboljem slučaju; analiza prosječnog slučaja daje očekivanu složenost i najbolje izražava suštinu algoritma, a analiza najgoreg slučaja daje gornju granicu vremena izvršavanja - garantovana složenost.

Primjer:

```
while i<n do
=========
```

$i=n-1$ najbolji slučaj $\qquad$ $i=1$ - najgori slučaj

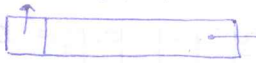


CPU - centralna procesorska jedinica izvršava sve aritmetičke operacije i upravlja radom čitavog sistema

I/O - ulazno - izlazni uređaji služe za komunikaciju (hard-disk)

M - memorija smještamo sve podatke i programe. Ona je organizovana iz više mem. riječi koje imaju svoje adrese. Mem. riječ - niz bita. stepen dužine 2^n

odredjuje znak



→ ostala vrijednost

1 - "-"
0 - "+"

Tip podatka - odredjen skupom vrijednosti i skupom operacija.

Npr. Cjelobrojni tip može uzeti vrednosti iz opsega $[-2^{n-1}, 2^{n-1}-1]$, ako su cijeli brojevi predstavljeni u drugom komplementu u lokaciji od n-bitova. A dozvoljeni operatori su +, -, *, div, mod. Tip podataka se može shvatiti kao metod interpretacije sadržaja memorije

Način organizovanja podataka čini strukturu podataka.

Apstraktni tip podataka - podrazumjeva matematički model sa skupom operacija koje korisnik na tom modelu definiše.

Npr Graf - primjer apstraktnog tipa sa čvorovima koji modeliraju elemente i granama koje predstavljaju njihove relacije, zajedno sa operacijama - obilazak grafa ... ⇒ Specifikacija apstraktnog tipa podataka se sastoji iz dva dijela: definicije vrednosti i definicije operacija.

Strukture podataka su skupovi elemenata istog tipa ili različitog tipa povezani na različite načine

Klasifikacija struktura podataka: 1° Linearna i nelinearna struct. podat.

Linearna - jedan element strukture u relaciji samo sa dva druga elementa strukture (prethodnikom i sledbenikom) (niz, stek, red, ulančana lista)

nelinearna - jedan elem. može biti u vezi sa više drugih elemenata (stablo i graf)

2° Statičke i dinamičke:

Statičke - imaju fiksnu veličinu, nemoguća promjena

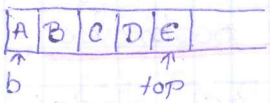
dinamičke - mogu povećavati i smanjivati u vreme izvršavanja programa shodno promenama (umetanje i brisanje)

Memorijska reprezentacija

- sekvencijalna reprezentacija: - elementi strukture se smeštaju jedan za drugim u jednom kontinualnom prostoru.
- ulančane reprezentacije: - elementi rasporedeni u nekontinualnom prostoru, na proizvoljnim mjestima u memoriji. (pokazivački tip) (memorijske adrese)

Stek (stack) - struktura koja predstavlja jednu vrstu Linearne Liste. Element se na stek može umetati ili sa njega uklanjati samo na jednom kraju linearne strukture. Ovo pristupno mjesto se zove vrh (top), dok je drugi kraj dno (bottom) steka. Element se uvijek može staviti na stek, ali brisanje samo ako stek ima bar jedan element.

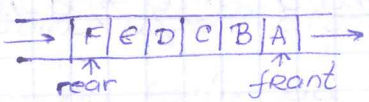
LIFO disciplina pristupa - "posljednji unutra - prvi napolje".



PUSH, POP - konstantne operacije

Red (queue) - struktura koja predstavlja jednu vrstu linearne liste. Za razliku od steka - red ima dva pristupna kraja, umetanje se vrši na jednom, a brisanje na drugom kraju. Mjesto gdje je prvi element - čelo (front) - ito je kraj sa kojeg se uklanja el. reda, a mjesto gdje je posljednji element - začelje (REAR) i iza njega se stavlja novi element.

Umetanje (INSERT) - uvijek moguće, a brisanje (delete) ako red nije prazan



- vektorska implementacija i kružni bafer (prvi elem se svodi na logičku sljedbenikom posljednjeg elem.)

- ulančana reprezentacija - ulančana lista. Čiji je prvi element - čelo reda, a posljednji element - začelje reda i na njega pokazuje pokazivač rear.

Prioritetni red - struktura kod koje uređenost po sadržaju ima odlučujući uticaj na to koji se element uklanja u operaciji brisanja. Rastući prioritetni red i opadajući prioritetni red. (brisanje - uvijek najmanji elem. tj. kod rastućih redova, a kod opadajućih brisanje - najvećeg)

Graf G - je par skupova (V, E) gdje je V konačan neprazan skup, a skup E predstavlja binarne relacije elemenata skupa V. Elementi sk. V - čvorovi, a sk. E - grane. Broj elem. sk. V - Red grafa.

Svakoj grani x iz E odgovara samo jedan par čvorova (u,v) iz V. x grana je incidentna čvorovima u i v. Ako su (u,v) - uređeni = usmeren graf, suprotno - neusmjereni graf. Ukupan br. incident. grana - stepen čvora.

Predstavljanje grafova: matrica susjedstva i liste susednost

matrična reprezentacija: $G=(V,E)$ sa n-čvorova numerisanih sa 1 do n.

Matrica susjedstva $A_{n\times n} = a[i,j]$ $i,j = \overline{1,n}$.

$$a[i,j] = \begin{cases} 1, & (i,j)\in E \\ 0, & (i,j)\notin E \end{cases}$$

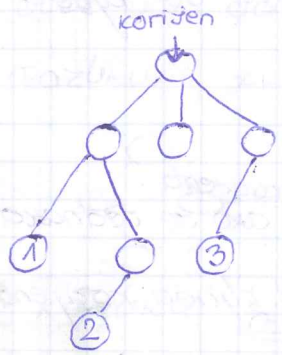
ulančana reprezentacija - lista susedstva

Forma niza od n ulančanih listi, gdje je po jedna lista predviđena za svaki čvor grafa. Elementi ove ulančane liste sadrže informaciju o susednim čvorovima čvora kojem odgovara ova lista.

Stabla:

Stablo T je konačan, neprazan skup elemenata proizvoljnog tipa - čvorova takav da: 1° postoji jedan poseban čvor koji se naziva koren (root)

2° ostali čvorovi se mogu razdvojiti u $m\geq 0$ disjunktnih podskupova $T_1,\dots,T_m$ koji su stabla (podstabla korijena)



nivo 0

nivo 1

nivo 2

nivo 3

Stepen čvora - br. grana koje izlaze iz čvora

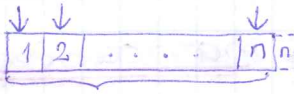
- čvorovi sa nultim stepenom Listovi (1,2,3)
- čvor od kojeg se dalje granaju podstabla = (otac), a koreni tih podstabala = (sinovi)

Visina (dubina) stabla - maksimalna vrijednost nivoa listova u stablu.

Memorijska reprezentacija

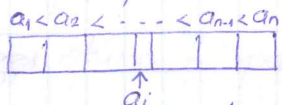
- ulančana reprezentacija: čvor se predstavlja elementom koji pored informacionog dijela ima i pokazivače na sinove
- predstavljanjem nizom ulančanih listi: Postoji po jedna jednostruko ulančana lista za svaki čvor, ~~(a odgovarajući član niza sadrži)~~ u odgovarajućoj listi čvora A nalaze se svi njegovi sinovi i to u definisanom poretku.

Binarna stabla - rekurzivno definišemo kao konačan skup čvorova koji je ili prazan ili se sastoji od korena sa dva posebna podstabla, lijevim i desnim, takođe binarna stabla.

Pretraživanje niza: 

u n+1 - ključ (el. koji tražimo u nizu)

$O(n)$ - najgori slučaj ; $O(1)$ - najbolji slučaj - $O(n)$ - prosječni slučaj

Uređeni niz: 

x - ključ / binarno pretraživanje/

ako je $x < a_i$ - pretražujemo lijevo od i-tog ; $x > a_i$ - desno od i

⇒ prepolovimo pretraživanje $O(\log_2 n)$

Stablo binarnog pretraživanja

Neka svaki čvor u binarnom stablu ima polje koje se naziva ključ (KEY)

= Tada se stablo binarnog pretraživanja definiše kao binarno stablo za koje važi:

1° za svaki ključ K postoji najviše jedan čvor sa adresom p tako da važi KEY(p) = K

2° ključevi svih potomaka p_l u lijevom podstablu su manji od ključa tog čvora (KEY(p_l) < K)

3° ključevi svih potomaka p_r u desnom podstablu su veći od ključa tog čvora (KEY(p_r) > K)

Osnovne operacije:

pretraživanje: upoređivanje date vrijednosti sa ključem korijena, ako je jednaka - završeno pretraživanje, ukoliko je vrijednost manja od ključa korijena onda se rekurzivno poziva procedura u lijevo podstablo, ako je pak veća onda u desnom podstablu dok se vrijednosti ne poklope ili pak dok se ne dođe do praznog podstabla pri ovoj proceduri.

Vremenska složenost je određena brojem poređenja ključa - a taj broj odgovara broju nivoa stabla gdje se nalazi traženi čvor, a on ne može biti veći od visine stabla h = maksimalna složenost reda $O(h)$

umetanje: umetanje čvora sa novim ključem u bin. stablo je umetanje čvora kao lista. Ova operacija ima takođe složenost $O(h)$, jer je

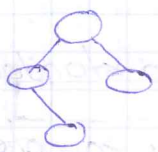
veoma slična operacija pretraživanja: kad uršimo pomjeranje pokazivača od korijena do mjesta gdje bi trebalo da se nalazi.

Brisanje: Moguće brisanje bilo kog čvora. Ako je taj čvor list, onda treba samo resetovati u ocu pokazivač na njega. Ako pak ima sina, onda se sin "podiže" i zamjeni ga. Najsloženije kad ima oba sina. Tada treba naći sljedbenika čvora, koji treba da zamjeni obrisani čvor i otac obrisanog sada ukazuje na sljedbenika. A otac sljedbenika preuzima njegovo desno podstablo, ~~zato što lijevo stablo~~ odnosno lijevo podstablo (zavisno koje ima). I na kraju, sljedbenik preuzima oba podstabla obrisanog čvora. Maximalna vremenska složenost je također $\mathcal{O}(h)$

Balansiranje stabla

Primjer visinsko balansiranog binarnog stabla pretraživanja je AVL-stablo.

AVL-stablo je stablo binarnog pretraživanja kod kojeg za svaki čvor važi da mu se visine lijevog podstabla i desnog podstabla ne razlikuje za više od 1.



- jeste AVL-stablo

Razlika $h_l - h_r$ se naziva balansom čvora (balansni faktor)

Kod visinsko balansiranog stabla $h_l - h_r \in \{-1, 0, 1\}$. 1 → čvor nagnut ulijevo, a čvor sa balansom −1 → čvor nagnut udesno.

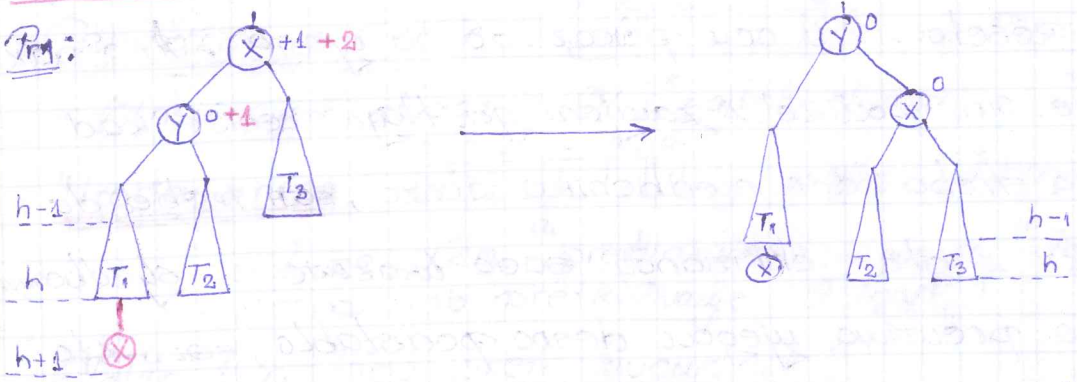
Pri radu umetanja i brisanja može se narušiti balansiranje, pri tome se koriste rotacije (jednostruke ili dvostruke, odnosno lijeve ili desne)

Najniži čvor (najviše udaljen od čvora) kod koga se narušava balansni faktor (nije −1, 0 ili 1) naziva se kritični čvor.

Dva slučaja balansiranja: (pri umetanju čvora u AVL-stablo)

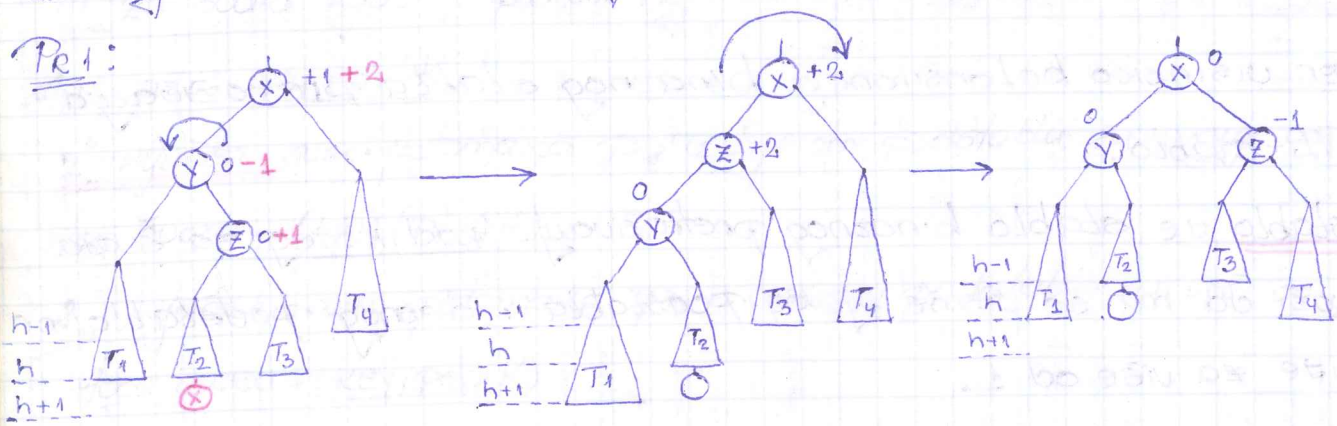
Prvi slučaj, koji nastaje kada se sin kritičnog čvora nagiuje na istu stranu kao i njegov otac. Ovdje se treba vršiti jednostruku rotaciju u desnu stranu

Pr1:



Sin kritičnog čvora nagnut suprotno od kritičnog čvora, ovo je drugi slučaj. Ovdje se prvo vrši ljeva rotacija oko Y pa onda desna rotacija oko X. (dvostruka rotacija). Prvom rotacijom dovodi do toga da stablo bude nagnuto na istu stranu.

Pr1:



Brisanje je nešto složenije od umetanja. Čvor se izbriše na standardan način, a zatim ako je poremećen balans, stablo se vraća u ravnotežu primjenom istih operacija rotacije kao i kod umetanja.

Kod AVL-stabala operacije pretraživanja, umetanja i brisanja i dalje imaju složenost $O(\log n)$, čak i u najgorem slučaju.

Jedna jednostruka, kao i jedna dvostruka rotacija nas košta $O(1)$, tj. konstantan broj pomjeranja pokazivača.

ukupno nas umetanje košta $O(1) + O(h) = O(\log n)$
$O(\log n)$

ukupno nas brisanje košta $O(1) + O(\log n) = O(\log n)$

2. čas: # Sortiranje

$a_1, \ldots, a_n$ - elementi totalno uređenog skupa. Treba ih poredati tako da $b_i \leq b_j$ za $i = \overline{1,n-1}$ i $j = \overline{2,n}$.

```
for i=1 to n-1
for j=i+1 to n
    if a_i > a_j
    a_i ↔ a_j
```

} $\mathcal{O}(1)$

druga for petlja za fiksirano i

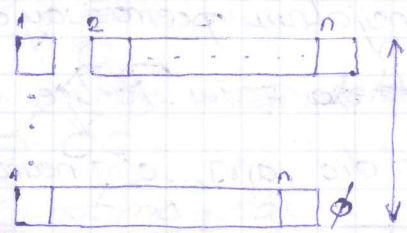
$n - (i-1) = n-i+1$

i =	1	2	3	...	n-1
br. koraka =	n	n-1	n-2	...	2

$\Rightarrow n + n-1 + \ldots + 2 = \frac{n(n+1)}{2} - 1 = \mathcal{O}(n^2)$

Metod umetanja:

U početku se uređeni dio sastoji samo od prvog elementa niza, a u neuređeni dio spadaju svi ostali elementi. Neka se poslije $i-1$ koraka u uređenom djelu nalaze elementi $a_1 \ldots a_{i-1}$. Tada se u koraku i uzima prvi element iz neuređenog djela a_i i ubacuje na mjesto koje mu po neopadajućem poretku odgovara u uređenom djelu, i ovaj dio se povećava za jedan element. Sortiranje se završava kad više nema el. u neuređenom djelu.



$a_1 \ldots a_i \; a_{i+1}$

najgori slučaj (do prvog mjesta)
$= i$-koraka $\rightarrow \mathcal{O}(i)$

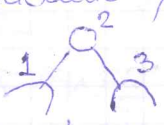
najbolji slučaj (ostaje na svoje mjesto)
$= 1$-korak $\rightarrow \mathcal{O}(1)$

ukupno: najgori slučaj $\rightarrow 1 + 2 + \ldots + n-1 = \mathcal{O}(n^2)$

najbolji slučaj $\rightarrow 1 + 1 + \ldots + 1 = \mathcal{O}(n)$

Najviše se gubi na umetanje u uređen niz! Osnovne operacije ove metode pretraživanje uređenog djela i umetanje na odgovarajuće mjesto. Ako umjesto sekvencijalnim pretraživanjem, mjesto umetanja tražimo efikasnije binarnim pretraživanje - smanjuje se broj poređenja na $\mathcal{O}(n \log n)$, ali broj premještanja ostaje isto, pa on dominantno određuje složenost, a to je $\mathcal{O}(n^2)$ - znači nema poboljšanja.

Ako pokušamo popravku izvršiti koristeći AVL-stablo kao strukturu:

inorder 

: za umetanje dobivamo složenost $O(\log i)$

ukupno: $\log 1 + \log 2 + \dots + \log(n-1) = \log(n-1)! = O(\log n!)$ $\lceil n! < n^n \rceil$

$= O(\log n^n) = O(n \log n)$

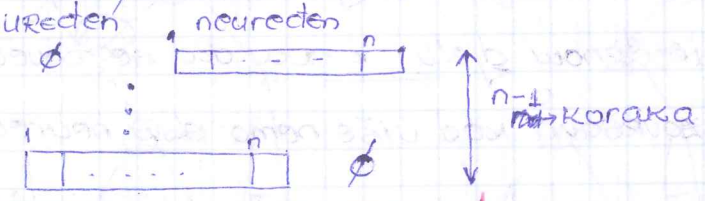
→ niz će nam ovim biti sortiran u AVL-stablo, sad treba iščitati to unaz.

Metod selekcije:

Odje postavljamo pitanje kako birati elem. iz neuređenog dijela da bi nas umetanje u uređen dio manje koštalo?

Osnovna strategija ove metode da se u svakom koraku izabere po jedan najmanji (ili najveći) element od neuređenih elemenata. Ovim se obezbjeđuje da nas umetanje košta $O(1)$ (jer el. samo kačimo za uređen niz pošto su svi njegovi elementi manji od elem. iz neuređenog dijela)

Sad je problem samo izbor minimalnog?



Prvo je cio niz neuređen dio. Zatim se sekvencijalnim pretraživanjem ovog dijela nađe najmanji elem, pa zamjeni mjesto sa prvim. Poslije $i-1$ koraka podniz $a[1], \dots, a[i-1]$ predstavlja uređen dio, $a[i], \dots a[n]$ neuređen dio.

U i-tom koraku selektira se najmanji el. neuređenog dijela, pa zamjeni mjesto sa prvim el. ovog dijela i tako prelazi u uređen dio. Ovaj postupak se vrši dok se ne isprazni neuređen dio!

Pošto se rad odvija u $n-1$ koraka, a obavlja se po jedna zamjena u prolazu $\Rightarrow$ ukupan broj zamjena uvijek $n-1$ – pa nije kritičan.

Složenost diktira broj poređenja, kojih u svakom koraku ima $n-i$ pa je ukupno $\sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = O(n^2)$.

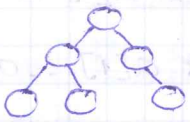
Poboljšanje: Smanjenje broja poređenja! Podijelimo neuređen dio na manje dijelove, pa u svakom nađemo min, pa će globalni min - minimum biti lokalnih minimuma. Taj global. min. se smjesti u uređen dio i makne

svog dijela, u kojem se nađe zatim novi min i stavi se u skup lok. min. iz (skupa lokalnih minimuma), onda se u tom skupu nađe novi glob. min. i sve tako dok ne iscrpimo skup. lok minimuma. Glavna ušteda što se u narednom koraku ne treba vršiti ponovna poređenja u drugim (dijelovima) grupama, jer njihovim lok. min. ostaju isti, već je dovoljno poređenje lokalnih min. djelova ($\sqrt{n}$-djelova po $\sqrt{n}$ elemenata; svaki korak poslije prvog ima najviše $\sqrt{n}-2$ poređenja u dijelu iz kojeg je globalni min. da bi našli novi lok. min tog djela, zatim $\sqrt{n}-1$ poređ. između lok. min - da bi našli novi glob. min). Red složenosti je $O(n\sqrt{n})$.

Heap – (skoro) kompletno binarno stablo sa specifičnim svojstvom uređenosti, kod njega važi da je ključ oca uvijek veći ili jednako od ključeva njegovih sinova (**maximalni heap**). A ako je ključ oca manji ili jednak od ključa sinova (**minimalni heap**).

Kompletno stablo – svaki čvor ima dva sina, osim listova koji su svi na istom nivou. Listovi koji nedostaju – nedostaju s desna na lijevo.

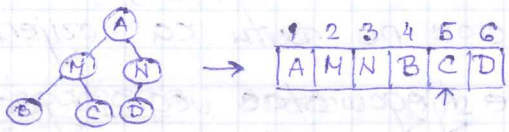
 - kompletno

 - nije kompletno

- **Pamptimo heap u obliku niza** tako da **oca** smještamo na poziciju i, onda su **njegovi sinovi** na pozicijama $2i$ (lijevi) i $2i+1$ (desni)

pozicija sina → pozicija oca

$$K \longmapsto \left\lfloor \frac{K}{2} \right\rfloor$$



- **Korijen** stabla obavezno predstavlja **najveći element u stablu**.

Heapsort - zasnovan na specifičnoj vrsti binarnog stabla koja se naziva HEAP.

Umetanje i Brisanje u heap-u:

DELETE-MAX - uzimanje el. najvećeg prioriteta (brisanje maximalnog)

[~~moramo sve pregledati~~ ~~O(n)~~]

Pošto je korijen maximalni elem, uzmemo korijen, a da bi stablo ostalo kompletno, a na njegovo mjesto dovodimo zadnji, i zatim popravljamo stanje uređenosti - upoređujemo sa većim od sinova, pa ako je sin veći od korijena vršimo zamjenu i sve tako dok ne dođe do lista ili dok taj elem. ne postane veći od oba sina. Ova operacija nas košta

$O(1)$ (jedna kontrola) + $O(h) = O(\log n)$

INSERT (umetanje) - umećemo na prvo slobodno mjesto. Ako je čvor koji se umeće veći od oca - zamjene mjesta i sve tako dok ne dođemo do korijena ili dok otac ne postane veći od x. Složenost ove operacije je $O(\log n)$.

Heapsort se sastoji iz dvije faze:

1° formiranje heap-a

2° selekcija elemenata

Heap se generiše na mestu ulaznog niza sukcesivnim ubacivanjem po jednog člana, počevši od el. a[1] koji za sebe predstavlja inicijalni heap. Poslije i-1 koraka deo niza a[1]...a[i-1] predstavlja heap. U i-tom koraku, stablu se priključi a[i] kao novi el. heap-a na mjestu lista u skoro kompletnom stablu. Na osnovu njegovog indeksa s izračunamo se pozicija oca f. Ako sin nije veći od oca - ovaj korak je završen. Ako je veći od oca, onda zamjene mjesta i postupak se dalje ponavlja na višim nivoima sve dok na putu ka korijenu element ne dođe do mjesta gdje je njegov otac veći ili jednak njemu ili ne dođe do korijena. Ovim se formira heap od n-elemen. i može početi faza selekcije.

Pošto je korijen najveći elem, selekcija se svodi na njegovo uklanjanje sa heap-a i prebacivanje u sortirani dio. Pošto je cilj da se sortiran niz ostavi na mestu ulaznog niza, ovaj elem. treba

prebaciti na krajnju poziciju istiskujući tako el. heap-a Last koji se tamo nalazi. Pošto je mjesto korijena ostalo upražnjeno, na njega ostavimo Last ako nije manji od sinova korijena, ako to nije ispunjeno, onda se on zamjeni sa većim sinom i sve tako dok ne dođe na mjesto gdje nije manji od oba sina ili na mjesto lista. Ovim postupkom uređen dio (gornji dio niza) sukcesivno povećava na račun heap-a, koji se smanjuje, dok heap ne nestane.

Heapsort nešto kompaktnije realizovan: (procedura ADJUST stablo sa korenom i, čiji nijedan čvor nema indeks veći od n pretvara u heap, pretpostavljajući levo i desno podstablo već jesu heap-ovi)

```
ADJUST (i, n)
  K = a[i]
  j = 2i
  while (j ≤ n) do
      if ((j<n) and (a[j] < a[j+1])) then
          j = j+1
      end_if
      if (K ≥ a[j]) then  a[j/2] = K  return
                  else  a[j/2] = a[j]
                        j = 2j
      end_if
  end_while
      a[j/2] = K.

  HEAPSORT
1° { for i = n/2 downto 1 do
       ADJUST(i,n)
     end_for
2° { for i = n-1 downto 1 do
       a[i+1] ↔ a[1]
       ADJUST(1,i)
     end_for.
```

algoritam HEAPSORT-a

Napomena: čvorovi $(\lceil n/2 \rceil + 1)$ do n su listovi stabla pa su oni već heap-ovi.

Pri procesiranju heap-a u svakom koraku prvi i poslednji čvor na heap-u zamjene mjesta, poslednji elem. izloži izvan heap-a, a zatim se heap preuredi pozivom ADJUST.

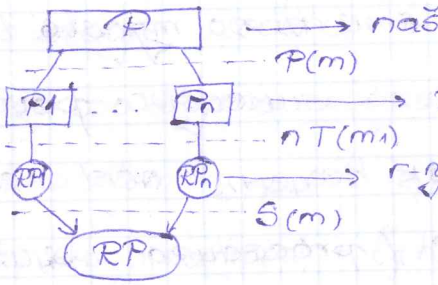
Složenost: 1° umetanje, ako ima i-elemenata, nas košta $O(\log i)$

ukupno: $\sum_{i=1}^{n} \log i = O(n \log n)$ → složenost za formiranje heap-a

2° složenost heap-a: $\log n + \log n-1 + \dots + \log 1 = O(n \log n)$

ukupnost složenosti: $O(n \log n)$

Podijeli pa vladaj



naš problem

podjela problema /jednake veličine/

r-nja problema

$$T(m) = nT(m_1) + P(m) + S(m)$$

vremenska složenost

$T(m_1)$ - složenost manjeg problema

n-manjih problema

podjela na manje probleme

spajanje RP1, –, RPn

Quick sort

Neka je zadat niz brojeva $S: x_1, x_2, -, x_n$ gdje je $n \geq 1$.

Opis algoritma: Iz niza S na slučajan način izaberimo jedan član X (pivot). Zatim, svi članovi x_i koji su manji od X, jednaki X, veći od X upišu se u djelimične nizove S_1, S_2, S_3. Pošto je S_2 već uređen, ako bi S_1 i S_3 bili uređeni onda bi posao bio okončan tj. odgovor bi bio ispisati niz S_1, zatim S_2 i nadovezati S_3. Ako S_1 i S_3 nisu uređeni, onda se oni urede na isti način (nađe se pivoti i izvrši se particija nizova na pomenuti način) ⇒ Quick Sort je rekurzivan.

tekst programa:

```
ulazne podatke x1, –, xn upišu se u niz S;
QUICK SORT(S);
štampaj članove niza S;
PROCEDURE QUICKSORT(S);
if |S| ≤ 1 then return S;
izaberi iz S na proizvoljan način član x;
formiraj podnizove S1, S2 i S3 čiji su članovi <x, =x, odnosno >x;
```

} tekst glavnog programa

$S_1 \leftarrow$ QUICKSORT (S_1);

$S_3 \leftarrow$ QUICKSORT (S_3);

$S \leftarrow$ članovi od S_1, članovi od S_2 i na kraju članovi od S_3;

RETURN S; kao pivot se uzme prvi el. iz Particije.

Particija: Elementi se razvrstavaju pomoću dva tekuća indexa i i j: i počinje od dna obrađivane particije, i prelazi preko elemenata koji su < od pivota i ukazuje na trenutnu gornju granicu donje particije. Index j počinje od vrha i prelazi preko elemenata koji su > od pivota i pokazuju trenutnu donju granicu gornje particije. Ako su a[i] i a[j] u obrnutom poretku, oni zamjene mjesta a[j] pivot postavi između donje i gornje particije na mjesto j, pa se algoritam rekurzivno primjenjuje na ove dve particije.

```
Low=1; high=n;
QuickSort(Low, high)
j= PARTITION (Low, high)
QUICKSORT (Low, j-1)
QuickSORT (j+1, high)
```

QuickSort - troši vrijeme na na podjelu niza (a to zavisi i od izbora pivota) pa pojedine PARTICIJSKOM sortiraju.

Složenost:

Najbolji slučaj: kada se obrađivana particija u svakom koraku prepolovi na dvije jednake particije. Tada se broj koraka polovljenja računa iz uslova $n/2^k = 1 \Rightarrow k = \log n$. Pošto je ukupan broj el. u svim particijama u određenom koraku polovljenja manja od n za broj el. koji su već na konačne pozicije $\Rightarrow$ gornja granica poredenja je reda $O(kn) = O(n \log n)$. I broj zamjena je istog reda $\Rightarrow$ složenost je $O(n \log n)$

$T(n) = 2T(\frac{n}{2}) + O(n)$ ← za particiju $\Rightarrow T(n) = O(n \log n)$

Najgori slučaj: Particije nerovnomjerne veličine. Ulazni niz već sortiran. Tada je pivot na svojoj poziciji, a gornja particija veličine n-1 u prvom koraku, n-2 u drugom itd. ($O(n)$-koraka), broj poredenja u prosjeku $\frac{n}{2} \Rightarrow O(n^2)$

Merge Sort

Ovo je primjer spojačkog sortiranja.

Podijeli se niz koji želimo sortirati na dva dijela i ta dva dijela se sortiraju nezavisno jedan od drugog. U ovoj metodi problem predstavlja spajanje ta dva niza (tu se troši vrijeme).

Opis algoritma spajanja dva niza u jedan izlazni sortiran niz:

Algoritam uspoređuje po jedan zapis iz oba niza i manji od njih upisuje u izlazni niz, pomjerajući se u nizu iz kojeg potiče taj manji na sljedeći zapis. Pošto se na ovaj način dođe do kraja jednog od ulaznih nizova, preostali zapisi iz drugog niza se jednostavno prepišu u izlazni niz.

Složenost: $T(n) = 2 \cdot T(\frac{n}{2}) + \underbrace{O(n)}_{\text{za spajanje}} = O(n \log n)$

Objašnjenje: u i-tom prolazu dobija će se nizovi dužine 2^i, a za konačno sortiranje je potrebno da bude $2^i \geq n \Rightarrow$ ukupan broj prolaza $\lceil \log n \rceil$. Kako se u svakom prolazu kopiraju svi zapisi, a to je najzahtjevnija vremenska operacija, složenost jednog prolaza je linearna - $O(n)$. Zato je ukupna složenost $O(n \log n)$.

- Ovo je metod pogodan kad u datoteku ima ima npr. 2000 zapisa koje treba sortirati, a u operativnu memoriju može da stane samo 1000. Onda sortiramo po 1000, sujestimo u pomoćne datoteke, pa ih spojimo.

3. čas:

Algoritmi za množenje velikih brojeva

n veličina našeg problema onda je složenost $T(n)$.
Podijelimo naš problem na a problema, pri čemu su oni veličine $\frac{n}{b}$ i trošimo neko dodatno vrijeme $O(n^d)$ (za podjelu problema i da ih kasnije sklopimo).

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$$

$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$; u zavisnosti od odnosa a i b imamo različita rješenja ove formule

$$T(n) = \begin{cases} O(n^{\log_b a}), & a > b^d \\ O(n^d \log n), & a = b^d \\ O(n^d), & a < b^d \end{cases}$$

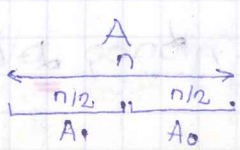
$O_B(\,)$ - bit složenosti

Neka je broj n broj cifara (broj bita) jednog i drugog broja.
Neka se složenost programa mjeri po kriteriju - "računanje bit po bit" tj. neka se broji koliko ima operacija nad bitovima.
= Ako se primjenjuje "školski" algoritam za množenje tj. uzastopno se šiftuje prvi činilac i uzastopno se vrše sabiranja. Složenost iznosi $T(n) = O(n^2)$.

Primjer:

$$\begin{array}{r} 1011 \cdot 1101 = 1011 \\ \quad\;\; 0000 \\ \quad 1011 \\ 1011 \\ \hline 10001111 \end{array}$$

(Karasube)
Polazimo na algoritam u kome se koristi pristup "podijeli pa vladaj". Sada se pretpostavlja da je broj n stepen broja 2. Postupak se sastoji u tome da se svaki od dva činioca podijeli na dva dijela, jedan i drugi dio od po $n/2$ binarnih cifara. Onda se množe ti djelovi itd. rekurzivno.



$A = A_1 \cdot 2^{n/2} + A_0$

$B = B_1 \cdot 2^{n/2} + B_0$

$A \cdot B = (A_1 2^{n/2} + A_0)(B_1 \cdot 2^{n/2} + B_0) = A_1B_1 \cdot 2^n + (A_1B_0 + A_0B_1) \cdot 2^{n/2} + A_0B_0$

Za ujinovo računanje treba izvršiti 4 operacije množenja. Svodimo ovaj problem na 4 problema duplo manje veličine. Složenost je

$T(n) = 4 \cdot T(\frac{n}{2}) + \underbrace{\mathcal{O}(n)}_{\text{za dodatno množenje (povezivanja)}}$; $4 > 2^1$ $(a > b^1) \Rightarrow T(n) = \mathcal{O}(n^{\log_2 4}) = \mathcal{O}(n^2)$

ništa nismo popravili dijeljenjem na 4 množ.

Međutim oni mogu da budu izračunati sa svega 3 operacije množenja i to:

$C_1 = A_1B_1$

$C_0 = A_0B_0$

$C_2 = (A_1 + A_0)(B_1 + B_0) = A_1B_1 + A_1B_0 + A_0B_1 + A_0B_0$

Znači, $A \cdot B = C_1 \cdot 2^n + (C_2 - C_1 - C_0) 2^{n/2} + C_0$

Koliko treba operacija nad bitovima $T(n)$ za ovo računanje?

Za tri računanja: $3T(\frac{n}{2})$ budući da ta množenja radimo po istom algoritmu. Množenje nekog broja sa osnovom brojnog sistema na stepen kao $c \cdot 2^n$ samo je pravilno množenje — šiftovanje, trošak je reda n. Trošak za sabiranje i oduzimanje je takođe reda n. Trošak za računanje C_2 može biti veći od $T(\frac{n}{2})$, jer a+b može imati $n/2 + 1$ cifara, kao i c+d! Za izraze reda n. Pa imamo:

$T(n) = 2 \cdot T(\frac{n}{2}) + T(\frac{n}{2} + 1) + \mathcal{O}(n)$ → dodatna sabiranja i prenose

odnosno: $T(n) = 3T(\frac{n}{2}) + \mathcal{O}(n)$ za $n = 2, 4, 8, \ldots$

Prema tome, važi i relacija:

$T(1) \leq c$ i $T(n) \leq 3T(\frac{n}{2}) + cn$, za $c > 0$ - const.

Želimo riješiti rekurentnu vezu relaciju. Ako pišemo $n = 2^k$; $a_k = T(2^k)$ onda je data $a_0 \leq c$ i $a_k \leq 3a_{k-1} + c \cdot 2^k$, $k = 1, 2, 3, \ldots$ Tako dobijamo razvojem ove rekurzije $a_k \leq \text{const} \cdot 3^k$ odnosno

$T(n) \leq \text{const} \cdot 3^{\log_2 n} = \text{const} \cdot n^{\log_2 3}$

Ukupno imamo $T(n) \leq \text{const} \cdot n^\alpha$, $\alpha = \log_2 3 = 1{,}59$. $T(n) = \mathcal{O}(n^\alpha)$.

Student četvrte godine, Tom, je dokazao da ∃ još bolji algoritam

Teorema: Za $\varepsilon > 0$ proizvoljno, postoji algoritam za množenje velikih brojeva sa složenošću $T(n) = O(n^{1+\varepsilon})$

Npr: Za $\varepsilon > 0{,}3$ možemo naći algoritam složenosti $O(n^{1.3})$

Algoritam Toma može se podijeliti u 5 djelova:

1° A i B - jednaki broj bita(n). Biramo k t.d. $k = k(\varepsilon)$.

A i B podijelimo u k blokova tj.

$A = A_{k-1} A_{k-2} \dots A_1 A_0$ i $B = B_{k-1} B_{k-2} \dots B_1 B_0$.

n - stepen broja k tj. $n = k^m$. Jedan blok $\frac{n}{k}$ bita, pa će naša osnova biti $2^{n/k}$.

$$\rightarrow A = A_0 + A_1 2^{n/k} + A_2 \cdot 2^{2n/k} + \dots + A_{k-1} 2^{(k-1)n/k}$$

$$B = B_0 + B_1 2^{n/k} + B_2 2^{2n/k} + \dots + B_{k-1} 2^{(k-1)n/k}$$

Tom je uočio sličnost između množenja polinoma i množenja brojeva. Umjesto množenja brojeva A i B, posmatramo množenje polinoma: $f(x) = A_0 + A_1 x + \dots + A_{k-1} x^{k-1}$ i

$g(x) = B_0 + B_1 x + \dots + B_{k-1} x^{k-1}$.

Tada je $A = f(2^{n/k})$ i $B = g(2^{n/k})$.

$h(x) = f(x)g(x)$ odnosno $C = AB = h(2^{n/k}) = f(2^{n/k}) \cdot g(2^{n/k}) = AB$

Neka su $x_0, \dots, x_{k-1}$ različite tačke.

Ako imamo $f(x_i)$, $i = \overline{0, k-1}$ onda možemo odrediti A_i (problem interpolacije)

$f(x), g(x)$ su stepena $k-1 \Rightarrow h(x)$ stepena $2k-2$.

Ako je koeficijent najvećeg člana $= 0$, onda se posmatra $h(x)$, stepena $2k-1$.

Izaberemo ove tačke $x_0, \dots, x_{2k-1}$ (zavise samo od k). k je fiksirano pa izbor tačaka ne utiče na složenost algoritma

2° Uzmimo da je $x_i = i$ i izračunajmo $f(x_i)$ i $g(x_i)$

3° $h(x_i) = f(x_i) g(x_i)$

4° Šema interpolacija: Na osnovu $h(x_i)$ da dobijemo koeficijente polinoma d_i

$h(x) = c_0 + c_1 x + \dots + c_{2k-2} x^{2k-2}$

$c_j = \sum_{i=1}^{2k-1} \beta_{ij} h(x_i)$, β_{ij} - nemora biti cio broj, već je racionalan

β_{ij} zavisi od $x_0, \dots, x_{2k-1} \Rightarrow \beta_{ij}(x_0, \dots, x_{2k-1}) = \frac{\gamma_{ij}}{p}$, $p, \gamma_{ij} \in \mathbb{Z}$ (unapred izračunato)

$\rightarrow c_j = \frac{1}{p} \sum_{i=1}^{2k-1} \gamma_{ij} h(x_i)$

5° Da sračunamo $C = AB = h(2^{n/k})$

Sad ćemo procijeniti složenost ovog algoritma po koracima:

1° $O(1)$

2° $O(1)$ - izbor tačaka

A koštaje računanja $f(x_i) = A_0 + A_1 x_i + \dots + A_{k-1} x_i^{k-1}$

1) treba izračunati x_i na neki stepen tj. x_i^l $0 \le l \le k$:

$\rightarrow O(1)$ (x_i - konstantno)

Zatim, za računanje $A_j x_i^l$: $\overset{\text{dužina bita}}{\downarrow}$ $l(A_i) = \frac{n}{k}$ i $l(x_j^l) = \text{const}$

$\Rightarrow l(A_i x_j^l) = \frac{n}{k} + \text{const}$ tj. košta nas $O(\frac{n}{k})$

Na kraju za dobijanje $f(x_i)$ nas košta $k \cdot O(\frac{n}{k} + \text{const})$

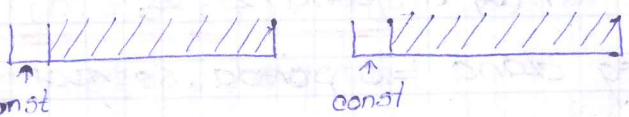
tj. $O(n)$

3° $f(x_i)$, $g(x_i)$ - dobijemo!

$h(x_i) = f(x_i) \cdot g(x_i)$

$l(f(x_i)) = \frac{n}{k} + \text{const}$ $l(g(x_i)) = \frac{n}{k} + \text{const}$

$T(\frac{n}{k} + \text{const}) = T(\frac{n}{k}) + O(n)$



$h(x_i)$, $i = \overline{1, 2k-1}$ $(2k-1) \cdot T(\frac{n}{k} + \text{const})$

ovoliko tačaka

4° $l(h(x_i)) = \frac{2n}{k} + \text{const}$ $\Leftarrow$ $\begin{cases} h(x_i) = f(x_i) g(x_i) \\ l(f(x_i)) = l(g(x_i)) = \frac{n}{k} + \text{const} \end{cases}$

$\gamma_{ij} h(x_i) = O(\frac{2n}{k})$

$l(\gamma_{ij}) = \text{const}$ $\rightarrow$ sabiranje ovakvih nas košta $O(n)$

5° $C = h(2^{n/k})$

$C = d_0 + d_1 2^{n/k} + \dots + d_{2k-2} 2^{(2k-2)n/k}$ — $O(n)$

Ukupno, košta je celog algoritma:

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

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

$a = 2k-1 \quad b = k \quad d = 1 \Rightarrow 2k-1 > k^1$ tj. $a > b^d \Rightarrow T(n) = O(n^{\log_k 2k})$

Znači, $T(n) = O(n^{\log_k (2k-1)})$

za $\varepsilon > 0$ proiz. izaberemo $k = k(\varepsilon)$ t.d. $\log_k (2k-1) < 1+\varepsilon$

$\log_k (2k-1) < \log_k 2k = \log_k k + \log_k 2 = 1 + \frac{1}{\log_2 k}$

izaberemo k t.d. $\log_2 k > \frac{1}{\varepsilon}$

$\Longrightarrow T(n) = O(n^{1+\varepsilon})$

Treba izabrati dovoljno veliko k za dovoljno malo ε da bi ovaj algoritam funkcionisao / i n mora biti dovoljno veliko /

1971. - Štrasen-Šerand $O(n \log n \log\log n)$

4.čas:

Diskretne Furiove transformacije
(brze Furiove transformacije)

Polinom se može zadati na dva načina:

1° preko niza koeficijenata

2° preko vrijednosti polinoma u određenih broj tačaka

Množenje dva broja kao množenje dva polinoma.

Neka su nam polinomi dati kao niz koeficijenata.

$P(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$

$Q(x) = b_0 + b_1 x + \dots + b_{n-1} x^{n-1}$

$\Big\} \; R(x) = P(x) \cdot Q(x)$

Sada želimo preći na vrijednosti polinoma u tačkama. Ovaj prelaz možemo zapisati na sledeći način:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 & \dots & x_{n+1}^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} P(x_1) \\ P(x_2) \\ \vdots \\ P(x_{n+1}) \end{bmatrix}$$

Ovo množenje nas skupo košta. Za računanje $P(x_1) = a_0 + a_1 x_1 + \dots + a_n x_1^n$
$= (\dots((a_n x_1 + a_{n-1})x_1 + a_{n-2})x_1 + \dots + a_1)x_1 + a_0$ složenost je $O(n)$
Pošto je ovo složenost računanja u jednoj tački, a mi imamo $n+1$ tačku, ukupna složenost je $O(n^2)$.

$P(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$ i neka je n-stepen 2 tj. $n = 2^k = 2m$; $m = 2^{k-1}$
Ovaj polinom podijelimo na dva sa parnim i neparnim koeficij.

$P(x) = a_0 + a_2 x^2 + \dots + a_{2m-2} x^{2m-2} + a_1 x + a_3 x^3 + \dots + a_{2m-1} x^{2m-1}$
$= P_0(x^2) + x P_1(x^2)$ gdje je

$P_0(t) = a_0 + a_2 t + \dots + a_{2m-1} t^{m-1}$
i
$P_1(t) = a_1 + a_3 t + \dots + a_{2m-1} t^{m-1}$

Kad računamo vrijednost u nekoj tački x_0.:

$P(x_0) = P_0(x_0^2) + x_0 P_1(x_0^2)$
$P(-x_0) = P_0(x_0^2) - x_0 P_1(x_0^2)$

Znajući poznato $P_0(x_0^2)$ i $P_1(x_0^2)$ možemo izračunati $P(x_0)$ i $P(-x_0)$ (vrijednost u dvije simetrične tačke). Ovim smo izbjegli da računamo 4 puta, već to radimo iz dva puta.
Ideja je da se pređe na kompleksne brojeve, jer ovo "dijeljenje" jedino možemo nastaviti u $\mathbb{C}$. (jer u $\mathbb{R}$ je prethodnim sve završeno)

$x^2 = 1 \Leftrightarrow x = \pm 1$
$x^4 = 1 \Leftrightarrow x = \pm 1$ u $\mathbb{R}$
$x = \pm 1, \pm i$ u $\mathbb{C}$

Zbog prethodnog uvodimo pojam primitivnog korijena 1 (ω) tj.
$\omega^n = 1$, $\omega^i \neq 1$ $0 < i < n$.

$1 = \cos 0 + i \sin 0$
$\omega = \sqrt[n]{1} = \cos \frac{2\pi}{n} + i \sin \frac{2\pi}{n}$, $k = 1$
$\omega = e^{i 2\pi / n}$ - preko Ojlerove formule

[Ako je $n = 2^k$ mi bi tada pri polovljenu problema trebali da obezbijedimo 2^k - korijena jedinice.]

ω^k - svi korijeni iz 1 su ovog oblika $\omega^k = e^{i\frac{2k\pi}{n}}$, $k = 0, 1, \dots, n-1$

$\omega, \omega^2, \omega^3, \dots, \omega^{n-1}, \omega^n = 1$

$(\omega^k)^n = (\omega^n)^k = 1^k = 1$

Biramo $x_j = \omega^j$ $(j = \overline{0, n-1})$. Za ove tačke važi $x_{\frac{n}{2}+j} = \omega^{\frac{n}{2}+j} = \omega^{n/2} \cdot \omega^j$

$= -\omega^j = -x_j$ tj. imamo simetrične tačke, a to nam je i bio cilj.

$$\left.\begin{array}{l} \omega^{\frac{n}{2}} \neq 1 \\ (\omega^{n/2})^2 = 1 \Leftrightarrow \omega^{n/2} = \pm 1 \end{array}\right\} \Rightarrow \omega^{n/2} = -1$$

($\frac{n}{2}$ je celo broj jer je $n = 2m$)

$$\omega^{n/2} = \left(\cos\frac{2\pi}{n} + i\sin\frac{2\pi}{n}\right)^{n/2} = \cos\pi + i\sin\pi = -1$$

Računanje polinoma u tim tačkama x_j je Furijeova transformacija

Definicija: **Diskretna Furijeova transformacija** vektora $(a_0, \dots, a_{n-1})^T$ je vektor $(b_0, \dots, b_{n-1})^T$ koji računamo po formuli

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \dots & (\omega^2)^{n-1} \\ & & \vdots & & \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \dots & (\omega^{n-1})^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$b_i = P(\omega^i) = \sum_{j=1}^{n-1} a_j \omega^{ij}$$

ω - n-ti korijen iz 1 | $n = 2m$ $\omega^{2m} = 1$

ω^2 - $\frac{n}{2}$-ti korijen iz 1 | $(\omega^2)^m = 1 \Rightarrow \omega^2$ - m-ti korijen iz 1

$$b_i = P(x_i) = P(\omega^i) = P_0((\omega^i)^2) + \omega^i P_1((\omega^i)^2)$$

$$= P_0((\omega^2)^i) + \omega^i P_1((\omega^2)^i) \quad \text{za } i < \frac{n}{2}$$

a kad je $i > \frac{n}{2}$ $\quad -\omega^i$

$$b_{\frac{n}{2}+i} = P_0\left(\left(\omega^{\frac{n}{2}+i}\right)^2\right) + \omega^{\frac{n}{2}+i} P_1\left(\left(\omega^{\frac{n}{2}+i}\right)^2\right)$$

$$= P_0((\omega^2)^i) - \omega^i P_1((\omega^2)^i)$$

Ovaj algoritam možemo nastaviti rekurzivno na niže.

$$b_i = P(\omega^i) = \sum_{j=1}^{n-1} a_j \omega^{ij} = P_0((\omega^2)^i) + \omega^i P_1((\omega^2)^i) = \sum_{j=0}^{m-1} a_{2j} (\omega^2)^{ij} + \omega^i \sum_{j=0}^{m-1} a_{2j+1} (\omega^2)^{ij}$$

$$b_{\frac{n}{2}+i} = \dots = \sum_{j=0}^{\frac{n}{2}-1} a_{2j} (\omega^2)^{ij} - \omega^i \sum_{j=0}^{\frac{n}{2}-1} a_{2j+1} (\omega^2)^{ij}$$

algoritam:

```
FFT(n, ω, A, B)
  A = (a0, a1, ..., an-1)
  B = (b0, b1, ..., bn-1)
  n = 2^k , ω^n = 1 , ω^j ≠ 1 , 0 < j < n
Begin
    if n=1 then b0 = a0
          else
              A' = (a0, a2, ..., a_{n/2-2})
              A'' = (a1, a3, ..., a_{n/2-1})
              FFT(n/2, ω^2, A', B')
              FFT(n/2, ω^2, A'', B'')
                 for i=0 to n/2 - 1 do
                     bi = B'[i] + ω^i B''[i]
                     b_{n/2+i} = B'[i] - ω^i B''[i]
                 end_for
    end_if
end_begin
```

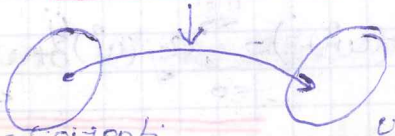
Za ulaz veličine n : $T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n)$

$FFT(\frac{n}{2})$ $FFT(\frac{n}{2})$ — za sabiranje i množenje u for petlji.

$$T(n) \leq 2T\left(\frac{n}{2}\right) + c \cdot n \Rightarrow \underline{T(n) = \mathcal{O}(n \log n)}$$

$$V(\omega) = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{n-1} \\ \dots & \dots & \dots & \dots \\ 1 & \omega^{n-1} & \dots & (\omega^{n-1})^{n-1} \end{bmatrix} \quad A = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} \quad B = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}$$

$B = V(\omega) \cdot A$ $\quad \mathcal{O}(n \log n)$



$A = (V(\omega))^{-1} \cdot B$

Nalaženje inverzne matrice (npr. Gausov metod)

$$[\,\cdots\,|\,1\;\ldots\;1\,]\quad [A\,|\,E] \sim [E\,|\,A^{-1}] = \underbrace{O(n^3)}_{\text{prilično skupo}}$$

(element. transf.)

Brza Furiova transformacija je pogodna za jeftinije traženje inverzne matrice.

$A = (V(\omega))^{-1} \cdot B \mapsto$ inverzna diskretna Furiova transformacija

IFT:

$$V(\omega^{-1}) \cdot V(\omega) = n \cdot E \iff (V(\omega))^{-1} = \frac{1}{n} V(\omega^{-1})$$

(ω^{-1} - primitivni n-ti korijen iz 1: $1 = \omega^n = \omega^{n-1}$)

$C = V(\omega^{-1})\, V(\omega)$

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj} = \sum_{k=0}^{n-1} (\omega^{-1})^{ik} \cdot \omega^{kj}$$

$$= \sum_{k=0}^{n-1} \omega^{k(j-i)} = \begin{cases} n, & j = i \\ \dfrac{1-(\omega^{j-i})^n}{1-\omega^{j-i}}, & j \neq i \end{cases}$$

($\sum_{k=0}^{n-1} \omega^{k \cdot 0} = n^{-1}$, $\omega^n = 1$)

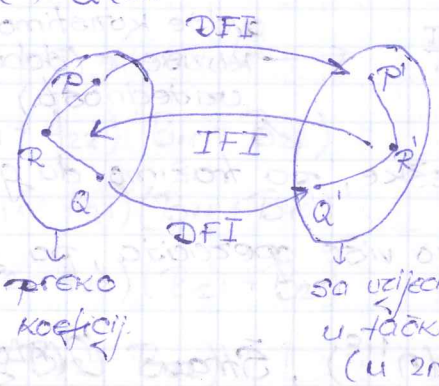
$$= \begin{cases} n, & i = j \\ 0, & i \neq j \end{cases}$$

$\Rightarrow C = nE$

$$\boxed{A = \frac{1}{n} \cdot V(\omega^{-1}) B} \rightarrow \text{inverzna Furiova transformacija} \quad O(n \log n)$$

Isti algoritam za direktne i za inverzne Furiove transformacije, samo kod direktne sa ω, a za inverzne ω^{-1}.

$R(x) = P(x) \cdot Q(x)$



$R(x_i) = P(x_i) \cdot Q(x_i)$

Naša pretpostavka je da množimo polinome stepena $n-1$, $n=2^k$, a rezultujući polinom je stepena $2n-2$

Ali možemo i ovako posmatrati

$P(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1} + 0 \cdot x^n + \dots + 0 \cdot x^{2n-1}$

$Q(x) = b_0 + b_1 x + \dots + b_{n-1} x^{n-1} + 0 \cdot x^n + \dots + 0 \cdot x^{2n-1}$

$a_0 \dots a_{n-1} 0 \dots 0 = [A,0]$

$b_0 \dots b_{n-1} 0 \dots 0 = [B,0]$

$FFT([A,0]) = C$
$FFT([B,0]) = D$
} $2\,\mathcal{O}(n\log n)$ +

$E = C \cdot D \ (E[i] = C[i] \cdot D[i])$ } $\mathcal{O}(n)$ +

$R = IFT(E)$ } $\mathcal{O}(n\log n)$

$= \mathcal{O}(n\log n)$

A - veliki broj, možemo ga zapisati u sl. obliku u osnovi B

$A = \sum_{i=0}^{n-1} a_i B^i$, $0 \le a_i < B$

evaluacija ↓

$P(x) = \sum_{i=0}^{n-1} a_i x^i$ $\quad A = P(B)$

Množenje velikih brojeva:

ako imamo: $A_1 \longrightarrow P_1(x)$

$A_2 \longrightarrow P_2(x)$ pomnožimo ih $R(x) = P_1(x) P_2(x)$ $\mathcal{O}(n\log n)$

onda je rezultat $C = R(B) = P_1(B) \cdot P_2(B) = A_1 A_2$

$\mathcal{O}(n)$ (jer pamtimo prenose)

$\omega = \cos\frac{2\pi}{n} + i\sin\frac{2\pi}{n}$ ← ovde koristimo primitivne korijene (dobijamo približne vrijednosti)

ima greške, pa tražimo da greška bude $< \frac{1}{2}$

što je n veće imamo više operacija, pa greška mora da bude što manja

Razvoj složenosti: $\mathcal{O}_B(n^{1.6})$, $\mathcal{O}_B(n^{1+\varepsilon})$, Štraus $\mathcal{O}(n\log\log\log n)$

$\mathcal{O}_B(n\log n\log\log n)$

Množenje matrica

$A_{n\times n}$, $B_{n\times n}$

$C = A \cdot B$ $\quad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ – $O(n)$

ukupno: $\underbrace{O(n^2)}_{n^2 \text{ brojeva } c_{ij}} \underbrace{O(n)}_{\text{dobivanje } c_{ij}} = O(n^3)$ → složenost za $A \cdot B$

$\begin{matrix} A & O \\ O & O \end{matrix}$ – ovako dopunjavamo matricu A do kvadratne ako nije kvadratna

↓ nula matrica

$$\begin{bmatrix} A & O \\ O & O \end{bmatrix} \begin{bmatrix} B & O \\ O & O \end{bmatrix} = \begin{bmatrix} AB & O \\ O & O \end{bmatrix}$$

– ove nula matrice ne moraju biti istog formata

$n = 2^k$ $\quad n \times n = (2^k) \times (2^k)$

Rastavimo matricu A na 4 matrice (i sve su upola manje dimenzije) i B također.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = A \cdot B = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$T(n) = 8 \cdot T\left(\frac{n}{2}\right) + O(n^2)$$

↓ red matrice ↓ vrijeme za zbrajanje matrica

$$\Rightarrow T(n) = O(n^{\log_2 8}) = O(n^3)$$

Štrasen je uočio da ovih 8 množenja svesti na 7 i predložio je formule.

$Q_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q_2 = (-A_{11} + A_{21})(B_{11} + B_{12})$

$Q_3 = (A_{12} - A_{22})(B_{21} + B_{22})$

$Q_4 = A_{11}(B_{12} - B_{22})$

$Q_5 = (A_{11} + A_{12})B_{22}$

$Q_6 = (A_{21} + A_{22})B_{11}$

$Q_7 = A_{22}(-B_{11} + B_{21})$

$C_{11} = D_1 + D_3 - D_5 + D_7$

$C_{12} = D_4 + D_5$

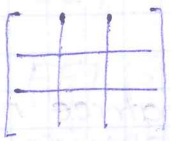
$C_{21} = D_6 + D_7$

$C_{22} = D_1 + D_2 + D_4 - D_6$

} Štrasenove formule

$T(n) = 7T\left(\frac{n}{2}\right) + \underbrace{O(n^2)}_{\text{za zbrajanje}} \Rightarrow T(n) = O(n^{\log_2 7}) = O(n^{2.81})$

Može i brže od ovog! Ako matricu podijelimo na 9 matrica

 - 27 množenja.

Zatim se uspjelo spustiti na 26, a zatim na 25.

Do sad je složenost spuštena na $O(n^{2.3})$

5.čas:

Tjuringova mašina

Mehanički opis Tjuringove mašine: Uređaj koji se sastoji od jedne beskonačne trake (beskonačna sa oba kraja) i izdijeljenja je na jednake djelove koji su numerisani. U jednu ovakvu ćeliju može da se upiše jedan simbol azbuke mašine. Mašina ima glavu koja se u svakom momentu nalazi iznad jednog polja i može se pomjerat jedno mjesto lijevo ili desno, da bi pročitala sadržaj tog polja, da upiše novi sadržaj ili da obriše postojeći sadržaj tog polja. Ima izvršni uređaj koji upravlja radom cijele mašine na osnovu programa. Postoji tjuringova mašina sa više traka ili sa trakom beskonačnom sa jedne strane. (beskonačna traka omogućava širenje trake u svakom trenutku)

Matematički model Tjuringove mašine:

Postoje različiti pristupi definicije Tjuringove mašine. Mi ćemo Tjuringovu mašinu definisati kao četvorku $M = (\Sigma, Q, q_0, P)$, gdje je Σ konačan skup simbola tj. $\Sigma = \{a_1, \_, a_n, \square\}$, $a_0 = \square$ (prazno polje; brisanje je isto što i upisivanje simbola a_0), Σ je ustvari azbuka Tjuringove mašine; Q je konačan skup tj. $Q = \{q_0, \_, q_s\}$ - skup stanja Tjuringove mašine, mašina se uvijek nalazi u nekom od ovih stanja; a $q_0 \in Q$ je polazno, odnosno početnim stanjem T.m.; P nazivamo programom ili operatorom prelaza i možemo ga definisati na različite načine. Mi ćemo P definisati sa:

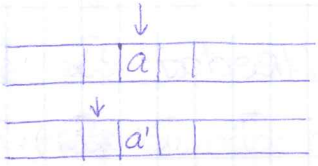
$P \subseteq (\Sigma \times Q) \times \Sigma \times (Q \cup \{\perp\}) \times L$, $\perp$ je završno stanje i $\perp \notin Q$., $L = \{l, 0, r\}$ (l - pomjeranje ulijevo, 0 - ostaje na isto mjesto, r - pomjeranje u desno)

P je binarna relacija između skupa $(\Sigma \times Q)$ i $\Sigma \times (Q \cup \{\perp\}) \times L$.

Odnosno, P je podskup petočki oblika (a, q, a', q', m). Smisao ove petorke je: ako se glava nalazi iznad polja sa simbolom a, a stanje je q, onda mašina treba umjesto a da upiše a',

i da pređe u stanje q' i da izvrši pomjeranje glave u skladu sa m.

Primjer:



$m = C$

Mi možemo zahtijevati još nešto osim binarne relacije da bi P bila f-ja tj. $P: (\Sigma \times Q) \to \Sigma \times (Q \cup \{\perp\}) \times L$.

Razlika je jer f-ja ne dozvoljava da se jedan element slika u dva elementa, dok relacija to dozvoljava. Da bi neka relacija bila f-ja to znači da mora da važi: $a \mathcal{P} b \wedge a \mathcal{P} c \Rightarrow b = c$

Ako je P-relacija onda govorimo o nedeterminističkoj tjuringovoj mašini (ako petočka počinje sa (a, q) tih petočki ima bar dve), a ako je P-fumkcija onda govorimo o determinističkoj Tjuringovoj mašini. (onda postoji najviše jedna petočka koja počinje sa (a, q)). Oznake DTM i NDTM.

Opis rada Tjuringove mašine (pojam izračunavanja i prihvat ulaza)

Polazna situacija: početno stanje q_0, glava se nalazi iznad polja 0 i većinu polja na traci čine prazni simboli, samo u konačno njih je upisan neki simbol azbuke $\longmapsto$ ovo smatramo ulazom T.m. tj. $x = (x_0, x_1, \ldots, x_m)$ je ulaz ako je

Veličina ulaza x je broj simbola tj. $S(x) = m+1$.

Konfiguracija mašine: $\$\Sigma^*(Q \times \Sigma)\Sigma^*\$$ - određena situacija naše T.m.

$Q \times \Sigma$ - pozicija glave, Σ^* - skup riječi nad azbukom Σ.

Matematička definicija Σ^*: ili prazna riječ $\Lambda \in \Sigma^*$ ili $w \in \Sigma^*, a \in \Sigma \Rightarrow w(a) \in \Sigma^*$

Npr. $\Sigma = \{a_1, a_2, a_3\}$ $\quad \Sigma^* \ni a_1 a_2 a_3 a_1$

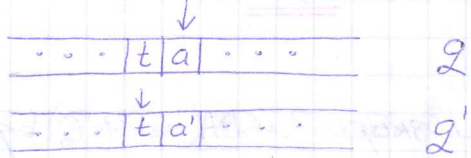
$(Q \times \Sigma)$ - uvodimo da bi istovremeno kodirali i stanje i simbol T.m. u datom trenutku.

Kod početne situacije $C_0 = \$(q_0, x_0)x_1 \ldots x_m\$$, $\$$-određuje početak i kraj konfiguracije.

Mi iz jedne prelazimo u drugu konfiguraciju saglasno programu P

što možemo opisati pravilima produkcije u skladu sa P.

Primjer: $t(q,a) \Rightarrow (q',t)a'$ – jedna produkcija koja odgovara (a,q,a',q',l)



Saglasno ovim produkcionim pravilima možemo napisati $C_i \Rightarrow C_{i+1}$

Pojam izračunavanja ~~je~~ niz konfiguracija $C_0C_1 \dots C_kC_{k+1} \dots$ i to nazivamo izračunavanjem T.m.;

Ako važi da je C_0 polazna konfiguracija, sve ostale konfiguracije se dobijaju ili se iz C_i ~~je~~ dobija C_{i+1} (tj. $C_i \Rightarrow C_{i+1}$) ili je $C_{i+1} = C_i$. A ako promijenimo C_0 dobijamo drugo izračunavanje (drugi niz konfiguracija).

Izračunavanje C odgovara ulazu x akko je C_0 početna konfiguracija koja odgovara ulazu x.

Konfiguracija C_s koja sadrži 1 se zove završna konfiguracija

Reći ćemo da izračunavanje C koje odgovara ulazu x je prihvatljivo ako sadrži završnu konfiguraciju.

Ulaz x je prihvatljiv Tjuringovom mašinom ako postoji izračunavanje C (koje odgovara ulazu x) koje je prihvatljivo.

C odgovara ulazu x = C je na ulazu x

$C_0C_1 \dots C_sC_{s+1} \dots$
C_s – završna konf.
C_0 – odgovara ulazu x
} tada je x prihvatljivo

Za x možemo imati više izračunavanja, a ono je prihvatljivo ako postoji bar jedno izračunavanje koje je prihvatljivo.

C_s – završna konfiguracija onda je $\forall j \geq s \; C_{j+1} = C_j$

x može biti neprihvatljivo ako:

1° konfiguracija se stalno mijenja tako da mašina nikad ne dolazi u završno stanje

2° mašina dođe u stanje q, glava iznad a, a P ne sadrži petorku koja počinje sa (a,q), pa se mašina zaustavi (nemože

dalje da računa)

Mi možemo da se dogovorimo da kad mašina treba da pređe ⊥ - ona više nema stanja - oznaka $(\ , a)$.

Problem DA-NE:

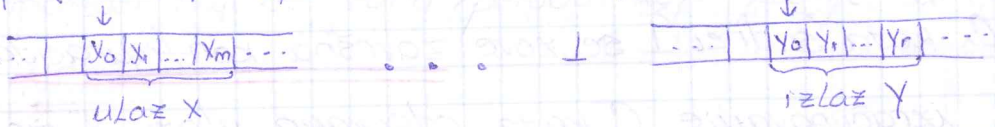
Da li nešto pripada nekom skupu? DA ili NE - problem odluke.

Da li $x \in A$? - Da li je x prihvatljivo ili ne? Odgovor DA ili NE!

Jezik (~~mašine~~) kod Tjuringove mašine su svi ulazi koje mašina prihvata

Neka je M T.m, njen jezik je $L_M = \{x \in \Sigma^* \mid M \text{ prihvata } x\}$

Ako imamo ulaz X i izračunavanje $C = C_0 \ldots C_s C_{s+1} \ldots$ koje odgovara ulazu x, pri čemu je C_s završna konfiguracija takva da je $C_s = \$(\bot, Y_0)y_1 \ldots y_r\$$ onda kažemo da T.m. transformiše X u $Y = (y_0, \ldots, y_r)$ tačnije mi T.m. možemo pridružiti preslikavanje $f: \Sigma^* \to \Sigma^*$ t.d. je $f(x) = Y$. (ovo je pitanje transformacija fj. šta naša mašina izračunava?)



DTM - f je zaista f-ja

Složenost ovog izračunavanja:

vremenska složenost - broj koraka od početne do završne konfiguracije - najkraći takav put

$t(M, x)$ - vremenska složenost mašine M na ulazu X

$t(M, x) = \min\{m : \exists C = C_0 \ldots C_m, C_m \text{ - završna konfiguracija}\}$

Vremenska složenost na ulazu određene veličine:

$x = (x_0, \ldots, x_m)$, $S(x) = m+1 \implies |x| = S(x) = m+1$

najgori slučaj:

$T(M, n) = \max\{t(M, x) : |x| = n, x \in L_M\}$ (najgori ulaz)

Prosečni slučaj:

$T_p(M, n) = \sum_{\substack{x : |x| = n \\ x \in L_M}} p(x)\, t(M, x)$, $p(x)$ - verovatnoća da se na ulazu pojavi x

prostorna složenost – broj simbola koji će da zauzme neka konfiguracija

prostorna složenost konfiguracije C_k = $s(C_k)$ – broj simbola u konfiguraciji C_k.

Pr: $C_k \equiv \$ z_1 \dots z_d (q, z_{d+1}) z_{d+2} \dots z_r \$$ $\quad s(C_k) = r$

prostorna složenost izračunavanja: ("najduža konfiguracija")

$C = C_0 \dots C_s C_{s+1} \dots$ $\quad s(M,C) = \max\{s(C_i) : i \in N\}$

prostorna složenost izračunavanja na ulazu x:

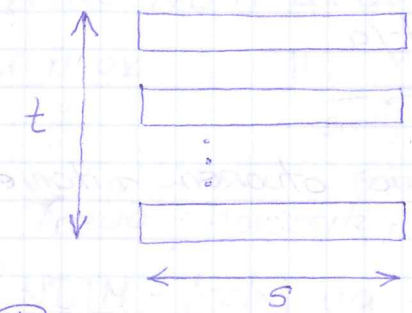
$s'(M,x) = \min\{s(M,C) : C$ – izračunavanje koje prihvata $x\}$

Složenost S mašine M na X – najgori slučaj:

$S(M,n) = \max\{s'(M,x) : |x| = n, x \in L_M\}$

prostorna složenost – najviše iskorišćeni prostor

Rad T.m. prikazujemo dijagramom (vremensko-prostorni dijagram):



broj vrsta = vremenska složenost (t)

dužina vrste = prostorna složenost (s)

Primjer T.m.:

(Σ, Q, q, P) : $\Sigma = \{0, 1, \square\}$, $Q = \{q, p\}$, q – polazno stanje

P:
$(0, q, 0, q, r)$
$(1, q, 1, q, r)$
$(\square, q, \square, p, l)$
$(0, p, 1, \perp, 0)$
$(1, p, 0, p, l)$
$(\square, p, 1, \perp, 0)$

$X \equiv (1011)_2 \equiv 11$

$\square\ (q,1)\ 0\ 1\ 1\ \square$
$\square\ 1\ (q,0)\ 1\ 1\ \square$
$\square\ 1\ 0\ (q,1)\ 1\ \square$
$\square\ 1\ 0\ 1\ (q,1)\ \square$
$\square\ 1\ 0\ 1\ 1\ (q,\square)$ ←
$\square\ 1\ 0\ 1\ (p,1)\ \square$
$\square\ 1\ 0\ (p,1)\ 0\ \square$
$\square\ 1\ (p,0)\ 0\ 0\ \square$
$\square\ 1\ (\perp,1)\ 0\ 0\ \square$

dalje se konfig. ponavlja

$Y \equiv (1100)_2 \equiv 12$

ovo je program koji računa $X = X+1$

prostorna složenost je 5, vremenska složenost je 9.

DA-NE zadatak ili zadatak prihvatanja jezika

Ulazni podatak je ijeć, a program saopšti-pripada li ta ijeć jeziku ili ne. Sada ćemo definisati tri važne klase:

* Označimo sa $\mathcal{P}$ skup svih jezika koji mogu da budu prihvaćeni od strane DTM čija je složenost polinomska, dakle

$$\mathcal{P} = \{L : \exists \text{ polinom } p(n) \text{ i } \exists \text{ DTM } \eta \text{ tako da je } T(\eta, n) \leq p(n) \wedge \text{da } \eta \text{ odgovara jeziku } L \text{ tj. } L_\eta = L\}$$

* Označimo sa $\mathcal{NP}$ skup svih jezika koji mogu da budu prihvaćeni od neke NDTM čija je složenost polinomska

$$\mathcal{NP} = \{L : (\exists \text{ polinom } p(n))(\exists \text{ NDTM } \eta)\ T(\eta, n) \leq p(n) \wedge L_\eta = L\}$$

Program kod NDTM je relacija, a kod DTM: fja

$$\text{DTM} \subseteq \text{NDTM} \Rightarrow \mathcal{P} \subseteq \mathcal{NP}$$

NP-hipoteza: $\mathcal{P}$ je pravi podskup od $\mathcal{NP}$ – još otvoreno pitanje = neriješen problem teorijskog programiranja.

$$\mathcal{P} \neq \mathcal{NP} \text{ v } \mathcal{P} = \mathcal{NP} ?$$

Ovaj problem je prouzrokovao pojavu još jedne klase tzv. klasa NP-kompletnih jezika (NPC)

* Za NP kažemo da je NPC ako je ispunjen sledeći uslov: Neka imamo deterministički tj. obični algoritam vremenske složen. $T(n) \geq n$ za prihvatanje jezika L_0. Tada za svaki jezik $L \in NP$ možemo da nađemo deterministički tj. obični algoritam za njegovo prihvatanje vremenske složenosti do $T(P_L(n))$, P_L - polinom koji zavisi od L. *

* Za jezik $L_0 \in NP$ kažemo da je NPC ako je: Za $\forall L \in NP$ $\exists$ obični alg. tj. TM čija je složenost polinomska, a koja $\forall$ ijeć $w \in L$ pretvara u $w_0 \in L_0$ t.d.: $w \in L \Leftrightarrow w_0 \in L_0$. Kaže

se da pomoću ove mašine jezik L se transformiše ili svodi na L₀ ⊕

Prethodna priča je odgovor na ~~pre~~ sledeću situaciju:

Posmatramo jezike L i odlučujemo da li neka riječ x pripada jeziku L? Naš problem treba da opišemo kao jezik. Mi umjesto $3^2 = 9$, zadajemo par (3,9) i pitamo se da li taj par pripada našem jeziku, opštije zadajemo par (x,y) pa pitamo da li je $y = x^2$? Za (3,9) odgovor je DA, a za (3,8) NE.

Nas interesuje za koje vrijeme možemo odgovoriti na pitanje da li $x \in L$? I tu nam odgovor daje prethodna podjela.

NDTM ima sposobnost da ako ima više puteva uvijek izabere najbolji put. Kod NDTM, od svih izračunavanja C na ulazu x izabrati ćemo najpovoljnije izračunavanje.

Neko NDTM zadaje na sledeći način: ima 2 ulaza: sertifikat i ulaz

Treba provjeriti: Da li je naš sertifikat r-nje?

NDTM - treba da provjeri r-nje, a DTM da smisli r-nje

NDTM - ima mogućnost da bira, pa i da pogađa r-nje i provjerava da li je to r-nje (za polinomno vrijeme)

6. čas:

Ideja: Ako svaki drugi zadatak iz NP možemo, za polinomijalno vrijeme, svesti na (određen) posmatran zadatak, tada taj zadatak nazivamo NP-težak zadatak.

Def1: Reći ćemo da je jezik L *polinomijalno svodljiv na jezik* L_1 ako postoji DTM M i polinom $P(\ )$ takav da je $T(M,n) \leq P(n)$ i za svaki ulaz x mašina M transformiše u Y na takav način da je $x \in L \Leftrightarrow y \in L_1$ (oznaka: $L \leq L_1$)

Def2: Za jezik L kažemo da je *NP-težak* ako je svaki jezik $L_1 \in NP$ polinomijalno svodljiv na L. Drugim rječima:

$$NP\text{-težak} = \{ L : (\forall L_1 \in NP)\ L_1 \leq L \}$$

Napomena: L ne mora pripadati NP. Moguća je situacija:

$NP\text{-teški} \cap NP = NPC$, NPC → klasa NP kompletnih zadataka

Oznaka za NP-teški je NP-T.

Def3: Zadatak je *NPC* ako:

1° pripada klasi NP

2° zadatak je NP-T

Napomena: jezik $\Leftrightarrow$ zadatak

Klasa NPC nije prazna, a to je prvi dokazao Kuk 1971. god. Dokazao je da je zadatak SAT NP-težak i pripada NP, tj. SAT je element NPC klase!

Posmatramo promjenljive $x_1, \ldots, x_n$ i njihove negacije $\bar{x}_i$, $i = \overline{1,n}$

Definišemo: $x_i^{\delta_i} = \begin{cases} x_i, & \delta_i = 1 \\ \bar{x}_i, & \delta_i = 0 \end{cases}$ u

Izraz $D_j = \bigvee_k x_{i_k}^{\delta_{ik}}$ (konjukcija promjenljivih) nazivamo KLAUZULOM

$\phi(x_1, \dots, x_n) = \bigwedge_j D_j$ nazivamo FORMULOM U KONJUKTIVNOJ NORMALNOJ FORMI (KNF)

Svakoj ovoj formuli možemo pridružiti jednu Bulovu f-ju:

$\phi \to f: \{0,1\}^n \to \{0,1\}$

$x_i \leftrightarrow 0,1$

$x_i = 1 \Leftrightarrow \bar{x}_i = 0$

$x_i = 0 \Leftrightarrow \bar{x}_i = 1$

Klauzula D_j je tačna: $\bigvee_i x_i^{\delta_i} = 1 \Leftrightarrow \exists i \; x_i = 1$

$\phi = \bigwedge_j D_j = 1 \Leftrightarrow \forall j \; D_j = 1$

Kažemo da je ϕ tautologija ako je $= 1$ za svaku vrijednost promjenljivih, a kažemo da je zadovoljiva ako postoji izbor vrijednosti promjenljivih takav da je $\phi = 1$.

Tautologija: $\forall x_1 \forall x_2 \dots \forall x_n \; \phi(x_1, \dots, x_n) = 1$

zadovoljiva: $\exists x_1 \exists x_2 \dots \exists x_n \; \phi(x_1, \dots, x_n) = 1$

Primjer: $\phi(x_1, x_2, x_3) = (\underset{0}{\bar{x}_1} \vee x_2)(\underset{0}{\bar{x}_2} \vee \bar{x}_3)\underset{1}{x_3}$

nije tautologija, jer za $x_3 = 0$ $\phi(x_1, x_2, 0) = 0$

jeste zadovoljiva: $x_1 = 0, x_2 = 0, x_3 = 1$ $\phi(0,0,1) = 1$

Primjer: $\phi(x_1, x_2, x_3) = (\bar{x}_3 \vee x_1)(\bar{x}_1 \vee x_2)(\bar{x}_2 \vee \bar{x}_3)x_3$

ϕ nije zadovoljiva

Formulacija SAT zadatka: (zadatak zadovoljivosti)

Data nam je formula u KNF. Pitanje je da li je ona zadovoljiva ili ne? Pripada jezik $SAT = \{\phi \mid \phi \text{ je zadovoljiva}\}$

$\phi \in SAT$?

Teorema (KUK): $SAT \in NPC$ (Zadatak SAT je NP-kompletan)

dokaz:

I) Dokažimo da $SAT \in NP$ (tj. možemo za polinijalno vrijeme provjeriti je li f-ja zadovoljiva)

1. korak: Svakoj propozicijskoj varijabli dodijelimo 0 ili 1 i to odradimo nedeterministički. 0 1 0 1 0 0 1 ... (ispod: x_1 x_2 ...)

Sada krenemo kroz formulu i provjeravamo da li je ona zadovoljiva za ovaj naš izbor propozicijskih. Pregledamo klauzulu po klauzulu, sve dok ne pregledamo sve klauzule ili dok ne naiđemo na klauzulu $D_j = 0$ i vraćamo odgovor "nije zadovoljiva", a ako su sve klauzule $= 1$ vratimo odgovor "jeste zadovoljiva".

U klauzuli gledamo sve literale koji je čine.

Ako smo pregledali sve i ako su svi $x_i^{\delta_i} = 0$ vratimo $D_i = 0$, a ako naiđemo bar na jedno $x_i^{\delta_i} = 1$ onda prekinemo dalje pregledanje i vraćamo $D_i = 1$

Koliko nas ovo košta? Najgori slučaj: $O(|\phi|)$, $|\phi|$ - veličina formule tj. algoritam je linearan. Dakle, $SAT \in NP$.

II) Dokažimo da je $SAT \in NP$-težak tj. $\forall L \in NP\ (SAT \leq L)$.

Uzmimo $L \in NP$ proizvoljno i dokažimo da ga možemo svesti na SAT tj. $\exists$ NDTM η i $\exists p(\,)$ t.d. je $T(\eta, n) \leq p(n)$ i $L_\eta = L$.

Umjesto sa $p(n)$ mi možemo $T(\eta, n)$ ograničiti sa n^d tj.

$$T(\eta, n) \leq n^d,\ d \in \mathbb{N}$$

Ideja je da se kodira rad mašine η tj. da kodiramo konfiguracije mašine η sa Bulovom formulom. Polazeći da imamo jezik mašine L i neka w bude ulaz za M. Sad polazeći od M i w mi ćemo konstruisati Bulov izraz w_0 takav da je w_0 ispunjiv akko M prihvata w. Tu konstrukciju obavi jedna DTM, čija je složenost polinomska. Ovo je bila ideja, a sad prelazimo na konstrukciju. ($w_0 = \phi(\quad) : w \in L \Leftrightarrow w_0 \in SAT$)

$a \in \Sigma$ - moguće je da se pojavi simbol a u konfiguraciji)

$(q, a) \in Q \times \Sigma$

$\Sigma_1 = \Sigma \cup (Q \times \Sigma)$, neka je $|\Sigma_1| = t = \text{const}$ (ne zavisi od ulaza (formule))

$\$ a_1 a_2 \ldots (q, a_k) a_{k+1} \ldots a_s \$$

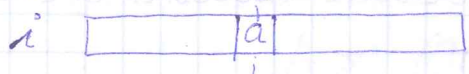
Pitanje: Kolika je maksimalna dužina naše konfiguracije?

Pretpostavićemo da naša mašina ne prelazi na negativan dio naše trake tj. ne prelazi lijevo od polja 0. Pošto je broj koraka $\leq n^d$, a na početku je iznad polja 0, onda se mašina može najviše pomjeriti udesno za n^d polja, pa je maksimalna dužina konfiguracije n^d (a ako bi mašina išla i na negativna polja, onda je max. dužina $2n^d$).

Smatraćemo da su sve konfiguracije dužine n^d.

→ Vrijeme da stignemo do krajnjeg reda.
– kodiraćemo ovu pravougaonu tablicu

U nekom momentu i posmatramo konfiguraciju C_i:



Sa (i, j, C) je u potpunosti određen moment i.

Simbole iz Σ_1 koje redamo, pamtićemo samo preko indexa

$\Sigma_1 = \{a_1, \ldots, a_t\}$: 

Zato uvodimo sledeću oznaku:

Promjenljiva $q(i, j, C)$ ima vrijednost tačno ako je u trenutku i u konfiguraciji C_i na j-toj poziciji upisan l-ti simbol a_l.

$(\underbrace{i, j, C}_{\text{promjenljive}})$

$n^d \; n^d \; t$ – const

$n^d \cdot n^d \cdot t = O(n^{2d})$

Definišemo pravilne konfiguracije ove mašine:

(1) U svakom momentu (u svakoj konfiguraciji) u svakom polju se nalazi tačno jedan simbol.

$\underbrace{\gamma_{ij}}^{1.1} = \bigvee_{l=1}^{k} q(i, j, l) = q(i, j, 1) \vee q(i, j, 2) \vee \ldots \vee q(i, j, t) \quad \forall i, j$

$\underbrace{f_{ij}}^{1.1}$ – formula zavisi od i i j i obezbeđuje da je upisan

tar jedan simbol

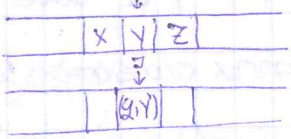
Sad treba obezbijediti da nisu dva upisana na isto mjesto

$l_1 \neq l_2 \quad \varphi^{1.2}_{i,j,c_1,c_2} = \neg q(i,j,c_1) \vee \neg q(i,j,c_2) \quad \forall i, \forall j, \forall c_1, \forall c_2 \; \forall l_1 \neq l_2$

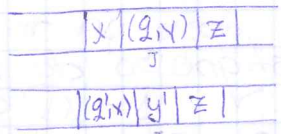
Ovim smo obezbijedili da je upisan tačno jedan

(2) Rad mašine je pravilan tj. da je prelazak iz jedne konfiguracije u drugu je ispravan, odnosno da ne možemo iz jedne konfiguracije preći u neku nepostojeću konfiguraciju

Primjer:



Nedozvoljeno (neregularno)



$P = (q, y, q', y', L) \rightarrow$ dozvoljeno

Situacija na polju j u konfiguraciji C_{i+1} zavisi od situacije na poljima $j-1, j, j+1$ u konfiguraciji $C_i \Rightarrow$ Pa, ova polja označimo kao četvorku (c_1, c_2, c_3, c_4). Treba od ovih četvorki izdvojiti one koje su neregularne tj.

$NR = \{(c_1, c_2, c_3, c_4) \mid c_i \in \Sigma_1 \text{ i } (c_1, c_2, c_3, c_4) \text{ nije regularna}\}$

Ne smije da važi ~~regularna~~ istovremeno (ne smije se istovremeno desiti ova četvorka):

$NR \Leftrightarrow \neg q(i, j-1, c_1) \vee \neg q(i, j, c_2) \vee \neg q(i, j+1, c_3) \vee \neg q(i+1, j, c_4)$

$\equiv \neg(q(i, j-1, c_1) \wedge q(i, j, c_2) \wedge q(i, j+1, c_3) \wedge q(i+1, j, c_4))$

$\forall i \in \{0, n^d - 1\}$ ($n^d - 1$ → jer završna konfiguracija nema sljedeću)

$\forall j \in \{0, n^d - 1\}$

$\forall (c_1, c_2, c_3, c_4) \in NR$

(3) Kodirajmo polaznu konfiguraciju C_0

C_0: $(q, x_1) x_2 \ldots x_s \underbrace{\square \square \ldots \square}_{\text{prazna polja}}$

$x_i \rightarrow c_i \quad (x_i = a_{c_i} \in \Sigma_1$ tj. x_i je c_i-ti simbol $\Sigma_1)$

$(q, x_1) \rightarrow c_1$

$\square \rightarrow c_0$