Dinamičko programiranje

Predmet: Uvod u Algoritme 17 - ESI053
Studijski program: Primenjeno softversko inženjerstvo

DEPARTMAN ZA RAČUNARSTVO I AUTOMATIKU

DEPARTMAN ZA ENERGETIKU, ELEKTRONIKU I KOMUNIKACIJE

Dinamičko programiranje (DP)

- Generalna i moćna tehnika za dizajn algoritama
- Često se koristi
- Pomaže kod tipova problema gde naivna detaljna pretraga dovodi do eksponencijalne složenosti
- DP čini da takvi problemi imaju polinomsku složenost
  - gruba sila se ubrzava preko pamćenja međurešenja

- Osnovna primena je u optimizacionim problemima – gde se vrši izbor između više (mnogo) mogućnosti
  - traženje min / max načega
    - npr. najkraći put
  - pogodno za detaljnu pretragu (EXP problem)

Kako prepoznati DP problem?

1. Problem ima više (mnogo) izbora i svaki izbor ima dodeljen skor (vrednost) tako da se traži jedan izbor sa najboljim skorom (optimalan skor, a od definicije problema to ili minimalan ili maksimalan skor)
2. Broj izbora je suviše velik tako da upotreba grube sile i isprobavanje svih njih nema smisla
   - Broj izbora ima eksponencijalnu složenost
3. Do rešenja se dolazi traženjem optimalnih rešenja za jednostavnije potprobleme
4. Potproblemi se preklapaju
   - Potproblemi imaju svoje potprobleme i isti potproblem se ponavlja kod njihovog rešavanja
   - Rešavanje grubom silom bi iste potprobleme rešavalo mnogo puta

Primena DP

- Treba opisati (osobina 3) kako se optimalno rešenje dobija rešavanjem manjih problema
  - Tipično se upotrebljava rekurzija i definiše se ka manjim problemima - pristup od gore ka dole
  - Direktna implementacija rekurzije dovodi do eksponencijlane složenosti, ali...
  - Rešenja manjih problema se pamte i umesto ponovnog rešavanja preuzima se zapamćeno rešenje
- Način pamćenje manjih rešenja zavisi od pristupa:
  1. Pristup od dole ka gore: prvo se odrede rešenja najmanjih problema, pa se rešavaju veći problemi koristeći rešenja manjih, i tako redom sve do glavnog problema
  2. Pristup od gore ka dole: kreće se od glavnog problema čije rešavanje zahteva rekurzivno izvršavanje opisa i tako se nailazi na manje probleme, a rešenje svakog se zapamti (kada se reši).
     - Kod rešavanja manjeg problema prvo se proveri da li je rešen – zapamćen
     - To pamćenje se naziva **memoizacija** (bez „r“, nije memorizacija!) i omogućava da se svaki manji problem rešava samo jednom, a ne ponovo i ponovo - što pravi EXP složenost

DP i rekurzija

- Rekurzija je upotrebljava u definisanju veza potproblema
- Može se programski implementirati u pristupu od gore ka dole, ali i ne mora ako se primeni pristup od dole ka gore

- Veza problema i potproblema podseća na strategiju „podeli i osvoji“, ali postoje suštinske razlike: strategija „podeli i osvoji“ se sprovodi na disjunktim skupovima podataka, dok kod DP problema imamo preklapanja potproblema.

Ideja DP

- Osnovna ideja DP ima tri koraka:
  1. Definisati potprobleme
  2. Pokazati kako se rešenje zadatog problema može konstruisati polazeći od (rešenih) podproblema - upotrebiti rekurziju
  3. Prepoznati i rešiti osnovne slučajeve

- Kasnije će biti detaljnije razmotreno.

Primer: Fibonačijevi brojevi (1)

- Definicija Fibonačijevih brojeva je:

  $F_1 = F_2 = 1$

  $F_k = F_{k-1} + F_{k-2}, \quad k = 3, 4, 5 \ldots$

- Fibonačijevi brojevi su: $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$
- Problem: naći n-ti Fibonačijev broj

- Ovo nije DP problem jer nema mnogo izbora za traženje F_k, ali se dobro ilustruju neke osobine DP
  - Osnovni problem $F_n = F_{n-1} + F_{n-2}$
  - Svaki potproblem je isti kao i osnovni problem $F_k = F_{k-1} + F_{k-2}$
  - Rekurzija je sadržana u definiciji Fibonačijevog broja
  - Osnovni slučajevi su $F_1 = F_2 = 1$

Fibonačijevi brojevi – naivno rešenje

```
FIB(n)
if n ≤ 2, f = 1
else f = Fib(n-1) + Fib(n-2)   // rekurzija
return f
```

- Eksponencijalno vreme izvršavanja!

$$T(n) = T(n-1) + T(n-2) + O(1) \geq 2T(n-2) = \Theta\left(2^{\frac{n}{2}}\right) = \Theta(\phi^n)$$

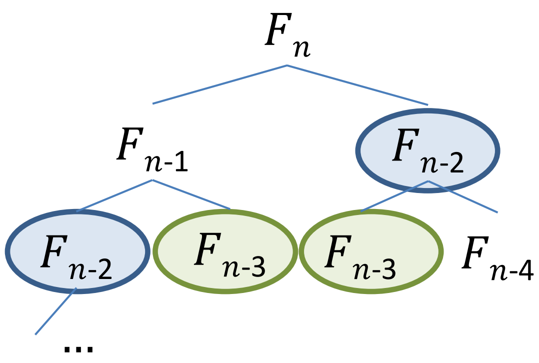
$$T(n) \geq 2T(n-2) \geq 2\big(2T(n-4)\big) \geq 2\Big(2\big(2T(n-6)\big)\Big) \geq \cdots \geq 2^{\frac{n}{2}}\mathrm{c} = \Theta\left(2^{\frac{n}{2}}\right) = \Theta(\phi^n)$$

Fibonačijevi brojevi – rešenje sa memoizacijom

- Memoizacija (engl. *memoization*) – optimizaciona tehnika namenjena za ubrzanje programa gde se rezultat složene funkcije kešira nakon poziva da bi se kasnije ponovo upotrebio.
  - Termin specifičan za računarstvo, nastao od reči memorandum ili memo a znači upamtiti.

```
memo = {}                    // rečnik – za memoizaciju
FIB(n)
  if n in memo, return memo[n]   // keširan Θ(1)
  if n ≤ 2, f = 1
  else f = Fib(n-1) + Fib(n-2)
  memo[n] = f                // upamti
  return f
```

Rešenje sa memoizacijom (2)



- Očito, ranije urađen posao se ponavlja pa se može iskoristiti – ponovo upotrebiti
- $Fib(k)$ - rekurzija se koristi samo za prvo izračunavanje (važi za $\forall k$), tj. ima samo n ne-memo poziva
- memo poziv je "jeftin", $\Theta(1)$
- Vreme izvršavanje:
  - Nerekurzivni deo je $\Theta(1)$, ignorišemo rekurziju
  - $T(n) = n\Theta(1) = \Theta(n)$

Fibonačijevi brojevi – rešenje „od dole ka gore“

```
FIB(n)
m = {}
for k in [1,2,…,n]
    if k ≤ 2, f = 1
    else f = m[k-1] + m[k-2]
    m[k] = f
return m[n]
```

- Isto računanje kao i kod rešenja sa memoizacijom
- Složenost $\Theta(n)$ (analiza je očigledna)
- Praktično je malo brže jer nema rekurzije
- Memorijski prostor se može smanjiti
  - upamtiti samo poslednja 2 broja

DP ≈ rekurzija + memoizacija + „pogađanje“

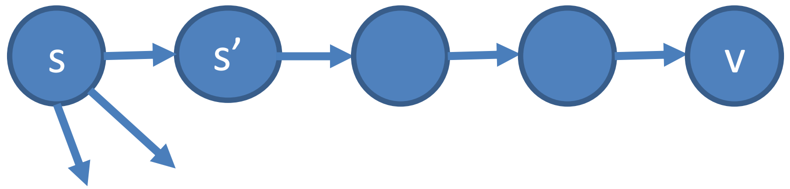
- Kako pristupiti problemu?
- Ako ne znamo odgovor onda pogađamo.
  - Probamo sve moguće pretpostavke i
  - Uzeti jednu, najbolju pretpostavku

- Zapamtiti i ponovo upotrebiti rešenja za potprobleme koji pomažu da se reši osnovni problem
- Vreme izvršavanje = #potproblema * (vreme izvršavanja potproblema)
  - Potproblem se jednom izvršava (računa), a kasnije se koristi rezultat $\Theta(1)$, ignorišu se rekurzije

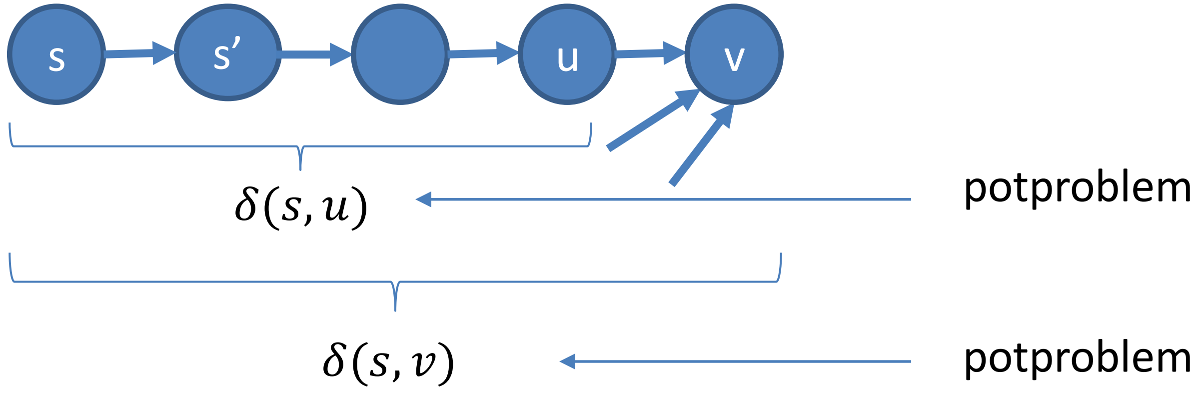
Primer: Najkraći put u grafu

- Problem: odrediti najkraća rastojanja $\delta(s, v)$ u grafu $G = (V, E)$ počev od datog čvora s do ostalih čvorova $v \in V$.

- Rešenje upotrebom DP: krenemo iz „s" i probamo sve putanje



- ili na više načina možemo da završimo u "v", probamo ih sve i izaberemo najbolji.



- Veza potproblema: $\delta(s,v) = \min_{(u,v)\in E}\{\delta(s,u) + w(u,v)\}$

  $\delta(s,s) = 0$
- Broj potproblema je jednak broju čvorova $|V|$
- Vreme potrebno za svaki potproblem je srazmerno broju grana koje završavaju u čvoru $(1 + indegree(v))$.
- Ukupno vreme je $\sum_{v\in V}\left(1 + indegree(v)\right) = \Theta(|V| + |E|)$

- Ovo radi samo u DAG, tj. nije primenljivo na grafove sa ciklusima (zatvorenim putanjama) jer se događa beskonačna rekurzija.
- ...

- Preformulisan problem: odrediti najkraća rastojanja $\delta_k(s, v)$ u grafu upotrebom najviše k grana.

- Sada je: $\delta_k(s, v) = \min_{(u,v)\in E}\{\delta_{k-1}(s, u) + w(u, v)\}$
- Broj potproblema m je jednak je proizvodu broja čvorova $|V|$ i mogućem broju dužina putanja do čvora $\{1,2, \dots, |V| - 1\}$

$$m = \Theta(|V|^2)$$

- Vreme po jednom potproblemu je $t = \frac{\Theta(|V|+|E|)}{|V|}$
- Ukupno vreme: $mt = \Theta(|V|^2 + |E||V|) = \Theta(|E||V|)$

Belman-Ford

DP koraci

Korak:

1. Definisati potprobleme
2. Probati (moguće izbore)
3. Naći odnos potproblema
4. Definisati algoritam
   - Rekurzija + memoizacija, ili
   - DP tabela „od dole ka gore“

   Izbeći ciklične potprobleme!
5. Rešiti originalan problem

Određivanje složenosti:

Prebrojati potprobleme m

Prebrojati moguće izbore

Odrediti vreme potproblema t

Ukupno vreme $= mt$

.

.

.

Dodatno vreme?

Primeri: DP koraci

Primer	Fibonači	Najkraći put
Potproblemi	$F_k, 1 \leq k \leq n$	$\delta_k(s,v), v \in V, 0 \leq k < \lvert V \rvert$
Broj potproblema	n	$\lvert V \rvert^2$
Probati	-	Grane koje ulaze u v
Broj izbora	1	$1 + indegree(v)$
Odnos potproblema	$F_k = F_{k-1} + F_{k-2}$	$\delta_k(s,v) = \min_{(u,v)\in E}\{\delta_{k-1}(s,u) + w(u,v)\}$
Vreme potproblema	$\Theta(1)$	$\Theta(1 + indegree(v))$
Algoritam	for $k = 1,2,\ldots,n$	for $k = 0,1\ldots,\lvert V \rvert - 1$ for $v \in V$
Ukupno vreme	$\Theta(n)$	$\Theta(\lvert E \rvert \lvert V \rvert)$
Originalan problem	F_n	$\delta_{\lvert V \rvert - 1}(s,v), v \in V$
Dodatno vreme	$\Theta(1)$	$\Theta(\lvert V \rvert)$

Primer: Sečenje cevi

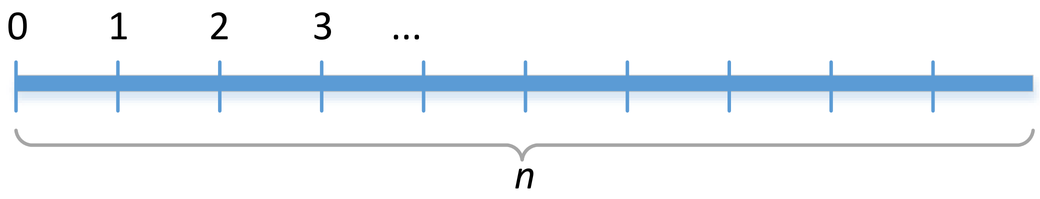
Opis problema: Firma se bavi prodajom parčadi cevi za šta je utvrđen cenovnik

Dužina L_i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Cena c_i	1	5	8	9	10	17	17	20	24	30

Parčadi cevi se prave sečenjem dugačke cevi (dužine n).
Kako iseći tu cev da bi se postigla najveća zarada?

(Radi jednostavnosti dužine su celi brojevi, a cena se može posmatrati kao zarada za jedno parče.
Može postojati više rešenja sa istom zaradom.)

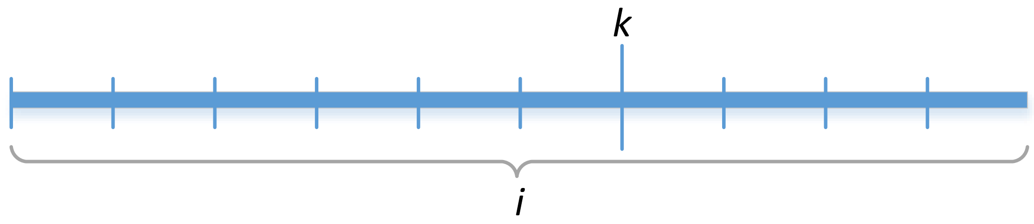
- Cev i mesta gde se može seći
  - Ako se svaka pozicija sečenja predstavi jednim bitom (1=seci, 0=nemoj) onda je rešenje problema reč od n bita.
  - Broj mogućih rešenja je 2^{n-1} što je eksponencijalna složenost



Sečenje cevi – pristup rešenju

- Sečenje će napraviti dve kraće cevi (sem ako se „seče“ na samom početku)
- Kada se sečenje desi na izabranoj poziciji, onda se optimalno rešenje problema svodi na rešavanje dva manja problema (jer su parčadi cevi kraći od početne), što je isti polazni problem ali je dužina drugačija
- Zarada za cev dužine i je:

$$z_i = \max_{k \in 1..i} (z_k + z_{i-k})$$



- Nadalje, ovo se može pojednostaviti ako razmišljamo da mesto reza treba usaglasiti sa tabelom cenovnika tako da se može pojaviti samo na zadatim rastojanjima L_i

$$z_i = \max_{k \in 1..i}(c_k + z_{i-k})$$

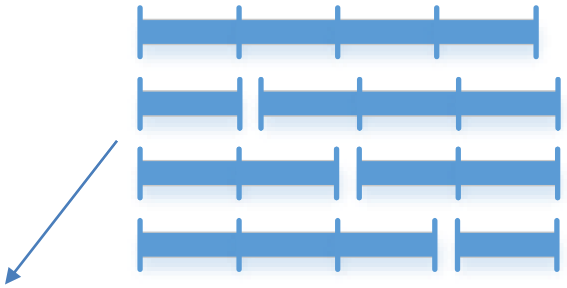
- Sada izračunamo zarade za razne dužine cevi:

$$z_0 = 0$$

$$z_1 = c_1 + z_0 = 1$$

$$z_2 = \max(c_1 + z_1, c_2 + z_0) = \max(2, 5) = 5$$

$$z_3 = \max(c_1 + z_2, c_2 + z_1, c_3 + z_0) = \max(6, 6, 8) = 8$$



$$z_4 = \max(c_1 + z_3, c_2 + z_2, c_3 + z_1, c_4 + z_0) = \max(9, 10, 9, 9) = 10$$

...

Dužina L_i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Cena c_i	0	1	5	8	9	10	17	17	20	24	30
Zarada z_i	**0**	**1**	**5**	**8**	**10**	**13**	**17**	**18**	**22**	**25**	**30**

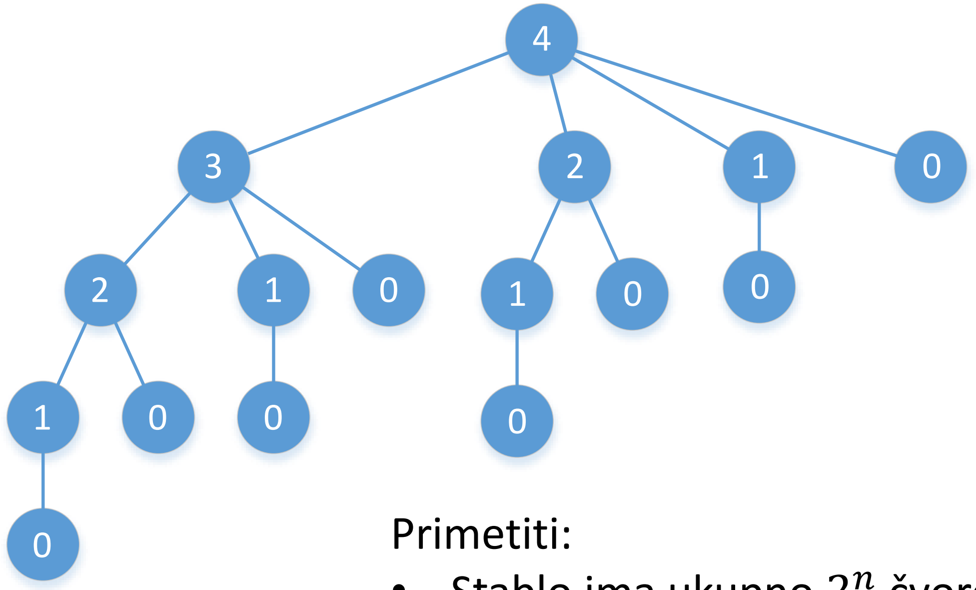
Naivno rešenje

Prihvatljivo je za malo n, ali zbog eksponencijalne složenosti dugo traje.

$$T(n) = 1 + \sum_{i=1}^{n} T(i) = 2^n$$

```
SEČENJE-CEVI-NAIVNO(c, n)
    if n == 0
        return 0
    z = -∞
    for i = 1 to n
        z = max(z, c[i] + SEČENJE-CEVI-NAIVNO(c, n-i))
    return z
```

Stablo rekurzivnih poziva za $n = 4$:



Primetiti:

- Stablo ima ukupno 2^n čvorova, tj. rekurzivnih poziva.
- Stablo ima 2^{n-1} listova, tj. mogućih ishoda sečenja.

Rešenje sa gore ka dole (memoizacija)

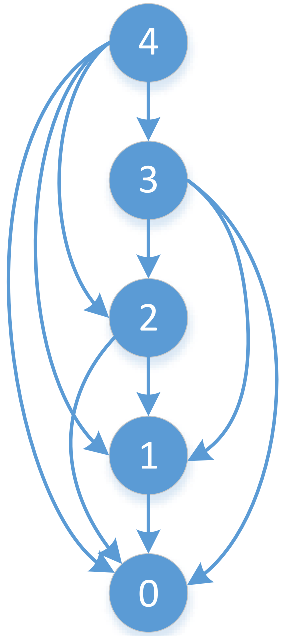
```
SEČENJE-CEVI-ODGOREKADOLE(c, n)
1   new z[0..n] je niz ili rečnik
2   for i = 0 to n
3       z[i] = -∝
4   return SEČENJE-CEVI-MEMO(c, n, z)

SEČENJE-CEVI-MEMO(c, n, z)
1   if z[n] ≥ 0
2       return z[n]
3   if n == 0
4       q = 0
5   else q = -∝
6       for k = 1 to n
7           q = max(q, c[k]+SEČENJE-CEVI-MEMO(c,n-k,z))
8   z[n] = q
9   return q
```

- Složenost je $\Theta(n^2)$
  - Nije tako očigledna zbog rekurzivnih poziva.
  - Petlja u redu 6 i 7 ima rekurzivne pozive koji se ponavljaju za svaku dužinu tačno jednom, tj. n puta.

Rešenje sa dole ka gore

- Složenost je očigledna zbog dve ugnježdene petlje i iznosi $\Theta(n^2)$

- Graf potproblema




```
SEČENJE-CEVI-ODDOLEKAGORE(c, n)
    new z[0..n] je niz ili rečnik
    z[0] = 0
    for i = 1 to n
        q = -∞
        for k = 1 to i
            q = max(q, c[k] + z[i-k])
        z[i] = q
    return z[n]
```

Gde je sečeno?

- Kada se od ažurira maksimalna zarada q onda se zapamti koje dužine je cev k.

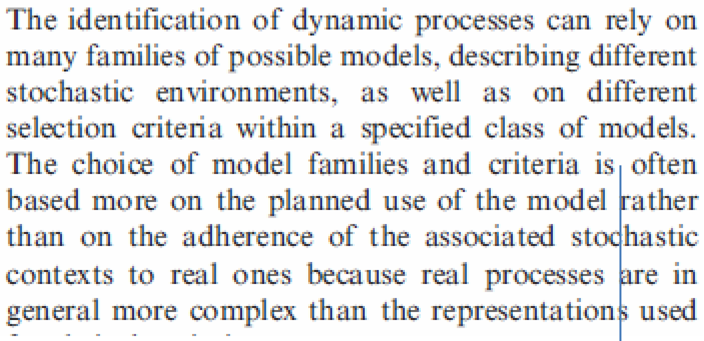
```
SEČENJE-CEVI-ODDOLEKAGORE(c, n)
new z[0..n], s[0..n] su nizovi
z[0] = 0
for i = 1 to n
    q = -∝
    for k = 1 to i
        if q < c[k] + z[i-k]
            q = c[k] + z[i-k]
            s[i] = k
    z[i] = q
return z, s
```

```
ISPIS-SEČENJA(c, n)
(z,s) = SEČENJE-CEVI-ODDOLEKAGORE(c,n)
while n > 0
    Print s[n]
    n = n - s[n]
```

Korak	Sečenje cevi
Potproblemi	Maksimalna zarada za i-tu dužinu cevi
Broj potproblema	$\Theta(n)$, n je ukupna dužina
Probati	Seći na svaku jediničnu dužinu?
Broj izbora	$1..n = \Theta(n)$
Odnos potproblema	$z_i = \max\limits_{k \in 1..i}(c_k + z_{i-k})$ $s[i] = k$ za koje je max (tj. gde je sečeno)
Vreme potproblema	$\Theta(i),\ \ i = 1..n$
Algoritam	Izračunati $z[i],\ \ i = 1..n$
Ukupno vreme	$\Theta(n^2)$
Originalan problem	$z[n]$

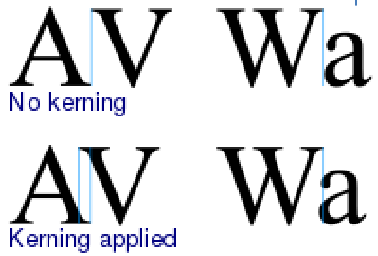
Primer: Poravnavanje teksta

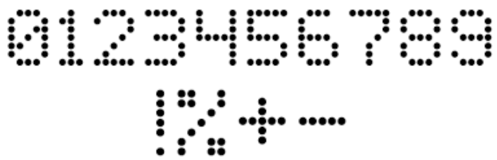
- Problem tekst procesora, poput MS Word-a ili Open Office-a:
  Dati niz reči treba poravnati u stupcu (sa obe strane).


The identification of dynamic processes can rely on many families of possible models, describing different stochastic environments, as well as on different selection criteria within a specified class of models. The choice of model families and criteria is often based more on the planned use of the model rather than on the adherence of the associated stochastic contexts to real ones because real processes are in general more complex than the representations used

Neki tekst ispisan proporcionalnim fontom, gde su širine svih slova jednake.

Neki tekst ispisan neproporcionalnim fontom, gde su širine slova nejednake.





Poravnavanje teksta - rešenje

- Rešenje:
  - Definisati skor $s(i,j)$ na svakom redu koga čine reči od i do $j-1, i<j$

$$s(i,j) = \begin{cases} (\text{širina stupca} - \text{ukupna širina slova})^3 \\ \infty \text{, kada tekst ne staje u red} \end{cases}$$

  - Podeliti tekst tako da se minimizuje ukupan skor (po svim redovima).

Korak	Poravnavanje teksta
Potproblemi	Minimalan skor za sve reči iza i-te
Broj potproblema	$\Theta(n)$, n je ukupan broj reči
Probati	Gde se završava prva linija?
Broj izbora	$n - i = \Theta(n)$
Odnos potproblema	$X[i] = \min(s(i,j) + X[j])$ $p[i] = j$ za koje je min for $j = i + 1, \dots, n$ $X[n] = 0$
Vreme potproblema	$\Theta(n)$
Algoritam	Izračunati $X[i]$ for $i = n, n - 1, \dots, 1, 0$
Ukupno vreme	$\Theta(n^2)$
Originalan problem	$X[0]$

- Gde su počeci redova? (*p - parent pointers*)

$$0 \to p[0] \to p\big[p[0]\big] \to p\Big[p\big[p[0]\big]\Big] \dots$$

Rešenje u Juliji

```
tekst = "Dinamicko programiranje je …"   # prikazan se samo deo teksta

reči = split(tekst, " ")
d = [length(r)+1 for r in reči]     # dužine reči (+1 sa razmakom)

n = length(reči)    # broj reči
s = zeros(n, n)     # skor

const MAXSLOVA = 80
# matrica skorova podstringova sastavljenih od reči
for i = 1:n
    for j = i:n
        brojSlova = sum(d[i:j])     # broj slova od reči i do reči j, zaključno
        if brojSlova > MAXSLOVA
            s[i,j] = Inf
        else
            if j == n
                s[i,j] = 0          # reči iz poslednjeg reda. Skor ==0 jer se poravnava na levo.
            else
                s[i,j] = (MAXSLOVA - brojSlova)^3
            end
        end
    end
end
```

```
…
X = zeros(n)            # skor kada red počinje i-tom reči
p = zeros(Int64,n)      # za i-tu reč indeks prve reči u narednom redu

for i = n:-1:1          # po svim mogućim počecima reda
    minj = n + 1        # gde treba prelomiti posmatrani red
    mins = s[i, n]      # mins je minimalan skor
    for j = i+1 : n     # po svim sufiksima, tj. probamo da prelomimo red iza svake naredne reči
        if s[i, j-1] + X[j] < mins      # minimizacija skora: odnos potproblema
            mins = s[i, j-1] + X[j]
            minj = j
        end
    end
    X[i] = mins
    p[i] = minj
end

# ispis
pr = 1                  # indeks reči na početku reda
while pr <= n
    global pr
    kr = p[pr] - 1
    println( join(reči[pr : kr], ' ') )
    pr = kr + 1
end
```

Dinamicko programiranje je naziv popularne tehnike u programiranju kojom drasticno mozemo smanjiti slozenost algoritma: od eksponencionalne do polinomijalne. Rec programiranje u samom nazivu tehnike se odnosi na tzv. tablicni metod, a ne na samo kucanje kompjuterskog koda. Slicno metodi podeli pa vladaj (eng. divide and conquer), dinamicko programiranje resavanje jednog problema svodi na resavanje podproblema. Za ovakve probleme se kaze da imaju optimalnu strukturu (eng. optimal substructure). Podeli pa vladaj algoritmi vrse particiju glavnog problema na nezavisne podprobleme. Zatim nastupa rekurzivno resavanje podproblema, kako bi se njihovim spajanjem dobilo resenje polaznog problema. Algoritam koji je dobar predstavnik ove klase jeste sortiranje ucesljavanjem (eng. merge sort), algoritam za sortiranje niza. I tako dalje...

Rezultat

Naše rešenje

Dinamicko programiranje je naziv popularne tehnike u programiranju kojom drasticno mozemo smanjiti slozenost algoritma: od eksponencionalne do polinomijalne. Rec programiranje u samom nazivu tehnike se odnosi na tzv. tablicni metod, a ne na samo kucanje kompjuterskog koda. Slicno metodi podeli pa vladaj (eng. divide and conquer), dinamicko programiranje resavanje jednog problema svodi na resavanje podproblema. Za ovakve probleme se kaze da imaju optimalnu strukturu (eng. optimal substructure). Podeli pa vladaj algoritmi vrse particiju glavnog problema na nezavisne podprobleme. Zatim nastupa rekurzivno resavanje podproblema, kako bi se njihovim spajanjem dobilo resenje polaznog problema. Algoritam koji je dobar predstavnik ove klase jeste sortiranje ucesljavanjem (eng. merge sort), algoritam za sortiranje niza. I tako dalje...

Word rešenje

Dinamicko programiranje je naziv popularne tehnike u programiranju kojom drasticno mozemo smanjiti slozenost algoritma: od eksponencionalne do polinomijalne. Rec programiranje u samom nazivu tehnike se odnosi na tzv. tablicni metod, a ne na samo kucanje kompjuterskog koda. Slicno metodi podeli pa vladaj (eng. divide and conquer), dinamicko programiranje resavanje jednog problema svodi na resavanje podproblema. Za ovakve probleme se kaze da imaju optimalnu strukturu (eng. optimal substructure). Podeli pa vladaj algoritmi vrse particiju glavnog problema na nezavisne podprobleme. Zatim nastupa rekurzivno resavanje podproblema, kako bi se njihovim spajanjem dobilo resenje polaznog problema. Algoritam koji je dobar predstavnik ove klase jeste sortiranje ucesljavanjem (eng. merge sort), algoritam za sortiranje niza. I tako dalje...

Primer: Zagrade

- Zadatak: odrediti optimalno izračunavanje asocijativnih izraza upotrebom zagrada.

- Npr. množenje matrica: $A_0 A_1 A_2 \dots A_{n-1}$
  - Izraz: $(A_{5\times1} B_{1\times5}) C_{5\times1}$ ima 50 množenja
  - Izraz: $A_{5\times1} (B_{1\times5} C_{5\times1})$ ima 10 množenja
- Izraz $B_{m\times r} \cdot C_{r\times k}$ ima $m \cdot r \cdot k$ operacija množenja.

- Gde ubaciti zagrade?
- Broj mogućih rešenja je $\Omega(2^n)$ (tačnije $\Omega(4^n/n^{3/2})$) tako da je rešenje grubom silom loša strategija!
- Ovde se ne rešava množenje matrica nego se traži rešenje za optimalan način (redosled) izračunavanja

Primer: zagrade kod množenja matrica

- Ideja: razmatramo poslednje množenje

$$(A_1 \dots A_{i-1})(A_i \dots A_n)$$

- i jedno množenje pre njega

$$(A_1 \dots A_{i-1})\big(A_i \dots A_j\big)\big(A_{j+1} \dots A_n\big)$$

- Potproblem je optimalno množenje $A_i \dots A_j$, gde se posmatra deo originalnog problema kao substring

Korak	Množenje matrica
Potproblemi	$A_i \ldots A_j, \qquad 1 \le i \le j \le n$
Broj potproblema	$\Theta(n^2)$
Probati	Gde postaviti zagradu u potproblem? $(A_i \ldots A_k) \cdot (A_{k+1} \ldots A_j)$
Broj izbora	$j - i = \Theta(n)$
Odnos potproblema	$m(i,j) = \min(m(i,k) + m(k+1,j) + bom)$ for $k = i, i+1, \ldots, j-1$ $m(i,i) = 0$
Vreme potproblema	$\Theta(n)$
Algoritam	Izračunati $m(i,j)$ for $i = 1, \ldots, n-1$ for $j = 2, \ldots, n$
Ukupno vreme	$\Theta(n)\Theta(n^2) = \Theta(n^3)$
Originalan problem	$m(1,n)$

bom – broj operacija množenja $B \cdot C, B = A_i \ldots A_k, C = A_{k+1} \ldots A_j$

Rešenje „od dole ka gore“

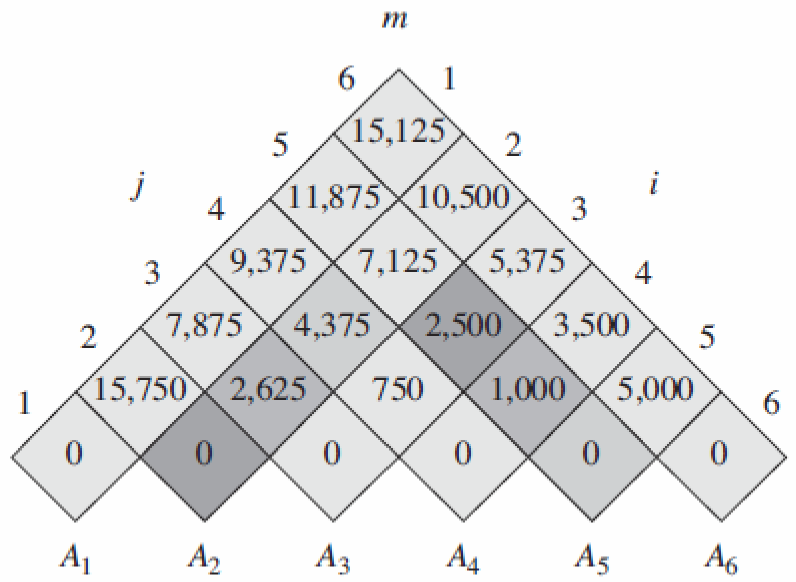
- Ulaz: niz p sadrži dimenzije matrica. Matrica $A_i, i = 1, \ldots, n$ ima dimenzije $p_{i-1} \times p_i$

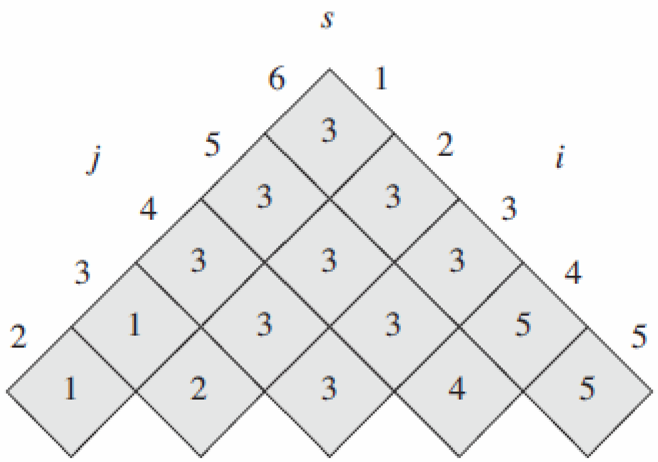
```
RedosledMnoženjaMatrica(p)
n=p.length-1
m[1..n,1..n], s[1..n-1,2..n]  // tabele
for i=1 to n
   m[i,i]=0
for l=2 to n              // dužina lanca činioca
   for i=1 to n-l+1
      j = i+l-1
      m[i,j] = ∞
      for k=i to j-1
         q = m[i,k] + m[k+1,j] + p_{i-1}p_k p_j
         if q < m[i,j]
            m[i,j] = q
            s[i,j] = k   // gde je množenje
return m, s
```

Primer redosleda množenja matrica

A_1	A_2	A_3	A_4	A_5	A_6
30×35	35×15	15×5	5×10	10×20	20×25

$$
\begin{aligned}
m[2,5] &= \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 &= 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 &= 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 &= 11{,}375 \end{cases} \\
&= 7125\,.
\end{aligned}
$$





Štampanje rezultata

- Kada se pozove ŠTAMPANJEZAGRADA(s,1,6) dobije se:

$$\Big(\big(A_1(A_2A_3)\big)\big((A_4A_5)A_6\big)\Big)$$

```
ŠTAMPANJEZAGRADA(s,i,j)
  if i==j
     Print “A”_i
  else Print “(“
     ŠTAMPANJEZAGRADA(s,i,s[i,j])
     ŠTAMPANJEZAGRADA(s,s[i,j]+1,j)
     Print “)“
```

Julija ...

```
p = [30,35,15,5,10,20,25]

n = length(p) - 1
m = zeros(Int64,n,n)
s = zeros(Int16,n,n)

for l = 2 : n
    for i = 1 : (n - l + 1)
        j = i + l - 1
        m[i,j] = typemax(Int64)     # Inf je float
        for k = i : j-1
            q = m[i,k] + m[k+1,j] + p[i]*p[k+1]*p[j+1]
            if q < m[i,j]
                m[i,j] = q
                s[i,j] = k      # gde je množenje
            end
        end
    end
end
```

```
m =
     0   15750    7875    9375   11875   15125
     0       0    2625    4375    7125   10500
     0       0       0     750    2500    5375
     0       0       0       0    1000    3500
     0       0       0       0       0    5000
     0       0       0       0       0       0
```

```
s =
   0   1   1   3   3   3
   0   0   2   3   3   3
   0   0   0   3   3   3
   0   0   0   0   4   5
   0   0   0   0   0   5
   0   0   0   0   0   0
```

Popunjavanje „m“

```
m =
        0   15750       0       0       0       0
        0       0    2625       0       0       0
        0       0       0     750       0       0
        0       0       0       0    1000       0
        0       0       0       0       0    5000
        0       0       0       0       0       0
m =
        0   15750    7875       0       0       0
        0       0    2625    4375       0       0
        0       0       0     750    2500       0
        0       0       0       0    1000    3500
        0       0       0       0       0    5000
        0       0       0       0       0       0
m =
        0   15750    7875    9375       0       0
        0       0    2625    4375    7125       0
        0       0       0     750    2500    5375
        0       0       0       0    1000    3500
        0       0       0       0       0    5000
        0       0       0       0       0       0
m =
        0   15750    7875    9375   11875       0
        0       0    2625    4375    7125   10500
        0       0       0     750    2500    5375
        0       0       0       0    1000    3500
        0       0       0       0       0    5000
        0       0       0       0       0       0
m =
        0   15750    7875    9375   11875   15125
        0       0    2625    4375    7125   10500
        0       0       0     750    2500    5375
        0       0       0       0    1000    3500
        0       0       0       0       0    5000
        0       0       0       0       0       0
```