Teorija Algoritama

Dinamičko programiranje i pohlepni algoritmi

Dinamičko programiranje

Dinamičko programiranje

- Rešavanje (uglavnom optimizacionih) problema:
  - Kombinovanjem optimalnih rešenja potproblema (*optimal substructure*);
  - Korišćenjem činjenice da se potproblemi, u određenoj meri, poklapaju (*overlapping sub-problems*).
- Za razliku od *divide-and-conquer* pristupa, koji je u osnovi *top-down* (kreće se od početnog problema, koji se deli na manje potprobleme, čijim se rešavanjem rešava i početni problem), dinamičko programiranje koristi *bottom-up* pristup (polazi se od rešavanja manjih problema, čiji rezultati će doprineti rešavanju velikog problema).
- Osnovni zadatak - kreiranje strukture optimalnog rešenja.
- Pristup memoizacije - umesto rekurzivnih poziva, rešenja potproblema se memorišu u nekoj strukturi podataka i po potrebi koriste.

Primer 1

Fibonačijevi brojevi - *F(0) = 1, F(1) = 1, F(n) = F(n-1) + F(n-2)*

Rekurzivni pristup - V: $O(2^n)$ P: $O(n)$

```
function fib_rec(n):

    if(n <= 1):
        return 1

    return fib_rec(n-1)
           + fib_rec(n-2)
```

Pristup memoizacije - V: $O(n)$ $\Omega(1)$, P: $O(n)$

```
def fib_memo(n, memo):

    if(memo[n].computed)
        return memo[n]

    if(n <= 1)
        return 1
    memo[n] = fib_memo(n-1, memo)
           + fib_memo(n-2, memo)

    memo.computed = n

    return memo[n]
```

Primer 2

- Izračunavanje vrednosti binomnog koeficijenta ***n*** i ***k*** (***n*** nad ***k***): $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$
  - Na koliko načina iz skupa od ***n*** elemenata se može izabrati ***k*** elemenata.
- Optimalna podstruktura:
  - *C(n, k) = C(n-1, k-1) + C(n-1, k)*;
  - *C(n, 0) = C(n, n) = 1*.
- Rekurzivni pristup:

```
function bin_rec(n, k)
    if(k == 0 or k == n) return 1
    if(k > n) return 0
    return bin_rec(n-1, k-1) + bin_rec(n-1, k)
```

- Vremenska složenost je jednaka upravo rezultatu koji tražimo: $O\left(\frac{2^n}{\sqrt{(n)}}\right)$
  - Koliko je ovo neoptimalno, možemo videti na primeru *n = 30* i *k = 15*, gde će se funkcija izvršiti preko 150 miliona puta.
- Prostorna složenost je linearna (maksimalna dubina rekurzivnog poziva)

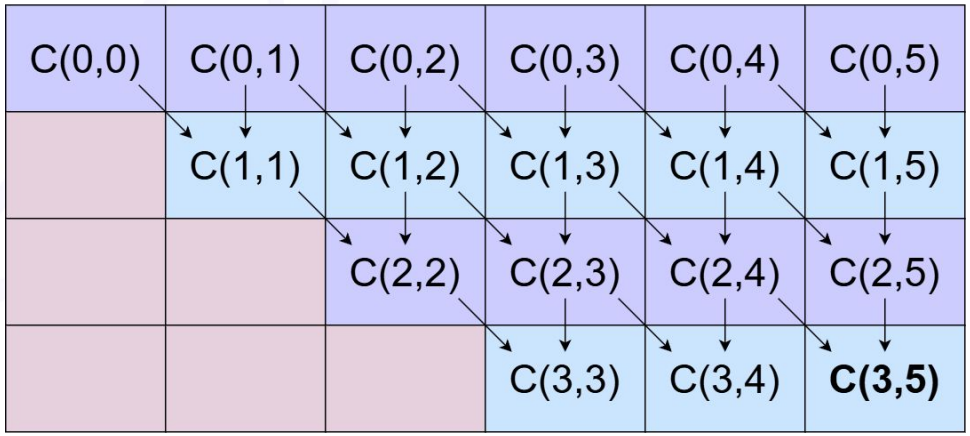
Primer 2

- Pristup memoizacije:

```
def bin_memo(n, k, memo)
    if(memo[n][k].computed)
        return memo[n][k]
    if(k == 0 or k == n) return 1
    if(k > n) return 0
    memo[n][k] = bin_mem(n-1, k-1, memo)
               + bin_mem(n-1, k, memo)
    memo.computedX = n
    memo.computedY = k
    return memo[n][k]
```

- Vremenska složenost - $O(n \cdot k)$.
- Prostorna složenost - $O(n \cdot k)$.

Paskalov trougao

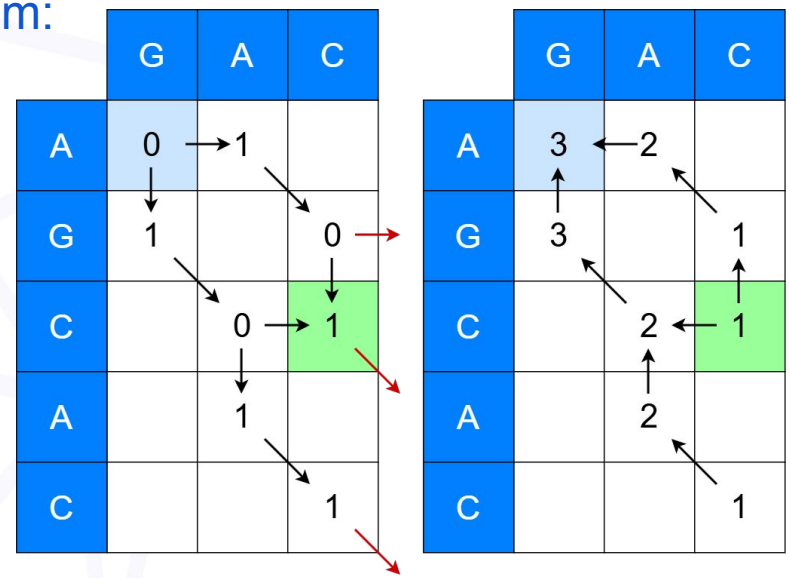


Primer 3

- Najduži zajednički podniz (LCS):
  - Za dva stringa (niza karaktera) pronaći dužinu najdužeg zajedničkog podniza (ne nužno uzastponih) karaktera;
  - Primer: “**S**lik**a**” i “**Sa**t” najduži podniz je “Sa”.
- Problem:
  - Intuitivni pristup je da se redom proveravaju karakteri dva stringa - A = “Slika” i B = “Sat”.
  - Pri prvoj iteraciji zaključujemo da je *A[0] == B[0]*, dakle pronašli smo podniz dužine 1.
  - U narednoj iteraciji primećujemo da *A[1] != B[1]*, i sada imamo dilemu:
    - Nastaviti pretragu kroz string A, a stati na 2. karakteru u stringu B;
    - Stati na 2. karakteru u stringu A, a nastaviti kroz string B.
  - Nije dovoljno odlučiti se za jednu od dve opcije, već se moraju obraditi obe i zadržati opcija koja je pronašla duži podniz.
  - Rekurzivnim pristupom dobijamo eksponencijalnu složenost, što definitivno nije dobra opcija.

Primer 3

- Rešavanje problema LCS sa memoizacijom:
  - Poredimo stringove “AGCAC” i “GAC”




```
function LCS(s1, i, s2, j, memo)
    if(i == s1.size or j == s2.size)
        return 0

    if(memo[i][j].computed)
        return memo[i][j]

    if(s[i] == s[j])
        memo[i][j] = 1 + LCS(s1, i+1, s2, j+1, memo)
    else
        memo[i][j] = max(LCS(s1, i+1, s2, j, memo),
                         LCS(s1, i, s2, j+1, memo))

    return memo[i][j]
```

Primer 4

- Da li je moguće zadati niz prirodnih brojeva A podeliti na dva disjunktna podniza jednakih suma? (Partition Equal Subset Sum - PESS)
- “Naivni” pristup - određivanjem sume svih podnizova niza A, ispitati da li postoji podniz sa sumom S/2 (gde je S suma svih elemenata niza A).
- Mogući pristupi:
  - Rekurzija;
  - Memoizacija;
  - Tabulacija.
- Koja je složenost pristupa sa rekurzijom?
- Koja je složenost pristupa sa memoizacijom?

Primer 4 - Rekurzija

- Ideja:
  - Prolaziti redom kroz elemente niza;
  - Imamo opciju uračunati ih u trenutnu sumu ili ne uračunati ih;
  - Ovaj princip nam stvara rekurziju koja se u svakom koraku deli na dva podstabla rekurzije.

```
function PESS_rec(arr, i, sum, target)
    if(sum == target) return true
    if(i == arr.size) return false
    if(sum > target) return false # Radi za N, ne radi Z
    excl = PESS_rec(i+1, sum, arr, target)
    incl = PESS_rec(i+1, sum + arr[i], arr, target)
    return excl || incl
```

- Vremenska složenost - $O(2^n)$
- Prostorna složenost - $O(n)$

Primer 4 - Memoizacija

- Ideja:
  - Dosta slično pristupu sa običnom rekurzijom;
  - U ovom pristupu pamtimo da li smo već bili na ovom indeksu sa ovom sumom.

```
function PESS_memo(arr, i, sum, target, memo)
    if(sum == target) return true
    if(i == arr.size) return false
    if(current > target) return false
    if(memo[i][sum].computed) return memo[i][sum]
    memo[i][sum] = PESS_rec(i+1, sum, arr, target)
                || PESS_rec(i+1, sum + arr[i], arr, target)
    return memo[i][sum]
```

- Vremenska složenost - $O(n \cdot target)$
- Prostorna složenost - $O(n \cdot target)$

Primer 4 - Tabulacija

- Ideja:
  - Proveravamo za svaki broj [0, *target*], da li je moguće izračunati ga brojevima iz niza;
  - Služimo se jednostavnom matematikom - Ako imamo u polaznom skupu broj X, a proveravamo da li je moguća suma Y, ona je moguća samo, ako je moguća i suma X-Y;
  - Radi samo za prirodne brojeve.

```
function PESS_tab(arr, target)
    dp[target + 1] = {0}
    dp[0] = true # Suma 0 je uvek moguća
    for i = 0 to arr.size - 1
        for s = target down to arr[i]
            dp[s] = dp[s] or dp[s - arr[i]]

    return dp[target]
```

- Vremenska složenost: $O(n \cdot target)$
- Prostorna složenost: $O(target)$

Tabulacija vs memoizacija

- Tabulacija - *bottom-up* pristup:
  - Iterativna implementacija;
  - Niz/matrica se popunjava u potpunosti;
  - Poslednji element niza/matrice predstavlja rešenje;
- Memoizacija - *top-down* pristup:
  - Rekurzivna implementacija;
  - Niz/matrica se popunjava po potrebi;
  - Rezultat svakog rekurzivnog poziva se memoriše i po potrebi kansije koristi.

Primer 5

- Za zadati niz različitih prirodnih brojeva A, pronaći dužinu najvećeg podniza međusobno deljivih parova.
- Primer: A = {18, 1, 3, 6, 13, 17}, D = {1, 3, 6, 18}, Rešenje - 4.
- Rešavanje ovog problema oslanja se na **tranzitivnost** - ako su X i Y međusobno deljivi i Y i Z su međusobno deljivi, onda su i X i Z međusobno deljivi.
- Koraci u rešavanju problema:
  - Sortirati niz - na ovaj način znamo da svaki element može biti deljiv isključivo sa elementima pre njega;
  - Formirati niz *D* koji je iste dužine kao i niz *A,* gde *D[i]* predstavlja dužinu najvećeg podniza koji se završava tačno sa elementom *A[i]*;
  - Inicijalne vrednosti niza *D* su jedinice, jer je svaki element sigurno deljiv sa samim sobom;
  - Svaki element poredimo sa svim prethodnim, ukoliko je deljiv, preuzima njegovu *D* vrednost uvećanu za jedan. (`if(A[i] % A[j]) D[i] = MAX(D[i], D[j] + 1)`);
  - Rešenje je najveći element niza D.

Zadatak 1

- Najjeftinija putanja:
  - U zadatoj matrici M elementi predstavljaju cenu prolaska kroz to polje;
  - Cene su strogo pozitivne vrednosti;
  - Cilj je iz gornjeg levog polja doći do donjeg desnog uz minimalnu cenu;
  - Iz polja (i, j) moguće je pomeriti se u:
    - (i + 1, j) - desno;
    - (i, j +1) - dole;
    - (i + 1, j + 1) - dijagonalno dole desno;
    - Nije moguće pomeriti se u nepostojaće polje (ispasti van matrice).

Zadatak 2

- *String edit distance* - za dva zadata stringa *str1* i *str2* odrediti minimalan broj operacija (dodavanje, brisanje ili izmena) potrebnih da bi se od *str1* napravio *str2*.

Zadatak 3

- Minimalan broj skokova:
  - Dat je niz celih brojeva (strogo pozitivnih) koji predstavljaju maksimalnu dužinu skoka (unapred) sa tog elementa.
  - Odrediti minimalan broj skokova potrebnih da bi se sa prvog mesta došlo do poslednjeg.
- Primer:
  - A = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9};
  - A = {**1**, **3**, 5, **8**, 9, 2, 6, 7, 6, 8, **9**};
  - Rešenje: 3 skoka.

Zadatak 4

- Za *N* kvadrova, koji su dimenzija $\{x_i, y_i, z_i\}$ pronaći visinu najviše kule dobijene slaganjem kvadrova jednih na druge.
- Ograničenje: stranice kvadrova koje se dodiruju, moraju biti takve da stranica gornjeg kvadra mora biti strogo manja od stranice donje u obe dimenzije.
- Kvadrovi se ne mogu rotirati, tj. stranica određena dimanzijama x i y je uvek okrenuta nagore (nadole).

Pohlepni (*Greedy*) algoritmi

Pohlepna strategija

- Rešavanje optimizacionog problema korak po korak, preduzimanjem najpogodnijeg koraka u datoj situaciji, odnosno koraka koji donosi najviše koristi.
- Često se pohlepni algoritmi definišu rekurzivno, a implementiraju iterativno.
- Uglavnom jednostavna implementacija
- Većća vremenska efikasnost od ostalih strategija rešavanja optimizacionih problema.
- Nekad su netačni, tj. ne daju globalno optimalno rešenje, pošto preduzimanje najpogodnijeg koraka u nekom trenutku ne mora značiti da to nužno vodi optimalnom rešenju.

Primer 1: Odabir aktivnosti

- Problem:
  - Dat je niz aktivnosti u vidu parova koji predstavljaju početak i kraj izvršenja aktivnosti.
  - Cilj je odrediti maksimalni broj aktivnosti, tako da se vremena izvršavanja ne poklapaju.
- Primer:
  - [(0, 2), (2, 6), (1, 4), (5, 7), (3, 4), (7, 9)]
- Koraci pri rešavanju:
  - Sortirati niz aktivnosti po vremenu završetka;
  - Izabrati prvu aktivnost;
  - Ponavljati dok ima aktivnosti u nizu:
    - Ako je vreme početka trentne aktivnosti, veće ili jednako od kraja prethodno izabrane aktivnosti - odabrati trenutnu aktivnost.
- Rešenje primera:
  - [(0, 2), (2, 6), (1, 4), (5, 7), (3, 4), (7, 9)];
  - Sortiran niz: [(0, 2), (1, 4), (3, 4), (2, 6), (5, 7), (7, 9)];
  - Odabrane aktivnosti: [**(0, 2)**, (1, 4), **(3, 4)**, (2, 6), **(5, 7)**, **(7, 9)**]

Primer 2: Egipatski razlomci

- Problem:
  - Razlomak (između 0 i 1) predstaviti kao zbir jedinstvenih jediničnih razlomak (1/x);
- Primeri:
  - 6/14 = 1/3 + 1/11 + 1/231;
  - 12/13 = 1/2 + 1/3 + 1/12 + 1/156;
  - 5/121 = 1/33 + 1/121 + 1/363.
- Koraci pri rešavanju:
  - Pronaći najveći jedinični razlomak koji je manji ili jednak od početne vrednosti;
  - Sačuvati ga i oduzeti njegovu vrednost od početne vrednosti;
  - Ponavljati postupak dok početna vrednosti ne postane 0.
- Rešenje primera 6/14:
  - 1/2, **1/3** > 6/14 - 1/3 = 2/21;
  - 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, **1/11** > 2/21 - 1/11 = 1/231
  - 1/12, 1/13, ..., 1/230, **1/231** > 1/231 - 1/231 = **0**

Primer 3: Raspoređivanje poslova

- Problem:
  - Dat je niz poslova i za svaki posao data je njegova vrednost i rok izvršenja;
  - Trajanje svakog posla je jednako i iznosi 1;
  - Potrebno je odabrati poslove tako da se maksimizuje vrednost, tako da se izvršenje poslova ne preklapa.
- Primer:
  - Poslovi (oznaka, rok vrednost): (a, 2, 100), (b, 1, 19), (c, 2, 27), (d, 1, 25), (e, 3, 15);
  - Maksimalna vrednost se dobija za odabir poslova: [c, a, e].
- Koraci pri rešavanju:
  - Sortirati poslove po vrednosti opadajući;
  - Izabrati prvi posao;
  - Iterirati dalje kroz poslove i izabrati svaki koji se ne preklapa sa već izabranim poslovima.
- Rešenje primera:
  - Sortirati poslove: (a, 2, 100), (c, 2, 27), (d, 1, 25), (b, 1, 19), (e, 3, 15)
  - Odabrati posao ***a***, nakon njega možemo izabrati i posao ***c***, jer im je rok izvršenja do 2 i stižemo u tom periodu da odradimo oba posla. Poslovi ***d*** i ***b*** ne stižu da se odrade, ali posao ***e*** stiže.

Primer 4: Policajci i lopovi

- Problem:
  - Dat je niz karaktera dužine ***n*** sa mogućim vrednostima: ***'P'*** i ***'L'***;
  - Jedan policajac može da uhvati najviše jednog lopova;
  - Policajac može uhvatiti lopova samo ako je udaljen od njega najviše ***k*** koraka (mesta);
  - Koliko lopova može maksimalno biti uhvaćeno.
- Primeri:
  - {'P', 'L', 'P', 'L', 'L', 'P'}, k = 1 - Odgovor 3;
  - {'L', 'L', 'P', 'L', 'P', 'P'}, k = 2 - Odgovor 2.
- Koraci pri rešavanju:
  - Svaki policajac hvata najbližeg/najdaljeg lopova - netačno;
  - Kreće se od najnižih indeksa za policajce (p) i lopove (l);
  - Ukoliko je hvatanje moguće, uvećava se broj uhvaćenih lopova i prelazi se na sledeće indekse;
  - Ukoliko hvatanje nije moguće, manji od indeksa (p ili l) se uvećava i ponavlja prethodni korak.
- Rešenje primera {'L', 'L', 'P', 'L', 'P', 'P'}, k = 2:
  - {**'L'**, 'L', **'P'**, 'L', 'P', 'P'} - moguće hvatanje;
  - {'L', **'L'**, 'P', 'L', **'P'**, 'P'} - hvatanje nije moguće - prelazimo na sledećeg lopova;
  - {'L', 'L', 'P', **'L'**, **'P'**, 'P'} - moguće hvatanje;
  - Prethodno uhvaćeni lopov je ujedno bio i poslednji u redu, tako da je odgovor 2.

Primer 5: Miševi u rupama

- Problem:
  - Dato je ***N*** miševa i ***N*** rupa, svaki miš i svaka rupa se nalaze na nekoj poziciji na celobrojnoj pravoj liniji;
  - Za 1 minut miš može da se premesti na poziciju +/- 1 u odnosu na trenutnu;
  - Rasporediti miševe u rupe tako da poslednji miš stigne u svoju rupu u najkraćem vrmenu.
- Primeri:
  - M = {4, -4, 2}, R = {4, 0, 5} - Odgovor 4;
  - M = {-10, -79, -79, 67, 93, -85, -28, -94}, R = {-2, 9, 69, 25, -31, 23, 50, 78} - Odgovor 102.
- Koraci pri rešavanju:
  - Sortirati oba niza;
  - Pronaći razlike korespodentnih elemenata;
  - Najveća razlika je rešenje.
- Rešenje primera:
  - M = {-4, 2, 4}, R = {0, 4, 5}, diff = {**4**, 2, 1}.
- Pokušati promeniti raspored elemenata, pa zatim izračunati razlike:
  - Da li je moguće postići bolje vreme?

Još o pohlepnom pristupu

- U određenim slučajevima izbor trenutno najpogodnijeg koraka ne vodi optimalnom rešenju problema. (primer na sledećem slajdu)
- Često su dobra aproksimacija za NP optimizacione probleme.
  - Primer: Zadatak 4
- Kada pohlepni pristup ne donosi dovoljno dobro rešenje, preporčljivo je problem rešavati dinamičkim programiranjem.

Primer netačnosti pohlepnog pristupa

Vizuelizovana rešenja Zadatka 1 korišćenjem dinamičkog i pohlepnog pristupa:

Dinamički pristup

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

Rešenje: 56

Pohlepni pristup (u slučaju jednakih vrednosti uvek bira redom DESNO, DIJAGONALNO, DOLE)

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

Rešenje: 66