

Matematički fakultet  
Univerzitet u Beogradu

---

# REŠAVANJE PROBLEMA MAKSIMALNOG RANCA

---

Seminarski rad u okviru kursa  
Računarska inteligencija

Mentor:  
prof. dr Aleksandar Kartelj

Studenti:  
Miloš Krsmanović,  
263/2015

Danilo Vučković  
87/2015

Januar 2019. godine

## Sadržaj

Apstrakt.....	3
Opis osnovnog problema maskimalnog ranca.....	3
Motivacija za rešavanje problema maksimalnog ranca.....	4
Formalni opis problema .....	4
Poboljšanje algoritma za rešavanje jednodimenzionog problema ranca.....	5
Dvodimenzionalni problem maksimalnog ranca .....	6
Poređenje rada prethodna dva pristupa .....	8
Zaključak.....	9
Korišćena literatura.....	10

## Apstrakt

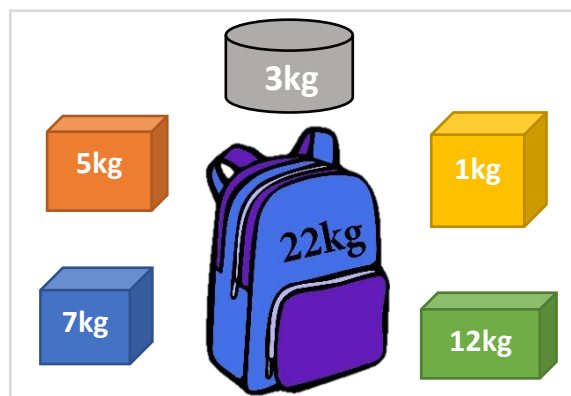
Na samom početku, kao uvod, biće dat koncept problema jedodimenzionalnog maksimalnog ranca. Ovaj problem poslužiće kao dobra osnova za upoznavanje sa relativno kompleksnijim, ali višestruko važnijim, dvodimezionalnim problemom ranca gde će se pored veličine, posmatrati i vrednost koju predmet ima. Za oba pomenuta problema biće navedeni algoritmi koji ih rešavaju, kao i njihova implementacija u programskom jeziku C. Problem će biti rešen naivnom metodom, kao i unapređenim pristupom koji koristi dinamičko programiranje. Takođe, biće predstavljen rezultat izvršavanja kao i vreme koje je bilo potrebno ovim pristupima u rešavanju realnih problema.

Poseban akcenat će biti stavljen na unapređivanju algoritma koji se koristi prilikom rešavanja ovog problema. U radu će biti reči i o tome kako je ovaj problem našao svoju primenu u mnogim sferama industrije i proizvodnje, ali i u svakodnevnom životu.

## Opis osnovnog problema maskimalnog ranca

Početni problem kojim ćemo se baviti je problem gde je predmet opisan samo svojom veličinom.

Ovaj problem ima svoju primenu u mnogim sferama svakodnevnog života. Umesto ranca – problem se može posmatrati i kao popunjavanje teretnjaka, magacina ili silicijumskog čipa na što bolji način, odnosno sa što manjim gubicima prostora. Mogu to biti i druge situacije iz svakodnevnog života što će biti opisano u narednom odeljku.



*Slika 1 Grafički predstavljen problem*

Ukratko, pored predmeta i njihovih težina dat je i kapacitet ranca koji ne sme biti pređen. Potrebno je napuniti ranac stvarima ali tako da prostor bude maksimalno iskorišćen, odnosno da bude što manje praznog prostora. Iako deluje jednostavno, ovaj problem je veoma složen za veće skupove predmeta gde postoji veliki broj mogućnosti, a on tek predstavlja uvod u višedimenzionalni problem ranca.

## Motivacija za rešavanje problema maksimalnog ranca

Potreba za algoritmom koji rešava ovaj problem postoji u mnogim aspektima svakodnevnog života. Veoma slikovit primer može biti sledeći: za reklamu određenog proizvoda izdvojeno je 5 miliona dinara. Potrebno je izabrati najbolji način oglašavanja gde su bitne stavke - cena takvog vida oglašavanja i broj ljudi do kojih reklama stiže.

Opcije koje postoje su sledeće:

- *Sportsko takmičenje*: 3 miliona dinara za oko 80 miliona ljudi
- *Radio kampanja*: 800.000 dinara za oko 20 miliona slušalaca
- *TV reklama*: 500.000 dinara za oko 22 miliona gledalaca
- *Oglas u najprodavanijim novinama*: 2 miliona dinara za približno 75 miliona ljudi
- *Internet kampanja*: košta 600.000 dinara i očekuje se da je vidi 10 miliona ljudi

Koji je najpogodniji, odnosno optimalni način oglašavanja koji treba izabrati?

U ovakvim, ali i mnogim drugim situacijama, problem ranca se može u potpunosti primeniti i dati optimalno rešenje. [1]

## Formalni opis problema

Dat je prirodan broj  $K$  koji predstavlja kapacitet ranca i  $n$  predmeta različitih veličina, tako da  $i$ -ti predmet ima veličinu  $k_i$ ,  $1 \leq i \leq n$ . Potrebno je pronaći podskup predmeta čija je suma veličina jednaka tačno  $K$  ili ustanoviti da takav podskup ne postoji. Podrazumeva se da se veličine predmeta daju naknadno tj. da one nisu deo inicijalnog problema. Dakle,  $P(i, K)$  označava problem sa prvih  $i$  predmeta i rancem veličine  $K$ .

Induktivna hipoteza: Umemo da resimo  $P(n-1, k)$  za sve  $k$ ,  $0 \leq k \leq K$ .

Bazni slucaj  $P(1, k)$  se lako rešava: za  $k = 0$  uvek postoji rešenje (trivijalno, nula sabiraka), a u protivnom rešenje postoji akko je  $k_1 = k$ .

Problem  $P(n, k)$  svodi se na dva problema  $P(n-1, k)$  i  $P(n-1, k-k_n)$  (drugi problem otpada ako je  $k-k_n < 0$ ). Oba ova problema se mogu rešiti indukcijom. Prepreka na koju se nailazi je veoma mala efikasnost algoritma. Dakle, problem veličine  $n$  sveden je na dva problema velicine  $n - 1$ . Svaki od dobijenih problema svodi se na po dva naredna problema, što dovodi do *eksponencijalnog algoritma*.

Srećom, u većini slučajeva moguće je poboljšati efikasnost prilikom rešavanja ovakvih problema. Može se uočiti da ukupan broj različitih problema nije preveliki jer prvi parametar može imati najviše  $n$ , a drugi najviše  $K$  različitih vrednosti. Dakle, ukupan broj različitih problema je najviše  $nK$ . [2]

## Poboljšanje algoritma za rešavanje jednodimenzionog problema ranca

Kako je ukupan broj različitih problema  $nK$ , očigledno je da su neki od njih dva puta rešavani. U tom slučaju, koristan je metod koji pamti sva nađena rešenja da bi se izbeglo ponovno traženje rešenja istog problema.

Metod koji se koristi za rešavanje ovog problema je dinamičko programiranje. Suština dinamičkog programiranja je formiranje velikih tabela (u opštem slučaju višedimenzionalnih) sa svim rezultatima. Tabele se konstruišu iterativno, a svaki element se izračunava na osnovu već izračunatih elemenata.

Za smeštanje svih izračunatih rezultata koristi se posebna matrica dimenzije  $n \times K$ . Element  $(i, k)$  matrice sadrži informacije o rešenju problema  $P(i, k)$ . Svođenje po induktivnoj hipotezi ekvivalentno je izračunavanju jedne vrste na osnovu prethodne vrste matrice. Ako je potrebno odrediti i podskup sa zadatim zbirom, onda uz svaki element matrice treba sačuvati i informaciju o tome da li je odgovarajući element skupa uključen u zbir u tom koraku. Prateći ove informacije unazad od elementa  $(n, K)$ , može se rekonstruisati podskup sa zbirom  $K$ . [3]

Ispod je dat pseudokod<sup>1</sup> koji implementira ovaj način rešavanja.

**Ulaz:**  $S$  (vektor dužine  $n$  sa veličinama predmeta) i  $K$  (zapremina ranca).

**Izlaz:**  $P$  (matrica, tako da je  $P[i,k]$ .Postoji = *true* akko postoji rešenje problema ranca sa prvih  $i$  predmeta, za veličinu ranca  $k$ ;  $P[i,k]$ .Pripada = *true* akko  $i$ -ti predmet pripada tom rešenju).

**begin**

$P[0, 0]$ .Postoji := *true*;

**for**  $k := 1$  **to**  $K$  **do**

$P[0,k]$ .Postoji := *false*;

        {elementi  $P[i, 0]$  za  $i \geq 1$  biće izračunati na osnovu  $P[0, 0]$ }

**for**  $i := 1$  **to**  $n$  **do**

**for**  $k := 0$  **to**  $K$  **do** {izračunavanje elementa  $P[i, k]$ }

$P[i, k]$ .Postoji := *false*; {polazna vrednost}

**if**  $P[i - 1, k]$ .Postoji **then**

$P[i, k]$ .Postoji := *true*;

$P[i,k]$ .Pripada := *false*

**else if**  $k - S[i] \geq 0$  **then**

**if**  $P[i - 1, k - S[i]]$ .Postoji **then**

$P[i, k]$ .Postoji := *true*;

$P[i,k]$ .Pripada := *true*;

**end**

---

<sup>1</sup> Pseudokod za rešavanje jednodimenzionalnog problema ranca u celosti je preuzet iz knjige "Algoritmi" čiji je autor redovni profesor na Matematičkom fakultetu, dr Miodrag Živković. [2]

## Dvodimenzionalni problem maksimalnog ranca

Pored navedenog jednodimenzionalnog, problem koji zahteva veću pažnju i koji je višestruko bitniji je dvodimenzionalni problem ranca kada je predmet opisan svojom veličinom i vrednošću. Cilj je pronaći podskup predmeta tako da njihova veličina ne prelazi zadatu granicu (kapacitet ranca) a da je pritom suma vrednosti predmeta iz tog podskupa maksimalna. [4]

Ovaj problem je prvo rešen naivnom metodom koja daje ispravan rezultat, ali po cenu velike složenosti, a zatim metodom dinamičkog programiranja pomoću kojeg je složenost umanjena.

Naivni metod kojim se rešava problem maksimalnog ranca dat je vidu C koda.

```
//Funkcija vraca maksimalnu vrednost koja moze biti smestena u ranac kapaciteta K

int ranac(int K, int tezine[], int vrednosti[], int n) {

    // Bazni slucaj
    if (n == 0 || K == 0)
        return 0;

    // Ako je tezina n-tog predmeta veka od kapaciteta ranca K tada predmet
    // nece biti ubacen u ranac.

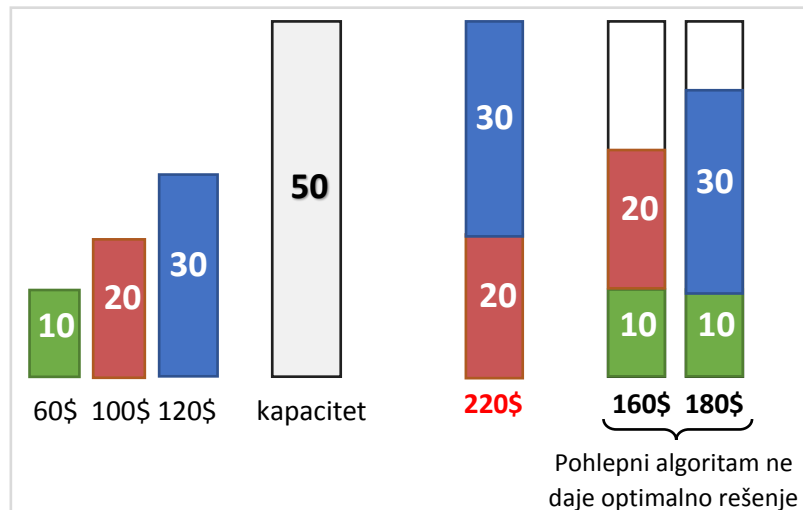
    if (tezine[n-1] > K)
        return ranac(K, tezine, vrednosti, n-1);

    //Vraca maksimum naredna dva slucaja:
    //1. uzima se n-ti predmet
    //2. ne uzima se n-ti predmet

    else return max( vrednosti[n-1] + ranac(K-tezine[n-1], tezine, vrednosti, n-1),
                    ranac(K, tezine, vrednosti, n-1) );

}
```

Iako je rešenje intuitivno jasno i prati formalni opis problema, ovakav algoritam ima veoma veliku složenost. Vremenska složenost navedenog naivnog metoda je eksponencijalna, odnosno  $O(2^n)$ .



Slika 2 Grafički prikaz mogućih rezultata različitih metoda

Napredniji metod koji rešava problem maksimalnog ranca ima svojstva dinamičkog programiranja. Kao i kod drugih problema koji se rešavaju dinamičkim programiranjem, umesto ponovnog izračunavanja potproblema koriste se matrice u kojima se pamte svi dotadašnji rezultati. [5]

//Funkcija vraća maksimalnu vrednost koja može biti smestena u ranac kapaciteta K  
 int ranac(int kapacitet, int tezine[], int vrednosti[], int n){

int R[n+1][kapacitet+1];

// Popunjavanje tabele odozdo nagore

for (int i = 0; i <= n; i++){

for (int w = 0; w <= kapacitet; w++){

if (i==0 || w==0)

R[i][w] = 0;

else if (tezine[i-1] <= w)

R[i][w] = max(vrednosti[i-1] + R[i-1][w-tezine[i-1]], R[i-1][w]);

else

R[i][w] = R[i-1][w];

}

}

return R[n][kapacitet];

}

Složenost ovakvog pristupa je  $O(nK)$  gde je  $n$  broj predmeta, a  $K$  kapacitet ranca. U nastavku se lako može uočiti koliko vremensko poboljšanje donosi ovaj pristup pri većim dimenzijama problema.

## Poređenje rada navedenih pristupa

Navedene metode su pokrenute na dva različita hardverska sistema, dok je korišćeni operativni sistem bio isti - GNU/Linux Ubuntu. Prilikom kompilacije, korišćen je integrisani kompilator - GCC.

Rezultati koji su navedeni u prvoj tabeli dobijeni su na sistemu sa sledećim komponentama:  
Dvojezgarni Intelov Celeron procesor koji radi na frekvenciji od 2.10 GHz uz 2GB radne memorije.

Težine	Vrednosti	Kapa- citet	Rezultat	Naivna metoda	Dinam. program.
9, 11, 10, 12	45, 67, 69, 75	20	114	0.000002 s	0.000004 s
2, 3, 4, 5	3, 4, 5, 6	5	7	0.000002 s	0.000002 s
1, 5, 3, 4	15, 10, 9, 5	8	29	0.000004 s	0.000002 s
10, 20, 30, 10, 10, 10	600, 100, 120, 90, 90, 90	50	880	0.000004 s	0.000007 s
23, 26, 20, 18, 32, 27, 29, 26, 30, 27	505, 352, 458, 220, 354, 414, 498, 545, 473, 543	67	1270	0.000005 s	0.000013 s
3, 5, 3, 4, 1, 1, 2, 3	1500, 1000, 900, 500, 1000, 1000, 1200, 1300	8	4800	0.000006 s	0.000003 s
3, 5, 3, 4, 1, 1, 2, 3, 4, 6, 7, 1, 1, 1	1500, 1000, 900, 500, 1000, 1000, 1200, 1300, 1100, 1500, 1550, 300, 400, 330	8	5100	0.000024 s	0.000004 s
3, 5, 3, 4, 1, 1, 2, 3, 4, 6, 7, 1, 1, 1, 2, 1, 1, 3	1500, 1000, 900, 500, 1000, 1000, 1200, 1300, 1100, 1500, 1550, 300, 400, 330, 350, 670, 360, 1000	8	5370	0.000105 s	0.000005 s

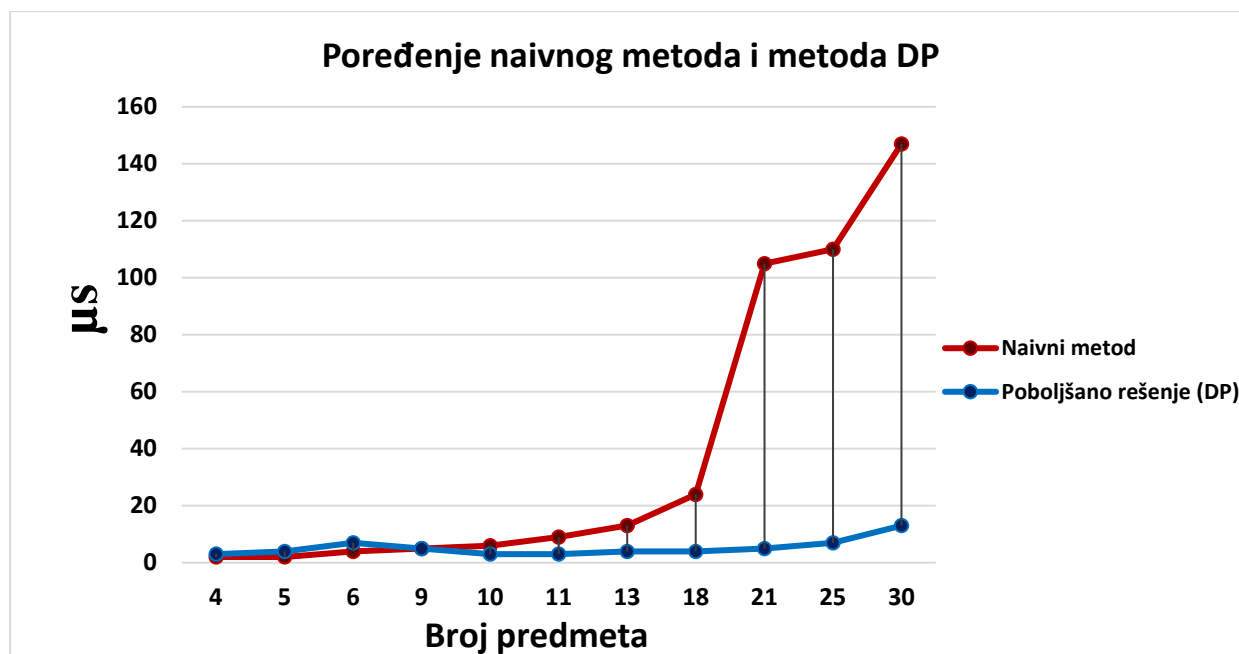
Komponente drugog sistema na kome su testirani navedeni kodovi su sledeće: Intel Core i5 procesor sa 2 fizička i 4 logička jezgra sa radom na 3.30 GHz i 8GB radne memorije.

Težine	Vrednosti	Kapa- citet	Rezultat	Naivna metoda	Dinam. program.
9, 11, 10, 12	45, 67, 69, 75	20	114	0.000006 s	0.000009 s
2, 3, 4, 5	3, 4, 5, 6	5	7	0.000004 s	0.000007 s
1, 5, 3, 4	15, 10, 9, 5	8	29	0.000005 s	0.000008 s
10, 20, 30, 10, 10, 10	600, 100, 120, 90, 90, 90	50	880	0.000008 s	0.000017 s
23, 26, 20, 18, 32, 27, 29, 26, 30, 27	505, 352, 458, 220, 354, 414, 498, 545, 473, 543	67	1270	0.000014 s	0.000019 s
3, 5, 3, 4, 1, 1, 2, 3	1500, 1000, 900, 500, 1000, 1000, 1200, 1300	8	4800	0.0000013 s	0.000008 s
3, 5, 3, 4, 1, 1, 2, 3, 4, 6, 7, 1, 1, 1	1500, 1000, 900, 500, 1000, 1000, 1200, 1300, 1100, 1500, 1550, 300, 400, 330	8	5100	0.000048 s	0.000011 s
3, 5, 3, 4, 1, 1, 2, 3, 4, 6, 7, 1, 1, 1, 2, 1, 1, 3	1500, 1000, 900, 500, 1000, 1000, 1200, 1300, 1100, 1500, 1550, 300, 400, 330, 350, 670, 360, 1000	8	5370	0.000182 s	0.000013 s



Upoređivanjem izmerenih vremena uočena je očekivana tendencija da metod koji koristi dinamičko programiranje zahteva manje vremena za izvršavanje kada je dimenzija problema veća. Naivni pristup je imao isto pa čak i kraće vreme izvršavanja za manje dimenzije, ali čim je dimenzija problema počela da raste, došlo je i do naglog porasta količine vremena potrebnog da se naivni metod izvrši, što se može jasno videti sa [grafika](#) ispod.

Dodatno, iz navedenih rezultata može se uočiti zanimljivost da su programi radili brže, odnosno da se ciljna funkcija izvršavala kraće, na sistemu koji ima sporiji radni takt, manje keš i radne memorije.



*Grafik 1: Poređednje vremena rada obe metode*

## Zaključak

Problem maksimalnog ranca koji je u ovom radu predstavljen jeste jedan od poznatijih problema u računarstvu koji ima višestruku primenu u svakodnevnom životu. Upravo zbog toga, ovom problemu se pristupa na mnogo različitih načina. Ipak, mogu se izdvojiti dva pristupa: naivni i dinamičko programiranje. Prvi je jednostavniji za razumevanje i do njega se lako dolazi, daje ispravne rezultate ali uz veću potrebu za vremenom. Drugi, optimizovani pristup je očekivano bolji. Već prilikom malo većih dimenzija problema (20 i više) mnogo je brži.

Svi opisani algoritmi se zasnivaju na nekoj vrsti metoda pretraživanja što u najgorem slučaju može imati eksponencijalno vreme. To bi predstavljalo zadovoljavajući rezultat ako bismo nekako mogli da dokažemo da nije moguće pronaći algoritam koji radi u polinomijalnom vremenu. Kako nije pronađen nijedan dokaz da razmatrane varijante problema ranca ne mogu biti rešene u polinomijalnom vremenu, ostaje nam da tragamo za pristupom koji bi nas doveo do te željene optimalnosti.

## Korišćena literatura

- [1] M. B. I. K. Kleinberg, „A Knapsack Secretary Problem with Applications,” *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pp. 16-28, 2007.
- [2] prof. dr M. Živković, *Algoritmi*, Beograd: Matematički fakultet, 2000.
- [3] Arnaud Fréville, „The multidimensional 0–1 knapsack problem: An overview,” *European Journal of Operational Research*, t. 155, br. 1, pp. 1-21, May 2004.
- [4] P. C. Beasley, „A Genetic Algorithm for the Multidimensional Knapsack Problem,” *Journal of Heuristics*, t. 4, br. 1, pp. 63-86, June 1998.
- [5] D. P. P. T. Silvano Martello, „Dynamic Programming and Strong Bounds for the 0-1 Knapsack Problem,” *Management Science*, t. 45, br. 3, pp. 297-454, 1 March 1999.
- [6] V. Kann, „School of Electrical Engineering and Computer Science,” 20 3 2000. [Na mreži]. Available: <https://www.nada.kth.se/~viggo/wwwcompendium/node211.html#7374>. [Poslednji pristup 17. januar 2019].