Univerzitet u Tuzli

Fakultet elektrotehnike


# Uvod u Računarske Algoritme

---

## Zadaća 3

## Memoizacija i dinamičko programiranje. Backtracking algoritmi.

---

19. maj 2019.

## Zadatak 1:

Neka je sekvenca cijelih brojeva definisana na sljedeći način:

*a(0) = 2*
*a(1) = 4*
*a(2) = 12*
*a(n) = 3*a(n-3)+2*a(n-2)+a(n-3)*

Potrebno je napisati funkciju koja će za proslijeđeni cijeli broj *n* vratit n-ti broj u prethodno definisanoj sekvenci. Rješenje implementirati rekurzivno bez memoizacije, sa memoizacijom, te nerekurzivno primjenom principa dinamičkog programiranja.

Napisati program koji sa proizvoljnim parametrima testira funkciju.

## Zadatak 2:

Potrebno je napisati funkciju koja će za proslijeđeni cijeli broj n vratiti broj različitih načina na koje možemo dobiti broj *n* kao zbir 1, 3 i 4. Rješenje implementirati rekurzivno bez memoizacije, sa memoizacijom, te nerekurzivno primjenom principa dinamičkog programiranja.
Naprimjer, za n = 5 imamo 6 različitih kombinacija brojeva 1, 3 i 4 čijim sabiranjem možemo dobiti 5:
5 = 1 + 1 + 1 + 1 + 1
= 1 + 1 + 3
= 1 + 3 + 1
= 3 + 1 + 1
= 1 + 4
= 4 + 1

## Zadatak 3 - Najduža rastuća podsekvenca

Implementirati algoritam koji će za zadanu sekvencu cijelih brojeva pronaći dužinu njene najduže rastuće podsekvence. Na primjer, za sekvencu {9, 21, 8, 32, 20, 40, 59} treba da vrati broj 5 (jer je najduža rastuća podsekvenca date sekvence {9, 21, 32, 49, 59}). Implementirati rekurzivno rješenje bez memoizacije, sa memoizacijom, te nerekurzivno rješenje primjenom principa dinamičkog programiranja.

## Zadatak 4 - Generisanje binarnih brojeva

Napisati funkciju koja implementira rekurzivni algoritam za generisanje i ispisivanje svih binarnih brojeva od n cifara gdje je broj cifara 1 veći ili jednak od broja cifara 0 za svaki prefiks datog broja.
Primjer ispisa ovakvog algoritma:
1111
1110
1101

1100
1011
1010

Broj 1001 nije ispisan, jer se u prefiksu sa 3 elementa (100) nalazi više cifara 0 nego cifara 1.

Algoritam ne smije ulaziti u one grane rekurzije gdje uslov nije zadovoljen (backtracking).
Sam algoritam koji se poziva u main-u treba imati što jednostavniji potpis funkcije. Moguće je koristiti i pomoćne funkcije sa složenijim potpisom u implementaciji.

Napisati program koji sa proizvoljnim parametrima testira funkciju.

## Zadatak 5 - Sudoku

Implementirati rekurzivni algoritam koji kompletira parcijalno popunjenu sudoku slagalicu. Sudoku je jedan oblik magičnog kvadrata u kome je cilj kvadrat sa mrežom 9x9 ispuniti brojevima od 1 do 9 i to tako da se u svakom redu, koloni, kao i kvadratu dimenzija 3x3 (koji je sadržan unutar glavnog kvadrata 9x9), određeni broj pojavi samo jednom. Naprimjer, za početno stanje kao na slici 1, algoritam treba da generiše brojeve označene crvenom bojom na način kao na slici 2.

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

*slika 1*

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

*slika 2*

Navedenu funkciju implementirati na takav način da sljedeći kod:

```
int main() {
  int mreza[9][9] = {{5, 3, 0, 0, 7, 0, 0, 0, 0},
                     {6, 0, 0, 1, 9, 5, 0, 0, 0},
                     {0, 9, 8, 0, 0, 0, 0, 6, 0},
                     {8, 0, 0, 0, 6, 0, 0, 0, 3},
                     {4, 0, 0, 8, 0, 3, 0, 0, 1},
                     {7, 0, 0, 0, 2, 0, 0, 0, 6},
                     {0, 6, 0, 0, 0, 0, 2, 8, 0},
                     {0, 0, 0, 4, 1, 9, 0, 0, 5},
                     {0, 0, 0, 0, 8, 0, 0, 7, 9}};


  if (rijesiSudoku(mreza))
        ispis(mreza);
  else
        std::cout << "Rjesenje za dato pocetno stanje ne postoji" << std::endl;
}
```

Rezultira sljedećim ispisom nakon izvršenja:
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9

## Zadatak 6 - Knapsack problem

Dječak je došao u zabavni park sa ukupno 10KM i želi potrošiti što više vremena na različitim vožnjama. Pošto svaka vožnja ima različitu cijenu i dužinu trajanja, dječak je odlučio napraviti efikasan algoritam (polinomske vremenske složenosti) koji će mu u što kraćem roku dati optimalno rješenje sa stanovišta utrošenog vremena u skladu sa njegovim budžetom. Zadatak je implementirati takav algoritam uzimajući u obzir da dječak nije htio ni na jednu vožnju ići više od jednom.

| tip vožnje | cijena | trajanje (min) |
|---|---|---|
| 1 | 10 | 40 |
| 2 | 5 | 18 |
| 3 | 4 | 35 |
| 4 | 2 | 2 |
| 5 | 3 | 4 |
| 6 | 1 | 10 |

Algoritam treba da vrati ukupno trajanje svih odabranih vožnji. Također, potrebno je osmisliti i
implementirati funkcionalnost za ispis tipova vožnji koje je najbolje odabrati da bi ukupno trajanje bilo maksimalno.