Autor: hrckov

Napredni algoritmi i strukture podataka

3. Laboratorijska vježba
Dinamičko programiranje

Zadatak:

Riješiti neki problem primjenom dinamičkog programiranja.
Za kolokviranje vježbe važno je pregledno i jasno opisati problem i njegove karakteristike koje ga čine prikladnim za primjenu dinamičkog programiranja.
Nije potrebno programirati rješenje, slobodno iskoristite program dostupan na stranici predmeta

Uvod:

Dinamičkim programiranjem riješen je problem izračuna n-tog člana finonaccijevog niza.
Fibonacciev niz je izražen formulom:

$$F(n) := \begin{cases} 0 & \text{ako je } n = 0; \\ 1 & \text{ako je } n = 1; \\ F_{n-1} + F_{n-2} & \text{ako je } n > 1. \end{cases}$$

Opis problema:

Kako bi problem bio rješiv načelom dinamičkog programiranja, problem mora zadovoljavati uvjete optimalne podstrukture i preklopljenosti podproblema.

Optimalna podstruktura
Vrijednost člana fibonaccievog niza se sastoji od zbroja vrijednosti njegova dva prethodnika, te se svaki od njih sastoji od svojih prethodnika, rekurzivno do prva dva člana. Time je zadovoljen uvjet optimalne podstrukture – optimalno rješenje sadrži u sebi optimalna rješenja podproblema.

Preklopljenost podproblema
Kako bi se izračunala vrijednost člana fibonaccijevog niza potrebno je izračunati vrijednosti prethodnika rekurzivno do prva dva člana. Pri tom izračunu dolazi do preklapanja vrijednosti, tj višestrukog računanja vrijednosti. Time je zadovoljen uvjet preklopljenosti podproblema – rješavanje problema zahtjeva podproblema čija su rješenja jednaka.

Točno preneseni matematički algoritam (definicija) za izračun vrijednosti člana fibonaccievog niza stvara kod eksponencijalne složenosti, te je kao takav ne efikasan.

```
function fib(n)
       if n = 0 return 0
       if n = 1 return 1
       return fib(n − 1) + fib(n − 2)
```

Pseudokod matematičke definicije

Znatno brže rješenje se dobiva koristeći dinamičko programiranje. Umjesto stvaranja stabla poziva funkcije i računanja vrijednosti nekih članova više puta, pamte se prethodne vrijednosti i koriste za izračun trenutne.
To je moguće na dva načina: od dna prema vrhu (bottom-up) ili od vrha prema dnu (top-down).

1. Top-down način
Umjesto višestrukog računanja članova niza, ovim postupkom spremamo dobivene vrijednosti u listu i koristimo kada su potrebne. Algoritam se rekurzivno spušta do prva dva člana, te na povratku puni listu i koristi vrijednosti iz liste za izračun. Time složenost algoritma postaje O(n).

```
list l = list(0,1)
function fib(n)
       if (length(l)-1 < n)
              l[n] = fib(n − 1) + fib(n − 2)
              return l[n]
```

Pseudokod top-down načina

2. Bottom-up način
Umjesto višestrukog računanja članova niza, ovaj algoritam računa vrijednosti od najnižih do tražene, pri tome pamti zadnje dvije vrijednosti koje se korite za izračun nove. Time složenost algoritma postaje O(n).

```
function fib(n)
       previousFib = 0
       currentFib = 1
       if n = 0
              return 0
       else
              for(i=0;i<n-1;i++)
                     newFib = previousFib + currentFib
                     previousFib = currentFib
                     currentFib = newFib
       return currentFib
```

Pseudokod bottom-up načina

Zaključak

Koristeći metode dinamičkog programiranja algoritam eksponencijalne složenosti pretvoren je u algoritam složenosti O(n). Ta činjenica dobro prikazuje prednosti korištenja metoda dinamičkog programiranja radi ubrzanja rada sustava i manjeg opterećenja memorije. Aplikacija ima implementirana sva tri gore navedena algoritma i prati vrijeme potrebno za izračun kako bi se prikazala razlika u brzini izvođenja pojedinog algoritma.