

Fakultet elektrotehnike i računarstva
Zavod za primjenjeno računarstvo

Napredni algoritmi i strukture podataka

2. laboratorijska vježba

Zagreb, 11.12.2015

1. Zadatak

Zadatci za 5 bodova

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

Problem

Do ispita je ostalo još 17 h. Od svih materijala za učenje potrebno je odabrati one koje će doprinijeti najviše koristi. Materijali su navedeni u tablici (A do F), zajedno s korisnošću i vremenom potrebnim da se nauče.

	A	B	C	D	E	F
Korisnost (value)	10	7	20	15	15	7
Vrijeme (cost)	5	2	12	11	9	3

2. Rješenje zadatka

2.1. Teorijski uvod

Dinamičko programiranje (Dynamic Programming) je strategija (metoda) kojoj je osnovno načelo postupno graditi rješenje složenog problema koristeći rješenja istovrsnih manje složenih problema (bottom-up pristup). Da bi bio rješiv po načelu dinamičkog programiranja, problem mora zadovoljavati sljedeća dva uvjeta:

1. optimalna podstruktura (optimal substructure)

- svojstvo problema da optimalno rješenje sadrži u sebi optimalna rješenja nezavisnih podproblema (sastoji se od njih)

2. preklopljenost podproblema (overlapping subproblems)

- svojstvo problema da njegovo rješavanje zahtijeva (vodi u) višekratno rješavanje identičnih pod ... podproblema pa se prethodni rezultati mogu iskoristiti za brže rješavanje kasnijih koraka (primjer: Fibbonacievi brojevi)
- pod ... podproblemi i dalje moraju biti nezavisni

2.2. Implementacija

Popunjavamo tablicu kojoj redovi predstavljaju vrijeme, a stupci materijali. Uzimamo u razmatranje jednu po jednu stvar i donosimo odluke primjenom rekurzivne formule:

$$v_0(.) = 0$$

$$v_k(c) = \max \{v_{k-1}(c), v_{k-1}[c - \text{cost}(k)] + \text{value}(k)\}$$

Kako bi se algoritam ubrzao sortirat ćemo materijale po vremenu od manje prema većem:

	B	F	A	E	D	C
Korisnost (value)	7	7	10	15	15	20
Vrijeme (cost)	2	3	5	9	11	12

	B	F	A	E	D	C
2	7	7	7	7	7	7
3	7	7	7	7	7	7
5	7	14	14	14	14	14
7	7	14	17	17	17	17
8	7	14	17	17	17	17
9	7	14	17	17	17	17
10	7	14	24	24	24	24
11	7	14	24	24	24	24
12	7	14	24	24	24	24
13	7	14	24	24	24	24
14	7	14	24	29	29	29
15	7	14	24	29	29	29
16	7	14	24	32	32	32
17	7	14	24	32	32	34

Materijali u koje ulazi crvena linija uzimamo kao konačno rješenje, a one u koje ulazi plava linija ne uzimamo.

Rješenje zadatka je:

Da bi maksimizirali korisnost prije ispita, učit ćemo iz materijala B, C i F.

2.3. Provjera programskim kodom

Za provjeru sam koristio program dostupan na stranici predmeta. Programski kod ne sortira predmete po value, kao što sam ja, pa sam stoga podatke zadao u već sortiranom obliku tako da se izlazna tablica podudara sa našom tablicom.

```
static void Main(string[] args)
{
    int i, j;

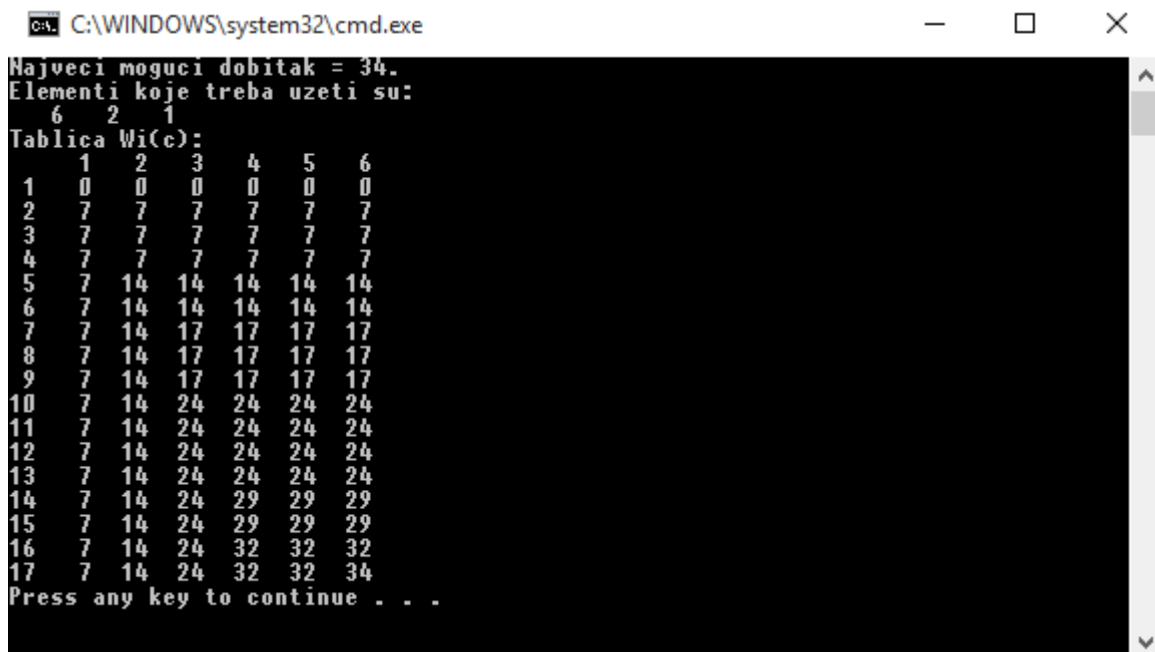
    //Podatci za učenje (mala tablica i plitka rekurzija).
    //const int items = 3;           //broj stvari
    //const int maxcost = 2;         //najveća dozvoljena cijena
    //int[] values = new int[items + 1] { 0, 5, 3, 3 };
    //int[] costs = new int[items + 1] { 0, 2, 1, 1 };

    //Složeniji problem.
    const int items = 6;           //broj stvari
    const int maxcost = 17;        //najveća dozvoljena cijena
    int[] values = new int[items + 1] { 0, 7, 7, 10, 15, 15, 20 }; //za items=6
    int[] costs = new int[items + 1] { 0, 2, 3, 5, 9, 11, 12 }; //za items=6

    int[,] tableW = new int[maxcost + 1, items + 1]; //Tablica W. redci = cijene, stupci = elementi
    bool[,] decisions = new bool[maxcost + 1, items + 1]; //tablica za rekonstrukciju niza odluka
    //Ista struktura kao i tablica vrijednosti ciljne funkcije. Polje [i,j] =true ako se u
    //polje [i,j] tablice vrijednosti dolazi uzimanjem j-tog elementa, inače =false.

    //Rješenje problema je rekurzivna relacija
    // Vi(c) = max( Vj(c); Vi(c-ci)+values(i), j!=i ) ; j=1,2,...,n. Izravna primjena te
    //relacije je relativno sporo rješenje, ali rješavanje se može ubrzati primjenom memoizacije,
    //tj. održavanjem tablice W kojoj su elementi W[c,i]=Vi(c). Prvi redak i stupac (indeks =0)
    //tablice W se pune nulama radi logike nerekurzivnog programa, a sva ostala polja se
    //inicijaliziraju na -1 jer rezultati u tablici ne mogu biti negativni pa se to koristi za
    //memoizaciju (kao pokazatelj da ta vrijednost još nije izračunana) i formatiranje ispisa.
    //U običnom rekurzivnom rješenju tablica nije potrebna, ali je tu zbog kontrole i učenja! :)
```

Nakon pokretanja programa rezultat nam se ispiše u konzoli:



```
C:\WINDOWS\system32\cmd.exe
Najveci moguci dobitak = 34.
Elementi koje treba uzeti su:
6 2 1
Tablica Wi(c):
1 0 0 0 0 0 0
2 7 7 7 7 7 7
3 7 7 7 7 7 7
4 7 7 7 7 7 7
5 7 14 14 14 14 14
6 7 14 14 14 14 14
7 7 14 17 17 17 17
8 7 14 17 17 17 17
9 7 14 17 17 17 17
10 7 14 24 24 24 24
11 7 14 24 24 24 24
12 7 14 24 24 24 24
13 7 14 24 24 24 24
14 7 14 24 29 29 29
15 7 14 24 29 29 29
16 7 14 24 32 32 32
17 7 14 24 32 32 34
Press any key to continue . . .
```

Kao što vidimo tablica i izlaz programa se podudaraju.

3. Zaključak

Dinamičko programiranje je jako korisno jer štedi vrijeme i resurse za rješavanje nekih zadataka. Glavna prednost je korištenje pristupa bottom-up, gdje se izračunata pod rješenja koriste za daljnje računanje cijelog problema. U ovom zadatku vrlo lako možemo odrediti i koje je materijale najbolje koristiti i za manje od 17 h, jednostavnim pogledom na već napravljenu tablicu (jer su to pod rješenja ukupnog rješenja). Isto tako bi mogli proširiti tablicu ukoliko bi trebalo računat najbolju korisnost za više od 17 h. To bi imalo smisla raditi do max 42 h, jer nakon toga se tablica ne bi mijenjala, jer bi tada mogli naučit sve materijale.

4. Literatura

[1] Napredne strukture podataka (Advanced Data Structures), Copyright (C) Nikica Hlupić and Damir Kalpić 2009, All rights reserved