10 Programare Dinamica

10.1 Obiective

Scopul acestei lucrari este de a va familiariza cu paradigma programarii dinamice. Se prezinta pe scurt fundamentele teoretice, si se propun cateva probleme care pot fi rezolvate utilizand aceasta tehnica.

10.2 Noțiuni teoretice

Programarea dinamica este o tehnică algoritmică bazată de regulă pe o formula de recurență și una (sau mai multe) stări inițiale. De obicei, metoda este adecvata problemelor care solicita determinarea unui optim(minim sau maxim), in urma unui proces decizional care are mai multi pasi. O particularitate ale acestui tip de probleme este ca foloseste structuri tabelare pentru stocarea solutiilor intermediare pentru a putea fi folosite pe viitor. O sub-solutie a problemei este construita din rezultatele descopeite in pasi anteriori. Problemele rezolvate folosind programarea dinamica au o complexitate polinomiala, care asigura un timp de executie mult mai bun ca si tehnici precum backtracking sau brute-force search.

În sintagma ”programarea dinamică” cuvantul programare se referă la ideea de planificare și nu la programare în sensul informatic. Cuvantul dinamic se refera la maniera in care sunt construite tabelele in care se retin informatiile referitoare la solutiile partiale.

Programarea dinamica seamană cu strategia ”divide and conquer” în ideea în care și aici avem de a face cu divizarea soluției inițiale în sub-probleme. Diferența esențială este accea că, în programarea dinamică subproblemele se suprapun, sunt dependente. Astfel că, soluția unei probleme se utilizează în construirea soluțiilor altor subprobleme (din acest motiv soluțiile parțiale trebuie stocate).

Pentru a se aplica metoda, problema trebuie să îndeplinească două condiții: să aibă substructură oprimală și suprob-lemele în care se descompune să se suprapună parțial. Substructura optimală înseamnă că o soluție optimală a problemei e conținută în soluțiile optimale ale subproblemelor. Ca urmare, prin programare dinamică o soluție optimă este găsită printr-o secvență de decizii care depind de hotărâri luate anterior și care satisfac principiul optimalității. Etapele rezolvării unei probleme utilizând metoda programării dinamice are următorii pași:

1. Verificarea principiului optimalității și identificarea subproblemelor.
2. Alegerea unei structuri de date corespunzatoare care să rețină soluțiile subproblemelor.
3. Determinarea unei relații de recurență care să caracterizeze structura optimală (dependența soluției subproblemei curente de rezulatele subproblemelor în care se descompune).
4. Rezolvarea recurentei in ordinea crescatoare a dimensiunilor subproblemelor (de obicei bottom-up).
5. Construirea unei soluții optime pe baza valorii calculate pentru soluția optimă.

Cea mai intuitvă problemă care se poate aborda folosind principiile programării dinamice este determinarea numerelor din șirul lui Fibonacci: $F_n = F_{n-1} + F_{n-2}$ cu $F_1 = 1$ si $F_2 = 1$.

Fie $< s_0, s_1, \ldots s_n >$ o secventa de stari unde s_0 este starea initiala si s_n este starea finala. In starea finala se ajunge prin luarea unei secvente de decizii $d_1, d_2, \ldots d_n$ (decizia d_i transforma starea s_{i-1} in starea s_i). Daca secventa de decizii $< d_i, d_{i+1}, \ldots d_j >$ care schimba starea s_{i-1} in starea s_j (cu starile intermediare $s_i, s_{i+1}, \ldots s_{j-1}$) este optima si daca pentru orice $i \leq k \leq j-1$ atat $< d_i, \ldots d_k >$ cat si $< d_{k+1}, d_{k+2}, \ldots d_j >$ sunt secvente de decizii optimea, care stransforma starea s_{i-1} in starea s_k, respectiv starea s_k in starea s_j, atunci este satisfacut principiul optimalitatii.

10.3 Mersul lucrarii

10.3.1 Probleme obligatorii

Ex. 1 — Plata unei sume S cu monede - varianta 1: Scrieți o funcție care determină numărul maxim de combinații în care poate fi plătită o sumă S, folosind monedele date.

Presupunem că avem monedele 1,2,5 si S = 12.

Pentru rezolvarea problemei se creează un vector de dimensiune $S+1$. Îi vom spune acestui vector combinații. Valoarea din fiecare celulă din vector va reprezenta numărul maxim de combinații pentru fiecare sumă. Fiecare locație din vector se asociază unei sume de bani (cantitatea). Spre exemplu locația a șasea din vector va corespunde numărului de combinatii care au suma 6. Vom itera prin întregul vector având fiecare moneda ca și parametru. Relația de recursivitate pentru completarea vectorului din problema noastră este ilustrata mai jos:

```
if cantitate >= val_monedei
combinatii[cantitate] += combinatii[cantitate - val_monedei]
else
Valoarea din vectorul combinatii ramane neschimbata.
```

Inițializăm combinații[0] cu 1 ca si valoare de start.

0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0	0

Iterăm peste toate cantitățile (de la 1 la 12) și verificăm unde se satisface condiția de recursivitate pentru a numara o combinatie. Spre exemplu cantitatea 1 poate fi reprezentată folosind moneda de valoare 1. După aplicarea formulei recursive de mai sus obtinem $combinatii[1] = 1$; Valoare vectorului $combinatii$ dupa folosirea monedei 1 are forma de mai jos, lucru care este normal de vreme ce orice sumă (cantitate) poate fi scrisă cu ajutorul monedei unitate.

0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1

În acest moment considerăm următoarea monedă (moneda 2). Se sare peste cantitatea 1, fiindcă $1 < 2$ și se ajunge la cantitatea 2 (pentru că 2-2 == 0). Se reînnoiește vectorul pentru fiecare poziție >= 2 și se obține rezultatul de mai jos:

0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	2	3	3	4	4	5	5	6	6	7

În sfârșit, să ne uităm la modificările pe care le aduce moneda de valoare 5 în vector. De vreme ce moneda are valoare mai mare de 4, toate valorile până la 5 vor rămâne nemodificate (pentru că o suma cu valoarea 4 nu poate fi scrisă ca și sumă de monede mai mari decât ea).

0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	2	3	4	5	6	7	8	10	11	13

Se observa că numărul maxim de combinații în care poate fi scrisă suma 12 folosind monedele date se află pe poziția combinatii[12]. Se mai poate remarca și faptul că putem obține numărul maxim de combinații pentru orice sumă mai mică decât 12 folosind monezile date. Putem verifica numărul de combinații, pentru suma 5 cu monezile date $(1, 2, 5)$:

```
5 = 1 + 1+ 1+1+1
5 = 1 + 1+1 + 2
5 = 1 + 2 + 2
5 = 5
```

Deși această problemă ar putea fi rezolvată si cu tehnica greedy, această metoda nu ne poate asigura tot timpul soluția optimă, de aceea este nevoie de folosirea programării dinamice.

Ex. 2 — Plata unei sume S cu monede - varianta 2: Scrieți o funcție care determină numărul minim de monede al cărei sumă este S. Pot fi folosite oricâte monede.

Presupunem că avem monedele 1, 3, 5 și S = 11 și dorim să găsim numarul minim de monede al căror suma este S.

Asemănător cazului precedent, trebuie să identificam o stare a carei solutii optime o putem gasi si care sa ne ajute la generarea starii urmatoare. Prima data vom marca starea 0 (suma 0), cu 0, avand semnificația că am găsit o soluție cu un număr minim de 0 monede. Mergem apoi la suma 1. Mai intâi marcăm inexistența unei soluții pentru această sumă (plasăm pe acea poziție o valoare infinit). Apoi remarcăm că singura monedă, cea de valoare 1, este valabilă pentru a construi suma curentă. Pentru că am adaugat o moneda rezultatului, vom avea o solutie cu o moneda pentru suma.

Mergem apoi la urmatorul pas, suma 2. Observăm, din nou, că singura modalitate de a obtine suma, este de a folosi moneda de cu valoarea 1. Solutia optima gasita pentru suma (2-1) = 1 este moneda cu valoare 1. Aceasta moneda de 1 se va adauga inca unei monede cu valoarea 1 pentru a obtine suma dorita 2. Mergand mai departe la suma cu valoare 3 observam ca avem 2 monede care pot fi candidat (moneda de valoare 1 si cea de valoare 3). Exista o solutie pentru a obtine o suma de 2 din doua monede, asadar putem obtine si suma de 3 cu ajutorul inca a unei monede cu valoarea 1, rezultand 3 monede. Luand a doua moneda, cu cantitatea 3, observam ca valoarea la care trebuie adaugata aceasta moneda pentru a obtine suma de 3 este 0. Stiind ca suma de 0 este 0 rezulta ca putem obtine suma 3 folosind doar o moneda. Solutia nou gasita este mai buna decat cea precedenta (care folosea trei monede). Procedam in mod similar pentru toate celelalte sume ramase. Pseudocodul programului este prezentat mai jos:

```
SETEAZA Min[i] LA INFINIT pentru toti i
Min[0] = 0

For i = 1 la S
   For j = 0 la N -1
    Daca (Vj <= i AND Min[i-Vj] + 1 < Min[i]) Atunci
   Min[i] = Min[i - Vj] + 1

Output Min[S]
```

sum	0	1	2	3	4	5	6	7	8	9	10	11
nr. minim monede	0	1	2	1	2	1	2	3	2	3	2	3
Val. monedă adaugată la suma precedentă	–	1(0)	1(1)	3(0)	1(3)	5(0)	3(3)	1(6)	3(5)	1(8)	5(5)	1(10)

Ca si rezultat am obtinut numarul minim de 3 monezi a căror sumă are valoarea 11 (5 + 5 + 1).

Ex. 3 — Implementati toate exemplele discutate de cursul de Programare Dinamica.

10.3.2 Probleme extracredit*

Ex. 4 — (*) Subsecventă maximala ordonată crescătoare. Fie un vector a cu N elemente. Numim subsecventă a lui a de lungime k un sir $a' = (a_{i_1}, a_{i_2}, ..., a_{i_k})$ astfel incat sa avem $i_1 < i_2 < ... < i_k$. Sa se determine o subsecventa a lui a care este ordonata strict crescator si care are lungimea maxima.
Date de intrare:
Fisierul de intrare *scmax.in* conține pe prima linie numărul N reprezentând numărul de elemente ale vectorului a. Pe cea de-a doua linie se afla N numere naturale reprezentând elementele vectorului a.
Date de iesire: În fișierul de iesire *scmax.out* se va afișa pe prima linie $Lmax$, lungimea celui mai lung subșir crescător al șirului a. Pe cea de-a doua linie se vor afla $Lmax$ numere naturale reprezentând cel mai lung subșir crescător al vectorului a. Dacă există mai multe soluții se poate afisa oricare.
Exemplu:

scmax.in	scmax.out
5 24 12 15 15 19	3 12 15 19

Ex. 5 — Se citesc n numere naturale. Se cere să se tipărească cea mai mare sumă care se poate forma utilizând cele n numere naturale (fiecare numar participă o singură dată în calculul sumei) şi care se divide cu n, precum şi numerele care alcătuiesc această sumă.

Ex. 6 — Se citesc doua numere naturale n si m cu $1 <= m, n <= 100$ si apoi o matrice cu n linii si m coloane având elementele numere întregi cu cel mult 4 cifre fiecare. Afisati pentru fiecare coloana a matricii numarul de elemente al celui mai lung subsir strict crescator care se poate forma parcurgând elementele coloanei de sus în jos. Pentru citire se va folosi fisierul inputPD.in, iar pentru afisare fisierul outputPD.out.
Exemplu:

inputPD.in	outputPD.out
4 4 1 4 2 3 2 9 8 7 3 6 3 8 1 2 3 3	3 2 2 3