CURS 12:

Programare dinamică
- II -

Structura

- Ce este programarea dinamică ?
- Aplicație: problema discretă a rucsacului
- Tehnica memoizării
- Aplicație: înmulțirea optimală a matricilor
- Aplicație: închiderea tranzitivă a unei relații binare

Ce este programarea dinamică ?

- Este o tehnică de rezolvare a problemelor care pot fi descompuse în subprobleme care se suprapun – poate fi aplicată problemelor ce au proprietatea de substructură optimă

- Particularitatea programării dinamice constă în faptul că rezolvă fiecare subproblemă o singură dată și stochează soluțiile subproblemelor într-o structură tabelară

Ce este programarea dinamică ?

Etapele principale:

- Analiza structurii unei soluții: se stabilește legatura dintre soluția problemei și soluțiile subproblemelor (este echivalentă cu verificarea proprietății de substructură optimă). In aceasta etapă se identifică problema generică și subproblemele corespunzătoare.
- Determinarea relației de recurență dintre valoarea (criteriul de optim) corespunzătoare soluției problemei și valorile corespunzătoare soluțiilor subproblemelor.
- Dezvoltarea (în manieră ascendentă) a relației de recurență și completarea structurii tabelare utile în construirea soluției.
- Construirea soluției (utilizând informațiile completate în etapa anterioară)

Aplicație: problema rucsacului

Considerăm un set de n obiecte, fiecare fiind caracterizat de o dimensiune d și de o valoare v, și un rucsac de capacitate C. Să se selecteze un subset de obiecte astfel încât dimensiunea totală a obiectelor selectate să fie mai mică decât C iar valoarea totală a obiectelor selectate să fie maximă.

Variante:

(i) Varianta continuă (fracționară): pot fi selectate obiecte in întregime sau fracțiuni ale obiectelor.

(ii) Varianta discretă(0-1): obiectele pot fi transferate doar în întregime

Aplicație: problema rucsacului

Ipoteza (varianta simplificată):

Capacitatea rucsacului (C) și dimensiunile obiectelor $d_1,\ldots,d_n$ sunt numere naturale

Problema rucsacului poate fi reformulată astfel:

se caută $(s_1,s_2,\ldots,s_n)$ cu s_i in {0,1} astfel încât:

$s_1d_1 +\ldots+ s_nd_n <= C$ (restricție)

$s_1v_1 +\ldots+ s_nv_n$ este maximă (criteriu de optim)

Obs.

tehnica greedy poate fi aplicată și în acest caz însă NU garantează obținerea soluției optime

Aplicație: problema rucsacului

Exemplu: n=3,

C=5,

$d_1=1$, $d_2=2$, $d_3=3$

$v_1=6$, $v_2=10$, $v_3=12$

Valoare relativă: $vr_i=v_i/d_i$

$vr_1=6$, $vr_2=5$, $vr_3=4$

Ideea tehnicii greedy:

- Se sortează lista de obiecte descrescător după valoarea relativă ($vr_i=v_i/d_i$)
- Se selectează obiectele în această ordine până când nu mai încap elemente in rucsac

Soluția greedy: (1,1,0)

Valoarea totală: V=16

Obs: aceasta nu este soluția optimă;

soluția (0,1,1) este mai bună întrucât V=22

Aplicație: problema rucsacului

1. Analiza structurii unei soluții optime

Fie P(i,j) problema generică a selecției din setul de obiecte $\{o_1,\dots,o_i\}$ pentru a umple optimal un rucsac de capacitate j.

Obs:

- P(n,C) este problema inițială
- Daca i<n, j<C atunci P(i,j) este o subproblemă a lui P(n,C)
- Fie s(i,j) o soluție optimă a problemei P(i,j). Sunt posibile două situații:
  - s_i=1 (obiectul o_i este selectat) => se ajunge la subproblema P(i-1,j-d_i) și dacă s(i,j) este optimă pt pb. P(i,j) atunci și s(i-1,j-d_i) trebuie să fie soluție optimă pt subproblema P(i-1,j-d_i)
  - s_i=0 (obiectul o_i nu este selectat) => se ajunge la subproblema P(i-1,j) și dacă s(i,j) este optimă pt pb. P(i,j) atunci și s(i-1,j) trebuie să fie solutie optimă pentru subproblemă

Deci problema rucsacului are proprietatea de substructură optimă

Aplicație: problema rucsacului

2. Stabilirea relației de recurență

Fie V(i,j) valoarea corespunzătoare soluției a problemei P(i,j)

$$V(i,j) = \begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \quad \text{(mulțimea de obiecte este vidă sau capacitatea disponibilă în rucsac e 0)} \\ V(i-1,j) & \text{daca } d_i>j \text{ sau } V(i-1,j)>V(i-1,j-d_i)+v_i \\ & \text{(obiectul i nu încape în rucsac sau prin selecția lui s-ar obține o soluție mai puțin bună decât dacă obiectul nu s-ar selecta)} \\ V(i-1,j-d_i)+v_i & \text{în celelalte cazuri} \end{cases}$$

Aplicație: problema rucsacului

Relația de recurență poate fi descrisă și astfel:

$$V(i,j) = \begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ V(i-1,j) & \text{dacă } d_i>j \text{ (obiectul nu încape)} \\ \max\{V(i-1,j),\ V(i-1,j-d_i)+v_i\} & \text{dacă } d_i<=j \text{ (obiectul încape, dar e selectat doar dacă prin includerea lui se obține o valoare totală mai mare)} \end{cases}$$

Obs:

- Pentru problema P(n,C), tabelul bidimensional V are (n+1) linii și (C+1) coloane
- V(n,C) este valoarea corespunzătoare soluției optime

Aplicație: problema rucsacului

Exemplu:

$$V(i,j) = \begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ V(i-1,j) & \text{dacă } d_i>j \\ \max\{V(i-1,j),\ V(i-1,j-d_i)+v_i\} & \text{if } d_i<=j \end{cases}$$

d: 1 2 3

v: 6 10 12

V

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	6	10	16	16	16
3	0	6	10	16	18	22

Aplicație: problema rucsacului

3. Dezvoltarea relației de recurență

$$V(i,j) = \begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ V(i-1,j) & \text{if } d_i > j \\ \max\{V(i-1,j),\ V(i-1,j-d_i)+v_i\} & \text{if } d_i <= j \end{cases}$$

$i=0..n,\ j=0..C$

Algoritm:

```
computeV (v[1..n],d[1..n],C)
  FOR i←0,n DO V[i,0] ←0 ENDFOR
  FOR j←1,C DO V[0,j] ←0 ENDFOR
  FOR i←1,n DO
      FOR j ← 1,C  DO
        IF j<d[i] THEN V[i,j] ←V[i-1,j]
        ELSE
            V[i,j] ←max(V[i-1,j],V[i-1,j-d[i]]+v[i])
        ENDIF
      ENDFOR
  ENDFOR
  RETURN V[0..n,0..C]
```

Aplicație: problema rucsacului

4. Construirea soluției

Exemplu:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	$\boxed{0}$	6	6	6	6	6
2	0	6	$\boxed{10}$	16	16	16
3	0	6	10	16	18	$\boxed{22}$

Etape:

- Compară V[3,5] cu V[2,5]. Intrucât valorile sunt diferite inseamnă că o3 este selectat
- Se trece la V[2,5-d_3]=V[2,2]=10 și se compară cu V[1,2]=6. Intrucât valorile sunt diferite inseamnă că o_2 este de asemenea selectat
- Se trece la V[1,2-d_2]=V[1,0]=0. Intrucât s-a ajuns la 0 rezultă că s-a ajuns la soluție

Soluția obținută este {o_2,o_3} adică s=(0,1,1)

Obs: se presupune că cel puțin un obiect are dimensiunea mai mică decât capacitatea rucsacului

Aplicație: problema rucsacului

4. Construirea soluției

Exemplu:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	6	10	16	16	16
3	0	6	10	16	18	22

Algoritm:

```
Construct(V[0..n,0..C],d[1..n])
  FOR i←1,n DO s[i] ← 0 ENDFOR
  i←n;   j←C
  WHILE V[i,j]>0 DO
     IF V[i,j]==V[i-1,j]
                  THEN  i←i-1
     ELSE
        s[i] ←1
        j←j-d[i]
        i←i-1
    ENDIF
  ENDWHILE
  RETURN s[1..n]
```

Aplicație: problema rucsacului

Obs

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	6	6	6	6	6
2	0	6	10	16	16	16
3	0	6	10	16	18	22

Pt a construi soluția sunt suficiente doar valorile marcate

Numărul calculelor poate fi redus dacă se calculează doar valorile necesare construcției solutiei

Acest lucru se poate realiza prin îmbinarea abordării descendente cu cea ascendentă (cu reținerea valorilor calculate)

Aceasta este denumita tehnica memoizării (engleză: memoization)

Tehnica memoizării

Scop: se rezolvă doar subproblemele a căror soluție intervine in soluția problemei inițiale (în plus o subproblemă este rezolvată o singură dată)

Idee: se combină abordarea descendentă (top down) cu cea ascendentă (bottom up)

Motivație:

- Abordarea descendentă clasică rezolvă doar subproblemele ce contribuie la soluția problemei însă o subproblemă este rezolvată de câte ori apare (din acest motiv implementarea recursivă este în general ineficientă)
- Abordarea ascendentă clasică rezolvă toate subproblemele (chiar și cele care nu contribuie la soluția optimă) însă fiecare problemă este rezolvată o singură dată
- Tehnica memoizării rezolvă o singură dată doar subproblemele ce contribuie la soluția problemei

Tehnica memoizării

Etape în aplicarea tehnicii memoizării:

- Se inițializează tabelul cu o valoare virtuală (această valoare trebuie să fie diferită de orice valoare s-ar obține prin dezvoltarea relației de recurență)
- Se calculează valoarea țintă (ex: V[n,C]) în manieră recursivă însă toate valorile intermediare se stochează și se utilizează atunci când e necesar

Obs:
v[1..n], d[1..n] și
V[0..n,0..C] sunt variabile globale

Apel: comp(n,C)

Inițializare cu valoarea virtuală:

```
FOR i←0,n DO
  FOR j←0,C DO  V[i,j] ← -1 ENDFOR
ENDFOR
```

Implementare recursivă:

```
comp(i,j)
IF i=0 OR j=0 THEN V[i,j] ←0; RETURN V[i,j]
ELSE
   IF V[i,j] != -1 THEN RETURN V[i,j]
   ELSE
     IF j<d[i] THEN V[i,j] ←comp(i-1,j)
     ELSE
        V[i,j] ←
              max(comp(i-1,j),comp(i-1,j-d[i])+v[i])
     ENDIF
    RETURN V[i,j]
ENDIF
ENDIF
```

Aplicație: înmulțirea optimală a matricilor

Se consideră n matrici A_1, A_2, …, A_n si se dorește calculul produsului $A_1 * A_2 * \ldots * A_n$. Să se determine o modalitate de grupare a matricilor factor astfel încât numărul produselor scalare (între elemente ale matricilor) să fie minim

Obs:

1. Dimensiunile matricilor sunt compatibile. Presupunem că dimensiunile matricilor sunt: $p_0, p_1, \ldots, p_n$. Matricea A_i are p_{i-1} linii si p_i coloane

2. Diferitele grupări ale factorilor conduc la același rezultat (întrucât înmulțirea matricilor este asociativă) însă pot conduce la valori diferite ale numărului de inmulțiri de elemente

Aplicație: înmulțirea optimală a matricilor

Exemplu: Fie A_1, A_2 și A_3 trei matrici având dimensiunile: (2,20), (20,5) si (5,10)

$p_0=2$ $p_1=20$ $p_2=5$ $p_3=10$

Considerăm următoarele grupări:

- $(A_1*A_2)*A_3$ - aceasta necesită (2*20*5)+2*5*10=300 înmulțiri scalare (la nivel de element)

- $A_1*(A_2*A_3)$ – aceasta necesită (20*5*10)+2*20*10=1400 înmulțiri scalare

Obs: pentru valori mari ale lui n numărul de grupări posibile poate fi foarte mare

Aplicație: înmulțirea optimală a matricilor

Gruparea factorilor are, în cazul general, un caracter ierarhic:

- Primul nivel al grupării corespunde ultimei înmulțiri efectuate
- Celelalte nivele corespund grupărilor factorilor rămași

Gruparea este identificată prin poziția ultimei înmulțiri. De exemplu gruparea

$$(A_1*\ldots*A_k)*(A_{k+1}*\ldots*A_n)$$

este specificată prin indicele de grupare k

La primul nivel există (n-1) grupări posibile (1<=k<=n-1) dar există o serie de grupări posibile ale fiecărui factor $(A_1*\ldots*A_k)$ respectiv $(A_{k+1}*\ldots*A_n)$

Aplicație: înmulțirea optimală a matricilor

Numărul de grupări pentru un produs cu n factori este:

$$K(n)=\begin{cases} 1 & n<=2 \\ K(1)*K(n-1)+\ldots+K(i)*K(n-i)+\ldots+K(n-1)*K(1) & n>2 \end{cases}$$

Obs:

K(n)=C(n-1) unde C(0),C(1) … sunt numerele lui Catalan:

$$C(n)=Comb(2n, n)/(n+1)$$

Ordinul de mărime al lui K(n) este $4^{n-1}/(n-1)^{3/2}$

Tehnica forței brute este inaplicabilă!

Aplicație: înmulțirea optimală a matricilor

1. Analiza structurii unei soluții optime

Fie A(i..j) produsul $A_i*A_{i+1}*\ldots*A_j$ (i<=j)

Dacă produsul optim corespunde grupării la poziția k (i<=k<j) atunci calculul lui A(i..k) și al lui A(k+1..j) ar trebui să fie de asemenea optim (altfel calculul lui A(i..j) nu ar fi optim)

Deci este satisfacută proprietatea de substructură optimă

Aplicație: înmulțirea optimală a matricilor

2. Identificarea relației de recurență

Fie c(i,j) numărul de înmulțiri scalare necesare pentru a calcula A(i..j).

$$c(i,j)=\begin{cases} 0 & \text{dacă } i=j \\ \min\{c(i,k)+c(k+1,j)+p_{i-1}p_kp_j \mid i<=k<j\} & \text{daca } i<j \end{cases}$$

Costul calculului A(i..k)

Costul calculului A(k+1..j)

Costul înmulțirii lui A(i..k) cu A(k+1..j)

Toate valorile posibile pentru indicele de grupare k (i<=k<j) sunt analizate și se alege cea mai bună dintre ele

Aplicație: înmulțirea optimală a matricilor

3. Dezvoltarea relației de recurență

$$c(i,j)=\begin{cases} 0 & \text{if } i=j \\ \min\{c(i,k)+c(k+1,j)+p_{i-1}p_k p_j \mid i<=k<j\}, & \text{if } i<j \end{cases}$$

Exemplu

$p0=2$
$p1=20$
$p2=5$
$p3=10$

Doar partea superior triunghiulară a matricii este utilizată

	1	2	3
1	0	200	300
2	-	0	1000
3	-	-	0

Elementele sunt calculate incepând de la diagonala (j-i=0), după care se calculează elementele ai căror indici satisfac j-i=1 samd…

Aplicație: înmulțirea optimală a matricilor

3. Dezvoltarea relației de recurență

$$c(i,j)=\begin{cases} 0 & \text{if } i=j \\ \min\{c(i,k)+c(k+1,j)+p_{i-1}p_kp_j \mid i<=k<j\}, & \text{if } i<j \end{cases}$$

Fie q=j-i. Tabelul se completează pentru q variind de la 1 la n-1

Pe parcursul calculului lui c indicele grupării este stocat în structura s.

s(i,j) = indicele de grupare corespunzător calculului optimal al lui A(i..j)

Algoritm

```
Compute(p[0..n])
 FOR i←1,n DO c[i,i] ←0 ENDFOR
 FOR q ← 1,n-1 DO
   FOR i ← 1,n-q DO
      j ← i+q
      c[i,j] ← c[i,i]+c[i+1,j]+p[i-1]*p[i]*p[j]
      s[i,j] ← i
      FOR k ← i+1,j-1 DO
         r ← c[i,k]+c[k+1,j]+p[i-1]*p[k]*p[j]
         IF c[i,j]>r THEN  c[i,j] ← r
                           s[i,j] ← k
         ENDIF
      ENDFOR
   ENDFOR   ENDFOR
 RETURN c[1..n,1..n],s[1..n,1..n]
```

Aplicație: înmulțirea optimală a matricilor

Analiza complexității:

Dimensiunea problemei: n

Operație dominantă: înmulțire scalară

Ordin complexitate: $\theta(n^3)$

Algoritm

```
Compute(p[0..n])
 FOR i←1,n DO c[i,i] ←0 ENDFOR
 FOR q ← 1,n-1 DO
   FOR i ← 1,n-q DO
     j ← i+q
     c[i,j] ← c[i,i]+c[i+1,j]+p[i-1]*p[i]*p[j]
     s[i,j] ← i
     FOR k ← i+1,j-1 DO
       r ← c[i,k]+c[k+1,j]+p[i-1]*p[k]*p[j]
       IF c[i,j]>r THEN  c[i,j] ← r
                         s[i,j] ← k
       ENDIF
     ENDFOR
   ENDFOR   ENDFOR
 RETURN c[1..n,1..n],s[1..n,1..n]
```

Aplicație: înmulțirea optimală a matricilor

4. Construirea soluției

Variante ale problemei:

- Determinarea numărului minim de înmulțiri scalare
  Soluție: este dată de c(1,n)

- Calcul A(1..n) în manieră optimală
  Solutie: algoritm recursiv (opt_mul)

- Identificarea grupării optimale a factorilor (plasarea parantezelor)
  Solutie: algoritm recursiv (opt_group)

Aplicație: înmulțirea optimală a matricilor

Calculul lui A(1..n) în manieră optimală

Ipoteze:

- A[1..n] un tablou global având elemente de tip matrice (A[i] este matricea A_i)
- s[1..n,1..n] este o variabilă globală iar classic_mul este o funcție pentru calculul produsului a două matrici

```
opt_mul(i,j)
  IF i=j THEN RETURN A[i]
  ELSE
       X← opt_mul(i,s[i,j])
       Y ← opt_mul(s[i,j]+1,j)
       Z ← classic_mul(X,Y)
       RETURN Z
ENDIF
```

Aplicație: înmulțirea optimală a matricilor

Afișarea grupării optimale (pozițiile unde se plasează parantezele)

```
opt_group(i,j)
  IF i!=j THEN
          opt_group(i,s[i,j])
          WRITE i-1, s[i,j], j
          opt_group(s[i,j]+1,j)
  ENDIF
```

Cursul următor va fi despre…

… Backtracking

… și aplicații

Intrebare de final

Se consideră 3 matrici A_1, A_2 și A_3 cu dimensiunile:

A_1 are 2 și 4 coloane

A_2 are 4 linii și 3 coloane

A_3 are 3 linii și 5 coloane

Care este numărul minim de operații de înmulțire care permite calculul $A_1*A_2*A_3$

Variante de răspuns:

a) 2*4+4*3+3*5=35
b) 2*4*3=24
c) 2*4*3+2*3*5=54
d) 4*3*5=60