CURS 11:

Programare dinamică

- I -

Structura

- Ce este programarea dinamică ?
- Etapele principale în aplicarea programării dinamice
- Relații de recurență: dezvoltare ascendentă vs.dezvoltare descendentă
- Aplicații ale programării dinamice

Ce este programarea dinamică?

- Este o tehnică de proiectare a algoritmilor pentru rezolvarea problemelor care pot fi descompuse în subprobleme care se suprapun – poate fi aplicată problemelor de optimizare care au proprietatea de substructură optimă

- Particularitatea metodei constă în faptul că fiecare suproblemă este rezolvată o singură dată iar soluția ei este stocată (într-o structură tabelară) pentru a putea fi ulterior folosită pentru rezolvarea problemei inițiale.

Obs.

- Programarea dinamică a fost dezvoltată de către Richard Bellman in 1950 ca metodă generală de optimizare a proceselor de decizie.
- In programarea dinamică cuvântul programare se referă la planificare și nu la programare în sens informatic.
- Cuvântul dinamic se referă la maniera în care sunt construite tabelele în care se rețin informațiile referitoare la soluțiile parțiale.

Ce este programarea dinamică?

- Programarea dinamică este corelată cu tehnica divizării întrucât se bazează pe divizarea problemei inițiale în subprobleme. Există însă câteva diferențe semnificative între cele două abordări:
  - divizare: subproblemele în care se divide problema inițială sunt independente, astfel că soluția unei subprobleme nu poate fi utilizată în construirea soluției unei alte subprobleme
  - programare dinamică: subproblemele sunt dependente (se suprapun) astfel că soluția unei subprobleme se utilizează în construirea soluțiilor altor subprobleme (din acest motiv este important ca soluția fiecărei subprobleme rezolvate să fie stocată pentru a putea fi reutilizată)
- Programarea dinamică este corelată și cu strategia căutării local optimale (greedy) întrucât ambele se aplică problemelor de optimizare care au proprietatea de substructură optimă

Structura

- Ce este programarea dinamică ?
- Etapele principale în aplicarea programării dinamice
- Relații de recurență: dezvoltare ascendentă vs.dezvoltare descendentă
- Aplicații ale programării dinamice

Etapele principale în aplicarea programării dinamice

1. Se analizeaza structura soluției: se stabilește modul in care soluția problemei depinde de soluțiile subproblemelor. Această etapă se referă de fapt la verificarea proprietății de substructură optimă și la identificarea problemei generice (forma generală a problemei inițiale și a fiecărei subprobleme).

2. Identificarea relației de recurență care exprimă legătura între soluția problemei și soluțiile subproblemelor. De regulă in relația de recurență intervine valoarea criteriului de optim.

3. Dezvoltarea relației de recurență. Relația este dezvoltată în manieră ascendentă astfel încât să se construiască tabelul cu valorile asociate subproblemelor

4. Construirea propriu-zisă a soluției – se bazează pe informațiile determinate în etapa anterioară.

Structura

- Ce este programarea dinamică ?
- Etapele principale în aplicarea programării dinamice
- Relații de recurență: dezvoltare ascendentă vs.dezvoltare descendentă
- Aplicații ale programării dinamice

Dezvoltarea relațiilor de recurență

Există două abordări principale:

- Ascendentă (bottom up): se pornește de la cazul particular și se generează noi valori pe baza celor existente.

- Descendentă (top down): valoarea de calculat se exprimă prin valori anterioare, care trebuie la rândul lor calculate. Această abordare se implementează de regulă recursiv (și de cele mai multe ori conduce la variante ineficiente – eficientizarea se poate realiza prin tehnica memoizării (cursul următor))

Dezvoltarea relațiilor de recurență

Exemplu 1. Calculul celui de al m-lea element al secvenței Fibonacci

$f_1=f_2=1$; $f_n=f_{n-1}+f_{n-2}$ for $n>2$

Abordare descendentă:

```
fib(m)
IF (m=1) OR (m=2) THEN
    RETURN 1
ELSE
   RETURN fib(m-1)+fib(m-2)
ENDIF
```

Eficiența:

$$T(m) = \begin{cases} 0 & \text{if } m<=2 \\ T(m-1)+T(m-2)+1 & \text{if } m>2 \end{cases}$$

T:

0 0 1 2 4 7 12 20 33 54 …

Fibonacci:

1 1 2 3 5 8 13 21 34 55 …

f_n apartine lui $O(phi^n)$,

phi=(1+sqrt(5))/2

Complexitate exponențială!

Dezvoltarea relatiilor de recurenta

Exemplu 1. Calculul celui de al m-lea element al secvenței Fibonacci

$f_1=f_2=1$; $f_n=f_{n-1}+f_{n-2}$ for n>2

Abordare ascendentă:

```
fib(m)
f[1]←1; f[2] ← 1;
FOR i ← 3,m DO
    f[i] ← f[i-1]+f[i-2]
ENDFOR
RETURN f[m]
```

Eficienta:

T(m)=m-2 => complexitate liniara

Obs: eficienta în timp este platită prin utilizarea unui spațiu adițional. Dimensiunea spațiului adițional poate fi semnificativ redusă

```
fib(m)
f1 ← 1; f2 ← 1;
FOR i ← 3,m DO
    f2 ← f1+f2; f1 ← f2-f1;
ENDFOR
RETURN f2
```

Dezvoltarea relatiilor de recurenta

Exemplu 2. Calculul coeficienților binomiali C(n,k) (combinări de n luate câte k)

$$C(n,k)=\begin{cases} 0 & \text{dacă } n<k \\ 1 & \text{dacă } k=0 \text{ sau } n=k \\ C(n-1,k)+C(n-1,k-1) & \text{altfel} \end{cases}$$

Abordare descendenta:

```
comb(n,k)
  IF (k=0) OR (n=k) THEN
     RETURN 1
  ELSE
     RETURN comb(n-1,k)+comb(n-1,k-1)
  ENDIF
```

Eficiența:

Dim pb: (n,k)

Op. dominantă: adunare

$T(n,k)=0$ dacă k=0 sau k=n

$T(n-1,k)+T(n-1,k-1)$

Nr adunări = nr noduri în arborele de apeluri recursive

$T(n,k) >= 2^{\min\{k,n-k\}}$

$T(n,k) \in \Omega(2^{\min\{k,n-k\}})$

Dezvoltarea relatiilor de recurenta

Exemplu 2. Calculul coeficienților binomiali C(n,k)

$$C(n,k)=\begin{cases} 0 & \text{daca } n<k \\ 1 & \text{daca } k=0 \text{ sau } n=k \\ C(n-1,k)+C(n-1,k-1) & \text{altfel} \end{cases}$$

Abordare descendentă: construirea triunghiului lui Pascal

	0	1	2	…	k-1	k
0	1					
1	1	1				
2	1	2	1			
…						
k	1			…		1
…						
n-1	1				C(n-1,k-1)	C(n-1,k)
n	1					C(n,k)

Dezvoltarea relatiilor de recurenta

Algoritm:

```
Comb(n,k)
FOR i←0,n DO
  FOR j ← 0,min{i,k} DO
    IF (j=0) OR (j=i) THEN
    C[i,j] ← 1
   ELSE
     C[i,j] ← C[i-1,j]+C[i-1,j-1]
   ENDIF
  ENDFOR
ENDFOR
RETURN C[n,k]
```

Eficienta:

Dim pb: (n,k)

Op. dominanta: adunarea

$T(n,k)=(1+2+\ldots+k-1) +(k+\ldots+k)$

$=k(k-1)/2+k(n-k+1)$

$T(n,k)\in\Theta(nk)$

Obs. Dacă trebuie calculat doar C(n,k) este suficient să se utilizeze un tablou cu k elemente ca spațiu suplimentar

Aplicații ale programării dinamice

Cel mai lung subșir strict crescător

Fie $a_1,a_2,\ldots,a_n$ o secvență. Să se determine cel mai lung subșir având proprietatea $a_{j1}<a_{j2}<\ldots<a_{jk}$ (un subșir strict crescator având numărul de elemente maxim).

Exemplu:

a = (2,5,1,3,6,8,2,10,4)

Subșiruri strict crescătoare de lungime 5 (lungimea maximă):

(2,5,6,8,10)

(2,3,6,8,10)

(1,3,6,8,10)

Cel mai lung subșir strict crescător

1. Analiza structurii solutiei.

Fie s=(a_{j1}, a_{j2},…,$a_{j(k-1)}$,a_{jk}) soluția optimă. Inseamnă că nu există nici un element în a[1..n] aflat după a_{jk} care să fie mai mare decât a_{jk} . In plus nu există element în șirul inițial având indicele cuprins între $j_{(k-1)}$ și j_k iar valoarea cuprinsă între valorile acestor elemente ale subșirului s (s nu ar mai fi soluție optimă). Aratăm că s'=(a_{j1}, a_{j2},…,$a_{j(k-1)}$) este soluție optimă pentru problema determinării celui mai lung subșir care se termină în $a_{j(k-1)}$. Pp ca s' nu este optimal. Rezultă că există un subșir s" de lg. mai mare. Adaugând la s" elementul a_{jk} s-ar obține o soluție mai bună decât s, implicând că s nu este optim. Se ajunge astfel la o contradicție, deci s' este solutie optima a subproblemei determinării unui subșir crescător care se termină în $a_{j(k-1)}$

Deci problema are proprietatea de substructura optima

Cel mai lung subșir strict crescător

2. Construirea unei relatii de recurenta

Fie B_i numarul de elemente al celui mai lung subsir strict crescator care se termina in a_i

$$B_i = \begin{cases} 1 & \text{if } i=1 \\ 1+ \max\{B_j \mid 1<=j<=i-1, a_j<a_i\} \end{cases}$$

Exemplu:

a = (2,5,1,3,6,8,2,10,4)

B = (1,2,1,2,3,4,2,5,3)

Cel mai lung subșir strict crescător

3. Dezvoltarea relației de recurență

$$B_i = \begin{cases} 1 & \text{if } i=1 \\ 1+\max\{B_j \mid 1<=j<=i-1,\ a_j<a_i\} \end{cases}$$

Complexitate: $\theta(n^2)$

```
calculB(a[1..n])
B[1]←1
FOR i ← 2,n DO
   max ← 0
   FOR j ← 1,i-1 DO
      IF a[j]<a[i] AND max<B[j]
           THEN max ← B[j]
      ENDIF
   ENDFOR
   B[i] ← max+1
ENDFOR
RETURN B[1..n]
```

Cel mai lung subșir strict crescător

4. Construirea solutiei

Se determina maximul lui B

Se construieste s succesiv pornind de la ultimul element

Complexitate: θ(n)

```
construire(a[1..n],B[1..n])
m ← 1
FOR i ← 2,n DO
  IF B[i]>B[m] THEN m ← i ENDIF
ENDFOR
k ← B[m]
s[k] ← a[m]
WHILE B[m]>1 DO
   i ← m-1
  WHILE a[i]>=a[m] OR B[i]<>B[m]-1 DO
       i ← i-1
  ENDWHILE
  m ← i;  k ← k-1;  s[k] ← a[m]
ENDWHILE
RETURN s[1..k]
```

Cel mai lung subșir strict crescător

```
calculB(a[1..n])
B[1]:=1;  P[1]:=0
FOR i:=2,n DO
   max:=0
   P[i]:=0
   FOR j:=1,i-1 DO
      IF a[j]<a[i] AND max<B[j]
      THEN max:=B[j]
              P[i]:=j
      ENDIF
   ENDFOR
   B[i]:=max+1
ENDFOR
RETURN B[1..n]
```

```
construire(a[1..n],B[1..n],P[1..n])
m:=1
FOR i:=2,n DO
   IF B[i]>B[m] THEN m:=i ENDIF
ENDFOR
k:=B[m]
s[k]:=a[m]
WHILE P[m]>0 DO
     m:=P[m]
     k:=k-1
     s[k]:=a[m]
ENDWHILE
RETURN s[1..k]
```

P[i] este indicele elementului ce il precede pe a[i] in subsirul optim.
Utilizarea lui P[1..n] simplifica construirea solutiei

Cel mai lung subșir comun

Fiind date două șiruri (secvențe) $a_1,\dots, a_n$ si $b_1,\dots,b_m$ să se determine un subșir $c_1,\dots c_k$ care satisface:

- Este subșir comun al șirurilor a și b, adică există $i_1,\dots,i_k$ si $j_1,\dots,j_k$ astfel incât

  $c_1=a_{i1}=b_{j1}$, $c_2=a_{i2}=b_{j2}$, … , $c_k=a_{ik}=b_{jk}$
- k este maxim (cel mai lung subșir comun)

Obs : această problemă este un caz particular al problemelor din bioinformatică unde se analizeaza similaritatea dintre doua șiruri de nucleotide (ADN) sau aminoacizi (proteine) – cu cât au un subșir comun mai lung cu atât sunt mai similare cele doua șiruri inițiale

Cel mai lung subsir comun

Exemplu:

a: 2 1 4 3 2

b: 1 3 4 2

Subșiruri comune:

1, 3

1, 2

4, 2

1, 3, 2

1, 4, 2

Variantă a problemei: determinarea celei mai lungi subsecvențe comune de elemente consecutive

Exemplu:

a: 2 1 3 4 5

b: 1 3 4 2

Subsecvențe comune:

1, 3

3, 4

1, 3, 4

Cel mai lung subsir comun

1. Analiza structurii unei soluții optime

Fie P(i,j) problema determinarii celui mai lung subșir comun al șirurilor a[1..i] și b[1..j]. Dacă a[i]=b[j] atunci soluția optimă conține acest element comun iar restul elementelor este reprezentat de soluția optimă a subproblemei P(i-1,j-1) (adica determinarea celui mai lung subșir comun al șirurilor a[1..i-1] respectiv b[1..j-1]. Dacă a[i]<>b[j] atunci soluția optimă coincide cu cea mai bună dintre soluțiile subproblemelor P(i-1,j) respectiv P(i,j-1).

2. Deducerea relatiei de recurență. Fie L(i,j) lungima soluției optime a problemei P(i,j). Atunci:

$$L[i,j]=\begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ 1+L[i-1,j-1] & \text{dacă } a[i]=b[j] \\ \max\{L[i-1,j],L[i,j-1]\} & \text{altfel} \end{cases}$$

Cel mai lung subșir comun

Exemplu:

a: 2 1 4 3 2

b: 1 3 4 2

$$L[i,j]=\begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ 1+L[i-1,j-1] & \text{dacă } a[i]=b[j] \\ \max\{L[i-1,j],L[i,j-1]\} & \text{altfel} \end{cases}$$

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	1
2	0	1	1	1	1
3	0	1	1	2	2
4	0	1	2	2	2
5	0	1	2	2	3

Cel mai lung subsir comun

Dezvoltarea relației de recurență:

$$L[i,j]=\begin{cases} 0 & \text{dacă } i=0 \text{ sau } j=0 \\ 1+L[i-1,j-1] & \text{dacă } a[i]=b[j] \\ \max\{L[i-1,j],L[i,j-1]\} & \text{altfel} \end{cases}$$

```
calcul(a[1..n],b[1..m])
FOR i:=0,n DO L[i,0]:=0 ENDFOR
FOR j:=1,m DO L[0,j]:=0 ENDFOR
FOR i:=1,n DO
  FOR j:=1,m DO
   IF a[i]=b[j]
   THEN  L[i,j]:=L[i-1,j-1]+1
   ELSE
     L[i,j]:=max(L[i-1,j],L[i,j-1])
   ENDIF
ENDFOR ENDFOR
RETURN L[0..n,0..m]
```

Cel mai lung subșir comun

Construirea solutiei (varianta recursiva):

```
Construire(i,j)
IF i>=1 AND j>=1 THEN
   IF a[i]=b[j]
  THEN
      construire(I-1,j-1)
      k:=k+1
      c[k]:=a[I]
  ELSE
      IF L[i-1,j]>L[i,j-1]
      THEN  construire(i-1,j)
      ELSE  construire (i,j-1)
ENDIF ENDIF ENDIF
```

Observatii:

- a, b, c si k sunt variabile globale
- Inainte de apelul functiei, variabila k se initializeaza (k:=0)
- Functia de construire se apeleaza prin

  construire(n,m)

Cursul urmator…

…alte aplicatii ale programarii dinamice

… tehnica memoizarii