Metoda programării dinamice (PD)

Metoda programării dinamice este o metodă de programare care se aplică problemelor a căror soluţie se poate construi dinamic în timp, adică deciziile care conduc la obţinerea rezultatului se pot lua pas cu pas, pe baza deciziilor de la pasul/paşii precedenţi.

De obicei, metoda programării dinamice este adecvată în cazul problemelor care solicită determinarea unui optim (minim sau maxim), în urma unui proces decizional care se desfăşoară în mai multe etape. , Astfel, se porneşte de la o stare iniţială şi la fiecare pas se ia o decizie care determină o nouă stare, până când se ajunge la soluţia finală, optimă.

Iniţiatorul metodei, profesorul şi cercetătorul Richard Bellman, a publicat în 1957 o carte cu titlul "Dynamic programming", în care a enunţat principiul optimalităţii:

O strategie are proprietatea că oricare ar fi starea iniţială şi decizia iniţială, deciziile rămase trebuie să constituie o strategie optimă privitoare la starea care rezultă din decizia anterioară.

Demonstrarea corectitudii algoritmului de rezolvare a unei probleme prin programare dinamică se face, de cele mai multe ori, prin inducţie matematică.

Metoda PD se poate aplica cu următoarele abordări:

- Metoda **înainte**: pentru rezovarea problemei se pleacă de la starea finală
- Metoda **înpoi**: pentru rezovarea problemei se pleacă de la starea iniţială
- Metoda **mixtă**: o combinaţie a primelor două

În problemele care pot fi rezolvate utilizând PD identificăm două proprietăţi:

a. **substructura optimală** – problema poate fi descompusă în subprobleme iar soluţiile optime ale subproblemelor determină soluţia optimă a problemei iniţiale (în acest caz s-ar putea aplica şi una dintre metodele Greedy sau Divide-et-Impera)

b. **subprobleme superpozabile** – subproblemele nu sunt independente ci se suprapun (Divide-et-Impera nu se poate aplica; redundanţa foarte mari a operaţiilor o face inutilizabilă, vezi Fibonacci); soluţiile subproblemelor se vor reţine într-un tablou (vector sau matrice)

Etapele rezolvării unei probleme utilizând metoda programării dinamice

1. identificarea subproblemelor
2. alegerea unei structuri care să reţină soluţiile subproblemelor
3. determinarea unei relaţii de recurenţă care să caracterizeze substructura optimală (dependenţa soluţiei subproblemei curente de soluţiile subproblemelor în care se descompune)
4. rezolvarea recurenţei în mod bottom-up (în ordinea crescătoare a dimensiunilor subproblemelor) pentru determinarea souţiei optime a problemei iniţiale

Aplicaţii

- Subşir crescător maximal
- Subşir comun maximal
- Sumă în triunghi
- Problema discretă a rucsacului

1. Subşir crescător maximal

Fie sirul A=(a1, a2,…an). Numim subşir al şirului A o succesiune de elemente din A, în ordinea în care acestea apar în A, nu neapărat pe poziţii consecutive. Pentru un sir dat, determinaţi un subşir crescător de lungime maximă.

Exemplu: A=(8,3,6,50,10,8,100,30,60,40,80)
o soluţie poate fi: (3,6,20,30,60,80)

Algoritm PD:

Etapa 1
Problema: determinarea celui mai lung subşir crescător al şirului (a1, a2,…an)
Subproblema: determinarea celui mai lung subşir crescător al şirului, care începe la poziţia k.

Etapa 2 – structura de date pentru memorarea soluţiilor parţiale:

- Construim vectorul **Lmax**(n),

Lmax[i]=lungimea celui mai lung subşir crescător care începe la poziţia i

- Construim vectorl **urm(n)**

urm[i]=indicele elementului care urmează lui a[i] în subşirul crescător maximal, dacă există următor sau -1 dacă nu există alt element după a[i]

(vetorul urm este necesar pentru reconstituirea subşirului)

Soluţia finală a problemei va fi maximul din vectorul Lmax iar subşirul căutat va începe la pozmax, poziţia la care apare acest maxim.

Etapa 3 - relaţia de recurenţă

- Lmax[n]=1, urm[n]=0;
- Lmax[i]=max{Lmax[j]+1, daca a[i]<=a[j]},
- urm[i]=indicele elementului pentru care s-a găsit maximul

Obs. Reconstituirea subşirului maximal pe baza vectorului urm şi a pozmax

```
for (i=pozmax; i!=-1; i=urm[i])
     fout<<a[i]<<' ';
```

2. Subşir comun maximal

Fie $X=(x_1,x_2,\ldots x_n)$ şi $Y=(y_1,y_2,\ldots y_m)$ două şiruri de numere întregi. Determinaţi subşirul comun de lungime maximă.

Algoritm PD:

Etapa 1 - Subproblema: determinarea suşirului comun maximal care începe pe poziţia **i** în şirul X şi pe poziţia **j** în şirul Y

Etapa 2 – structura de date pentru memorarea soluţiilor parţiale:

- Construim vectorul **Lmax**(n+1,m+1),

Lmax[i][j]=lungimea subţirului comun maximal care începe la poziţia i în X şi la poziţia j în Y

Soluţia finală a problemei va fi Lmax[n][m] iar subşirul comun va fi reconstituit cu ajutorul acestei matrici.

Etapa 3 - relaţia de recurenţă

- `Lmax[i][0]=0, i=1,...,n`
- `Lmax[0][j]=0, j=1,...,m`
- `Lmax[i][j]=Lmax[i-1][j-1]+1, daca X[i]=Y[j]`
- `Lmax[i][j]=max{Lmax[i-1][j], Lmax[i][j-1], daca X[i]!=Y[j]}`

3. Sumă în triunghi

Se consideră un triunghi format din n linii (1<n<=100), fiecare linie conţinând numere întregi din domeniul [1,99], ca în exemplul următor:

```
          7
       3     8
    8     1     0
 2     7     4     4
4     5     2     6     5
```

Să se determine cea mai mare sumă de numere aflate pe un drum între numărul de pe prima linie şi un număr de pe ultima linie. Fiecare număr din acest drum, cu excepţia primului, este situat imediat sub precedentul, la stânga sau la dreapta acestuia.
Să se afişeze suma maximă şi un drum pe care se poate obţine. Drumul este format din perechi de indici `lin col`, scrise câte o pereche pe o linie.

Algoritm PD:

Etapa 1 - Subproblema: determinarea sumei maxime care se poate obţine cu numerele aflate pe un drum care începe cu `a[i][j]` şi se termină pe ultima linie. (`a(n,n)`, matricea în care se memorează, sub diagonala principală, triunghiul de numere dat)

Etapa 2 – structura de date pentru memorarea soluţiilor parţiale:

- Construim matricea **`Smax`**`(n,n),`

`Smax``[i][j]=` suma maximă care se poate obţine cu numerele aflate pe un drum care începe cu `a[i][j]` şi se termină pe ultima linie

Soluţia finală a problemei va fi `Smax[1][1]` iar drumul pe care a fost construit va fi reconstituit cu ajutorul matricilor S şi a.

Etapa 3 - relaţia de recurenţă7

- `Smax[n][i]=a[n][i];`
- `Smax[i][j]=a[i][j]+max{Smax[i+1][j]+Smax[i+1][j+1] },`
  `i=n-1,...1; j=1,...,i`

4. Problema discretă a rucsacului

Un hoţ are un rucsac cu care poate transporta o greutate maxima Gmax. El îşi poate umple rucsacul cu obiecte selectate dintr-o mulţime de **n** obiecte pe care le-a găsit, pentru care cunoaşte greutatea $\mathbf{g_i}$ şi câştigul $\mathbf{c_i}$ pe care îl obţine selectând obiectul **i**. Un obiect poate fi luat doar în întregime.
Să se determine o modaliatate de umplere a rucsacului astfel încât câştigul total obţinut să fie maxim. Dacă nu există soluţii se va afişa IMPOSIBIL.

Datre de intrare: din rucsac.in se citesc n Gmax din prima linie apoi, din linia 2 se citesc cele n greutăţi iar din linia 3 cele n câştiguri ce se pot obţine alegând câte un obiect obiect.
În rucsac.out se va scrie pe prima linei câştigul maxim obţinut sau IMPOSIBIL. Dacă se poate, se vor scrie pe linia a doua, obiectele selectate pentru a obţine câştigul maxim, separate prin câte un spaţiu.

$1<=n<=100;\ 1<=Gmax<=300;\ g_i,\ c_i<=100$

rucsac.in	**rucsac.out**
7 100 80 10 20 10 100 30 55 8 20 5 1 10 30 20	29 1 2 4

Algoritm PD:

Etapa 1 - Subproblema: selectarea unor obiecte cu câştig maxim care permit încărcarea unui rucsac de capacitate S<=Gmax (suma greutăţilor obiectelor selectate este S)

Etapa 2 – structura de date pentru memorarea soluţiilor parţiale:

- Construim vectorul **Cmax**(Gmax),

Cmax[S]=câştigul maxim obţinut pentru selectarea unor obiecte cu suma greutăţilor S

- Construim matricea caracteristică **uz(Gmax+1)(n)**

uz[S][i]=1, dacă obiectul i a fost selectat în soluţia de greutate totală S şi **uz[S][i]**=0, altfel

Etapa 3 - relaţia de recurenţă

Pentru umplerea unui rucsac de capacitate S vom alege un obiect i, daca g[i]<=S. Capacitatea rămasă liberă, S-g[i], trebuie incarcata optimal, obtinand castigul maxim Cmax[S-g[i]]
Trebuie sa ne asiguram ca obiectul i nu a mai fost folosit in incarcarea optimala a capacitatii S-g[i], adica uz[s-g[i]][i]=0.
Recurenta

- Cmax[0]=0
- Cmax[S]=max{Cmax[S-g[i]]+c[i], daca g[i]<=S si uz[S-g[i]][i]=0}

Probleme propuse

campion.edu.ro/arhiva

- barbie
- lacusta
- codul
- tren1
- sant

Bibliografie

Cerchez, Emanuela, Şerban, Marinel, *Programarea în Limbajul C/C++ pentru liceu, Ed. Polirom, Iaşi, 2005*

http://www.campion.edu.ro/arhiva