Costin Badica
Stoica Spahiu Cosmin

Laborator 7 - Programarea Dinamică.

Programarea dinamică rezolvă problemele prin descompunerea lor în subprobleme şi prin combinarea rezolvărilor acestora (termenul „programare" se referă aici la o metodă tabulară). Spre deosebire de divide et impera, care considera că subproblemele sunt independente, programarea dinamică se aplică atunci când subproblemele nu sunt independente. Într-un astfel de caz, divide et impera ar efectua calcule redundante, rezolvând fiecare subproblemă ca şi când nu ar mai fi întâlnit-o. Programarea dinamică, însă, salvează rezultatul fiecărei subprobleme într-o tabelă, evitând astfel rezolvarea redundantă a aceleiaşi probleme.

Programarea dinamică se aplică în general problemelor de optimizare, atunci când dorim să determinăm rapid soluția optimă pentru o problemă. De fapt, aplicând această tehnică determinăm **una** din soluțiile optime, problema putând avea mai multe soluții optime.

Aplicarea acestei tehnici de programare poate fi descompusă în următoarea secvență de paşi:

1. Descoperirea structurii şi "măsurii" pe care o are o soluție optimă.
2. Definirea recursivă a valorii care caracterizează o soluție optimă.
3. Calcularea "de jos în sus" a acestei valori.
4. Construirea soluției optime pornind de la calculele efectuate anterior.

Primii trei paşi reprezintă baza programării dinamice. Dacă trebuie doar să aflăm doar valoarea soluției optime, ultimul pas nu mai este necesar. De aceea, pentru cazul în care dorim să determinăm şi soluția optimă poate fi nevoie de construirea unor structuri de date suplimentare.

Cel Mai Lung Subşir Ordonat.

O să exemplificăm această tehnică prin rezolvarea următoarei probleme: care este cel mai lung subşir ordonat crescător al unui şir oarecare. Este evident că „măsura" soluției optime este dată de lungimea subşirului. Să vedem în continuare cum putem calcula această lungime.

Să presupunem că primul element al acestui subşir se află pe poziția i şi că lungimea subşirului este x. Este evident că cel mai mare subşir ordonat crescător care se găseşte în dreapta lui i nu poate avea lungime mai mare decât x. În plus, elementul aflat pe poziția i este mai mic sau egal decât primul element al unui subşir ordonat crescător de lungime x – 1, aflat în dreapta lui i. Deci, pentru a calcula lungimea subşirului care începe cu elementul de pe poziția i, trebuie mai întâi să calculăm lungimea celui mai lung subşir din dreapta lui, subşir al cărui prim element este mai mare sau egal decât elementul de pe poziția i. Dar

această lungime o putem determina în acelaşi mod. Prin urmare avem următoarea formulă de recurență:

$$lungime(i) = \max_{j=i+1, v[i] \le v[j]}^{n-1} lungime(j) + 1$$

Pentru a calcula lungimea celui mai mare subşir ordonat crescător vom calcula mai întâi lungimile celor mai mari subşiruri care încep cu fiecare element al vectorului. Aceste valori le vom memora în vectorul *maxime*. Pentru ultimul element, această valoare este evidentă:

maxime[n - 1] = 1;

Conform formulei anterioare, pentru a calcula *maxime[i]* va trebui mai întâi să determinăm care este cel mai lung subşir din dreapta lui i care începe cu un element mai mare sau egal cu *v[i]*:

```
maxime[i] = 1;
for (j = i + 1; j < n; j++){
    if (v[i] <= v[j] && maxime[i] <= maxime[j]){
                maxime[i] = maxime[j] + 1;
    }
```

Dacă nu există nici un astfel de şir, lungimea celui mai lung subşir ordonat care începe cu elementul de pe poziția i va fi 1. Pentru a determina mai apoi şi elementele subşirului ordonat, mai folosim un vectorul *următorul*, care conține pe poziția i indexul primului element al subşirului folosit pentru calcularea lui *maxime[i]*:

```
maxime[i] = 1;
urmatorul[i] = -1;
for (j = i + 1; j < n; j++){
    if (v[i] <= v[j] && maxime[i] <= maxime[j]){
                maxime[i] = maxime[j] + 1;
                urmatorul[i] = j;
          }
}
```

Dacă ținem cont de faptul că vom începe calculele cu elementul de pe poziția n – 2, după ce inițializăm valorile pentru n – 1, avem:

```
maxime[n - 1] = 1;
urmatorul[n - 1] = -1;
     for(i = n - 2; i >= 0; i--){
           maxime[i] = 1;
           urmatorul[i] = -1;
           for (j = i + 1; j < n; j++){
               if (v[i] <= v[j] && maxime[i] <=
maxime[j]){
                        maxime[i] = maxime[j] + 1;
                        urmatorul[i] = j;
               }
           }
     }
```

Pentru şirul:	77	1	34	2	3	545	5464	32	34	5	654	43	22	42
Avem următorul *maxime*:	3	6	3	5	4	2	1	3	2	3	1	1	2	1
Şi *următorul*:	5	3	5	4	7	6	-1	8	10	12	-1	-1	13	-1

Acum pentru a afla lungimea celui mai mare subşir nu trebuie decât să determinăm maximul din vectorul *maxime*, în acelaşi timp determinând şi primul element al subşirului:

```
lungime = maxime[0];
inceput = 0;
for (i = 1; i < n; i++){
      if (lungime < maxime[i]){
                  lungime = maxime[i];
                  inceput = i;
      }
}
```

Acum, pentru a determina elementele subşirului ne vom folosi de vectorul *următorul*, pornind, firește, de la *început*:

```
subsir[0] = v[inceput];
     for (i = 1; i < lungime; i++){
           inceput = urmatorul[inceput];
           subsir[i] = v[inceput];
     }
```

Pentru exemplul anterior, algoritmul selectează următorul subşir: 1 2 3 32 34 654
Funcția completă care implementează algoritmul descris anterior este:

```
int celMaiLungSubsirOrdonat(int v[], int n, int subsir[])
{
      int lungime;
      int inceput;
      int i;
      int j;
      int *maxime = new int[n];
      int *urmatorul = new int[n];
      maxime[n - 1] = 1;
  urmatorul[n - 1] = -1;
      for(i = n - 2; i >= 0; i--){
            maxime[i] = 1;
            urmatorul[i] = -1;
            for (j = i + 1; j < n; j++){
                if (v[i] <= v[j] && maxime[i] <=
 maxime[j]){
                        maxime[i] = maxime[j] + 1;
                        urmatorul[i] = j;
                  }
            }
      }
      lungime = maxime[0];
      inceput = 0;
      for (i = 1; i < n; i++){
```

```
            if (lungime
< maxime[i]){
                  lungime = maxime[i];
                  inceput = i;
            }
      }
      subsir[0] = v[inceput];
      for (i = 1; i < lungime; i++){
            inceput = urmatorul[inceput];
            subsir[i] = v[inceput];
      }
      d    elete urmatorul;
      delete maxime;
      return lungime;
}
```

Probleme propuse.

1. Să se găsească cel mai lung subşir comun pentru două şiruri.
2. Să se găsească cel mai lung subşir ordonat, comun pentru două şiruri.
3. Să se găsească cea mai lungă secvență comună pentru două şiruri.
4. Să se găsească cea mai lungă secvență ordonată, comună pentru două şiruri.
5. Se citesc n numere naturale. Se cere să se tipărească cea mai mare sumă care se poate forma utilizân-d cele n numere naturale (fiecare nr participă o singură dată în calculul sumei) şi care se divide cu n, precum şi numerele care alcătuiesc această sumă.