Algoritmul lui LEE. TEORIE și APLICAȚII

Radu Vișinescu, Violeta Vișinescu

Cum trebuie prezentată varianta Lee a metodei programării dinamice? Cum se gândește și se rezolvă o problemă folosind această tehnică? Sunt întrebări la care acest articol caută să dea un posibil răspuns.

Vom analiza în articolul de față cazuri de folosire ale algoritmului *Lee*.

Încercăm să arătăm prin exemple cum se efectuează trecerea de algoritmul clasic de parcurgere în lățime, la prelucrarea dinamică a nodurilor unui graf, folosind de fiecare dată structura de date de tip *coadă*.

Articolul conține o prezentare teoretică a problemei generice urmată de patru aplicații, dintre care două inedite.

Parcurgerea în adâncime

Problema *generică* permite următorul enunț: *se consideră un graf neorientat conex cu N noduri, ale cărui muchii au costuri egale cu unitatea; definim distanța dintre două noduri ca fiind costul celui mai scurt lanț elementar care le unește; dându-se un nod de plecare I_0, să se parcurgă nodurile grafului în ordinea crescătoare a distanțelor față de nodul de plecare*.

Rezolvare

Memorăm graful prin matricea de adiacență *A* care conține *N* linii și *N* coloane. Notăm prin *D* vectorul de costuri (distanțe), iar prin *Părinte* vectorul de predecesori. Acești doi vectori au, fiecare, câte *N* elemente.

Se utilizează ca structură de date auxiliară o coadă *C* (păstrată sub forma unui vector) ale cărui elemente sunt noduri ale grafului. De asemenea, se păstrează referințele *P* și *U* către primul, respectiv ultimul, element al cozii.

Un ultim vector utilizat este *Sel* în care se memorează nodurile parcurse.

Varianta în pseudocod a algoritmului de rezolvare este următoarea:

```
Algoritm BF(A, D, Părinte):
 pentru I ← 1, N execută
  SEL[I] ← 0
 sfârșit pentru
 P ← 0
 U ← 0
 U ← U + 1
 C[U] ← I0
 Sel[I0] ← 1
 D[I0] ← 0
 Părinte[I0] ← 0
 cât timp P < U execută
  P ← P + 1
  NOD ← C[P]
  D ← D[P]
  scrie NOD, D[NOD]
  pentru J ← 1, N execută
   dacă (A[NOD, J] = 1) atunci
    dacă (SEL[J] = 0) atunci
     D[J] ← D + 1
     SEL[J] ← 1
     U ← U + 1
     C[U] ← J
     Părinte[J] ← NOD
    sfârșit dacă
   sfârșit dacă
  sfârșit pentru
 sfârșit cât timp
sfârșit algoritm
```

Drumul minim de la nodul I_0 la nodul *H* se obține folosind următoarea secvență:

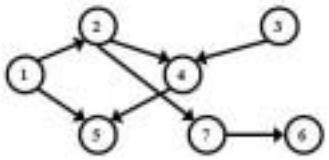
```
Algoritm Drum(H, Părinte):
 dacă (Părinte[H]<>0) atunci
  Drum(Părinte[H], Părinte)
 sfârșit dacă
 scrie H
sfârșit algoritm
```

Ordinul de complexitate al algoritmului este $O(N^2)$ dacă se folosește matricea de adiacență și devine $O(N + M)$ dacă se folosesc listele de adiacență.

Se poate demonstra că algoritmul vizitează nodurile în ordinea crescătoare a distanțelor de la nodul de plecare.

Algoritmul prezentat anterior poate fi aplicat și în cazul grafurilor orientate.

Exemplu
Pentru graful orientat din figura următoare parcurgerea *BF* pornind din vârful 1 atinge vârfurile în ordinea: 1, 2, 5, 4, 7, 6.
Vârful 3 nu poate fi atins. Se presupune că de fiecare dată este ales vârful cu eticheta cea mai mică.



Acest algoritm poate fi privit în același timp drept algoritmul *generic* pentru o clasă de probleme de optim, rezolvate printr-o variantă deosebită a programării dinamice.

Astfel, principiul de optimalitate poate fi enunțat în modul următor: Fie V un nod și $I_0, I_1, \ldots, I_{p-1}, I_p = V$ un drum de cost minim de la I_0 la V. În acest caz, $I_0, I_1, \ldots, I_{p-1}$ este drum de cost minim de la I_0 la I_{p-1}.

Ca urmare, se poate aplica metoda *înainte*. Vom determina drumuri de cost minim de la I_0 la orice alt nod al grafului. Substructura optimă este dată de subgraful parcurs la un moment dat. Optimul problemei este dat de unul dintre cele mai îndepărtate noduri.

În aplicații criteriul de optim se poate schimba, dar pricipiul de optimalitate și substructura rămâne aceeași. De asemenea, în permanență se folosește o coadă de noduri. De multe ori graful este dat implicit ca fiind indus de o matrice cu N linii și M coloane, iar în acest caz nu construim efectiv graful, ci operăm asupra matricei inițiale.

Aplicații

Următoarea problemă prezintă o variantă simplă de parcurgere în lățime cu anumite restricții. Problema a fost propusă spre rezolvare la ediția 2004 a *Olimpiadei Județene de Informatică*. Prezentăm enunțul într-o formă mai scurtă.

Romeo și Julieta

Romeo și *Julieta* se află într-un oraș a cărui hartă au reprezentat-o sub forma unei matrice cu N linii și M coloane, în matrice fiind marcate cu spațiu zonele prin care se poate trece (străzi lipsite de pericole) și cu *X* zonele prin care nu se poate trece.

Poziția lui *Romeo* este marcată cu *R*, iar poziția *Julietei* cu *J*. Ei se pot deplasa numai prin zonele care sunt marcate cu spațiu, din poziția curentă în oricare dintre cele opt poziții învecinate (pe orizontală, verticală sau diagonale).

Ei au hotărât că trebuie să aleagă un punct de întâlnire în care atât *Romeo*, cât și *Julieta* să poată ajunge în același timp, plecând de acasă. Ei estimează timpul necesar pentru a ajunge la întâlnire prin numărul de elemente din matrice care constituie drumul cel mai scurt de acasă până la punctul de întâlnire.

Scrieți un program care să determine o poziție pe hartă la care *Romeo* și *Julieta* pot să ajungă în același timp. Dacă există mai multe soluții, programul trebuie să determine o soluție pentru care timpul este minim.

Date de intrare

Fișierul de intrare **`rj.in`** conține pe prima linie numerele naturale N și M, care reprezintă numărul de linii și respectiv de coloane ale matricei, separate prin spațiu.

Fiecare dintre următoarele N linii conține M caractere (care pot fi doar *R*, *J*, *X* sau spațiu) reprezentând matricea.

Date de ieșire

Fișierul de ieșire **`rj.out`** va conține o singură linie pe care sunt scrise trei numere naturale separate prin câte un spațiu *tmin*, *x* și *y*, având semnificația: (*x*, *y*) reprezintă punctul de întâlnire (*x* - numărul liniei, *y* - numărul coloanei), iar *tmin* reprezintă timpul minim în care *Romeo* (respectiv *Julieta*) ajunge la punctul de întâlnire.

Restricții și precizări

- $2 \le N, M \le 100$;
- liniile și coloanele matricei sunt numerotate începând cu 1;
- pentru datele de test există întotdeauna soluție.

Exemplu

`rj.in`
```
5 8
XXR   XXX
 X   X   X
J X X   X
        XX
XXX   XXXX
```

`rj.out`
```
4 4 4
```

Explicație

Traseul lui *Romeo* poate fi: (1, 3), (2, 4), (3, 4), (4, 4). Așadar, timpul necesar lui *Romeo* pentru a ajunge de acasă la punctul de întâlnire este 4.

Traseul *Julietei* poate fi: (3, 1), (4, 2), (4, 3), (4, 4). Timpul necesar *Julietei* pentru a ajunge de acasă la punctul de întâlnire este, de asemenea, 4.

În plus (4, 4) este punctul cel mai apropiat de ei cu această proprietate.

Rezolvare

Pornim alternativ din pozițiile lui *Romeo* și a *Julietei*. La fiecare pas K generăm în două cozi separate punctele din matrice aflate la distanța K de pozițiile inițiale pentru *Romeo* și *Julieta*. Comparăm cele două liste de puncte. Dacă găsim un punct comun algoritmul se încheie, dacă nu generăm punctele aflate la distanțe $K + 1$ ș.a.m.d.

În alte situații de optim este util să folosim o coadă cu priorități. Este și cazul următoarei probleme, propuse la ediția 2003 a *Olimpiadei Județene de Informatică*.

Taxe

Într-o țară în care corupția este în floare și economia la pământ, pentru a obține toate aprobările necesare în scopul demarării unei afaceri, investitorul trebuie să treacă prin mai multe camere ale unei clădiri în care se află birouri.

Clădirea are un singur nivel în care birourile sunt lipite unele de altele formând un caroiaj pătratic de dimensiune $n \times n$. Pentru a facilita accesul în birouri, toate camerele vecine au uși între ele. În fiecare birou se află un funcționar care pretinde o taxă de

trecere prin cameră (taxă care poate fi, pentru unele camere, egală cu 0).

Investitorul intră încrezător prin colțul din stânga-sus al clădirii (cum se vede de sus planul clădirii) și dorește să ajungă în colțul opus al clădirii, unde este ieșirea, plătind o taxă totală cât mai mică.

Știind că el are în buzunar *S* euro și că fiecare funcționar îi pretinde taxa imediat ce investitorul intră în birou, se cere să se determine dacă el poate primi aprobările necesare și, în caz afirmativ, care este suma maximă de bani care îi rămâne în buzunar la ieșirea din clădire.

Date de intrare

Fișierul de intrare **taxe.in** conține pe prima linie numerele *S* și *n*, despărțite printr-un spațiu, iar pe următoarele *n* linii câte *n* numere separate prin spații care reprezintă taxele cerute de funcționarii din fiecare birou.

Date de ieșire

Fișierul de ieșire **taxe.out** conține o singură linie pe care se află numărul maxim de euro care îi rămân în buzunar sau valoarea -1 dacă investitorului nu-i ajung banii pentru a obține aprobarea.

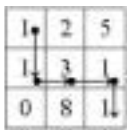
Restricții și precizări

- $3 \le N \le 100$;
- $1 \le S \le 10000$;
- valorile reprezentând taxele cerute de funcționarii din birouri sunt numere naturale, o taxă nedepășind valoarea de 200 de euro;
- la încheierea programului nu se va solicita apăsarea unei taste.

Exemplu

taxe.in

```
10 3
1 2 5
1 3 1
0 8 1
```



taxe.out

```
3
```

Rezolvare

Folosim o coadă circulară cu priorități, iar pe fiecare nivel al cozii păstrăm trei valori: linia, coloana și valoarea minimă plătită până în acel punct. Particularitatea constă în faptul că prin fiecare celulă a matricei trecem o singură dată, iar la fiecare pas continuăm procesarea celulei din vârful cozii.

De asemenea, folosim o matrice auxiliară de elemente deja selectate, bordată inițial cu valoarea ***true***.

Ordinul de complexitate al algoritmului este $O(N^4)$, deoarece dimensiunea maximă a cozii este N^2 și se execută maxim N^2 introduceri.

Varianta în pseudocod este următoarea:

```
subalgoritm init
 pentru J ← 0, N+1 execută
  sel[0,J] ← adevărat
  sel[N+1,J] ← adevărat
 sfârșit pentru
 pentru I ← 1, N execută
  sel[I,0] ← adevărat
  sel[I,N+1] ← adevărat
 sfârșit pentru
sfârșit subalgoritm

subalgoritm insert(A,B,VALOARE)
 U ← U + 1
 dacă U > 2001 atunci
  U ← 1
 sfârșit dacă
 CAPAT ← P - 1
 dacă CAPAT=0 atunci
  CAPAT ← N
 sfarșit dacă
 dacă U = 1 atunci
  POZ ← 2001
 altfel
  POZ ← U - 1
 sfârșit dacă
 cât timp (POZ ≠ CAPAT) și
    (VALOARE > Q[3,POZ]) execută
  POZ2 ← POZ + 1
  dacă POZ2 > 2001 atunci
   POZ2 ← 1
  sfârșit dacă
  Q[1,POZ2] ← Q[1,POZ]
  Q[2,POZ2] ← Q[2,POZ]
  Q[3,POZ2] ← Q[3,POZ]
  POZ ← POZ - 1
  dacă POZ = 0 atunci
   POZ ← 2001
  sfârșit dacă
 sfârșit cât timp
 POZ ← POZ + 1
 dacă POZ > 2001 atunci
  POZ ← 1
 sfârșit dacă
 Q[1,POZ] ← A
 Q[2,POZ] ← B
 Q[3,POZ] ← VALOARE
sfârșit subalgoritm

algoritm Lee
 P ← 0
 U ← 0
 U ← U + 1
 dacă U > 2001 atunci
  U ← 1
 sfârșit dacă
 Q[1,U] ← 1
 Q[2,U] ← 1
 Q[3,U] ← S - A[1,1]
 SEL[1,1] ← adevărat
 OK ← adevărat
 cât timp OK execută
  P ← P+1
  dacă P > 2001 atunci
   p ← 1
  sfârșit dacă
  L ← Q[1,P]
  C ← Q[2,P]
  V ← Q[3,P]
  dacă nu SEL[L-1,C] atunci
   insert(L-1,C,V-A[L-1,C])
   dacă (L - 1 = N) și (C = N)
                          atunci
    OK ← fals
   sfârșit dacă
   dacă nu OK atunci
    dacă (V - A[L-1,C]) ≥ 0
                          atunci
     scrie V-A[L-1,C]
    altfel
     scrie -1
    sfârșit dacă
   sfârșit dacă
   SEL[L-1,C] ← adevărat;
  sfârșit dacă
  dacă OK atunci
   dacă nu SEL[L+1,C] atunci
    insert(L + 1,C,V -
                    A[L+1,C]);
    dacă (L + 1 = N)and(C = N)
                          atunci
     OK ← fals
    sfârșit dacă
    dacă nu OK atunci
     dacă (V - A[L+1,C]) >= 0
                          atunci
```

```
        scrie V - A[L+1,C]
       altfel
        scrie -1
       sfârșit dacă
      sfârșit dacă
      SEL[L+1,C] ← adevărat
    sfârșit dacă
   sfârșit dacă
   dacă OK atunci
    dacă nu SEL[L,C-1] atunci
     insert(L,C - 1,
                  V - A[L,C-1])
     dacă (L = N) și (C-1 = N)
                          atunci
      OK ← fals
     sfârșit dacă
     dacă nu OK atunci
      dacă (V-A[L,C-1]) >= 0
                          atunci
       scrie V-A[L,C-1]
      altfel
       scrie -1
      sfârșit dacă
     sfârșit dacă
     SEL[L,C-1] ← adevărat
    sfârșit dacă
   sfârșit dacă
   dacă OK atunci
    dacă nu SEL[L,C+1] atunci
     insert(L,C+1,V-A[L,C+1])
     dacă (L = N) and (C + 1 = N)
                          atunci
      OK ← fals;
     sfârșit dacă
     dacă nu OK atunci
      dacă (V - A[L,C+1]) >=0
                          atunci
       scrie V - A[L,C+1]
      altfel
       scrie -1
      sfârșit dacă
     sfârșit dacă
     SEL[L,C+1] ← adevărat
    sfârșit dacă
   sfârșit dacă
  sfârșit cât timp
sfârșit algoritm
```

Există cazuri în care algoritmul lui *Lee* trece de mai multe ori prin celulele suprafeței. Este și situația următoarei probleme.

Alpinist

O regiune muntoasă este codificată printr-o matrice patratică A de dimensiune N cu elemente numere naturale, în care înălțimea zonei de coordonate (I, J) este valoarea $A[I, J]$.

Un alpinist dorește să parcurgă diferite trasee. El poate pleca din orice zonă, la fiecare etapă a traseului se poate deplasa o poziție pe una din direcțiile *sus*, *jos*, *stânga*, *dreapta*, fără a ieși din regiunea dată și cu condiția ca de fiecare dată să aleagă o nouă zonă cu înălțimea strict mai mare decât cea anterioară.

Determinați care este lungimea maximă (în număr de etape) pe care o poate avea un astfel de traseu.

Date de intrare

Fișierul de intrare **alpinist.in** conține pe prima linie valoarea N.

Pe fiecare dintre următoarele N linii se află câte N numere întregi, reprezentând în ordine înălțimile zonelor de pe respectivele linii ale matricei A separate prin cate un spațiu.

Date de ieșire

Fișierul de ieșire **alpinist.out** va conține, pe o singură linie, valoarea MAX reprezentând lungimea maximă cerută.

Restricții și precizări:

- $1 \le N \le 100$;
- $0 \le A[I, J] \le 10000$,

Exemplu:

alpinist.in

```
4
4 5 8 10
3 10 1 12
2 20 15 14
10 21 21 4
```

alpinist.out

```
10
```

Explicație

Traseul cel mai lung este format din valorile `2 3 4 5 8 10 12 14 15 20 21`.

Rezolvare

Se folosește o matrice auxiliară L, unde $L[I, J]$ va reține în final lungimea maximă a unui traseu care are punct final celula de coordonate (I, J).

La limită, putem pleca din orice celulă, dar pentru a optimiza algoritmul este suficient să plecăm din acele celule pentru care toți vecinii lor au cote mai mari sau egale.

Algoritmul cozii se repetă pentru fiecare astfel de celulă. În final calculăm valoarea maximă a matricei L care ne va da rezultatul problemei.

Varianta în pseudocod este următoarea:

```
subalgoritm init
 pentru i ← 1, N execută
  pentru J ← 1, N execută
   L[I,J] ← -1
  sfârșit pentru
 sfârșit pentru
 K ← 0
 pentru I ← 1, N execută
  pentru J ← 1, N execută
   OK ← adevărat
   dacă (I > 1) și (A[I-1,J] <
                A[I,J]) atunci
      OK ← fals
   sfârșit dacă
   dacă (I < N) și (A[I+1,J] <
                A[I,J]) atunci
    OK ← fals
   sfârșit dacă
   dacă (J > 1) și (A[I,J-1] <
                A[I,J]) atunci
    OK ← fals
   sfârșit dacă
   dacă (J<N) și (A[I,J+1] <
                A[I,J]) atunci
      OK ← fals
   sfârșit dacă
   dacă OK atunci
    L[I,J] ← 0
    K ← K + 1
    X[1,K] ← I
    X[2,K] ← J
   sfârșit dacă
  sfârșit pentru
 sfârșit pentru
sfârșit subalgoritm

subalgoritm intr(C,D)
 U ← U + 1
 dacă U > 2000 atunci
  U ← 1
 sfârșit dacă
 Q[1,U] ← C
 Q[2,U] ← D
sfârșit subalgoritm
```

```
subalgoritm extr(C,D)
 P ← P+1
 dacă P > 2000 atunci
  P ← 1
 sfârșit dacă
 C ← Q[1,P]
 D ← Q[2,P]
sfârșit subalgoritm

subalgoritm Lee(X,Y)
 P ← 0, U ← 0
 intr(X,Y)
 cât timp P ≠ U execută
  extr(X,Y)
  dacă X > 1 atunci
   dacă A[X-1,Y] > A[X,Y]
                        atunci
    dacă L[X-1,Y] < L[X,Y]+1
                        atunci
     L[X-1,Y] ← L[X,Y]+1
     intr(X-1,Y)
    sfârșit dacă
   sfârșit dacă
  sfârșit dacă
  dacă X < N atunci
   dacă A[X+1,Y] > A[X,Y]
                        atunci
    dacă L[X+1,Y] < L[X,Y]+1
                        atunci
     L[X+1,Y] ← L[X,Y]+1
     intr(X+1,Y)
    sfârșit dacă
   sfârșit dacă
  sfârșit dacă
  dacă Y > 1 atunci
   dacă A[X,Y-1] > A[X,Y]
                        atunci
    dacă L[X,Y-1] < L[X,Y]+1
                        atunci
     L[X,Y-1] ← L[X,Y] + 1
     intr(X,Y-1)
    sfârșit dacă
   sfârșit dacă
  sfârșit dacă
  dacă Y < N atunci
   dacă A[X,Y+1] > A[X,Y]
                        atunci
    dacă L[X,Y+1] < L[X,Y]+1
                        atunci
     L[X,Y+1] ← L[X,Y]+1
     intr(X,Y+1)
    sfârșit dacă
   sfârșit dacă
  sfârșit dacă
 sfârșit cât timp
sfârșit subalgoritm
```

```
subalgoritm scriere
 MAX ← -1
 pentru I ← 1, N execută
  pentru J ← 1, N execută
   dacă L[I,J] > MAX atunci
    MAX ← L[I,J]
   sfârșit dacă
  sfârșit pentru
 sfârșit pentru
 scrie MAX
sfârșit subalgoritm

algoritm main
 init
 pentru I ← 1, K execută
  Lee(X[1,I],X[2,I])
 sfârșit pentru
 scriere
sfârșit algoritm
```

În final propunem spre rezolvare următoarea problemă:

Speedy

Speedy Gonzales este fermier în *Texas*. El are un *ranch* situat într-o zonă denivelată care se poate reprezenta printr-o matrice A cu N linii și M coloane, elementul $A[I, J]$ reprezentând înălțimea zonei de coordonate (I, J). *Speedy* nu are astâmpăr.

El pornește din zona inițială de coordonate (A, B) și se poate deplasa pe cele patru direcții: *sus*, *jos*, *stânga*, *dreapta*. Dacă prin deplasare într-o celulă învecinată *Speedy* urcă K unități de nivel, atunci viteza lui scade cu $2 \cdot K$ unități de viteză. Dacă prin de-plasarea într-o celulă învecinată el coboară K unități de nivel, atunci viteza sa crește cu K unități de viteză. În ambele situații, din cauza oboselii viteza scade constant cu o unitate.

Deoarece nu vrea să se oprească, el nu se va putea deplasa într-o celulă vecină decât dacă viteza rămâne strict pozitivă.

Cunoscând viteza inițială V a lui *Speedy*, determinați viteza maximă cu care poate ajunge într-o zonă aflată pe marginea *ranch*-lui.

Date de intrare

Fișierul de intare **speedy.in** conține pe prima linie valorile N, M, A, B și V, separate prin câte un spațiu.

Pe fiecare din următoare N linii se află câte M valori reprezentând înălțimile de pe linia corespunzătoare din matrice separate prin câte un spațiu.

Date de ieșire

Fișierul de ieșire **speedy.out** va conține, pe o singură linie, valoarea MAX care reprezintă viteza maximă, sau valoarea `-1` dacă *Speedy* nu poate ajunge la marginea *ranch*-ului.

Restricții și precizări

- $1 \le N, M \le 100$;
- $1 \le V \le 2000$;
- $1 \le A \le N$;
- $1 \le B \le M$;
- $0 \le A[I, J] \le 1000$;
- toate valorile sunt numere întregi.

Exemplu

speedy.in

```
4 4 3 3 2000
10 5 0 1000
25 300 500 1000
50 2 0 1000
1000 1000 1000 1000
```

speedy.out

```
1944
```

Explicație

Drumul optim este: $(3, 3) \to (3, 2) \to (3, 1) \to (2, 1) \to (1, 1) \to (1, 2) \to (1, 3)$,

Încheiere

Lăsăm cititorului plăcerea de a rezolva ultima problemă. Aceasta poate fi abordată în maniera problemei ***Taxe***, folosind o coadă cu priorități în care vârful cozii reține în permanență coordonatele și viteza celulei de viteză maximă. O altă posibilitate este aplicarea variantei din problema ***Alpinist***, în care trecem de mai multe ori printr-o celulă (ori de câte ori viteza mai poate crește).

Bibliografie

1. Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, *Computer Algorithms*, Computer Science Press, 1998.
2. ***, probleme propuse la concursurile naționale de programare.