Algoritmul FILL. TEORIE și APLICAȚII

Radu Vișinescu, Violeta Vișinescu

Articolul prezintă aspectele teoretice, cu implemetări recursive și iterative ale algoritmului FILL, urmate de o serie de aplicații și adaptări în probleme de concurs.

Algoritmul *FILL* este privit cu indulgentă de mulți programatori, fiind considerat un caz banal. Este adevărat, dar utilizarea sa presupune atunci stăpânirea mecanismului recursivității.

Pe de altă parte, considerăm că acest algoritm este sugestiv în ceea ce privesc aspectele lui teoretice si legăturile sale cu tehnica *backtracking* și cu parcurgerea în adâncime a unui graf.

Considerații teoretice

Algoritmul *FILL* realizează umplerea (colorarea) unei suprafețe închise. Considerăm o matrice ale cărei elemente pot avea una dintre valorile 1 și 0 cu semnificația: 1 reprezintă un perete, iar 0 o zonă liberă.

Pornind dintr-o zonă de valoare 0, algoritmul trebuie să coloreze cu valoarea 2 toate celelalte celule în care se poate ajunge mergând un număr finit de pași pe direcțiile: sus, jos, stânga, dreapta.

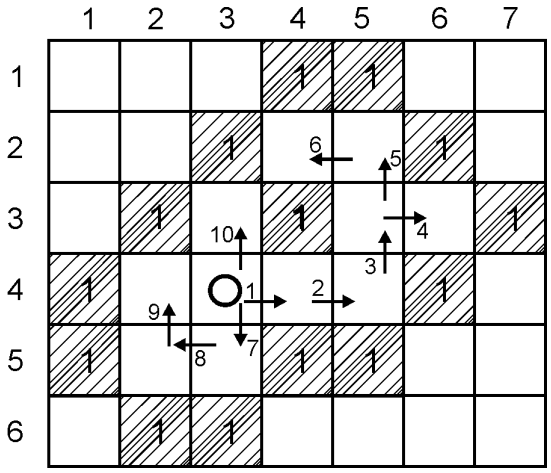
Pentru simplitate, considerăm că matricea este bordată cu elemente a căror valoare este 1.

Algoritmul recursiv este următorul:

```
algoritm fill(X, Y)
  dacă A_XY = 0 atunci

    A_XY ← 2

    fill(X, Y + 1)
    fill(X + 1, Y)
    fill(X, Y - 1)
    fill(X - 1, Y)

  sfârșit dacă
sfârșit algoritm
```

Putem prezenta același algoritm într-o formă care pune în evidență caracterul de *backtracking* al acestuia:

```
DX ← (0, 1, 0, -1)
DY ← (1, 0, -1, 0)

algoritm fill2(X, Y)

  dacă A_XY = 0 atunci
    A_XY ← 2
    pentru I ← 1, 4 execută
      fill2(X + DX_I, Y + DY_I)
    sfârșit pentru
  sfârșit dacă
sfârșit algoritm
```



Dacă privim matricea ca fiind reprezentarea unui graf neorientat în care nodurile sunt reprezentate de celulele matricei, iar muchiile sunt reprezentate de perechi de celule adiacente de valoare 0, algoritmul *FILL* reprezintă o particularizare a operației de parcurgere în adâncime a grafurilor (*DF*).

În figura anterioară este prezentată succesiunea acestor apeluri pentru un caz particular, în situația în care ordinea apelurilor recursive corespunde direcțiilor *est*, *sud*, *vest*, *nord*.

În cursul executării algoritmului prin fiecare celulă a matricei se trece o singură dată (un singur apel). Din acest motiv, deși se comportă ca un algoritm de căutare cu revenire (*backtracking*), ordinul de complexitate al algoritmului este $O(N \cdot M)$, unde N și M reprezintă dimensiunile matricei.

Ordinul de complexitate este polinomial, spre deosebire de majoritatea cazurilor în care este utilizată tehnica *backtracking*.

Prezentăm acum o variantă iterativă de implementare a acestui algoritm.

```
algoritm fill_iterativ(X,Y)
  dacă A_XY = 0 atunci

    K ← 1
    A_XY ← 2
    ST_1K ← X
    ST_2K ← Y

    cât timp K > 0 execută
      continuă ← adevărat
      X ← ST_1K
      Y ← ST_2K

      cât timp continuă
                        execută
        continuă ← fals

        dacă X > 1 și A_X-1,Y = 0
              și nu continuă
                        atunci

          continuă ← adevărat
          A_X-1,Y ← 2
          K ← K + 1
          ST_1K ← ST_1,K-1 - 1
          ST_2K ← ST_2,K-1
        sfârșit dacă

        dacă X < N și A_X+1,Y = 0
              și nu continuă
                        atunci
          continuă ← adevărat
          A_X+1,Y ← 2
          K ← K + 1
          ST_1K ← ST_1,K-1 + 1
          ST_2K ← ST_2,K-1
        sfârșit dacă

        dacă Y > 1 și A_X,Y-1 = 0
              și nu continuă
                        atunci

          continuă ← adevărat
          A_X,Y-1 ← 2
          K ← K + 1
          ST_1K ← ST_1,K-1
          ST_2K ← ST_2,K-1 - 1
        sfârșit dacă

        dacă Y < M și A_X,Y+1 = 0
              și nu continuă
                        atunci

          continuă ← adevărat
          A_X,Y+1 ← 2
          K ← K + 1
          ST_1K ← ST_1,K-1
          ST_2K ← ST_2,K-1 + 1
        sfârșit dacă

      sfârșit cât timp
      K ← K - 1
    sfârșit cât timp
  sfârșit dacă
sfârșit algoritm
```

Aplicații

Există o serie de probleme care pot fi privite drept aplicații și adaptări ale algoritmului *FILL*.

Prezentăm în continuare patru astfel de probleme.

Labirint

Un labirint este codificat printr-o matrice binară cu N linii și M coloane.

Considerăm că singura ieșire din labirint este celula de coordonate (N, M).

Fiind dată o celulă oarecare din labirint, ale cărei coordonate sunt (X, Y), să se determine un drum de la aceasta și până la ieșirea din labirint. Nu este neapărat necesar ca drumul determinat să aibă lungimea minimă.

Fotografie

O fotografie alb-negru este codificată printr-o matrice binară cu N linii și M coloane.

Un obiect este format din elemente cu valoarea 1 care se învecinează pe linii, pe coloane sau pe diagonale.

Determinați numărul de obiecte din fotografie.

Bilă

Se consideră o matrice A cu N linii și M coloane ale cărei elemente sunt numere naturale. Elementele acestei matrice reprezintă înălțimile unor zone.

O bilă pornește dintr-o celulă a matricii și se poate deplasa la fiecare pas în celulele învecinate pe orizontală sau verticală dacă și numai dacă celula destinație are o înălțime mai mică sau egală decât cea din care a plecat.

Astfel, după un anumit număr de pași, bila poate ajunge pe o margine a matricei.

Să se determine perechile de coordonate ale celulelor din care bila ***nu*** poate ajunge la margine.

Restricție:
• $1 \le N, M \le 100$.

Exemplu:
$N = 4$ $M = 4$
```
1 6 9 3
9 6 2 3
5 4 9 4
1 9 8 4
```

Răspuns corect:
```
2 3
3 2
```

Această problemă a fost propusă spre rezolvare elevilor de clasa a VIII-a la ediția 2005 a Olimpiadei

Locale de Informatică din județul *Prahova*.

Soluție
Prezentăm în continuare soluția în *pseudocod*. Este utilizată o matrice auxiliară *B* ale cărei elemente sunt valori logice.

```
subalgoritm fill(L, C)
  dacă nu B_LC atunci
    B_LC ← adevărat

    dacă L < N și A_L+1,C ≥ A_LC
                          atunci
      fill(L + 1, C)
    sfârșit dacă

    dacă L > 1 și A_L-1,C ≥ A_LC
                          atunci
      fill(L - 1, C)
    sfârșit dacă

    dacă C < M și A_L,C+1 ≥ A_LC
                          atunci
      fill(L, C + 1)
    sfârșit dacă

    dacă C > 1 și A_L,C-1 ≥ A_LC
                          atunci
      fill(L, C - 1)
    sfârșit dacă

  sfârșit dacă
sfârșit subalgoritm

algoritm bila

  // inițializarea matricei B
  pentru I ← 1, N execută
    pentru J ← 1, M execută
      B_IJ ← fals
    sfârșit pentru
  sfârșit pentru

  // "umplerea" matricei B
  pentru toate celulele (L, C) de
          pe marginea matricei B
            și nu B_LC execută

    fill(L, C)
  sfârșit pentru

  // afișarea rezultatelor
  pentru I ← 1, N execută
    pentru J ← 1, M execută
      dacă nu B_IJ atunci
        scrie I, J
      sfârșit dacă

    sfârșit pentru
  sfârșit pentru

sfârșit algoritm
```

Ordinul de complexitate al acestui algoritm este $O(N \cdot M)$.

Operațiune militară

Pentru a evita atacurile teroriștilor, forțele armate implicate în conflictul din *Kari* au hotărât să se retragă în cazematele construite pe terenul de luptă, codificat printr-o matrice binară cu *N* linii și *M* coloane, în care valoarea 1 reprezintă un perete, iar valoarea 0 reprezintă spațiu liber. Cazematele sunt zonele de pe teren care sunt în întregime înconjurate de pereți.

Va trebui să scrieți un program care, pe baza configrației terenului de luptă, determină:
- numărul cazematelor de pe terenul de luptă;
- numărul soldaților care se pot adăposti în cazemate, considerând că fiecare soldat ocupă, în caz de necesitate, o singură locație din matrice, care nu este perete și care se află în interiorul cazematei;
- cunoscându-se numărul *K* al detașamentelor și numărul soldaților din fiecare detașament, trebuie determinată, dacă este posibil, o distribuție a soldaților, astfel încât detașamentele să nu se scindeze;
  - ♦ în cazul în care acest lucru nu este posibil, se va afișa mesajul `imposibil`;
  - ♦ dacă există o astfel de posibilitate, atunci, pentru fiecare detașament, se va afișa un triplet de forma: numărul de ordine al detașamentului, linia și coloana unei poziții libere din cazemată în care se vor distribui soldații acelui detașament.

Trebuie menționat faptul că în nici una dintre cazematele de pe câmpul de luptă nu pot intra mai multe detașamente.

Datele de intrare se citesc din fișierul **`input.txt`**, astfel:
- pe prima linie se află numerele *N*, *M* și *K*, separate prin câte un spațiu;
- pe următoarele *N* linii se află matricea binară, cu elementele separate prin câte un spațiu;
- pe ultima linie se află *K* numere, separate prin câte un spațiu, corespunzătoare numărului de soldați din fiecare detașament.

Datele de ieșire se vor afișa în fișierul **`output.txt`**, pe trei linii. Fiecare linie va corespunde uneia dintre cele trei cerințe.

Exemplu:

`input.txt`
```
4 5 3
0 0 0 0 0
0 1 1 0 0
1 0 0 1 0
0 1 1 1 0
2 3 1
```

`output.txt`
```
1
2
Imposibil
```

Această problemă, într-o formă puțin diferită, a fost propusă spre rezolvare elevilor de clasa a X-a la ediția 2005 a Olimpiadei Locale de Informatică din județul *Prahova*.

Încheiere

Ne exprimăm speranța că acest articol se va dovedi util pentru clarificarea mecanismului algoritmului *FILL*.

De asemenea, propunem cititorilor rezolvarea aplicațiilor enunțate în cadrul acestui articol.

Bibliografie

1. **Radu Vișinescu**, Bazele programării - prin exerciții și probleme - , Editura *Petrion*, 2002;
2. ***, probleme propuse la concursurile de programare.