Algoritmul lui Lee

(Categoria *Algoritmi*, Autor *Simoiu Robert*)

* **Conţinut:**
* [Introducere](http://www.infoarena.ro/algoritmul-lee#sectiune1)
* [Prezentare](http://www.infoarena.ro/algoritmul-lee#sectiune2)
* [Aplicaţia #1](http://www.infoarena.ro/algoritmul-lee#sectiune3)
* [Aplicaţia #2](http://www.infoarena.ro/algoritmul-lee#sectiune4)
* [Aplicaţia #3](http://www.infoarena.ro/algoritmul-lee#sectiune5)
* [Aplicaţia #4](http://www.infoarena.ro/algoritmul-lee#sectiune6)

Introducere

În continuare vom prezenta *algoritmul lui Lee*, pentru cei care nu ştiu, este identic cu *parcurgerea în lăţime* doar ca e aplicat pe o grila, nu pe un graf oarecare. Este eficient, având o complexitate deO(M\*N), şi frecvent utilizat. Acesta determină drumul minim de ieşire dintr-un labirint, sau în probleme asemănătoare.

Prezentare

*Algoritmul lui Lee* presupune doi paşi importanţi:

1. Primul şi poate cel mai important pas este folosirea unei **Cozi**, sub forma unui vector de structuri (de preferabil), care va menţine toţi paşii pe care o să-i facem de acum în colo. În această coadă se pun, pentru fiecare pas, locurile care s-au marcat la punctul anterior.
2. Se marchează cu numere consecutive toate locurile posibile prin care putem trece, parcurgând în ordine elementele cozii, până când nu mai putem marca, sau am ajuns la final.

Aplicaţia #1 -> Problema Labirintului

*Se dă o matrice cu M linii şi N coloane. Ştiind locul de plecare, marcat cu*-1*, se cere să se determine drumul de lungime minimă până la o ieşire, iar in caz că nu există, se va afişa*-1

Rezolvare

După cum observaţi, este o aplicaţie a *algoritmului lui Lee*. Această problemă se poate rezolva şi cu metoda [backtracking](http://en.wikipedia.org/wiki/Backtracking), dar această metodă nu este una eficientă, complexitatea fiind O(4(M\*N)), sauO(3(M\*N)) după caz, ceea ce este foarte mult. În primul pas vom pune în coadă coordonatele locului de plecare, urmând apoi să parcurgem pe rând coada, până când nu mai există cale de ieşire sau am găsit una. Vă voi da un exemplu pentru a vă arata mai bine cum se marchează fiecare vecin în parte:

|  |  |  |
| --- | --- | --- |
| **fişier intrare** | **fişier ieşire** | **explicaţii** |
| 5 5   1 1 1 1 1 1 0 0 0 -1 1 0 0 1 1 1 0 0 1 1 1 0 1 1 1 | 6 | 1 1 1 1 1  1 *3* *2* *1* -1  1 *4* *3* *2* 1  1 *5* *4* 1 1  1 *6* 1 1 1 |

Aplicaţia #2 -> [Muzeu](http://infoarena.ro/problema/muzeu)

*Un muzeu are forma patratica si contine N\*N camere ce pot fi vizitate. Unele camere sunt deschise si contin opere de arta, altele sunt inchise (sunt folosite pentru alte scopuri). In unele din camerele libere, se afla paznici. Directorul muzeului se teme de eventualitatea unei spargeri si de aceea doreste sa evalueze cat de bine au fost asezati paznicii in camerele libere. Mai precis, el doreste sa afle, pentru fiecare camera libera, care este distanta minima pana la cel mai apropiat paznic (numarul minim de camere prin care trebuie sa intre un paznic pentru a ajunge la camera respectiva). Paznicii se pot deplasa numai in camerele libere din Nord, Est, Sud sau Vest (cu conditia sa nu paraseasca muzeul).*

Rezolvare

Această problemă se rezolvă cu *algoritmul lui Lee*, doar că în coadă vom pune iniţial nu o coordonată, ci toate coordonatele paznicilor. Pentru asta se citesc ca şir de caractere fiecare caracter în parte, şi se convertesc acestea (la citire) în numere astfel: zidurile (#) vor fi notate cu -2, paznicii (P) cu -1, iar căile libere cu 0. După executarea algoritmului, se verifică şi se înlocuiesc cifrele de 0 cu -1(pentru că în problemă ne cere să afişăm cu -1 locurile care nu au fost vizitate), iar cifrele de -1 (reprezentând paznicii) cu 0. Aveţi un exemplu de citire transpus în pseudocod:

[?](http://www.infoarena.ro/algoritmul-lee)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | citeşte m     pentru i=1,m          pentru j=1,m           început              citeşte s                         //variabila cu care citim caracterele pentru a le putea converti              dacă (s=='#') ct[i][j]=-2         //convertim caracterul în cifre              altfel dacă (s=='.') ct[i][j]=0;  //-----------------------------              altfel dacă (s=='P') ct[i][j]=-1; //-----------------------------              dacă (ct[i][j]==-1)               //verificăm dacă există un paznic               început                  q[l].st=i;                    //punem paznicul în coadă                  q[l].dr=j;                    //-----------------------                  l++;                          //contorul cu care numărăm paznicii               sfârşit(dacă)          sfârşit(pentru)     q1=l;                                      //contorul pe care îl folosim pentru a număra şi a pune în coadă locurile vizitate |

## Aplicaţia #3 -> [Insule](http://infoarena.ro/problema/insule)

*Dată fiind harta arhipelagului să se determine câte insule aparţin fiecărei ţări, precum şi lungimea minimă a unui pod care să satisfacă condiţiile din enunt.*

### Rezolvare

Pentru determinarea numărului de insule pentru fiecare ţară, se utilizează altgoritmul [FLOOD FILL](http://en.wikipedia.org/wiki/Flood_fill). Acest algoritm este algoritmul lui Lee, doar simplificat. Pentru partea cu podul, se utilizează algoritmul lui Lee, doar că în coadă vom pune zonele de ape, care au vecini o ţară R. Se parcurge coada până găsim o zonă de apă care are vecin o ţară G.

## Aplicaţia #4 -> [Alee](http://infoarena.ro/problema/alee)

*Scrieti un program care sa determine numarul minim de dale necesare pentru construirea unei alei continue de la o poarta la cealalta.*

### Rezolvare

Această problemă se rezolvă cu algoritmul lui Lee, iniţializând toată matricea cu -2, apoi pe parcurs ce citim poziţiile modificăm matricea cu -1, adică pomi, iar la sfârşit punem în coadă intrarea, o marcăm cu -1, iar ieşirea o marcăm cu -3. Parcurgem elementele din coadă până dăm de poarta de ieşire.