# Breadth First Search u Matricama

Pretraživanje po matricama je najjednostavnije izvesti skeniranjem matrice u dvostrukoj petlji. Međutim veoma često to neće zadovoljiti poslove koje u nekoj konkretnoj matrici moramo odraditi.

Strategija pod imenom “Flood Fill” (poplavno popunjavanje) je u ovakvim slučajevima dobitna kombinacija. Postoje dva načina da se implementira ova strategija. To su BSF i DSF algoritmi.

# Breadth First Search algoritam posećuje prvo sve okolne susede nekog polja. Pri tome svakog koji je posećen stavlja na kraj reda za čekanje. Ona polja koja su na početku se prva obradjuju. Postupak se ponavlja za sva polja sa početka liste. Algoritam tako ide u širinu dokle god može a implementira se uz pomoć strukture koja se naziva Queue (fifo storage – First in, first out). Queue struktura se implementira pomoću dva paralelna niza (ili niza slogova) od kojih u jednom pamtimo x a u drugom y koordinatu polja koje čeka na red za obradu. Uz ova dva niza potrebna su i dva celobrojna pointera koji pokazuju gde je početak a gde kraj Queue-a.

U C++ je queue ugradjen kontejner, pa ga takvog možete i koristiti.

# Takođe je veoma korisan pomoćni konstantan niz od 4 (ili 8) članova u kojima definišemo promenu r odnosno c koordinate susednih polja.

# int Dr[4] = { -1, 0, 1, 0};

# int Dc[4] = { 0, 1, 0, -1};

# { Promena x i y koordinata za redom susede: gore, desno, dole, levo. Ako se pod susedom smatraju i dijagonalna polja, tada je potrebno proširiti ove konstantne nizove na po 8 članova } ;

# Pustimo BSF iz nekog polja i radimo sledeće:

# 

Dogod se kju ne iprazni:

* Sve susede polja koje obradjujemo a koji nisu do sada posećeni, stavljamo ih u kju i obeležavamo ih da jesu posećeni.

| . | 1 | 2 | 3 | 4 | 5 | 6 | # | . | 1 | 2 | 3 | 4 | . | . |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | # | . | # | 2 | 3 | 4 | . | . | . |
| 2 | 3 | 4 | 5 | 6 | # | . | . | . | # | 4 | . | . | . | . |
| 3 | 4 | 5 | 6 | # | . | . | . | . | . | # | . | . | . | . |
| 4 | 5 | 6 | # | . | . | . | . | . | . | . | # | . | . | . |
| 5 | 6 | # | . | . | . | . | . | . | . | . | . | # | . | . |
| 6 | # | . | . | . | # | # | # | # | # | . | . | . | # | . |
| # | . | . | . | . | # | . | . | . | # | . | . | . | . | # |
| . | # | . | . | . | # | # | # | # | # | . | . | . | # | . |
| . | . | # | . | . | . | . | . | . | . | . | . | # | . | . |
| . | . | . | # | . | . | . | . | . | . | . | # | . | . | . |
| . | . | . | . | # | . | . | . | . | . | # | . | . | . | . |
| . | . | . | . | . | # | . | . | . | # | . | . | . | . | . |
| . | . | . | . | . | . | # | . | # | . | . | . | . | . | . |
| . | . | . | . | . | . | . | # | . | . | . | . | . | . | . |

# Implementacija celog algoritma bi bila.

**char Mapa[505][505];**

**int N,M;**

# void BFS (int r, int c){

# queue <int> Qr, Qc;

# 

# Qr.push(r);

# Qc.push(c);

# Mapa[r][c]='p';

# While (Qr.size()){

# r=Qr.front();

# c=Qc.front();

# for (int k=0; k<4; k++) { //desno, dole, levo, gore

# if (Mapa[r+Dr[k]][c+Dc[k]] == '.') {// poseta samo ‘.’}

# 

# Qr.push(r+Dr[k]);

# Qc.push(c+Dc[k]);

# Mapa[r+Dr[k]][c+Dc[k]]='p'; /// označavam da je sused posećen

/// ovde ide bilo kakva obrada zbog koje se ovaj algoritam koristi

# }

# }

# Qr.pop();

# Qc.pop();

# }

# }

# /// Za posetu svih polja u matrici, pusti se TZV. full BFS:

# int main(){

# { učitaju se dimenzije N x M matrica “Mapa” }

# for (int i=1; i<=N; i++)

# for (int j=1; j<=M; j++)

# if (Mapa[i][j] == ‘.’)

# BFS(i,j); { ovde se mogu setovati parcijalne sume brojači i sl.}

# Return 0;

# }

Zadaci :

<https://petlja.org/biblioteka/r/Problems/2009-skolsko-ss-muzicari>

<https://petlja.org/biblioteka/r/Problems/2010-skolsko-ss-molekularna-masa>