# **Depth 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 BFS i DFS algoritmi.

**Depth First Search** algoritam posećuje recimo prvo levog suseda pa onda njegovog levog, i td dogod postoje levi susedi. Tek kad ih nema više prelazi se na recimo donjeg suseda, a onda opet na levog ako je moguće itd. Algoritam tako ide u **dubinu** dokle god može a implementira se rekurzivno.

# **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.

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

# **Const Dx : array [1..4 ] of ShortInt = ( -1, 0, 1, 0);**

# **Const Dy : array [1..4 ] of ShortInt = ( 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 } ;

# Algoritam će ispravno raditi ako ima dovoljno stack memorije. Takodje je bitno da se posećena polja nekako obečeže, inače neće biti ispunjen kriterijum završetka u rekurzivnoj proceduri. Ako nam je matrica iz nekog razloga potrebna za dalje, tada je korisno napraviti kopiju matrice u kojoj se ovo obeležavanje odvija. Takođe zbog pojednostavljenja Flood Fill algoritma dobra taktika je da se cela matrica opšije okvirom od 1 do 2 polja i da se taj okvir označi kao posećen.

# Algoritam je sličan kao u BSF ali je rekurzivan pa ga je samim tim lakše napisati.

# Pustimo DSF iz nekog polja. To podrazumeva posetu njegovog prvog suseda u nekom odabranom smeru (ako do tada nije bio posećen), a zatim suseda od ovog suseda u istom tom smeru itd. Kada vise nema slobodnih suseda u prvoizabranom, prelazi se na sledeći smer.

# Ovaj postupak se ponavlja dogod se svi susedi ne posete.

| . | . | . | . | . | . | . | # | . | . | . | . | . | . | . |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| . | . | . | . | . | . | # | . | # | . | . | . | . | . | . |
| . | . | . | . | . | # | . | . | . | # | . | . | . | . | . |
| . | . | . | . | # | . | . | . | . | . | # | . | . | . | . |
| . | . | . | # | . | . | . | . | . | . | . | # | . | . | . |
| . | . | # | . | . | . | . | . | . | . | . | . | # | . | . |
| . | # | . | . | . | # | # | # | # | # | . | . | . | # | . |
| # | . | . | . | . | # | . | . | . | # | . | . | . | . | # |
| . | # | . | . | . | # | # | # | # | # | . | . | . | # | . |
| . | . | # | . | . | . | . | . | . | . | . | . | # | . | . |
| . | . | . | # | . | . | . | . | . | . | . | # | . | . | . |
| . | . | . | . | # | . | . | . | . | . | # | . | . | . | . |
| . | . | . | . | . | # | . | . | . | # | . | . | . | . | . |
| . | . | . | . | . | . | # | . | # | . | . | . | . | . | . |
| . | . | . | . | . | . | . | # | . | . | . | . | . | . | . |

# Implementacija celog algoritma bi bila.

# **Const** Dx : array [1..4] of ShortInt = (-1, 0, 1, 0);

# Dy : array [1..4] of ShortInt = ( 0, 1, 0, -1);

# **Var** Mapa : array [1..N, 1..M] of Char; { Matrica je dimenzija N x M }

# **Procedure** DFS (x, y : LongInt);

# **var** Kuda: LongInt;

# **Begin**

# Inc(Brojac);

# Mapa[x,y] := '#';

# **For** Kuda := 1 **to** 4 **do** {za sva 4 smera: Gore, desno, dole, levo }

# **if** Mapa[x+Dx[Kuda], y+Dy[Kuda]] <> '#' { poseta samo ‘.’}

# **then** DFS(Mapa[x+Dx[Kuda], y+Dy[Kuda]]);

# **end**;

# { Po zavrsetku Brojac će sadržati broj praznih polja u prostoru koji sadrži poč. polje.

# Ako je Mapa bitna, tada se u DSF-u koristi kopija matrice. }

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

# **Begin**

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

# **For** i := 1 **to** N **do**

# **For** j:= 1 **to** M **do**

# **If** Mapa[i,j] <> ‘#’

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

# 

# **End**.