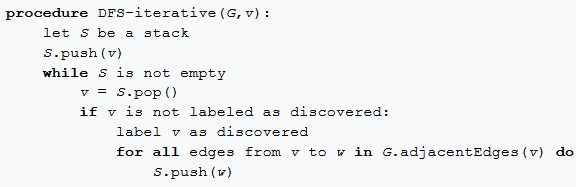
Pregled v globino vizualizacija – Opis razvoja

## Povzetek

Vizualizacija je spisana z uporabo WindowsForms knjižnice v C# jeziku. Za IDE sem uporabil Visual Studio 15. Za izhodišče vizualizacije sem izbral nerekurzivni algoritem iskanja v globino.



Vir: Wikipedia 1

Ker vizualizacijo poganjamo s klikom na gumb Naprej, je bilo treba algoritem prepisati brez zanke while(drugače se koda izvede v trenutku in vizualizacija ni ustrezna).

## KONCEPT – IDEJA

DFS sam po sebi ni zelo zahteven algoritem. Za vizualizacijo pa je treba premisliti na katerih mestih ga ustavimo za čimboljšo uporabniško izkušnjo. Najprej bom opisal kako se izvede vizualizacija z uporabo potrebnih metod znotraj Form1 razreda, nato sledi še opis kode in vseh uporabljenih metod(in orodij). Vmes bom še predstavil diagram poteka.

## 1.Pristop

Ker je najbolj intuitivno uporabiti while zanko tako kot v psevdokodi zgoraj, sem se najprej prikaza lotil z večnitnim programiranjem, kjer se znotraj glavne niti(main thread) vrši koda – vizualizacija, dokler v sklad ne dodamo novega vozlišča. Takrat se glavna nit ustavi in čaka da se izvede pomozna nit(pritisnemo gumb za Naprej). Ko se ta zakljuci se glavni proces nadaljuje naprej, dokler spet ne dodamo vozlišča oz. se algoritem zaključi. Za večnitno programiranje sem uporabil razred AutoRestEvent, ki ga uvozimo iz System.Threading knjižnice. Zaradi nestabilnosti pri uporabi te rešitve sem se na koncu odločil za drugačen pristop, zato se v podrobnosti večnitne rešitve v tem poročilu ne bom spuščal.

## 2.Pristop

Celotna logika algoritma se izvaja znotraj funkcij, ki ju pokličeta gumba Generiraj in Naprej. Na začetku, zunaj konstruktorja definiramo potrebne spremenljivke:

* SlovarSosedov – slovar, ki kot ključe vsebuje zaporedno številko(int) posameznega vozlišča, kot vrednosti pa hrani sezname sosednjih vozlišč. SlovarSosedov se naključno ustvari ob kliku na gumb Generiraj.
* Seznam Point objektov Coords – Uporabimo za shranjevanje točk(vozlišč) na platnu
* Sklad celih števil sklad – Del algoritma, vanj sproti dodajamo sosede trenutnega vozlišča
* Sklad CoordStack – uporabljamo za vizualizacijo algoritma na pomožni matriki
* Int start – začetno vozlišče(0)
* Seznam obiskani – Notri shranjujemo obiskana vozlišča.
* Seznam neobiskani\_sosedje – Neobiskani sosedje trenutnega vozlišča v algoritmu.
* Int counter – Preklopnik, ki omogoča spremljanje algoritma z uporabo 2 faz.
* Seznam helper – Seznam, ki hrani vozlišča, za vizualni prikaz.
* Int steviloVozlisc – število vozlišč, ki ga določimo v drugem Frame-u.

**generate\_graph**

Pred zagonom programa imamo tako definirane vse potrebne tipe. Ko pritisnemo gumb za generiranje grafa, se integerji nastavijo na 0, vsi seznami pa se spraznijo. Nato se zažene metoda **generate\_graph**, ki sprejme parameter **stevilo\_vozlisc**(definiran v predhodnem pojavnem oknu). Na gumb za generiranje grafa lahko kliknemo na kateremkoli koraku vizualizacije, zato je pomembno, da se vse spremenljivke ponovno nastavijo. Ob klicu metode **generate\_graph** se ustvarijo vozlišča – prikažejo se na grafičnem vmesniku in se hkrati shranijo kot objekti point v seznam **Coords**. Na začetku se prikažejo v rdeči barvi, zraven je zapisana zaporedna številka vsakega vozlišča.

Generate\_graph() :

X = 100 + (int)(200\*(1+Math.Cos(2 \* Math.PI \* i / stVozlisc)));

Y = 100 + (int)(150\*(1+Math.Sin(2 \* Math.PI \* i / stVozlisc)));

X in Y sta koordinati posameznega vozlišča, formula jih priredi tako, da vsa vozlišča ležijo enakomerno razporejena na krožnici(v tem primeru elipsi: a=200, b==150).

**draw\_edges**

Ko ustvarimo vozlišča, dodamo še povezave z metodo **draw\_edges**(). Pri te metodi najprej napolnimo slovar **SlovarSosedov** s praznimi seznami kot vrednostmi posameznega vozlišča kot ključa. Nato iteriramo skozi vozlišča in vsakemu v **SlovarSosedov** dodamo največ 2 novi povezavi do naključnih vozlišč. Pri vsaki iteraciji tudi preverimo če že obstaja povezava na trenutno vozlišče, v tem primeru dodamo tudi povezavo v obratno smer, ker gre za neusmerjen graf.

**Dodamo povezavo v obratno smer, če obstaja v trenutno vozlišče.**

foreach (var x in SlovarSosedov[i])

{

var xcor2 = Coords[x].X;

var ycor2 = Coords[x].Y;

c = (float)Math.Sqrt(Math.Pow(xcor2 - xcor, 2) + Math.Pow(ycor2 - ycor, 2));

x\_delta = 10 \* (xcor - xcor2) / c;

y\_delta = 10 \* (ycor - ycor2) / c;

arg.Graphics.DrawLine(pen, xcor - x\_delta, ycor - y\_delta, xcor2 + x\_delta, ycor2 + y\_delta);

platno.Invalidate();

}

Sedaj sledi glavni del vizualizacije; graf je ustvarjen in prikazan na vrhu risalne površine razreda **Graphics** znotraj knjižnice **System.Drawing**. Vsi potrebni podatki so pripravljeni, imamo slovar vozlišč in soležnih povezav, ter začetno vozlišče.

**Gumb naprej**

Vsi nadaljnji koraki se izvedejo znotraj metode **button3\_Click**, ki jo kliče gumb Naprej. Ob kliku funkcija najprej preveri, če imamo izbrani vsaj 2 vozlišči, v nasprotnem primeru se algoritem ustavi.

Nato v pomožni matriki pobarvamo na belo vse celice, ki niso svetlo modre barve(svetlo modre so dodana vozlišča, sosednja vozlišča v trenutnem koraku so obarvana z zeleno – ko gremo naprej dobimo nova sosednja vozlišča).

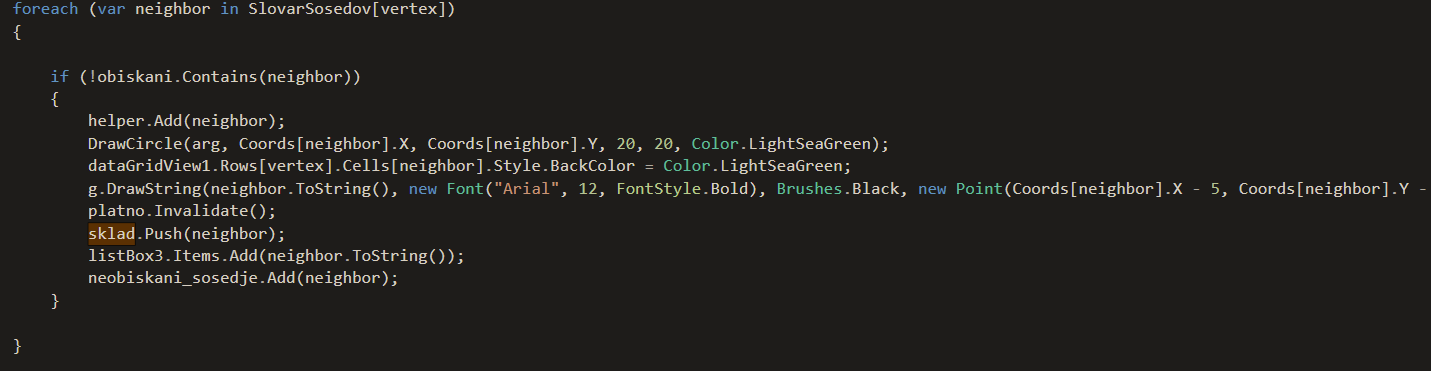
Če je slučajno **SlovarSosedov** prazen, zaključimo vizualizacijo in prikažemo opozorilo. Nadaljujemo s pogojnim stavkom, ki je najbolj pomemben za uspešen prikaz algoritma in ima 3 možnosti:

* if (counter == 0 & sklad.Count > 0)

Algoritem se izvaja dokler sklad neobiskanih vozlišč ni prazen, integer counter deluje le kot stikalo med dvema glavnima deloma vizualizacije. V tem prvem na grafu in v pomožni matriki dodamo obiskana vozlišča(obarvamo svetlo modro) in posodabljamo vse potrebne spremenljivke in podatkovne strukture, ki hranijo trenutno stanje algoritma. Hkrati tudi rumeno obarvamo povezavo do naslednjega vozlišča v algoritmu. Posodobimo tudi TextBox okvirje za sklad, dodana vozlišča in komentar trenutnega stanja algoritma.

* else if (counter == 1)

Else if pogoj je v bistvu samo umetno ustvarjen drugi korak algoritma, ko določimo(in rdeče obarvamo) neobiskana vozlišča, ki ne mejijo na trenutno vozlišče. Ta vozlišča hranimo v seznamu **helper**. V tem koraku tudi oranžno obarvamo trenutno vozlišče in ga dodamo v **listBox1**, ki beleži vsa dodana vozlišča, dodamo ga tudi v seznam obiskani. V tem koraku tudi iteriramo čez sosednja vozlišča in dodamo še neobiskana v **sklad:**



Vsa sosednja obarvamo z zeleno barvo. Na koncu tudi komentiramo trenutno stanje v okvir richTextBox2, ter nastavimo counter = 0, tako da se ob naslednjem kliku na gumb Naslednji zažene prvi del pogoja, ko se obarva dodano vozlišče na svetlo modro.

- else

Na koncu, ko je **sklad** prazen, še prikažemo pojavno okno da se je vizualizacija zaključila in ponastavimo potrebne spremenljivke za ponoven zagon.

else

{

arg.Graphics.Clear(Color.White);

this.Invalidate();

SlovarSosedov.Clear();

Coords.Clear();

listBox1.Items.Clear();

var formPopup = new PopupForm2();

this.Hide();

formPopup.ShowDialog(this);

}