УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Милан Стевановић, ПР 128/2018

2Д визуелизација шеме електродистрибутивне мреже са фокусом на перформансе визуелизације

ПРОЈЕКАТ
- Примењено софтверско инжењерство (ОАС) -

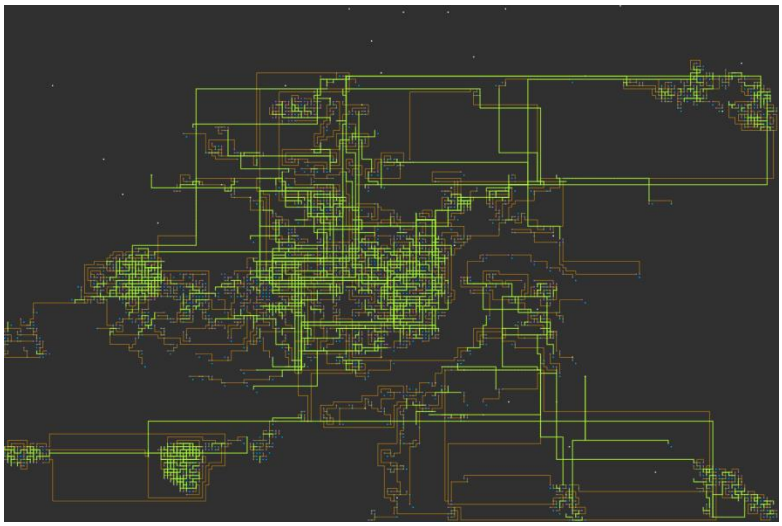
Нови Сад, 30.06.2022.

SADRŽAJ

OPIS REŠAVANOG PROBLEMA

Pojam 2D grafike se odnosi na slike koje imaju samo 2 dimenzije, kao što je slika koja se prikazuje na ekranu, fotografija ili tehnički crtež [1]. Tema projekta je iscrtavanje entiteta elektrodistributivne mreže Novog Sada na osnovu realnih podataka dobijenih od privrednog društva "Elektrovojvodina". Entiteti elektrodistributivne mreže su čvorovi, podstanice i prekidači. Radi lakšeg raspoznavanja entiteta na grafičkom prikazu svaki tip entiteta je prikazan različitom bojom.
Entiteti koji se nalaze na istim koordinatama se prikazuju jedan pored drugog tako što se vrši aproksimacija pozicije na najbliži slobodni podeok na kojem entitet može da se iscrta.

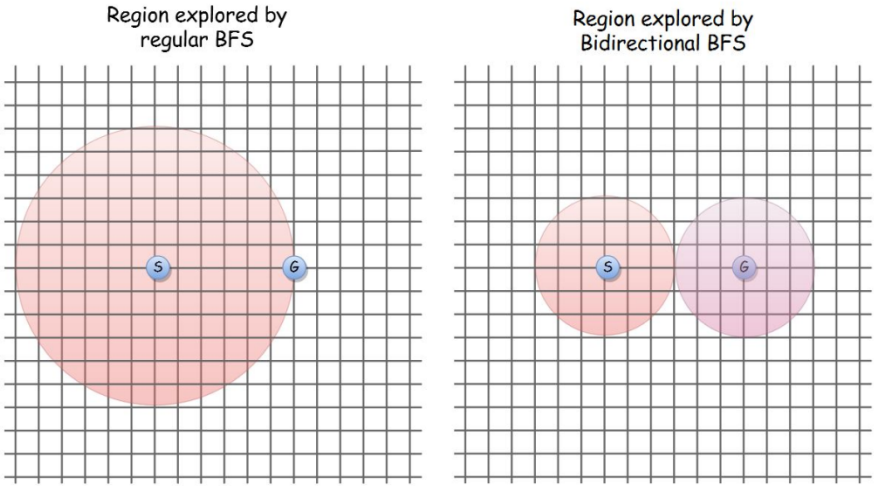
Vodovi spajaju entitete i iscrtavaju se kao prave linije koje mogu da skreću samo pod pravim uglom.
Vodovi se iscrtavaju pomoću BFS (*Breadth First Search*) algoritma kroz dve iteracije. U prvoj iteraciji se iscrtavaju vodovi kod kojih je moguće pronaći najkraću putanja bez presecanja već iscrtanih vodova. U drugoj iteraciji se iscrtavaju vodovi za koje u prvoj iteraciji nije bilo moguće naći putanju bez presecanja i tada se oni iscrtavaju sa presecanjem već iscrtanih vodova. Primer vizuelizacije se može videti prikazan na slici (*Slika 1.1*)



Slika 1.1 2D Vizuelni prikaz šeme elektrodistributivne mreže grada Novog Sada

Kanvas na kojem se vrši vizuelizacija je promenljive veličine. Sa mogućnošću promene veličine kanvasa dobijamo i različite veličine matrice koja je osnova BFS algoritmu za pronalaženje najkraće moguće putanje što znatno utiče na performanse i vreme potrebno za iscrtavanje vodova.

Cilj ovog projekta jeste ubrzanje performansi već postojećeg BFS algoritma uvođenjem dvosmernog BFS algoritma i upoređivanje potrebnog vremena za iscrtavanje vodova za različite veličine kanvasa. Na slici (*Slika 1.2*) vidi se razlika između performansi BFS i dvosmernog BFS algoritma.



Slika 1.2 BFS i dvosmerni BFS [2]

U nastavku rada opisaće se tehnologije koje su korišćene u izradi rešenja, realni podaci koji se koriste za vizuelizaciju, algoritam aproksimacije pozicije entiteta kao i logika dvosmernog BFS algoritma.

OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

Tehnolgije korišćene prilikom izrade rešenja projektnog zadatka su sledeće:

1. C#
2. WPF (*Windows Presentation Foundation*)
3. Microsoft Visual Studio

1. **C#** - moderan, objektno-orijentisan programski jezik opšte namene kreiran od strane Microsoft-a zajedno sa .NET platformom. Danas, C# je jedan od najpopularnijih programskih jezika korišćen od strane velikog broja programera. Koristi se za izradu veb, desktop, mobilnih aplikacija kao i igrica [3].

2. **WPF (*Windows Presentation Foundation*)** - *UI Framework* koji služi za kreiranje desktop klijentskih aplikacija. WPF koristi jezik sa otvarajućim i zatvarajućim tagovima poznatiji kao XAML uz pomoć kojeg se programira izgled aplikacije.

3. **Microsoft Visual Studio** - integrisano razvojno okruženje napravljeno od strane Microsoft-a i koristi se za razvoj različitih tipova softvera kao što su računarski programi, veb stranice, veb usluge i mobilne aplikacije. Sadrži alate za automatsko dovršavanje koda, kompajlere kao i druge funkcije koje olakšavaju proces razvoja softvera.

OPIS REŠENJA PROBLEMA

Modeli podataka koji su korišćeni su:

- **PowerEntity**
- **NodeEntity** – nasleđuje klasu PowerEntity
- **SubstationEntity** – nasleđuje klasu PowerEntity
- **SwitchEntity** – nasleđuje klasu PowerEntity
- **LineEntity**
- **MatrixCell**

Model **PowerEntity** ima sledeća polja:

- Id – (long) jedinstveni identifikator entiteta
- Name – (string) ime entiteta
- UtmX – (double) *Universal Transverse Mercator* X koordinata
- UtmY – (double) *Universal Transverse Mercator* Y koordinata
- Longitude – (dobule) geografska dužina dobijena iz UtmY koordinate
- Latitude – (double) geografska širina iz UtmX koordinate
- MatrixRow – (int) red u matrici u kojem će se entitet nalaziti nakon skaliranja i aproksimacije
- MatrixCol – (int) kolona u matrici u kojoj će se entitet nalaziti nakon skaliranja i aproksimacije

Model **LineEntity** ima sledeća polja:

- Id – (long) jedinstveni identifikator entiteta
- Name – (string) ime entiteta
- FirstEnd – (long) jedinstveni identifikator entiteta od kog počinje vod
- SecondEnd – (long) jedinstveni identifikator entiteta u kom se završava vod
- StartMatrixRow – (int) red u matrici u kojem se nalazi entitet od kog počinje vod
- StartMatrixCol – (int) kolona u matrici u kojem se nalazi entitet od kog počinje vod
- EndMatrixRow – (int) red u matrici u kojem se nalazi entitet u kom se završava vod
- EndMatrixCol – (int) kolona u matrici u kojoj se nalazi entitet u kom se završava vod
- IsDrawn – (bool) označava da li je vod iscran na kanvasu

Model **MatrixCell** ima sledeća polja:

- Row – (int) red u matrici u kojem se nalazi ćelija
- Cell – (int) kolona u matrici u kojoj se nalazi ćelija

Program počinje učitavanjem XML fajla pod nazivom „Geographic.xml“ koji u sebi sadrži realne podatke o elektrodistibutivnoj mreži Novog Sada sa preciznim koordinatama u UTM formatu gde se svaki entitet nalazi. Takođe sadrži i podatke o tome koji vod povezuje koji entitet.

Nakon što su podaci učitani, vrši se konvertovanje koordinata iz UTM formata u geografsku širinu i dužinu uz pomoć funkcije *ToLatLon(double utmX, double utmY, int zoneUTM, out double latitude, out double longitude)*. Uz pomoć dobijene geografske širine i dužine, vrši se skaliranje i aproksimacija na predviđeno mesto u matrici u kojoj bi entitet trebao da se nalazi. Tu dolazi do problema ako se više entiteta nalazi na istim koordinatama. Rešenje tog problema je objašnjeno u daljem teksu uz pomoć funkcije *CheckField*.

U program se unosi širina kanvasa u pikselima dok se visina kanvasa automatski izračunava po proporciji 3:2. U zavisnosti od unete veličine kanvasa, kreira se matrica koja će biti korišćena za pozicioniranje entiteta kao i matrica karaktera za pronalaženje putanje uz pomoć dvosmernog BFS algoritma. Opseg veličine kanvasa je od 300x200 piskela do 2250x1500 piksela a veličina matrica je srazmerno skalirana prema veličini kanvasa od 100x66 ćelija do 500x333 ćelija. Shodno tome, veličina jednog entiteta na kanvasu se dobija kao širina kanvasa podeljena sa brojem kolona matrice (npr. U slučaju najveće matrice 2250px / 500 = 4.5 piksela).

Nakon uspešnog skaliranja i kreiranja matrica, na red dolazi aproksimacija pozicije entiteta u matrici prema kojoj će se entiteti iscrtavati na kanvasu. To se izvršava uz pomoć funkcije *CheckField* (*Slika 2.1*).

Funkciji se prosleđuje entitet, red i kolona u kojoj bi entitet trebao da se nalazi ukoliko to polje nije već zauzeto i korak za koliko će se tražiti sledeće slobodno mesto ukoliko je predviđeno mesto zauzeto. U prvoj iteraciji korak koji je prosleđen je 0. Ukoliko je pozicija u matrici sa predviđenim redom i kolonom zauzeta, u sledećoj iteraciji korak se povećava i pretražuju se okolne ćelije počevši od gornje leve do donje desne. Funkcija se izvršava sve dok se entetetu ne pronađe slobodno mesto u matrici.

```
public static bool CheckField(PowerEntity entity, int row, int col, int step)
{
    for (int i = -step; i <= step; i++) // rows
    {
        for (int j = -step; j <= step; j++) // cols
        {
            if (col + j < Data.matrixCols && col + j > -1 && row + i < Data.matrixRows && row + i > -1)
            {
                if (Data.matrix[row + i, col + j] == null)
                {
                    // Found empty field
                    Data.matrix[row + i, col + j] = entity;
                    Data.allEntities[entity.Id].MatrixRow = row + i;
                    Data.allEntities[entity.Id].MatrixCol = col + j;
                    return true;
                }
            }
        }
    }
    return false; // Not found
}
```

Slika 2.1 Funkcija *CheckField*, Aproksimacija najbližeg slobodnog mesta entiteta

Iscrtavanje entieta se izvršava tako što se prolazi kroz dvodimenzionalnu matricu i proverava se da li je trenutno polje prazno ili se u njemu nalazi entitet.

Ukoliko se u ćeliji nalazi entitet, proverava se kog je tipa (čvor, podstanica ili prekidač) i u zavisnosti od tipa, objektu *Rectangle* koji predstavlja entitet na kanvasu, se dodeljuje određena boja. Pozicija gde će se entitet (*Rectangle*) iscrtati se dobija proizvodom veličine entiteta sa koordinatama matrice gde se entitet nalazi.

Pomeraj po X osi se dobija proizvodom veličine entiteta i kolone u kojoj se entitet nalazi dok se pomeraj po Y osi dobija proizvodom veličine entiteta i reda u kojem se entitet nalazi u matrici.

Bidirectional Search – Dvosmerna pretraga (BFS)

U ovom trenutku na kanvasu su iscrtani svi učitani entiteti. Da bismo imali potpunu sliku vizuelizacije potrebno je da pronađemo najkraću putanju za svaki učitani vod. Već smo spomenuli da smo kreirali dvodimenzionalnu statičku matricu karaktera (*char[matrixRows, matrixCols] linesMatrix*) koja će nam služiti za pronalaženje putanje.

Tip podataka koji se čuva u matrici je karakter (*char*). Da bismo razlikovali vod (zauzeto polje) od slobodnog polja u našoj matrici, moramo poljima u matrici da dodelimo određene vrednosti.

U zavisnosti od karaktera koji se nalazi u polju matrice dobijamo različite rezultate:

- 'S' – (*Source*) označava polje iz kog kreće vod
- 'D' – (*Destination*) označava polje u kom se završava vod
- 'X' – zauzeto polje, označava da je se u tom polju već nalazi vod
- ' ' – (razmak) označava slobodno polje

U procesu pronalaženja putanje ključne su četiri funkcije:

- *BidirectionalSearch* – proverava da li je moguće pronaći putanju
- *IsValid* – proverava da li je ćelija validna
- *GetNeighborCells* – kao povratnu vrednost daje listu susednih ćelija
- *ReconstructPath* –rekonstruiše putanju i kao povratnu vrednost ima listu ćelija

Funkcija *BidirectionalSearch* za ulazne parametre ima vod (*LineEntity*) kao i matricu karaktera sa podacima o tome koja polja su već zauzeta sa vodovima. U startu, funkcija označava početak ('S', *source*) i kraj ('D', *destination*) u prosleđenoj matrici i inicijalizuje neophodne promenljive za dalji rad (*Slika 2.2*).

```
14          linesMatrix[line.StartMatrixRow, line.StartMatrixCol] = 'S'; // Line FirstEnd
15          linesMatrix[line.EndMatrixRow, line.EndMatrixCol] = 'D'; // Line SecondEnd
16
17          MatrixCell source = new MatrixCell(line.StartMatrixRow, line.StartMatrixCol);
18          MatrixCell destination = new MatrixCell(line.EndMatrixRow, line.EndMatrixCol);
19
20          Queue<MatrixCell> sourceQueue = new Queue<MatrixCell>();
21          Queue<MatrixCell> destinationQueue = new Queue<MatrixCell>();
22
23          sourceQueue.Enqueue(source);
24          destinationQueue.Enqueue(destination);
25
26          bool[,] sourceVisited = new bool[Data.matrixRows, Data.matrixCols];
27          bool[,] destinationVisited = new bool[Data.matrixRows, Data.matrixCols];
28
29          MatrixCell[,] sourcePrevious = new MatrixCell[Data.matrixRows, Data.matrixCols];
30          MatrixCell[,] destinationPrevious = new MatrixCell[Data.matrixRows, Data.matrixCols];
```

Slika 2.2 Inicijalizacija novih promenljivih

Promenljive *source* i *destination* tipa *MatrixCell* predstavljaju ćelije gde vod treba da počne i da se završi.

Promenljive *sourceQueue* i *destinationQueue* tipa *Queue<MatrixCell>* predstavljaju FIFO (*First In First Out*) strukturu u koju će se dodavati nove i preuzimati već učitane ćelije. Prve ćelije koje će biti dodate u ova dva reda će biti objekti *source* i *destination*.

Promenljive *sourceVisited* i *destinationVisited* predstavljaju dvodimenzionalne matrice istih dimenzija kao i *linesMatrix* (matrica karaktera). Ove matrice služe za evidenciju već posećenih ćelija. Samo ćelije koje još uvek nisu posećene će biti dodate u red *sourceQueue* / *destinationQueue* u zavisnosti da li pretraga počinje od početka / kraja. Na početku svi elementi matrice posećenosti se inicijalizuju na stanje '*False*' osim elemenata koji su inicijalno dodati u red. Ti elementi su *source* i *destination* od kojih kreće pretraga.

Promenljive *sourcePrevious* i *destinationPrevious* su dvodimenzionalne matrice koje služe za evidenciju putanje od početne ili krajnje ćelije pa sve do trenutne ćelije do koje je algoritam stigao. Ove dve matrice igraju ključnu ulogu u rekonstrukciji putanje.

Nakon završene inicijalizacije pomoćnih promenljivi možemo da krenemo sa logikom algoritma (*Slika 2.3*) koja će biti objašnjena u daljem tekstu.

```
47          // Bidirectional Search
48          while (sourceQueue.Count != 0 && destinationQueue.Count != 0)
49          {
50              if (sourceQueue.Count != 0)
51              {
52                  MatrixCell currentCell = sourceQueue.Dequeue();
53
54                  if (currentCell == destination || destinationQueue.Contains(currentCell))
55                  {
56                      // Path Found
57                      linesMatrix[line.StartMatrixRow, line.StartMatrixCol] = ' ';
58                      linesMatrix[line.EndMatrixRow, line.EndMatrixCol] = ' ';
59                      return ReconstructPath(currentCell, sourcePrevious, destinationPrevious);
60                  }
61                  else
62                  {
63                      foreach(MatrixCell cell in GetNeighborCells(currentCell))
64                      {
65                          if(isValid(cell.Row, cell.Col, sourceVisited, linesMatrix))
66                          {
67                              sourceQueue.Enqueue(new MatrixCell(cell.Row, cell.Col));
68                              sourceVisited[cell.Row, cell.Col] = true;
69                              sourcePrevious[cell.Row, cell.Col] = currentCell;
70                          }
71                      }
72                  }
73              }
74
```

Slika 2.3 Logika algoritma *Bidirectional Search*

Funkcija se izvršava dokle god se redovi *sourceQueue* i *destinationQueue* ne isprazne.

Ukoliko se redovi isprazne a putanja u međuvremenu nije pronađena funkcija kao povratnu vrednost vraća '*null*' što znači da nije moguće pronaći putanju bez presecanja drugih vodova.

Ukoliko je pronađena tačka preseka između dve nezavisne pretrage (iz smera početka prema kraju i iz smera kraja prema početku) poziva se funkcija *ReconstructPath* koja će biti objašnjena u daljem tekstu i funkcija kao povratnu vrednost ima putanju (*List<MatrixCell>*) kojom će vod biti iscrtan.

U slučaju da redovi nisu prazni uzima se prva ćelija učitana u red i ispituje se da li smo došli do destinacije ili se ta ćelija nalazi u redu ćelija koje su posećene pretragom od strane kraja prema početku. Ako je neki od uslova zadovoljen, polja u matrici karaktera se resetuju na prazan karakter da ne bi došlo do kolizije tokom sledećih prolaza kroz matricu i poziva se metoda *ReconstructPath*.

Ako uslovi nisu zadovoljeni to znači da još uvek nismo stigli do destinacije i da ne postoji nijedna presečna tačka između dve nezavisne pretrage. U tom slučaju dobavljamo sve susedne ćelije uz pomoć metode *GetNeighborCells* (*Slika 2.4*). Funkcija vraća listu ćelija koje se nalaze gore, dole, levo i desno od prosleđene ćelije.

```
113     private static List<MatrixCell> GetNeighborCells(MatrixCell cell)
114     {
115         List<MatrixCell> neighborCells = new List<MatrixCell>();
116         neighborCells.Add(new MatrixCell(cell.Row - 1, cell.Col));
117         neighborCells.Add(new MatrixCell(cell.Row + 1, cell.Col));
118         neighborCells.Add(new MatrixCell(cell.Row, cell.Col - 1));
119         neighborCells.Add(new MatrixCell(cell.Row, cell.Col + 1));
120         return neighborCells;
121     }
```

Slika 2.4 Funkcije *GetNeighborCells*

Prolaskom kroz svaki element liste dobijenih susednih ćelija, proverava se da li je ćelija validna. Ćelija se validira uz pomoć funkcije *IsValid* (*Slika 2.5*) koja za ulazne parametre ima red i kolonu u kojoj se ćelija nalazi, matricu karaktera (*linesMatrix*) kao i matricu posećenosti.

```
105     private static bool isValid(int row, int col, bool[,] visited, char[,] linesMatrix)
106     {
107         if (row >= 0 && col >= 0 &&
108             row < Data.matrixRows &&
109             col < Data.matrixCols &&
110             linesMatrix[row, col] != 'X' &&
111             visited[row, col] == false)
112             return true;
113         else
114             return false;
115     }
```

Slika 2.5 Funkcija *isValid*

Ćelija je validna ako zadovoljava sledeće uslove:

- Ne izlazi iz opsega dvodimenzionalne matrice
- Polje matrice karaktera na toj poziciji ne sadrži karakter 'X' tj. na toj poziciji se ne nalazi već iscrtan vod
- Polje matrice posećenosti na toj poziciji ima vrednost '*False*', tj. ćelija nije posećena

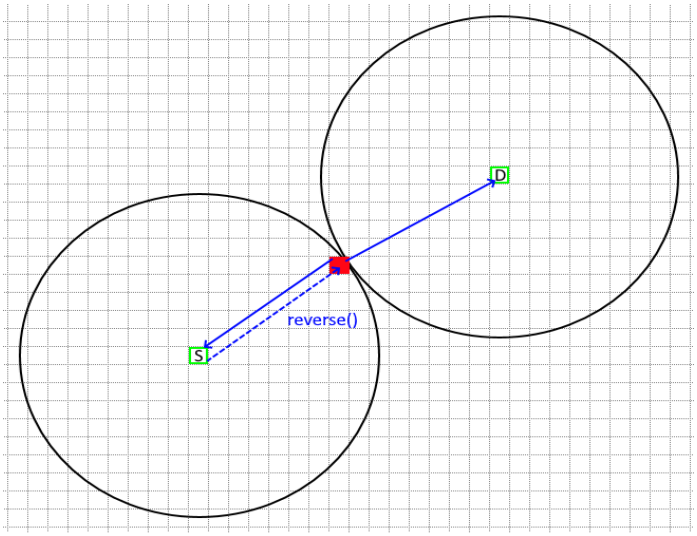
Ako je prosleđena ćelija validna dodaje se u red, element u matrici posećenosti sa koordinatama gde se ćelija nalazi se postavlja na '*True*' što znači da je ta ćelija posećena i na kraju se ćelija dodaje u matricu *sourcePrevious* radi evidentiranja putanje od početne tačke do trenutne ćelije.

Do sada smo objasnili algoritam koji radi pretragu od početka prema kraju. Isti princip (*Slika 2.3* linije 50-73) se primenjuje i za pretragu od kraja prema početku.

Rekonstrukcija putanje (Funkcija *ReconstructPath*)

Važnu ulogu u dvosmernoj pretrazi vrši funkcija *ReconstructPath*. Kao ulazne parametre funkcija ima ćeliju pod nazivom *intersection* tipa *MatrixCell* kao i dvodimenzionalne matrice *sourcePrevious* i *destinationPrevious*.

Na slici 2.6 može da se vidi princip po kojem se pronalazi putanja.



Slika 2.6 Princip pronalaženja putanje pomoću funkcije *ReconstructPath*

Putanja je predstavljena kao lista elemenata tipa *MatrixCell* pomoću kojih će vod biti iscrtan. Putanja je podeljena na dva dela.

Prvi deo je putanja od tačke preseka do početka (*source*), a drugi deo je od tačke preseka do kraja (*destination*).

```
127         public static List<MatrixCell> ReconstructPath(MatrixCell intersection, MatrixCell[,] sourcePrev, MatrixCell[,] destinationPrev)
128         {
129             List<MatrixCell> path = new List<MatrixCell>();
130
131             // Find Path from Source to Intersection Cell
132             for (MatrixCell cell = intersection; cell != null; cell = sourcePrev[cell.Row, cell.Col])
133             {
134                 path.Add(cell);
135             }
136             path.RemoveAt(0); // This cell will be added below, in path from intersection to destination
137             path.Reverse();
138
139             // Find Path from Intersection to Destination
140             for (MatrixCell cell = intersection; cell != null; cell = destinationPrev[cell.Row, cell.Col])
141             {
142                 path.Add(cell);
143             }
144
145             return path;
146         }
147     }
```

Slika 2.7 Funkcija *ReconstrucPath*

Na slici 2.7 (linije 131-137) prolazi se kroz sva polja matrice *sourcePrevious* koja imaju vrednost različitu od '*null*'. Kreće se od tačke preseka pa sve do početka i dodajemo svaku ćeliju na koju naiđemo u putanju. Na ovaj način dobijamo putanju od tačke preseka (*intersection*) do početka uključujući i te dve tačke. Pošto se na isti način dobija i putanja od tačke preseka do destinacije vidimo da će se u krajnjoj putanji dva puta naći tačka preseka. Da bismo rešili ovaj problem potrebno je da nakon kreiranja prvog dela putanje izbrišemo prvi element u nizu (tačku preseka) i da obrnemo redosled elemenata u nizu da bismo dobili putanju od početka do tačke preseka a ne obrnuto.

Nakon što smo dobili prvi deo putanje, isti princip se primenjuje i na drugi deo. Na dobijenu putanju dodajemo ćelije od tačke preseka pa sve do naše destinacije i samim tim dobijamo kompletno rekonstruisanu putanju po kojoj će vod biti iscrtan.

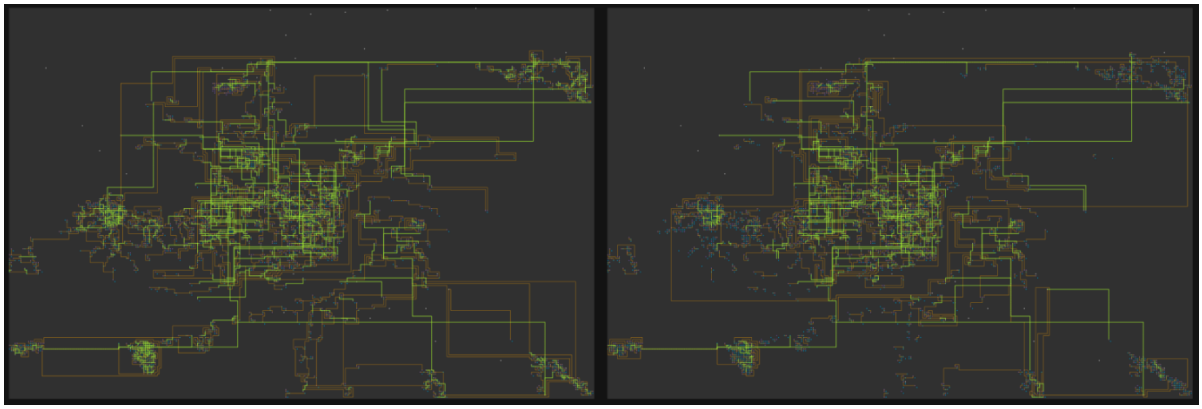
Iscrtavanje linija na kanvas se vrši uz pomoć funkcije *BFSDrawLine* koja za ulazni parametar ima objekat tipa *LineEntity*. Funkcija poziva gore objašnjenu funkciju *BidirectionalSearch* i pomoću nje dobija putanju. Ukoliko je putanja pronađena, polje pod nazivom *IsDrawn* se postavlja na pozitivnu vrednost '*True*'. U drugoj iteraciji se uzimaju u obzir samo vodovi čije polje *IsDrawn* ima vrednost '*False*'. Oni se iscrtavaju po istom principu, jedina razlika jeste što se zanemaruje činjenica da će se linija iscrtati preko već postojećeg voda.

Linije se iscrtavaju na kanvas uz pomoć klase *Polyline* biblioteke *Shapes* (*Slika 2.8*). Prolaženjem kroz dobijenu putanju, kolekciji *Points* se dodaju tačke po kojima će vod biti iscrtan. Pored pozicioniranja liniji se dodaju i boja kao i *ToolTip* koji se prikazuje prelaskom kursora preko iscrtanog voda. Istovremeno, u matricu karaktera se upisuju vrednost 'X' na pozicije putanje kojom vod prolazi.

```
226             Polyline polyLine = new Polyline();
227             polyLine.Stroke = Brushes.Orange;
228             polyLine.StrokeThickness = Data.entitySize / 4F;
229             polyLine.ToolTip = line.ToString();
230             polyLine.Tag = line;
231             PointCollection pointCollection = new PointCollection();
232
233             foreach (MatrixCell cell in path)
234             {
235                 Data.linesMatrix[cell.Row, cell.Col] = 'X';
236                 pointCollection.Add(new System.Windows.Point(cell.Col * Data.gridCellSize + Data.entitySize / 2, cell.Row * Data.gridCellSize + Data.entitySize / 2));
237             }
238
239             polyLine.Points = pointCollection;
240             mainCanvas.Children.Add(polyLine);
241             drawnLines++;
```

Slika 2.8 Iscrtavanje linije na kanvas

Radi kasnijeg prikazivanja rezultata bitno je napomenuti da postoji opcija iscrtavanja celokupne mreže kao i prikaza mreže gde se ne iscrtavaju vodovi koji počinju iz entiteta koji su tipa čvor, što znatno utiče na vreme iscrtavanja elektrodistributivne mreže. Razlika u 2D grafičkom prikazu može se videti na slici 2.9.



Slika 2.9 Elektrodistibutivna mreža sa (levo) i bez (desno) iscrtavanja svih vodova

Rezultat - Poboljšanje performansi

Rezultat uvođenja dvosmerne pretrage umesto prvobitne pretrage u jednom smeru biće prikazana tabelarno sa vremenskim rezultatima (*Tabela 1.*) . Za svaki slučaj koji je naveden u tabeli program je pokrenut tri puta a u tabelu je unet najbolji (najbrži) rezultat. Iscrtavanje elektrodistributivne mreže se vršilo u pet različitih veličina kanvasa sa i bez iscrtavanja vodova koji počinju iz entiteta tipa čvor. U tabelarni prikaz je uneto i vreme potrebno jednosmernom BFS algoritmu za iscrtavanje celokupne elektrodistributivne mreže radi prikazivanja poboljšanja performansi u odnosu na dvosmerni BFS algoritam.

Rezolucija kanvasa (pikseli)	Veličina matrice	Dvosmerni (Bidirectional) BFS		Jednosmerni BFS
		Vreme - nepotpuna mreža (sekunde)	Vreme - potpuna mreža (sekunde)	Vreme - potpuna mreža (sekunde)
300 x 200 px	100 x 66	1.11	1.31	2.71
750 x 500 px	192 x 128	1.36	1.88	6.54
1125 x 750 px	269 x 179	2.54	3.51	9.79
1800 x 1200 px	407 x 771	6.8	9.28	18.43
2250 x 1500 px	500 x 330	11.91	14.74	26.55

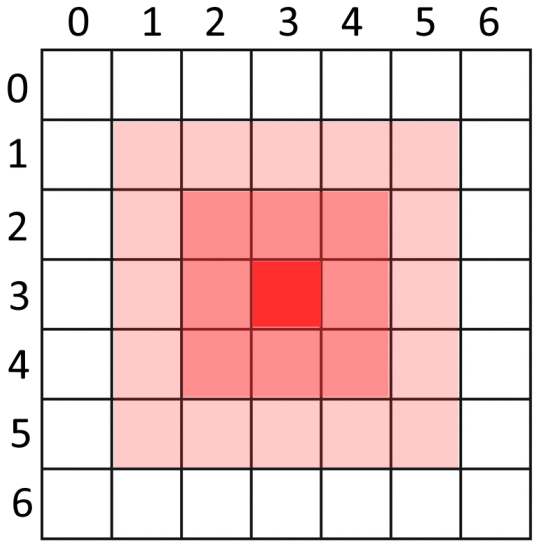
Tabela 1. Tabelarni prikaz performansi

Iz vremenskih rezultata prikazanih u tabeli (*Tabela 1.*) možemo da zaključimo da je vreme neophodno za iscrtavanje nepotpune mreže manje nego vreme potrebno za iscrtavanje potpune mreže.
Takođe, na osnovu dobijenih vremenskih rezultata možemo da vidimo da je u pojedinim slučajevima implementacija dvosmerne pretrage doprinela trostrukom smanjenju vremena potrebnog za iscrtavanje potpune elektrodistibutivne mreže u odnosu na jednosmernu pretragu.

Još jedan dokaz koji pokazuje koliko je dvosmerna pretraga brža je sledeći: Ako predpostavimo da je faktor grananja stabla ***b*** i rastojanje od izvora do cilja ***d***, onda bi složenost jednosmernog BFS algoritma bila ***O(b^d)***. S druge strane ako se koristi dvosmerna pretraga složenost alogoritma bi bila ***O(b^(d/2))*** za svaku pretragu a ukupna složenost bi bila ***O(b^(d/2)+b^(d/2))*** što je daleko manje od ***O(b^d)***. [4]

PREDLOZI ZA DALJA USAVRŠAVANJA

Problem koji trenutno utiče na performanse programa jeste deo koda za aproksimaciju pozicije entiteta u dvodimenzionalnoj matrici.
Ukoliko entitet treba da se smesti na poziciju koja je već zauzeta, algoritam za aproksimaciju će krenuti da proverava sve susedne ćelije udaljenje jedan korak od gornje leve do donje desne ćelije. Ukoliko se u prvom prolazu ne pronađe slobodno mesto, korak se povećava za jedan. To znači da se sa svakom novom iteraciom ispituje veći deo matrice koji je prethodno ispitan. Kako se povećava korak tako će se i povećavati i broj nepotrebnih ispitivanja podmatrice.



Slika 3.1 Mana algoritma za aproksimaciju pozicije entiteta

Na konkretnom primeru (*Slika 3.1*) ako zamislimo da su sva polja zauzeta i da je algoritam stigao do treće iteracije vidimo da je polje sa koordinatama (3, 3) ispitivano 3 puta za slobodno mesto a njegovi susedi 2 puta. Što dovodi do zaključka da srazmerno koracima raste i nepotreban broj provera već proverenih polja.

Projekat se bavi vizuelizaciom elektrodistributivne mreže grada Novog Sada i s obzirom da radi sa realnim podacima, jedno od poboljšanja u vidu vizuelizacije bi bilo dodavanje mape (u obliku slike ili *GoogleMaps*) uz pomoću koje bismo jasno mogli da odredimo gde se koji entitet nalazi i kuda prolaze vodovi.

Program već poseduje mogućnost čuvanja iscrtanog kanvasa u obliku slike sa trenutnim datumom i vremenom kada je slika napravljena. Takođe poseduje i funkcionalnosti iscrtavanja dodatnih 2D grafičkih elemenata na kanvas pored već iscrtane elektrodistributivne mreže kao što su elipse, poligoni i dodavanje tekst.
Pored toga jedno od mogućih funkcionalnosti koje bi se mogle dodati jeste iscrtavanje samo određenog dela mreže. Kao i selektovanje proizvoljne površine na kanvasu i sakrivanje/prikazivanje dela elektrodistributivne mreže radi lakšeg snalaženja i orijentacije na kanvasu.

Što se tiče podataka, trenutno podaci mogu da se dodaju, modifikuju i brišu jedino kroz direktnu interakciju sa *Geographic.xml* fajlom. U tom smislu bilo bi dobro da se obezbede CRUD (*Create, Read, Update, Delete*) operacije kroz sam program tako da bi se korisniku olakšala manipulacija podataka.

LITERATURA

[1] ***Dragan Cvetković: Računarska grafika ISBN 86-7991-287-5, 2006***
[2] ***theoryofprogramming.com, Bidirectional Search, 2018***
[3] ***Svetlin Nakov i Veselin Kolev, Fundamentals of Computer Programming with CSharp, 2013***
[4] ***Thomas H. Cormen, Intorduction To Algorithms, Third Edition, 2009.***