FLOIDA-VARŠAĻA ALGORITMS ĪSĀKĀ MARŠRUTA APRĒĶINĀŠANAI STARP LATVIJAS PILSĒTĀM
FLOYD–WARSHALL ALGORITHM FOR SHORTEST ROUTE CALCULATION BETWEEN LATVIA'S CITIES

Autori: **Raivis GAVARS**, e-pasts: raivis.gavars1@gmail.com,
Einārs NETLIS-GALEJS, e-pasts: einarsng@inbox.lv,
Jānis Artūrs LAZDIŅŠ, e-pasts: janisarturslazdins@gmail.com
Darba vadītājs: Mg.math., Dr.paed., docents **Ilmārs KANGRO**
Rēzeknes Tehnoloģiju akadēmija,
Atbrīvošanas aleja 115, Rēzekne, Latvija

Abstract. *The Floyd–Warshall algorithm is a good choice for computing paths between all pairs of vertices indense graphs, in which most or all pairs of vertices are connected by edges. For sparse graphs with non-negative edgeweights, a better choice is to use Dijkstra's algorithm from each possible starting vertex. Also, a very good thing is that thesolution is very accurate, when using a computer. In this paper, the authors tried to apply a solution using C++programming language to make possible many entries.*

Keywords: *Algorithms; C++; Dijkstra; Floyd-Warshall; programming.*

Ievads

Ar Floida-Varšala algoritma palīdzību ir iespējams atrast īsāko ceļu (attālums starp 2. virsotnēm) no katras grafa virsotnes līdz visām pārējām virsotnēm. Algoritms saglabā datus 2D masīvā kā īsākos attālumus starp visām grafa virsotnēm.

Sākotnēji tajā ievieto ievada datus, un tad algoritma darbība norisinās,izvēloties katras trīs virsotnes i, k un j. Apskata ceļa posmu no i līdz j, izvēloties starp-virsotni k, un ja ceļšno i līdz k ”plus” no k līdz j izrādās īsāks nekā no i līdz j, tad ceļš ar starp-virsotni k tiek uzskatīts par iepriekšējā ceļa uzlabojumu. Ar īsāko ceļu saprotam minimālo grafa posmu svaru summu. Pielietojot šo algoritmu jāievēro, ka k-tās – vidējās virsotnes ciklam jābūt pirmajam.

Referātā tiek apskatīts grafu teorijas pielietojums īsākā ceļa atrašanai starp grafa virsotnēm, izmantojot C++ programmēšanas valodu.

C++ vidē tika veiksmīgi izstrādāta programma, kura izveido ”logu”datuievadei un, izmantojot Floida-Varšaļa algoritmu, aprēķina īsāko ceļustarp grafa virsotnēm, un beigās izvada atrisinājumu skaitliskā veidā un vizuālā formā.

Darba mērķis:

- pilnveidot matemātikas zināšanas par grafu teorijas nodaļu – īsākā ceļa aprēķināšana;
- pilnveidot zināšanas par programmēšanas valodu C++ - realizēt konkrētu algoritmu īsākā ceļa aprēķināšanai grafā;
- izstrādāt aplikāciju 2. punkta programmas realizācijai.

Darba uzdevumi:

- iepzīties ar teorētisko materiālu, saistītu ar grafu teorijas lietojumiem īsākā ceļa aprēķināšanai starp grafa virsotnēm;
- izpētīt iepriekš apskatītā teorētisko materiāla pielietošanas iespējas īsāko maršrutu aprēķināšanai grafā, izmantojot Floida-Varšaļa algoritmu;
- uzrakstīt programmu C++ valodā 2. punkta izpildei:
  a) izveidot lietotājam draudzīgu teksta failu lasīšanas programmu parametru ievadei;
  b) realizēt Floida-Varšaļa algoritmu;
  c) izvadīt iegūtos rezultātus lietotājam saprotamā skaitliskā un vizuālā formā.

http://dx.doi.org/10.17770/het2019.23.4397

Algoritma izvēle

Īsākā ceļa aprēķināšanai starp dažādām virsotnēm pastāv dažādas teorēmas kuras visas ir iespējams realizēt C++ programmēšanas valodā.

Deikstra algoritms.

Deikstra algoritms strādā ar mērķi aprēķināt ceļu no vienas dotās virsotnes līdz pārējām.

Floida-Varšaļa algoritms.

Atšķirībā no Deikstra algoritma Floida-Varšaļa algoritms atļauj aprēķināt īsāko ceļu no jebkuras virsotnes uz jebkuru citu.

.NET vide un C++ programmēšanas valoda

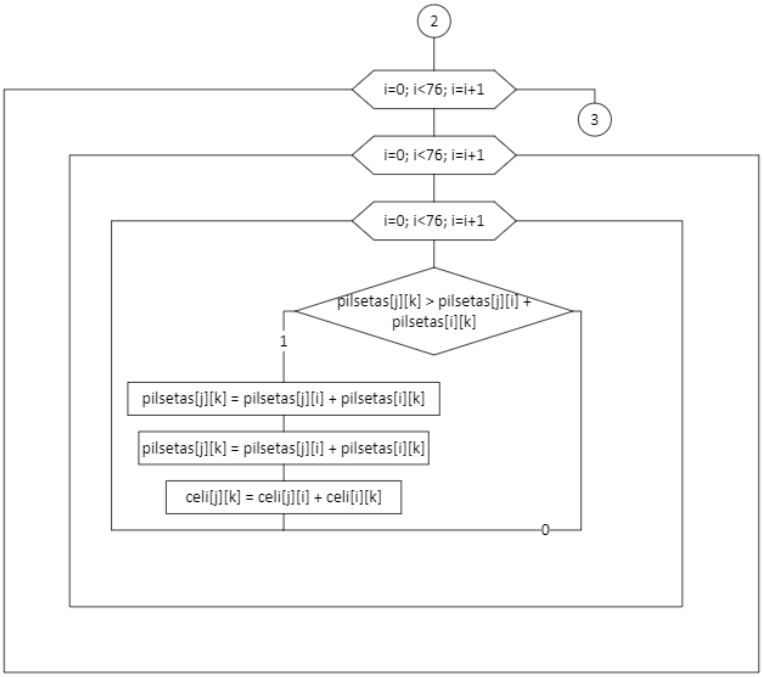
C++ ir objektorientēta programmēšanas valoda, kura ir paredzēta darbam ar .NET Framework platformas, kuras izveidoja Microsoft. .NET vide (izrunājama kā "dot net") ir programmu vide, kurā lielākoties strādā uz Microsoft Windows operētāj sistēmas. Tā satur lielu bibliotēku, ar kuras palīdzību ir iespējama dažādu programmēšanas valodu mijiedarbība.

C++ valoda ir plaši pazīstama un tiek arī izvēlēta tā paša iemesla dēl. Tās priekšrocības ir relatīvā viegluma pakāpe, kā arī ekstensīva standarta funkciju bibliotēka, pateicoties .NET platformai. Viens mīnuss ir tāds, ka, lai izmantot šādas aplikācijas, lietotājam uz datora jābūt ieinstalētam pareizās versijas .NET framework, bet gadījumā ja lieto Windows 8 un augstāk, tad tā jau nāk kopā ar OS. To var arī ielādēt no Microsoft web lapas par velti.

Programmu veidošana ar C++ parasti notiek ar Microsoft VisualStudio, programmēšanas vidi, kura ir cieši integrēta ar Microsoft valodām un .NET, kas ļauj intuitīvu projektu veidošanu un testēšanu.

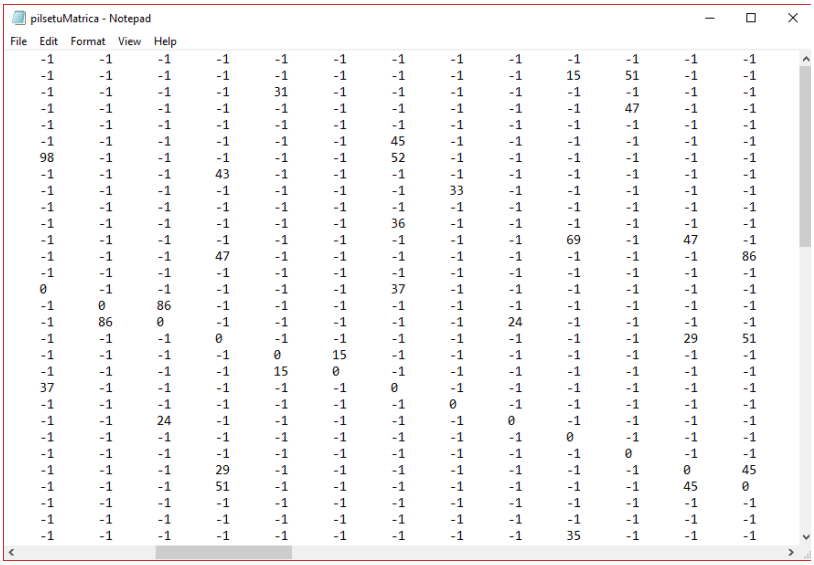
Blokshēma un izejkods

Gan karte, gan programma tiek balstīta uz pieņēmuma – iespējams pārvietoties abos virzienos.



1. attēls. **Floida-Varšaļa algoritma blokshēma**

Pilsētas, kuras nav savienotas, tika aizvietotas ar -1 vieglākai importēšanai programmā



2. attēls**. Sākotnējā matrica**

3. attēls**. Pilsētu saraksts**

Tika izveidots pilsetu sarakstu, kas tiks importēts programmā.[4]
Šī koda daļa tiek izpildīta, tikko palaižot programmu.

```
private: System::Void Saskarne_Load(System::Object^  sender, System::EventArgs^  e) {
    array<String^>^ pilsetuSaraksts;
    bool failsNavAtrasts = false;
    try {
        pilsetuSaraksts = System::IO::File::ReadAllLines(L"pilsetas.txt");
        // izmanto sistēmas funkcijas lai aizpildītu sarakstu saskarne
    }
    catch (...)
    {
        failsNavAtrasts = true;
    }
    if (!failsNavAtrasts)
    {
        this->comboBox1->Items->AddRange(pilsetuSaraksts);
        // izveido sarakstu ar pilsetam comboBox elementaa
        this->comboBox2->Items->AddRange(pilsetuSaraksts);

        // aprekinasanas sakums
        std::ifstream ifs("pilsetas.txt");
        // atver failu kurs satur pilsetu sarakstu
        std::string line;
        msclr::interop::marshal_context convert;
        // mainiga parveidotajs lai nodrosinatu savienojamibu ar objektiem
        //int sakumaIndex = -1, beiguIndex = -1;
        double a;

        if (ifs.is_open())
            // ja fails ir atrasts turpinat
        {
            for (int i = 0; getline(ifs, line); i++)
                // nolasa visas rindas no pilsetas.txt un saglaba uz masiva
            {
                pilsetuSarakstsNosauk[i] = line;
            }
            ifs.close();

            ifs.open("pilsetuMatrica.txt");
            // pec tam kad nolasits pilsetu saraksts atver nakamo failu kurs visus datu ka pilsetas ir savienotas
            if (ifs.is_open())
            {
                for (int i = 0, j = 0; ifs >> a; i++)
                {
                    if (a == -1)
        // ja nolasitais elements ir -1 tad matricas elementam tika pieskirta bezgaliba lai neizmantotu so elementu apreikin
                        pilsetas[j][i] = std::numeric_limits<double>::infinity();
                    else // preteji ievietot nolasito elementu matrica
                    {
                        pilsetas[j][i] = a;
                        celi[j][i] = pilsetuSarakstsNosauk[i];
                    }
                    if (i == 75) // ja ir sasniegts rindas beigas tad parej uz nakamo rindu
                    {
                        i = -1;
                        j++;
                    }
                }
            }
            else
            {
                MessageBox::Show(L"Fails kurš satur pilsētu matricu netika atrasts.", L"Paziņojums", MessageBoxButtons::OK,
            }
            ifs.close();
            // ↓ aprekinu dala ↓
            for (int i = 0; i < 76; i++)
            {
                for (int j = 0; j < 76; j++)
                {
                    for (int k = 0; k < 76; k++)
                    {
                        if (pilsetas[j][k] > pilsetas[j][i] + pilsetas[i][k])
                        {
                            pilsetas[j][k] = pilsetas[j][i] + pilsetas[i][k];
                            celi[j][k] = celi[j][i] + "," + celi[i][k];
                        }
                    }
                }
            }
            // ↑ aprekinu dala ↑
        }
        else
            MessageBox::Show(L"Fails kurš satur pilsētu sarakstu netika atrasts.", L"Paziņojums", MessageBoxButtons::OK, Mes

    }
    else
        MessageBox::Show(L"Fails kurš satur pilsētu sarakstu netika atrasts.", L"Paziņojums", MessageBoxButtons::OK, Message
}
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
```

***4. attēls.* Programmas starta kods**

Šī koda daļa tiek izpildīta, kad tiek uzspiesta poga “Aprēķināt”.

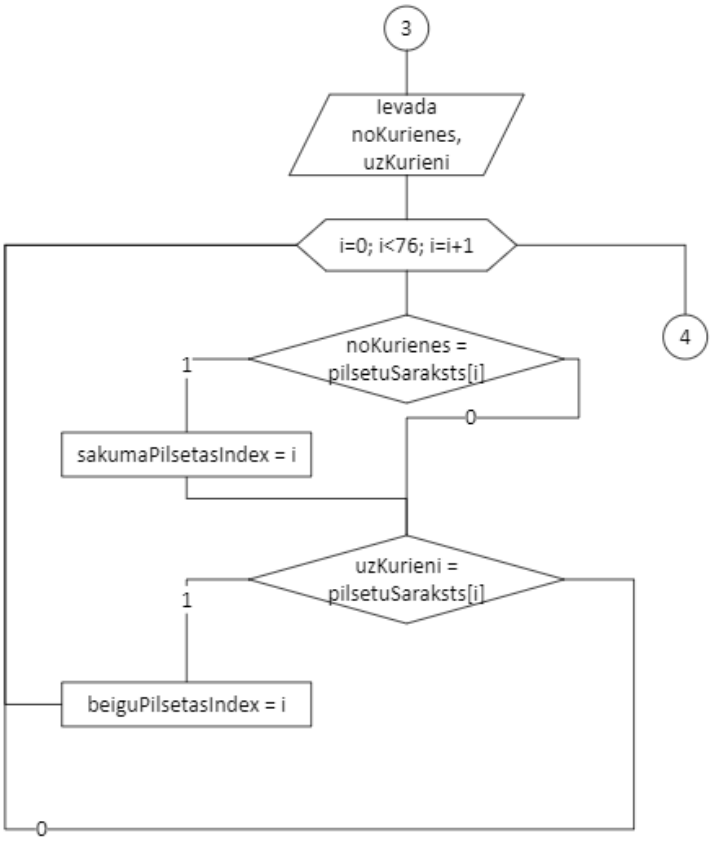
```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
    msclr::interop::marshal_context convert;
    std::string sakums = convert.marshal_as<std::string>(this->comboBox1->Text); // tiek parveidots mainīgais
    std::string beigas = convert.marshal_as<std::string>(this->comboBox2->Text); // tiek parveidots mainīgais
    std::string izveletaisCels, temp = "";
    int sakumaIndex = -1, beiguIndex = -1, i, j;  // default vertibas

    for (i = 0; i < 76; i++) // atrod pilsetu indeksus kuri tika izveleti
    {
        if (pilsetuSarakstsNosauk[i] == sakums)
            sakumaIndex = i;
        if (pilsetuSarakstsNosauk[i] == beigas)
            beiguIndex = i;
    }

    if (sakumaIndex != -1 && beiguIndex != -1) // ja tika izveletas pilsetas turpinat koda izpildi
    {
        //std::cout << int(pilsetas[sakumaIndex][beiguIndex]) << std::endl;
        this->rezultats->Text = convert.marshal_as<String^>(std::to_string(int(pilsetas[sakumaIndex][beiguIndex]))) + L" km"; // saskaras tiks parradits kopejais marsruta garums

        izveletaisCels = pilsetuSarakstsNosauk[sakumaIndex] + "," + celi[sakumaIndex][beiguIndex]; // viss marsruta saraksts
        for (i = 0, j = 0; i < (int)izveletaisCels.length(); i++)
        {
            if (izveletaisCels.c_str()[i] == ',') // komats tika izmantots lai atdalitu pilsetas (formatejums)
            {
                this->listBox1->Items->Add(convert.marshal_as<String^>(std::to_string(j + 1)) + "\t" + convert.marshal_as<String^>(temp)); // pievieno pilsetu sarakstam
                j++;
                temp = "";
            }
            else
                temp += izveletaisCels.c_str()[i]; // pilsetas nosaukums tika veidots pa burtiem
        }
        this->listBox1->Items->Add(convert.marshal_as<String^>(std::to_string(j + 1)) + "\t" + convert.marshal_as<String^>(temp)); // pievieno pedejo pilsetu sarakstam
        temp = "";
    }
    else
        MessageBox::Show(L"Netika izvēlēta sākuma un/vai beigu pilsēta.", L"Paziņojums", MessageBoxButtons::OK, MessageBoxIcon::Exclamation); // preteja gadijuma pazinot ka nav izveletas pilsetas
```

5. attēls. **Programmas aktīvā daļa**



6. attēls. **Pilsētu indeksu meklēšana**

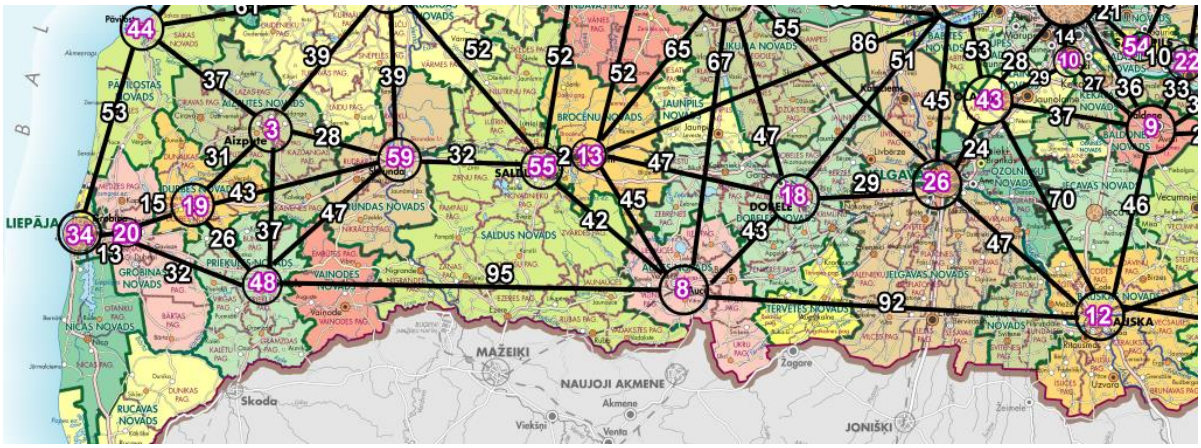
Kad lietotājs ir ievadījis sākuma un beigu pilsētu nosaukumus, šo pilsētu nosaukumi tiek salīdzināti ar sarakstu (3. attēls.) un tiek atrasti attiecīgi indeksi.

```
// tiek aprekinats isakais cels
        for (int i = 0; i < 76; i++)
        {
           for (int j = 0; j < 76; j++)
           {
              for (int k = 0; k < 76; k++)
              {
                 if (pilsetas[j][k] > pilsetas[j][i] + pilsetas[i][k])
                 {
                    pilsetas[j][k] = pilsetas[j][i] + pilsetas[i][k];
                    celi[j][k] = celi[j][i] + "\n\t" + celi[i][k];
                 }
              }
           }
        }
        //
         cout << "Ievadiet no kuras pilsetas uz kuru tiks veikts cels:\nno: ";
        cin >> noKurienes;
        cout << "lidz: ";
        cin >> uzKurieni;[5]
```

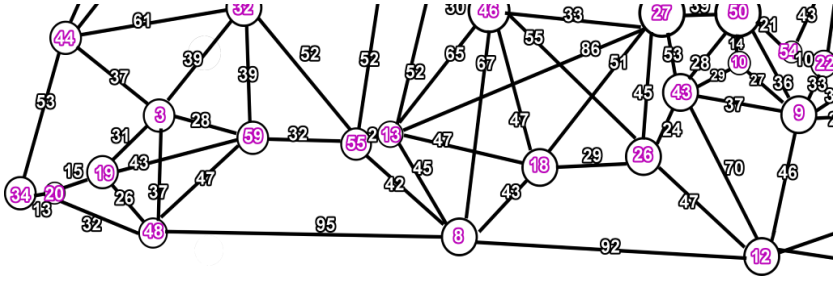
Šī koda daļa ir Floida-Varšaļa algoritms, kas aprēķina īsāko ceļu. Pirmā maršruta aprēķinātā vērtība tiek salīdzināta ar katru nākamo aprēķināto vērtību, ja šī vērtība ir mazāka par jau eksistējoši, tad tā tiek aizvietota.

Kontrolpiemēru un izejkoda rezultāti

Ceļi un to maršuti ir balstītit uz Google Earth kartes, jau sagatavotiem ceļiem. Lai pārliecinātos, ka ir iekļauti visi pieejamie ceļi, karte tiek pārbaudīta, izmantojot mobīlo aplikāciju Waze. Ja maršrutā starp 2. pilsētām ir jābrauc caur trešo pilsētu, šis maršruts tiek dalīts 2. daļās. Ir iespējams, ka pastāv vairāk variāciju, kuras var iekļaut programmā un kartes izveidošanā, taču darba veidotājiem nav iespējams pārbaudīt visus Latvijas ceļus. Veidot daļu no kartes precīzāku būtu neobjektīvi, tāpēc karte tika veidota izmantojot vienu metodi, kuru var pielietot visā Latvijā. Par sākuma un beigu punktu tiek uzskatīti pilsētas centri. Pilsētas tiek numurētas alfabēta secībā._Shematisks zīmējus var būt maldinošs un var tikt izmantots tikai attāluma attēlošanai. [6]



7. attēls. **Kartes daļa**



8. attēls. **Shematisks attēls**

Izmantojot 3. un 4. attēlu, varētu pieņemt, ka īsākais maršruts starp 34.(Liepāja) un 12. (Bauska) ir aprēķināts šādi: 13(->20) + 32(->48) + 95(->8) + 92(->12) = 232. Taču programmas rezultējošā vērtība ir 228 = 13(->20) + 15(->19) + 43(->59) + 32(->55) + 2(->13) + 47(->18) + 29(->26) + 47(->12).

Secinājumi

1. Tika izpētītas Floida-Varšaļa algoritma iespējas īsāko maršrutu aprēķināšanai grafā.
2. Tika apzinātas iepriekš minēto teorētisko nostādņu praktiskās realizācijas iespējas – konkretizēts uzdevums par īsāko maršrutu atrašanu starp Latvijas pilsētām izmantojot Google Earth kartes.
3. Tika uzrakstīta programma C++ valodā grafu teorijas pielietojuma – Floida-Varšaļa algoritma praktiskai realizācijai.
4. Īsākā ceļa aprēķināšanā Floida-Varšaļa algoritma priekšrocība ir ātrdarbības laiks, salīdzinot ar Deikstras algoritmu, jo viss tiek aprēķināts vienu reizi.
5. Izveidotie testa piemēri apstiprināja uzrakstītās programmas precīzo darbību Floida-Varšaļa algoritma izpildē.

Summary

As starting point in our work we decided it will be making of a visual reprezentation of the map. Creating this map would make it easier to create the starting table (2. attēls) as well as cheking if programm works correctly. Initially a table containing all the possible ways was noted in an Excel spreadsheet. The value where the cities were not connected was replaced by -1 to simplify the import process. To import the values all the contents of the spreadsheet were copied over to a text file and then read by the software into a two dimensional array. Also a list containing all the cities was imported to show the path taken to get to the city specified. Using the Floyd-Warshall algorithm the shortest path was calculated to all cities in the list. After the calculation process the user was prompted to enter a starting city as well as a destination city after which the user was notified of the shortest path taken and the distance length in kilometers.

Izmantotie avoti

1. C++ [online] [Skatīts 10.04.2019.] Pieejams: https://msdn.microsoft.com/en-us/en-en/library/kx37x362.aspx
2. Overview of the .NET Framework [online] [Skatīts 14.04.2019.]
3. Pieejams: https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx
4. Floyd-Warshall Algorithm | Shortest Path Algorithm [Skatīts 04.04.2019.] https://www.gatevidyalay.com/floyd-warshall-algorithm-shortest-path-algorithm/
5. Latvijas pilētu saraksts.[Skatīts 14.04.2019.] Pieejams:http://www.pilsetas.lv/pilsetas
6. Deikstras algoritma paskaidrojums. [Skatīts 14.04.2019.] Pieejams:www.politeh.lv/home/download/deikstras.doc