KAUNO TECHNOLOGIJOS UNIVERSITETAS

KOMPIUTERIŲ KATEDRA

###### **Algoritmų sudarymas ir analizė**

###### Trečias laboratorinis darbas

Atliko: Aivaras Klišauskas IFK-8gr.

Kaunas 2011

# Užduotis

Varinato numeris : 17

1. Dijkstros algoritmas
2. Maksimalaus pelno gavimas vygdant kosminius skrydžius

# Dijkstros algoritmas

Dijkstros algoritmas– algoritmas randantis trumpiausius kelius nuo vienos viršūnės iki kitų svoriniame grafe su neneigiamais svoriais.

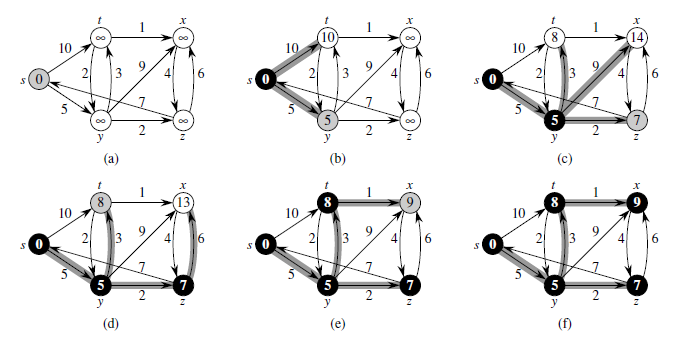
Pavyzdžiui, jei grafo viršūnės vaizduoja miestus ir kraštinių svoriai vaizduoja atstumą tarp tų miestų, sujungtą tiesioginiu keliu, Dijkstra algoritmas naudojamas surasti trumpiausius kelius tarp tų miestų.

Algoritmo pradiniai duomenys yra kryptinis grafas G su kraštinių svoriais ir šaltinių viršūne S grafe G. Mes pavadinsime V visų viršūnių rinkinį grafe G. Kiekviena grafo kraštinė yra sutvarkyta viršūnių pora (u, v), parodanti ryšį tarp viršūnių u ir v. Visų kraštinių rinkinys yra E. Kraštinių svoriai yra užduodami pagal svorių funkciją w:E; čia u(u, v) yra ne neigiama kaina tiesiogiai einant nuo viršūnės U iki viršūnės V. Kraštinės kaina yra laikoma atstumu tarp šitų dviejų viršūnių. Kelio kaina tarp dviejų viršūnių yra kraštinių kainų suma tame kelyje.

Duotai porai viršūnių s ir t visų viršūnių rinkinyje V algoritmas suranda kelią nuo s į t su mažiausia kaina (trumpiausias kelias). Jis taip pat gali būti naudojamas rasti trumpiausių kelių kainai nuo vienos viršūnės s iki kitų viršūnių grafe.

Algoritmas dirba surasdamas kiekvienai viršūnei trumpiausio kelio kainą d[v] rastą tame kelyje tarp s ir v. Iš pradžių ši vertė yra 0 šaltinio viršūnei s (d[s]=0) ir begalybė kitoms viršūnėms, pripažįstant faktą, kad mes nežinome jokių kelių iki tų viršūnių (d[v]=∞ kiekvienam v iš V, išskyrus s). Kai algoritmas baigsis, d[v] bus trumpiausias kelias nuo s iki v, ar begalybė, jei toks kelias neegzistuoja.

Pagrindinė Djikstros algoritmo operacija yra kraštinių atlaisvinimas, jei yra kraštinė nuo u iki v, tai trumpiausias žinomas kelias nuo s iki u (d[u]) gali būti išplėstas į kelią nuo s iki v, pridedant kraštinę (u, v) prie galo. Toks kelias turės ilgį d [u]+w(u, v). Jei tai yra mažiau negu esamas d[v], mes galime pakeisti esamą vertę d[v] nauja verte. Kraštinių atlaisvinimas yra taikomas, kol visos vertės d[v] atitinka trumpiausio kelio kainą nuo s iki v. Algoritmas yra organizuojamas taip, kad kiekviena kraštinė yra atlaisvinama tik vieną kartą, kai d[u] pasiekia savo galutinę vertę.



Pav. 2.1. Dijkstros algoritmo pavizdys

1. **Maksimalaus pelno gavimas vygdant kosminius skrydžius**

# Programos testas

|  |  |  |
| --- | --- | --- |
| **Nr** | **Briaunų Sk** | **Darbo Laikas** |
| 1 | 2 | 15541 |
| 2 | 3 | 21839 |
| 3 | 4 | 36128 |
| 4 | 5 | 39956 |
| 5 | 6 | 45898 |
| 6 | 7 | 51314 |
| 7 | 8 | 54618 |
| 8 | 9 | 69922 |

Lentelė 4.1 Deijkstors algoritmo testo duomenys

Pav 4.1 Dejkstros algotimo testo grafikas

# Tyrimo išvados.

Ištyrėme Dijkstros algoritmą ir ekperimentiškai nustatėme, kad labiausiai jo darbo laiką įtakoja mazgų kiekis – kvadratiškai, o briaunų kiekis tiesiškai, taigi patvirtinome teorinį įvertinimą O(V2 + E).

# Priedas

Programos kodas

package lab3;

/\*\*

\*

\* @author Aivaras

\*/

public class Main {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

ReadData test = new ReadData();

DijVertex []grafas = test.skaitytiDijFaila("Dij.txt");

int []mass = new int[grafas.length\*3];

mass = fillMass(mass);

Dijkstra.pajieska(grafas, 0, mass);

PGVertex []grafas2 = test.skaitytiPGjFaila("PG.txt");

PajieskaGilyn test1 = new PajieskaGilyn();

test1.pajieska(grafas2, 0, 0);

}

public static void print(DijVertex []grafas){

for(int i=0; i< grafas.length; i++){

System.out.print(i);

System.out.print(":");

grafas[i].print();

}

}

public static int[] fillMass(int []mass){

for(int i=0; i< mass.length; i++ ){

mass[i] = -1;

}

return mass;

}

public static void prtin(PGVertex []grafas ){

for(int i =0; i<grafas.length; i++){

System.out.print(i);

System.out.print(":");

grafas[i].print();

}

}

}

package lab3;

/\*\*

\*

\* @author Aivaras

\*/

public class Dijkstra {

// Kintamieji

// Funkcijos

public static void pajieska(DijVertex []g, int pradVirs, int [] mass){

DijEdge br; // briauna

int []seka = mass;

int laisViet = 0; //vieta sarase

int senasSvoris = 0;

int naujasSvoris =0;

seka[0] = pradVirs; // naudojama virsune

g[seka[0]].setSvoris(0);

for(int i = 0; i<seka.length; i++){

if(seka[i] == -1) return;

br = g[seka[i]].getGret();

while(br != null){

senasSvoris = g[br.getId()].getSvoris();

naujasSvoris = br.getSvoris()+g[seka[i]].getSvoris();

if(senasSvoris > naujasSvoris){

g[br.getId()].setSvoris(br.getSvoris()+g[seka[i]].getSvoris());

laisViet++;

seka[laisViet] =br.getId();

}

br = br.getKita();

}

}

}

}

package lab3;

import java.io.BufferedReader;

import java.io.DataInputStream;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.InputStreamReader;

/\*\*

\*

\* @author Aivaras

\*/

public class OpenFile {

// Kintamieji

protected BufferedReader failoStremas;

protected String duomFailas = "testas.txt";

// Funkcijos

OpenFile() {

failoStremas = null;

}

public void atverti(String failas){

duomFailas = failas;

try{

this.duomFailas = failas;

FileInputStream fstream = new FileInputStream(failas);

DataInputStream in = new DataInputStream(fstream);

BufferedReader br = new BufferedReader(new InputStreamReader(in));

failoStremas = br;

} catch(FileNotFoundException e){

System.out.print("failas nerastas: "+duomFailas);

}

}

public void closeFile(){

try {

failoStremas.close();

} catch (IOException ex) {

System.out.print("failo neoavyko uzverti sekmingai po skaitymo: "+duomFailas);

}

}

}

package lab3;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Aivaras

\*/

public class ReadData extends OpenFile {

// Kintamieji

// Funkcijos

public DijVertex[] skaitytiDijFaila(String failas){

String duom;

DijVertex []g = null;

int dydis, a, b;

atverti(failas);

int index =0;

try {

duom = failoStremas.readLine();

dydis = Integer.parseInt(duom);

g = new DijVertex [Integer.parseInt(duom)];

duom = failoStremas.readLine();

for(int i = 0; i< dydis; i++){

g[i] = new DijVertex();

index = 3;

while(duom.length()>index+1){

a = duom.codePointAt(index)-48;

b = duom.codePointAt(index+2)-48;

DijEdge rts = new DijEdge(a, b);

g[i].setGret(rts);

index = index+4;

}

duom = failoStremas.readLine();

}

} catch (IOException ex) {

Logger.getLogger(ReadData.class.getName()).log(Level.SEVERE, null, ex);

}

return g;

}

public PGVertex[] skaitytiPGjFaila(String failas){

String duom;

PGVertex []g = null;

int dydis, a;

atverti(failas);

int index =0;

try {

duom = failoStremas.readLine();

dydis = Integer.parseInt(duom);

g = new PGVertex [Integer.parseInt(duom)];

duom = failoStremas.readLine();

for(int i = 0; i< dydis; i++){

g[i] = new PGVertex();

index = 3;

while(duom.length()>index+1){

a = duom.codePointAt(index)-48;

DijEdge rts = new DijEdge(a, 0);

g[i].setGret(rts);

index = index+2;

}

duom = failoStremas.readLine();

}

} catch (IOException ex) {

Logger.getLogger(ReadData.class.getName()).log(Level.SEVERE, null, ex);

}

return g;

}

}

package lab3;

/\*\*

\*

\* @author Aivaras

\*/

public class PGVertex extends DijVertex {

// Kintamieji

private int inNr; // iejimo nr

private int outNr; // isejimo nr

private boolean apl;

// Funkcijos

PGVertex(){

inNr = outNr = 0;

apl = false;

}

public void setInNr(int nr){

this.inNr = nr;

}

public void setOutNr(int nr){

this.outNr = nr;

}

public void setAplankyta(){

this.apl = true;

}

public int getInNr(){

return this.inNr;

}

public int getOutNr(){

return this.outNr;

}

public boolean getAplankyta(){

return this.apl;

}

public void print(){

System.out.print(inNr);

System.out.print("/");

System.out.print(outNr);

System.out.print("\n");

}

}

package lab3;

/\*\*

\*

\* @author Aivaras

\*/

public class PajieskaGilyn {

// Kintamieji

// Funkcijos

public int pajieska(PGVertex []g, int pradVirs, int sk){

int inNr = sk +1 ;

DijEdge gret = g[pradVirs].getGret();

g[pradVirs].setInNr(inNr);

g[pradVirs].setAplankyta();

while(gret != null){

if(!g[gret.getId()].getAplankyta())

inNr = pajieska(g, gret.getId(), inNr);

gret = gret.getKita();

}

inNr = inNr +1 ;

g[pradVirs].setOutNr(inNr);

return inNr;

}

}

package lab3;

/\*\*

\*

\* @author Aivaras

\*/

public class DijEdge {

// Kintamieji

private int id;

private int svoris;

private DijEdge kita;

// Funkcijos

DijEdge(int id, int svoris){

kita = null;

this.id = id;

this.svoris = svoris;

}

public void setKita(DijEdge kita){

this.kita = kita;

}

public int getId(){

return id;

}

public int getSvoris(){

return svoris;

}

public DijEdge getKita(){

return kita;

}

}

package lab3;

/\*\*

\*

\* @author Aivaras

\*/

public class DijVertex {

// Kintamieji

protected int svoris;

protected DijEdge pirma;

protected DijEdge pask;

// Funkcijos

DijVertex(){

svoris = Integer.MAX\_VALUE;

pirma = null;

pask = null;

}

public void setSvoris(int svoris){

this.svoris = svoris;

}

public void setGret(DijEdge nauja){

if( pirma == null) {

pirma = nauja;

pask = nauja;

} else {

pask.setKita(nauja);

pask = nauja;

}

}

public int getSvoris(){

return svoris;

}

public DijEdge getGret(){

return pirma;

}

public void print(){

System.out.println(svoris);

}

}