Minimalna ortogonalna sieć połączeń

Adam Nowik

Koncepcja rozwiązania problemu

Minimalna ortogonalna sieć połączeń (AAL-W 13.)

Na płaszczyźnie dany jest zbiór punktów o współrzędnych całkowitych. Przez ścieżkę pomiędzy parą punktów tego zbioru rozumiemy sekwencję odcinków równoległych do osi układu współrzędnych. Długość ścieżki to suma odległości odcinków wchodzących w jej skład. Ścieżka jest minimalna, jeżeli nie istnieje inna krótsza ścieżka łącząca tę parę punktów (może być wiele ścieżek minimalnych). Minimalną ortogonalną siecią połączeń dla zadanego zbioru punktów płaszczyzny nazywamy taki zbiór odcinków spinających zadane punkty, w którym każda para punktów ma połączenie minimalne i koszt całego zbioru (suma wszystkich odcinków) jest minimalny. Opracować algorytm znajdujący minimalną ortogonalną sieć połączeń.

Proponowane rozwiązanie:

I. Założenia i struktury danych:

Dane generowane do pliku tekstowego o dowolnej nazwie podczas działania programu (wybór opcji w menu). Dane wczytywane z pliku tekstowego o dowolnej nazwie podczas działania programu (wybór opcji w menu). Dane na których działa algorytm przechowywane w postaci listy (klasa vector z biblioteki STL) par liczb całkowitych (współrzędnych x,y). Dwie dodatkowe listy par liczb całkowitych (Xpoints, Ypoints). Dane wyjściowe w postaci ciągu odcinków (par wierzchołków sieci) przechowywane w liście par liczb całkowitych.

II. Algorytm:

1. Sortowanie listy punktów w pierwszej kolejności po x, w drugiej po y. (merge – sort)
2. Tworzymy 2 listy punktów, listę Xpoints i listę Ypoints (puste).
3. for (i =0; i < n; i++) {
4. Wybieramy i – ty punkt z listy
5. Czyścimy Xpoints i Ypoints.
6. Badamy czy żadne wierzchołki nie znajdują się w linii prostej ortogonalnej od wybranego, jeśli tak, dopisujemy pary wierzchołków do sieci.
7. for (j = i+1; j < n; j++) {
8. Wybieramy j – ty element z listy,
9. Sprawdzamy czy w sieci nie istnieje już ścieżka zawierający badane punkty, spełniająca warunki kraty typu manhattan rozpiętej między badanymi punktami, jeśli tak, to j++, przechodzimy do kroku 8. jeśli nie:
10. Dopisujemy wierzchołki kraty, jakie punkt j – ty tworzy z punktem i – tym :

wierzchołek o współrzędnej x takiej samej jak punkt i – ty do listy Xpoints,
wierzchołek o współrzędnej y takiej samej jak punkt i – ty do listy Ypoints.

11. Jeśli w kracie znajduje się fragment sieci, (jeśli nie jest połączony łączymy go odcinkiem) wybieramy punkt z tego fragmentu sieci znajdujący się najbliżej punktu j – tego. Badamy czy można połączyć te punkty odcinkiem, jeśli tak, to dopisujemy odcinek do sieci, i przechodzimy do kolejnej iteracji (pkt 8). Jeśli nie, powtarzamy algorytm od pkt 8. dla wybranego punktu. Jeśli doszliśmy do końca listy punktów, powrót do kroku 4.

}

12. Wybieramy punkt z list Xpoints i Ypoints, którego odległość od punktu i – tego jest najmniejsza, powrót do kroku 5.

}

Słowny opis oraz określenie i uzasadnienie wzoru na dokładną złożoność algorytmu:

Pierwszym krokiem mojego algorytmu jest wstępne posortowanie punktów za pomocą algorytmu merge-sort (o złożoności n*logn). Następnie dla każdego punktu znajdującego się na liście danych, przeglądane są wszystkie kolejne punkty na liście, w celu sprawdzenia czy możliwe jest przeprowadzenie pomiędzy nimi odcinka ortogonalnego, w najbardziej pesymistycznym przypadku n-1 punktów. Następnie dla każdego z tych punktów tworzone są 2 listy punktów będących wierzchołkami krat typu Manhattan (w przypadku pesymistycznym 2*(n-1)), a następnie przeglądane w poszukiwaniu punktu najbliższego, (2*(n-1)). Kolejnym krokiem jest przeszukanie odcinków sieci w poszukiwaniu punktu znajdującego się w sieci, spełniającego warunki kraty typu Manhattan rozpiętej między punktami, między którymi tworzony jest nowy fragment sieci, w przypadku najbardziej pesymistycznym wymaga to przejrzenia 2*(n) punktów. Cała procedura powtarzana jest n razy. Ze wzoru na skończoną sumę ciągu arytmetycznego otrzymujemy wzór na złożoność:

$T(n) = n*\log n + (n*((n-1) +2*(n-1)+ 2*(n-1) +2*(n)))/2$

$T(n) = n*\log n + (7n*n – 5*n)/2$

$T(n) = n^2$

Interpretacja otrzymanych wyników:

n:	t(n)[ms]:	Q(n):
1000	910	0,550771
2000	3319	0,502466
3000	9352	0,629376
4000	20078	0,760146
5000	37540	0,909664
6000	63151	1,062740
7000	97064	1,200120
8000	142342	1,347500
9000	199670	1,493530
10000	271630	1,645780

Otrzymane wyniki pomiarów czasów dla odpowiednich ilości badanych punktów zostały przedstawione w powyższej tabeli. Analiza powyższych danych pozwala mi stwierdzić, że szacowana złożoność obarczona jest niewielkim błędem niedoszacowania.

Table of Contents

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all documented files with brief descriptions:

Class Documentation

Generator Class Reference

`#include <Generator.h>`

Public Member Functions

- **Generator** ()
- **Generator** (string)
- **~Generator** ()
- void **setName** (string)
- void **generate** ()
- void **readData** (vector< **Para** > &)
- void **mergeSort** (vector< **Para** > &)

Private Attributes

- string **nazwa**

Detailed Description

Obiekty klasy **Generator** odpowiedzialne sa za tworzenie plikow zawierajacych parzysta, losowa liczbe losowo wygenerowanych liczb calkowitych, odczytywanie ich i na ich podstawie tworzenie list punktow o losowych wspolrzednych, obiekty te maja takze mozliwosc sortowania list punktow (za pomoca algorytmu Merge-Sort).

Constructor & Destructor Documentation

Generator::Generator ()

Konstruktor domyslny obiektow klasy **Generator**.

Generator::Generator (string *name*)

Konstruktor jednoargumentowy klasy **Generator**. Obiekt utworzony za pomoca tego generatora bedzie dzialal na pliku o nazwie przekazanej jako parametr.

Parameters:

string nazwa pliku.

Generator::~Generator ()

Destruktor obiektu klasy **Generator**.

Member Function Documentation

void Generator::generate ()

Metoda sluzaca generacji parzystej, losowej liczby wartosci calkowitych i zapisaniu jej do pliku o nazwie przekazywanej jako parametr.

See also:

nazwa

void Generator::mergeSort (vector< Para > & *pomoc*)

Metoda slozaca do sortowania punktow za pomoca algorytmu Merge-Sort, na liscie punktow przekazanej jako parametr.

Parameters:

vector & referencja na liste na ktorej dane maja byc posortowane.

See also:

Para

void Generator::readData (vector< Para > & *kolejka*)

Metoda sluzaca do odczytu danych z pliku o nazwie przechowywanej w atrybucie nazwa. I umieszczenia ich jako listy punktow w obiekcie klasy vector<Para> do ktorego referencja przekazywana jest jako parametr.

Parameters:

vector & referencja na liste do ktorej maja byc zapisane dane z pliku.

See also:

nazwa
Para

void Generator::setName (string *name*)

Metoda ustawiajaca parametr nazwa na wartosc podana jako parametr. **Generator** bedzie dzialal na pliku o takiej nazwie.

Parameters:

string nazwa pliku.

See also:

nazwa

Member Data Documentation

string Generator::nazwa `[private]`

Nazwa pliku na ktorym ma dzialac obiekt klasy **Generator**.

The documentation for this class was generated from the following files:

- **Generator.h**
- **Generator.cpp**

Net Class Reference

`#include <Net.h>`

Public Member Functions

- **Net** ()
- **~Net** ()
- void **createNet** (vector< **Para** > &)
- bool **isLine** (**Para** &, **Para** &)
- bool **isBetween** (**Para** &, **Para** &, **Para** &)
- **Para findNearest** (**Para** &)
- void **findNearestInLists** (**Para** &)
- void **createRoute** (vector< **Para** > &, **Para** &, int)
- void **drawNet** (vector< **Para** > &)

Private Attributes

- vector< **Para** > **xList**
- vector< **Para** > **yList**
- **Para activePoint**
- **Para nearestPoint**
- **Para nearestPointInList**
- vector< **Para** > **net**

Detailed Description

Klasa **Net** umozliwia tworzenie minimalnej ortogonalnej sieci pomiedzy zbiorem punktow na plaszczyznie.

See also:

Para

Constructor & Destructor Documentation

Net::Net ()

Konstruktor domyslny obiektu klasy **Net**.

Net::~Net ()

Destruktor obiektu klasy **Net**.

Member Function Documentation

void Net::createNet (vector< Para > & *lista*)

Metoda tworzaca minimalna ortogonalna siec dla listy punktow podanych jako parametr.

Parameters:

vector & referencja na liste punktow.

See also:

Para

void Net::createRoute (vector< Para > & *lista*, Para & *target*, int *i*)

Metoda tworzaca sciezke pomiedzy punktami wywolujaca sie rekurencyjnie

Parameters:

vector & referencja na liste punktow dla ktorych tworzona jest siec.
Para& referencja na punkt od ktorego tworzone sa polaczenia do nastepnych punktow w sieci.
int numer elementu sieci od ktorego tworzone sa polaczona od badanego punktu

See also:

Para

void Net::drawNet (vector< Para > & *lista*)

Metoda rysujaca siec w pliku SVG (tworzaca plik i parsujaca go).

Parameters:

vector & lista punktow miedzy ktorymi rysowana jest siec.

See also:

Para

Para Net::findNearest (Para & *badany*)

Metoda znajdujaca i zwracajaca punkt znajdujacy sie najblizej punktu podanego jako parametr.

Parameters:

Para& referencja na punkt, dla ktorego poszukiwany jest punkt najblizszy znajdujacy sie w sieci.

Returns:

Para punkt znajdujacy sie w sieci, polozony najblizej punktu podanego jako parametr.

See also:

Para

void Net::findNearestInLists (Para & *target*)

Metoda znajdujaca w listach xList i yList punkt znajdujacy sie najblizej punktu podanego jako parametr.

Parameters:

Para& referencja na punkt dla ktorego poszukiwany jest punkt najblizszy sposrod punktow z xList i yList.

See also:

Para
xList
yList

bool Net::isBetween (Para & *badany*, Para & *sectionStart*, Para & *sectionEnd*)

Metoda sprawdzajaca czy punkt podany jako pierwszy parametr lezy na odcinku utworzonym przez punkty podane jako parametry drugi i trzeci.

Parameters:

Para& referencja na pierwszy punkt.
Para& referencja na drugi punkt.
Para& referencja na trzeci punkt.

Returns:

True jesli punkt lezy na odcinku utworzonym przez punkty drugi i trzeci, false jesli lezy poza tym odcinkiem.

See also:

Para

bool Net::isLine (Para & *punkt1*, Para & *punkt2*)

Metoda sprawdzajaca czy przez dwa punkty mozna poprowadzic ortogonalna linie prosta.

Parameters:

Para& referencja na pierwszy punkt dla ktorego jest badane istnienie ortogonalnej linii prostej.
Para& referencja na drugi punkt dla ktorego jest badane istnienie ortogonalnej linii prostej.

Returns:

True jesli mozna utworzyc linie prosta, false jesli utworzenie linii jest niemozliwe.

See also:

Para

Member Data Documentation

Para Net::activePoint `[private]`

Punkt, dla ktorego rozpinamy siec.

See also:

Para

Para Net::nearestPoint `[private]`

Najblizszy punkt w sieci pomiedzy punktem aktywnym a aktualnie badanym podczas rozpinania sieci punktem.

See also:

Para

Para Net::nearestPointInList `[private]`

Najblizszy punkt wsrod punktow z list xList i yList dla punktu activePoint.

See also:

Para
xList
yList

vector<Para> Net::net `[private]`

Lista punktow przechowujaca odcinki sieci (kazde dwa sasiednie punkty to poczatek i koniec odcinka)

See also:

Para

vector<Para> Net::xList `[private]`

Lista punktow bedacych wierzcholkami krat typu manhattan pomiedzy punktami dla ktorych klasa **Net** tworzy minimalna ortogonalna siec sciezek. Punkty na tej liscie posiadaja taka sama wspolrzedna X co punkt dla ktorego rozpinamy siec.

See also:

Para

vector<Para> Net::yList `[private]`

Lista punktow bedacych wierzcholkami krat typu manhattan pomiedzy punktami dla ktorych klasa **Net** tworzy minimalna ortogonalna siec sciezek. Punkty na tej liscie posiadaja taka sama wspolrzedna Y co punkt na ktorego rozpinamy siec.

See also:

Para

The documentation for this class was generated from the following files:

- **Net.h**
- **Net.cpp**

Para Class Reference

`#include <Para.h>`

Public Member Functions

- **Para** ()
- **Para** (int, int)
- **~Para** ()
- int **getX** ()
- int **getY** ()
- void **setX** (int)
- void **setY** (int)
- bool **operator==** (const **Para** &)
- bool **operator!=** (const **Para** &)
- bool **operator<** (const **Para** &)
- bool **operator>** (const **Para** &)
- bool **operator<=** (const **Para** &)

Private Attributes

- int **x**
- int **y**

Detailed Description

Klasa **Para** sluzy do tworzenia obiektow przechowujacych pary liczb, ktore reprezentuja punkt na plaszczyznie. Miedzy obiektami tej klasy rozpinana jest minimalna ortogonalna siec polaczen.

See also:

Net

Constructor & Destructor Documentation

Para::Para ()

Konstruktor domyslny obiektu klasy **Para**

Para::Para (int *i*, int *j*)

Konstruktor dwuargumentowy obiektu klasy **Para**. Tworzy obiekt klasy para o wspolrzednych rownych liczbom calkowitym przekazanym jako parametry.

Parameters:

int wspolrzedna X.
int wspolrzedna Y.

Para::~Para ()

Destruktor obiektu klasy **Para**.

Member Function Documentation

int Para::getX ()

Metoda zwracajaca wspolrzedna X obiektu.

Returns:

int wspolrzedna X.

int Para::getY ()

Metoda zwracajaca wspolrzedna Y obiektu.

Returns:

int wspolrzedna Y.

bool Para::operator!= (const Para & *cel*)

Przeciazany operator != dla obiektow typu **Para**.

Returns:

True jesli wspolrzedne X lub Y dwoch punktow sa rozne, false w przeciwnym przypadku.

bool Para::operator< (const Para & *cel*)

Przeciazany operator < dla obiektow typu **Para**.

Returns:

True jesli lewy operand operatora jest mniejszy niz prawy (ma mniejsza wspolrzedna X, lub rowna wspolrzedna X i mniejsza wspolrzedna Y), false w przeciwnym przypadku.

bool Para::operator<= (const Para & *cel*)

Przeciazany operator <= dla obiektow typu **Para**.

Returns:

True jesli lewy operand jest mniejszy lub rowny prawemu operandowi, false w przeciwnym przypadku.

bool Para::operator== (const Para & *cel*)

Przeciazany operator == dla obiektow typu **Para**.

Returns:

True jesli wspolrzedne X i Y dwoch punktow sa takie same, false w przeciwnym przypadku.

bool Para::operator> (const Para & *cel*)

Przeciazany operator > dla obiektow typu **Para**.

Returns:

True jesli lewy operand operatora jest wiekszy niz prawy (ma wieksza wspolrzedna X, lub rowna wspolrzedna X i wieksza wspolrzedna Y), false w przeciwnym przypadku.

void Para::setX (int *aX*)

Metoda ustawiajaca wspolrzedna X obiektu, na wartosc przekazana jako parametr.

Parameters:

int wspolrzedna X.

void Para::setY (int *aY*)

Metoda ustawiajaca wspolrzedna Y obiektu, na wartosc przekazana jako parametr.

Parameters:

int wspolrzedna Y.

Member Data Documentation

int Para::x `[private]`

Wspolrzedna X punktu.

int Para::y `[private]`

Wspolrzedna Y punktu.

The documentation for this class was generated from the following files:

- **Para.h**
- **Para.cpp**

Timer Class Reference

`#include <Timer.h>`

Public Member Functions

- **Timer** ()
- **~Timer** ()
- void **setStart** ()
- void **setEnd** ()
- double **getTime** (int)
- void **calculateTime** (int)
- void **calculateExactTime** (int, int)
- void **calculateQ** ()
- void **writeToFile** ()

Private Attributes

- double **timetab** [10]
- timeval **start**
- timeval **end**
- double **etimetab** [10]
- double **qtab** [10]
- int **amount** [10]

Detailed Description

Klasa **Timer** pozwala na tworzenie obiektow rejestrujacych czasy oraz wyliczajacych zlozonosc algorytmu tworzenia minimalnej ortogonalnej sieci polaczen.

Constructor & Destructor Documentation

Timer::Timer ()

Konstruktor domyslny obiektu typu **Timer**.

Timer::~Timer ()

Destruktor obiektu typu **Timer**.

Member Function Documentation

void Timer::calculateExactTime (int *i*, int *n*)

Obliczenie T(n)

Parameters:

int indeks do ktÃ³rego zostanie zapisany T(n)
int iloÅ›Ä‡ punktÃ³w dla jakiej uruchomiono algorytm.

void Timer::calculateQ ()

Obliczenie q

void Timer::calculateTime (int *i*)

Obliczenie czasu trwania algorytmu. Zapisanie wartosci w tablicy timetab.

Parameters:

int indeks pod ktorym zapisywany jest czas trwania.

double Timer::getTime (int *i*)

Metoda zwracajaca wartosc czasu trwania algorytmu spod indeksu podanego jako parametr.

Parameters:

int wartosc indeksu spod ktorego zwracany jest czas algorytmu.

Returns:

double czas algorytmu (w milisekundach)

void Timer::setEnd ()

Zarejestrowanie chwili zakonczenia algorytmu.

void Timer::setStart ()

Zarejestrowanie chwili startu algorytmu.

void Timer::writeToFile ()

Zapisanie wynikÃ³w pomiarÃ³w do pliku

Member Data Documentation

int Timer::amount[10] `[private]`

Tablica przechowujaca ilosc punktow dla jakiej uruchomiono algorytm.

timeval Timer::end `[private]`

Rejestruje chwile, z ktora algorytm wyznaczania sieci konczy swoje dzialanie.

double Timer::etimetab[10] `[private]`

Tablica dokÅ‚adnych czasÃ³w trwania algorytmu.

double Timer::qtab[10] `[private]`

Tablica q.

timeval Timer::start `[private]`

Rejestruje chwile, z ktora startuje algorytm wyznaczania sieci.

double Timer::timetab[10] `[private]`

Tablica czasow (w milisekundach) trwania dzialania algorytmu wyznaczania sieci.

The documentation for this class was generated from the following files:

- **Timer.h**
- **Timer.cpp**

File Documentation

Generator.cpp File Reference

```
#include "Includes.h"
#include "Generator.h"
#include "Para.h"
```

Detailed Description

Plik zawierajacy implementacje metod klasy **Generator**. Klasa **Generator** sluzy do generowania, wczytywania i sortowania danych.

Date:

2009-06-03

Author:

Adam Nowik

Generator.h File Reference

```
#include "Includes.h"
#include "Para.h"
```

Classes

- class **Generator**

Detailed Description

Plik naglowkowy klasy Generetor. Plik zawiera definicje ciala klasy **Generator**, oraz prototypy metod tej klasy.

Date:

2009-06-03

Author:

Adam Nowik

Includes.h File Reference

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string>
#include <sys/time.h>
#include <vector>
#include <math.h>
```

Defines

- #define **MAKS_N** 50;
- #define **MAKS_J** 600;
- #define **INF** 9999999;

Detailed Description

Plik zawierajacy instrukcje dolaczajacy biblioteki potrzebne do prawidlowego dzialania programu.

Date:

2009-06-03

Author:

Adam Nowik

Net.cpp File Reference

```
#include "Includes.h"
#include "Net.h"
#include "Para.h"
```

Detailed Description

Plik zawierajacy implementacje klasy **Net**. Klasa **Net** pozwala na utworzenie minimalnej ortogonalnej sieci dla punktow przechowywanych w obiekcie klasy vector z biblioteki STL.

Date:

2009-06-04

Author:

Adam Nowik

Net.h File Reference

```
#include "Includes.h"
#include "Para.h"
```

Classes

- class **Net**

Detailed Description

Plik naglowkowy klasy **Net**. Zawiera definicje ciala klasy **Net** oraz prototypy metod tej klasy.

Date:

2009-06-03

Author:

Adam Nowik

Para.cpp File Reference

```
#include "Includes.h"
#include "Para.h"
```

Detailed Description

Plik zawierajacy implementacje metod klasy **Para**. Obiekty klasy **Para** reprezentuja punkty na plaszczyznie.

Date:

2009-06-03

Author:

Adam Nowik

Para.h File Reference

```
#include "Includes.h"
```

Classes

- class **Para**

Detailed Description

Plik naglowkowy klasy **Para**. Plik zawiera definicje ciala klasy **Para**, oraz prototypy metod tej klasy.

Date:

2009-06-03

Author:

Adam Nowik

Timer.cpp File Reference

```
#include "Includes.h"
#include "Timer.h"
```

Detailed Description

Plik zawierajacy implementacje metod klasy **Timer**. Klasa **Generator** sluzy do zliczania czasów i wyliczania q.

Date:

2009-06-08

Author:

Adam Nowik

Timer.h File Reference

```
#include "Includes.h"
```

Classes

- class **Timer**

Detailed Description

Plik zawierajacy definicje ciala kalsy **Timer**, oraz prototypu jej metod Obiekty klasy timer pozwalaja na rejestracje czasu dzialania algorytmu wyznaczania minimalnej ortogonalnej sieci oraz wyliczenie zlozonosci algorytmu.

Date:

2009-06-08

Author:

Adam Nowik

Index