Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Факультет Вычислительной математики и кибернетики

Отчёт по лабораторной работе №1

Реализация приоритетной очереди на основе d-кучи и её применение для реализации алгоритма Крускала

Выполнил:

студент ф-та ВМК гр. 82-01

Савичев М. Р.

Проверил:

ассистент каф. МО ЭВМ, ВМК

Кустикова В. Д.

Нижний Новгород

2013

# Введение

d-куча – представление взвешенного множества в виде d-арного корневого дерева, узлам которого ставятся во взаимно однозначное соответствие элементы рассматриваемого множества. Соответствие между узлами дерева обладает следующим свойством: ключ элемента, приписанного узлу, не превосходит ключей, приписанных его потомкам. Данное правило называет окученностью.

Алгоритм Крускала (или алгоритм Краскала) - алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа.

Разделенные множества абстрактная структура данных для представления непересекающихся множеств, которая будет использоваться данной лабораторной работе.

Итак, в данной лабораторной работе будет разработана приоритетная очередь, основанная на d-куче, которая будет использоваться в реализации алгоритма Крускала.

# Постановка задачи

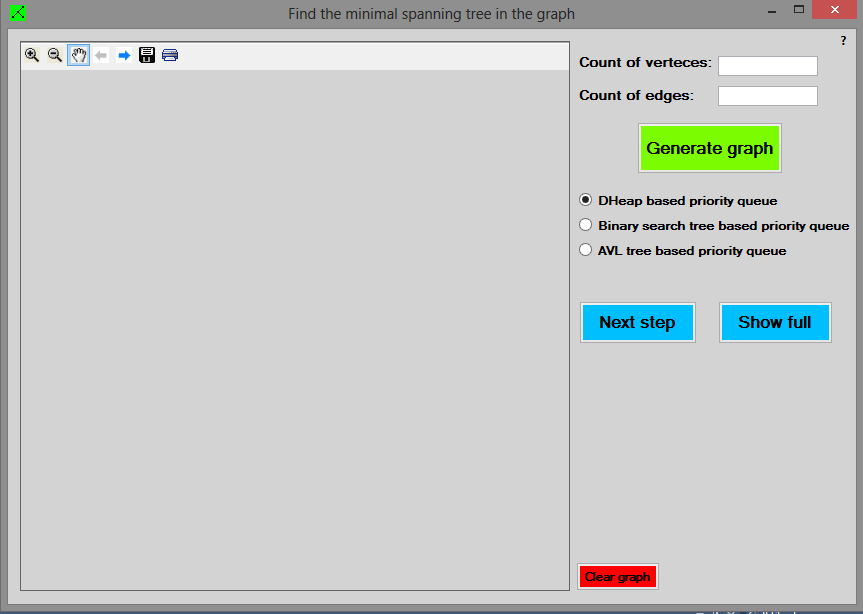
Целью данной лабораторной работы является разработка программы, иллюстрирующей работу алгоритма Крускала по поиску минимального остовного дерева во взвещенном неориентированном графе. Исходный граф необходимо генерировать случайным образом. Отображение на форме осуществить с помощью компоненты “Microsoft Automatic Graph Layout” (MSAGL). Ребра графа, принадлежащие остовному дереву, будут подсвечиваться красным цветом. Также реализовать возможность отображения следующего ребра остовного дерева по нажатию кнопки, то есть один клик – одно ребро.

Исходные данные:

* numVerteces – количество вершин;
* numEdges – количество ребер.

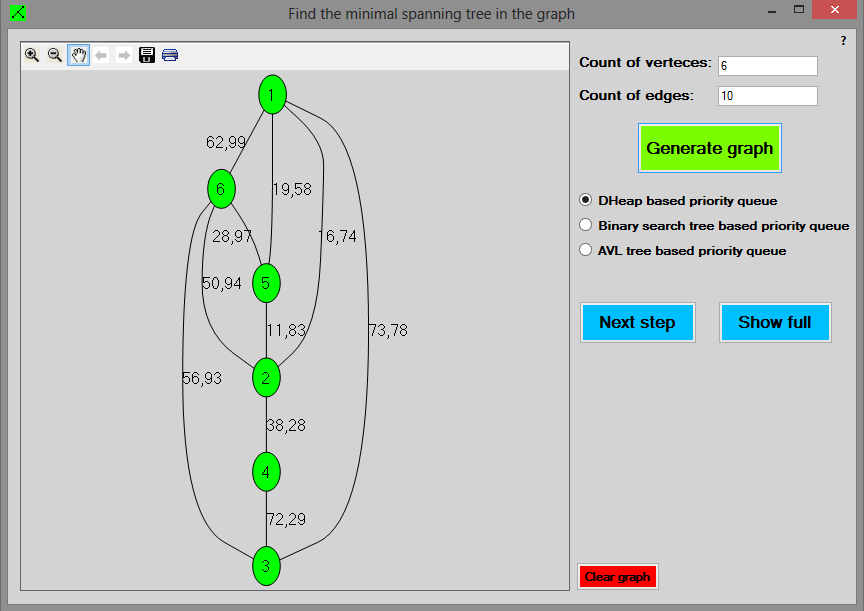
# Руководство пользователя

Чтобы воспользоваться реализованной программой необходимо запустить бинарный файл KruskalAlgorithm.exe. Необходимо, чтобы с ним в одной папке находились исполняемы файлы динамических библиотек компоненты MSAGL. После запуска появится окно, показанное на рис 1.



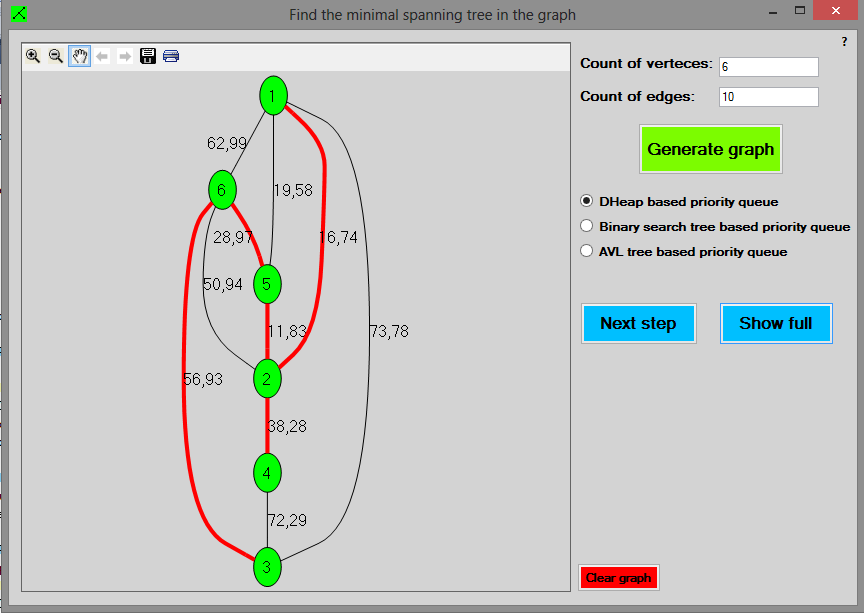
1. Начальное окно программы

Далее необходимо ввести исходные данные в специально предназначенные для этого поля и сгенерировать граф, нажав на кнопку “Generate graph”. Пример на рис 2.



1. Пример программы с исходными данными и сгенерерованным графом

На следующем шаге нужно с помощью переключателя выбрать тип приоритетной очереди, с помощью которой будет производиться вычисление остовного дерево. В данной лабораторной работе мы используем d-кучу, поэтому нужно выбрать первый вариант. Если нажать на кнопку “Next step”, то по одному будут появляться ребра остовного дерева, пока всё дерево не будет подсвечено красным. Кнопка “Show full” подсвечивает все остовное дерево сразу. Пример окончательно картинки на экране на рис 3. Также имеется кнопка “Clear graph”, которая очищает область отрисовки графа.



1. Окончание работы программы

# Руководство программиста

## Описание структуры программы

Данная программа разрабатывалась в среде Visual Studio на языке программирования C++ . Решение состоит из трех проектов:

1. KruskalAlgorithm – основная программа
2. KruskalLib – динамическая библиотека
3. TestApp – тестовая программа

Основная программа использует статическую библиотеку KruskalLib, которая содержит все объявления и реализации классов, и компонент для Visual Studio для отрисовки графа на экране.

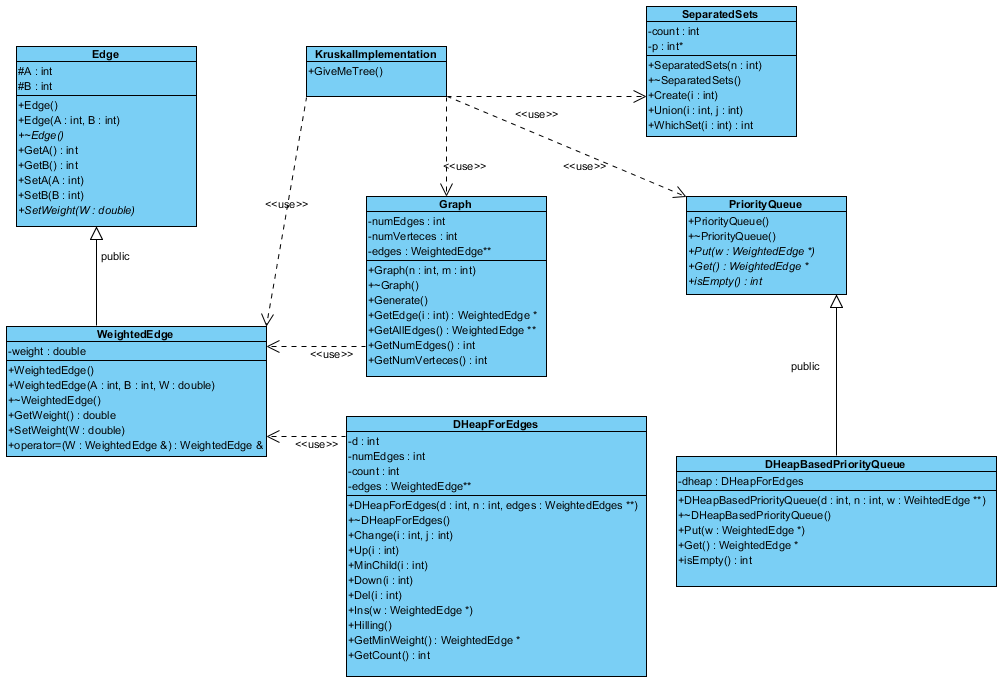
Тестовая программа, используя Google Testing Framework, тестирует методы классов из библиотеки KruskalLib.

Библиотека KruskalLib состоит из 6 модулей:

* Graph – модуль, содержащий класс ребра графа (Edge), взвешенного ребра графа (WeightedEdge) и класс графа (Graph)
* DHeapForEdges – d-куча, элементами которой являются объекты класса Edge
* PriorityQueue – абстрактный класс приоритетной очереди
* DHeapBasedPriorityQueue – класс приоритетной очереди, основанной на d-куче
* SeparatedSets – класс разделенных множеств
* KruskalImplementation – статический класс с реализацией алгоритма Крускала

Основная программа содержит форму, на которой содержатся такие компоненты как gViewer для отрисовки графа, Button, TextBox, RadioButton, Label для взаимодействия с пользователем. Заголовочный файл формы содержит класс формы, в котором описаны все компоненты, а также функции, срабатывающие на событие onClick.

## Описание структур данных

Схема наследования классов в динамической библиотеке KruskalLib (рис 4).

1. Схема наследования классов

### Описание классов

1. Класс Edge

|  |  |
| --- | --- |
| Поле | Описание |
| int A | Начало ребра |
| int B | Конец ребра |

|  |  |
| --- | --- |
| Метод | Описание |
| Edge() | Конструктор по умолчанию |
| Edge(int, int) | Конструктор с параметрами |
| ~Edge() | Виртуальный деструктор |
| int GetA() | Возвращает начало ребра |
| int GetB() | Возвращает конец ребра |
| void SetA(int) | Установить начало ребра |
| void SetB() | Установить конец ребра |

1. Класс WeightedEdge

|  |  |
| --- | --- |
| Поле | Описание |
| int weight | Вес ребра |

|  |  |
| --- | --- |
| Метод | Описание |
| WeightedEdge() | Конструктор по умолчанию |
| WeightedEdge(int, int, double) | Конструктор с параметрами |
| ~WeightedEdge() | Виртуальный деструктор |
| double GetWeight() | Возвращает вес ребра |
| void SetWeight(double) | Устанавливает вес ребра |
| WeightedEdge& operator=(WeightedEdge&) | Перегрузка оператора присваивания |

1. Класс Graph

|  |  |
| --- | --- |
| Поле | Описание |
| int numVerteces | Количество вершин |
| int numEdges | Количество ребер |
| WeightedEdge \*\*egdes | Массив из указателей на ребро |

|  |  |
| --- | --- |
| Метод | Описание |
| Graph(int, int) | Конструктор с параметрами: количество ребер и вершин |
| ~Graph() | Десктруктор |
| void Graph() | Генерирует граф |
| WeightedEdge\* GetEdge(int) | Возвращает ребро по индексу |
| WeightedEdge\*\* GetAllEdges() | Возвращает массив из указателей на ребро |
| int GetNumEdges() | Возвращает количество ребер |
| int GetNumVerteces() | Возвращает количество вершин |

1. Класс DHeapForEdges

|  |  |
| --- | --- |
| Поле | Описание |
| int d | Арность дерева |
| int numEdges | Количество ребер в дереве |
| int count | Реальное количество ребер в дереве |
| WeightedEdge \*\*edges | Массив из указателей на ребро |

|  |  |
| --- | --- |
| Метод | Описание |
| DHeapForEdges(int, int, WeightedEdge\*\*) | Конструктор с параметрами: арность, количество ребер и сами ребра |
| ~DHeapForEdges() | Деструктор |
| void Change(int, int) | Меняет местами два узла |
| void Up(int) | Всплытие узла |
| int MinChild(int) | Возвращает индекс узла с минимальным весом |
| void Down(int) | Погружение узла |
| void Del(int) | Удаление узла |
| void Ins(WeghtedEdge\*) | Вставка узла |
| void Hilling() | Окучивание дерева |
| WeightedEdge\* GetMinWeight() | Возвращает ребро с минимальным весом |
| int GetCount() | Возвращает количество ребер в дереве |

1. Класс PriorityQueue

|  |  |
| --- | --- |
| Метод | Описание |
| PriorityQueue() | Конструктор по умолчанию |
| ~PriorityQueue() | Виртуальный деструктор |
| void Put(WeightedEdge\*)=0 | Вставка в очередь |
| WeigtedEdge\* Get()=0 | Получение из очереди |
| int isEmpty()=0 | Проверка на пустоту |

1. Класс DHeapBasedPriorityQueue

|  |  |
| --- | --- |
| Поле | Описание |
| DHeapForEdges \*dheap | d-куча |

|  |  |
| --- | --- |
| Метод | Описание |
| DHeapBasedPriorityQueue(int,int,WeightedEdge\*\*) | Конструктор с параметрами: арность дерева, количество ребер и ребра |
| ~DHeapBasedPriorityQueue() | Виртуальный деструктор |
| void Put(WeightedEdge\*) | Вставка ребра в очередь |
| WeightedEdge\* Get() | Получение ребра из очереди |
| int isEmpty() | Проверка на пустоту |

1. Класс SeparatedSets

|  |  |
| --- | --- |
| Поле | Метод |
| int count | Количество элементов |
| int \*p | Массив, элементами которого являются множества |

|  |  |
| --- | --- |
| Метод | Описание |
| SeparatedSets(int) | Конструктор с параметром: количество элементов |
| ~SeparatedSets() | Деструктор |
| void Create(int) | Создание синглетона для элемента |
| void Union(int, int) | Объединение двух элементов в одно множество |
| int WhichSet(int) | Возвращает, какому множеству принадлежит элемент |

1. Класс KruskalImplementation

|  |  |
| --- | --- |
| Метод | Описание |
| void GiveMeTree(Graph\*, PriorityQueue\*, Graph\*) | Функция определения остовного дерево в графе. Параметры: исходный граф, приоритетная очередь и остовное дерево |

Если в приведенных выше таблицах в каком-то классе не был описан метод, то, значит, он был описан выше, в классе предка. «=0» после прототипа функции означает, что функция чисто виртуальная и реализация находится в классе-наследнике.

## Описание алгоритмов

Большинство функций, реализованных в процессе выполнения лабораторной работы, являются простыми, алгоритм которых ясен сразу же из их описания. Поэтому в этом разделе будут описаны наиболее сложные алгоритмы.

### Алгоритмы, используемые в d-куче

Всплытие i-го узла:

пока i > 0

1. p = (i-1) / d – определяем предка
2. если ключ предка больше ключа нашего узла, меняем их местами, i = p
3. иначе выходим из цикла

Окучивание дерева:

Проходим по дереву от последнего узла до корня и выполняем погружение.

### Алгоритм Крускала

Пусть дан неориентированный, связанный, взвешенный граф с n вершинами и m ребрами.

1. Создать разделенные множества из n синглетонов
2. Из множества ребер составить приоритетную очередь
3. Достать из приоритетной очереди ребро с минимальным весом
4. Определить разделенное множество, которому принадлежит начало ребра
5. Определить разделенное множество, которому принадлежит конец ребра
6. Если множества из предыдущих пунктов не совпадают, объединить эти множества
7. Ребро занести в остовное дерево
8. Повторить пункты 3-7, пока либо не будет заполнено остовное дерево, либо очередь не станет пуста

# Заключение

Ходе выполнения лабораторной работы была разработана программная система, позволяющая находить минимальное остовное дерево в случайно сгенерерованном неориентированном связном взвешенном графе. Проведенные эксперименты показали, что программа работает корректно, даже при больших размерностях.

# Литература

1. *Страуструп Б*. Программирование: принципы и практика использования C++.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2011.
2. *Никлаус Вирт* Алгоритмыи структуры данных / Пер. с англ. Ткачев Ф.В. – М.:ДМК Пресс, 2010.

# Приложение. Исходные Коды

## Приложение А. Graph.h

#ifndef \_\_GRAPH\_H\_\_

#define \_\_GRAPH\_H\_\_

namespace KruskalLib

{

class Edge

{

protected:

int A, B;

public:

\_\_declspec(dllexport) Edge();

\_\_declspec(dllexport) Edge(int, int);

\_\_declspec(dllexport) virtual ~Edge();

\_\_declspec(dllexport) virtual void print();

\_\_declspec(dllexport) int GetA();

\_\_declspec(dllexport) int GetB();

\_\_declspec(dllexport) void SetA(int);

\_\_declspec(dllexport) void SetB(int);

\_\_declspec(dllexport) virtual void SetWeight(double)=0;

};

class WeightedEdge: public Edge

{

double weight;

public:

\_\_declspec(dllexport) WeightedEdge();

\_\_declspec(dllexport) WeightedEdge(int, int, double);

\_\_declspec(dllexport) ~WeightedEdge();

\_\_declspec(dllexport) void print();

\_\_declspec(dllexport) double GetWeight();

\_\_declspec(dllexport) void SetWeight(double);

\_\_declspec(dllexport) WeightedEdge& operator=(WeightedEdge&);

};

class Graph

{

int numVerteces;

int numEdges;

WeightedEdge \*\*edges;

public:

\_\_declspec(dllexport) Graph(int, int);

\_\_declspec(dllexport) ~Graph();

\_\_declspec(dllexport) void Generate();

\_\_declspec(dllexport) WeightedEdge\* GetEdge(int);

\_\_declspec(dllexport) WeightedEdge\*\* GetAllEdges();

\_\_declspec(dllexport) int GetNumEdges();

\_\_declspec(dllexport) int GetNumVerteces();

};

}

#endif

## Приложение Б. Graph.cpp

#include <time.h>

#include "Graph.h"

#include <iostream>

namespace KruskalLib

{

//

//Edge

//

Edge::Edge()

{

A=-1;

B=-1;

}

Edge::Edge(int \_A, int \_B)

{

A=\_A;

B=\_B;

}

Edge::~Edge()

{

A=-1;

B=-1;

}

void Edge::print()

{

std::cout << A << ' ' << B << std::endl;

}

int Edge::GetA()

{

return A;

}

int Edge::GetB()

{

return B;

}

void Edge::SetA(int \_A)

{

A=\_A;

}

void Edge::SetB(int \_B)

{

B=\_B;

}

//

//WeightedEdge

//

WeightedEdge::WeightedEdge()

{

A=-1;

B=-1;

weight=-1;

}

WeightedEdge::WeightedEdge(int \_A, int \_B, double \_weight)

{

A=\_A;

B=\_B;

weight=\_weight;

}

WeightedEdge::~WeightedEdge()

{

weight=-1;

}

void WeightedEdge::print()

{

std::cout << A << ' ' << B << ' ' << weight << std::endl;

}

double WeightedEdge::GetWeight()

{

return weight;

}

void WeightedEdge::SetWeight(double \_weight)

{

weight=\_weight;

}

//

//Graph

//

Graph::Graph(int \_numVerteces, int \_numEdges)

{

numVerteces=\_numVerteces;

numEdges=\_numEdges;

edges=new WeightedEdge\*[numEdges];

for (int i=0; i<numEdges; i++)

edges[i]=new WeightedEdge;

}

Graph::~Graph()

{

for (int i=0; i<numEdges; i++)

delete edges[i];

delete[] edges;

numVerteces=0;

numEdges=0;

}

int Isset(int \*arr, int n, int k)

{

for (int i=0; i<n; i++)

{

if (arr[i]==k)

{

return 0;

}

}

return 1;

}

int IssetEdge(WeightedEdge \*\*edges, int numEdges, int A, int B)

{

for (int i=0; i<numEdges; i++)

{

if (edges[i]->GetA() == A && edges[i]->GetB() == B || edges[i]->GetA() == B && edges[i]->GetB() == A)

{

return 0;

}

}

return 1;

}

void Graph::Generate()

{

srand(time(0));

double rand3;

int \*tmp=new int[numVerteces];

int rand2;

int rand1=rand()%numVerteces+1;

tmp[0]=rand1;

edges[0]->SetA(rand1);

for (int i=0; i<numVerteces-1; i++)

{

do

{

rand2=rand()%numVerteces+1;

}

while (Isset(tmp,numVerteces,rand2) == 0);

edges[i]->SetB(rand2);

if (i!=numVerteces-2) edges[i+1]->SetA(rand2);

tmp[i+1]=rand2;

}

delete tmp;

for (int i=numVerteces-1; i<numEdges; i++)

{

do

{

do

{

rand1=rand() % (numVerteces)+1;

rand2=rand() % (numVerteces)+1;

}

while (rand1==rand2);

}

while (IssetEdge(edges, numEdges, rand1, rand2) == 0);

(edges[i])->SetA(rand1);

(edges[i])->SetB(rand2);

}

for (int i=0; i<numEdges; i++)

{

rand3=double(rand()%101)+double(rand())/RAND\_MAX;

(edges[i])->SetWeight(rand3);

}

}

WeightedEdge\* Graph::GetEdge(int i)

{

return edges[i];

}

WeightedEdge& WeightedEdge::operator=(WeightedEdge &SecondOperand)

{

if (this == &SecondOperand) return \*this;

this->A=SecondOperand.A;

this->B=SecondOperand.B;

this->weight=SecondOperand.weight;

return \*this;

}

WeightedEdge\*\* Graph::GetAllEdges()

{

return edges;

}

int Graph::GetNumEdges()

{

return numEdges;

}

int Graph::GetNumVerteces()

{

return numVerteces;

}

}

## Приложение В. DHeapForEdges.h

#ifndef \_\_DHEAP\_H\_\_

#define \_\_DHEAP\_H\_\_

#include "Graph.h"

#include <iostream>

namespace KruskalLib

{

class DHeapForEdges

{

int d;

int numEdges;

int count;

WeightedEdge \*\*edges;

public:

\_\_declspec(dllexport) DHeapForEdges(int,int, WeightedEdge\*\*);

\_\_declspec(dllexport) ~DHeapForEdges();

\_\_declspec(dllexport) void Change(int, int);

\_\_declspec(dllexport) void Up(int);

\_\_declspec(dllexport) int MinChild(int);

\_\_declspec(dllexport) void Down(int);

\_\_declspec(dllexport) void Del(int);

\_\_declspec(dllexport) void Ins(WeightedEdge\*);

\_\_declspec(dllexport) void Hilling();

\_\_declspec(dllexport) WeightedEdge\* GetMinWeight();

\_\_declspec(dllexport) void print();

\_\_declspec(dllexport) int GetCount();

};

}

#endif

## Приложение Г. DHeapForEdges.cpp

#include "Graph.h"

#include "DHeapForEdges.h"

#include <iostream>

#define MIN(A,B) ((A) < (B) ? (A) : (B))

namespace KruskalLib

{

//

//DHeapForEdges

//

DHeapForEdges::DHeapForEdges(int \_d, int \_numEdges, WeightedEdge \*\*\_edges)

{

d=\_d;

numEdges=\_numEdges;

count=\_numEdges;

edges=new WeightedEdge\*[numEdges];

for (int i=0; i<numEdges; i++)

{

edges[i]=new WeightedEdge;

\*(edges[i])=\*(\_edges[i]);

}

this->Hilling();

}

DHeapForEdges::~DHeapForEdges()

{

if (edges == 0 ) return;

for (int i=0; i<numEdges; i++)

delete edges[i];

delete[] edges;

d=-1;

numEdges=-1;

}

void DHeapForEdges::Change(int i, int j)

{

WeightedEdge tmp;

tmp=\*(edges[i]);

\*(edges[i])=\*(edges[j]);

\*(edges[j])=tmp;

}

void DHeapForEdges::Up(int i)

{

int p;

while (i>0)

{

p=(i-1)/d;

if (edges[p]->GetWeight() > edges[i]->GetWeight())

{

Change(i,p);

i=p;

}

}

}

int DHeapForEdges::MinChild(int i)

{

if (i\*d+1 >= count) return -1;

int left=i\*d+1;

int right=MIN(i\*d+d, count-1);

WeightedEdge min=\*(edges[left]);

int minId=left;

for(int j=left+1; j<=right; j++)

{

if (edges[j]->GetWeight() < min.GetWeight())

{

min=\*(edges[j]);

minId=j;

}

}

return minId;

}

void DHeapForEdges::Down(int i)

{

int c=MinChild(i);

while (c!=-1 && edges[i]->GetWeight() > edges[c]->GetWeight())

{

Change(i,c);

i=c;

c=MinChild(i);

}

}

void DHeapForEdges::Hilling()

{

for(int i=count-1; i>=0; i--)

Down(i);

}

void DHeapForEdges::Ins(WeightedEdge \*tmp)

{

if (count == numEdges) return;

count++;

\*(edges[count-1])=\*tmp;

Up(count);

}

void DHeapForEdges::Del(int i)

{

Change(i, count-1);

count--;

int p=(i-1)/d;

if (edges[p]->GetWeight() > edges[i]->GetWeight()) Up(i);

else Down(i);

}

WeightedEdge\* DHeapForEdges::GetMinWeight()

{

Change(0, count-1);

count--;

Down(0);

return edges[count];

}

void DHeapForEdges::print()

{

for (int i=0; i<count; i++)

{

std::cout << edges[i]->GetWeight() <<' ';

if (i%d == 0) std::cout<<std::endl;

}

}

int DHeapForEdges::GetCount()

{

return count;

}

}

## Приложение Д. PriorityQueue.h

#ifndef \_\_PRIORITYQUEUE\_H\_\_

#define \_\_PRIORITYQUEUE\_H\_\_

#include "Graph.h"

namespace KruskalLib

{

class PriorityQueue

{

public:

\_\_declspec(dllexport) PriorityQueue() {};

\_\_declspec(dllexport) virtual ~PriorityQueue() {};

\_\_declspec(dllexport) virtual void Put(WeightedEdge\*)=0;

\_\_declspec(dllexport) virtual WeightedEdge\* Get()=0;

\_\_declspec(dllexport) virtual int isEmpty()=0;

};

}

#endif

## Приложение Е. DHeapBasedPriorityQueue.h

#ifndef \_\_DHEAPBASEDPRIORITYQUEUE\_H\_\_

#define \_\_DHEAPBASEDPRIORITYQUEUE\_H\_\_

#include "Graph.h"

#include "DHeapForEdges.h"

#include "PriorityQueue.h"

namespace KruskalLib

{

class DHeapBasedPriorityQueue: public PriorityQueue

{

DHeapForEdges \*dheap;

public:

\_\_declspec(dllexport) DHeapBasedPriorityQueue(int,int,WeightedEdge\*\*);

\_\_declspec(dllexport) virtual ~DHeapBasedPriorityQueue();

\_\_declspec(dllexport) virtual void Put(WeightedEdge\*);

\_\_declspec(dllexport) virtual WeightedEdge\* Get();

\_\_declspec(dllexport) int isEmpty();

};

}

#endif

## Приложение Ж. DHeapBasedPriorityQueue.cpp

#include "PriorityQueue.h"

#include "DHeapBasedPriorityQueue.h"

#include "DHeapForEdges.h"

namespace KruskalLib

{

DHeapBasedPriorityQueue::DHeapBasedPriorityQueue(int \_d, int \_numEdges, WeightedEdge \*\*\_edges)

{

dheap=new DHeapForEdges(\_d,\_numEdges,\_edges);

dheap->Hilling();

}

DHeapBasedPriorityQueue::~DHeapBasedPriorityQueue()

{

delete dheap;

}

void DHeapBasedPriorityQueue::Put(WeightedEdge \*tmp)

{

dheap->Ins(tmp);

}

WeightedEdge\* DHeapBasedPriorityQueue::Get()

{

return dheap->GetMinWeight();

}

int DHeapBasedPriorityQueue::isEmpty()

{

return dheap->GetCount();

}

}

## Приложение З. SeparatedSets.h

#ifndef \_\_SEPARATEDSETS\_H\_\_

#define \_\_SEPARATEDSETS\_H\_\_

namespace KruskalLib

{

class SeparatedSets

{

int count;

int \*p;

public:

\_\_declspec(dllexport) SeparatedSets(int);

\_\_declspec(dllexport) ~SeparatedSets();

\_\_declspec(dllexport) void Create(int);

\_\_declspec(dllexport) void Union(int, int);

\_\_declspec(dllexport) int WhichSet(int);

};

}

#endif

## Приложение И. SeparatedSets.cpp

#include "SeparatedSets.h"

#include <iostream>

namespace KruskalLib

{

SeparatedSets::SeparatedSets(int \_count)

{

count=\_count;

p=new int[count];

for (int i=0; i<count; i++)

p[i]=0;

}

SeparatedSets::~SeparatedSets()

{

count=0;

delete[] p;

p=0;

}

void SeparatedSets::Create(int who)

{

if (p[who-1] !=0) return;

p[who-1]=who;

}

int SeparatedSets::WhichSet(int who)

{

return p[who-1];

}

void SeparatedSets::Union(int A, int B)

{

for (int i=0; i<count; i++)

if (p[i]==B) p[i]=A;

}

}

## Приложение К. KruskalImplementation.h

#ifndef \_\_KRUSKALIMPLEMENTATION\_H\_\_

#define \_\_KRUSKALIMPLEMENTATION\_H\_\_

#include "Graph.h"

#include "PriorityQueue.h"

namespace KruskalLib

{

class KruskalImplemetation

{

public:

\_\_declspec(dllexport) static void GiveMeTree(Graph\*, PriorityQueue\*, Graph\*);

};

}

#endif

## Приложение Л. KruskalImplementation.cpp

#include "KruskalImplementation.h"

#include "SeparatedSets.h"

#include "DHeapBasedPriorityQueue.h"

#include "Graph.h"

#include "DHeapForEdges.h"

#include "PriorityQueue.h"

#include <iostream>

using namespace std;

namespace KruskalLib

{

void KruskalImplemetation::GiveMeTree(Graph\* g, PriorityQueue \*q, Graph \*T)

{

int numEdges=g->GetNumEdges();

int numVerteces=g->GetNumVerteces();

SeparatedSets SS(numVerteces);

for (int i=1; i<=numVerteces; i++)

SS.Create(i);

WeightedEdge \*e;

int set1=0, set2=0;

int k=0;

int A,B;

while (q->isEmpty() != 0 && k!=numVerteces-1)

{

e=q->Get();

A=e->GetA();

B=e->GetB();

set1=SS.WhichSet(A);

set2=SS.WhichSet(B);

if (set1 != set2)

{

SS.Union(set1,set2);

\*(T->GetAllEdges())[k]=\*e;

k++;

}

}}}