Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение  
высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт Информационных технологий, математики и механики

Отчёт по учебной практике

Реализация приоритетной очереди на основе d-кучи и ее применение для нахождения кратчайших путей в графе с помощью алгоритма Дейкстры.

Выполнил:

студент гр. 381606-3

Волков А.А.

Проверил:

к.т.н., доцент МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г.

**Содержание**

Введение 3

Постановка задачи 4

Руководство пользователя 5

Руководство программиста 6

Описание структуры программы 6

Описание структур данных 6

Описание алгоритмов 6

Заключение 7

**Введение**

Лабораторная работа направлена на практическое освоение:

1. Алгоритма Дейкстры и разработке статических библиотек, реализующие следующие структуры данных:

* d-куча;
* приоритетная очередь, основанная на d-куче;

2.Написать приложение для демонстрации работы приоритетной очереди, основанной на d-куче (алгоритм Дейкстры):

* входные данные - связный неориентированный взвешенный граф без петель со стартовой вершиной;
* выходные данные - список расстояний до каждой вершины графа.

Рассматриваемые в данной лабораторной работе алгоритмы являются начальным введением в область машинных вычислений.

**Постановка задачи**

* Разработать статические библиотеки, реализующие следующие структуры данных:
* Д-куча;
* Приоритетная очередь.
* Написать приложение для демонстрации работы приоритетной очереди (алгоритм Дейкстры):
* Входные данные - связный неориентированный взвешенный граф без петель со стартовой вершиной;
* Выходные данные - список расстояний до каждой вершины графа;

**Руководство пользователя**

**Использование реализации алгоритма Дейкстры**

Запуск приложения и ввод данных

Программа предназначена для поиска кратчайших путей во взвешенном неориентированном графе от некоторой вершины, называемой текущей, до всех остальных вершин графа. Для запуска приложения нужно открыть исполняемый файл main.exe. Программа попросит ввести количество вершин графа и ребра с весом. Также будет запрошен номер вершины, являющейся стартовой. Результатом будет вывод вектора, содержащего номера вершин, что предшествуют вершинам, являющимся индексами вектора, для построения дерева, позволяющего восстановить обход и вывод списка расстояний от стартовой вершины до каждой из следующих вершин в порядке возрастания номера.

Шаг 1 Ввод количества вершин графа

Шаг 2 Ввод ребер и их весов

Шаг 3 Выбор стартовой вершины

Шаг 4 Вывод результата и завершение работы программы

**Руководство программиста**

**Описание структуры программы**

D-куча - завершенное d-арное дерево, содержащее набор однотипных элементов, со следующими свойствами:

* каждый узел, не являющийся листом, за исключением, быть может, одного имеет ровно d потомков. Один узел, являющийся исключением, может иметь от 1 до d-1 потомка;
* если h - глубина дерева, то для любого i = 1, ..., k-1 такое дерево имеет ровно d^i узлов глубины i;
* количество узлов глубины k в дереве глубины k может варьироваться от 1 до d^k;
* каждый узел имеет вес. Иначе говоря, каждому узлу дерева присвоен ключ такого типа данных, на котором определен порядок сравнения;
* ключ элемента, приписанного узлу i, не превосходит ключа любого из своих потомков.

Приоритетная очередь — это динамическая структура данных, содержащая элементы, каждый из которых имеет определенный приоритет. Элемент с более высоким приоритетом находится перед элементом с более низким приоритетом. Если у элементов одинаковые приоритеты, они располагаются в зависимости от своей позиции в очереди.

Основные операции:

* Вставка элемента: ВСТАВИТЬ(x).
* Удаление старшего элемента: УДАЛИТЬ().
* Получение старшего элемента: ПОЛУЧИТЬ().
* Проверка на пустоту: ПУСТОТА().
* Проверка на полноту: ПОЛНОТА().

***Описание структур данных***

В лабораторной работе Д-куча представлена классом DAryHeap, содержащим нижеизложенные public-методы:

#include <iostream>

#include <algorithm>

#include "Graph.h"

#define maxSize 1000

typedef int dataType;

class Data {

public:

float priorities;

};

class DAryHeap {

protected:

Data \*\*keys;

int d;

int lastIdx;

public:

DAryHeap(int d);

DAryHeap(const DAryHeap &heap);

~DAryHeap();

void add(Data \*&key);

void addSet(Data \*\*key, int num);

void transposition(int i, int j);

void surfacing(int i);

void immersion(int i);

void spudding();

int isFull();

int isEmpty();

private:

int minChild(int i);

};

* DAryHeap (int) - конструктор. Принимает на вход стартовый размер вектора памяти для хранения d-кучи
* DAryHeap (const DAryHeap &heap )- конструктор копирования;
* DAryHeap (void) - деструктор;
* getSizeTree(void)const - возвращает количество узлов, содержащихся в д-куче;
* void add(Data \*&key);- возвращает ключ узла, номер которого передан в качестве параметра;
* void addSet(Data \*\*key, int num);- возвращает размер свободной памяти в выделенном ранее векторе памяти;
* getD(void)const - возвращает арность кучи;
* void immersion(int i)- выполняет всплытие узла с индексом, указанным в качестве аргумента;
* insert(const KeyType&, mem\_rc flag = ALLOW\_MEMORY\_REALLOCATION\_WCV, int size = 0)
* int isFull();- проверка на полноту
* int isEmpty();-проверка на пустоту
* MinChild(int) - минимальный "ребенок".
* void transposition(int i, int j)-транспонирование 2-х элементов.

Приоритетная очередь - динамическая структура данных, хранящая набор однотипных элементов с определенным отношением порядка на этих данных. Сам порядок определяет отношение преобладания одного элемента над другим. В данной лабораторной работе старшинство приоритета отдается меньшему по порядку значению.

* Для хранения и реализации приоритетной очереди удобно использовать д-кучу.
* Приоритетная очередь содержит нижеизложенные public-методы:

#pragma once

#include "d-heap.h"

enum QueueID {

HEAP = 0,

};

class PriorityQueue {

public:

PriorityQueue() {};

virtual void push(Data \*&key) = 0;

virtual Data\* pop() = 0;

virtual int isFull() = 0;

virtual int isEmpty() = 0;

};

class PriorityQueueHeap : public PriorityQueue {

protected:

DAryHeap \*heap;

public:

PriorityQueueHeap(int d = 4);

PriorityQueueHeap(const PriorityQueueHeap &queue);

PriorityQueueHeap(Data \*\*keys, int num, int d = 4);

~PriorityQueueHeap();

virtual void push(Data \*&key);

virtual Data\* pop();

virtual int isFull();

virtual int isEmpty();

};

class QueueFactory {

public:

static PriorityQueue\* createQueue(QueueID qid)

{

PriorityQueue \*queue;

queue = new PriorityQueueHeap();

return queue;

}

};

* PRIORITY\_QUEUE(int) - конструктор; в качестве аргумента принимает арность дерева хранения очереди (по умелчанию эзначение равно трем);
* PRIORITY\_QUEUE(const PRIORITY\_QUEUE<KeyType>&) - конструктор копирования;
* ~PRIORITY\_QUEUE(void) - деструктор;
* getHeap(void) - водвращает д-кучу, хранящую элементы очереди (требуется для проведения тестирования);
* pop(void) - удаляет первый в очереди элемент;
* push(const KeyType&) - добавляет элемент в очередь;
* back(void) const - возвращает первый элемент в очереди (без удаления);
* int isFull();- проверка на полноту
* int isEmpty();-проверка на пустоту

***Описание алгоритмов***

Алгоритм Дейкстры

Вход

Задается взвешенный (каждому ребру присвоено значение веса) связный (из любой вершины можно добраться до всех остальных вершин) граф. Задается стартовая вершина.

Выход

При окончании работы алгоритма выходом является пара массивов, содержащих:

* список предшествующих вершин для текущих, являющихся индексами, для построения дерева обхода;
* список расстояний от стартовой вершины до всех остальных.

Алгоритм

* Задается взвешенный граф из n вершин.
* Задается стартовая вершина CUR.
* Создается вектор для отметки посещения вершины V (все элементы зануляются, V[CUR] = CUR).
* Создается результирующий вектор расстояний DIST (все элементы устремлены в бесконечность, DIST[CUR] = 0).
* Создается приоритетная очередь, в которую кладется расстояние от текущей вершины до неё же.
* Пока очередь не пуста:
* Вынимается минимальный элемент.
* Если метка вынутого элемента больше метки, хранящейся в массиве DIST, переход к следующему шагу.
* По всем ребрам от текущей вершины: Если результирующее расстояние от смежной вершины больше, чем результирующее расстояние до вынутой вершины в сумме с меткой ребра, выполняем:
* В вектор V по номеру смежной вершины кладется значение вершины, вынутой на этапе .
* В вектор DIST по номету смежной вершины пишется новое значение расстояния, равное сумме результирующего расстояния до вынутой вершины и меткой ребра,
* Обработанное ребро кладется в очередь.

Результатом работы алгоритма становится вектор расстояний до каждой вершины графа (DIST) и вектор обхода вершин графа для получения указанных значений (V).

#include "Deijkstra.h"

DataFloat::DataFloat(int v, float dist)

{

this->v = v;

priorities = dist;

}

void Dijkstra::dijkstra(Graph \*&graph, int s, float \*&distance, int \*&up)

{

int n = graph->getVerticesNum();

int m = graph->getRealSize();

if ((s < 0) || (s >= n))

throw "Dijkstra: Invalid start vertex!";

Data\*\* dist = new Data\*[n];

up = new int[n];

PriorityQueue \*queue = QueueFactory::createQueue(static\_cast<QueueID>(0));

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

up[i] = i;

dist[i] = new DataFloat(i, FLT\_MAX);

if (i == s)

dist[s]->priorities = 0;

queue->push(dist[i]);

}

Edge\*\* edges = graph->getEdgeSet();

while (!queue->isEmpty())

{

int vConsidered = ((DataFloat\*)queue->pop())->v;

float delta;

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

{

int vIncident = -1;

if (edges[i]->K == vConsidered)

vIncident = edges[i]->N;

if (edges[i]->N == vConsidered)

vIncident = edges[i]->K;

if (vIncident == -1) continue;

float way = dist[vConsidered]->priorities + graph->getWeight(vConsidered, vIncident);

delta = dist[vIncident]->priorities - way;

if (delta > 0)

{

dist[vIncident]->priorities = way; //поскольку в очереди лежат указатели на объекты dist[i],

up[vIncident] = vConsidered; //то значения в очереди автоматически изменятся.

}

}

}

distance = new float[n];

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

distance[i] = dist[i]->priorities;

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

delete dist[i];

delete[]dist;

delete queue;

}

**Заключение**

В ходе лабораторной работы были реализованы структуры данных "д-куча", "приоритетная очередь" с использованием шаблонных классов. Написаны консольное приложение: алгоритм Дейкстры для поиска кратчайших путей от некоторой стартовой вершины связного неориентированного взвешенного графа без петель до всех прочих.