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

Алгоритм Дейкстры на D-куче

Выполнил:
студент гр. 381606-3

Каганов Д.А.

Проверил:
к.т.н., ст. преп. каф. МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород
2017 г.

Содержание

Введение

Алгоритм Дейкстры находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек. С этой целью в лабораторной работе изучаются различные варианты структуры хранения стеков и разрабатываются методы и программы решения ряда задач с использованием стеков. В качестве области приложений выбрана тема вычисления арифметических выражений, возникающей при трансляции программ на языке программирования высокого уровня в исполняемые программы. При вычислении произвольных арифметических выражений возникают две основные задачи: проверка корректности введённого выражения и выполнение операций в порядке, определяемом их приоритетами и расстановкой скобок. Существует алгоритм, позволяющий реализовать вычисление произвольного арифметического выражения за один просмотр без хранения промежуточных результатов. Для реализации данного алгоритма выражение должно быть представлено в постфиксной форме. Рассматриваемые в данной лабораторной работе алгоритмы являются начальным введением в область машинных вычислений.

Постановка задачи

В рамках лабораторной работы ставится задача реализации программ, обеспечивающих поддержку стеков, и разработки программных средств, производящих обработку арифметических выражений, включая проверку правильности записи выражения, перевод в постфиксную форму и вычисление результата. В начальной – самой простой постановке – можно предполагать, что проверка записи выражения состоит в контроле правильности расстановки скобок, перевод в постфиксную форму производится только для корректных выражений, а вычисление – для корректных выражений, содержащих только числовые операнды и допустимые знаки операций.

Руководство пользователя

1. Запустите программу.
2. В появившемся окне введите (команда должна вывести значение выражения)

Руководство программиста

Описание структуры программы

1) ../include
    a) Dejkstra.h
    Содержит описание класса Dejkstra
    b) DHeap.h
    Содержит описание класса DHeap
    c) Graph.h
    Содержит описание класса Graph
    d) PriorityQueue.h
    Содержит описание класса PriorityQueue
2) ../sample
    a) GenerateGraph.cpp
    Содержит основной код программы
3) ../src
    a) Dejkstra.h
    Содержит реализацию класса Dejkstra
    b) DHeap.h
    Содержит реализацию класса DHeap
    c) Graph.h
    Содержит реализацию класса Graph
    d) PriorityQueue.h
    Содержит реализацию класса PriorityQueue
4) ../build
    Содержит директорию с решением и проектом для MS Visual Studio

Описание структур данных

Шаблонный класс Dejkstra

```
class DataFloat : public Data {
public:
	DataFloat(int s, float dist);
	int s;
};

class Dejkstra {
public:
	static void dejkstra(Graph *&graph, int s, float *&distance, int *&up);
};
```

Описание методов:

1. dejkstra - алгоритм Дэйкстры

Шаблонный класс DHeap

```
typedef int dataType;

class Data {
public:
	float priorities;
};

class DHeap {
protected:
	Data **keys;
	int d;
	int idx;
public:
	DHeap(int d);
	DHeap(const DHeap &heap);
	~DHeap();

	void Add(Data *&key);
	void AddSet(Data **key, int num);
	Data* Delete();
	Data* Remove(int i);

	void Transpose(int i, int j);
	void Surfacing(int i);
	void Sinking(int i);
	void Hilling();
	int IsFull();
	int IsEmpty();

private:
	int MinChild(int i);
};
```

Описание методов:

1. Transpose - транспонирование
2. Surfacing - всплытие узла
3. Sinking - погружение
4. Hilling - окучивание
5. MinChild - погружение узла i

Шаблонный класс Graph

```
class Edge {
public:
	int Ne;
```

```
        int Ke;
        float W;

        Edge(int Ne, int Ke, float W);
};

class Graph
{
private:
        int n;
        int m;
        int m_cur;
        Edge** edges;
        int* vertices;
public:
        Graph(int n);
        Graph(int n, int m);
        ~Graph();

        void Generate(float minRange, float maxRange);
        void AddEdge(int Ne, int Ke, float weight);
        void DelEdge(int Ne, int Ke);
        int GetVerticesNum();
        int GetEdgeSize();
        int GetRealSize();
        Edge** GetEdgeSet();
        Edge*  GetEdge(int i);
        float GetWeight(int Ne, int Ke);
        void PrintList();
private:
        void GenerateVertices(int &Ne, int &Ke);
        float GenerateWeight(float minRange, float maxRange);
        void Clean();
        int FindEdge(int Ne, int Ke);
};
```

Описание методов:

1. Generate -
2. AddEdge -
3. DelEdge -
4. GetVerticesNum -
5. GetEdgeSize -
6. GetRealSize -
7. GetEdgeSet -
8. GetEdge -
9. GetWeight -
10. PrintList
11. GenerateVertices -
12. GenerateWeight -
13. Clean -
14. FindEdge -

Описание алгоритмов

Дан взвешенный ориентированный граф G(V,E) без дуг отрицательного веса. Найти кратчайшие пути от некоторой вершины a графа G до всех остальных вершин этого графа.

- n — множество вершин графа
- m — множество рёбер графа
- minRange — минимальный вес ребра
- maxRange — максимальный вес ребра
- s — вершина, расстояния от которой ищутся

В простейшей реализации для хранения чисел $d[i]$ можно использовать массив чисел, а для хранения принадлежности элемента множеству U — массив булевых переменных.

В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бóльшим максимального возможного пути в графе). Массив флагов заполняется нулями. Затем запускается основной цикл.

На каждом шаге цикла мы ищем вершину v с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины u. Если в них (в u) расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его. Цикл завершается, когда флаги всех вершин становятся равны 1, либо когда у всех вершин с флагом 0 d[i]=∞. Последний случай возможен тогда и только тогда, когда граф G несвязный.

```
DataFloat::DataFloat(int s, float dist)
{
	this->s = s;
	priorities = dist;
}

void Dejkstra::dejkstra(Graph *&graph, int s, float *&distance, int *&up)
{
	int n = graph->GetVerticesNum();
	int m = graph->GetRealSize();
	if ((s < 0) || (s >= n))
		throw "Dejkstra: 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())->s;
		float delta;

		for (int i = 0; i < m; i++)
		{
			int vIncident = -1;
			if (edges[i]->Ke == vConsidered)
				vIncident = edges[i]->Ne;
			if (edges[i]->Ne == vConsidered)
				vIncident = edges[i]->Ke;
			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;
```

```
                        up[vIncident] = vConsidered;
                        queue->Refresh();
                    }
                }
            }

            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 рамках лабораторной работы мы реализовали программу, обеспечивающую поддержку стеков, и разработали программные средства, производящие обработку