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 от выделенной вершины до всех остальных вершин графа с помощью Алгоритма Дейкстры.**

Выполнил:

студент гр. 381606-3

Басов А.С.

Проверил:

к.т.н., ст. преп. каф. МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г.

Содержание

Введение. 3

Постановка задачи. 4

Руководство пользователя. 5

Запуск приложения и ввод данных. 5

* Введите количество вершин и ребер и нажмите клавишу "Enter". 5

Руководство программиста. 6

Описание структуры проекта: 6

Описание алгоритмов. 7

Описание структур данных. 8

Заключение 10

В ходе лабораторной работы была разработана программа, удовлетворяющая поставленным задачам. Реализованы классы D-арная куча и приоритетная очередь. Реализован Алгоритм Дейкстры. 10

Литература 11

Приложения 12

Введение.

Алгоритм Дейкстры находит кратчайшие пути от одной из вершин графа до всех остальных вершин графа. Алгоритм работает только для графов, которые не имеют ​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.

**Цель данной лабораторной работы** — изучение структур данных типа "d-куча" и разделенные множества.

В качестве примера реализации структуры данных(d-куча), разработать алгоритм подсчета кратчайших путей от выделенной вершины до всех остальных вершин графа с помощью Алгоритма Дейкстры. Создать консольное приложение, демонстрирующее работу алгоритма, где входные данные — количество вершин, ребер и вес ребра, а результат — кратчайших путей от выделенной вершины до всех остальных вершин графа.

Руководство пользователя.

Запуск приложения и ввод данных.

Данная программа предназначена для нахождения кратчайших путей от выделенной вершины до всех остальных вершин графа. Для запуска программы, необходимо запустить файл **sample\_dijkstra.exe** и далее вводить требуемые значения.

***Пример:***

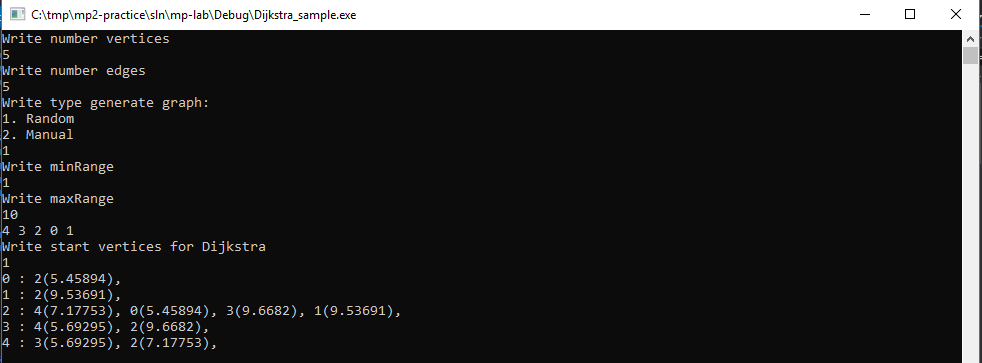
* Введите количество вершин и ребер и нажмите клавишу "Enter".



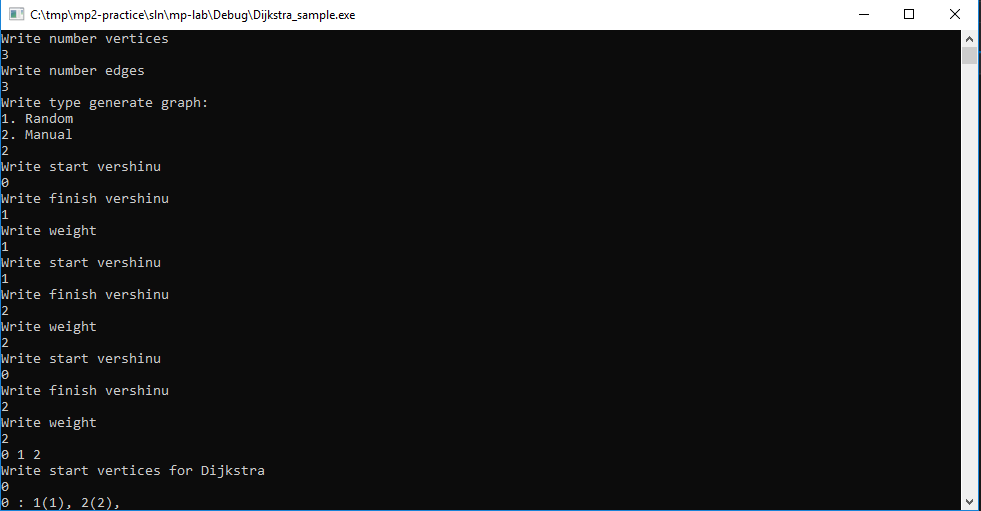
* Выберите способ генерации, нажав клавишу **"0"** или **“1”**, котораясоответствует вашему выбору, после нажмите клавишу **"Enter".**



* При выборе “**рандомной”** генерации, вам потребуется ввести **минимальное** и **максимальное** значение **веса** ребра и **начальную вершину** для **Алгоритма Дейкстры**, после чего программа выдаст конечный результат.



* При выборе **“ручной”** генерации, вам потребуется ввести **начальные** и **конечные** **вершины**, их **вес** и **начальную вершину** для **Алгоритма Дейкстры**, после чего программа выдаст конечный результат.



Руководство программиста.

**Используемые инструменты:**

В ходе лабораторной работы использовались следующие инструменты:

* Система **GitHub**.
* Среда разработки **Microsoft Visual Studio 2017**.

**Общая структура проекта:**

*Структура проекта:*

* **include** — директория для размещения **заголовочных** файлов.
* **samples** — директория для размещения **тестового** приложения.
* **sln** — директория с файлами решений и проектов для **Visual Studio 2017**.
* **src** — директория для размещения **исходных кодов** (cpp-файлы).
* **doc** — отчет о выполненной лабораторной работе.

*Cлужебные файлы:*

* **.gitignore** — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.

Описание структуры проекта:

*Программа состоит из 5 проектов:*

* **Priority-Queue** — статическая библиотека, содержащая реализацию приоритетной очереди на основе **d-арного** дерева.
* **Graph** — статическая библиотека, которая содержит объявление и реализацию классов **Graph** и **Edges**.
* **Dijkstra** — статическая библиотека, содержащая объявление и реализацию классов **Dijkstra** и **DataFloat**.
* **D-Heap** — статическая библиотека, содержащая объявление и реализацию классов **DHeap** и **Data**.
* **Sample-Dijkstra** — консольное приложение, демонстрирующее работу алгоритма Дейкстры.

Описание алгоритмов.

Алгоритм Дейкстры.

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

**Алгоритм состоит из нескольких действий:**

1. Создание множества расстояний **dist[i]** со значениями равным бесконечности.
2. Создание массива с нулевыми значениями **up[i]** содержащего по окончании работы алгоритма кратчайший путь.
3. Создание приоритетной очереди **queue**.
4. Пока приоритетная очередь не пуста **(!queue ->IsEmpty)** повторяются действия:
   1. Изымается вершина с минимальным приоритетом.
   2. **(lastidx=((Data\*)queue->idx(0))->tek\_vershina;)**
   3. В цикле от **0** до **rebra**(количества ребер) ищем ребра совпадающие началом(*концом*) с **lastidx**.
   4. Если такое ребро нашлось, то мы проверяем, является ли новый путь короче, чем ранее имевшийся до этой вершины.
      1. Если да, то мы добавляем расстояние до этой вершины в множество **dist**.
      2. Добавляем конец(*начало*) ребра в массив кратчайших путей **up.**
      3. Обновляем приоритетную очередь. **(queue->update();)**
      4. Далее мы возвращаемся в пункт **4.**

Описание структур данных.

**Структура данных "D-арная куча".**

Завершенное **d-арное дерево** – корневое дерево, у каждого узла которого, за исключением, быть может, одного узла, ровно d потомков. При этом возможна следующая нумерация узлов:

1. Корень имеет номер **0**.
2. Если какой-то узел имеет номер i, то он имеет потомков с номерами ***id* + 1,*id* + 2, ... , *id* + *d*.**

В таком случае узел с номером i имеет предка (i-1)/d.

Если каждому узлу завершенного D-арного дерева приписать элемент приоритетной очереди так, чтобы вес каждого узла не превосходил весов элементов, находящихся в его потомках, то такое дерево называется D-кучей. **Базовые операции над d-кучами (транспонирование, всплытие, погружение) и их трудоемкость.** Сохранить d-кучу можно в обычном массиве (сначала корень, потом его потомков, затем потомков первого потомка корня и т.д), но с точки зрения программной реализации это не совсем удобно, т.к. при выполнении некоторых операций с приоритетными очередями может потребоваться переупаковка элементов.

Сначала рассмотрим некоторые базовые операции над узлами d-кучи:

* **Транспортирование узлов с номерами i и j.**
* **Всплытие узла с номером.**
* **Погружение узла с номером.**
* **Операция изъятия элемента с минимальным ключом (весом).**
* **Операция окучивания.**

**Описание Классов.**

* **Поля:**
* **keys**  - массив для хранения ключей.
* **lastIdx (int)** - количество элементов в дереве.
* **d (int)** - арность кучи.
* **Методы:**
* **DHeap(int d)** - конструктор. Принимает значение арности кучи.
* **~DHeap()** - деструктор.
* **void Add (Data \*&key)** - метод добавления новых узлов.
* **void AddSet (Data \*\*keys, int num)** - метод добавления группы узлов.
* **Data\* Erase()** - метод удаляющий последний узел.
* **Data\* Erase(int i)** - метод удаляющий i-ый узел.
* **void Transposition (int i, int j)** - метод меняющий два указанных узла местами.
* **void Surfacing(int i)** - метод всплытия i-ого узла.
* **void Immersion(int i)** - метод погружения i-ого узла.
* **void Hilling()** - метод окучивания.
* **int IsFull()** - метод сигнализирующий о полноте кучи.
* **int IsEmpty()** - метод сигнализирующий о пустоте кучи.
* **int MinChild(int i)** - метод нахождения минимального потомка i-ого узла.

**Программная реализация "приоритетная очередь".**

**Приоритетная очередь** — это абстрактная структура данных на подобии стека или очереди, где у каждого элемента есть 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. Обычно приоритетные очереди реализуются с помощью куч (heap).

**DHeapBasedPriorityQueue** является наследником класса **PriorityQueue** и переопределяет его методы.

* **Поля:**
* **\*heap (DHeap)** - указатель на кучу.
* **Методы:**
* **DHeapBasedPriorityQueue(int d=4)** - Конструктор.
* **DHeapBasedPriorityQueue (Data \*\*keys, int num, int d)** - конструктор с добавлением гуппы элементов.
* **~ DHeapBasedPriorityQueue ()** - деструктор.
* **virtual void Add(Data \*&key)** — метод вставки элемента в приоритетную очередь с указанным ключом.
* **virtual Data\* Pop(int i)** — метод выполняющий всплытие элемента приоритетной очереди с i-ым ключом.
* **virtual void Refresh()** — метод упорядочивающий приоритетную очередь.
* **virtual int IsFull()** — метод сигнализирующий о переполненности приоритетной очереди.
* **virtual int IsEmpty()** — метод сигнализирующий о пустоте приоритетной очереди.

Заключение

В ходе лабораторной работы была разработана программа, удовлетворяющая поставленным задачам. Реализованы классы D-арная куча и приоритетная очередь. Реализован Алгоритм Дейкстры.

Литература

* Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: «Вильямс», 2006. — С. 1296.
* Белоусов А. И., Ткачев С. Б. Дискретная математика. — М.: МГТУ, 2006. — 744 с.

Приложения