Санкт-Петербургский политехнический университет Петра Великого

Институт прикладной математики и информатики

Высшая школа прикладной математики и вычислительной физики

Лабораторная работа №2 по дисциплине

Дискретная математика

Тема: «Графы»

Вариант 5 – Алгоритм Дейкстры

Выполнил студент гр. 5030102/20202 Соколов А.Н.

Руководитель Новиков Ф. А.

«___» _____ 202__ г.

Санкт-Петербург

2024

1. Формулировка задачи и ее формализация 3
2. Использованные технологии 4
Исходные файлы программы: 4
3. Описание алгоритма кодировки 5
4. Практическая реализация 10
6. Формат входных и выходных данных 13
7. Сравнение работы алгоритма на различных допустимых входных данных 14
8. Вывод 16

1. Формулировка задачи и ее формализация

Формулировка задачи

1. Необходимо разработать консольное приложение, реализующее функции поиска выгоднейшего пути по графам.
2. Поддержать возможность вывода числа оценки пути и визуализацию пути.
3. Указать сложность алгоритма и доказать, что она именно такая.
4. Сравнение работы алгоритма на различных допустимых входных данных: на каких графах алгоритм работает лучше, на каких – хуже, на каких – вообще не работает
5. Объяснить почему был выбран тот или иной способ представления графов в программе.

2. Использованные технологии

Язык программирования

- C++ 23

Система сборки

- CMake 3.27
- Ninja 1.12.1

Исходные файлы программы:

https://github.com/azya0/dm2025/tree/master/lab2

3. Описание алгоритма кодировки

Алгоритм Дейкстры — это известный алгоритм для поиска кратчайшего пути в взвешенном орграфе с **неотрицательными весами** ребер. Он назван в честь нидерландского математика Эдсгера Дейкстры, который предложил его в 1956 году. Рассмотрим, как работает этот алгоритм, а также его сложность.

Принцип работы алгоритма

Инициализация:

У нас есть граф с вершинами и весами ребер.
Мы выбираем начальную вершину и устанавливаем для нее расстояние до самой себя равным 0. Для всех остальных вершин расстояния устанавливаем "бесконечностью"*****.
Создаем множество, которое будет хранить все вершины, для которых мы уже нашли кратчайшие пути. Изначально оно содержит одну вершину (исходную).

Обработка вершин:

Пока есть необработанные вершины, выбираем вершину с минимальным расстоянием из начальной. Если расстояние до выбранной вершины "бесконечно"*****, значит, все доступные вершины были обработаны, и алгоритм завершает свою работу.
Для каждой соседней вершины (то есть вершины, которая связана с текущей) проверяем, можно ли улучшить найденное расстояние к ней.

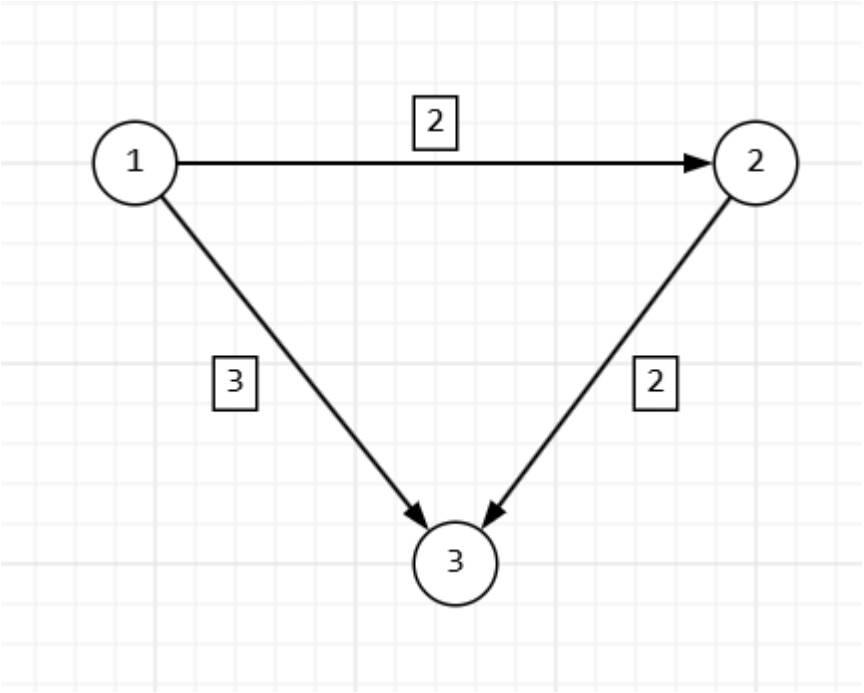
Алгоритм Дейкстры рационально применять в задачах оптимизации маршрутов и логистики задачах. Например: Навигационные системы, Игры (стратегии), Планировка и логистика, Веб-приложениях, Сетей доставок и т.д.

** “бесконечность” - условное обозначение в оригинальном описании алгоритма. В рамках моей работы на практике никакая*

“бесконечность” или предельные значения типов данных не используются.

Пример:

Предположим, у нас есть взвешенный орграф:



Пусть мы рассматриваем маршруты от вершины “1”

1. Мы сопоставляем вершине “1” число 0, т.к. мы итак изначально находимся в ней, а всем остальным “бесконечность”
2. **Рассматриваем все ребра из “1”**: “1 -> 2”, “1 -> 3”
3. **Т.к. алгоритм Дейкстры - жадный алгоритм**, то мы выбираем кратчайшее ребро, а именно ребро “1 -> 2” с весом 2
4. **Пройдя по ребру,** мы рассчитываем ценность маршрута: 0 + 2 = 2
5. **Теперь мы сравниваем вес ребра “1 - > 2” с бесконечностью.** Очевидно, что 2 меньше бесконечности, поэтому теперь вершина “2” сопоставлена с числом 2
6. **Теперь рассмотрим ребро “1 -> 3”.**
7. **Пройдя по ребру,** мы рассчитываем ценность маршрута: 0 + 3 = 3

8. **Сравним.** Очевидно, что 3 меньше бесконечности. Поэтому оценку пути из “1” в “3” устанавливаем на 3.
9. **Выбираем наименьшее ребро: “2 -> 3”** с весом 2
10. **Пройдя по ребру,** мы рассчитываем ценность маршрута: 2 + 2 = 4
11. **Оценим.** 4 > 3, поэтому обновлять цену маршрута из “1” в “3” не будем
12. **Рассмотрим ребра из “3”**. Их нет.
13. **Алгоритм завершен.**

Таким образом из вершины “1” мы можем добраться кратчайшим маршрутом до вершины “2” за 2 и до “3” за 3

Сложность алгоритма

Сложность алгоритма Дейкстры зависит от использования различных структур данных:

В классической реализации с использованием списка смежности и простого массива:

O(N), где N — количество вершин.

Обновление расстояний для всех соседей занимает

$O(E)$, где E — количество ребер.

Общая сложность:

$O(E * N + N)$

В моей реализации предполагается, что вершин будет больше, чем ребер ($E << N^2$) $\sim N$, поэтому используются односвязные списки, стек и хеш-таблица:

В алгоритме используется цикл, который продолжает выполняться, пока есть не обработанные узлы.

На практике, в исходной версии алгоритма добавление в конец динамического массива происходит за $O(1)$, а поиск минимального элемента происходит за $O(N)$. В моей реализации поиск минимального элемента упорядоченного односвязного списка происходит за $O(1)$, а добавление элемента за $O(N)$

Стоит отметить, что изначально поиск минимального элемента гарантированно проверит все N элементов массива, а добавление элемента в упорядоченную очередь будет искать элемент больше себя, который может быть не последним

Таким образом сложность моего алгоритма составляет:

$O(E * N + N)$

Вывод: на густых графах он неэффективен.

Почему такая сложность?

Основная причина такой сложности заключается в том, что алгоритм обходит каждую вершину и каждое ребро графа, чтобы гарантировать, что все кратчайшие пути найдены.

возникает из-за необходимости искать минимальную вершину среди всех непроверенных, что делает его неэффективным для больших графов.

С переходом на более эффективные структуры данных (например приоритетные очереди) значительно улучшается время работы, так как выбор минимального расстояния и обновление расстояний происходит быстрее.

4. Практическая реализация

В программе граф представлен как хеш-таблица, в которой ключи - имена графов, а значения - умные указатели на объекты класса Node:

```
class Node {
public:
    using Rib = std::pair<std::shared_ptr<Node>, int>;

    Node(std::string const & name);
    Node(std::shared_ptr<std::vector<Rib>> nodes, std::string const &
name);

    std::shared_ptr<std::vector<Rib>> Nodes();

    void addRib(std::shared_ptr<Node> rib, int weight);

    std::vector<Node::Rib> getRibs();

    std::string const & getName();
private:
    std::vector<Node::Rib> ribs;
    std::string name;
};
```

Такое представление было выбрано, т.к. с ним удобно работать и оно подразумевает использование всей выделенной под него памяти. Например, если бы я представлял граф как матрицу, то часть значений были бы однотипной записью по типу “-1”, которое расходует память в пустую и требует $O(n^2)$ сложность вывода в консоль.

Отличием от стандартного алгоритма Дейкстры заключается в его частичной оптимизации:

- Использование хеш-таблиц
- Использование односвязных списков (очередей)

```
    // OWL вместо стека для удобной
    // сортировки для нахождения
```

```
// минимального элемента
// для оптимизации жадного
// алгоритма
typedef struct OWL {
    std::shared_ptr<OWL> next;
    std::shared_ptr<Pair> value;
} OneWayList;

std::unordered_map<
    std::shared_ptr<Node>,
    std::shared_ptr<Pair>
> ways;
```

Необработанные вершины

Программа предполагает, что вместе с исполняемым файлом пользователь будет хранить файл произвольного расширения, описывающий граф в формате:

A 2 B 5 C 9
B 3 A 2 C 3 D 1
C 1 A 9
D 1 C 1
E 0

Где каждая строка - это имя вершины, количество ребер. Для каждого ребра - имя вершины, в которую ведет ребро, а также вес (цену) этого ребра.

После запуска исполняемого файла необходимо указать путь до файла.

5. Область применения

Файл с графом

- Неверный формат файла:
    - В файле содержится неправильный формат
- Отрицательные веса в файле:
    - Алгоритм Дейкстра не работает с отрицательными весами

Ввод названия исходной вершины:

- Ввод несуществующей вершины

Работа программы

- Оценка маршрута превышает размер типа данных **unsigned int:** больше, чем 4 294 967 294

Во всех остальных случаях программа будет работать корректно

6. Формат входных и выходных данных

На вход программа получает:

1. Название файла с графом в определенном формате:
   “Название вершины” количество ребер “Название вершины” “Вес ребра”...

Пример:

A 2 B 5 C 9
B 3 A 2 C 3 D 1
C 1 A 9
D 1 C 1
E 0

2. Исходную вершину

В качестве вывода программа выведет все возможные маршруты с оценкой или сообщит, что такой маршрут невозможен:

Пример:

no way A -> E
A -> B : 5 A -> B
A -> D : 6 A -> B -> D
A -> C : 7 A -> B -> D -> C
A -> A : 0 A

7. Сравнение работы алгоритма на различных допустимых входных данных

Алгоритм Дейкстры оптимален в определенных ситуациях, но его производительность и корректность зависят от типа графа, с которым он работает. Давайте рассмотрим различные типы графов и как алгоритм Дейкстры себя ведет на каждом из них:

Графы с неотрицательными весами

Как работают: На графах с неотрицательными весами алгоритм Дейкстры работает очень эффективно и корректно. Он способен находить кратчайшие пути от одной стартовой вершины ко всем остальным.

Графы с отрицательными весами

Как работают: На графах с отрицательными весами алгоритм Дейкстры не может гарантировать правильность. Он может завершить работу, не найдя реальный кратчайший путь или не завершить работу вовсе.

Сложные графы (редкие и густые)

Редкие графы: Для графов с маленьким количеством ребер по сравнению с количеством вершин (например, с графами с предельной связностью, такими как деревья), алгоритм Дейкстры будет эффективным, поскольку количество операций по обновлению расстояний будет меньше.

Густые графы: В графах с большим количеством ребер (вдобавок к количеству вершин) эффективность уменьшается из-за увеличения времени, затрачиваемого на обработку ребер. Но алгоритм все равно будет работать корректно, его производительность просто будет ниже.

Ациклические графы (DAGs)

Как работают: На ациклических направленных графах алгоритм Дейкстры будет работать корректно и эффективно. Поскольку в DAG нет циклов, алгоритм будет проходить по каждой вершине только один раз, что делает его эффективным.

Полные графы

Как работают: В полных графах каждая пара вершин соединена ребром. Алгоритм Дейкстра будет работать достаточно эффективно, хотя при большом количестве вершин общее количество ребер будет расти, что влияет на оперативность выполнения.

Разреженные графы

Как работают: В разреженных графах количество ребер много меньше, чем количество вершин в квадрате. Алгоритм Дейкстра будет работать с лучшей скоростью. (Причины описаны в обосновании времени работы алгоритма)

8. Вывод

В рамках данной лабораторной работы был реализован алгоритм поиска оптимального пути по графу Дейкстра.