**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**“Московский государственный университет геодезии и картографии” (МИИГАИК)**

**Факультет геоинформатики и информационной безопасности**

**Кафедра геоинформационных систем и технологий**

**Лабораторная работа №5**

**"Алгоритм поиска кратчайшего пути"**

**Преподаватель: Лебедев Евгений Денисович**

**Работу выполнил: Зазыкин Иван Дмитриевич**

**Группа: 2024-ФГИИБ-ПИ-1б**

**Вариант: 11**

**Москва 2025**

**Формулировка задания**

В ходе работы необходимо реализовать на C++ алгоритм Дейкстры для поиска кратчайшего маршрута в реальной дорожной сети города Варшава, Польша. Программа должна считывать граф из файла формата .graphml, находить путь между двумя заданными улицами и визуализировать результат с помощью библиотеки SFML. Также требуется выполнить пошаговый ручной расчет алгоритма для графа, указанного в варианте, и провести несколько экспериментальных замеров для разных маршрутов, оформив результаты в виде таблицы.

**Ссылка на GitHub репозиторий:**

<https://github.com/re-side/Inf_university/tree/main/ALG/lab5>

**Формулировка алгоритма Дейкстры**

**Алгоритм Дейкстры** — это итерационный жадный алгоритм для нахождения кратчайших путей от одной (стартовой) вершины до всех остальных вершин в взвешенном графе. Ключевое требование для корректной работы алгоритма — все веса рёбер в графе должны быть неотрицательными.

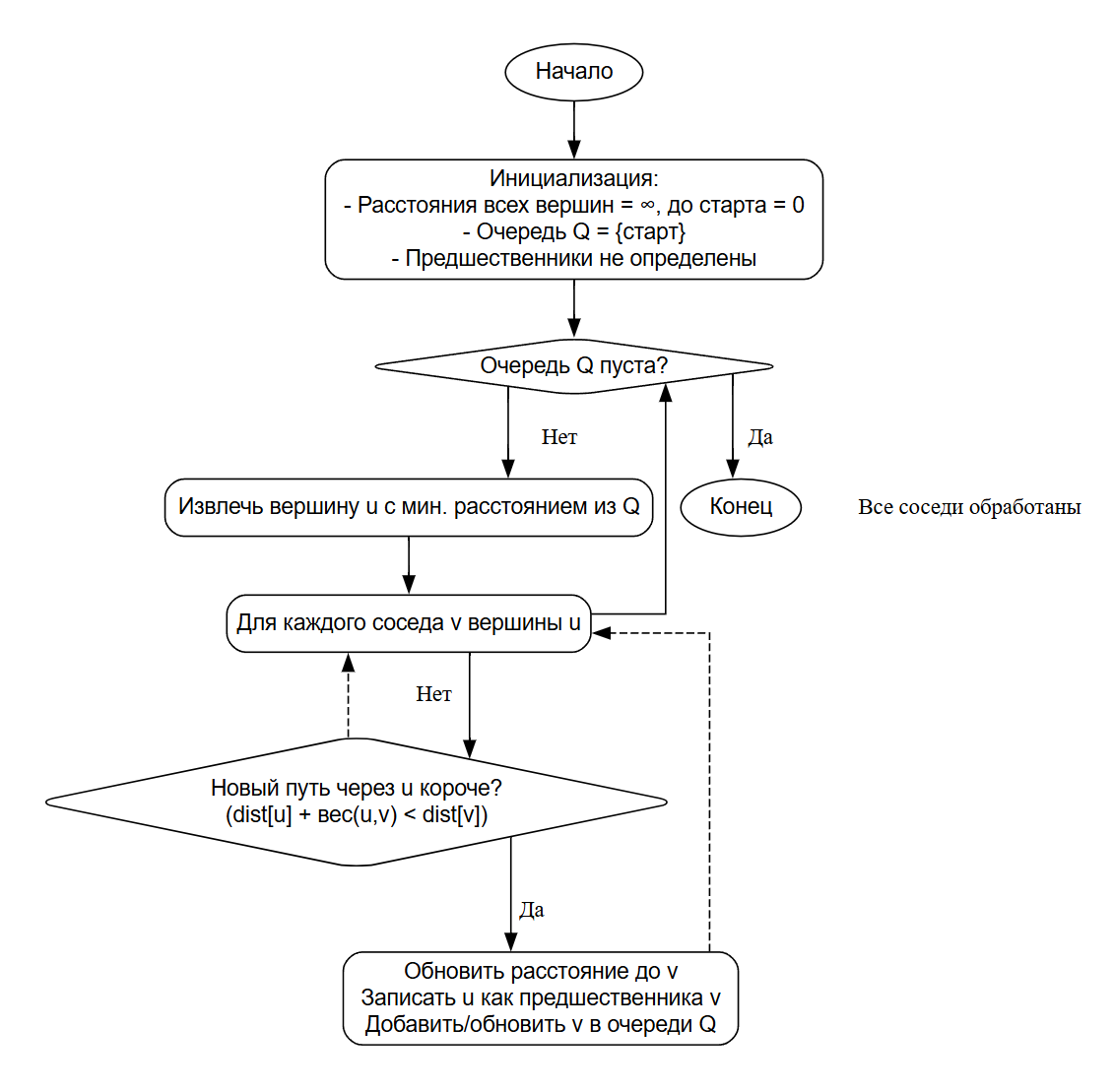
**Основная идея** заключается в том, чтобы пошагово расширять множество вершин, для которых уже известен кратчайший путь. На каждом шаге алгоритм выбирает "ближайшую" из еще не посещенных вершин и добавляет ее в это множество.

Для работы алгоритму требуются следующие структуры данных:

1. **Массив расстояний**: Хранит текущую известную длину кратчайшего пути от стартовой вершины до каждой другой. Изначально для стартовой вершины расстояние равно 0, для всех остальных — бесконечности.
2. **Очередь с приоритетом**: Хранит вершины, которые предстоит посетить. Приоритет определяется текущим расстоянием до вершины (чем меньше расстояние, тем выше приоритет). Это позволяет эффективно реализовывать "жадный" выбор — на каждом шаге брать самую близкую из еще не обработанных вершин.
3. **Множество посещенных вершин**: Хранит вершины, для которых кратчайший путь уже окончательно найден. Это предотвращает повторную обработку и зацикливание.
4. **Массив предшественников**: Хранит для каждой вершины ту, из которой был сделан самый "удачный" (короткий) шаг. Этот массив используется в конце для восстановления самого маршрута.

**Процесс работы алгоритма:**

1. **Инициализация**:
   * Расстояние до стартовой вершины S устанавливается в 0, до всех остальных — в бесконечность.
   * В очередь с приоритетом добавляется стартовая вершина S с приоритетом 0.
2. **Основной цикл**: Пока очередь с приоритетом не пуста, выполняются следующие действия:
   * Из очереди извлекается вершина u с наименьшим расстоянием (наивысшим приоритетом).
   * Если вершина u уже была посещена, переход к следующей итерации.
   * Вершина u помечается как посещенная.
   * Для каждого соседа v вершины u выполняется операция **релаксации**:
     + Вычисляется новое, потенциальное расстояние до v через u: новое\_расстояние = dist[u] + вес\_ребра(u, v).
     + Если новое\_расстояние меньше текущего dist[v], то dist[v] обновляется этим новым значением, а предшественником v становится u. Обновленная пара (dist[v], v) добавляется в очередь.
3. **Завершение**: Алгоритм завершается, когда очередь с приоритетом становится пустой. К этому моменту в массиве dist содержатся длины кратчайших путей от S до всех достижимых вершин. Путь до любой вершины можно восстановить, двигаясь от нее назад по массиву предшественников до стартовой вершины.



**Расчеты для графа по варианту 11**

**Условие:**

* **Граф:** A–B: 2, A–C: 4, A–D: 5, B–C: 3, B–D: 6, C–D: 1.
* **Задание:** Найти кратчайший путь из вершины **D** в вершину **B**.

**Решение с помощью алгоритма Дейкстры:**

**Шаг 1: Инициализация**

* Начальная вершина: **D**.
* Массив расстояний Dist инициализируется нулем для стартовой вершины и бесконечностью для остальных.
* Dist = { A: ∞, B: ∞, C: ∞, D: 0 }
* Множество посещенных вершин H пустое: H = {}.

**Итерация 0**

1. Выбираем из непосещенных вершин ту, что с наименьшим расстоянием: **D** (расстояние 0).
2. Помещаем **D** в множество посещенных: H = {D}.
3. Рассматриваем всех соседей вершины D (A, B, C) и обновляем расстояния до них (релакс):
   * Dist[A] = min(∞, Dist[D] + вес(D, A)) = min(∞, 0 + 5) = **5**
   * Dist[B] = min(∞, Dist[D] + вес(D, B)) = min(∞, 0 + 6) = **6**
   * Dist[C] = min(∞, Dist[D] + вес(D, C)) = min(∞, 0 + 1) = **1**
4. Текущие известные кратчайшие расстояния: Dist = { A: 5, B: 6, C: 1, D: 0 }.

**Итерация 1**

1. Выбираем следующую непосещенную вершину с наименьшим расстоянием: **C** (расстояние 1).
2. Помещаем **C** в множество посещенных: H = {D, C}.
3. Рассматриваем соседей вершины C (A, B, D). Вершина D уже посещена, ее пропускаем.
   * Dist[A] = min(5, Dist[C] + вес(C, A)) = min(5, 1 + 4) = **5** (расстояние не изменилось).
   * Dist[B] = min(6, Dist[C] + вес(C, B)) = min(6, 1 + 3) = **4** (путь через C короче, **обновляем расстояние**).
4. Текущие известные кратчайшие расстояния: Dist = { A: 5, B: 4, C: 1, D: 0 }.

**Итерация 2**

1. Выбираем следующую непосещенную вершину с наименьшим расстоянием: **B** (расстояние 4).
2. Вершина **B** является нашей конечной целью. Алгоритм можно завершить.

**Ответ:**

* Кратчайшее расстояние от **D** до **B** равно **4**.
* **Восстановление пути** (идем обратно от конечной вершины по предшественникам):
  + В вершину **B** мы пришли из **C**, так как Dist[C] + вес(C,B) = 1 + 3 = 4.
  + В вершину **C** мы пришли из **D**, так как Dist[D] + вес(D,C) = 0 + 1 = 1.
* Итоговый кратчайший путь: **D → C → B**.

**Граф для города по варианту (Варшава, Польша)**

В рамках лабораторной работы был использован реальный граф дорог для города **Варшава, Польша**. Данные были загружены с помощью библиотеки OSMnx в среде Python и сохранены в файл warsaw\_road\_network.graphml для последующей обработки в C++.

Анализ загруженного графа показал следующие характеристики:

* **Количество вершин (узлов):** 19710
* **Количество ребер (дорог):** 44755

****Ниже представлен скриншот визуализации части дорожной сети Варшавы, полученный в результате работы программы. Серыми линиями показана общая дорожная сеть, а красным цветом выделен один из построенных кратчайших маршрутов.

**Таблица 1: Расчет кратчайших путей для города Варшава, Польша**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Пункт старта (улица)** | **Пункт назначения (улица)** | **Расстояние (км)** | **Рисунок (результат)** | **Время работы алгоритма (с)** |
| Adama Asnyka | Astronautów | 4.34 |  | 0.191 |
| Jeziorna | Rodziny Hiszpańskich | 4.93 |  | 0.212 |
| Adama Asnyka | Świętej Rodziny | 7.52 |  | 0.278 |
| Żwirki i Wigury | Powstańców Warszawy | 21.42 |  | 0.548 |