Санкт-Петербургский государственный
политехнический университет

Кафедра “Прикладная математика”

Отчет по лабораторной работе 2 – Графы. Алгоритм Дейкстры. Дисциплина “Дискретная математика”

Выполнл студент группы 5030102/20201 Сивошенко А.Ю

30 октября 2024г.

Оглавление

Введение

В данной лабораторной работе требуется реализовать алгоритм Дейкстры для нахождения кратчайших путей в взвешенном ориентированном графе.

Язык программирования и его версия

Реализация выполнена на языке программирования C++ с использованием стандарта C++17.

Описание алгоритма

Алгоритм Дейкстры — это алгоритм для нахождения кратчайших путей от одной вершины (источника) ко всем другим вершинам в графе с неотрицательными весами дуг.

Псевдокод алгоритма:

```
function Dijkstra(Graph, source):
    create vertex set Q
    for each vertex v in Graph:
        dist[v] := INFINITY
        prev[v] := UNDEFINED
        add v to Q
    dist[source] := 0
    while Q is not empty:
        u := vertex in Q with min dist[u]
        remove u from Q
        for each neighbor v of u:
            alt := dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] := alt
                prev[v] := u
    return dist[], prev[]
```

Описание:

- Инициализация: расстояния всех вершин устанавливаются в бесконечность, для начальной вершины — в ноль.
- Выбор вершины с минимальным расстоянием и её удаление из множества нерассмотренных.
- Релаксация дуг текущей вершины.
- Продолжение до тех пор, пока все вершины не будут обработаны.

Сложность алгоритма и её обоснование

Алгоритм Дейкстры реализован с помощью приоритетной очереди и представления графа в виде списка смежности. В таком варианте алгоритм достигает временной сложности (O((V + E) log V)), где (V) — количество вершин, а (E) — количество дуг графа.

1. Инициализация: (O(V))
   - Каждая вершина сначала инициализируется с бесконечным расстоянием, и для всех предшествующих вершин устанавливается начальное значение (−1) или аналогичное. Эта операция требует (O(V)) времени, так как это происходит для каждой из (V) вершин.
2. Операции на очереди с приоритетом:
   - Мы используем `std::priority_queue` для выбора вершины с минимальным текущим расстоянием. Каждое добавление и извлечение из приоритетной очереди выполняется за (O(log V)).
   - Мы помещаем (V) вершин в очередь и обрабатываем каждую вершину один раз; таким образом, это струкруно (O(V log V)).
3. Посещение дуг: (O(E log V))
   - Для каждой вершины мы рассматриваем все её исходящие дуги. В сумме, во всём графе будет рассматриваться (E) дуг.
   - При каждом рассмотрении дуги мы можем обновить расстояние до соседней вершины, что требует вставки в приоритетную очередь (если новое расстояние оказалось меньшим), также выполняемой за (O(log V)). Следовательно, общая стоимость обработки дуг (релаксации) составляет (O(E log V)).

Сравнение работы алгоритма на различных типах графов

- Вклад (V log V): Это максимально возможное количество операций вставки и извлечения в приоритетной очереди, где (V) — количество вершин. Процесс выбирает вершину с наименьшим расстоянием и разбирает его дуги.
- Вклад (E log V): Все дуги должны быть обработаны; каждая обработка требует возможного обновления в приоритетной очереди, что делает сложность (O(log V)) за одну операцию в худшем случае.
- Эффективность: Сложность показывает, что алгоритм Дейкстры наиболее эффективен для разреженных графов, где (E approx V), так как процесс обработки дуг существенно зависит от количества связанных компонент.

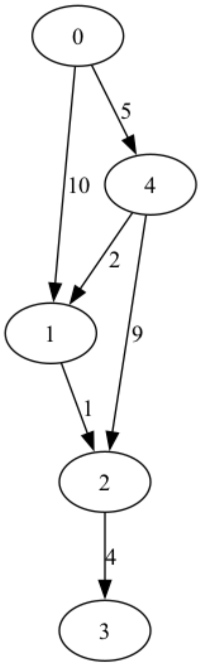
Выбор представления графа

Графы в этой программе представлены в виде списка смежности, потому что:

- Эффективность в разреженных графах: Способствует снижению затрат по памяти, так как хранится только существующие дуги. Это особенно полезно, когда граф разрежен.
- Оптимальные операции на соседях: Извлечение и обработка каждого списка соседей для каждой вершины происходит быстрее по сравнению с матрицей инцидентности и улучшает производительность для алгоритмов на графах, таких как алгоритм Дейкстры.

Демонстрация работы алгоритма

Для демонстрации алгоритма возьмём простой граф с 5 вершинами, представленный в формате DOT. Представим его следующим образом:




```
digraph G {
    0 -> 1 [label="10"];
    0 -> 4 [label="5"];
    1 -> 2 [label="1"];
    4 -> 1 [label="2"];
    4 -> 2 [label="9"];
    2 -> 3 [label="4"];
}
```

Исходные условия

Мы начнём с вершины 0. В начале алгоритма все расстояния от источника (вершина 0) до остальных вершин установлены в бесконечность, за исключением расстояния до самой себя, которое равно 0.

Инициализация:

- $(\mathrm{T}[0] = 0)$
- $(\mathrm{T}[1] = \infty)$
- $(\mathrm{T}[2] = \infty)$
- $(\mathrm{T}[3] = \infty)$
- $(\mathrm{T}[4] = \infty)$

Шаги алгоритма

Шаг 1: Обработка вершины 0

1. Извлекаем вершину 0, так как её расстояние равно нулю — оно минимальное.
2. Рассчитываем новые расстояния для её соседей:
   - Для вершины 1 через 0: ($T[1] = 0 + 10 = 10$)
   - Для вершины 4 через 0: ($T[4] = 0 + 5 = 5$)

Очередь теперь: ([1 (10), 4 (5)])

Вектор расстояний (T):

- ($T[0] = 0$)
- ($T[1] = 10$)
- ($T[2] = \infty$)
- ($T[3] = \infty$)
- ($T[4] = 5$)

Вектор предшественников (Pi):

- ($Pi[0] = -1$) (или `undefined`)
- ($Pi[1] = 0$)
- ($Pi[2] = -1$)
- ($Pi[3] = -1$)
- ($Pi[4] = 0$)

Шаг 2: Обработка вершины 4

1. Извлекаем вершину 4 (расстояние 5).
2. Рассчитываем новые расстояния для её соседей:
   - Для вершины 1 через 4: ($T[1] = \min(10, 5 + 2) = 7$) (обновляем)
   - Для вершины 2 через 4: ($T[2] = 5 + 9 = 14$)

Очередь теперь: ([1 (7), 2 (14)])

Вектор расстояний (T):

- ($T[0] = 0$)
- ($T[1] = 7$)
- ($T[2] = 14$)
- ($T[3] = \infty$)
- ($T[4] = 5$)

Вектор предшественников (Pi):

- ($Pi[0] = -1$)
- ($Pi[1] = 4$)
- ($Pi[2] = 4$)
- ($Pi[3] = -1$)

- $(Pi[4] = 0)$

Шаг 3: Обработка вершины 1

1. Извлекаем вершину 1 (расстояние 7).
2. Рассчитываем новые расстояния для её соседей:
   - Для вершины 2 через 1: $(T[2] = \min(14, 7 + 1) = 8)$ (обновляем)

Очередь теперь: $([2\ (8), 2\ (14)])$

Вектор расстояний (T):

- $(T[0] = 0)$
- $(T[1] = 7)$
- $(T[2] = 8)$
- $(T[3] = \infty)$
- $(T[4] = 5)$

Вектор предшественников (Pi):

- $(Pi[0] = -1)$
- $(Pi[1] = 4)$
- $(Pi[2] = 1)$
- $(Pi[3] = -1)$
- $(Pi[4] = 0)$

Шаг 4: Обработка вершины 2

1. Извлекаем вершину 2 (расстояние 8).
2. Рассчитываем новые расстояния для её соседей:
   - Для вершины 3 через 2: $(T[3] = 8 + 4 = 12)$

Очередь теперь: $([3\ (12)])$

Вектор расстояний (T):

- $(T[0] = 0)$
- $(T[1] = 7)$
- $(T[2] = 8)$
- $(T[3] = 12)$
- $(T[4] = 5)$

Вектор предшественников (Pi):

- $(Pi[0] = -1)$
- $(Pi[1] = 4)$
- $(Pi[2] = 1)$
- $(Pi[3] = 2)$

- $(\mathrm{Pi}[4] = 0)$

Шаг 5: Обработка вершины 3
1. Извлекаем вершину 3 (расстояние 12). Никаких соседей для обновления.

Очередь теперь пуста, алгоритм завершается.

Кратчайшие расстояния от начальной вершины 0 до всех остальных вершин:
- Vertex 0: расстояние 0 (начальная вершина)
- Vertex 1: расстояние 7, кратчайший путь: 0 -> 4 -> 1
- Vertex 2: расстояние 8, кратчайший путь: 0 -> 4 -> 1 -> 2
- Vertex 3: расстояние 12, кратчайший путь: 0 -> 4 -> 1 -> 2 -> 3
- Vertex 4: расстояние 5, кратчайший путь: 0 -> 4

Область применения и возможные ошибки

Алгоритм Дейкстры используется там, где требуется определение кратчайшего пути между узлами в графе.

Возможные ошибки:

1. Отрицательные веса дуг: Алгоритм Дейкстры работает корректно с дугами отрицательной длины, только в случае, если сумма модулей дуг меньше INT_MAX.
2. Неполные входные данные: Если начальная вершина или дуги не указаны корректно в формате входных данных, это приведёт к некорректной работе алгоритма.

Формат входных и выходных данных

- Входные данные: Граф задаётся в формате DOT, используемом для описания графов в Graphviz. Каждое дуга содержится в отдельной строке с указанием его веса в атрибуте “label”.

  Пример входного файла (ненаправленный граф):

```
digraph G {
    0 -> 1 [label="10"];
    0 -> 4 [label="5"];
    1 -> 2 [label="1"];
    4 -> 1 [label="2"];
    4 -> 2 [label="9"];
    2 -> 3 [label="4"];
}
```

- Выходные данные: Результаты выполнения алгоритма сохраняются в текстовый файл. Там указываются:
  - Кратчайшие расстояния от начальной вершины до каждой вершины.
  - Предшествующие вершины для построения пути.

  Пример выходного файла:

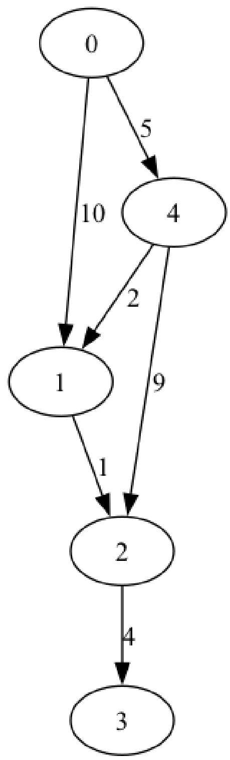
```
Distances from source:
Vertex 0: 0
Vertex 1: 7
Vertex 2: 8
Vertex 3: 12
Vertex 4: 5

Paths (preceding vertex):
Vertex 0: -1
Vertex 1: 4
Vertex 2: 1
Vertex 3: 2
Vertex 4: 0
```

Примеры работы

1. Пример 1:

   Входной граф:

```
digraph G {
    0 -> 1 [label="10"];
    0 -> 4 [label="5"];
    1 -> 2 [label="1"];
    4 -> 1 [label="2"];
    4 -> 2 [label="9"];
    2 -> 3 [label="4"];
}
```



Результат:

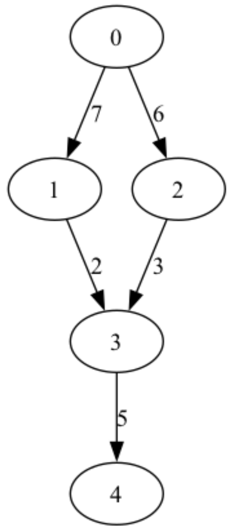
```
from source:
Vertex 0: 0
Vertex 1: 7
Vertex 2: 8
Vertex 3: 12
Vertex 4: 5

Paths (preceding vertex):
Vertex 0: -1
Vertex 1: 4
Vertex 2: 1
```

```
Vertex 3: 2
Vertex 4: 0
```

2. Пример 2:

Входной граф:

```
digraph G {
    0 -> 1 [label="7"];
    1 -> 3 [label="2"];
    0 -> 2 [label="6"];
    2 -> 3 [label="3"];
    3 -> 4 [label="5"];
}
```

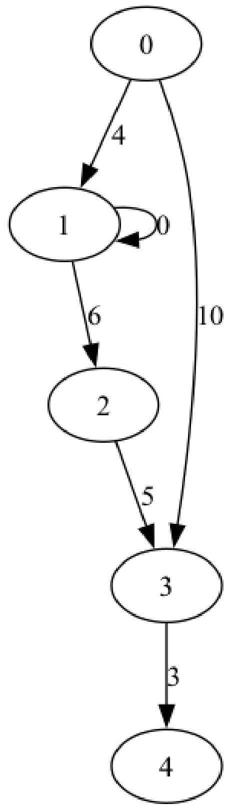


Результат:

```
Distances from source:
Vertex 0: 7
Vertex 1: 0
Vertex 2: 5
Vertex 3: 2
Vertex 4: 7

Paths (preceding vertex):
Vertex 0: 1
Vertex 1: -1
Vertex 2: 3
Vertex 3: 1
Vertex 4: 3
```

3. Пример 3:

Zero граф:

`digraph G {`
`    0 -> 1 [label="4"];`
`    1 -> 2 [label="6"];`
`    2 -> 3 [label="5"];`
`    3 -> 4 [label="3"];`
`    0 -> 3 [label="10"];`
 	
`    1 -> 1 [label="0"];`
`}`
	

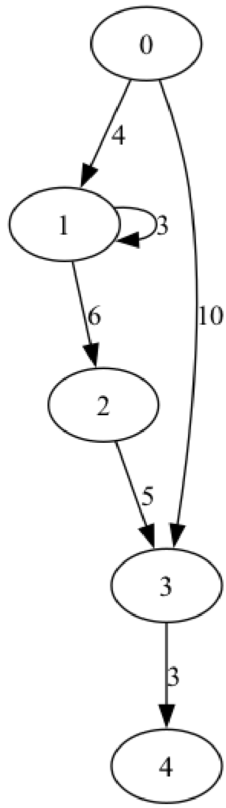
Результат:

```
from source:
Vertex 0: 0
Vertex 1: 4
Vertex 2: 10
Vertex 3: 10
Vertex 4: 13

Paths (preceding vertex):
Vertex 0: -1
Vertex 1: 0
Vertex 2: 1
Vertex 3: 0
Vertex 4: 3
```

6. Пример 6:

Nzero граф:

```
digraph G {
    0 -> 1 [label="4"];
    1 -> 2 [label="6"];
    2 -> 3 [label="5"];
    3 -> 4 [label="3"];
    0 -> 3 [label="10"];

    1 -> 1 [label="3"];
}
```



Результат:

```
from source:
Vertex 0: 0
Vertex 1: 4
Vertex 2: 10
Vertex 3: 10
Vertex 4: 13

Paths (preceding vertex):
Vertex 0: -1
Vertex 1: 0
Vertex 2: 1
Vertex 3: 0
Vertex 4: 3
```

Заключение

В ходе работы была реализована версия алгоритма Дейкстры. Разработанная программа потенциально пригодна для применения во многих практических задачах и может быть модифицирована под расширенные требования.