Санкт-Петербургский Политехнический Университет Петра Великого
Институт машиностроения, материалов и транспорта
Кафедра "Мехатроника и роботостроение (при ЦНИИ РТК)"

Курсовая работа

Дисциплина: объектно-ориентированное программирование
Тема: Алгоритм Дейкстры

Работу выполнил: Пантелеев М.Д.
студент группы 3331506/80401
Преподаватель: Кузнецова Е.М.

Санкт-Петербург
2021

Содержание

1 Введение

Графы в том или ином виде присутствуют во многих сферах деятельности. Они используются и в естественных науках (физике и химии) и в социальных науках (например, социологии), но наибольших масштабов применение графов получило в информатике и сетевых технологиях. Одной из важнейших задач на графах же является поиск кратчайшего пути.

Алгоритм Дейкстры, созданный Эдсгером Вибе Дейкстрой в 1956 году, позволяет решить эту задачу для взвешенного ориентированного графа без отрицательных рёбер.

2 Описание работы

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

Алгоритм:

1. Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. На начальном этапе метки всех вершин, кроме a, равны ∞ (это означает, что расстояние до них пока не известно). Также введем множество p(V), которое содержит предпоследнюю вершину на пути от a к любой вершине из V.

2. Все вершины графа помечаются как непосещенные

3. Пока все вершины не посещены:

   3.1 Выбрать еще не посещенную вершину u, метка которой минимальна.

   3.2 Пометить вершину u как посещенную;

   3.3 Для каждого соседа u (соседи u – вершины, в которые ведут ребра из u) рассмотрим новую длину пути, вычисляемую как сумма метки u и длины ребра из u к соседу.

   3.4 Если полученное значение меньше текущей метки соседа, то заменяем его метку полученным значением пути, а также вносим вершину u в множество p.

3 Реализация алгоритма

На листинге 1 представлена реализация алгоритма Дейкстры на C++.

Листинг 1: Программный код реализации

```
void Dijkstra(int graphTable[MAX_SIZE][MAX_SIZE], int startNode, int fieldSize)
{
    int distance[MAX_SIZE];
    bool visited[MAX_SIZE];

    for (int i=0; i<fieldSize; i++)
    {
        distance[i]=INT_MAX;
        visited[i]=false;
    }
    distance[startNode]=0;
    int curNode = 0;

    for (int counter=0; counter<fieldSize-1; counter++)
    {
        int minDistance=INT_MAX;
        int minIndex = 0;
        for (int i=0; i<fieldSize; i++)
            if (!visited[i] && distance[i]<=minDistance)
            {
                minDistance=distance[i];
                minIndex=i;
            }
        curNode=minIndex;
        visited[curNode]=true;
        for (int i=0; i<fieldSize; i++)
            if (!visited[i] && graphTable[curNode][i] && distance[curNode]!=INT_MAX &&
         distance[curNode]+graphTable[curNode][i]<distance[i])
            {
```

```
                    distance[i]=distance[curNode]+graphTable[curNode][i];
                }
        }
}
```

Пример результата работы программы для графа, представленного на рисунке 1, изображён на рисунке 2.

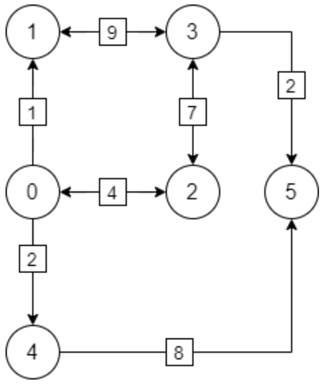

Рис. 1: Граф

Path length:
0 -> 0 = 0
0 -> 1 = 1
0 -> 2 = 4
0 -> 3 = 10
0 -> 4 = 2
0 -> 5 = 10

Рис. 2: Вывод программы

4 Анализ выполнения

Время работы алгоритма Дейкстры складывается из двух основных составляющих: время нахождения вершины с наименьшей меткой (1) и время изменения значения метки при необходимости (2).

Сложность алгоритма в основном зависит от структуры данных, используемой для представления графа. При использовании матрицы смежности для задания графа, асимптотика операции 1 составит $O(n)$, а операции 2 - $O(1)$. Первая операция выполняется n раз, а вторая m раз (n – количество вершин графа; m – количество ребер графа). Таким образом, итоговая асимптотика для наивной реализации будет равна $O(n^2 + m)$. Для плотных графов $m \approx n^2$ данная асимптотика является оптимальной.

Построим зависимость времени выполнения программы от размера входного графа. Для этого воспользуемся функцией, представленной в листинге 2. Она генерирует случайную таблицу смежности и запускает алгоритм поиска пути на постепенно увеличивающимся участке этой таблицы. Время выполнения в микросекундах записывается в строку, по которой plotly строит график, представленный на рисунке 3.

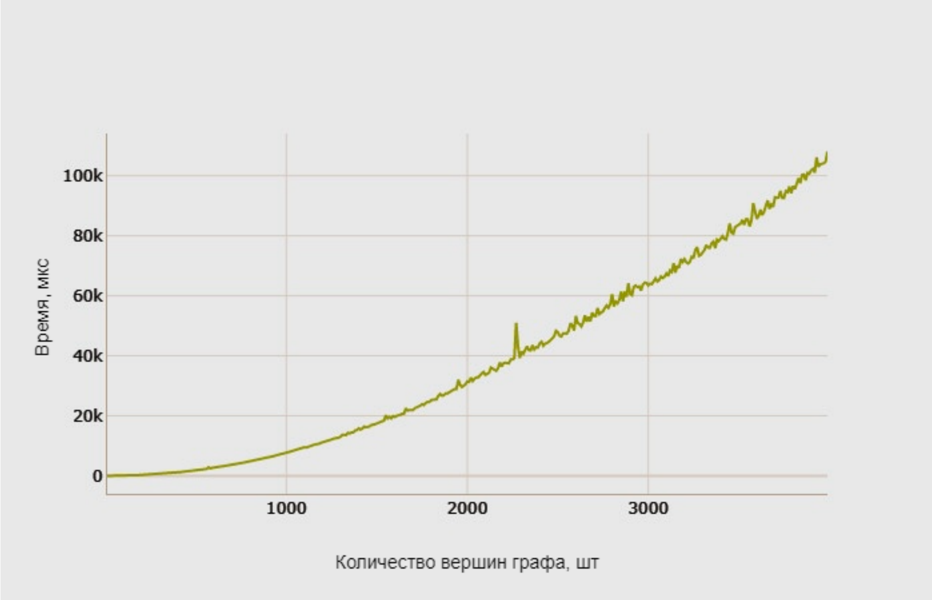

Рис. 3: График зависимости времени выполнения алгоритма от размера входных данных

Листинг 2: Функция для замера времени

```
std::string Measure()
{
    std::string plotData = "{\"kind\":{ \"plotly\": true },\"data\":[{";
    std::string plotSize = "\"x\": [";
    std::string plotTime = "\"y\": [";
    const int maxSize = MAX_SIZE;
    int times[maxSize];

    for (int y = 0; y < MAX_SIZE; y++)
    {
        for (int x = 0; x < MAX_SIZE; x++)
        {
            int prob = std::rand() % 10;
            if (prob < 2 )      // 20
                int weight = std::rand() % 10+1;
                graphTable[x][y] = weight;
                if (prob < 1)       // 10
                    graphTable[y][x] = weight;
                }
            }
        }
    }

    for (int size = 10; size < maxSize; size+=10)
    {
        auto t1 = steady_clock::now();
        Dijkstra(graphTable, 0, size);
        auto t2 = steady_clock::now();
        auto ms_int1 = duration_cast<microseconds>(t2 - t1);

        auto t3 = steady_clock::now();
        Dijkstra(graphTable, size/2, size);
        auto t4 = steady_clock::now();
        auto ms_int2 = duration_cast<microseconds>(t4 - t3);

        auto t5 = steady_clock::now();
        Dijkstra(graphTable, size-1, size);
        auto t6 = steady_clock::now();
        auto ms_int3 = duration_cast<microseconds>(t6 - t5);

        auto ms_int = (ms_int1 + ms_int2 + ms_int3)/3;

        plotTime += std::to_string(ms_int.count()) + ", ";
        plotSize += std::to_string(size) + ", ";
    }
    plotTime = plotTime.substr(0, plotTime.length()-2);
```

```
    plotTime += "],";
    plotSize = plotSize.substr(0, plotSize.length()-2);
    plotSize += "],";
    plotData += plotSize + plotTime + "\"layout\": {\"title\": \"Dijktra's execution time\"}";
    plotData += "}]}";

    return plotData;
```

5 Заключение

Алгоритм Дейкстры широко применяется в программировании и технологиях за счёт своей простоты и надёжности. В частности, с его использованием можно столкнуться при решении задачи навигации в робототехнике, планировании автомобильных маршрутов или в составе других, более сложных алгоритмов. Однако данный алгоритм имеет ограничения в применении, связанные со скоростью работы и невозможностью его использования на графах с отрицательными значениями ребер.

Список литературы

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms Third Edition. – The MIT Press. – 2009. – 658-664 c.

[2] MAXimal :: algo :: Нахождение кратчайших путей от заданной вершины до всех остальных вершин алгоритмом Дейкстры

[3] Использование алгоритмов поиска кратчайшего пути на графах: статья / Н.Г. Аксак, С.А. Партыка, Ю.Ю Завизиступ, 2004.