Санкт-Петербургский политехнический университет Петра Великого
Институт металлургии, машиностроения и транспорта
Высшая школа автоматизации и робототехники

К У Р С О В А Я   Р А Б О Т А

Дисциплина: «Программирование на языках высокого уровня»

«Алгоритм Дейкстры»

Выполнил
студент гр. 3331506/70401 <*подпись*> Я.А. Шкабара

Руководитель
<*подпись*> М.С. Ананьевский

«___» __________ 2020 г.

Санкт-Петербург

2020

Оглавление

Введение

Современное общество характеризуется бурным развитием информационных технологий во всех областях человеческой деятельности. Внедрение этих технологий невозможно без использования компьютерной техники, но с ростом числа источников и потребителей информации, объединенных в вычислительные сети, возникает проблема эффективного обмена информацией внутри сетей.

Для решения данной проблемы используются алгоритмы поиска кратчайшего пути, задачей которых является минимизация стоимости пути к адресату по каким-либо критериям: задержка при передаче пакетов, пропускная способность каналов связи, денежная стоимость передачи по данной линии и т.д.

Одним из таких алгоритмов является алгоритм Дейкстры, используемый протоколами маршрутизации OSPF и IS-IS.

Описание алгоритма

Формулировка задачи: дан взвешенный ориентированный граф *G* (*V*, *E*) [*V* – множество вершин графа; *E* – множество ребер графа] 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*.

Реализация алгоритма на C++

Ниже представлена реализация алгоритма Дейкстры на языке программирования C++:

```
#include <limits.h>
#include <iostream>

constexpr auto number_of_vetrices = 5;

// Функция определяет еще не посещенную вершину u, метка которой минимальна
int minDistance(int distance[], bool vertex_checked[]) {
	int min = INT_MAX;
	int min_index;

	for (int v = 0; v < number_of_vetrices; v++)
		if (vertex_checked[v] == false && distance[v] <= min) {
			min = distance[v];
			min_index = v;
		}
	return min_index;
}

void printSolution(int distance[], int start_vetrix, int p[]) {

	using namespace std;

	cout<<"Вершина       Расстояние      Путь \n";
	for (int i = 0; i < number_of_vetrices; i++) {
		cout <<"   "<< i << "\t\t" << distance[i] << "\t    ";
		int reverse[number_of_vetrices];
		int counter = 0;
		int temp = p[i];
		cout << start_vetrix;

		while (temp != start_vetrix) {
			reverse[counter] = temp;
			counter++;
			temp = p[temp];
		}
		for (counter; counter > 0; counter--)
			cout << "->" << reverse[counter-1];

		cout<<"->"<< i << "\n";
	}
}

void dijkstra(int graph[number_of_vetrices][number_of_vetrices], int start_vetrix) {

	int distance[number_of_vetrices];
	bool vertex_checked[number_of_vetrices];

	for (int i = 0; i < number_of_vetrices; i++) {
		distance[i] = INT_MAX;
		vertex_checked[i] = false;
	}

	distance[start_vetrix] = 0;
	int p[number_of_vetrices];
	p[start_vetrix] = start_vetrix;
```

```
        for (int count = 0; count < number_of_vetrices; count++) {

                int current_vertex = minDistance(distance, vertex_checked);
                vertex_checked[current_vertex] = true;
                for (int v = 0; v < number_of_vetrices; v++)
                        if (!vertex_checked[v] && graph[current_vertex][v] &&
                        distance[current_vertex] != INT_MAX &&
                        distance[current_vertex] + graph[current_vertex][v] < distance[v]) {
                                distance[v] = distance[current_vertex] +graph[current_vertex][v];
                                p[v] = current_vertex;
                        }

        }
        printSolution(distance, start_vetrix, p);
}
```

Пример работы программы представлен на рисунке 1 слева (начальная вершина – 2). Граф, для которого выполнялись вычисления, изображен на рисунке 1 справа.

Вершина	Расстояние	Путь
0	8	2->0
1	11	2->0->1
2	0	2->2
3	1	2->3
4	3	2->3->5->4
5	2	2->3->5

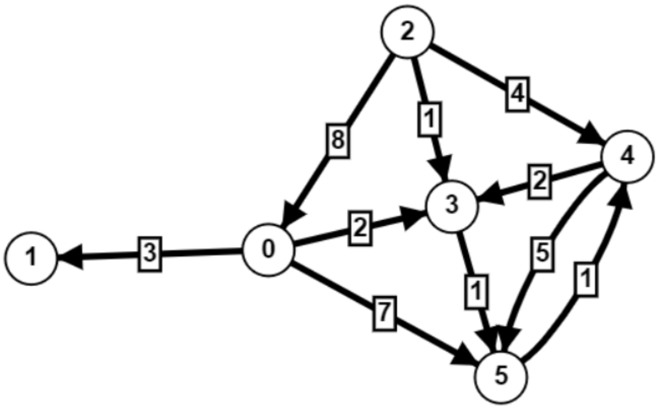

Рисунок 1 – Результат работы программы

Анализ алгоритма

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 используемой мной реализации, когда граф задается матрицей смежности, операция 1 потребует $O(n)$ времени, а операция 2 - $O(1)$ времени (n – количество вершин графа; m – количество ребер графа).

Первая операция выполняется $O(n)$ раз, а вторая $O(m)$ раз. Таким образом итоговая ассимптотика будет равна $O(n^2 + m)$. Слагаемое m можно отбросить в том случае, когда граф сильно разрежен (то есть количество ребер m меньше, чем максимально возможное количество ребер n^2).

График зависимости времени выполнения алгоритма от количества вершин графа изображен на рисунке 2 (время в микросекундах).

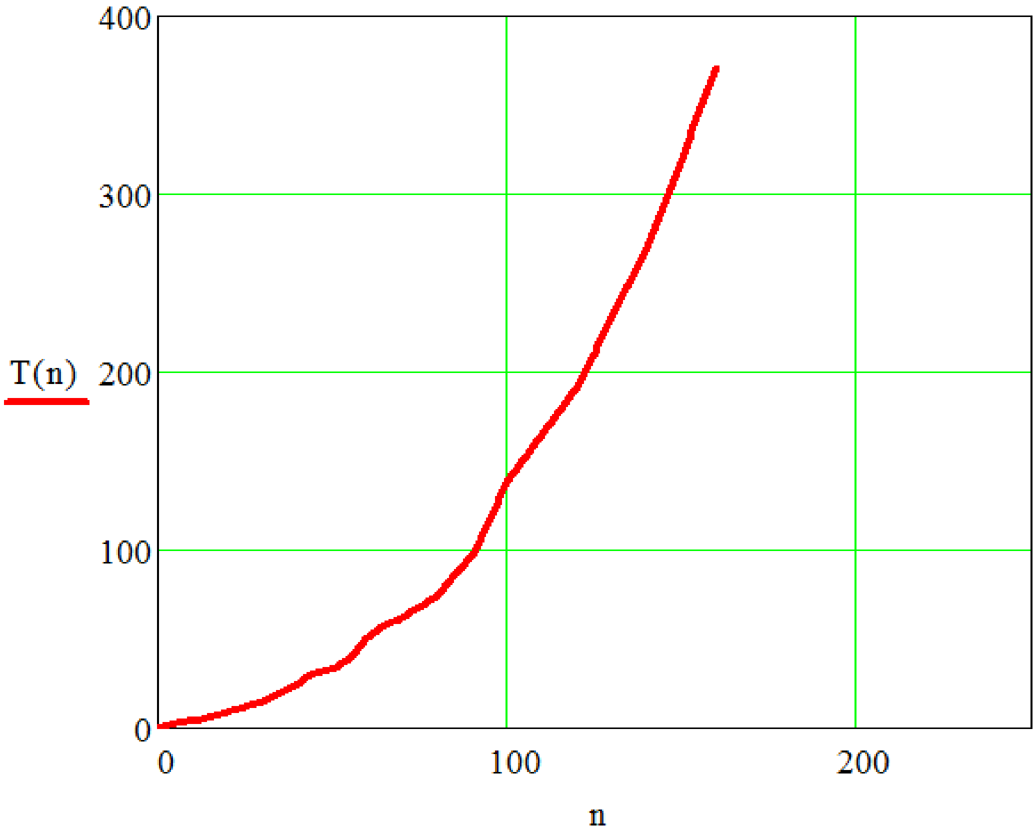

Рисунок 2 – Зависимость времени выполнения алгоритма от количества вершин графа

Заключение

Алгоритм Дейкстры, созданный Э́дсгером Ви́бе Де́йкстрой в 1956 году, широко применяется в программировании и технологиях. В частности, с его использованием можно столкнуться в протоколах маршрутизации, при планировании автомобильных маршрутов, при решении задачи навигации в робототехнике и так далее.

Однако данный алгоритм имеет ограничения, связанные с невозможностью его использования на графах с отрицательными значениями ребер.

Список литературы

1. Алгоритмы: построение и анализ, 3-е изд: Томас Кормен [и др].: Пер. с англ. – М. : ООО «И. Д. Вильямс», 2013. – 1328 с. :ил. – Парал. тит. англ.
2. Использование алгоритмов поиска кратчайшего пути на графах: статья / Н.Г. Аксак, С.А. Партыка, Ю.Ю Завизиступ, 2004.