Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения материалов и транспорта

Высшая школа автоматизации и робототехники

Курсовая работа

**Тема: алгоритм Беллмана-Форда**

Выполнил А.Д. Фиронов

студент гр. 3331506/90401

Руководитель М.С. Ананьевский

« »\_\_\_\_\_\_\_\_2022г.

**Описание алгоритма**

Алгоритм Беллмана-Форда – это алгоритм на графах, решающий задачу нахождения кратчайших путей от одной из вершин графа до всех остальных. В отличие от алгоритма Дейкстры, этот алгоритм корректно работает при наличии в графе рёбер отрицательного веса.

Для каждой вершины графа создаётся переменная – метка. В начале работы алгоритма метка исходной вершины равна нулю, метки всех остальных вершин – бесконечности.

В процессе работы на каждой итерации алгоритм посещает одну из вершин. Среди вершин графа, которые не посещались ранее, выбирается вершина с минимальным значением метки (при первой итерации это всегда начальная вершина). Далее для каждой вершины, имеющей с посещаемой на данной итерации общее ребро и не посещённой ранее, вес общего ребра складывается с весом метки посещаемой вершины. Если вычисленная сумма меньше собственной метки смежной вершины, оно записывается в эту метку.

Итерации повторяются до тех пор, пока все вершины графа не будут посещены.

**Реализация алгоритма**

В ходе работы алгоритм реализован на языке C++.

Для алгоритма Форда-Беллмана, в отличие от многих других графовых алгоритмов, более удобно представлять граф в виде одного списка всех рёбер (а не  списков рёбер — рёбер из каждой вершины). В приведённой реализации заводится структура данных «» для ребра.

Входными данными для алгоритма являются:

-  – количество вершин;

- веса рёбер (нулевой вес принимается за отсутствие связи);

- список рёбер ;

- номер стартовой вершины  по умолчанию принимаемый нулевым.

Далее все вершин нумеруются с 1 по .

Код алгоритма для С++:

#include <cstdlib>  
#include <iostream>  
#include <limits>  
#include <vector>  
using namespace std;  
  
const int inf = numeric\_limits<int>::max(); *// Принимаем за "бесконечность" наибольшее число int*struct edge {  
 int a, b, cost; *//a - номер предыдущей вершины, b - номер новой вершины, cost - вес ребра*};  
  
  
void solve(const vector<edge> &e, int n, int start\_value)  
{  
 vector<int> d(n, inf);  
 d[start\_value] = 0; *// вес ребра для нулевой вершины - 0* for(;;)  
 {  
 bool any = false;  
 for (int k = 0; k < e.size(); ++k)  
 if (d[e[k].a] < inf) *//Проверка для рёбер отрицательного веса* if (d[e[k].b] > d[e[k].a] + e[k].cost)  
 {  
 d[e[k].b] = d[e[k].a] + e[k].cost;  
 any = true;  
*//====================БЛОК ВЫВОДА=========================* for (int i = 1; i < n; ++i)  
 {  
 if (d[i] != inf) cout << i << ": " << d[i] << endl;  
 else cout << i << ": NO" << endl;  
 }  
 cout << endl << endl;  
*//========================================================* }  
 if (!any) break;  
 }  
}  
  
int main()  
{  
 edge temp;  
 vector<edge> e;  
 int n = 0, w = 0, i = 0, j = 0;  
  
 cout<<"Количество вершин > "; cin>>n;  
  
 for (i = 0; i < n; i++)  
 for (j = 1; j<n; j++)  
 {  
 if (i < j) {  
 cout << "Вес " << i << " -> " << j << " > ";  
 cin >> w;  
 if (w != 0) { *// нулевой вес = отсутствие связи* temp.a = i;  
 temp.b = j;  
 temp.cost = w;  
 }  
 }  
 e.push\_back(temp);  
 }  
  
 solve(e, n, 0);  
  
  
 system("pause");  
}

Рассмотрим результат работы программы на примере графа, представленного на рисунке 1.

-1

4

6

-2

8

Рисунок 1 – Рассматриваемый граф

Результат работы программы и входные данные представлены на рисунке 2.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Результат работы программы

На выходе программа выдаёт правильные данные: кратчайший путь до соответствующей вершины.

Также показаны промежуточные итерации выполнения программы: на первых 3 шагах (напомню, в рассматриваемом случае n = 3) происходит заполнение весов для прямого попадания в вершины. На 4 шаге происходит опрос оставшихся вершин из вершины с наименьшим весом (в нашем случае - 1). 5 шаг – аналогичен шагу 4, но теперь из новой вершины с наименьшим весом из оставшихся – 2.

**Анализ алгоритма**

Основной трудностью при реализации алгоритма является наличие отрицательного цикла: при наличии ребра с отрицательным весом классическое исполнение этого алгоритма, с циклом с постусловием, уходит в рекурсию (зачастую этот алгоритм используется как раз для поиска отрицательных циклов в графе). В связи с этим приходится вводить дополнительное ограничение на число итераций: их число должно быть , где m – число рёбер.

Временная сложность алгоритма –

В таблице 1 представлена зависимость времени выполнения алгоритма от количества вершин (для простоты, считаем все вершины соединёнными между собой), а на рисунке 3 – соответствующий график этой зависимости.

Таблица 1 – Зависимость времени выполнения t от числа вершин n

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | 2 | 5 | 10 | 18 | 24 | 30 |
| t, мс | 235 | 1 280 | 4 068 | 15 147 | 26 691 | 42 084 |

t, мс

n

Рисунок 3 – График зависимости времени выполнения от ко-ва вершин

**Применение алгоритма**

Алгоритм может применяться в задачах построение кратчайшего маршрута, а также для поиска наличия отрицательных циклов в графе. Для реализации последнего – необходимо увеличить количество выполняемых итераций с до . В случае изменения выходных параметров на последнем шаге, можно утверждать о наличии отрицательного цикла.

**Список литературы**

1. Электронный ресурс: <http://comp-science.narod.ru/KPG/Deikstr.htm>
2. Электронный ресурс: <http://comp-science.narod.ru/KPG/BelmanFord.htm>
3. Электронный ресурс: <http://e-maxx.ru/algo/negative_cycle>
4. Электронный ресурс: <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0_%E2%80%94_%D0%A4%D0%BE%D1%80%D0%B4%D0%B0>
5. Электронный ресурс: <https://www.cyberforum.ru/cpp-beginners/thread2435646.html>