Санкт-Петербургский политехнический университет Петра Великого

Институт металлургии, машиностроения и транспорта

Кафедра робототехники и роботостроения при ЦНИИ РТК

Курсовая работа

**Тема: алгоритм Дейкстры**

Выполнил Халявин Н.А.

студент гр. 33335/2

Руководитель Ананьевский М. С.

« »\_\_\_\_\_\_\_\_2018г.

Описание алгоритма

Алгоритм Дейкстры – это алгоритм на графах, решающий задачу нахождения самого кратчайших путей от одной из вершин графа до всех остальных. Работает только если в графе нет 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 (на 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 этого заведены следующие классы:

1. Класс List – хранит в себе список вершин графа, соединённые с вершиной, к которой относится экземпляр класса, и веса соответствующих рёбер (так как связи между парой вершин хранятся в независимых экземплярах класса List, рёбра могут иметь разный вес в разных направлениях или вообще быть односторонними).
2. Класс Point – экземпляр данного класса соответствует одной вершине графа. Содержит метку, номер вершины при движении из которой эта метка получена, а также экземпляр класса List со списком выходящих из вершины рёбер. Данный класс содержит в себе все необходимые методы для исполнения алгоритма Дейкстры и вывода результатов.

Код класса List:

**class** List { *//хранит список рёбер, выходящих из одной вершины графа***private**:  
 **int** listSize;  
**public**:  
 **int** \* dataNames;  
 **int** \* dataLength;  
  
 List (**int** size): listSize(size) { *//конструктор* dataNames = **new int**[size];  
 dataLength = **new int**[size];  
 }  
  
 List (List& Arg){ *//конструктор крпий* listSize = Arg.listSize;  
 dataNames = **new int**[listSize];  
 memcpy(dataNames, Arg.dataNames,listSize\***sizeof**(**int**));  
 dataLength = **new int**[listSize];  
 memcpy(dataLength, Arg.dataLength,listSize\***sizeof**(**int**));  
 }  
  
 List (){ *//конструктор пустого экземпляра класса для объявления по умолчанию* dataNames = (**int**\*)0; *//используется при объявлени массива экземпляров класса point* dataLength = (**int**\*)0;  
 }  
  
 **int** getSize(){  
 **return** listSize;  
 }  
  
 **void operator** = (List& Arg){ *//оператор присвоения для переписывания пустого экземпляра класса* listSize = Arg.listSize;  
 **if**(dataNames != (**int**\*)0)  
 **delete** dataNames;  
 dataNames = **new int**[listSize];  
 memcpy(dataNames, Arg.dataNames,listSize\***sizeof**(**int**));  
 **if**(dataLength != (**int**\*)0)  
 **delete** dataLength;  
 dataLength = **new int**[listSize];  
 memcpy(dataLength, Arg.dataLength,listSize\***sizeof**(**int**));  
 }  
  
 ~List(){ *//деструктор* **if**(dataNames != (**int**\*)0)  
 **delete** dataNames;  
 **if**(dataLength != (**int**\*)0)  
 **delete** dataLength;  
 }  
};

Код класса point:

**class** point { *//класс, экземпляр которого соответствует вершине графа***private**:  
 **bool** isNotVisited;  
 **int** label;  
 **int** way; *//номер вершины, путь из которой можно добраться с весом label* List ConnectTo;  
  
 **bool** areNotAllVisited(point \* Graph, **int** size){ *//метод, проверяющий условие завершения работы алгоритма - посещение всех вершин* **bool** NotAllVisits = 0;  
 **for**(**int** i = 0; i < size; i++){  
 NotAllVisits |= Graph[i].isNotVisited;  
 }  
 **return** NotAllVisits;  
 }  
  
 **int** getMinNumber(point \* Graph, **int** size){ *//метод, возвращающий номер вершины, которая должна быть посещена следующей* **int** min = **INT\_MAX**;  
 **int** out = 0;  
 **for**(**int** i = 0; i < size; i++){  
 **if**(Graph[i].isNotVisited)  
 **if**(Graph[i].label < min){  
 out = i;  
 min = Graph[i].label;  
 }  
 }  
 **return** out;  
 }  
  
 **void** Visit(point \* Graph, **int** thisName) { *//метод, осуществляющий посещение вершины* **for** (**int** i = 0; i < ConnectTo.getSize(); i++) {  
 **int** temp = ConnectTo.dataNames[i];  
 **if** (Graph[temp].isNotVisited) {  
 **if** (ConnectTo.dataLength[i] + label < Graph[ConnectTo.dataNames[i]].label) {  
 Graph[ConnectTo.dataNames[i]].label = ConnectTo.dataLength[i] + label;  
 Graph[ConnectTo.dataNames[i]].way = thisName;  
 }  
 }  
 }  
 isNotVisited = 0;  
 }  
  
**public**:  
 point(){ *//пустой конструктор для объявления в массиве* }  
  
 point(point& Arg){ *//конструктор копий* isNotVisited = Arg.isNotVisited;  
 label = Arg.label;  
 way = Arg.way;  
 ConnectTo = Arg.ConnectTo;  
 }  
  
 **void operator** = (point& Arg) { *//оператор присвоения* isNotVisited = Arg.isNotVisited;  
 label = Arg.label;  
 way = Arg.way;  
 ConnectTo = Arg.ConnectTo;  
 }  
  
 point(List& connection): ConnectTo(connection) { *//конструктор по сформированному списку подключений* label = **INT\_MAX**;  
 way = 0;  
 isNotVisited = 1;  
 }  
  
 **void** printResults(point \* Graph, **int** size){ *//вывод минимального веса пути в каждую вершину* **for**(**int** i = 0; i < size; i++){  
 printf(**"%d\t%d\t%d\n"**, i, Graph[i].label,Graph[i].way);  
 }  
 }  
  
 **void** printWayTo(point \* Graph, **int** TargetName, **int** StartName){ *//вывод кратчайшего пути в заданную вершину* **int** pointer = TargetName;  
 printf(**"//////////\n"**);  
 **while**(pointer != StartName){  
 printf(**"%d\t%d\n"**, pointer, Graph[pointer].label);  
 pointer = Graph[pointer].way;  
 }  
 printf(**"%d\t%d\n"**, pointer, Graph[pointer].label);  
 }  
  
 **void** DeicstraAlg(point \* Graph, **int** size, **int** startNumber){ *//функция, исполняющая алгоритм Дейкстры* Graph[startNumber].label = 0;  
 **while**(Graph[0].areNotAllVisited(Graph,6)){  
 **int** thisPoint = Graph[0].getMinNumber(Graph,6);  
 **if**(thisPoint > size) **return**;  
 Graph[thisPoint].Visit(Graph, thisPoint);  
 }  
 }  
};

Пример использования классов:

**int** main() {  
 point Graph[6]; *//граф объявляется как массив неинициализированных вершин* List input0(3); *//создание и инициализация списка нулевой вершины* input0.dataNames[0] = 1;  
 input0.dataLength[0] = 7;  
 input0.dataNames[1] = 2;  
 input0.dataLength[1] = 9;  
 input0.dataNames[2] = 5;  
 input0.dataLength[2] = 14;  
 point p0(input0); *//создание инициализированной вершины* Graph[0] = p0; *//копирование вершины в нулевую вершину массива* List input1(3); *//аналогично - инициализация остальных вершин* input1.dataNames[0] = 3;  
 input1.dataLength[0] = 15;  
 input1.dataNames[1] = 2;  
 input1.dataLength[1] = 10;  
 input1.dataNames[2] = 0;  
 input1.dataLength[2] = 7;  
 point p1(input1);  
 Graph[1] = p1;  
  
 List input2(4);  
 input2.dataNames[0] = 0;  
 input2.dataLength[0] = 9;  
 input2.dataNames[1] = 1;  
 input2.dataLength[1] = 10;  
 input2.dataNames[2] = 3;  
 input2.dataLength[2] = 11;  
 input2.dataNames[3] = 5;  
 input2.dataLength[3] = 2;  
 point p2(input2);  
 Graph[2] = p2;  
  
 List input3(3);  
 input3.dataNames[0] = 1;  
 input3.dataLength[0] = 15;  
 input3.dataNames[1] = 2;  
 input3.dataLength[1] = 11;  
 input3.dataNames[2] = 4;  
 input3.dataLength[2] = 6;  
 point p3(input3);  
 Graph[3] = p3;  
  
 List input4(2);  
 input4.dataNames[0] = 3;  
 input4.dataLength[0] = 6;  
 input4.dataNames[1] = 5;  
 input4.dataLength[1] = 9;  
 point p4(input4);  
 Graph[4] = p4;  
  
 List input5(3);  
 input5.dataNames[0] = 0;  
 input5.dataLength[0] = 14;  
 input5.dataNames[1] = 2;  
 input5.dataLength[1] = 2;  
 input5.dataNames[2] = 4;  
 input5.dataLength[2] = 9;  
 point p5(input5);  
 Graph[5] = p5;  
  
  
 Graph[0].DeicstraAlg(Graph, 6, 0); *//выполнение алгоритма Дейкстры* Graph[0].printResults(Graph, 6); *//вывод длины кратчайшего пути для всех вершин* Graph[0].printWayTo(Graph, 4, 0); *//вывод пути в 4 вершину из 0* **return** 0;  
}

Введённый в примере граф представлен на рисунке 1.

0

1

5

3

2

4

7

15

10

11

6

14

2

9

9

Рисунок 1 – Введённый граф

Результат работы программы с пояснениями представлен на рисунке 2.

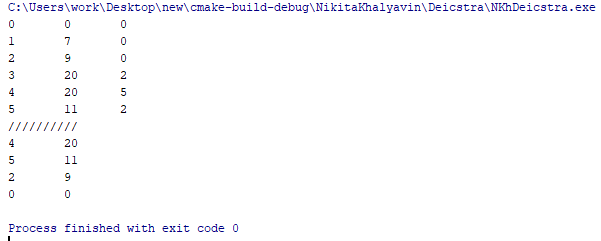


Рисунок 2 – Результат работы программы

Вывод функции printWayTo, полное отображение кратчайшего пути в заданную вершину. Первый столбец – номер вершины, второй – длина уже пройденного к этой вершине пути.

Вывод функции printResults, первый столбец – номер вершины, второй – длина кратчайшего пути, третий – номер предпоследней вершины этого пути

Анализ алгоритма

Пусть n – число вершин графа, а m – число рёбер. В ходе работы алгоритм последовательно проходит все вершины по одному разу, то есть основной цикл выполняется n раз. На каждой итерации алгоритм проверяет, все ли вершины посещены, после чего ищет непосещённую вершину с минимальной меткой, в обоих случаях проходя последовательно все вершины. Далее для найденной вершины с минимальной меткой алгоритм проходит каждое ребро. Так как ребра соединены с двумя вершинами, каждое ребро проходится дважды, поэтому всего алгоритм производит проход по ребру 2m раз. Таким образом, время работы алгоритма равно k1∙n2 + k2∙m, а значит после пренебрежения константами временная сложность – O(n2 + m).

Применение алгоритма

Алгоритм может применяться в задачах оптимизации: решение задачи коммивояжёра, построение кратчайшего маршрута, маршрутизация каналов связи.