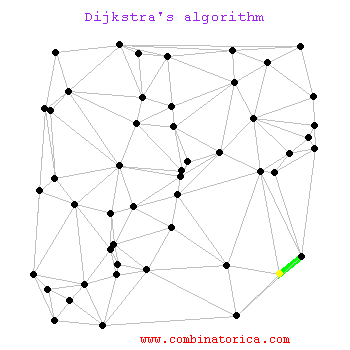
Алгоритм Dextra для определения кратчайшего пути является одним из самых популярных алгоритмов, которые используются для этой цели, и он имеет важные приложения, которые мы используем ежедневно.

Данные, которые передаются через Интернет, проходят разными путями по всему миру, поэтому важно, чтобы они направлялись короткими и менее загруженными путями, и это возможно с помощью алгоритма Dextra.

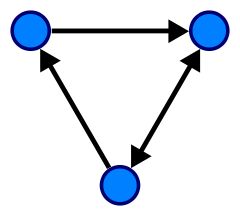
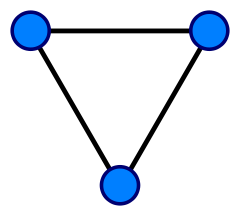
 Ибрахем Навар 09-934 2 курсе

Например, OSPF - это сокращение от Open Shortest Path First, которое используется в Интернет-протоколе (IP).

Его можно было использовать в качестве карт, которые могли определять карты, которые могли быть определены нами, которые были для нас короче, и мы могли пройти его, мы могли пройти

Алгоритм очень простой и удобный для рисования и рисования (теория графов).

Теория графов или теория графов - это теория в математике и информатике, которая изучает свойства диаграмм, в которых группа объектов, называемых вершинами или точками, представлена ​​(отображается синим цветом), связанных друг с другом многоугольниками и иногда называемыми дугами, которые могут быть ориентированы, т. Е. Имеют направление ( Вместо краев используются стрелки) или без направления (только ребра)

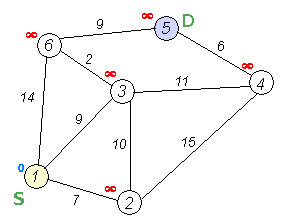
 

То, что алгоритм изначально получает график с указанными весами (расстояниями), означает, что мы определяем набор точек и дуг или длины дуг, а веса должны быть неотрицательными и определять начальную точку Но с какой схемой работает алгоритм?

Оператор (граф) представляет собой упорядоченную пару (G = (V, E), включающую группу {..., V {a, b, c, d) заголовков (станций или адресов станций), работающих на {..., {E {E c}, {b, d}, {a, d видимых сторон - это, в свою очередь, набор неупорядоченных частичных двоичных файлов V (например, двоичный {a, c} означает сторону между точками a и c). Этот тип данных определяет простой простой вектор.

При применении алгоритма мы будем знать, какое кратчайшее расстояние между исходной точкой и №74, другими точками

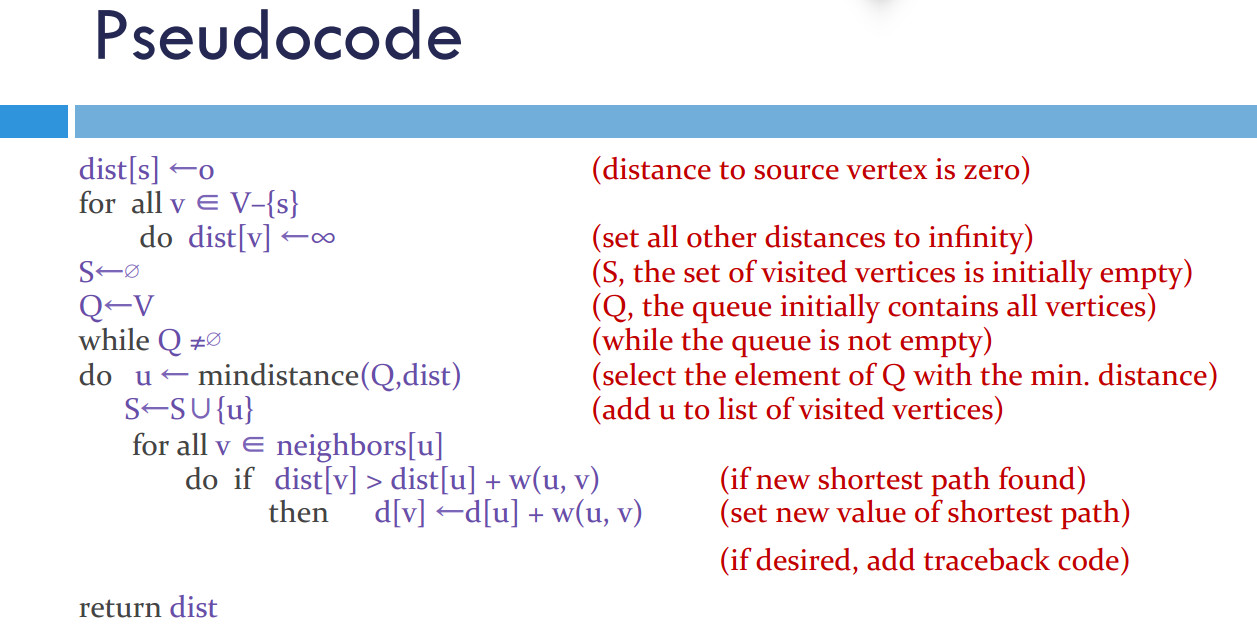
И после нескольких эпизодов на Алгоритм.



Теперь у нас есть граф (G = (V, E), этот граф взвешен, то есть каждая дуга или ребро с заданным весом. Этот вес может фактически быть длиной расстояния или количеством скученности или любым другим подходящим представлением в зависимости от проблемы.

И мы определили исходную точку s, которая является точкой, как и любая другая точка, принадлежащая набору точек V, определенному графиком G.

Алгоритм:



Матрица dist будет содержать значения расстояния для каждой точки. Например, [dist [v1] означает расстояние между v1 и s, которое является исходной точкой, которая будет действовать как начало координат.

Когда алгоритм реализуется, значения этой матрицы будут кратчайшим расстоянием от каждой точки от начала координат.

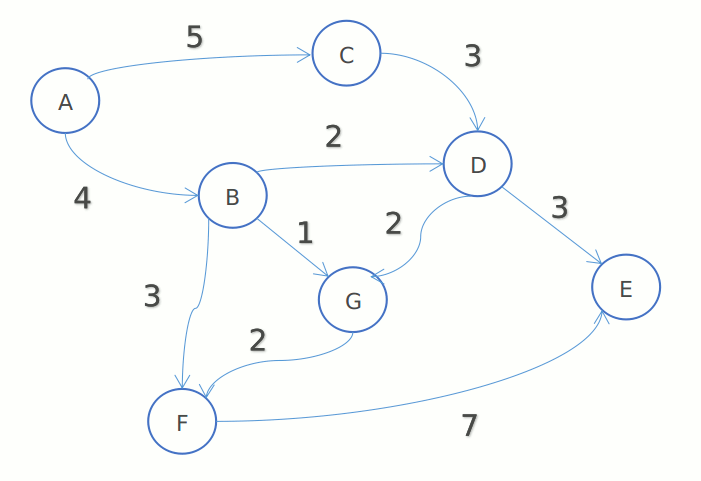
|  |  |
| --- | --- |
| Присвойте нулевое значение расстоянию между точкой s и исходной точкой (поскольку точка s совпадает с исходной точкой, и ее расстояние от себя будет равно нулю) | dist[s]←0 |
| Наведите указатель мыши на каждую v-точку в наборе точек, определенных на ранее данной диаграмме G = (V, E( | for all v ∈ V–{s} |
| Дайте бесконечное значение (очень высокое значение) (баллы) | do dist[v]←∞ |
| Присвойте группе S по умолчанию нулевое значение. Группа S - это магазин, в котором мы будем размещать посещенные точки (адреса или станции) | S←∅ |
| Присвойте Группе Q все точки, определенные в Таблице G (все адреса на карте) | Q←V |
| Пока группа Q не пуста (т.е. все еще содержит заголовки) | while Q ≠ ∅ |
| Выберите точку u из множества Q так, чтобы она находилась на наименьшем расстоянии от всех существующих точек, и удалите ее из множества Q  (Поскольку всем точкам присвоено бесконечное значение, за исключением точки s, которой было присвоено нулевое значение, на первом шаге будет выбрано s) | do u←mindistance(Q,dist) |
| Добавить точку u в группу S. | S←S ∪ {u} |
| Наведите указатель мыши на каждую точку рядом с (смежной) точкой u | for all v ∈ neighbors[u] |
| Если расстояние до точки v больше, чем расстояние до точки u + длина стороны (u, v) | do if dist[v] > dist[u] +w(u,v) |
| Сделайте расстояние до точки v = расстояние до точки u + длина стороны (u, v) | then d[v]←d[u]+w(u,v) |
| Вернуть массив расстояний  (Ибрахем Навар 09-934 2 курсе ) | return dist |

Теперь у нас есть такая схема (G = (V, E

Он содержит точечное множество V

V = {A, B, C, D, E, F, G}

А в скобках ставим E

E = {{A, C}, {A, B}, {C, B}, {B, G}, {G, F}, {F, E}, {B, F}, {D, E} , {D, G}, {C, D}}

Предположим, что исходная точка s - это точка A.

s = A

После выполнения шагов

dist [s] ← 0

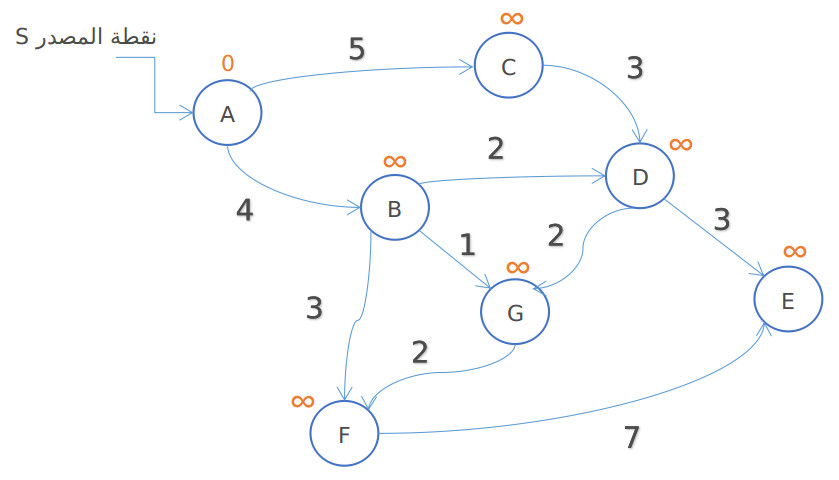
для всех v ∈ V– {s}

сделать dist [v] ← ∞

Обратите внимание, что цикл будет передавать все точки в наборе V, кроме точки A, потому что это исходная точка s

V– {s} = {B, C, D, E, F, G}

И становится матрицей расстояний

dist [A] = 0, dist [B-F] = ∞

После выполнения шагов

S ← ∅

Q ← V

У нас будет набор S, который мы будем использовать для хранения посещенных точек по умолчанию. Это пустой набор, не содержащий никаких точек.

И как группа Q, мы разместим все точки, определенные в G-диаграмме.

s = {∅}

Q = {A, B, C, D, E, F, G}

Следующим шагом является итеративный цикл, который будет повторять все следующие шаги, пока блок Q не пуст.

а Q ≠ ∅

При выполнении первого цикла while

do u ← mindistance (Q, расстояние)

S ← S ∪ {u}

Выберите точку с расстоянием желтого цвета между точками группы Q

Наименьшее расстояние - это расстояние до исходной точки A, потому что мы присвоили ей нулевое значение на первом шаге dist [A] ← 0

Так станет

u = A

S = {A}

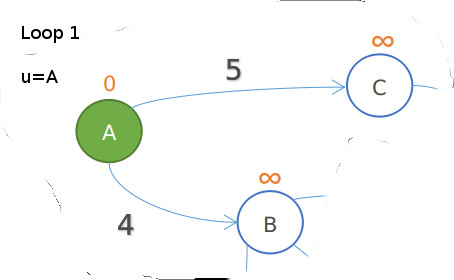
Затем удаляем точку u из множества Q

Поскольку u = A, множество Q становится

Q = {B, C, D, E, F, G}

Обратите внимание, что объединение пустой группы с другой группой равно той же группе

{∅} ∪ {A} = {A}



для всех v ∈ соседей [u]

Цикл for будет проходить через всех соседей точки u

Мы знаем u = A из предыдущего шага

Соседями точки A являются C и B.

Это означает, что когда выполняется первый цикл for, v = B, а второй v = C.

Мы проверяем это условие при выполнении первого цикла и v = B

делать, если dist [v]> dist [u] + w (u, v)

Когда v = B и u = A, вес дуги будет 4 = (w (A, B)

Следовательно

dist [B]> dist [A] + w (A, B) = истина

Потому как

∞> 0 + 4

тогда d [v] ← d [u] + w (u, v)

если

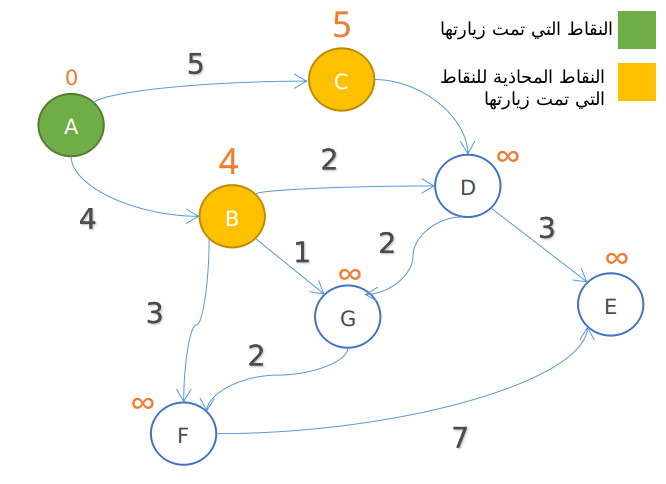
d [B] = 0 + 4;

При выполнении второго цикла for это будет v = C.

∞> 0 + 5

если

d [C] = 0 + 5;( Ибрахем Навар 09-934 2 курсе )

Когда внутренний цикл for завершен

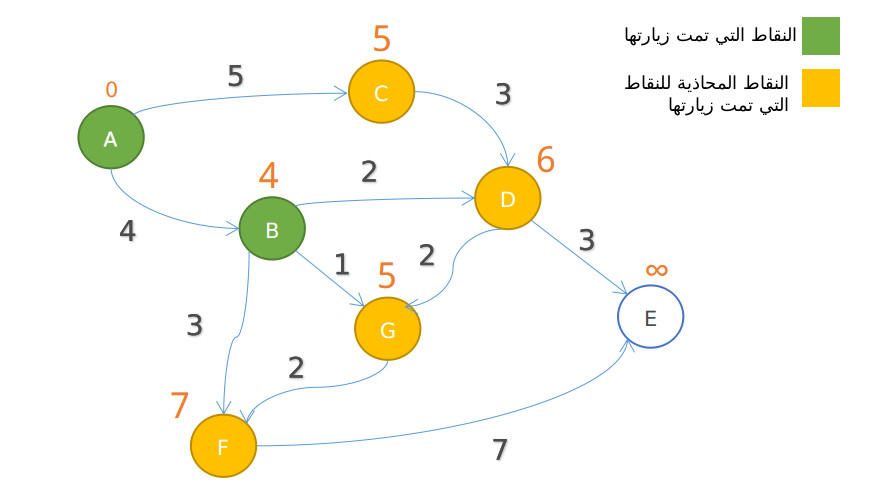
Он становится массивом расстояний

d [A] = 0, d [B] = 5, d [C] = 4, d [D-F] = ∞

И при выполнении второго цикла while

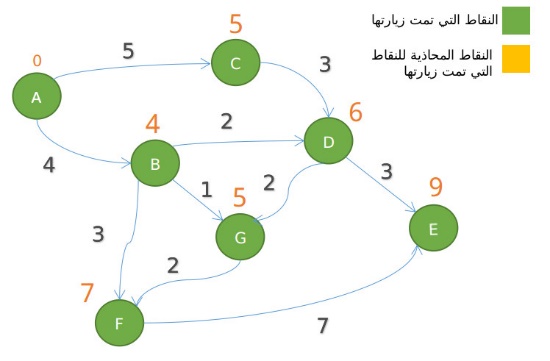
Точка B будет иметь наименьшее расстояние между точками в группе Q, которая равна {B, C, D, E, F, G}.

Помните, что мы удалили точку A из Q, потому что она была выбрана раньше.

Схема будет выглядеть так

return dist

Когда все точки будут посещены, мы выйдем из цикла while и вернем массив расстояний



Мы знали расстояние до каждой точки или станции, но мы все еще не знаем подстанции, которые мы проедем, чтобы достичь определенной точки с кратчайшим возможным расстоянием!

Собственно, исходный алгоритм Dextra не дает подпути, но мы можем простыми модификациями получить путь к любой точке.

The program :

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Алгоритм\_дейкстры

{

//Класс алгоритма поиска путей

public class Algorithm

{

private int[,] matrix; //Матрица смежности

private Graph graph; //Исходный граф

public Algorithm(Graph \_graph)

{

graph = \_graph;

int matrixSize = graph.Vertexes.Count;

matrix = new int[matrixSize, matrixSize];

//Проходимся по ребрам и выставляем веса соответственно ребрам (остальные равны 0)

foreach (Edge edge in graph.Edges)

{

matrix[edge.StartVertex.Id - 1, edge.EndVertex.Id - 1] = edge.Cost;

if (graph.GraphType == GraphType.NonOrientied)

matrix[edge.EndVertex.Id - 1, edge.StartVertex.Id - 1] = edge.Cost;

}

}

public List<int> ShortestWaysFromBeginVertexToAllVertexes(int BeginVertex)

{

int vertexesCount = graph.Vertexes.Count;

int[] shortestWays = new int[vertexesCount]; //Кратчайшие пути

int[] visitedVertexes = new int[vertexesCount]; //Посещенные вершины

//Инициализация вершин и расстояний

for (int i = 0; i < vertexesCount; i++)

{

shortestWays[i] = int.MaxValue;

visitedVertexes[i] = 1;

}

shortestWays[BeginVertex] = 0;

int minIndex = int.MaxValue;

do

{

minIndex = int.MaxValue;

int min = int.MaxValue;

for (int i = 0; i < vertexesCount; i++)

{

// Если вершину ещё не обошли

if (visitedVertexes[i] == 1 && shortestWays[i] < min)

{

min = shortestWays[i];

minIndex = i;

}

}

// Добавляем найденный минимальный вес

// к текущему весу вершины

// и сравниваем с текущим минимальным весом вершины

if (minIndex != int.MaxValue)

{

for (int i = 0; i < vertexesCount; i++)

{

if (matrix[minIndex, i] > 0)

{

int temp = min + matrix[minIndex, i];

if (temp < shortestWays[i])

shortestWays[i] = temp;

}

}

visitedVertexes[minIndex] = 0;

}

} while (minIndex < int.MaxValue);

return shortestWays.ToList();

}

// Восстановление пути

//Ибрахем Навар 09-934 2 курсе

public List<int> ShortestWaysFromBeginVertexToEndVertex(int[] shortestWays, int beginVertex, int endVertex)

{

int vertexesCount = graph.Vertexes.Count;

int[] visitedVertexes = new int[vertexesCount]; // массив посещенных вершин

visitedVertexes[0] = endVertex + 1; // начальный элемент - конечная вершина

int indexOfPreviousVertex = 1; // индекс предыдущей вершины

int weigth = shortestWays[endVertex]; // вес конечной вершины

while (endVertex != beginVertex) // пока не дошли до начальной вершины

{

for (int i = 0; i < vertexesCount; i++) // просматриваем все вершины

if (matrix[i, endVertex] != 0) // если связь есть

{

int temp = weigth - matrix[i, endVertex]; // определяем вес пути из предыдущей вершины

if (temp == shortestWays[i]) // если вес совпал с рассчитанным значит из этой вершины и был переход

{

weigth = temp;

endVertex = i;

visitedVertexes[indexOfPreviousVertex] = i + 1;

indexOfPreviousVertex++;

}

}

}

var result = visitedVertexes.ToList();

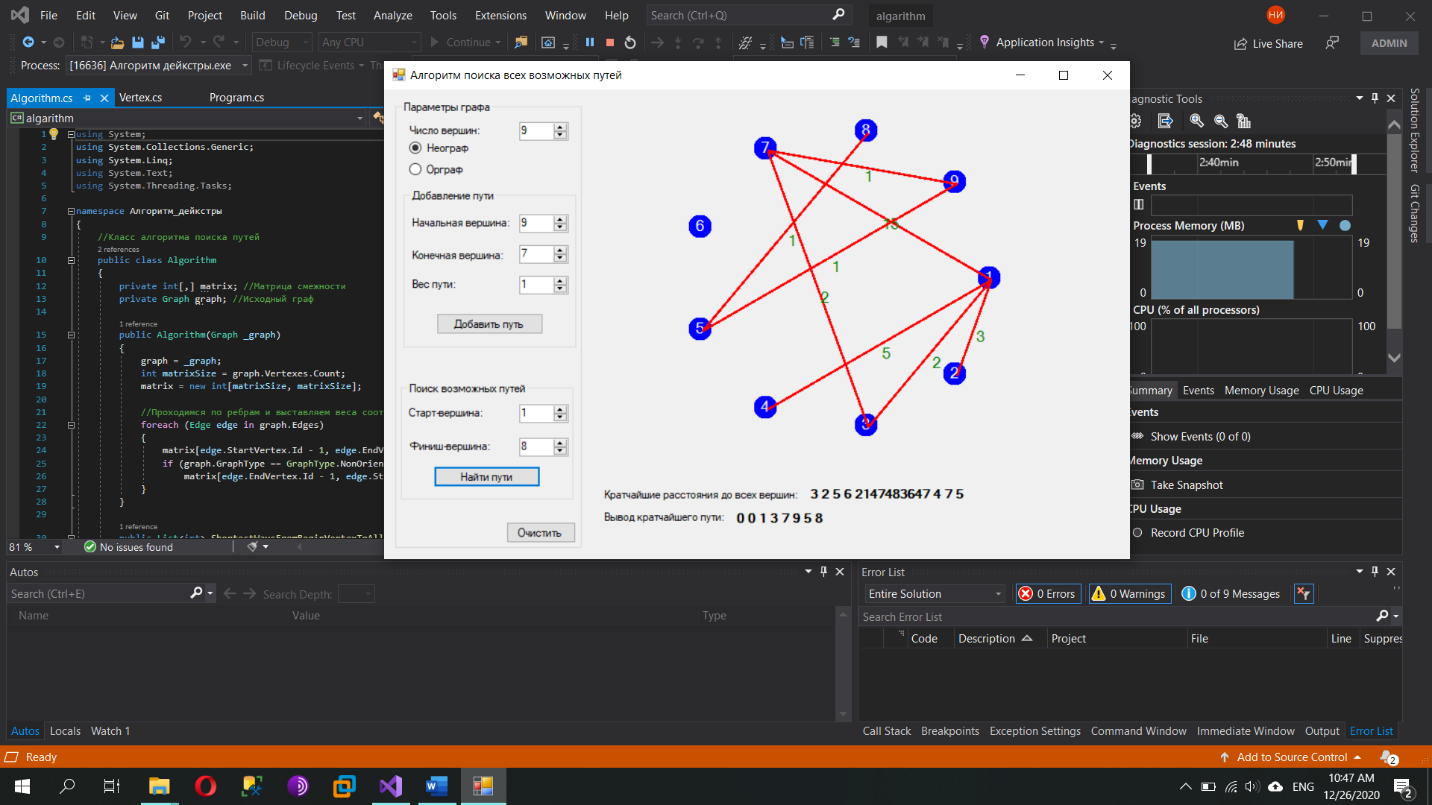
result.Reverse();

return result;

}

}

}



Ibrahem Nawar 09-934 2year

Ибрахем Навар 09-934 2 курсе