**РОСЖЕЛДОР**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ» (СГУПС)**

Кафедра «Информационные технологии транспорта»

«Игра водопроводчик»

Курсовой проект

по дисциплине «Объектно-ориентированное программирование»

Пояснительная записка

|  |  |  |
| --- | --- | --- |
| Проверил: |  | Выполнил: |
| Предподователь:  ­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_Жуков М.В |  | Студент гр. БИСТ-211  \_\_\_\_\_\_\_\_Боровик М.А |
|  |  |  |
| \_\_\_\_\_\_\_\_\_\_\_\_\_ |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *(дата проверки)* |  | *(дата сдачи на проверку)* |
|  |  |  |
|  |  |  |
|  |  |  |

Краткая рецензия:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(запись о допуске к защите)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(оценка по результатам защиты) (подпись преподавателя)*

Новосибирск, 2015

СОДЕРЖАНИЕ

АННОТАЦИЯ..............................................................................................................3

ВВЕДЕНИЕ.................................................................................................................4

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ....................................................................................5

1.1. Транспортная задача………......................................................................5

1.2. Алгоритм Флойда-Уоршелла....................................................................6

1.3. Алгоритм Форда-Беллмана........................................................................8

1.4. Алгоритм Дейкстра..................................................................................10

2. ПРАКТИЧЕСКАЯ ЧАСТЬ...................................................................................12

2.1. Граф. Расчет весов....................................................................................12

2.2. Критерии выбора коэффициента............................................................13

2.3. Коды алгоритмов......................................................................................15

2.4. Основные используемые программные модули....................................18

2.5. Визуальное оформление приложения. Возможности...........................20

3. ТЕСТИРОВАНИЕ ПРОГРАММЫ......................................................................21

ВЫВОД.......................................................................................................................23

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ..................................................24

ПРИЛОЖЕНИЕ А – КОД ПРОГРАММЫ..............................................................25

АННОТАЦИЯ

Данная работа посвящена решению транспортной задачи, а именно поиску кратчайшего пути между вершинами графа, с реализацией методов поиска программным приложением. В курсовом проекте представлен поиск путей при помощи трех алгоритмов.

Целью работы было написание алгоритмов; вывод кратчайших путей между вершинами графа для этих алгоритмов; вывод преобразованной матрицы историй и весов для алгоритма Флойда-Уоршелла.

Используемый язык программирования – Object Pascal, среда программирования – Lazarus 1.2.0.

Курсовой проект состоит из 32 страниц, 3 частей, 7 рисунков, 1 приложения.

ВВЕДЕНИЕ

Поиск кратчайшего пути от одной точки графа до другой можно произвести несколькими способами. Существует три базовых алгоритма для нахождения кратчайшего пути между вершинами графа. В данном курсовом проекте рассмотрены алгоритмы: Флойда-Уоршелла, Форда-Беллмана и Дейкстра. Каждый алгоритм имеет свой уровень сложности, а, следовательно, и свою реализацию. Разработать алгоритмы, а также их протестировать предлагалось либо в ручную на бумаге, либо написав соответствующую программу.

Мною была разработана программа для поиска кратчайшего пути в программной среде Lazarus, в виде приложения.

В программе организован вывод начальных матриц весов и историй, необходимая пользователю информацию для выбора алгоритма, а также кратчайший путь между начальной и конечной точками графа, которые соответствуют: остановке ДК Кирова и СГУПС.

Граф был составлен из предположения проживания вблизи остановки в г. Новосибирске, ДК Кирова. Сам граф, а также веса ребер представлены в практической части на рисунке 2.1.

Критерием для выбора значений весов, мною был выбран коэффициент, содержащий в себе сумму стоимости проезда на общественном транспорте и время в пути.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. Транспортная задача

Транспортная задача – математическая задача линейного программирования специального вида о поиске оптимального распределения однородных объектов из аккумулятора к приемникам с минимизацией затрат на перемещение. Для простоты понимания в данном курсовом проекте рассматривается, как задача о поиске оптимального, кратчайшего пути с минимальным весовым критерием между двумя точками.

Для классической транспортной задачи выделяют два типа критериев: критерий стоимости (достижение минимума затрат) или расстояний и критерий времени (затрачивается минимум времени).

Под названием транспортная задача, определяется широкий круг задач с единой математической моделью, эти задачи относятся к задачам линейного программирования и могут быть решены оптимальным методом. Однако, специальный метод решения транспортной задачи позволяет существенно упростить её решение, поскольку транспортная задача разрабатывалась для минимизации стоимости перевозок. Анализируя информацию, представленную по данному термину, можно сделать вывод, что решением транспортных задач, в основном, занимаются при перевозке каких-либо грузов.

Следовательно, в данном курсовом проекте требуется решить транспортную задачу немного иного вида, а именно найти оптимальный кратчайший путь между двумя точками с использованием построения графа, при этом реализуя разные алгоритмы поиска.

1.2. Алгоритм Флойда-Уоршелла

Наиболее часто используемое название, метод получил в честь двух американских исследователей Роберта Флойда и Стивена Уоршелла, одновременно открывших его в 1962 году. Реже встречаются другие варианты наименований: алгоритм Рой – Уоршелла или алгоритм Рой – Флойда. Рой – фамилия профессора, который разработал аналогичный алгоритм на 3 года раньше коллег (в 1959 г.), но это его открытие осталось безвестным. Алгоритм Флойда – Уоршелла – динамический алгоритм вычисления значений кратчайших путей для каждой из вершин графа.

Метод работает на взвешенных графах, с положительными и отрицательными весами ребер, но без отрицательных циклов, являясь, таким образом, более общим в сравнении с алгоритмом Дейкстры, т. к. последний не работает с отрицательными весами ребер, и к тому же классическая его реализация подразумевает определение оптимальных расстояний от одной вершины до всех остальных.

Для реализации алгоритма Флойда – Уоршелла сформируем матрицу смежности D[][] графа G=(V, E), в котором каждая вершина пронумерована от 1 до |V|. Эта матрица имеет размер |V|х|V|, и каждому ее элементу D[i][j] присвоен вес ребра, идущего из вершины i в вершину j. По мере выполнения алгоритма, данная матрица будет перезаписываться: в каждую из ее ячеек внесется значение, определяющее оптимальную длину пути из вершины i в вершину j (отказ от выделения специального массива для этой цели сохранит память и время). Теперь, перед составлением основной части алгоритма, необходимо разобраться с содержанием матрицы кратчайших путей.

Поскольку каждый ее элемент D[i][j] должен содержать наименьший из имеющихся маршрутов, то сразу можно сказать, что для единичной вершины он равен нулю, даже если она имеет петлю (отрицательные циклы не рассматриваются), следовательно, все элементы главной диагонали (D[i][i]) нужно обнулить. А чтобы нулевые недиагональные элементы (матрица смежности могла иметь нули в тех местах, где нет непосредственного ребра между вершинами i и j) сменили по возможности свое значение, определим их равными бесконечности, которая в программе может являться, например, максимально возможной длинной пути в графе, либо просто – большим числом.

Ключевая часть алгоритма, состоит из трех циклов, выражения и условного оператора:

Для k от 1 до |V| выполнять

Для i от 1 до |V| выполнять

Для j от 1 до |V| выполнять

Если D[i][k]+D[k][j]<D[i][j] то D[i][j] :=D[i][k]+D[k][j]

Кратчайший путь из вершины i в вершину j может проходить, как только через них самих, так и через множество других вершин k∈(1, …, |V|). Оптимальным из i в j будет путь или не проходящий через k, или проходящий. Заключить о наличии второго случая, значит установить, что такой путь идет из i до k, а затем из k до j, поэтому должно заменить, значение кратчайшего пути D[i][j] суммой D[i][k]+D[k][j].

1.3. Алгоритм Форда-Беллмана

История алгоритма связана сразу с тремя независимыми математиками: Лестером Фордом, Ричардом Беллманом и Эдвардом Муром. Форд и Беллман опубликовали алгоритм в 1956 и 1958 годах соответственно, а Мур сделал это в 1957 году. И иногда его называют алгоритмом Беллмана – Форда – Мура. Метод используется в некоторых протоколах дистанционно-векторной маршрутизации, например, в RIP (Routing Information Protocol – Протокол маршрутной информации).

Также, как и алгоритм Дейкстры, алгоритм Форда-Беллмана вычисляет во взвешенном графе кратчайшие пути от одной вершины до всех остальных. Он подходит для работы с графами, имеющими ребра с отрицательным весом. Но спектр применимости алгоритма затрагивает не все такие графы, ввиду того, что каждый очередной проход по пути, составленному из ребер, сумма весов которых отрицательна (т. е. по отрицательному циклу), лишь улучшает требуемое значение. Бесконечное число улучшений делает невозможным определение одного конкретного значения, являющегося оптимальным.

В связи с этим алгоритм Форда-Беллмана не применим к графам, имеющим отрицательные циклы, но он позволяет определить наличие таковых, о чем будет сказано позже.

Решить задачу, т. е. найти все кратчайшие пути из вершины s до всех остальных, используя алгоритм Форда-Беллмана, это значит воспользоваться методом динамического программирования: разбить ее на типовые подзадачи, найти решение последним, покончив тем самым с основной задачей. Здесь решением каждой из таких подзадач является определение наилучшего пути от одного отдельно взятого ребра, до какого-либо другого.

Для хранения результатов работы алгоритма заведем одномерный массив d[]. В каждом его i-ом элементе будет храниться значение кратчайшего пути из вершины s до вершины i (если таковое имеется).

Изначально, присвоим элементам массива d[] значения равные условной бесконечности (например, число заведомо большее суммы всех весов), а в элемент d[s] запишем нуль. Так мы задействовали известную и необходимую информацию, а именно известно, что наилучший путь из вершины s в нее же саму равен 0, и необходимо предположить недоступность других вершин из s. По мере выполнения алгоритма, для некоторых из них, это условие окажется ложным, и вычисляться оптимальные стоимости путей до этих вершин из s.

Задан граф G=(V, E), n=|V|, а m=|E|. Обозначим смежные вершины этого графа символами v и u, (а вес ребра (v, u) символом w. Иначе говоря, вес ребра, выходящего из вершины v и входящего в вершину u, будет равен w. Тогда ключевая часть алгоритма Форда-Беллмана примет следующий вид:

Для i от 1 до n-1 выполнять

Для j от 1 до m выполнять

Если d[v] + w(v, u) < d[u] то

d[u] := d[v] + w(v, u)

На каждом n-ом шаге осуществляются попытки улучшить значения элементов массива d[]: если сумма, составленная из веса ребра w(v, u) и веса хранящегося в элементе d[v], меньше веса d[u], то она присваивается последнему.

1.4. Алгоритм Дейкстры

Алгоритм голландского ученого Эдсгера Дейкстры находит все кратчайшие пути из одной изначально заданной вершины графа до всех остальных. С его помощью, при наличии всей необходимой информации, можно, например, узнать какую последовательность дорог лучше использовать, чтобы добраться из одного города до каждого из многих других, или в какие страны выгодней экспортировать нефть и тому подобное.

Минусом данного метода является невозможность обработки графов, в которых имеются ребра с отрицательным весом, т. е. если, например, некоторая система предусматривает убыточные для Вашей фирмы маршруты, то для работы с ней следует воспользоваться отличным от алгоритма Дейкстры методом.

Для программной реализации алгоритма понадобиться два массива: логический visited – для хранения информации о посещенных вершинах и численный distance, в который будут заноситься найденные кратчайшие пути.

Итак, имеется граф G=(V, E). Каждая из вершин входящих во множество V, изначально отмечена как не посещенная, т. е. элементам массива visited присвоено значение false.

Поскольку самые выгодные пути только предстоит найти, в каждый элемент вектора distance записывается такое число, которое заведомо больше любого потенциального пути (обычно это число называют бесконечностью, но в программе используют, например максимальное значение конкретного типа данных). В качестве исходного пункта выбирается вершина s и ей приписывается нулевой путь: distance[s]=0, т. к. нет ребра из s в s (метод не предусматривает петель).

Далее, находятся все соседние вершины (в которые есть ребро из s) [пусть таковыми будут t и u] и поочередно исследуются, а именно вычисляется стоимость маршрута из s поочередно в каждую из них:

distance[t]=distance[s]+весинцидентного s и t ребра;

distance[u]=distance[s]+ весинцидентного s и u ребра.

Но вполне вероятно, что в ту или иную вершину из s существует несколько путей, поэтому цену пути в такую вершину в массиве distance придется пересматривать, тогда наибольшее (неоптимальное) значение игнорируется, а наименьшее ставиться в соответствие вершине.

После обработки смежных с s вершин она помечается как посещенная: visited[s]=true, и активной становится та вершина, путь из s в которую минимален. Допустим, путь из s в u короче, чем из s в t, следовательно, вершина u становиться активной и выше описанным образом исследуются ее соседи, за исключением вершины s. Далее, u помечается как пройденная: visited[u]=true, активной становится вершина t, и вся процедура повторяется для нее. Алгоритм Дейкстры продолжается до тех пор, пока все доступные из s вершины не будут исследованы.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1. Граф. Расчет весов

Граф путей от остановки ДК Кирова до СГУПС представлен на рисунке 2.1.

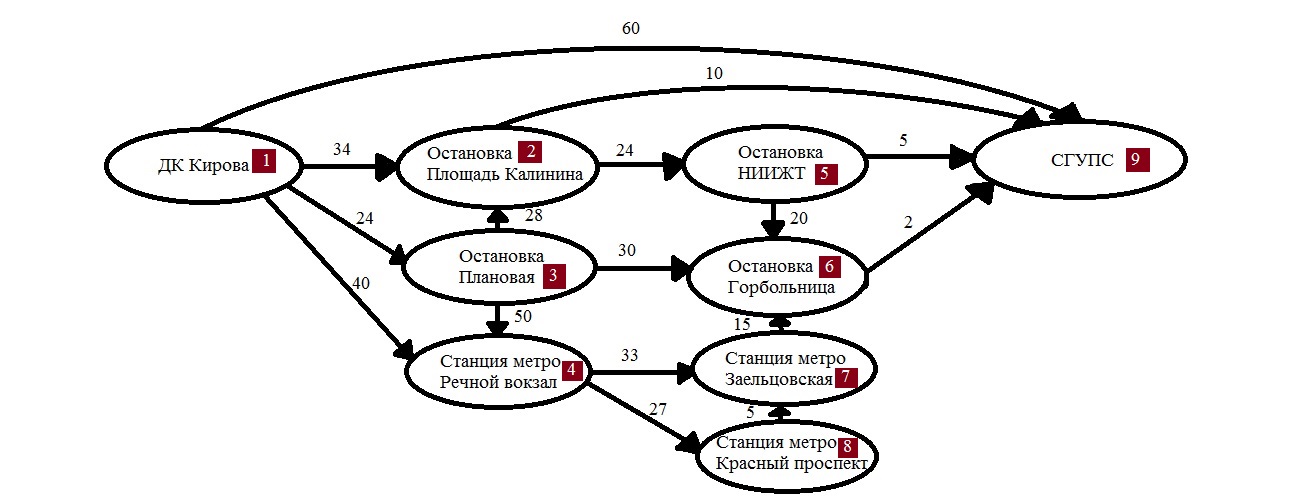


Рисунок 2.1. – Граф путей от ДК Кирова до СГУПС

Данный граф является ориентированным. На ребрах (дугах) графа обозначены веса, рассчитывающиеся как некоторый коэффициент. Критериями для расчета коэффициента послужили: стоимость проезда на общественном транспорте и время в пути. Расчет весов (коэффициентов), согласно транспортным тарифам на 1 апреля 2015 года представлены на рисунке 2.2.

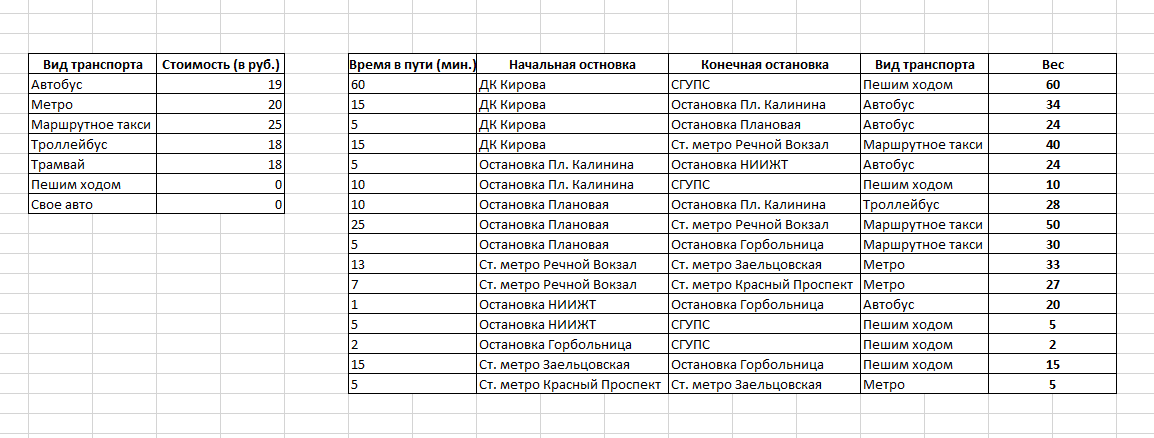


Рисунок 2.2 – Таблицы расчета весов графа

2.2. Критерии выбора коэффициента

Вес графа складывается из суммы проезда и времени в пути. Критерии были выбраны не случайно:

Случай 1. Вес был бы равен стоимости проезда.

Тогда путь из точки 1 в точку 9 (напрямую, пешком, бесплатно) был бы самым оптимальным, то есть это и был бы кратчайший путь. Но если подумать логически, то путь пешком, который занял бы около часа времени, не многие с практической точки зрения могут назвать самым оптимизированным для данного графа, хотя в теории и при расчетах, этот путь был бы кратчайшим.

Получается несоответствие теоретических и практических данных. Кто-то возможно бы счел часовой путь пешком, как кратчайший путь с наименьшими затратами.

Случай 2. Вес был бы равен времени в пути.

Тогда мы не можем сразу однозначно определить каким был бы кратчайший путь из точки 1 в точку 9, но можно сказать точно, что выбор такого коэффициента был бы не рационален. Так как, в данном случае кратчайший путь складывался бы из суммы путей с наименьшими временными затратами. С практической точки зрения этот путь не был бы оптимальным, так как стоимость проезда при нынешних тарифах играет одну из важных ролей.

Предположим, что в графе имеются 3 точки, соединенные ребрами, с весами равными стоимости проезда на маршрутном такси, то есть чтобы пройти данный, в нашем предположении кратчайший, так как наименьший по времени, путь необходимо потратить 50 рублей, 25 рублей за один проезд на маршрутном такси. Если предположить, что такой маршрут вы совершаете ежедневно, да еще и не один раз в день, то ежемесячно это обойдется вам в достаточно крупную сумму денег, а значит доказано, что с практической точки зрения данный коэффициент (вес = время в пути) применять иногда не рационально, хотя в теории это будет кратчайший путь.

Кому-то очень важно, чтобы передвижение на транспорте было максимально быстрым, и не важно за какую сумму; другие же предпочтут более дешевый вид транспорта, но менее быстрый, все зависит от вашего финансового состояния. Для меня же такой выбор веса (вес = время в пути) не рационален.

Поэтому чтобы уравновесить оба варианта (случая) критерием для выбора коэффициента считается сумма: *времени в пути + стоимости пути.* Что наиболее рационально, и приведет к правильному расчету кратчайшего пути.

2.3. Код алгоритмов

1. Алгоритм Флойда-Уоршелла

if (ComboBox1.Text='Алгоритм Флойда-Уоршелла') then

begin

for i:=2 to 10 do

for j:=1 to 9 do

for k:=1 to 9 do

if (w[j,i-1]<>1000) then

if (j<>k) then

if (w[j,k]>w[j,i-1]+w[i-1,k]) then

begin

w[j,k]:=w[j,i-1]+w[i-1,k];

h[j,k]:=h[j,i-1];

end;

Memo2.Lines.Add('Матрица историй');

Memo2.Lines.Add('');

Memo2.Lines.Add('H 1 2 3 4 5 6 7 8 9');

Memo2.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

s:='';

for i:=1 to 9 do

begin

s:=inttostr(i)+'|';

for j:=1 to 9 do

s:=s+' '+inttostr(h[i,j]);

Memo2.Lines.Add(s);

end;

Memo2.Lines.Add('');

Memo2.Lines.Add('Матрица весов');

Memo2.Lines.Add('');

Memo2.Lines.Add('W 1 2 3 4 5 6 7 8 9');

Memo2.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

s:='';

for i:=1 to 9 do

begin

s:=inttostr(i)+'|';

for j:=1 to 9 do

if (w[i,j]<100) then

s:=s+' '+inttostr(w[i,j])

else s:=s+' '+inttostr(0);

Memo2.Lines.Add(s);

end;

Далее выполняется вывод кратчайшего пути, он рассмотрен в следующем разделе.

2. Алгоритм Форда-Беллмана

m:=1;

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]<>1000) then

begin

edje[m].a:=i;

edje[m].b:=j;

edje[m].w:=w[i,j];

m:=m+1;

end;

d[1]:=0;

for i:=2 to 9 do

d[i]:=1000;

s:='';

for i:=1 to 9 do

begin

for j:=1 to 15 do

begin

f:=False;

if (d[edje[j].b]>d[edje[j].a]+edje[j].w) then

begin

d[edje[j].b]:=d[edje[j].a]+edje[j].w;

p[edje[j].b]:=edje[j].a;

f:=True;

end;

end;

if (f=False) then break;

end;

Далее выполняется вывод кратчайшего пути, он рассмотрен в следующем разделе.

3. Алгоритм Дейкстры

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]=1000) then

w[i,j]:=0;

for i:=1 to 9 do

begin

d[i]:=1000;

fl[i]:=False;

end;

d[1]:=0;

for j:=1 to 8 do

begin

min:=1000;

for i:=1 to 9 do

if (not fl[i]) and (d[i]<=min) then

begin

min:=d[i];

m:=i;

end;

k:=m;

fl[k]:=True;

for i:=1 to 9 do

if (not fl[i]) and (w[k,i]<>0) and (d[k]<>1000)

and (d[i]>d[k]+w[k,i]) then

begin

d[i]:=d[k]+w[k,i];

p[i]:=k;

end;

end;

Далее выполняется вывод кратчайшего пути, он рассмотрен в следующем разделе.

2.4. Основные используемые программные модули

1. Вывод кратчайшего пути на экран

1.1. Алгоритм Флойда-Уоршелла

Memo2.Lines.Add('');

Memo2.Lines.Add('Кратчайший путь из точки 1 в 9:');

k:=1; s:='1';

l[1]:=1;

i:=2;

repeat

l[i]:=h[l[i-1],9];

s:=s+'->'+inttostr(l[i]);

i:=i+1;

k:=k+1;

until (l[i-1]=9);

Memo2.Lines.Add(s);

Универсальность (смена на начальную и конечную точку)

1.2. Алгоритм Форда-Беллмана, алгоритм Дейкстры

m:=9;

s:=''; s1:='';

for i:=9 downto 1 do

begin

s:=s+inttostr(m);

if p[m]=0 then break

else m:=p[m];

end;

for i:=length(s) downto 1 do

s1:=s1+s[i]+'->';

delete(s1,length(s1)-1,2);

Memo2.Lines.Add('Кратчайший путь из точки 1 в 9:');

Memo2.Lines.Add(s1);

2. Заполнение необходимых матриц и полей, при создании формы.

ComboBox1.Items.Add('Алгоритм Флойда-Уоршелла');

ComboBox1.Items.Add('Алгоритм Форда-Беллмана');

ComboBox1.Items.Add('Алгоритм Дейкстры');

**for** i:=1 **to** 9 **do**

**for** j:=1 **to** 9 **do**

w[i,j]:=0;

//заполние таблицы весов

w[1,2]:=34; w[1,3]:=24; w[1,4]:=40; w[1,9]:=60; w[2,5]:=24; w[2,9]:=10;

w[3,2]:=28; w[3,4]:=50; w[3,6]:=30; w[4,7]:=33; w[4,8]:=27;

w[5,6]:=20; w[5,9]:=5; w[6,9]:=2; w[7,6]:=15; w[8,7]:=5;

//заполнение таблицы историй

**for** i:=1 **to** 9 **do**

**for** j:=1 **to** 9 **do**

**if** (w[i,j]<>0) **then** h[i,j]:=j

**else** h[i,j]:=0;

Memo1.Lines.Add('Матрица весов');

Memo1.Lines.Add('');

Memo1.Lines.Add('W 1 2 3 4 5 6 7 8 9');

Memo1.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

**for** i:=1 **to** 9 **do**

**begin**

s:=inttostr(i)+'|';

**for** j:=1 **to** 9 **do**

**if** (w[i,j]<10) **then**

s:=s+' '+inttostr(w[i,j])

**else** s:=s+' '+inttostr(w[i,j]);

Memo1.Lines.Add(s);

**end**;

Memo1.Lines.Add('');

Memo1.Lines.Add('Матрица историй');

Memo1.Lines.Add('');

Memo1.Lines.Add('H 1 2 3 4 5 6 7 8 9');

Memo1.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

**for** i:=1 **to** 9 **do**

**begin**

s:=inttostr(i)+'|';

**for** j:=1 **to** 9 **do**

s:=s+' '+inttostr(h[i,j]);

Memo1.Lines.Add(s);

**end**;

**for** i:=1 **to** 9 **do**

**for** j:=1 **to** 9 **do**

**if** (w[i,j]=0) **then** w[i,j]:=1000;

-данный алгоритм подготавливает матрицу весов (w) для заполнения

-затем программно заполняет ее, в соответствии с графом

-заполняется матрица историй (h), исходя из заполненных ячеек в матрице весов

-создаются «шапки» для оформления матриц в запущенном приложении на экране пользователя

-обе матрицы выводятся на экран

-нули в матрице весов заполняются большими числами (выбрано число 1000) для дальнейшей работы с алгоритмами

-заполняются поля записи, содержащей поля: начальная точка (поле «a»), конечная точка (поле «b»), вес (поле «w»).

2.5. Визуальное оформление приложения. Возможности

На рисунке 2.5.1 представлено визуальное оформление приложения

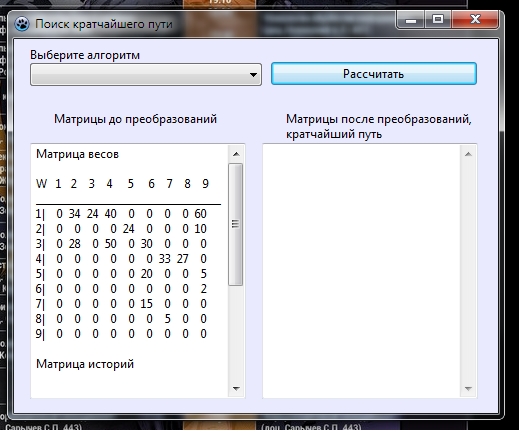


Рисунок 2.5.1 – Окно приложения про запуске

На рисунке 2.5.2 представлен набор функций (алгоритмов)

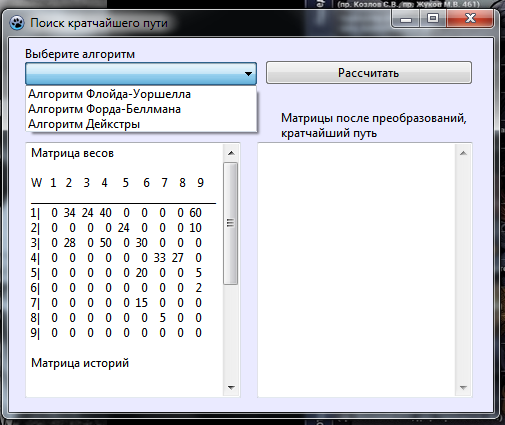


Рисунок 2.5.2 – Окно при выборе алгоритма

3. ТЕСТИРОВАНИЕ ПРОГРАММЫ

На рисунках 3.1-3.3 представлены кратчайшие пути, просчитанные разными алгоритмами.

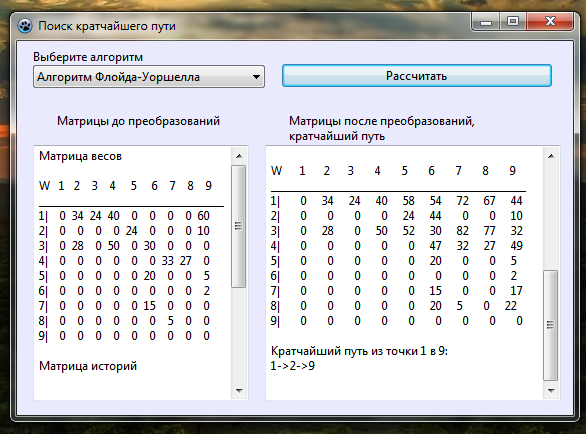


Рисунок 3.1 – Расчет кратчайшего пути по алгоритму

Флойда-Уоршелла

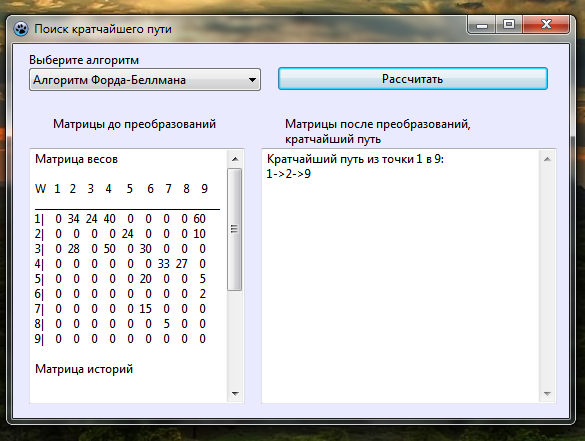


Рисунок 3.2 – Расчет кратчайшего пути по алгоритму

Форда-Беллмана

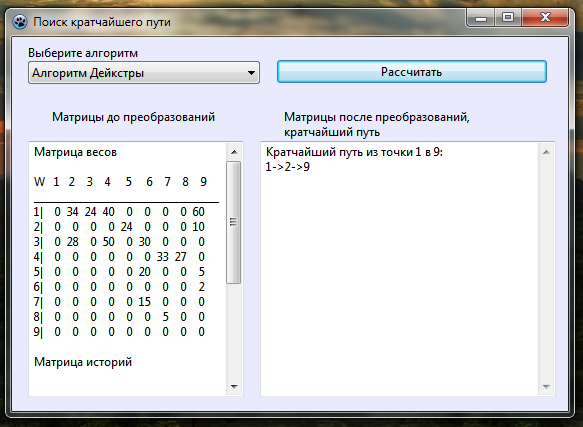


Рисунок 3.3 – Расчет кратчайшего пути по алгоритму

Дейкстры

ВЫВОД

Итогом курсового проекта стало создание программы решения транспортной задачи: поиска кратчайшего пути по трем алгоритмам: Флойда-Уоршелла, Форда-Беллмана и Дейкстры. Все из вышеназванных алгоритмов корректно работают в программе и выводят правильный результат.

Для подтверждения того, что кратчайший путь действительно найден правильно с использованием трех алгоритмов, достаточно посмотреть на рисунок 2.1. на котором представлен несложный граф. После его анализа, нетрудно догадаться, что кратчайшим и оптимальным путем из точки 1 в точку 9, будет маршрут через точку 2, так как он имеет наименьший вес.

В ходе выполнения курсовой работы были детально изучены алгоритмы, а также проанализирована справочная и научная информация по ним.

В завершении курсового проекта можно сделать вывод, что наиболее оптимальным из точки 1 в точку 9 оказался путь через точку 2. А именно, чтобы добраться до СГУПС кратчайшим путем с остановки ДК Кирова, нужно сесть на автобус, выйти на остановке площадь Калинина и дойти до университета пешком. На полный путь вы потратите около 25 минут и 19 рублей.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. Алгоритмы: построение и анализ. — 2-е изд. — М.: Вильямс», 2006. — С. 1296.

2. А. И. Белоусов, С. Б. Ткачев. Дискретная математика. — М.: МГТУ, 2006. — 744 с.

3. А. В. Левитин. Глава 8. Динамическое программирование: Алгоритм Флойда поиска кратчайших путей между всеми парами. — М.: Вильямс, 2006. — С. 349 — 353.

4. А. В. Левитин. Глава 9. Жадные методы: Алгоритм Дейкстры. — М.: «Вильямс», 2006. — С. 189—195.

ПРИЛОЖЕНИЕ А – КОД ПРОГРАММЫ

unit Unit1;

{$mode objfpc}{$H+}

interface

uses

Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls;

type

{ TForm1 }

TForm1 = class(TForm)

Button1: TButton;

ComboBox1: TComboBox;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Memo1: TMemo;

Memo2: TMemo;

procedure Button1Click(Sender: TObject);

procedure ComboBox1Change(Sender: TObject);

procedure FormCreate(Sender: TObject);

private

{ private declarations }

public

{ public declarations }

end;

var

Form1: TForm1;

i,j,k,m,min:integer;

f:Boolean;

d,p,l:array [1..10] of Integer;

w,h:array [1..10,1..10] of Integer;

fl:array [1..10] of Boolean;

s,s1:String;

edje:array [1..16] of Record

a,b,w:Integer;

end;

implementation

{$R \*.lfm}

{ TForm1 }

procedure TForm1.Button1Click(Sender: TObject);

begin

if (ComboBox1.Text='Алгоритм Флойда-Уоршелла') then

begin

for i:=2 to 10 do

for j:=1 to 9 do

for k:=1 to 9 do

if (w[j,i-1]<>1000) then

if (j<>k) then

if (w[j,k]>w[j,i-1]+w[i-1,k]) then

begin

w[j,k]:=w[j,i-1]+w[i-1,k];

h[j,k]:=h[j,i-1];

end;

Memo2.Lines.Add('Матрица историй');

Memo2.Lines.Add('');

Memo2.Lines.Add('H 1 2 3 4 5 6 7 8 9');

Memo2.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

s:='';

for i:=1 to 9 do

begin

s:=inttostr(i)+'|';

for j:=1 to 9 do

s:=s+' '+inttostr(h[i,j]);

Memo2.Lines.Add(s);

end;

Memo2.Lines.Add('');

Memo2.Lines.Add('Матрица весов');

Memo2.Lines.Add('');

Memo2.Lines.Add('W 1 2 3 4 5 6 7 8 9');

Memo2.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

s:='';

for i:=1 to 9 do

begin

s:=inttostr(i)+'|';

for j:=1 to 9 do

if (w[i,j]<100) then

s:=s+' '+inttostr(w[i,j])

else s:=s+' '+inttostr(0);

Memo2.Lines.Add(s);

end;

Memo2.Lines.Add('');

Memo2.Lines.Add('Кратчайший путь из точки 1 в 9:');

k:=1; s:='1';

l[1]:=1;

i:=2;

repeat

l[i]:=h[l[i-1],9];

s:=s+'->'+inttostr(l[i]);

i:=i+1;

k:=k+1;

until (l[i-1]=9);

Memo2.Lines.Add(s);

end;

if (ComboBox1.Text='Алгоритм Форда-Беллмана') then

begin

m:=1;

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]<>1000) then

begin

edje[m].a:=i;

edje[m].b:=j;

edje[m].w:=w[i,j];

m:=m+1;

end;

d[1]:=0;

for i:=2 to 9 do

d[i]:=1000;

s:='';

for i:=1 to 9 do

begin

for j:=1 to 15 do

begin

f:=False;

if (d[edje[j].b]>d[edje[j].a]+edje[j].w) then

begin

d[edje[j].b]:=d[edje[j].a]+edje[j].w;

p[edje[j].b]:=edje[j].a;

f:=True;

end;

end;

if (f=False) then break;

end;

m:=9;

s:=''; s1:='';

for i:=9 downto 1 do

begin

s:=s+inttostr(m);

if p[m]=0 then break

else m:=p[m];

end;

for i:=length(s) downto 1 do

s1:=s1+s[i]+'->';

delete(s1,length(s1)-1,2);

Memo2.Lines.Add('Кратчайший путь из точки 1 в 9:');

Memo2.Lines.Add(s1);

end;

if (ComboBox1.Text='Алгоритм Дейкстры') then

begin

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]=1000) then

w[i,j]:=0;

for i:=1 to 9 do

begin

d[i]:=1000;

fl[i]:=False;

end;

d[1]:=0;

for j:=1 to 8 do

begin

min:=1000;

for i:=1 to 9 do

if (not fl[i]) and (d[i]<=min) then

begin

min:=d[i];

m:=i;

end;

k:=m;

fl[k]:=True;

for i:=1 to 9 do

if (not fl[i]) and (w[k,i]<>0) and (d[k]<>1000)

and (d[i]>d[k]+w[k,i]) then

begin

d[i]:=d[k]+w[k,i];

p[i]:=k;

end;

end;

m:=9;

s:=''; s1:='';

for i:=9 downto 1 do

begin

s:=s+inttostr(m);

if p[m]=0 then break

else m:=p[m];

end;

for i:=length(s) downto 1 do

s1:=s1+s[i]+'->';

delete(s1,length(s1)-1,2);

Memo2.Lines.Add('Кратчайший путь из точки 1 в 9:');

Memo2.Lines.Add(s1);

end;

end;

procedure TForm1.ComboBox1Change(Sender: TObject);

begin

Memo2.Clear;

for i:=1 to 9 do

for j:=1 to 9 do

w[i,j]:=0;

//заполние таблицы весов

w[1,2]:=34; w[1,3]:=24; w[1,4]:=40; w[1,9]:=60; w[2,5]:=24; w[2,9]:=10;

w[3,2]:=28; w[3,4]:=50; w[3,6]:=30; w[4,7]:=33; w[4,8]:=27;

w[5,6]:=20; w[5,9]:=5; w[6,9]:=2; w[7,6]:=15; w[8,7]:=5;

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]<>0) then h[i,j]:=j

else h[i,j]:=0;

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]=0) then w[i,j]:=1000;

end;

procedure TForm1.FormCreate(Sender: TObject);

begin

ComboBox1.Items.Add('Алгоритм Флойда-Уоршелла');

ComboBox1.Items.Add('Алгоритм Форда-Беллмана');

ComboBox1.Items.Add('Алгоритм Дейкстры');

for i:=1 to 9 do

for j:=1 to 9 do

w[i,j]:=0;

//заполние таблицы весов

w[1,2]:=34; w[1,3]:=24; w[1,4]:=40; w[1,9]:=60; w[2,5]:=24; w[2,9]:=10;

w[3,2]:=28; w[3,4]:=50; w[3,6]:=30; w[4,7]:=33; w[4,8]:=27;

w[5,6]:=20; w[5,9]:=5; w[6,9]:=2; w[7,6]:=15; w[8,7]:=5;

//заполнение таблицы историй

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]<>0) then h[i,j]:=j

else h[i,j]:=0;

Memo1.Lines.Add('Матрица весов');

Memo1.Lines.Add('');

Memo1.Lines.Add('W 1 2 3 4 5 6 7 8 9');

Memo1.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

for i:=1 to 9 do

begin

s:=inttostr(i)+'|';

for j:=1 to 9 do

if (w[i,j]<10) then

s:=s+' '+inttostr(w[i,j])

else s:=s+' '+inttostr(w[i,j]);

Memo1.Lines.Add(s);

end;

Memo1.Lines.Add('');

Memo1.Lines.Add('Матрица историй');

Memo1.Lines.Add('');

Memo1.Lines.Add('H 1 2 3 4 5 6 7 8 9');

Memo1.Lines.Add('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');

for i:=1 to 9 do

begin

s:=inttostr(i)+'|';

for j:=1 to 9 do

s:=s+' '+inttostr(h[i,j]);

Memo1.Lines.Add(s);

end;

for i:=1 to 9 do

for j:=1 to 9 do

if (w[i,j]=0) then w[i,j]:=1000;

end;

end.