Федеральное агентство образования РФ

Ивановский государственный энергетический университет

Кафедра ПОКС

Курсовая работа

на тему

«Разработка программы определения оптимального маршрута»

Выполнил: студент 1 курса 41^{xx} группы

Шалунов Виталий Сергеевич

Проверил: Алыкова А.Л.

Иваново 2019

Содержание

Задание

Исходные данные:

- карта автомобильных дорог некоторого региона, представленная списком городов с указанием расстояния между городами, связанными автомобильными дорогами;
- список городов, которые необходимо посетить.

Выходные данные:

- маршрут поездки;
- общая протяженность маршрута.

Требования к функциональным характеристикам:

- считывание информации об автомобильных дорогах региона из текстового файла;
- удобный для пользователя список посещаемых дорог;
- определение оптимального (минимальной протяженности) маршрута посещения городов;
- наглядный вывод результатов.

Краткий анализ

Для решения поставленной задачи будем использовать перебор всех возможных вариантов. Для перебора будем использовать следующие алгоритмы: алгоритм Дейкстры, Флойда. С помощью данных алгоритмов мы сможем добираться до нужных нам городов даже тогда, когда между ними нет прямой дороги. При посещении необходимого города мы будем помечать его как посещенный. Порядок посещения городов мы будем сохранять в отдельном массиве.

При выполнении задачи нам понадобится матрица смежности, которую программа будет анализировать, она будет отображать расстояние и связь между городами.

Математическая модель

Для начала следует считать данные из файла, либо создать новый. В файл данные будут записываться определенным образом, чтобы сделать считывание файла универсальным. Пример содержания файла с картой автомобильных дорог:

11
0 - Шуя
1 - Кохма
2 - Иваново
3 - Вичуга
4 - Кинешма
5 - Родники
6 - Владимир
7 - Москва
8 - Палех
9 - Арзамас
10 -Нижний Новгород
0 19 0 0 90 60 100 0 60 0 0
19 0 13 0 0 0 0 0 0 0 0
0 13 0 0 0 50 70 0 0 0 0
0 0 0 0 19 19 0 0 0 0 0
90 0 0 19 0 20 0 0 0 0 0
60 0 50 19 20 0 0 0 0 0 0
100 0 70 0 0 0 0 200 0 0 0
0 0 0 0 0 0 200 0 0 0 0
60 0 0 0 0 0 0 0 0 100 190
0 0 0 0 0 0 0 0 100 0 70
0 0 0 0 0 0 0 0 190 70 0

По входному файлу мы заполним массив городов, матрицу смежности.

Затем я решил составить сразу матрицу кратчайших путей с помощью алгоритма Флойда, что позволит проверить является ли дорога между двумя городами кратчайшая.

После чего мы предоставляем список городов, которые загружены в карте и спрашиваем у пользователя, сколько городов он хочет посетить, запоминаем это число. Затем следует вывести на экран список городов, а пользователь должен ввести цифры, соответствующие тем городам, которые он желает посетить, также дополним этот ввод условие, что первый город будет тем городов, в котором сейчас находится пользователь. Далее формируем матрицу в следующем формате: в первой строке и в первом столбце у нас будут храниться индексы городов, а в остальных ячейках будут храниться расстояния между этими городами, а в дополнительной строке под каждым городом будем хранить значение, отвечающее за то, сколько раз данный город был посещен. Приведу пример образованной матрицы:

0 0 1 6 10
0 0 19 100 0
1 19 0 0 0
6 100 0 0 0
10 0 0 0 0
0 0 0 0 0

Затем начинается основная обработка этой матрицы. Реализуется перебор с помощью рекурсии. Создадим матрицу, которая будет отвечать за маршрут, там будут храниться индексы порядка посещения городов.

Цикл обработки будет работать до тех пор, пока не будут посещены все нужные города. Город из которого будем начинать поиск в другой город назначить следует, как стартовый, после чего будем искать город, в котором мы ещё не были, примем следующий город в качестве конечного. При посещении нового города, он будет становится стартовым.

Будем учитывать, что до каждого города мы добираемся по минимальному пути. Как только въезжаем в город, добавляем пройденное расстояние от прошлого города.

Как только у нас будут посещены все нужные города, следует сравнить пройденный путь с минимальным на данном этапе. Если настоящий маршрут окажется выгоднее, то нам следует его запомнить.

Реализация алгоритма Флойда.

Шаг 0. Начальная матрица $\mathbf{D_0}$ строятся непосредственно по заданной схеме дорог. Матрица $\mathbf{D_0}$ симметрична, за исключением пары элементов $\mathbf{d_{35}}$ и $\mathbf{d_{53}}$, где $\mathbf{d_{53}}$ равно бесконечности, поскольку невозможен переход от узла 5 к узлу 3:

D_0

	1	2	3	4	5
1	—	3	10	∞	∞
2	3	—	∞	5	∞
3	10	∞	—	6	15
4	∞	5	6	—	4
5	∞	∞	∞	4	—

Рисунок 1 - Начальное состояние матрицы D_0

Шаг 1. В матрице $\mathbf{D_0}$ выделены ведущие строка и столбец **(k = 1)**. Двойной рамкой представлены элементы $\mathbf{d_{23}}$ и $\mathbf{d_{32}}$, единственные среди элементов матрицы $\mathbf{D_0}$, значения которых можно улучшить с помощью треугольного оператора. Для получения матрицы $\mathbf{D_1}$, выполняем следующие действия:

Заменяем $\mathbf{d_{23}}$ на $\mathbf{d_{21} + d_{13} = 3 + 10 = 13}$ и устанавливаем $\mathbf{s_{23} = 1}$.

Заменяем $\mathbf{d_{32}}$ на $\mathbf{d_{31} + d_{12} = 10 + 3 = 13}$ и устанавливаем $\mathbf{s_{32} = 1}$.

Матрица $\mathbf{D_1}$ имеет следующий вид:

D_1

	1	2	3	4	5
1	—	3	10	∞	∞
2	3	—	**13**	5	∞
3	10	**13**	—	6	15
4	∞	5	6	—	4
5	∞	∞	∞	4	—

Рисунок 2 - Матрица D_1

Шаг 2. Полагаем **k = 2**; в матрице $\mathbf{D_1}$ выделены ведущие строка и столбец. Треугольный оператор применяется к элементам матрицы $\mathbf{D_1}$, выделенным двойной рамкой. В результате получаем матрицу $\mathbf{D_2}$:

D_2

	1	2	3	4	5
1	—	3	10	**8**	∞
2	3	—	13	5	∞
3	10	13	—	6	15
4	**8**	5	6	—	4
5	∞	∞	∞	4	—

Рисунок 3 - Матрица D_2

Шаг 3. Полагаем **k = 3**; в матрице $\mathbf{D_2}$ выделены ведущие строка и столбец. Треугольный оператор применяется к элементам матрицы $\mathbf{D_2}$, выделенным двойной рамкой. В результате получаем матрицу $\mathbf{D_3}$:

D_3

	1	2	3	4	5
1	—	3	10	8	**25**
2	3	—	13	5	**28**
3	10	13	—	6	15
4	8	5	6	—	4
5	∞	∞	∞	4	—

Рисунок 4 - Матрица D_3

Шаг 4. Полагаем **k = 4**, ведущие строка и столбец в матрице $\mathbf{D_3}$ выделены. Получаем новую матрицу $\mathbf{D_4}$:

D_4

	1	2	3	4	5
1	—	3	10	8	**12**
2	3	—	**11**	5	**9**
3	10	**11**	—	6	**10**
4	8	5	6	—	4
5	**12**	**9**	**10**	4	—

Рисунок 5 - Матрица D_4

Шаг 5. Полагаем **k = 5**, ведущие строка и столбец в матрице $\mathbf{D_4}$ выделены. Никаких действий на этом шаге не выполняем; вычисления закончены.

Конечная матрица $\mathbf{D_4}$ содержит всю информацию, необходимую для определения кратчайших путей между любыми двумя узлами сети.

Реализация алгоритма Дейкстры.

Метка самой вершины 1 полагается равной 0, метки остальных вершин – недостижимо большое число (в идеале — бесконечность). Это отражает то, что расстояния от вершины 1 до других вершин пока неизвестны. Все вершины графа помечаются как непосещенные. Начальный шаг показан на рисунке 6.

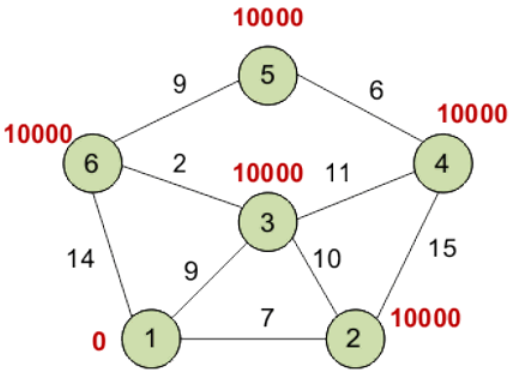


Рисунок 6 – Начальный шаг алгоритма Дейкстры

Первый шаг

Минимальную метку имеет вершина 1. Её соседями являются вершины 2, 3 и 6. Обходим соседей вершины по очереди. Первый сосед вершины 1 – вершина 2, потому что длина пути до неё минимальна. Длина пути в неё через вершину 1 равна сумме кратчайшего расстояния до вершины 1, значению её метки, и длины ребра, идущего из 1-й в 2-ю, то есть 0 + 7 = 7. Это меньше текущей метки вершины 2 (10000), поэтому новая метка 2-й вершины равна 7.

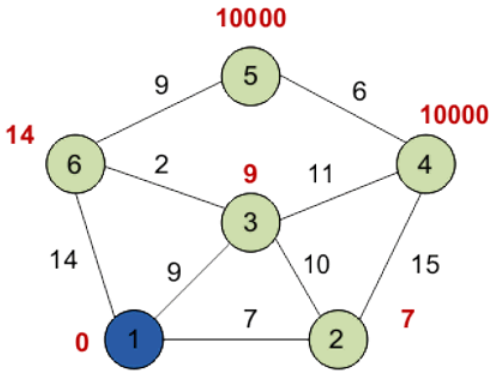


Рисунок 7 – Первый шаг алгоритма Дейкстры

Аналогично находим длины пути для всех других соседей (вершины 3 и 6). Все соседи вершины 1 проверены. Текущее минимальное расстояние до вершины 1 считается окончательным и пересмотру не подлежит. Вершина 1 отмечается как посещенная.

Второй шаг

Шаг 1 алгоритма повторяется. Снова находим «ближайшую» из непосещенных вершин. Это вершина 2 с меткой 7. Снова пытаемся уменьшить метки соседей выбранной вершины, пытаясь пройти в них через 2-ю вершину. Соседями вершины 2 являются вершины 1, 3 и 4.

Вершина 1 уже посещена. Следующий сосед вершины 2 — вершина 3, так как имеет минимальную метку из вершин, отмеченных как не посещённые. Если идти в неё через 2, то длина такого пути будет равна 17 (7 + 10 = 17). Но текущая метка третьей вершины равна 9, а 9 < 17, поэтому метка не меняется.

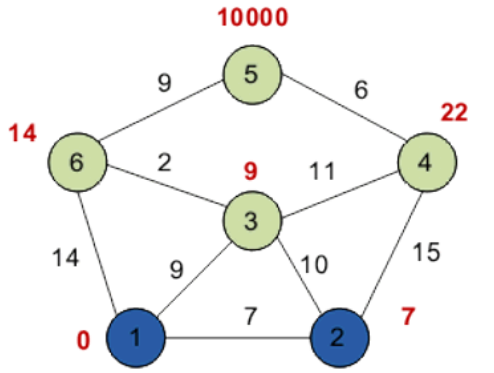


Рисунок 8 – Второй шаг алгоритма Дейкстры

Ещё один сосед вершины 2 — вершина 4. Если идти в неё через 2-ю, то длина такого пути будет равна 22 (7 + 15 = 22). Поскольку 22<10000, устанавливаем метку вершины 4 равной 22.

Все соседи вершины 2 просмотрены, помечаем её как посещенную.

Третий шаг

Повторяем шаг алгоритма, выбрав вершину 3. После её «обработки» получим следующие результаты.

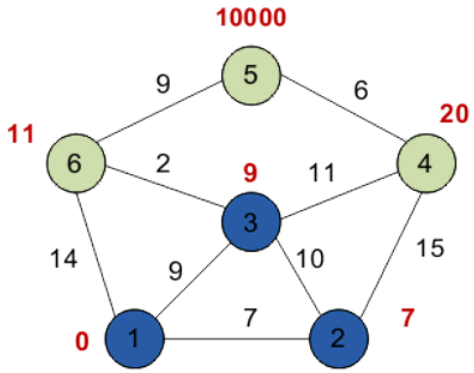


Рисунок 9 – Третий шаг алгоритма Дейкстры

Четвертый шаг

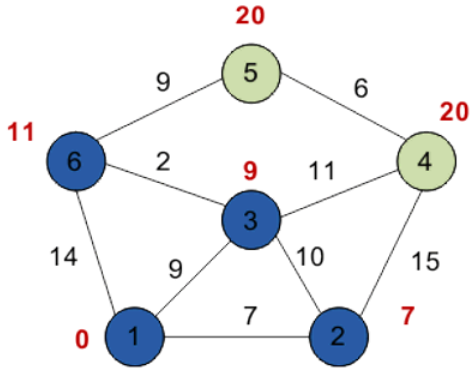


Рисунок 10 – Четвёртый шаг алгоритма Дейкстры

Пятый шаг

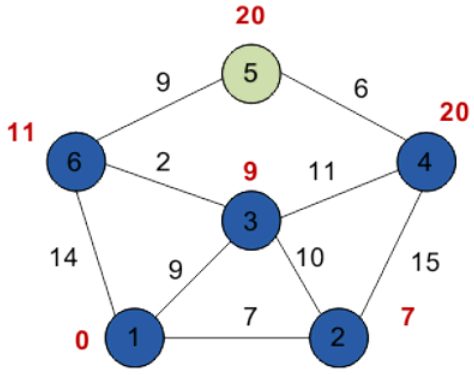


Рисунок 11 – Пятый шаг алгоритма Дейкстры

Шестой шаг

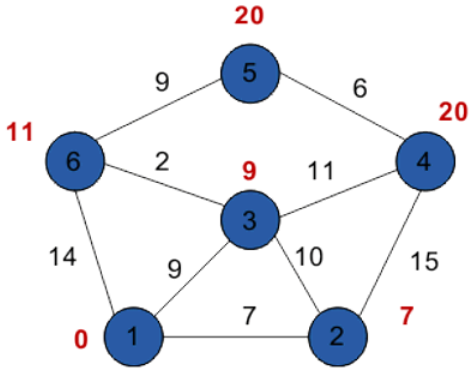


Рисунок 12 – Шестой шаг алгоритма Дейкстры

Таким образом, кратчайшим путем из вершины 1 в вершину 5 будет путь через вершины **1 — 3 — 6 — 5**, поскольку таким путем мы набираем минимальный вес, равный 20.

Для вывода кратчайшего пути будем рассматривать вершины с конца. Рассматриваем конечную вершину (в данном случае — вершина **5**), и для всех вершин, с которой она связана, находим длину пути, вычитая вес соответствующего ребра из длины пути конечной вершины. Так, вершина **5** имеет длину пути **20**. Она связана с вершинами **6** и **4**. Для вершины **6** получим вес **20 — 9 = 11 (совпал)**. Для вершины **4** получим вес **20 — 6 = 14 (не совпал)**.

Если в результате мы получим значение, которое совпадает с длиной пути рассматриваемой вершины (в данном случае — вершина **6**), то именно из нее был осуществлен переход в конечную вершину. Отмечаем эту вершину на искомом пути.

Далее определяем ребро, через которое мы попали в вершину **6**. И так пока не дойдем до начала. Если в результате такого обхода у нас на каком-то шаге совпадут значения для нескольких вершин, то можно взять любую из них — несколько путей будут иметь одинаковую длину.

Описание алгоритмов

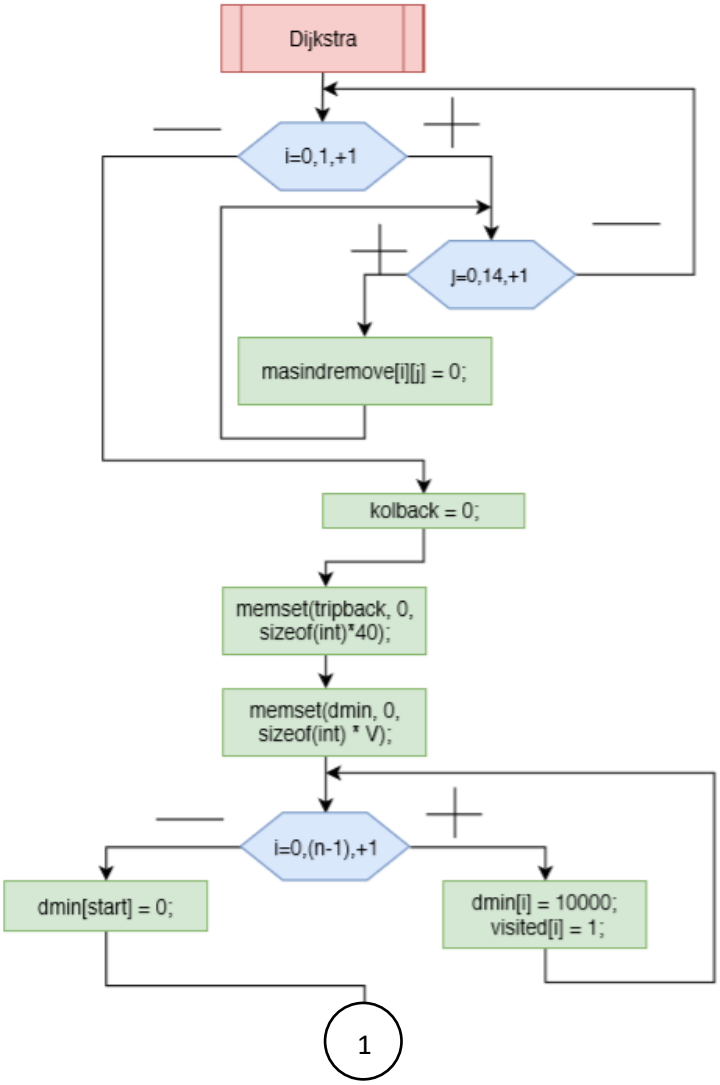
Алгоритм Дейкстры

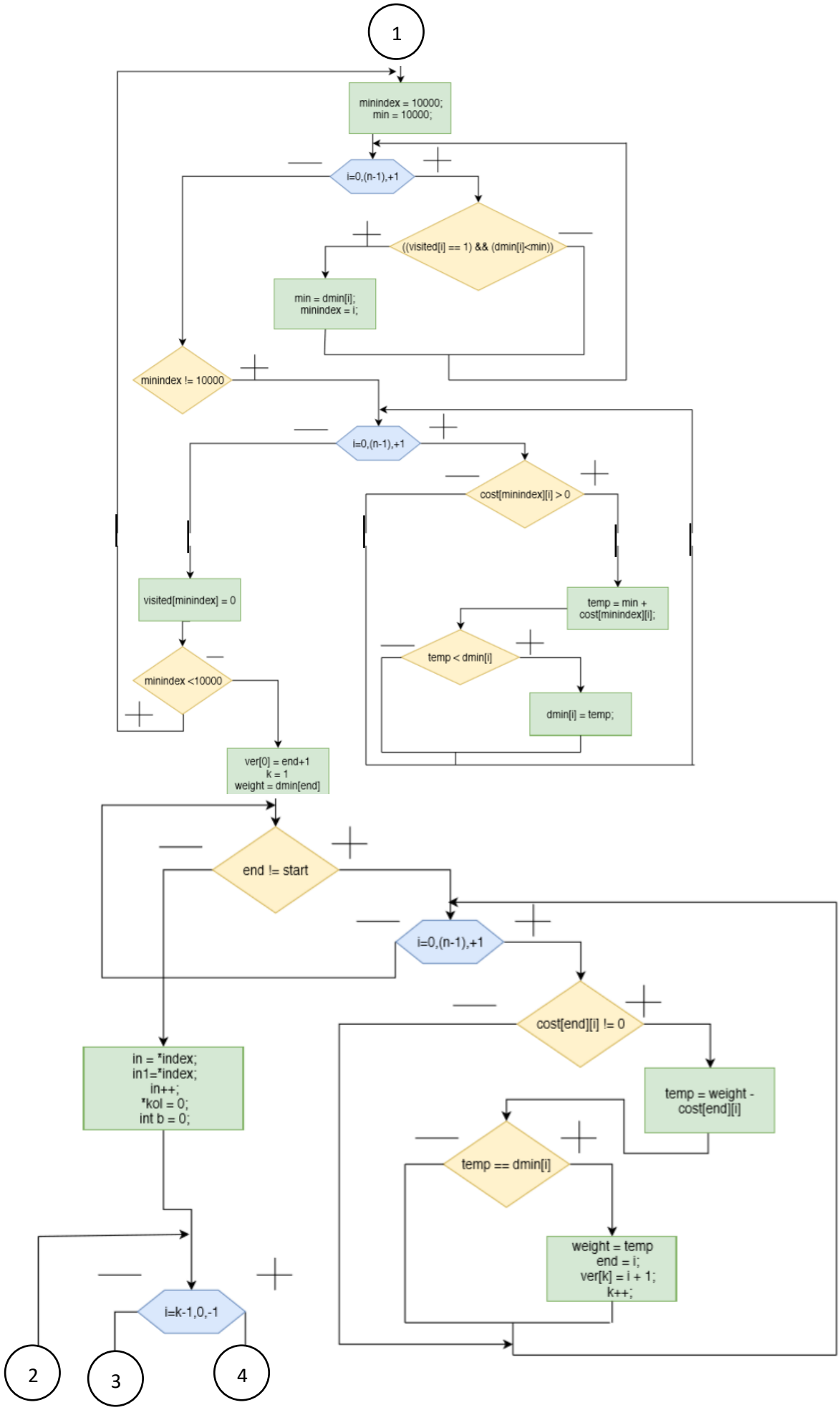
Находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

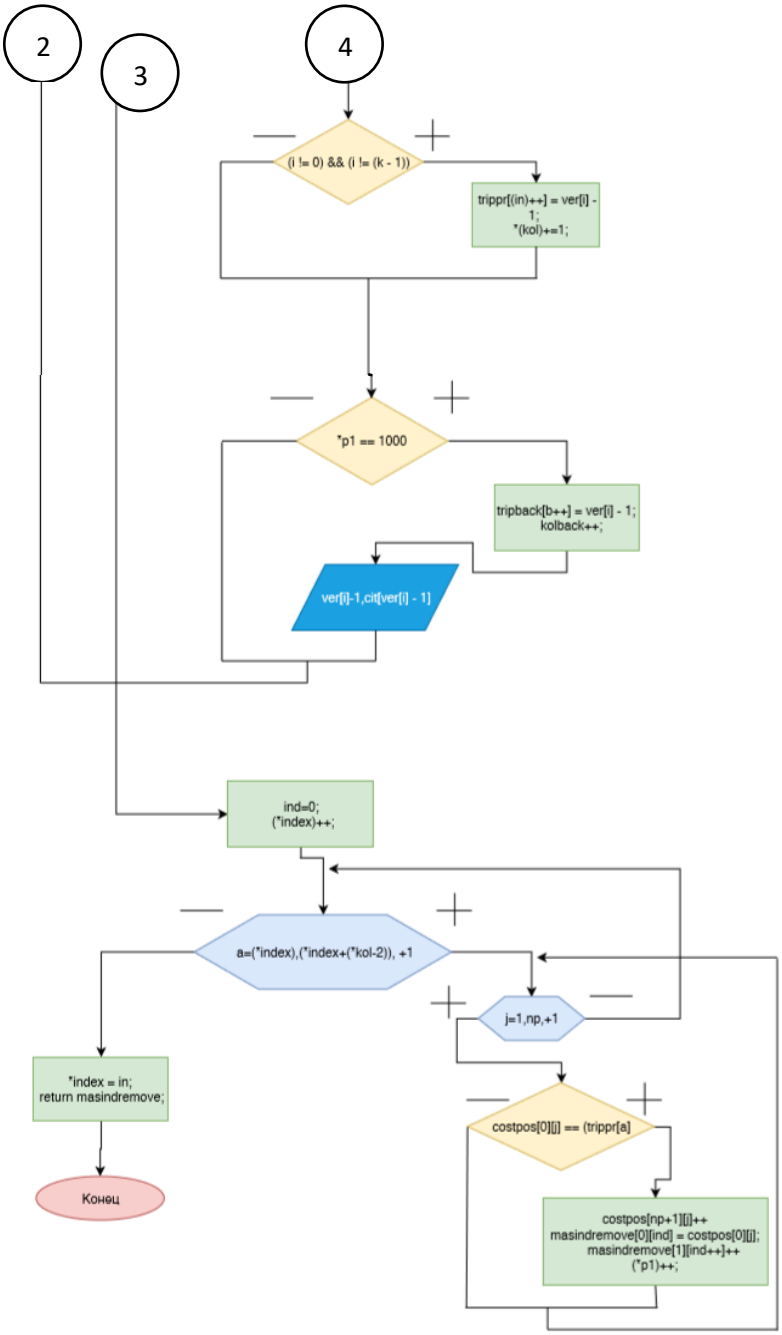
Используется матрица смежности d, где элементы матрицы – расстояния между городами. А для хранения принадлежности элемента множеству U — массив булевых переменных.

В начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом. Массив флагов заполняется нулями. Затем запускается основной цикл.

На каждом шаге цикла мы ищем вершину v с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины u. Если в них (в u) расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его. Цикл завершается, когда флаги всех вершин становятся равны 1, либо когда у всех вершин с флагом 0 d[i,j] = ∞. Последний случай возможен тогда и только тогда, когда граф G несвязный.




1
minindex = 10000;
min = 10000;
i=0,(n-1),+1
((visited[i] == 1) && (dmin[i]<min))
min = dmin[i];
minindex = i;
minindex != 10000
i=0,(n-1),+1
cost[minindex][i] > 0
visited[minindex] = 0
temp = min +
cost[minindex][i];
temp < dmin[i]
minindex <10000
dmin[i] = temp;
ver[0] = end+1
k = 1
weight = dmin[end]
end != start
i=0,(n-1),+1
cost[end][i] != 0
in = *index;
in1=*index;
in++;
*kol = 0;
int b = 0;
temp = weight -
cost[end][i]
temp == dmin[i]
weight = temp
end = i;
ver[k] = i + 1;
k++;
i=k-1,0,-1
2
3
4



Алгоритм Флойда

Алгоритм Флойда - динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа.

Ключевая идея алгоритма — разбиение процесса поиска кратчайших путей на **фазы**.

Перед k-ой фазой (k=1..n) считается, что в матрице расстояний d[ ][ ] сохранены длины таких кратчайших путей, которые содержат в качестве внутренних вершин только вершины из множества {1,2…,k-1} (вершины графа мы нумеруем, начиная с единицы).

Иными словами, перед k-ой фазой величина d[i][j] равна длине кратчайшего пути из вершины i в вершину j, если этому пути разрешается заходить только в вершины с номерами, меньшими k (начало и конец пути не считаются).

Легко убедиться, что чтобы это свойство выполнилось для первой фазы, достаточно в матрицу расстояний d[][] записать матрицу смежности графа: d[i][j]=g[i][j] — стоимости ребра из вершины i в вершину j. При этом, если между какими-то вершинами ребра нет, то записать следует величину "бесконечность" ∞. Из вершины в саму себя всегда следует записывать величину 0, это критично для алгоритма.

Пусть теперь мы находимся на k-ой фазе, и хотим пересчитать матрицу d[][] таким образом, чтобы она соответствовала требованиям уже для k+1-ой фазы. Зафиксируем какие-то вершины i и j. У нас возникает два принципиально разных случая:

• Кратчайший путь из вершины i в вершину j, которому разрешено дополнительно проходить через вершины{1,2,…,k}, совпадает с кратчайшим путём, которому разрешено проходить через вершины множества {1,2,…,k-1}.

В этом случае величина d[i][j] не изменится при переходе с k-ой на k+1-ую фазу.

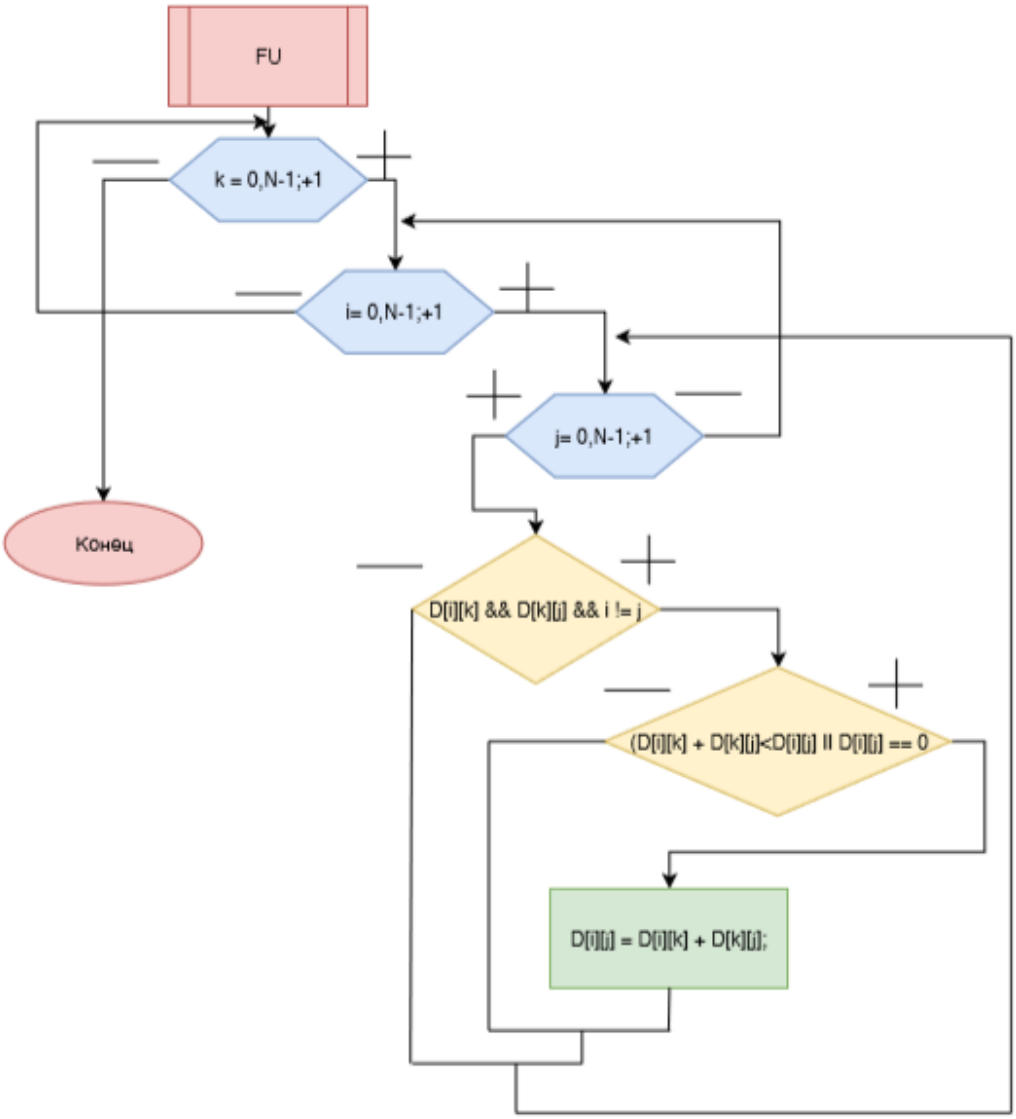
• "Новый" кратчайший путь стал **лучше** "старого" пути.

Это означает, что "новый" кратчайший путь проходит через вершину k. Сразу отметим, что мы не потеряем общности, рассматривая далее только простые пути (т.е. пути, не проходящие по какой-то вершине дважды).

Тогда заметим, что если мы разобьём этот "новый" путь вершиной k на две половинки (одна идущая i→k, а другая — k→j), то каждая из этих половинок уже не заходит в вершину k. Но тогда получается, что длина каждой из этих половинок была посчитана ещё на k-1-ой фазе или ещё раньше, и нам достаточно взять просто сумму d[i][k]+d[k][j], она и даст длину "нового" кратчайшего пути.

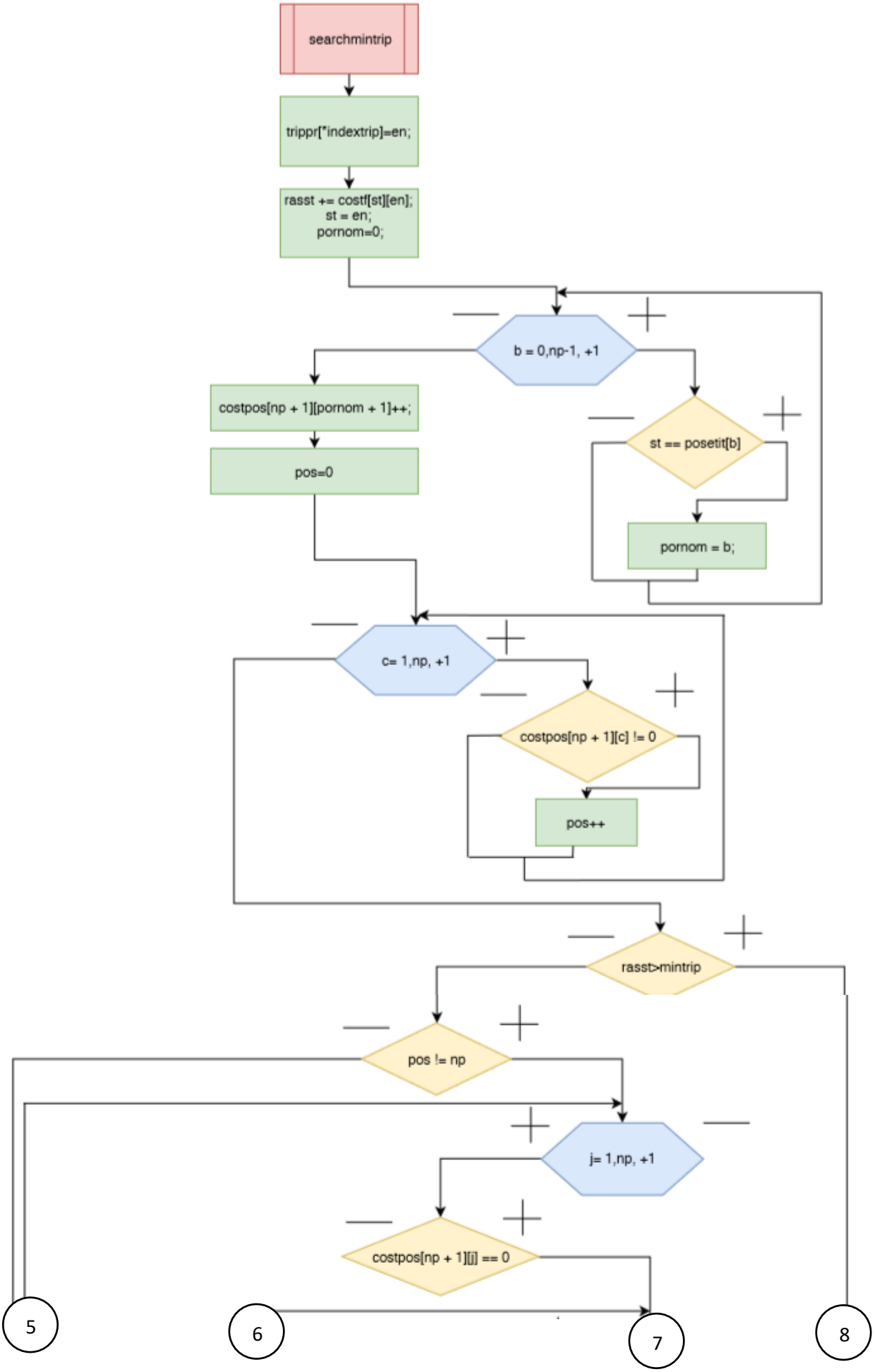
Объединяя эти два случая, получаем, что на k-ой фазе требуется пересчитать длины кратчайших путей между всеми парами вершин i и j следующим образом:

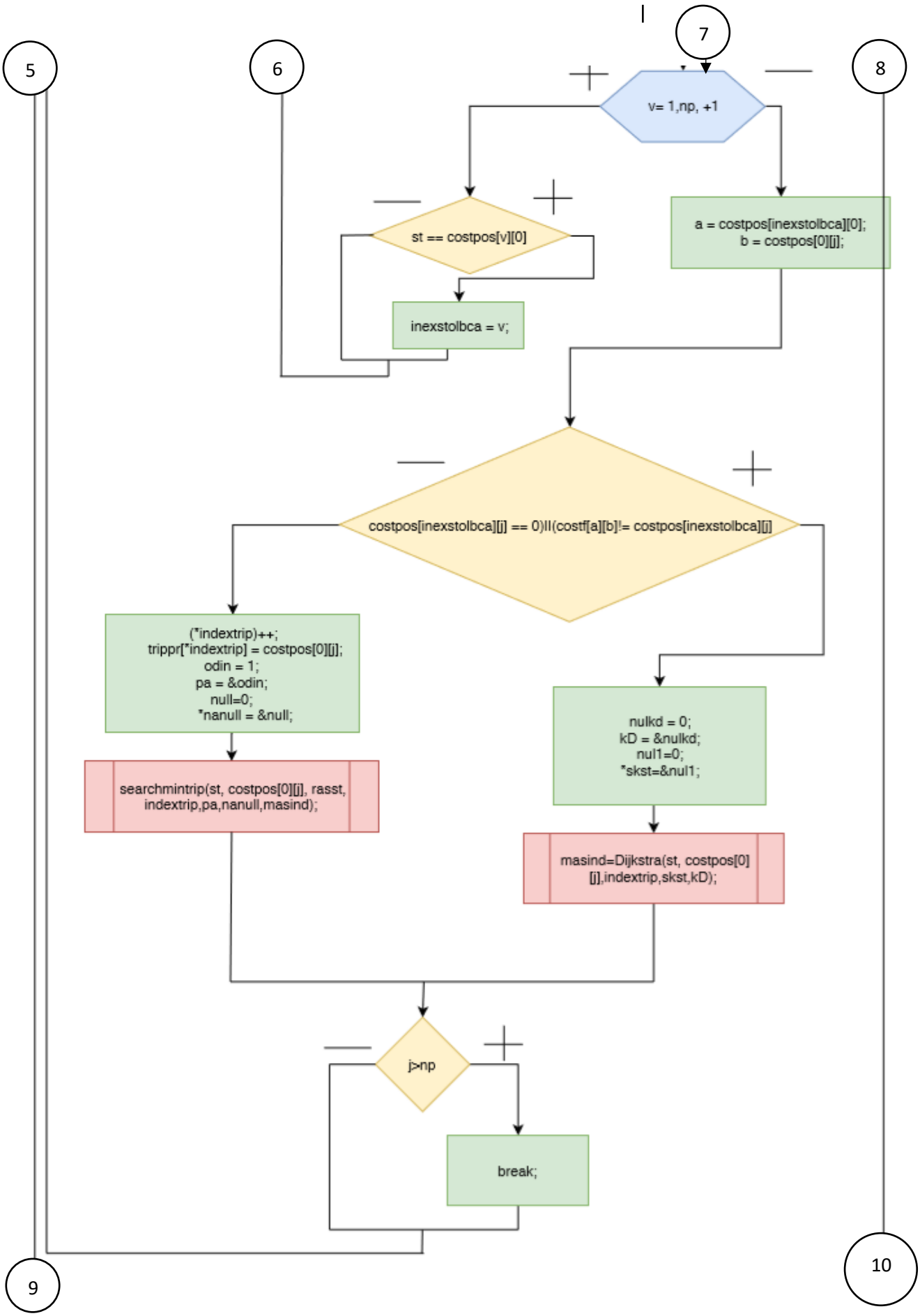
```
new_d[i][j] = min (d[i][j], d[i][k] + d[k][j]);
```

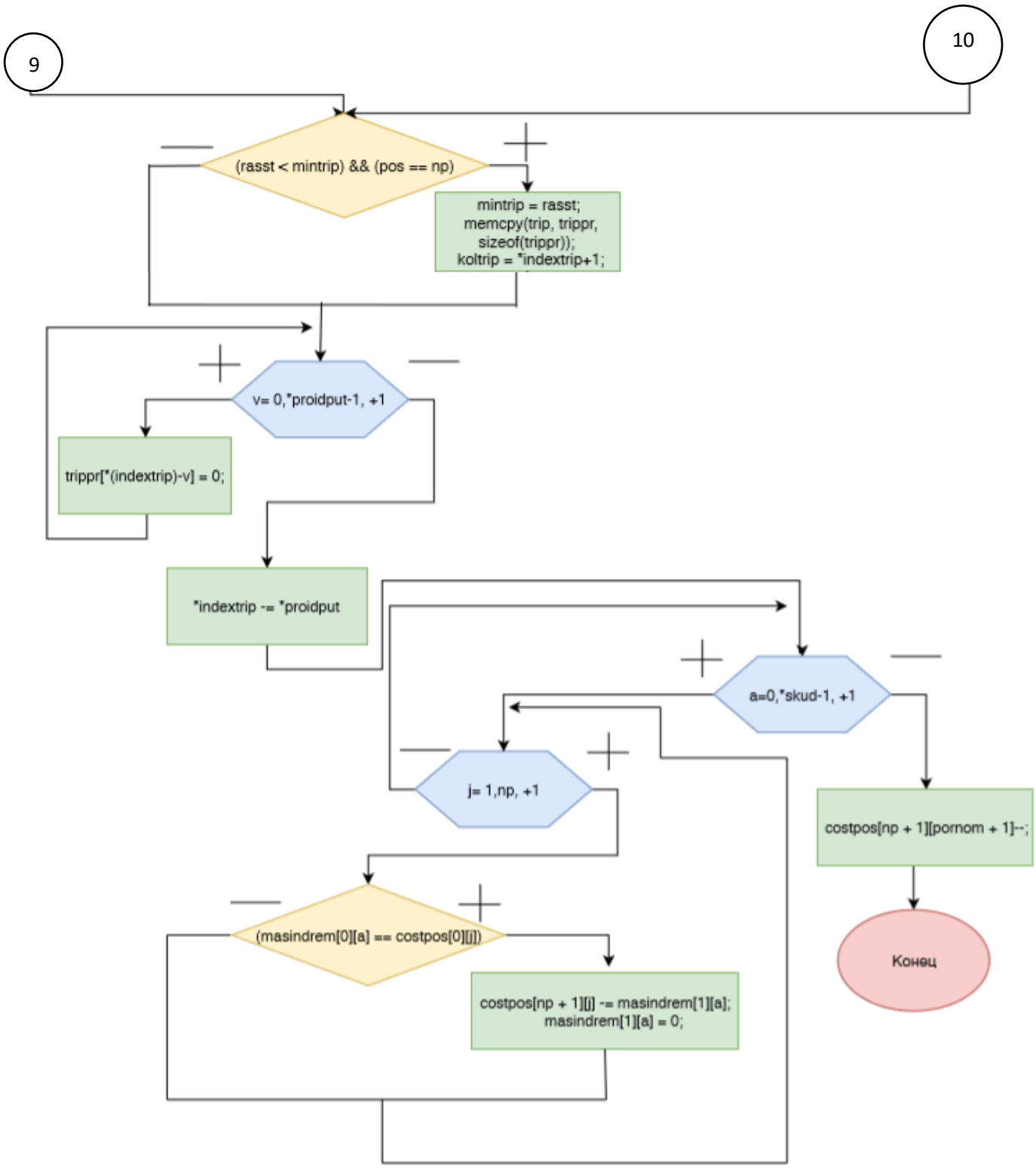


Перебор возможных маршрутов

Для поиска минимального маршрута следует осуществить перебор с использованием алгоритмов Дейкстры и Флойда. В процессе перебор мы будем изменяться матрицу, которая будет хранит расстояния между городами, которые следует посетить, а также содержит значение, которое отвечает, сколько раз данный город был посещен, данное значение будем меняться, когда мы будем возвращаться на предыдущие этапы построения маршрута, для этого действия у нас будет создан двумерный массив, который будет содержать: какой город и сколько раз был посещен при построении пути из одного пункта в другой. Как только мы посетим все необходимые города, следует сравнить с минимальным значением получившееся расстояние, если оно окажется меньше, то обновляем минимальное расстояние и копируем массив маршрута для дальнейшего отображения.


searchmintrip
trippr[*indextrip]=en;
rasst += costf[st][en];
st = en;
pornom=0;
b = 0,np-1, +1
costpos[np + 1][pornom + 1]++;
pos=0
st == posetit[b]
pornom = b;
c= 1,np, +1
costpos[np + 1][c] != 0
pos++
rasst>mintrip
pos != np
j= 1,np, +1
costpos[np + 1][j] == 0
5
6
7
8


5
6
7
8
v= 1,np, +1
st == costpos[v][0]
inexstolbca = v;
a = costpos[inexstolbca][0];
b = costpos[0][j];
costpos[inexstolbca][j] == 0)||(costf[a][b]!= costpos[inexstolbca][j]
(*indextrip)++;
trippr[*indextrip] = costpos[0][j];
odin = 1;
pa = &odin;
null=0;
*nanull = &null;
searchmintrip(st, costpos[0][j], rasst, indextrip,pa,nanull,masind);
nulkd = 0;
kD = &nulkd;
nul1=0;
*skst=&nul1;
masind=Dijkstra(st, costpos[0][j],indextrip,skst,kD);
j>np
break;
9
10



Построение карты дорог

В данной функции будет строиться:

- загруженная карта автомобильных дорог
- маршрут, по которому необходимо двигаться, чтоб посетить необходимые города
- маршрут, по которому необходимо двигаться для возвращения в точку отправления

Будем использовать графику на основе WinApi.

Следует представить граф, где вершины (города) будут располагаться по окружности. Для этого следует найти центр данной окружности, задать радиус и в зависимости от количества городов расположить вершины с использованием функций sin, cos. Координаты все городов следует хранить в массиве структур. Пример показан на рисунке 1.

Для построения маршрута через необходимые точки необходимо установить положение указателя для начала в первую точку пути, а затем последовательно нарисовать все дороги. Последовательность посещения городов хранится в массиве.

Приведу пример работы функции, которая отображает построенный маршрут на рисунке 2.

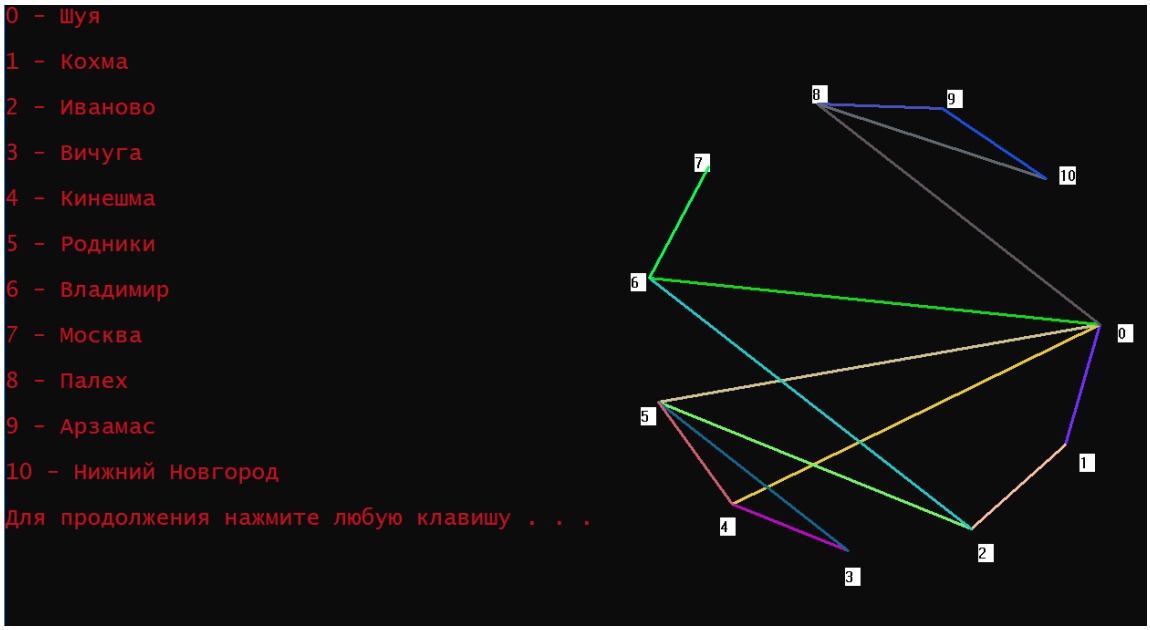


Рисунок 1 – Пример карты автомобильных дорог

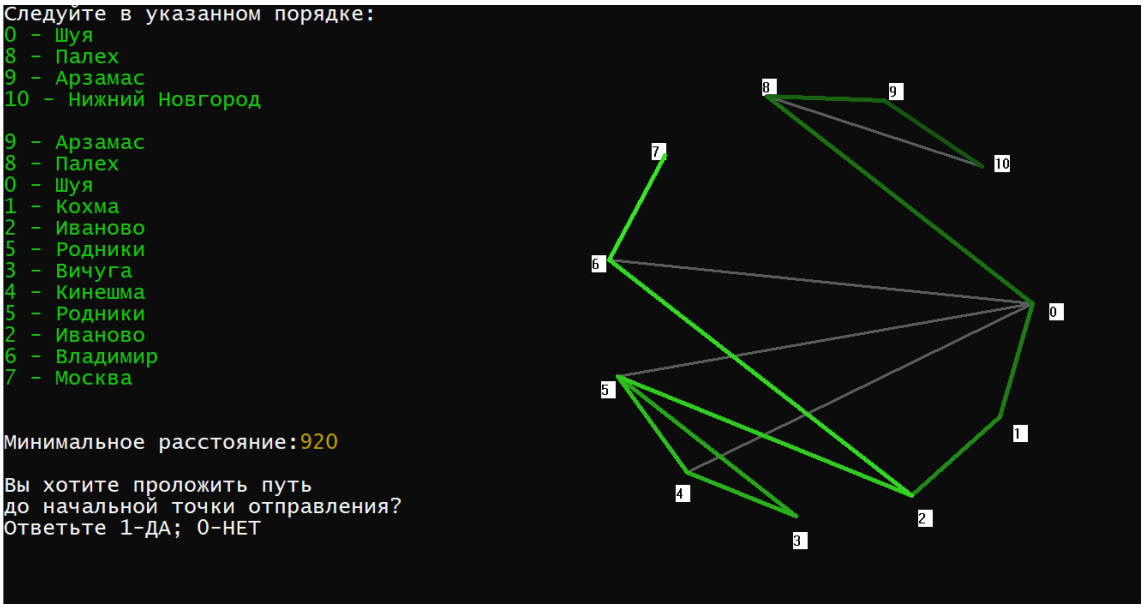


Рисунок 2 – Пример построенного маршрута

Выбор карты дорог

В данной программе реализована функция создания новой карты и выбора существующей карты. Следует описать, как работает выбор нужной карты.

Для того, чтоб сделать имя карт уникальным, я сделал, чтобы текстовые файлы с данными хранились с расширением .rtf. Будем использовать функции findfirstfile и findnextfile, которые будут возвращать нам имена найденных карт. Эти имена будут выводить построчно, пользователь сможет выбрать одну из них, после чего выбранная карта загрузится. Пример выбора необходимой карты представлен на рисунке 3.

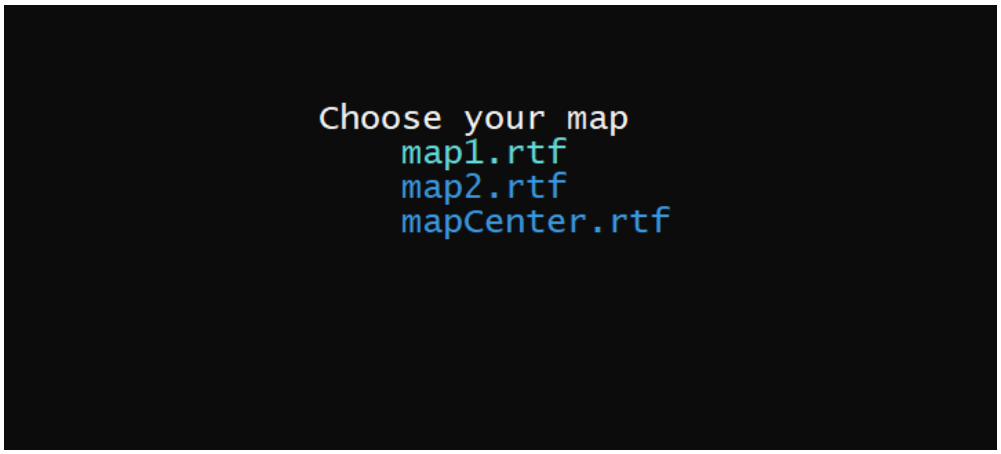


Рисунок 3 – Пример выбора загруженных карт

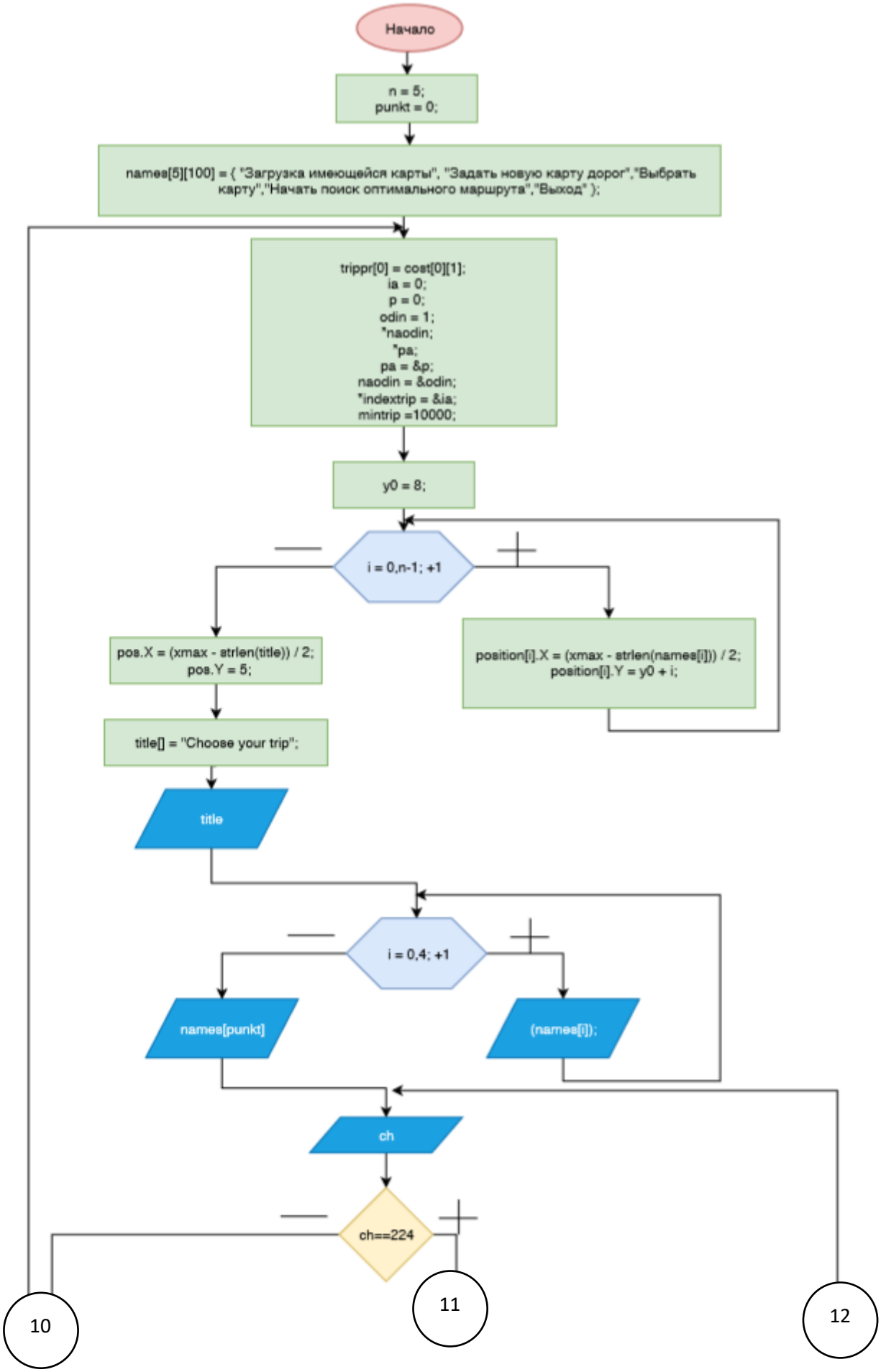
Создание новой карты дорог

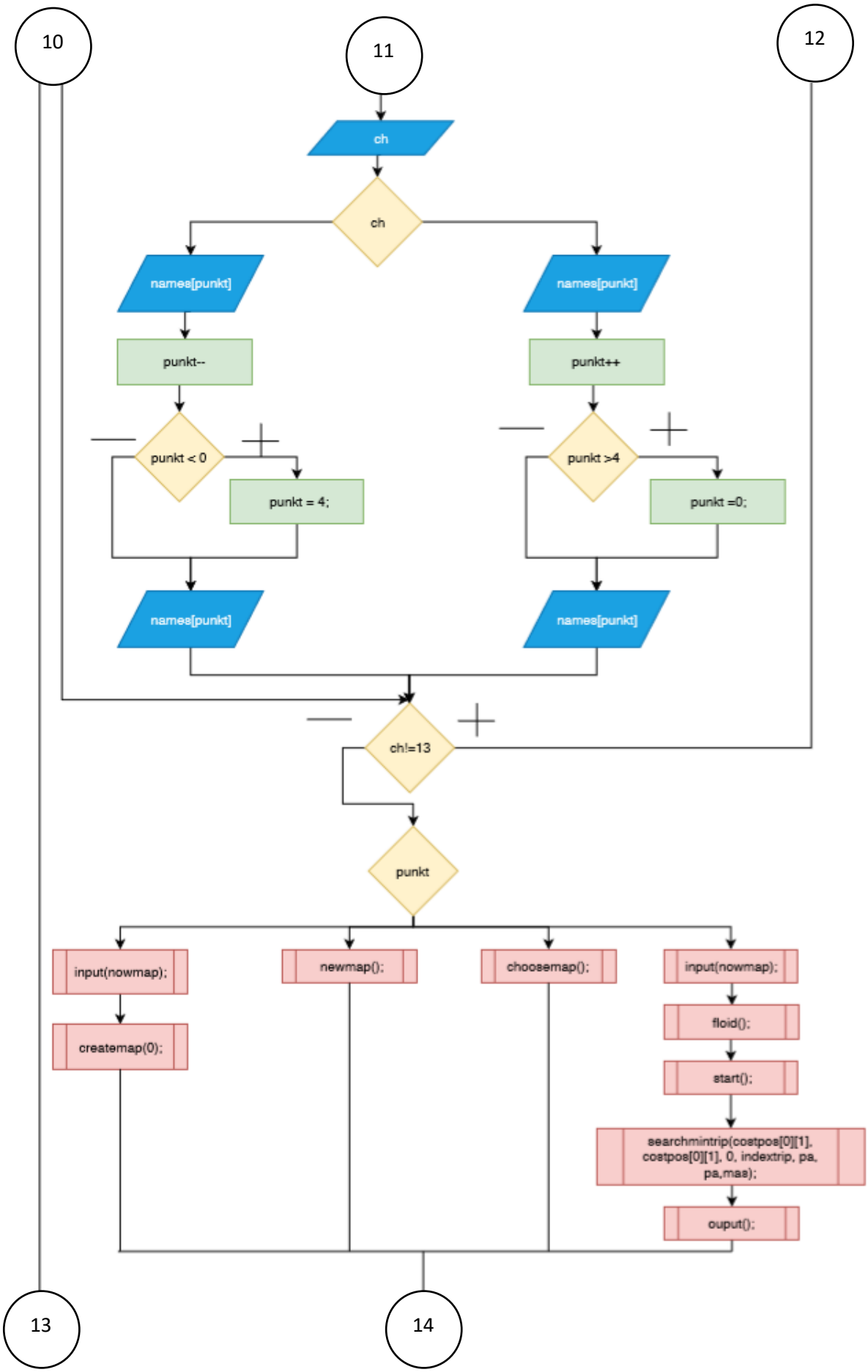
Для создание новой карты мы для начала спросим, какое бы название хотел бы задать пользователь для своей новой карты. После чего нам следует узнать, какое количество городов он будет вводить, затем по порядку спросить название городов.

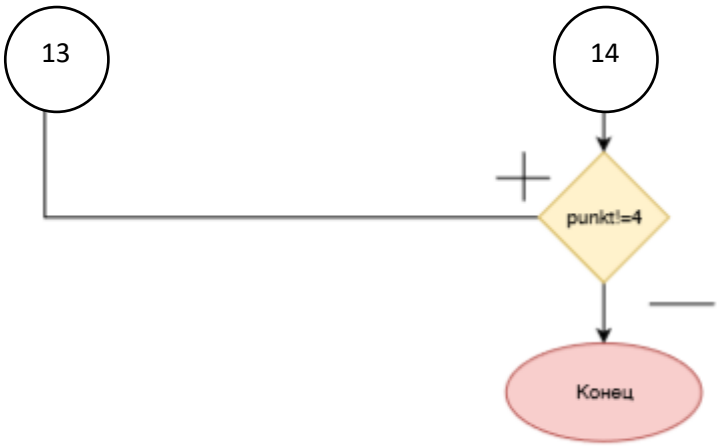
После этого мы будем спрашивать у пользователя расстояние между городами, учитывая, что дороги двусторонние, следует узнавать расстояние между двумя городами только один раз. Получается мы создадим треугольную матрицу и отразим её относительно главной диагонали. Так как пользователь создаёт новую карту, она ему должна быть интересна для дальнейшего рассмотрения, то мы её делаем активной для всей программы в целом.

Основное меню

В функции main() организовано меню для обеспечения удобства пользования будущему потребителю. Перемещение по пунктам реализуется с помощью стрелок и выбора необходимого пункта с помощью кнопки enter.


Начало
n = 5;
punkt = 0;
names[5][100] = { "Загрузка имеющейся карты", "Задать новую карту дорог","Выбрать карту","Начать поиск оптимального маршрута","Выход" };
trippr[0] = cost[0][1];
ia = 0;
p = 0;
odin = 1;
*naodin;
*pa;
pa = &p;
naodin = &odin;
*indextrip = &ia;
mintrip =10000;
y0 = 8;
i = 0,n-1; +1
pos.X = (xmax - strlen(title)) / 2;
pos.Y = 5;
position[i].X = (xmax - strlen(names[i])) / 2;
position[i].Y = y0 + i;
title[] = "Choose your trip";
title
i = 0,4; +1
names[punkt]
(names[i]);
ch
ch==224
10
11
12


10
11
12
ch
ch
names[punkt]
names[punkt]
punkt--
punkt++
punkt < 0
punkt >4
punkt = 4;
punkt =0;
names[punkt]
names[punkt]
ch!=13
punkt
input(nowmap);
createmap(0);
newmap();
choosemap();
input(nowmap);
floid();
start();
searchmintrip(costpos[0][1], costpos[0][1], 0, indextrip, pa, pa,maa);
ouput();
13
14



Описание функций

1. **Функция choosemap**
   - **1.1** ***Назначение:*** осуществляет выбор карты в соответствии с желаением пользователя
   - **1.2** ***Параметры:*** нет
2. **Функция createmap**
   - **2.1** ***Назначение:*** делает построение карты дорог
   - **2.2** ***Параметры:***
     - 2.2.1 wh – алгоритмы построения карты
3. **Функция Dijkstra**
   - **3.1** ***Назначение:*** осуществляет поиск кратчайшего пути между двумя точками
   - **3.2** ***Параметры:***
     - 3.2.1 start – точка отправления
     - 3.2.2 end – точка прибытия
     - 3.2.3 *index – индекс для построения массива маршрута
     - 3.2.4 *p1 – количество необходимых для посещения городов, пройденные в течение построения маршрута из start в end
     - 3.2.5 *kol – количество пройденных городов при построении маршрута из start в end
4. **Функция FU**
   - **4.1** ***Назначение:*** построить матрицу кратчайших путей между всеми городами
   - **4.2** ***Параметры:***
     - 4.2.1 D[][15] – массив расстояний между городами
     - 4.2.2 N – количество городов
5. **Функция input**
   - **5.1** ***Назначение:*** загрузка исходных данных
   - **5.2** ***Параметры:***

5.2.1 in – название карты

6. Функция main

6.1 ***Назначение:*** главная функция, содержит меню

6.2 ***Параметры:*** нет

7. Функция newmap

7.1 ***Назначение:*** сборка новой карты

7.2 ***Параметры:*** нет

8. Функция ouput

8.1 ***Назначение:*** вывод обработанной информации

8.2 ***Параметры:*** нет

9. Функция searchmintrip

9.1 ***Назначение:*** осуществить перебор всех возможных вариантов маршрута

9.2 ***Параметры:***

9.2.1 st – стартовая точка

9.2.2 en – конечная точка

9.2.3 rasst – расстояние в конкретный момент

9.2.4 *indextrip – индекс для построения маршрута

9.2.5 *proidput – количество городов, пройденных на прошлом этапе

9.2.6 *skud – количество городов необходимых для посещения, у которых нужно изменить количество раз, сколько данный город был посещен

9.2.7 **masindrem – двумерный динамический массив, который отвечает на сколько следует удалить в конкретном городе количество посещенных раз

10. Функция start

10.1 ***Назначение:*** узнать у пользователя, какие города он хотел бы посетить

10.2 ***Параметры:*** нет

Руководство пользователя

Для запуска программы выберите ярлык «курсовая (завершенная).exe» и кликните на него 2 раза левой клавишей мыши. Работа в программе будет реализовано с помощью клавиатуры. Для перемещения в меню используйте стрелки управления, для выбора необходимого пункта меню нажмите на enter.

Перед Вами откроется окно с меню, смотри рисунок 4.

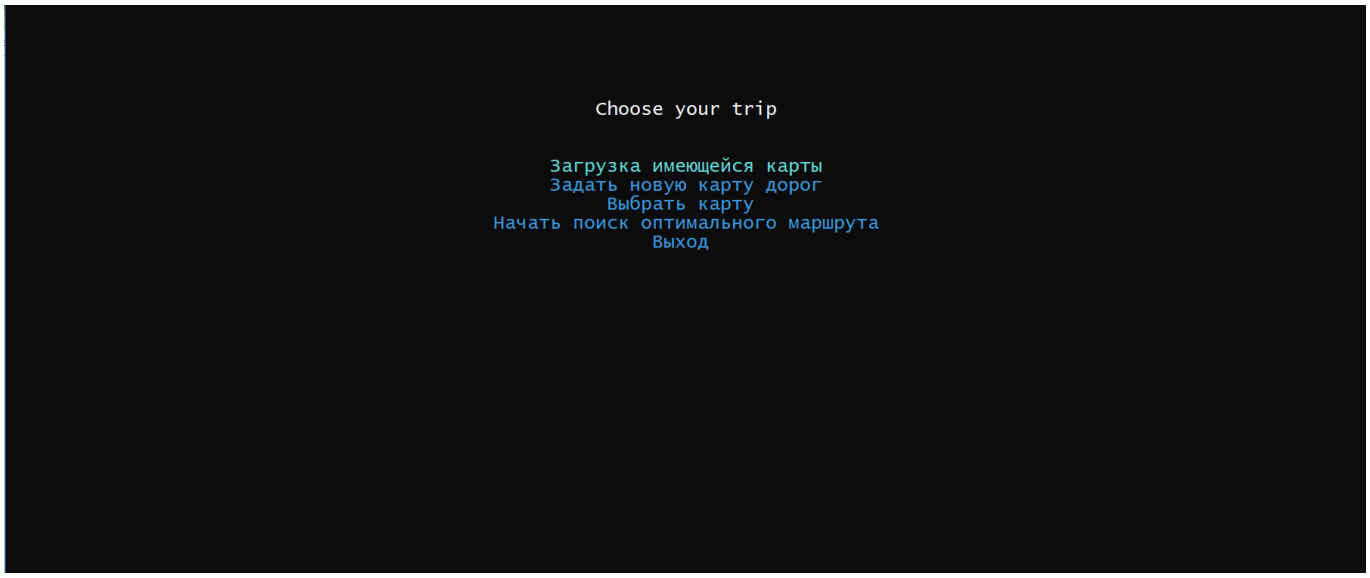


Рисунок 4 – Меню программы

Далее будет описан каждый пункт данного меню.

1. Загрузка имеющейся карты.

   При выборе этого пункта будет нарисована карта дороги, которая была выбрана Вами в качестве активной. Пример работы данного пункта меню смотри рисунок 5.

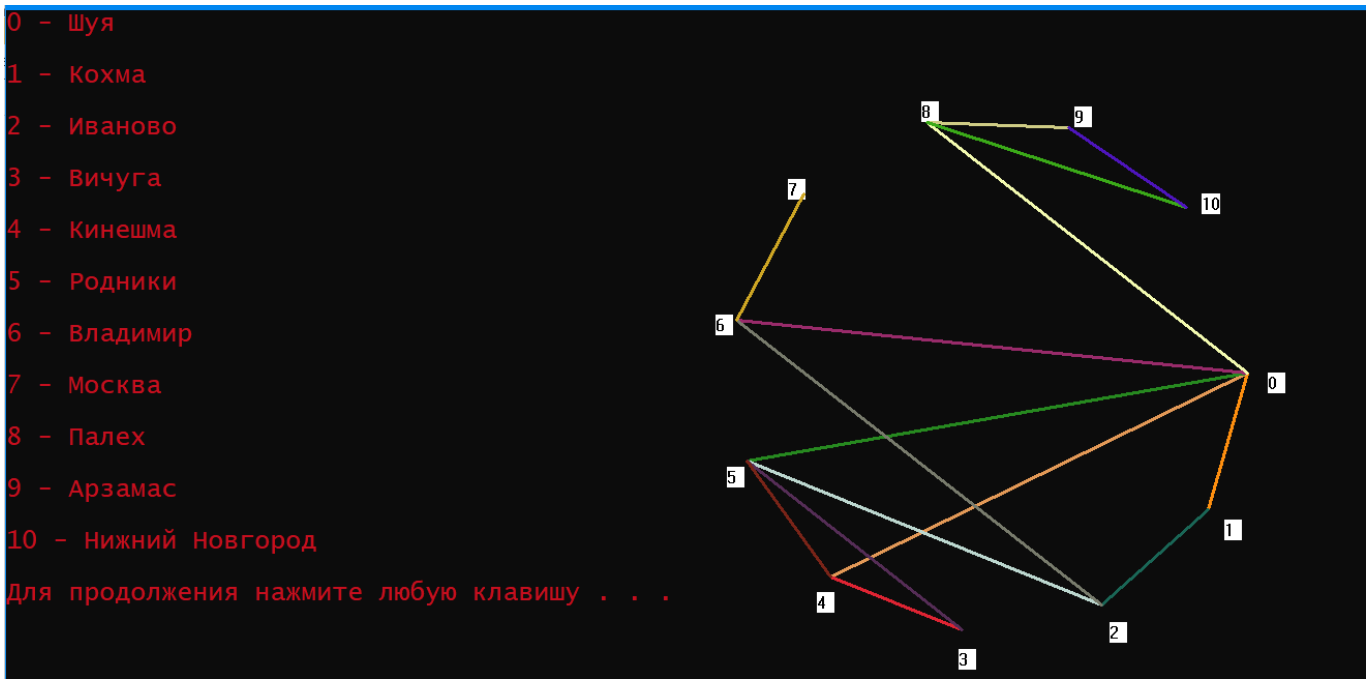


Рисунок 5 – Загрузка имеющейся карты

2. Задать новую карту.

   При выборе данного пункта Вам будет предложено задать новую карту дороги. Для этого вам будет необходимо сначала задать количество городов, которые будут находиться на данной карте, после чего программа Вас спросит расстояния между городами, вам нужно будет вводить числа и нажимать клавишу enter для подтверждения.

3. Выбрать карту.

   При выборе данного пункта Вам будет предложено выбрать карту, которые были загружены раньше. Для перемещения между строками с названиями карт используйте стрелки управления, для выбора активной карты клавишу enter, для выхода из данного пункта меню нажмите на клавишу esc. Пример открывающегося данного пункта меню показан на рисунке 6.

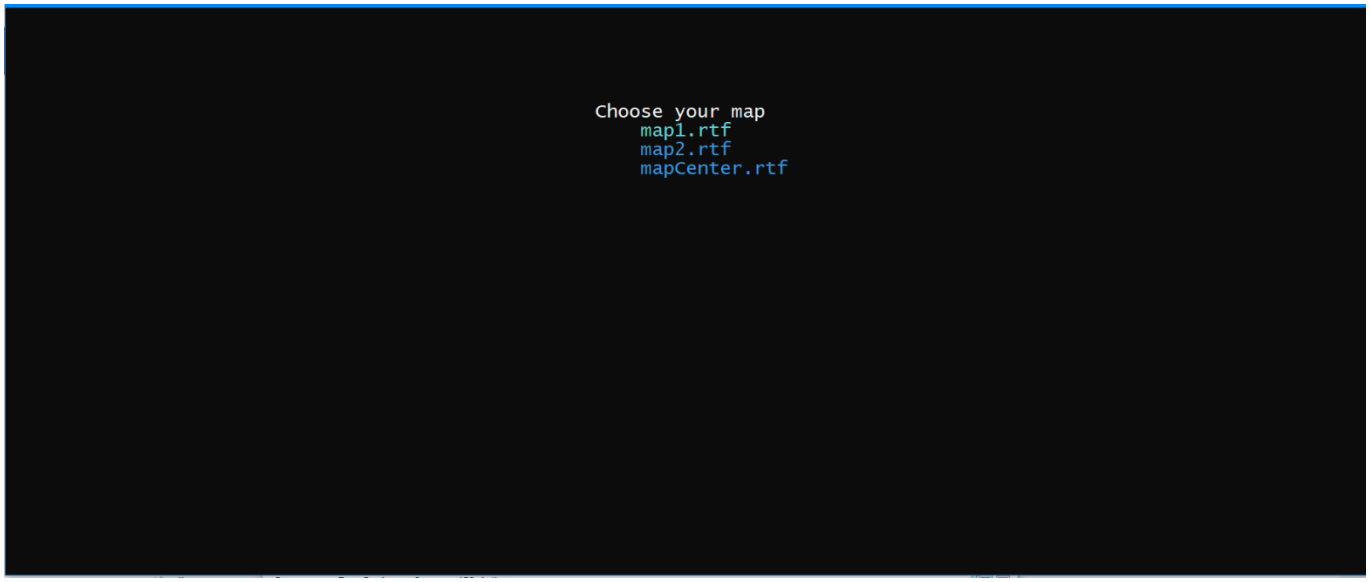


Рисунок 6 – Выбор карты

4. Начать поиск оптимального маршрута.

   При выборе данного пункта меню Вам будет предложено найти минимальный путь между теми городами, которые Вы бы хотели посетить. Для этого Вам следует сначала указать, какое количество городов Вы бы хотели посетить, а затем по очереди выбирать города из предложенного списка путём нажатия на цифру, соответствующему определенному городу, и подтверждения с помощью клавишу enter. После выполнения программы пере Вами покажется список городов, которые будут расположены в порядке следования маршрута, а также будет показана карта с выделением пути, по которому Вам необходимо следовать. Порядок следования будет нарисован от тёмно-зелёного оттенка к светло-зелёному, серым цветом будут показывать существующие дороги, по которым Вам не предстоит двигаться.

   Далее у Вас программа спросит, не хотите ли Вы проложить путь до начальной точки, нажмите на клавишу - 1, если хотите, на клавишу – 0, если не хотите, для подтверждения используйте клавишу enter.

5. Выход.

   При выборе данного пункта меню, программа закончит своё выполнение.

Код программы

```
#include"stdafx.h"
#include<string.h>
#include<Windows.h>
#include <conio.h>
#include <math.h>
#include<time.h>
#include<strsafe.h>

int n, np, mintrip;
int kolcit; //количество городов
int cost[15][15], costf[15][15], costpos[17][17], trippr[40], trip[40], tripback[40];
int posetit[15];
int kolback;
int koltrip = 1;//количество пройденных точек
FILE *file, *city;
const int V = 15;

char cit[15][30];
char map[30][10];
int kolmap;
wchar_t nowmap[30];

struct towns
{
	int x, y;
}tow;

#define WIDTH 1900
#define HEIGHT 700

void FU(int D[][15], int N)
{
	int k;
	int i, j;
	for (k = 0; k<N; k++)
		for (i = 0; i<N; i++)
			for (j = 0; j<N; j++)
				if (D[i][k] && D[k][j] && i != j)
					if (D[i][k] + D[k][j]<D[i][j] || D[i][j] == 0)
						D[i][j] = D[i][k] + D[k][j];

	//for (i = 0; i<N; i++)
	{
		//	for (j = 0; j<N; j++) printf("%d \t",D[i][j]);
		//	printf("\n");
	}
}

void floid()
{
	//printf("Матрица кратчайших путей: \n");
	FU(costf, n);
	//system("pause");
}
```

```
int **Dijkstra(int start, int end, int *index, int *p1, int *kol)
{
	int **masindremove;
	masindremove = new int*[2];
	for (int i = 0; i < 2; i++)
		masindremove[i] = new int[15];
	for (int i = 0; i < 2; i++)
		for (int j = 0; j < 15; j++)
			*(*(masindremove + i) + j) = 0;
	kolback = 0;
	memset(tripback, 0, sizeof(int) * 40);
	int dmin[V]; // минимальное расстояние
	memset(dmin, 0, sizeof(int) * V);
	int visited[V]; // посещенные вершины
					//memset(visited, 0, sizeof(int) * V);

	int temp;
	int minindex, min;
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
	//Инициализация вершин и расстояний
	for (int i = 0; i<n; i++)
	{
		dmin[i] = 10000;
		visited[i] = 1;
	}

	dmin[start] = 0; //от какой вершины
					// Шаг алгоритма
	do {
		minindex = 10000;
		min = 10000;
		for (int i = 0; i<n; i++)
		{ // Если вершину ещё не обошли и вес меньше min
			if ((visited[i] == 1) && (dmin[i]<min))
			{ // Переприсваиваем значения
				min = dmin[i];
				minindex = i;
			}
		}
		// Добавляем найденный минимальный вес к текущему весу вершины и сравниваем с текущим минимальным весом вершины
		if (minindex != 10000)
		{
			for (int i = 0; i<n; i++)
			{
				if (cost[minindex][i] > 0)
				{
					temp = min + cost[minindex][i];
					if (temp < dmin[i])
					{
						dmin[i] = temp;
					}
				}
			}
```

```
                visited[minindex] = 0;
        }
} while (minindex < 10000);

// Восстановление пути
int ver[V]; // массив посещенных вершин

ver[0] = end + 1; // начальный элемент - конечная вершина
int k = 1; // индекс предыдущей вершины
int weight = dmin[end]; // вес конечной вершины

while (end != start) // пока не дошли до начальной вершины
{
        for (int i = 0; i < n; i++) // просматриваем все вершины
                if (cost[end][i] != 0)   // если связь есть
                {
                        int temp = weight - cost[end][i]; // определяем вес пути из предыдущей вершины
                        if (temp == dmin[i]) // если вес совпал с рассчитанным
                        {               // значит из этой вершины и был переход
                                weight = temp; // сохраняем новый вес
                                end = i;       // сохраняем предыдущую вершину
                                ver[k] = i + 1; // и записываем ее в массив

                                k++;
                        }
                }
}
// Вывод пути (начальная вершина оказалась в конце массива из k элементов)
//printf("\nВывод кратчайшего пути\n");
int in = *index;
int in1 = *index;
in++;
//int kol = 0;
*kol = 0;
int b = 0;
SetConsoleTextAttribute(hConsole, (WORD)((0) | 10));
for (int i = k - 1; i >= 0; i--)
{
        if ((i != 0) && (i != (k - 1)))
        {
                trippr[(in)++] = ver[i] - 1; //выстраиваем маршрут
                *(kol) += 1;
        }
        if (*p1 == 1000)
        {
                tripback[b++] = ver[i] - 1;
                kolback++;
                if ((ver[i] - 1) == (n - 1))
                        printf("%d - %s", ver[i] - 1, cit[ver[i] - 1]);
                else printf("%d - %s", ver[i] - 1, cit[ver[i] - 1]);
        }
        //printf("%3d ", ver[i]);
}

*(kol) += 1;
//for (int b = 0; b < 2;b++)
//memset(masindremove, 0, sizeof(int)*n*2);
```

```
	int ind = 0;

	(*index)++;
	for (int a = (*index); a < (*index + (*kol - 1)); a++)
	{
		for (int j = 1; j <= np; j++)
			if (costpos[0][j] == (trippr[a]))
			{
				costpos[np + 1][j]++;
				/*if (costpos[np + 1][j] ==1) {*/

				/**(*(masindremove)+ind)= costpos[0][j];
				*(*(masindremove+1)+ind)= masindremove[1][ind++]++;*/
				masindremove[0][ind] = costpos[0][j];
				masindremove[1][ind++]++;
				//printf("%d", masindremove[1][0]);
				//(*p)++;
				(*p1)++;
				//}
				//else if (costpos[np + 1][j] != 0) {

				//	/**(*(masindremove)+ind) = costpos[0][j];
				//	*(*(masindremove + 1) + ind)++;
				//	ind++;*/
				//	masindremove[0][ind] = costpos[0][j];
				//	masindremove[1][ind++]++;
				//	(*p1)++;
				//
				//	break;
				//}

			}

	}

	*index = in;
	return masindremove;
}

void newmap()
{

	int k = 0;
	wchar_t masmap[15][30];
	WIN32_FIND_DATA FindFileData;
	HANDLE hFind = FindFirstFile(L"*.rtf", &FindFileData);
	//wcscpy(masmap[k], FindFileData.cFileName);
	k++;
	while (FindNextFile(hFind, &FindFileData))
	{
		wcscpy(masmap[k], FindFileData.cFileName);
		k++;
		//wprintf(FindFileData.cFileName);
	};

	system("cls");
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
```

```
int s;
wchar_t newmap[30];
printf("Введите название карты:\n");
wscanf(L"%s", newmap);
//wcscpy(newmap, L"map");
//wchar_t tex[100];
//swprintf_s(tex, L"%d", k + 1);
//wsprintf(tex, TEXT("%d"), k+1);
//wcscat(newmap, tex);
wcscat(newmap, L".rtf");
SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
printf("Сколько городов будет на карте?\n");
SetConsoleTextAttribute(hConsole, (WORD)((9) | 3));
scanf("%d", &s);

//remove("map2.txt");
//city = fopen("cities.txt", "w+");
file = _wfopen(newmap, L"w+");
fprintf(file, "%d\n", s);
char gor[30];
SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
printf("Введите города:\n");
SetConsoleTextAttribute(hConsole, (WORD)((9) | 3));
for (int i = 0; i < s; i++)
{
	printf("%d - ", i); if (i == 0) {
		while (getchar() != '\n');
		gets_s(gor);
		strcpy(cit[i], gor);
	}
	else
	{
		gets_s(gor);
		strcpy(cit[i], gor);
	}
	if ((i + 1) == s)
		fprintf(file, "%d - %s", i, gor);
	else fprintf(file, "%d - %s\n", i, gor);
}

memset(cost, 0, sizeof(int) * 11 * 11);

//remove("matrix.txt");
//file = fopen("matrix.txt", "w+");
//file = fopen("matrix.txt", "w+");
//fprintf(file, "%d", s);
SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
int r;
for (int i = 0; i<s; i++)
{
	cost[i][i] = 0;
	for (int j = i + 1; j<s; j++) {
		printf("Введите расстояние %s - %s: ", cit[i], cit[j]);
		scanf("%d", &r);
		cost[i][j] = r;
		cost[j][i] = r;
	}
```

```
	}

	for (int i = 0; i<s; i++)
	{
		fprintf(file, "\n");
		for (int j = 0; j<s; j++) {
			fprintf(file, "%d ", cost[i][j]);
		}
	}

	//fclose(city);
	wcscpy(nowmap, newmap);
	fclose(file);
}

void input(wchar_t in[30])
{
	system("cls");
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
	int i, j;

	memset(cost, 0, sizeof(int)*15 *15);
	memset(costf, 0, sizeof(int)*15*15);
	memset(costpos, 0, sizeof(int)*(17) *(17));
	memset(trip, 0, sizeof(int)*40);
	memset(trippr, 0, sizeof(int) * 40);
	memset(tripback, 0, sizeof(int) * 40);
	memset(posetit, 0, sizeof(int) * n);
	mintrip = 10000;
	koltrip = 1;

	file = _wfopen(in, L"r+");
	if (!file)
	{
		printf("Карта не загружена, следует создать новую\n");
		newmap();
	}

	char cf[30];
	int por = 0;
	fscanf(file, "%d", &kolcit);
	n = kolcit;
	fgets(cf, 30, file);
	int p = 0;
	for (int i = 0; i < kolcit; i++)
	{
		fflush(stdin);
		fseek(file, 4, SEEK_CUR);
		fgets(cit[por], 255, file);

		por++;
		p++;
	}



	for (int i = 0; i < kolcit; i++)
```

```
	{

		for (int j = 0; j < kolcit; j++)
		{
			fscanf(file, "%d", &cost[i][j]);
			//		if (cost[i][j]) masst[i]++;
			costf[i][j] = cost[i][j];
		}
	}
	fclose(file);

	SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
	//printf("\nЗагрузка карты прошла успешна \n");
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 4));
	//system("pause");
}

void choosemap()
{
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
	HWND hcon = GetConsoleWindow();


	system("cls");
	int k = 0;
	wchar_t masmap[15][30];
	WIN32_FIND_DATA FindFileData;
	HANDLE hFind = FindFirstFile(L"*.rtf", &FindFileData);
	wcscpy(masmap[k], FindFileData.cFileName);
	k++;
	while (FindNextFile(hFind, &FindFileData))
	{
		wcscpy(masmap[k], FindFileData.cFileName);
		k++;
		//wprintf(FindFileData.cFileName);
	};


	int kolpt = k;			//кол-во пунктов меню
	COORD *position = new COORD[kolpt];
	int punkt = 0;
	do
	{
		int xmax, ymax;
		// получение параметров окна
		PCONSOLE_SCREEN_BUFFER_INFO pwi = new CONSOLE_SCREEN_BUFFER_INFO;
		PWINDOWINFO pgwi = new WINDOWINFO;
		GetConsoleScreenBufferInfo(hConsole, pwi);
		GetWindowInfo(hcon, pgwi);
		xmax = pwi->dwSize.X;
		ymax = pwi->dwSize.Y;
		fflush(stdin);
		//printf("%s",map[0]);
		//puts(map[1]);
```

```
int y0 = 6;

for (int i = 0; i < kolpt; i++)
{
	position[i].X = (xmax - 7) / 2;
	position[i].Y = y0 + i;

}
SetConsoleTextAttribute(hConsole, (WORD)((0 | 15)));
system("cls");                          // очистка окна
char title[] = "Choose your map";
COORD pos;
pos.X = (xmax - strlen(title)) / 2;
pos.Y = 5;
SetConsoleCursorPosition(hConsole, pos);
puts(title);
SetConsoleTextAttribute(hConsole, (WORD)((3 | 1)));
//puts(map[0]);
//puts(map[1]);
for (int i = 0; i <kolpt; i++)
{
	SetConsoleCursorPosition(hConsole, position[i]);
	wprintf(masmap[i]);
	printf("\n");
}
SetConsoleTextAttribute(hConsole, (WORD)((3 | 11)));
SetConsoleCursorPosition(hConsole, position[punkt]); // выделение текущего пункта ярким цветом
wprintf(masmap[punkt]);
unsigned char ch;




do
{
	// обработка перемещения по меню клавишами со стрелками
	ch = getch();
	if (ch == 224)
	{
		ch = getch();
		switch (ch)
		{
		case 72:
			SetConsoleCursorPosition(hConsole, position[punkt]);
			SetConsoleTextAttribute(hConsole, (WORD)((3 | 1)));
			wprintf(masmap[punkt]);
			punkt--;
			if (punkt < 0) punkt = kolpt - 1;
			SetConsoleTextAttribute(hConsole, (WORD)((3 | 11)));
			SetConsoleCursorPosition(hConsole, position[punkt]);
			wprintf(masmap[punkt]); break;
		case 80:
			SetConsoleCursorPosition(hConsole, position[punkt]);
			SetConsoleTextAttribute(hConsole, (WORD)((3 | 1)));
			wprintf(masmap[punkt]);
			punkt++;
```

```
                                        if (punkt >(kolpt - 1)) punkt = 0;
                                        SetConsoleTextAttribute(hConsole, (WORD)((3 | 11)));
                                        SetConsoleCursorPosition(hConsole, position[punkt]);
                                        wprintf(masmap[punkt]); break;
                                }
                        }
                } while ((ch != 13) && (ch != 27));       // enter - выбор пункта меню
                if (ch == 27) goto tut;
                switch (punkt)
                {
                case 0:
                        wcscpy(nowmap, masmap[0]); input(masmap[0]);
                        //printf("первый файл"); system("pause");
                        break;
                case 1:
                        wcscpy(nowmap, masmap[1]); input(masmap[1]);
                        break;
                case 2:
                        wcscpy(nowmap, masmap[2]); input(masmap[2]);
                        break;
                case 3:
                        wcscpy(nowmap, masmap[3]); input(masmap[3]);
                        break;
                case 4:
                        wcscpy(nowmap, masmap[4]); input(masmap[4]);
                        break;
                case 5:
                        wcscpy(nowmap, masmap[5]); input(masmap[5]);
                        break;
                case 6:
                        wcscpy(nowmap, masmap[6]); input(masmap[6]);
                        break;
                case 7:
                        wcscpy(nowmap, masmap[7]); input(masmap[7]);
                        break;
                case 8:
                        wcscpy(nowmap, masmap[8]); input(masmap[8]);
                        break;
                case 9:
                        wcscpy(nowmap, masmap[9]); input(masmap[9]);
                        break;
                }

        } while (punkt >= kolpt);

tut:
        system("cls");
}



void start() {
        system("cls");
        HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

        SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
        printf("Загружены следующие города: \n");
```

```
	for (int i = 0; i < kolcit; i++)
	{

		printf("%d - %s", i, cit[i]);

	}
	printf("\n\n");

repeat:
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
	printf("Введите количество городов, которые вы хотите посетить: \n");
	SetConsoleTextAttribute(hConsole, (WORD)((9) | 3));
	scanf("%d", &np);
	if (np < 2) goto repeat;
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
	printf("Выберите города, при условии, что первый город будет тот, в котором вы находитесь \n");

	char c[30];

	SetConsoleTextAttribute(hConsole, (WORD)((9) | 3));

	for (int i = 0; i < np; i++)
	{
		scanf("%d", &posetit[i]);
	}
	int in = 1, jn = 1;
	for (int i = 0; i < np; i++)
	{
		costpos[i + 1][0] = posetit[i];
		costpos[0][i + 1] = posetit[i];
	}

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

		if ((costpos[in][0]) == i)
		{
			costpos[in][in] = 0;
			for (int c = 1; c <= np; c++)
			{
				costpos[in][c] = cost[i][posetit[jn - 1]];
				jn++;
			}
			in++;
			if (in == np + 1)break;
			jn = 1;
			i = -1;
		}
	for (int i = 0; i <np; i++)
		costpos[in][i] = 0;
	/*file = fopen("costposetit.txt", "w+");
	for (int i = 0; i <= np; i++)
	{
		fprintf(file, "\n");

		for (int j = 0; j <= np; j++)
			fprintf(file, "%d ", costpos[i][j]);
	}*/
```

```
	fclose(file);
	printf("\n");
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 4));
	system("pause");

}

void searchmintrip(int st, int en, int rasst, int *indextrip, int *proidput, int *skud, int **masindrem) //skud - сколько удалить промежуточных точек ; proidenput - для восстановления маршрута
{

	int **masind;
	masind = new int*[2];
	for (int i = 0; i < 2; i++)
		*masind = new int[15]; //для удаления посещенных, хранит индексы
	int i, j;
	int pos;
	trippr[*indextrip] = en; //посещенная вершина
								/*if (trippr[1] == 3)
								printf("");*/
	rasst += costf[st][en]; //
	st = en; //делаем конечную точку из конечной в начальную для дальнейшего движения
	int pornom = 0; //порядковый номер стартовой точки в посещении
	for (int b = 0; b < np; b++)
		if (st == posetit[b])pornom = b;
	costpos[np + 1][pornom + 1]++; //для цикла с вариантами движения

	if (rasst > mintrip)
		goto here;
	pos = 0;
	for (int c = 1; c <= np; c++) {
		if (costpos[np + 1][c] != 0)
			(pos)++;
	}
	while (pos != np)
	{

		{

			{
				for (j = 1; j <= np; j++)
					if (costpos[np + 1][j] == 0)
					{
						int inexstolbca;
						for (int v = 1; v <= np; v++)
							if (st == costpos[v][0]) inexstolbca = v;

						int a, b;
						a = costpos[inexstolbca][0];
						b = costpos[0][j];
						if ((costpos[inexstolbca][j] == 0) || (costf[a][b] != costpos[inexstolbca][j]))
						{
							int *kD; //пройденное количество точек при перемещении от одной к другой для восстановления пути
							int nulkd = 0;
							kD = &nulkd;
```

```
                                                int nul1 = 0;
                                                int *skst = &nul1;
                                                masind = Dijkstra(st, costpos[0][j], indextrip, skst, kD); //sk
- сколько удалить вершин

                                                  //printf("%d", masind[1][0]);

                                                  //pos = 0;

                                                  /*for (int c = 1; c <= np; c++) {


                                                  if (costpos[np + 1][c] != 0)

                                                  (pos)++;

                                                  }*/
                                                searchmintrip(st, costpos[0][j], rasst, indextrip, kD, skst,
masind); //skts - сколько удалить степеней
                                        }
                                        else
                                        {

                                                (*indextrip)++;
                                                trippr[*indextrip] = costpos[0][j];
                                                int odin = 1;
                                                int *pa;
                                                pa = &odin;
                                                int null = 0;
                                                int *nanull = &null;
                                                searchmintrip(st, costpos[0][j], rasst, indextrip, pa, nanull,
masind);
                                        }

                                }

                        }
                        if (j> np) break;
                }

        }

here:
        if ((rasst < mintrip) && (pos == np)) //проверяем, является ли этот путь наилучшим
        {
                mintrip = rasst;
                memcpy(trip, trippr, sizeof(trippr));
                koltrip = *indextrip + 1;
        }
        for (int v = 0; v < *proidput; v++) { //для возвращения на предыдущую вершину

                trippr[*(indextrip)-v] = 0;
        }

        *indextrip -= *proidput; //изменения индекса для построения дальнейшего маршрута
        for (int a = 0; a < *skud; a++) //вычитаем степени посещенных вершин на данном этапе
                for (int j = 1; j <= np; j++)
```

```
		{
			//if (masind[0][a] == costpos[0][j])
			if (masindrem[0][a] == costpos[0][j])
			{

				/*costpos[np + 1][j] = *(*(masind + 1) + a);
				*(*(masind + 1) + a)=0;
				*/
				costpos[np + 1][j] -= masindrem[1][a];
				masindrem[1][a] = 0;
			}

		}
	costpos[np + 1][pornom + 1]--; //вычитаем степень посещенной конечной вершины

//pos = 0;
//for (int c = 1; c <= np; c++) { //высчитываем количество посещенных точек
//
//	if (costpos[np + 1][c] != 0)
//		(pos)++;
//}

}

void createmap(int wh)
{
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
	HWND hWnd = GetConsoleWindow();
	HDC hDC = GetDC(hWnd);


	towns *tow = new towns[n];
	int de = 0;
	//puts(cit[0]);
	for (int i = 0; i < kolcit; i++)
	{
		int w = 250 * cos(de*3.1415926 / 180);
		int h = 250 * sin(de*3.1415926 / 180);
		int kolstr = strlen(cit[i]);


		WCHAR  tex[100];
		wsprintf(tex, TEXT("%d"), i);



		TextOut(hDC, WIDTH / 2 + (w)+20 * cos(de*3.1415926 / 180), HEIGHT / 2 + (h)+20 * sin(de*3.1415926 / 180), tex, 2);

		de += 360 / kolcit;
		tow[i].x = WIDTH / 2 + w;
		tow[i].y = HEIGHT / 2 + h;
		///system("pause");
		if (wh == 0) {
			printf("%d - ", i); puts(cit[i]);
		}
```

```
}

int movx, movy;
int tox, toy;

//DeleteObject(hPen);
//SetTextColor(hDC, RGB(0, 0xFF, 0));
//if (wh == 1) {
for (int i = 0; i < n; i++)
{

	for (int j = i + 1; j < n; j++) {
		if (cost[i][j] != 0)
		{
			int a, b, c;
			if (wh == 0) {
				a = rand() % 256;
				b = rand() % 256;
				c = rand() % 256;
			}
			else
			{
				a = 90;
				b = 94;
				c = 90;

			}
			HPEN hPen = CreatePen(PS_SOLID, 3, RGB(a, b, c));
			HPEN hOldPen = (HPEN)SelectObject(hDC, hPen);
			movx = tow[i].x;
			movy = tow[i].y;
			MoveToEx(hDC, movx, movy, NULL);
			tox = tow[j].x;
			toy = tow[j].y;
			LineTo(hDC, tox, toy);
			DeleteObject(hPen);
		}
	}
}
//}
//else
//{
float a = 9.0;
float b = 33.0;
float c = 7.0;
float as, bs, cs;
if (wh == 1) {
	as = (65 - 9) / koltrip;
	bs = (255 - 33) / koltrip;
	cs = (45 - 7) / koltrip;
}
else if (wh == 2)
{
	as = (65 - 9) / kolback;
	bs = (255 - 33) / kolback;
	cs = (45 - 7) / kolback;
}
```

```
	if (wh == 1) {
		for (int i = 0; i < (koltrip - 1); i++)
		{
			for (int j = 0; j < n; j++)
				if (trip[i] == j)
				{
					Sleep(500);
					a = round(a + as);
					b = round(b + bs);
					c = round(c + cs);
					HPEN hPen = CreatePen(PS_SOLID, 5, RGB(a, b, c));
					HPEN hOldPen = (HPEN)SelectObject(hDC, hPen);
					movx = tow[j].x;
					movy = tow[j].y;
					MoveToEx(hDC, movx, movy, NULL);
					tox = tow[trip[i + 1]].x;
					toy = tow[trip[i + 1]].y;
					LineTo(hDC, tox, toy);
					DeleteObject(hPen);
				}
		}
	}
	else if (wh == 2)
	{

		for (int i = 0; i < (kolback - 1); i++)
		{
			for (int j = 0; j < n; j++)
				if (tripback[i] == j)
				{
					Sleep(500);
					a = round(a + as);
					b = round(b + bs);
					c = round(c + cs);
					HPEN hPen = CreatePen(PS_SOLID, 5, RGB(a, b, c));
					HPEN hOldPen = (HPEN)SelectObject(hDC, hPen);
					movx = tow[j].x;
					movy = tow[j].y;
					MoveToEx(hDC, movx, movy, NULL);
					tox = tow[tripback[i + 1]].x;
					toy = tow[tripback[i + 1]].y;
					LineTo(hDC, tox, toy);
					DeleteObject(hPen);
				}
		}
	}


	//}
	//DeleteObject(hPen);
	delete[]tow;
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 4));
	if (wh == 0)system("pause");
}

void ouput()
{
```

```
	int *a;
	int nul = 0;
	a = &nul;
	system("cls");
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

	int por = 0;

	SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
	printf("Маршрут из города:");
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 10));
	printf("%s", cit[trip[0]]); SetConsoleTextAttribute(hConsole, (WORD)((0) | 15)); printf("в ");
	if (np>2)
	{
		printf("города: \n");
		SetConsoleTextAttribute(hConsole, (WORD)((0) | 10));
		for (int c = 1; c < np; c++)
			if (posetit[c] == (n - 1))
				printf("*%s\n", cit[posetit[c]]);
			else printf("*%s", cit[posetit[c]]);
			SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
			printf("построен \n\n");
			system("pause");
			system("cls");
			printf("Следуйте в указанном порядке:\n");
			SetConsoleTextAttribute(hConsole, (WORD)((0) | 10));
			for (int i = 0; i < koltrip; i++) {
				if (trip[i] == (n - 1))
					printf("%d - %s\n", trip[i], (cit[trip[i]]));
				else printf("%d - %s", trip[i], (cit[trip[i]]));
			}

	}
	else {
		SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
		printf("город: "); SetConsoleTextAttribute(hConsole, (WORD)((0) | 10));
		for (int c = 1; c < np; c++)
			if (posetit[c] == (n - 1))
				printf("*%s\n", cit[posetit[c]]);
			else printf("*%s", cit[posetit[c]]);
			SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
			printf("построен\n\n");
			system("pause");
			system("cls");
			printf("Следуйте в указанном порядке:\n"); SetConsoleTextAttribute(hConsole,
(WORD)((0) | 10));
			for (int i = 0; i < koltrip; i++) {
				if (trip[i] == (n - 1))
					printf("%d - %s\n", trip[i], (cit[trip[i]]));
				else printf("%d - %s", trip[i], (cit[trip[i]]));
			}
	}
	createmap(1);
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
	printf("\n\nМинимальное расстояние:"); SetConsoleTextAttribute(hConsole, (WORD)((0 | 6)));
printf("%d\n", mintrip);
	int otvet;
```

```
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
	printf("\nВы хотите проложить путь \nдо начальной точки отправления?\nОтветьте 1-ДА; 0-НЕТ\n");
	SetConsoleTextAttribute(hConsole, (WORD)((9) | 3));
	scanf("%d", &otvet);
	printf("\n");
	if (otvet)
	{
		system("cls");
		int k1 = 0, k2 = 1000;
		int *nak1 = &k1, *nak2 = &k2;
		int **p = Dijkstra(trip[koltrip - 1], trip[0], nak1, nak2, a);
		int a = trip[koltrip - 1];
		int b = trip[0];
		createmap(2);
		SetConsoleTextAttribute(hConsole, (WORD)((0) | 15));
		printf("\nРасстояние на обратную дорогу составляет:");
		SetConsoleTextAttribute(hConsole, (WORD)((0 | 6)));
		printf("%d\n", costf[a][b]);

	}
	printf("\n");
	SetConsoleTextAttribute(hConsole, (WORD)((0) | 4));
	system("pause");

}
void main()
{
	SetConsoleOutputCP(1251);
	SetConsoleCP(1251);
	srand(time(NULL));
	wcscpy(nowmap, L"map1.rtf");
	/*file = fopen("masmap.txt", "r+");
	fscanf(file, "%d", &kolmap);
	char f[30];
	fgets(f, 10, file);
	fgets(map[0], 30, file);
	strcpy(nowmap, map[0]);
	fclose(file);
	puts(nowmap);*/


	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
	HWND hcon = GetConsoleWindow();
	CONSOLE_FONT_INFO cfi;
	GetCurrentConsoleFont(hConsole, false, &cfi);
	COORD fs = GetConsoleFontSize(hConsole, cfi.nFont);
	PCONSOLE_FONT_INFOEX ccf = new CONSOLE_FONT_INFOEX;
	(*ccf).dwFontSize.X = 12;
	(*ccf).dwFontSize.Y = 20;
	(*ccf).nFont = 11;
	(*ccf).cbSize = sizeof(CONSOLE_FONT_INFOEX);
	ccf->FontWeight = 400;
	lstrcpyW((*ccf).FaceName, L"Lucida Console");
	(*ccf).FontFamily = FF_DONTCARE;
	bool b = SetCurrentConsoleFontEx(hConsole, false, ccf);
	fs = GetConsoleFontSize(hConsole, cfi.nFont);
	int n = 5;                              //кол-во пунктов меню
```

```
	COORD position[5];
	int punkt = 0;
	char names[5][100] = { "Загрузка имеющейся карты", "Задать новую карту дорог","Выбрать карту","Начать поиск оптимального маршрута","Выход" };
	int xmax, ymax;

	trippr[0] = cost[0][1]; //начальная точка маршрута
	int ia = 0;
	int p = 0;
	int odin = 1;
	int *naodin;
	int *pa;
	pa = &p;
	naodin = &odin;
	int *indextrip = &ia;
	mintrip = 10000;
	int **mas;
	mas = new int*[2];
	for (int i = 0; i < 2; i++)
		*mas = new int[15]; //для удаления посещенных, хранит индексы

	do
	{
		// получение параметров окна
		PCONSOLE_SCREEN_BUFFER_INFO pwi = new CONSOLE_SCREEN_BUFFER_INFO;
		PWINDOWINFO pgwi = new WINDOWINFO;
		GetConsoleScreenBufferInfo(hConsole, pwi);
		GetWindowInfo(hcon, pgwi);
		xmax = pwi->dwSize.X;
		ymax = pwi->dwSize.Y;

		int y0 = 8;
		for (int i = 0; i < n; i++)
		{
			position[i].X = (xmax - strlen(names[i])) / 2;
			position[i].Y = y0 + i;
		}
		SetConsoleTextAttribute(hConsole, (WORD)((0 | 15)));
		system("cls");                          // очистка окна
		char title[] = "Choose your trip";
		COORD pos;
		pos.X = (xmax - strlen(title)) / 2;
		pos.Y = 5;
		SetConsoleCursorPosition(hConsole, pos);
		puts(title);
		SetConsoleTextAttribute(hConsole, (WORD)((3 | 1)));
		for (int i = 0; i < 5; i++)
		{
			SetConsoleCursorPosition(hConsole, position[i]);
			puts(names[i]);
		}
		SetConsoleTextAttribute(hConsole, (WORD)((3 | 11)));
		SetConsoleCursorPosition(hConsole, position[punkt]); // выделение текущего пункта ярким цветом
		puts(names[punkt]);
		unsigned char ch;
```

```
		do
		{
			// обработка перемещения по меню клавишами со стрелками
			ch = getch();
			if (ch == 224)
			{
				ch = getch();
				switch (ch)
				{
				case 72:
					SetConsoleCursorPosition(hConsole, position[punkt]);
					SetConsoleTextAttribute(hConsole, (WORD)((3 | 1)));
					puts(names[punkt]);
					punkt--;
					if (punkt < 0) punkt = 4;
					SetConsoleTextAttribute(hConsole, (WORD)((3 | 11)));
					SetConsoleCursorPosition(hConsole, position[punkt]);
					puts(names[punkt]); break;
				case 80:
					SetConsoleCursorPosition(hConsole, position[punkt]);
					SetConsoleTextAttribute(hConsole, (WORD)((3 | 1)));
					puts(names[punkt]);
					punkt++;
					if (punkt > 4) punkt = 0;
					SetConsoleTextAttribute(hConsole, (WORD)((3 | 11)));
					SetConsoleCursorPosition(hConsole, position[punkt]);
					puts(names[punkt]); break;
				}
			}
		} while (ch != 13);		// enter - выбор пункта меню
		switch (punkt)
		{
		case 0:
			input(nowmap);  createmap(0);
			break;
		case 1:
			newmap();
			break;
		case 2:
			choosemap();
			break;
		case 3:
			input(nowmap); floid(); start(); searchmintrip(costpos[0][1], costpos[0][1], 0, indextrip, pa,
pa, mas); ouput();
			break;

		}

	} while (punkt != 4);


	system("cls");
}
```

Список используемых источников

1. IT-портал «Алгоритм Дейкстры нахождения кратчайшего пути» URL https://prog-cpp.ru/deikstra/
2. IT-портал «Алгоритм Флойда – Уоршелла» URL http://kvodo.ru/algoritm-floyda-uorshella.html
3. Wikipedia «Алгоритм Флойда — Уоршелла» URL https://ru.wikipedia.org/wiki/Алгоритм_Флойда_—_Уоршелла
4. Алгоритм Флойда URL http://khpi-iip.mipk.kharkiv.edu/library/datastr/book_sod/kgsu/din_0124.html