Московский энергетический институт

(Технический университет)

**Отчет по курсовой работе по дисциплине**

**Теория графов**

**«Проект Метро»**

 Выполнили: студенты группы А-13-07

Ахремцев Ярослав, Дудкина Анастасия, Захаров Данила

Проверил: доц. каф. ПМ Незнанов А.А.

**1.Интерфейс программы.**



Интерфейс программы представляет из себя окно с возможность выбора вершин и дальнейшего выбора алгоритма действий.

Как пользоваться программой:

1. Выбираем маршрут путем выбора вершин из списка.

2. Выбираем алгоритм вычислений и желаемый результат

3. Нажимаем на кнопку «Вычислить!»

4 Используем кнопку «Обнулить!» для очистки экрана от результатов и для начала нового выбора вершин.

**2.Реализация графа.**



Принадлежность линиям:

0 – пересадка;

1 – красная;

2 – коричневая;

3 – синяя;

Таблица задания графа (входной файл «Metro.txt»)

|  |  |  |
| --- | --- | --- |
| Ребра | Принадлежность линии | Время в пути (сек.) |
| 1 – 2  | 1 | 150 |
| 2 – 3  | 1 | 112 |
| 3 – 4  | 0 | 135 |
| 3 – 5  | 1 | 105 |
| 5 – 6  | 0 | 95 |
| 5 – 7  | 1 | 75 |
| 5 – 8  | 0 | 57 |
| 6 – 7  | 0 | 75 |
| 6 – 8  | 3 | 45 |
| 7 – 8  | 0 | 80 |
| 7 – 10  | 1 | 123 |
| 10 – 9  | 0 | 90 |
| 10 – 11  | 1 | 130 |
| 11 – 12  | 1 | 100 |
| 6 – 13  | 2 | 120 |
| 13 – 14  | 0 | 50 |
| 13 – 15  | 3 | 65 |
| 15 – 16  | 3 | 160 |
| 4 – 14  | 2 | 85 |
| 14 – 9  | 2 | 142 |
| 9 – 18  | 2 | 110 |
| 18 – 4  | 2 | 100 |
| 8 – 17  | 3 | 75 |
| 17 – 18  | 0 | 45 |
| 17 – 19  | 3 | 87 |
| 19 – 20  | 3 | 84 |

**3. Используемые классы.**

|  |  |
| --- | --- |
| 3.1 Класс вершин. | 3.2 Класс ребер. |
| class Vertex{public: int num; //номер вершины Edge \*Ed[4];//массив указателей на //инцидентные ребра}; | class Edge{public: Vertex \*v1, \*v2; //указатели на //смежные данному ребру вершины int nv1, nv2;// номера смежные //данному ребру вершины int line;// номер линии, которой //принадлежит данное ребро int time;//вес ребра (время) Vertex\* OtherVertex(Vertex\* v);// функция получения указателя на другую вершину //данного ребра}; |
| 3.3 Класс Графа. |
| class Graph{private: int D[20];//массив расстойний, где в элементе D[i-1] содержится растояние от источника до i int P[20];//массив педыдущих вершин, где в элементе P[i-1] содержится номер вершины, которая располодена в пути до i-1 int T[20];//массив номеров вершин, расположенные в массиве в такой же последовательности как и в пути int K[20];//массив номеров вершин, содержащий вершины пути(последовательно) для задачи коммивояжера int MKP[20];//массив номеров вершин, содержащий вершины пути(последовательно) для задачи минимального кол-ва пересадок int MTP[20];//массив номеров вершин, содержащий вершины пути(последовательно) для задачи минимального времени пересадок Vertex ver[20]; //массив вершин Edge edg[26];//массив ребер int kolver, koledg, alltimek; // количество вершин, количество ребер, время в //пути char name[81];//имя структуры bool used[20];//массив использованных вершин void Dijkstra(Vertex\* vbegin);//функция реализующая алгоритм Дейкстры int GetEdge(int n1, int n2); //возвращает вес ребра между вершинами n1 и n2 int minn(int n1, int n2);// возвращает минимум между n1 и n2 int kolel;// вспомогательная переменная, где хравнится количество элементов bool in(int x, int mas[], int n);// есть ли элемент x в массиве mas размера n bool allused(bool x[], int n);//все ли элементы логического массива x имеют //значении true public: int allves;// время в пути void init(char \*filename, char\* result);//процедура инициализации //графа void init(char \*filename);//процедура инициализации графа void mintime(char\* res);//функция вычисляющая минимальное вермя в пути//по заданным вершинам void GetTraektory(int ver1, int ver2);//получить траекторию пути //между двумя вершинами void GetString(int x[], char\* help);//получить строку для вывода в //главное окно void Kommivoyager(int SpisokV[], int n, char\* res);// функция //решающая задачу коммивояжера для заданных вершин void MinKolPeresadki(char\* res);// функция вычисляющая минимальное //количество пересадок на заданном пути int GetTime(int x[],int kolelem);//функция вычисляющая время пути по //вершинам, которые содержатся в массиве x void MinTimePeresadki(char\* res); //функция вычисляющая минимальное //время пересадок на заданном пути}; |

**4.Основные алгоритмы поиска.**

***Поиск кратчайшего пути***

|  |  |
| --- | --- |
| 4.1 Минимальное время в пути задается функцией “mintime”,Которая использует алгоритм Дейкстры |  |
| void Graph::mintime(char\* res){ int i=0; char help1[100]="Travel:\n"; char help2[12]=""; strcat(res,help1); strcpy(help1,""); allves=0; while (PV[i] && PV[i+1]){//пока в массиве элементы PV[i] и PV[i+1] больше 0 strcpy(help1,""); for(int j=0; j<20; j++) //обнуляем массив T T[j]=0; ***Dijkstra***(&ver[PV[i]-1]); //вызываем для вершины P[i] функцию реализуюущую алгоритм  //Дейкстры allves+=D[PV[i+1]-1];//прибавляем к общему времени пути время пути по ребру между  //вершинами PV[i] И PV[i+1] //получаем массив каждый элемент которого содержит вершину пути в порядке их прохождеие от источника к месту назначения GetTraektory(PV[i],PV[i+1]);//вызываем функцию высчитывающую траекторию от //вершины PV[i] И PV[i+1] if (i==0){ sprintf(help2,"%d",PV[i]); strcat(res,help2); } |  |

|  |
| --- |
| 4.2 Алгоритм Дейкстры |
| void Graph::Dijkstra(Vertex\* vbegin){ //функция требующая указатель на вершину с которой начинается маршрут set<int> S; //множество вершин , которые посетили  set<int> V;//множество всех вершин set<int> M; // множество не посещенных вершин (V-S) set<int> ::iterator m1; int min=0; for(int i=0; i<kolver; i++) //заполняем множество вершинами V.insert(i+1); S.insert(vbegin->num); //добавляем начальную вершину в множество S//////////////////////////////////////////////////////////////////////// for(int i=0; i<kolver; i++)//инициализация P (массив, где запоминается траектория P[i]=vbegin->num;//////////////////////////////////////////////////////////////////////// for(int i=0; i<kolver; i++)//инициализвция D (массив расстояний) D[i]=100000; //назначаем несуществующим ребрам вес 100000 for(int i=0; i< kolreb; i++) //ограничиваем число ребер для одной вершины//kolreb – максимальное число ребер инцидентных одной вершине if (vbegin->Ed[i]) //проверка существования ребра (Если 0- то ребра не сущ.) D[vbegin->Ed[i]->OtherVertex(vbegin)->num-1]=vbegin->Ed[i]->time;////////////////////////////////////////////////////////////////////////  for(int i=0; i<kolver-1; i++){ if (M.begin()==M.end()){// если множество M присвоить ему множество V M=V; M.erase(vbegin->num); }//////////////////////////////////////////////////////////////////////// m1=M.begin(); min=\*m1; for(m1=M.begin(); m1!=M.end(); ++m1)// поиск минимального D[w] if (D[\*m1-1]<D[min-1]) min=\*m1; S.insert(min);//добавляем в S минимальный элемент M.erase(min);//удаляем из М минимальный элемент///////////////////////////////////////////////////////////////////////// for (m1=M.begin(); m1!=M.end(); ++m1){//поиск для каждой вершины множества V\S=M нового D[v] int ge; ge=GetEdge(min,\*m1);//получить вес ребра между min m1 D[\*m1-1]=minn(D[\*m1-1], D[min-1]+ge);//найти минимум между //D[\*m1-1]и D[min-1]+ge if (D[min-1]+ge==D[\*m1-1]) // если время пути до вершины \*m1-1 равно //времени пути D[min-1]+ge  P[\*m1-1]=min; } }///////////////////////////////////////////////////////////////////////////} |

Принцип действия данного алгоритма легко можно найти в любом учебном пособии или в Интернете. Приведем лишь краткое описание этого алгоритма:

Алгоритм Дейкстры (Dijkstra) предназначен для решения задачи [Поиск кратчайших путей в графе](http://lib.custis.ru/index.php/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%BA%D1%80%D0%B0%D1%82%D1%87%D0%B0%D0%B9%D1%88%D0%B8%D1%85_%D0%BF%D1%83%D1%82%D0%B5%D0%B9_%D0%B2_%D0%B3%D1%80%D0%B0%D1%84%D0%B5).

Важным фактом, позволяющим исключить перебор, является то, что если у нас есть кратчайший путь от *v* до *w*, проходящий через вершину *y*, назовем его , то его первая часть от *v* до *y*, , тоже будет кратчайшим путем. Действительно, если бы это было не так, т.е. существовал бы путь длины меньшей, чем , то можно было бы улучшить оптимальный путь , заменив в нем на 

***Задача Коммивояжера***

|  |
| --- |
| 4.4 Алгоритм Ближайшего соседа |
| void Graph::Kommivoyager(int SpisokV[], int n, char\* res){ if (SpisokV[0]){ int i=0, mintime, numbermin; char help1[100]="Travel(salesman):\n"; char help2[12]=""; strcat(res,help1); strcpy(help1,""); alltimek=0; for(int j=0; j<20; j++) K[j]=0; bool used[20]; for(int i=0; i<kolver; i++) used[i]=1; for(int i=0; i<n; i++) used[SpisokV[i]-1]=0; int ref; used[SpisokV[0]-1]=1; numbermin=SpisokV[0]; while (!allused(used, n)){ mintime=0; ref=numbermin; K[i++]=ref; Dijkstra(&ver[ref-1]); for(int j=0; j<kolver; j++){ if (in(j+1, SpisokV, n) && !used[j] && ref-1!=j) { numbermin=j+1; mintime=D[numbermin-1]; } } for(int j=1; j<kolver; j++) if (D[j]<mintime && in(j+1, SpisokV, n) && !used[j] && ref-1!=j){ numbermin=j+1; mintime=D[j]; } alltimek+=mintime; used[numbermin-1]=1; } K[i++]=numbermin; Dijkstra(&ver[numbermin-1]); if (SpisokV[0] != numbermin) alltimek+=D[SpisokV[0]-1]; K[i]=SpisokV[0]; sprintf(help2,"%d",PV[0]); strcat(res,help2); GetString(K, help1); strcat(res,help1); sprintf(help2,"\nTime: %d\n",alltimek);  strcat(res,help2); }} |

Принцип действия:

Мы выбираем вершины, которые необходимо посетить , а алгоритм ищет минимальное время в пути для посещения выбранных вершин и возвращения в начальную .

Задача коммивояжера (В англоязычной литературе — [Traveling Salesman Problem](http://lib.custis.ru/index.php/Traveling_Salesman_Problem), сокращенно TSP), заключается в следующем:

Заданы *n* вершин  и попарные расстояния  между ними, являющиеся положительными целыми числами.

Чему равна наименьшая возможная длина кольцевого маршрута, проходящего по одному разу через все вершины? Иными словами, требуется найти минимально возможное значение суммы , где минимум берется по всем перестановкам  чисел *1,…,n*.

***Минимальное количество пересадок.***

|  |
| --- |
| 4.5 Алгоритм нахождения минимального количества пересадок  |
| void Graph::MinKolPeresadki(char\* res){ Graph grapht; int i=0; char help1[125]="Minimum number of changes:\n"; char help2[12]=""; strcat(res,help1); strcpy(help1,""); grapht.allves=0; grapht.init("metro.txt"); while (PV[i] && PV[i+1]){ strcpy(help1,""); for(int j=0; j<20; j++) grapht.T[j]=0; for(int z=0; z<grapht.koledg; z++) if (grapht.edg[z].line == 0) grapht.edg[z].time=1000; grapht.Dijkstra(&grapht.ver[PV[i]-1]); grapht.GetTraektory(PV[i], PV[i+1]); if (i==0){ sprintf(help2,"%d",PV[i]); strcat(res,help2); } grapht.GetString(grapht.T, help1); strcat(res,help1); grapht.allves+=GetTime(grapht.T, grapht.kolel); i++; } sprintf(help2,"\nTime: %d\n",grapht.allves); strcat(res,help2);} |

**Принцип действия:**

Принцип алгоритма заключается в том, что нужно присвоить большому числу все веса ребер(по сравнению с другим весами редер), которые являются пересадочными (то есть, по этим ребрам осуществляется пересадка). Тогда при выполнении функции, реализующей алгоритм Дейкстры, будут учитываться в первую очередь ребра по которым не осуществляется пересадка, и только если пути без пересадок нет, будут учитываться ребра по которым осуществляется пересадка, в итоге будет выбран путь с минимальным числом пересадок.

***Минимальное время пересадок.***

|  |
| --- |
| 4.6 Алгоритм нахождения минимального времени пересадок. |
| void Graph::MinTimePeresadki(char\* res){ Graph grapht; int i=0; char help1[125]="Minimum time of changes:\n"; char help2[12]=""; strcat(res,help1); strcpy(help1,""); grapht.allves=0; grapht.init("metro.txt"); while (PV[i] && PV[i+1]){ strcpy(help1,""); for(int j=0; j<20; j++) grapht.T[j]=0; for(int z=0; z<grapht.koledg; z++) if (grapht.edg[z].line != 0) grapht.edg[z].time=0; grapht.Dijkstra(&grapht.ver[PV[i]-1]); grapht.GetTraektory(PV[i], PV[i+1]); if (i==0){ sprintf(help2,"%d",PV[i]); strcat(res,help2); } grapht.GetString(grapht.T, help1); strcat(res,help1); grapht.allves+=GetTime(grapht.T, grapht.kolel); i++; } sprintf(help2,"\nTime: %d\n",grapht.allves); strcat(res,help2);} |

**Принцип действия:**

Принцип алгоритма заключается в том, что нужно присвоить нулю все веса ребер, которые не являются пересадочными (то есть, по этим ребрам не осуществляется пересадка). Тогда при выполнении функции, реализующей алгоритм Дейкстры, будут учитываться только веса пересадочных вершин, следовательно в результате будет найден путь с минимальным временем пересадок

**5. Тестирование программы.**



Тест №1

|  |  |  |  |
| --- | --- | --- | --- |
| Выбранные вершины | Алгоритм действий | Маршрут | Время |
| 20, 3 | Минимальное время в пути | 20-19-17-8-5-3 | 408 |
| 12, 20, 1 | Минимальное время в пути | 12-11-10-9-18-17-19-20-19-17-8-5-3-2-1 | 1316 |
| 12,14,8,13 | Минимальное время в пути | 12-11-10-9-14-13-6-8-6-13 | 842 |
| 1,19,7,13,9,1 | Минимальное время в пути | 1-2-3-5-8-17-19-17-8-7-6-13-14-9-18-4-3-2-1 | 1822 |

Проверка №1

1. Выбираем вершины {20, 3} из вершины 20 до вершины 3 можно добраться 5 наиболее кратчайшими путями:

 I. 20-19-17-18-4-3=84+87+45+100+135=451

 **II. 20-19-17-8-5-3=84+87+75+57+105=408 (Верно. Минимальное время)**

 III. 20-19-17-18-9-14-4-3 =84+87+110+142+85+135=643

 IV. 20-19-17-18-9-10-7-5-3=84+87+45+110+90+123+75+105=719

 V. 20-19-17-8-6-13-14-4-3=84+87+75+45+120+50+85+135=681

2. Выбираем вершины {12,20,1} из вершины 12 до вершины 20 и до вершины 1 можно добраться 15-ю наиболее короткими путями:

 I. 12-11-10-7-8-17-19-20-19-17-18-4-3-2-1=1392

 II. 12-11-10-7-8-17-19-20-19-17-8-5-3-2-1=1349

III. 12-11-10-7-8-17-19-20-19-17-18-9-14-4-3-2-1=1629

IV. 12-11-10-7-8-17-19-20-19-17-18-9-10-7-5-3-2-1=1660

V. 12-11-10-7-8-17-19-20-19-17-8-6-13-14-4-3-2-1=1622

VI. 12-11-10-9-18-17-19-20-19-17-18-4-3-2-1=1359

**VII. 12-11-10-9-18-17-19-20-19-17-8-5-3-2-1=1316 (Верно. Минимальное время)**

VIII. 12-11-10-9-18-17-19-20-19-17-18-9-14-4-3-2-1=1596

IX. 12-11-10-9-18-17-19-20-19-17-18-9-10-7-5-3-2-1=1627

X. 12-11-10-9-18-17-19-20-19-17-8-6-13-14-4-3-2-1=1589

XI.12-11-10-9-14-13-8-17-19-20-19-17-18-4-3-2-1=1636

XII. 12-11-10-9-14-13-8-17-19-20-19-17-8-5-3-2-1=1593

XIII. 12-11-10-9-14-13-8-17-19-20-19-17-18-9-14-4-3-2-1=1873

XIV. 12-11-10-9-14-13-8-17-19-20-19-17-18-9-10-7-5-3-2-1=1904

XV. 12-11-10-9-14-13-8-17-19-20-19-17-8-6-13-14-4-3-2-1=1866

3. Выбираем вершины {12,14,8,13} из вершины 12 до вершины 14 затем до 8 и до13 можно добраться 2-мя наиболее короткими путями:

**I. 12-11-10-9-14-13-6-8-6-13=842 (Верно. Минимальное время)**

II.12-11-10-7-8-6-13-14-13-6-8-6-13=1028

4. Выбираем вершины {1,19,7,13,9,1} из вершины 1 до вершины 19 затем до 7 потом13 в 9 и обратно в 1 можно добраться 15 путями

*Дудкина допишет*

Тест №2

|  |  |  |  |
| --- | --- | --- | --- |
| Выбранные вершины | Алгоритм действий | Последовательность посещения | Время |
| 20, 3 | Задача Коммивояжера | 20-3-20 | 816 |
| 12, 20, 1 | Задача Коммивояжера | 12-20-1-12 | 2111 |
| 12,14,8,13 | Задача Коммивояжера | 12-8-13-14-12 | 1110 |
| 1,19,7,13,9,1 | Задача Коммивояжера | 1-7-13-9-19-1 | 1657 |

Проверка №2

1. Выбираем вершины {20, 3}. Результат очевиден.

2. Выбираем вершины {12,20,1}. Существует два варианта посещения вершин.

 **I. 12-20-1-12=2111 (Верно. Минимальное время)**

 **II. 12-1-20-12=2111(Верно. Минимальное время)**

3. Выбираем вершины {12,14,8,13}. Существует 6 вариантов посещения вершин.

I. 12-13-8-14-12=1354

II. 12-8-14-13-12=1210

III. 12-14-8-13-12=1354

IV. 12-13-14-8-12=1210

**V. 12-8-13-14-12=1110 (Верно. Минимальное время)**

**VI. 12-14-13-8-12=1110 (Верно. Минимальное время)**

4. Выбираем вершины {1,19,7,13,9,1}. Существует 15 вариантов посещения вершин.

I. 1-9-13-7-19-1=1822

II.1-13-9-7-19-1=1765

III. 1-9-13-19-7-1=1810

**IV. 1-13-9-19-7-1=1650 (Минимальное время)**

V. 1-9-7-13-19-1=1928

VI. 1-13-7-9-19-1=2035

VII. 1-7-9-13-19-1=1760

VIII. 1-9-7-13-19-1=1928

IX. 1-7-9-19-13-1=1756

X. 1-9-7-19-13-1=1921

XII. 1-13-19-9-7-1=1756

XIII. 1-13-19-7-9-1=1921

**XIV. 1-7-13-9-19-1=1657 ( Результат полученный алгоритмом неверный)**

XV. 1-7-13-19-9-1=1813

Тест №3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выбранные вершины | Алгоритм действий | Маршрут  | Время | Кол-во  |
| 20, 3 | Минимальное количество пересадок | 20-19-17-8-5-3 | 408 | 1 |
| 12, 20, 1 | Минимальное количество пересадок | 12-11-10-7-8-17-19-20-19-17-8-5-3-2-1 | 1349 | 2 |
| 12,14,8,13 | Минимальное количество пересадок | 12-11-10-9-14-13-6-8-6-13 | 842 | 2 |
| 1,19,7,13,9,1 | Минимальное количество пересадок | 1-2-3-5-8-17-19-17-8-7-6-13-14-9-18-4-3-2-1 | 1822 | 5 |

Проверка №3 (**Оранжевым** цветом выделены пересадки.)

1. Выбираем вершины {20, 3} из вершины 20 до вершины 3 можно добраться 5 наиболее кратчайшими путями:

 I. 20-19-**17-18**-**4-3**=84+87+45+100+135=451 (2 пересадки)

 **II. 20-19-17-8-5-3=84+87+75+57+105=408 ( 1 пересадка)**

 III. 20-19-**17-18**-9-14-**4-3** =84+87+110+142+85+135=643 (2 пересадки)

 IV. 20-19-**17-18**-**9-10**-7-5-3=84+87+45+110+90+123+75+105=719 (2 пересадки)

 V. 20-19-17-8-6-**13-14**-**4-3**=84+87+75+45+120+50+85+135=681 (2 пересадки)

2. Выбираем вершины {12,20,1} из вершины 12 до вершины 20 и до вершины 1 можно добраться 15-ю наиболее короткими путями:

 I. 12-11-10-***7-8*-**17-19-20-19-**17-18**-**4-3**-2-1=1392 (3 пересадки)

 **II. 12-11-10-7-8-17-19-20-19-17-8-5-3-2-1=1349 (2 пересадки)**

III. 12-11-10-**7-8**-17-19-20-19-**17-18**-9-14-**4-3**-2-1=1629 (3 пересадки)

IV. 12-11-10-**7-8**-17-19-20-19-**17-18**-**9-10**-7-5-3-2-1=1660 (3 пересадки)

V. 12-11-10-**7-8**-17-19-20-19-**17-8**-6-**13-14**-**4-3**-2-1=1622 (4 пересадки

VI. 12-11-**10-9**-**18-17**-19-20-19-**17-18**-**4-3**-2-1=1359 (4 пересадки)

VII. 12-11-**10-9**-**18-17**-19-20-19-17-**8-5**-3-2-1=1316 ( 3 пересадки)

VIII. 12-11-**10-9**-**18-17**-19-20-19-**17-18**-9-14-**4-3**-2-1=1596 (4 пересадки)

IX. 12-11-**10-9**-**18-17**-19-20-19-**17-18**-9-10-7-5-3-2-1=1627 ( 4 пересадки)

X. 12-11-**10-9**-**18-17**-19-20-19-17-8-6-**13-14**-**4-3**-2-1=1589 (4 пересадки)

XI.12-11-**10-9**-**14-13**-8-17-19-20-19-**17-18**-**4-3**-2-1=1636 (4 пересадки)

XII. 12-11-**10-9**-**14-13**-8-17-19-20-19-17-**8-5**-3-2-1=1593 (3 пересадки)

XIII. 12-11-**10-9**-**14-13**-8-17-19-20-19-**17-18**-9-14-4-3-2-1=1873 ( 3 пересадки)

XIV. 12-11-**10-9**-**14-13**-8-17-19-20-19-**17-18**-**9-10**-7-5-3-2-1=1904 (4 пересадки)

XV. 12-11-**10-9**-**14-13**-8-17-19-20-19-17-8-6-**13-14**-**4-3**-2-1=1866 (4 пересадки)

3. Выбираем вершины {12,14,8,13} из вершины 12 до вершины 14 затем до 8 и до13 можно добраться 2-мя наиболее короткими путями:

**I. 12-11-10-9-14-13-6-8-6-13=842 ( 2 пересадки)**

II.12-11-10-**7-8**-6-**13-14-13**-6-8-6-13=1028 (3 пересадки)

4. Выбираем вершины {1,19,7,13,9,1} из вершины 1 до вершины 19 затем до 7 потом13 в 9 и обратно в 1 можно добраться ХХ путями

*Дудкина допишет*

Тест №4

|  |  |  |  |
| --- | --- | --- | --- |
| Выбранные вершины | Алгоритм действий | Маршрут  | Время |
| 20, 3 | Минимальное время пересадок | 20-19-17-8-5-3 | 408 |
| 12, 20, 1 | Минимальное время пересадок | 12-11-10-7-5-8-17-19-20-19-17-8-5-3-2-1 | 1401 |
| 12,14,8,13 | Минимальное время пересадок | 12-11-10-9-14-13-6-8-6-13 | 842 |
| 1,19,7,13,9,1 | Минимальное время пересадок | 1-2-3-5-8-17-19-17-8-7-6-13-14-9-18-4-3-2-1 | 1822 |

Проверка №4

*Дудкина допишет*

**Заключение:**

-Программа работает при любых условиях, и выдает корректные результаты, что подтверждается нашими тестами.

- При решении задачи Коммивояжера программа не всегда вычисляет оптимальный результат, ввиду того что решение задачи коммивояжер реализовано с помощью алгоритма ближайшего соседа, который является эвристическим.

-Вывод результатов ограничен длиной строки в 500 символов для 1 вычисления и 10000 символов для всех вычислений.

-Также ограничено количество ребер и вершин. (23,20)

-Программа универсальна, в том плане, что может использовать любой исходный граф (заданный во входном файле «Metro.txt»).

-Так, как мы заранее задали наш исходный граф, интерфейс программы писался для конкретного графа (схемы метро), который вы могли видеть выше.

- Из минусов программы можно отметить невозможность отменны одной из выбранных вершин , при назначении пути (в дальнейших версиях программы этот недостаток будет устранен).

**Вывод:**

Проделав данную работы мы узнали про множество алгоритмов обхода графа, научились реализовывать алгоритмы Дейксты, Коммивояжера (ближайшего соседа), написали собственный алгоритм поиска минимального количества пересадок, и минимального времени пересадок. Сделали универсальную программу в плане использования ее для других графов. Также сделали ее в виде win приложения т.е. независящей от Стрина «ПСУНа». Был написан собственный интерфейс. Реализована своя структура представления графа.

**Авторский вклад в проект :**

*Ахремцев Ярослав*: реализация алгоритмов поиска минимального времени в пути, минимального количества пересадок, минимального времени пересадок, написание основного тела программы.

*Дудкина Анастасия*: реализация задачи Коммивояжера, тестирование программы, написание отчета,

*Захаров Данила*: реализация алгоритма Дейкстры, написание интерфейса программы, тестирование программы, написание отчета.

**Список используемой литературы:**

1. Левитин А.В. Алгоритмы: введение в разработку и анализ. – М. : Вильямс, 2006.- 576с.

2. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы.: Пер. с англ.: Уч. пос. – М. : Вильямс, 2000. – 384 с.

3. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы: построение и анализ, 2-е изд. – М. : Вильямс, 2005. – 1296 с.

4. Алексеев Е. Р. Программирование в Microsoft Visual C++ и на Turbo C++ Explore. - Издательство: НТ Пресс, 2007 г. - 352 стр.