ФГБОУ ВО «Ивановский государственный энергетический университет имени В.И. Ленина»
Факультет информатики и вычислительной техники
Кафедра программного обеспечения компьютерных систем

Курсовая работа

по дисциплине "Программирование и основы алгоритмизации"
на тему: "Разработка программы определения минимального маршрута"

Выполнил:

студент гр. 1-42 Тямин А.Д. _____ _____
(подпись) (дата)

Руководитель:

доцент каф. ПОКС Алыкова А.Л. _____ _____
(подпись) (дата)

Иваново, 2020

Задание на проектирование

Разработать программу, определяющую оптимальный маршрут поездки между заданными городами.

Исходные данные:

1. Карта автомобильных дорого некоторого региона, представленная списком городов с указанием расстояния между городами, связанными автомобильными дорогами;
2. Список городов, которые необходимо посетить.

Выходные данные:

1. Маршрут поездки;
2. Общая протяжённость маршрута.

Требования к функциональным характеристикам:

1. Считывание информации об автомобильных дорогах региона из текстового файла;
2. Удобный для пользователя выбор списка посещаемых городов;
3. Определение оптимального (минимального по протяжённости) маршрута посещения городов;
4. Наглядный вывод результатов.

Анализ задачи

Данная задача по своей сути является задачей коммивояжёра (или **TSP** от англ. *Travelling salesman problem*) — одной из самых известных задач комбинаторной оптимизации, заключающейся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город.

В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов.

Оптимизационная постановка задачи относится к классу NP-трудных задач, впрочем, как и большинство её частных случаев. Версия «decision problem» (то есть такая, в которой ставится вопрос, существует ли маршрут не длиннее, чем заданное значение k) относится к классу NP-полных задач. Задача коммивояжёра относится к числу трансвычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет (размер пространства поиска факториально зависит от количества городов).

Вариантов решения данной задачи – огромное множество. Среди них существуют наиболее простые и дающие лишь приближённый (хотя достаточно хороший) результат, такие как полный перебор, случайный перебор, жадные алгоритмы (метод ближайшего соседа, метод включения ближайшего города, метод самого дешёвого включения), метод минимального

остовного дерева, метод имитации отжига, а также различные модификации более эффективных методов: метод ветвей и границ и метод генетических алгоритмов, а также алгоритм муравьиной колонии.

Здесь стоит отметить, что все эффективные (сокращающие полный перебор) методы решения задачи коммивояжёра — методы эвристические. В большинстве эвристических методов находится не самый эффективный маршрут, а приближённое решение. Зачастую востребованы алгоритмы постепенно улучшающие некоторое текущее приближенное решение.

Также стоит сказать, что другие популярные алгоритмы, такие как алгоритм Дейкстры или алгоритм Беллмана-Форда, для данной задачи являются не самыми подходящими, так как первый находит кратчайшее расстояние между лишь двумя точками, у второй – кратчайшее расстояние от одной точки до всех остальных.

Описание алгоритма

Для решения данной задачи был выбран жадный алгоритм – метод ближайшего соседа. Алгоритм неплохо работает на симметричной матрице городов при маршрутах, похожих на кольцевые. Выбран был из-за относительной простоты в реализации и достаточной эффективности в нахождении маршрута, близкого к кратчайшему, а также высокой скорости работы даже для большого количества городов. Кроме того, данный алгоритм является доработанным, благодаря чему находит решение, более близкое к идеальному, при достаточно большом количестве городов.

Идея алгоритма ближайшего соседа основана на простом эвристическом правиле: если мы будем посещать ближайший пункт на каждом шаге, то маршрут получится довольно хорош в целом. Перед коммивояжером ставится задача посещать ближайший из ещё не посещённых пунктов. В алгоритме существуют два важных ограничения:

1. Недопущение повторного заезда в пункт. Оно связано с необходимостью (по условию задачи) нахождения гамильтонова цикла, то есть цикла, в котором все пункты посещаются единожды.
2. Недопущение возврата преждевременного возврата в исходный пункт. Этот запрет вводится для предотвращения преждевременного зацикливания маршрута, которое повлечёт за собой неправильную работу алгоритма.

Для того, чтобы метод давал более точные результаты, можно ввести в алгоритм небольшое усовершенствование, которое позволит выбирать решение из большего числа маршрутов для одной матрицы расстояний. Возможность этого усовершенствования возможна в связи с тем, что объектом исследования является задача коммивояжера, решение которой подразумевает нахождение замкнутого цикла. Идея заключается в том, чтобы запускать алгоритм, циклически изменяя начальный пункт.

Усовершенствованный метод ближайшего соседа по результатам тестирования на различных матрицах размерностями от 10 до 150 даёт результаты в среднем на 16% лучше обычного метода. При малых размерностях (до 10-15 рассматриваемых в задаче пунктов) метод может не давать улучшения, ввиду малого количества рассматриваемых цепочек. При размерностях матрицы от 50 относительный выигрыш стабилизируется в окрестности 16%. Для того, чтобы дополнительно увеличить вероятность получения более выгодного маршрута, можно рассматривать также и обратные к цепочкам маршрута (просто изменяя направление обхода цикла). В рамках данной реализации программы это улучшение не производилось. В связи с ничтожным временем работы исходного метода увеличение количества операций в 2n на итоговом времени работы программы не сказывается критически.

Математическая модель

Вносим в программу список городов размером n. Создаём матрицу размером nxn (целочисленный двумерный массив). В эту матрицу из файла читаем элементы, обозначающие расстояния между городами, которые ввёл пользователь. Затем каждый пункт передаём в алгоритм в качестве стартового. Запоминаем путь, который был составлен алгоритмом (в данном случае запоминаются именно индексы городов из матрицы). Вместе с этим запоминаем длину маршрута. Затем отбираем кратчайшую длину, получавшуюся в результате работы алгоритма, и в соответствии с ней выводим путь из городов, который соответствует этой длине (для вывода индексы городов преобразуются в их имена). На этом основная работа закончена.

Метод решения

Для рассмотрения решения сразу перейдём к моменту, когда у нас уже имеется построенная матрица расстояний для списка городов, которые пользователь уже ввёл. В качестве примера возьмём матрицу размером 7x7. Расстояние в город из этого же города нулевое.

Табл. 1 Матрица для демонстрации работы алгоритма

N	0	1	2	3	4	5	6
0	0	75	64	69	60	63	29
1	66	0	35	59	96	80	74
2	58	30	0	26	110	111	57
3	64	53	21	0	14	15	86
4	51	94	109	8	0	45	95
5	57	70	106	6	35	0	76
6	22	71	55	76	92	67	0

Для данной матрицы алгоритм ближайшего соседа без модификации мы получим ответ: 0-6-2-3-4-5-1-0, и длина этого маршрута составит 29+55+26+14+45+70+66 = 305 (км в данном случае).

Теперь изменим отправную точку на 1. Точно по такому же принципу получили цепочку 1-2-3-4-5-0-6-1, длина которой равна 277. Получим и остальные маршруты.

Табл. 2 Образованные маршруты.

Новые цепочки	Расстояние
1-2-3-4-5-0-6-1	277
2-3-4-5-0-6-1-2	277
3-4-5-0-6-2-1-3	289
4-3-5-0-6-2-1-4	290
5-3-4-0-6-2-1-5	265
6-0-4-3-5-1-2-6	267

Можно увидеть, что для приведённой матрицы маршрут, предложенный модифицированным методом, занимает на 40 км меньше (относительный выигрыш – 15%).

Примечание: нетрудно заметить, что по сути, раз маршрут в задаче является кольцевым, то цепочки из городов с индексами от 1 до 6 являются решениями для того же исходного пункта 0. Например, в данном случае цепочку 1-2-3-4-5-0-6-1 можно преобразовать в 0-6-1-2-3-4-5-0, однако в данной работе такое преобразование не производилось.

Описание подпрограмм

Далее для всех функций описываются назначение и параметры.

1. Функция SetLanguage.

Назначение: установка кодировки Windows-1251 в поток ввода и вывода для отображения русского текста.

Параметры: -

2. Функция SetTypeLucidaConsole.

Назначение: установка консольного шрифта Lucida Console для работы с кодировкой Windows-1251.

Параметры: -

3. Функция getDirections.

Назначение: построить маршрут.

Параметры: -

4. Функция showWays.

Назначение: вывести маршруты из файла.

Параметры: -

5. Функция addWay.

Назначение: добавить маршрут между двумя городами

Параметры: -

6. Функция delWay.

Назначение: удалить маршрут

Параметры: -

7. Функция FindShortestWay.

Назначение: найти кратчайший путь путём применения к матрице расстояний жадного алгоритма ближайшего соседа.

Параметры:

1. Index – индекс города, он же является индексом строки, в которой ведётся поиск наименьшего кратчайшего расстояния.

8. Функция Continue.

Назначение: задержать вывод в конце каждого пункта меню.

Параметры: -

Описание основных алгоритмов

1. getDirections

Пользователь через запятую без пробелов и других знаков вводит названия городов в единую строку. Затем из строки извлекаются названия городов, считая, что запятая является разделителем. Все названия записываются в контейнер строкового типа данных. Из файла считываем весь список доступных городов (читаем запись и выделяем условный пункт "Откуда"). Проверяем, все ли города из введённого списка доступны. Создаём матрицу размерами nxn, а также булевый массив размером n для меток посещения городов. Из файла для каждого города из списка в соответствующую ячейку в матрице ставим считанное расстояние (между одним и тем же городом - 0). Затем для каждого города в матрице мы запускаем алгоритм нахождения кратчайшего пути – алгоритм ближайшего соседа (описывается далее в FindShortestWay, здесь пусть просто сделает свою работу). Данный алгоритм вернёт нам индекс последнего города, который был посещён. Прибавляем расстояние от этого последнего посещённого города до города, с которого мы начали своё движение. Во время работы алгоритм изменил заранее созданный контейнер (в него были записаны индексы посещённых городов), туда же вносим последний посещённый город. В другой контейнер вносим посчитанную длину текущего маршрута. Затем обнуляем рассчитанную длину и контейнер с метками посещённых городов, и повторяем действия выше уже для другого города. Находим кратчайшую длину, пока ищем, запоминаем индекс цепочки, с которой в контейнере начинаются нужные нам города. Затем в соответствии с индексами городов в матрице выводим их названия, а затем длину рассчитанного минимального пути.

2. showWays

Производим чтение записей из файла travels.bin, и при помощи форматированного вывода показываем из в консоль.

3. addWay

Создаём контейнер для хранения всех поездок. Просим пользователя ввести запись: запрашиваем пункт отправления, пункт назначения, расстояние. Создаём запись, в которой меняем местами пункты. Обе записи добавляем в контейнер. Затем из файла читаем все уже хранящиеся записи и вносим их в контейнер. Сортируем записи в лексикографическом порядке. Закрываем файл. Удаляем файл. Создаём новый файл. Записываем все записи из контейнера в файл. Получили файл с двумя добавленными записями.

4. delWay

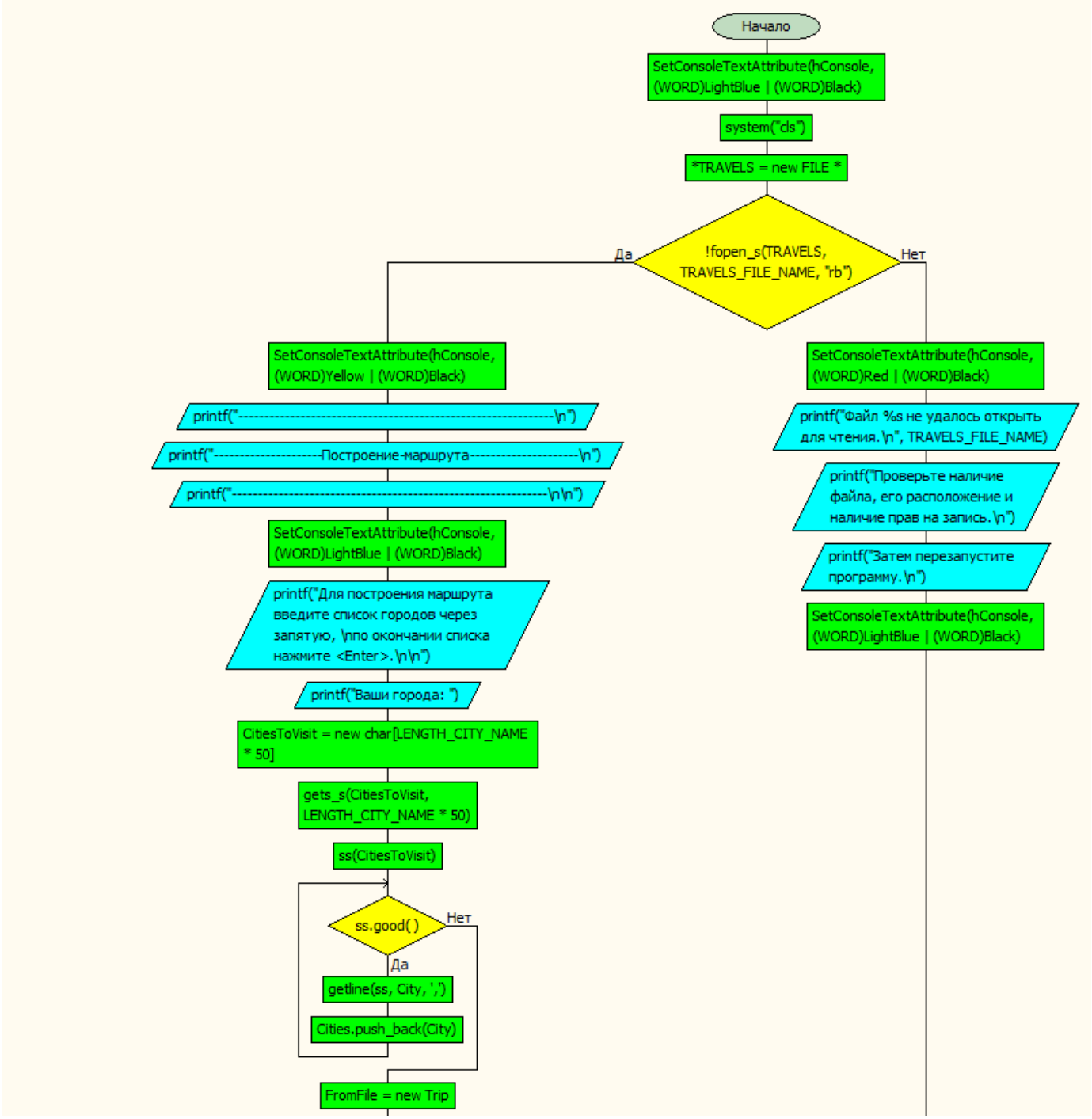
Создаём контейнер для хранения всех поездок без удалённых. Просим пользователя ввести запись: запрашиваем пункт отправления, пункт назначения, расстояние. Создаём запись, в которой меняем местами пункты. Из файла читаем записи. В контейнер добавляем только те записи, которые не равны двум записям для удаления. Таким образом, в контейнере остаются все записи, кроме удалённых. Закрываем файл. Удаляем файл. Создаём новый файл. Записываем все записи из контейнера в файл. Получили файл с двумя удалёнными записями.

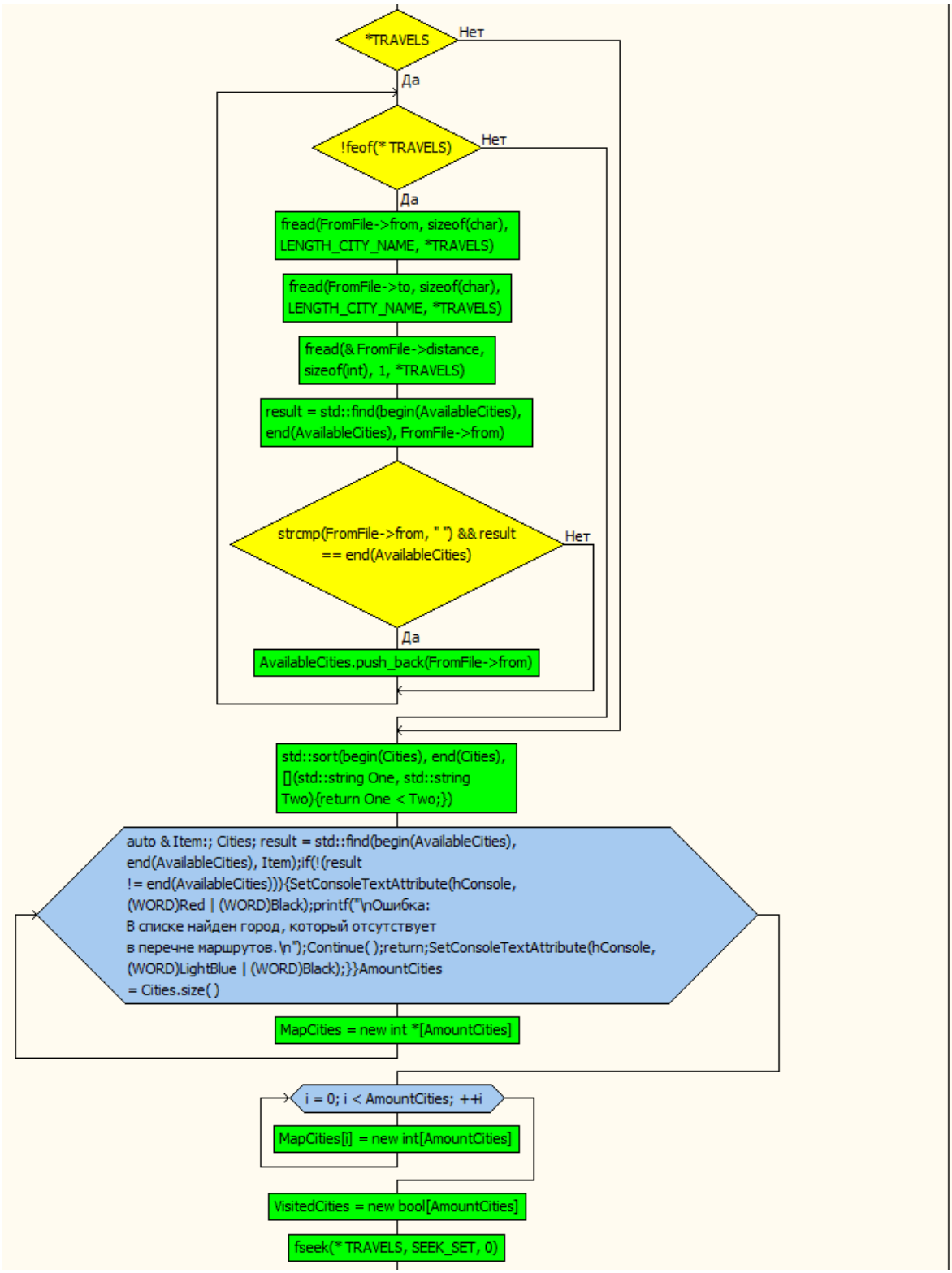
5. FindShortestWay

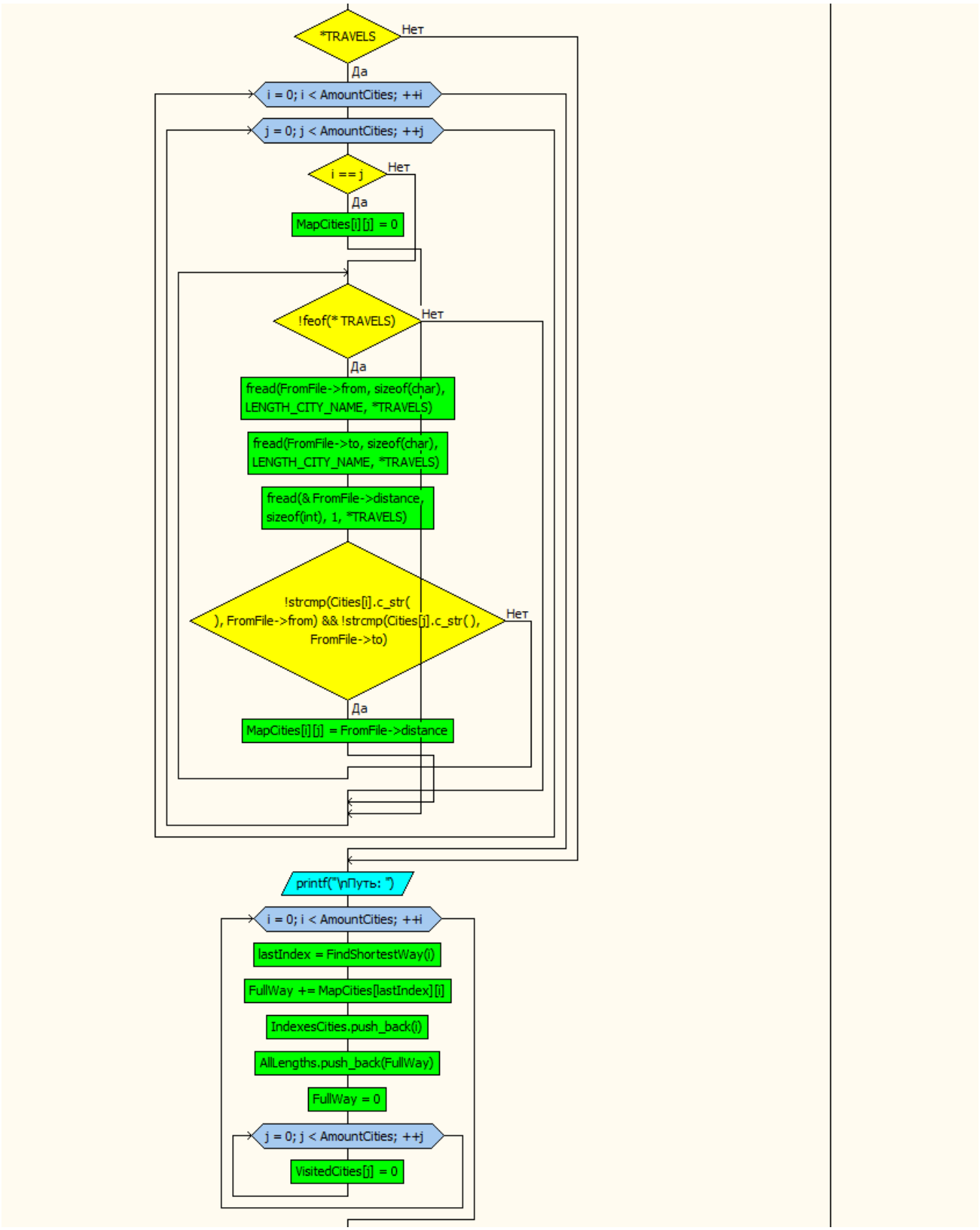
Данная функция является рекурсивной. На вход поступает индекс города, он же в данном случае индекс строки, в которой будет осуществляться поиск ближайшего города. В заранее созданный контейнер мы помещаем индекс-параметр функции и отмечаем в другом булевом массиве, что мы здесь уже побывали (ставим 1). Для минимума объявляем переменную, которую инициализируем большим числом (числовым аналогом бесконечности). Затем идём по всем строкам в матрице и находим город, до которого расстояние наименьшее, и запоминаем его индекс. Если мы модифицировали значение минимума (оно не бесконечно), то мы ещё раз идём в эту функцию, передав ей индекс города, до которого мы нашли кратчайшее расстояние, а если нет, то выходим из рекурсии, вернув индекс последнего посещённого города.

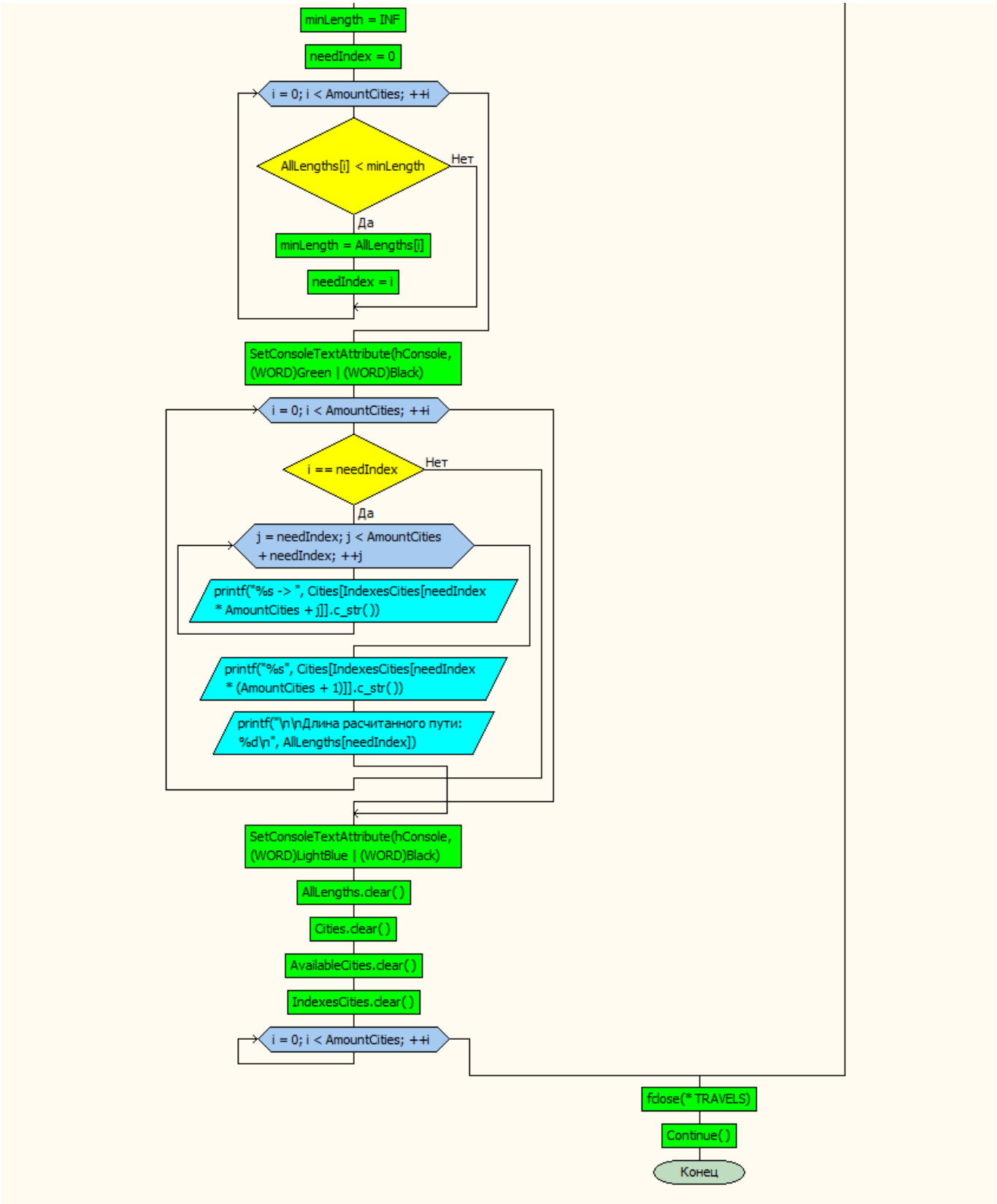
Блок-схемы основных алгоритмов

1. getDirections



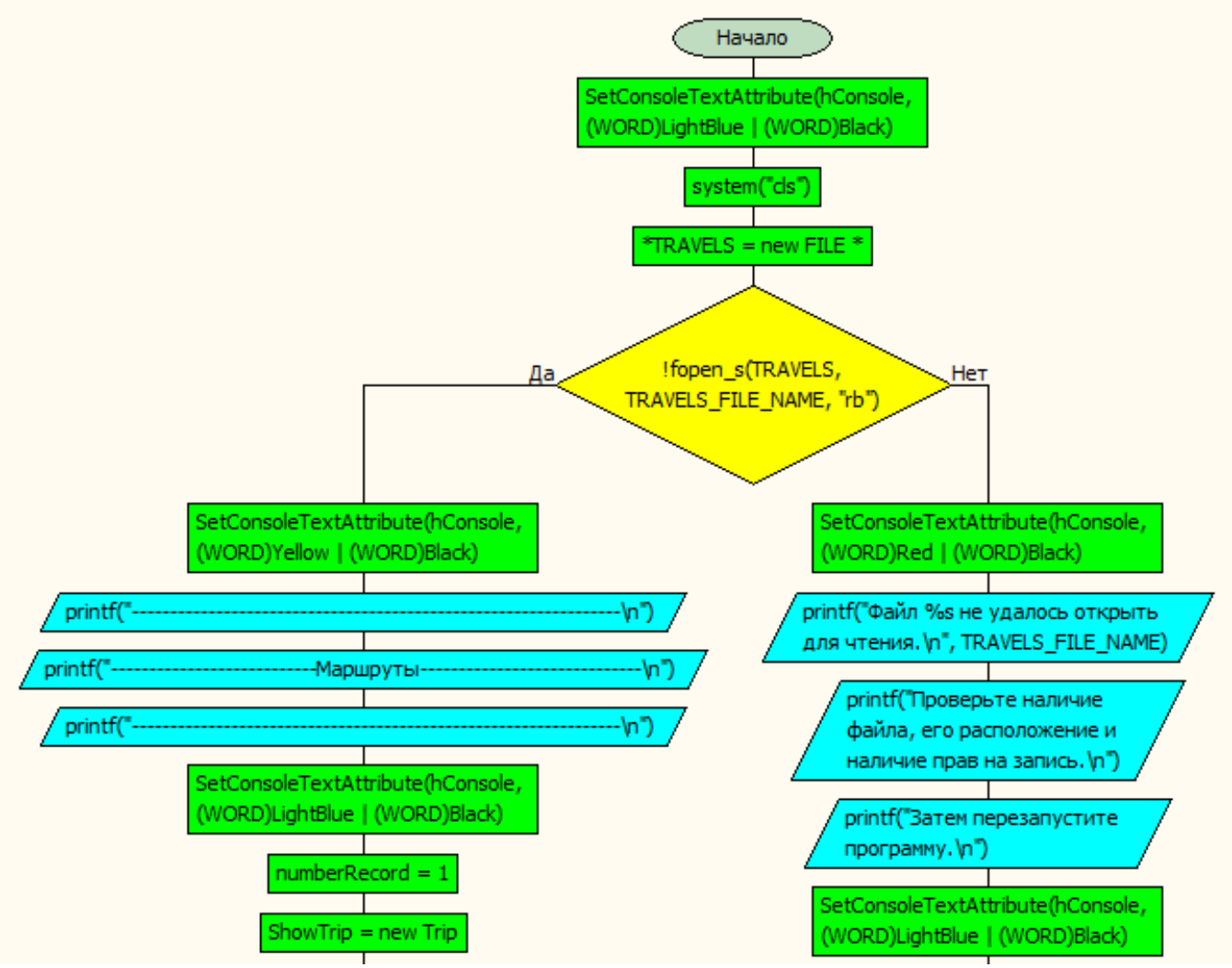

*TRAVELS
Нет
Да
!feof(* TRAVELS)
Нет
Да
fread(FromFile->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(FromFile->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(& FromFile->distance, sizeof(int), 1, *TRAVELS)
result = std::find(begin(AvailableCities), end(AvailableCities), FromFile->from)
strcmp(FromFile->from, " ") && result == end(AvailableCities)
Нет
Да
AvailableCities.push_back(FromFile->from)
std::sort(begin(Cities), end(Cities), [](std::string One, std::string Two){return One < Two;})
auto & Item:; Cities; result = std::find(begin(AvailableCities), end(AvailableCities), Item);if(!(result != end(AvailableCities))){SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black);printf("\nОшибка: В списке найден город, который отсутствует в перечне маршрутов.\n");Continue( );return;SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);}}AmountCities = Cities.size( )
MapCities = new int *[AmountCities]
i = 0; i < AmountCities; ++i
MapCities[i] = new int[AmountCities]
VisitedCities = new bool[AmountCities]
fseek(* TRAVELS, SEEK_SET, 0)

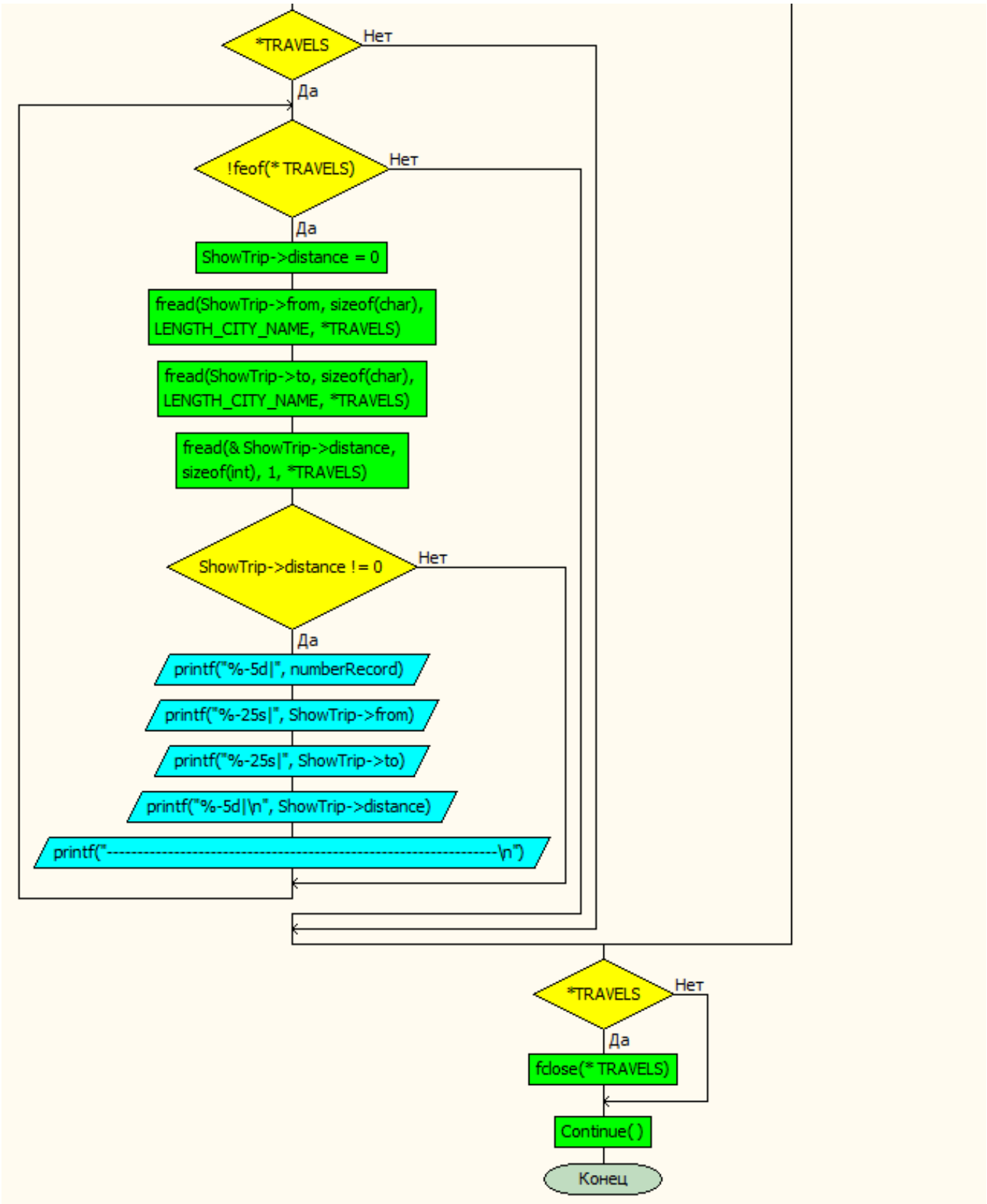

*TRAVELS
Нет
Да
i = 0; i < AmountCities; ++i
j = 0; j < AmountCities; ++j
i == j
Нет
Да
MapCities[i][j] = 0
!feof(* TRAVELS)
Нет
Да
fread(FromFile->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(FromFile->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(&FromFile->distance, sizeof(int), 1, *TRAVELS)
!strcmp(Cities[i].c_str( ), FromFile->from) && !strcmp(Cities[j].c_str( ), FromFile->to)
Нет
Да
MapCities[i][j] = FromFile->distance
printf("\nПуть: ")
i = 0; i < AmountCities; ++i
lastIndex = FindShortestWay(i)
FullWay += MapCities[lastIndex][i]
IndexesCities.push_back(i)
AllLengths.push_back(FullWay)
FullWay = 0
j = 0; j < AmountCities; ++j
VisitedCities[j] = 0



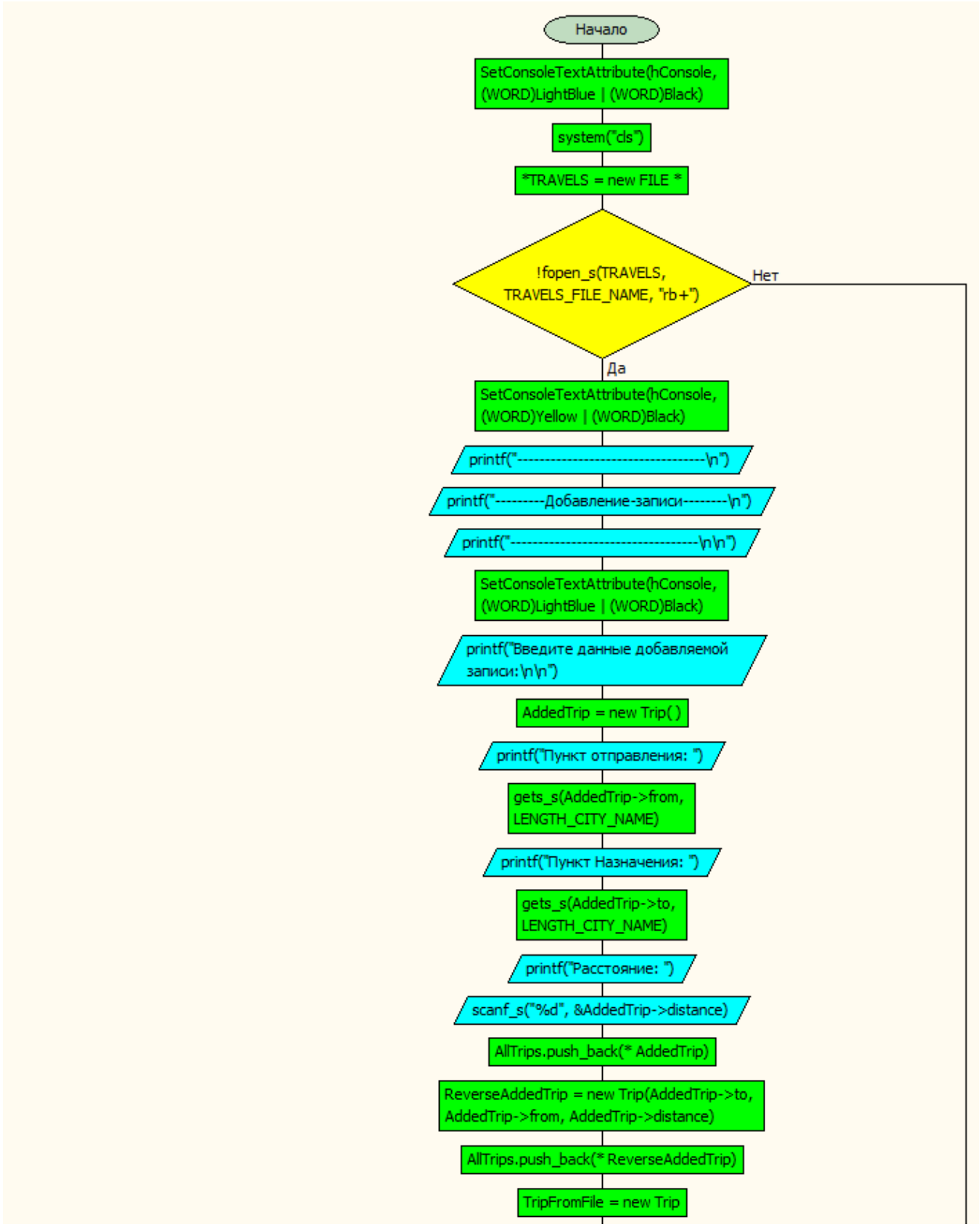
Примечание: линии разрыва между рисунками здесь и далее следует продолжать также вертикально вниз.

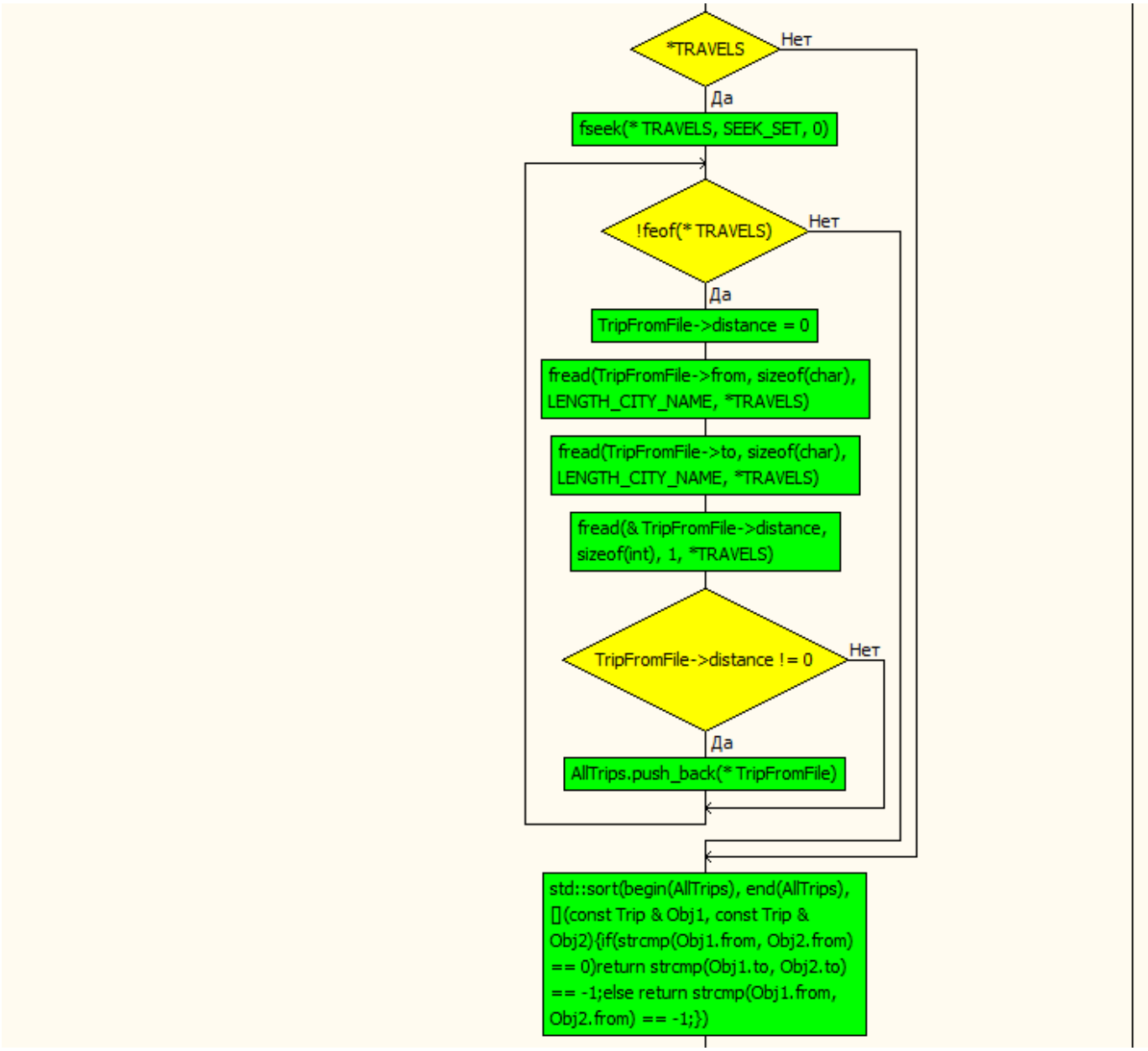
2. showWays

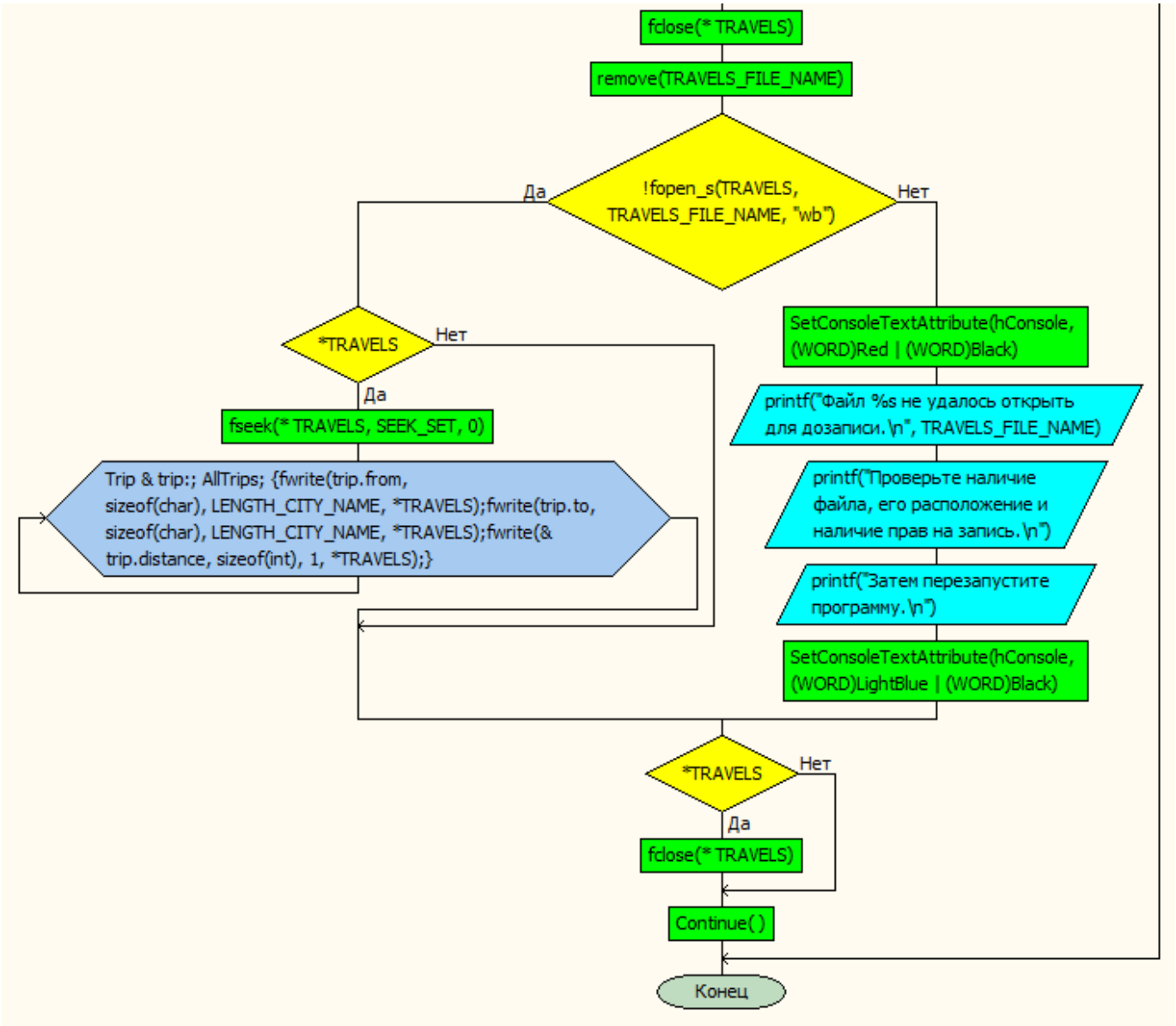

Начало
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)
system("cls")
*TRAVELS = new FILE *
!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "rb")
Да
Нет
SetConsoleTextAttribute(hConsole, (WORD)Yellow | (WORD)Black)
printf("----------\n")
printf("----------Маршруты----------\n")
printf("----------\n")
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)
numberRecord = 1
ShowTrip = new Trip
SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black)
printf("Файл %s не удалось открыть для чтения.\n", TRAVELS_FILE_NAME)
printf("Проверьте наличие файла, его расположение и наличие прав на запись.\n")
printf("Затем перезапустите программу.\n")
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)


*TRAVELS
Нет
Да
!feof(* TRAVELS)
Нет
Да
ShowTrip->distance = 0
fread(ShowTrip->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(ShowTrip->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(& ShowTrip->distance, sizeof(int), 1, *TRAVELS)
ShowTrip->distance != 0
Нет
Да
printf("%-5d|", numberRecord)
printf("%-25s|", ShowTrip->from)
printf("%-25s|", ShowTrip->to)
printf("%-5d|\n", ShowTrip->distance)
printf("----------\n")
*TRAVELS
Нет
Да
fclose(* TRAVELS)
Continue( )
Конец

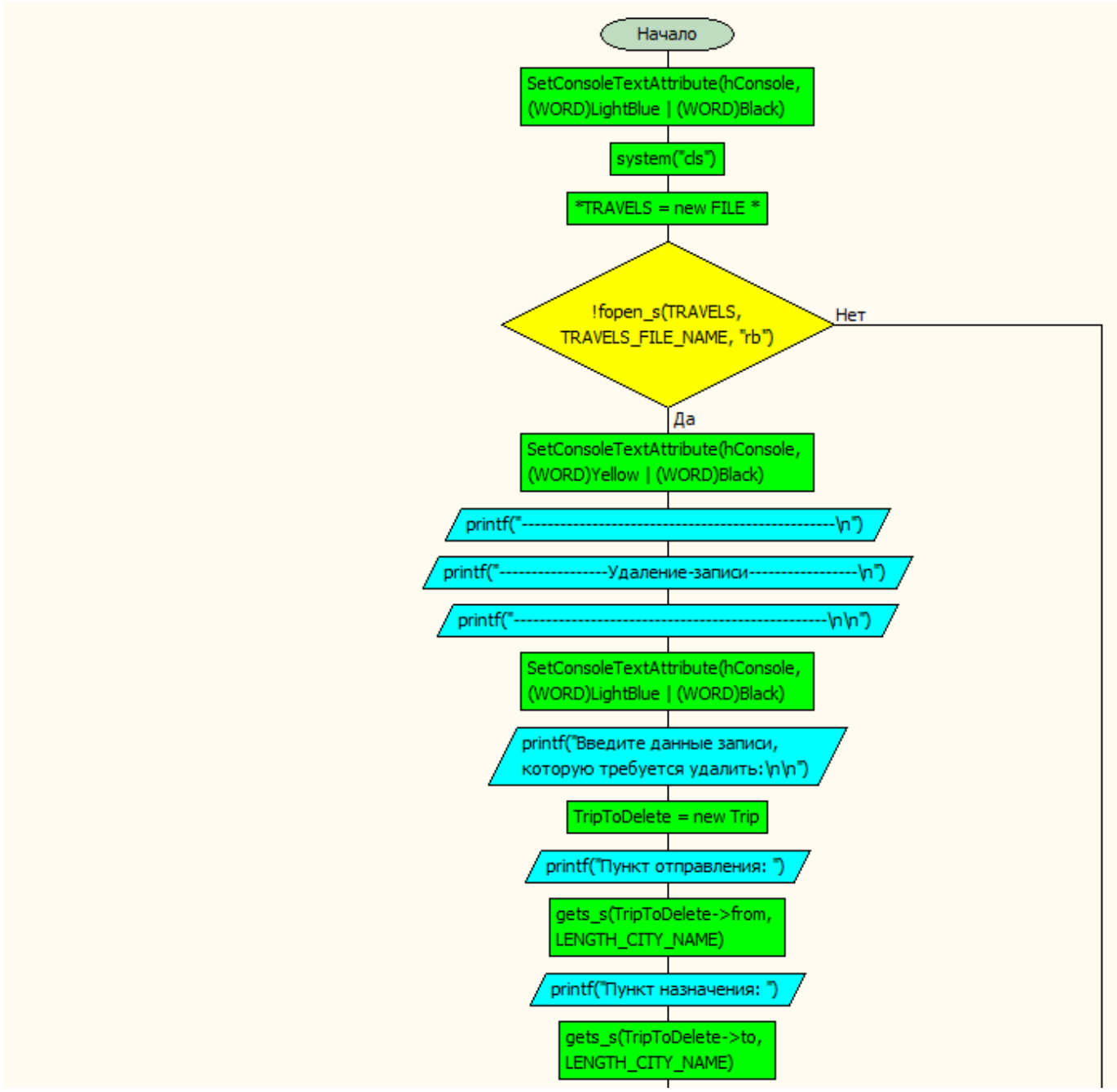
3. addWay

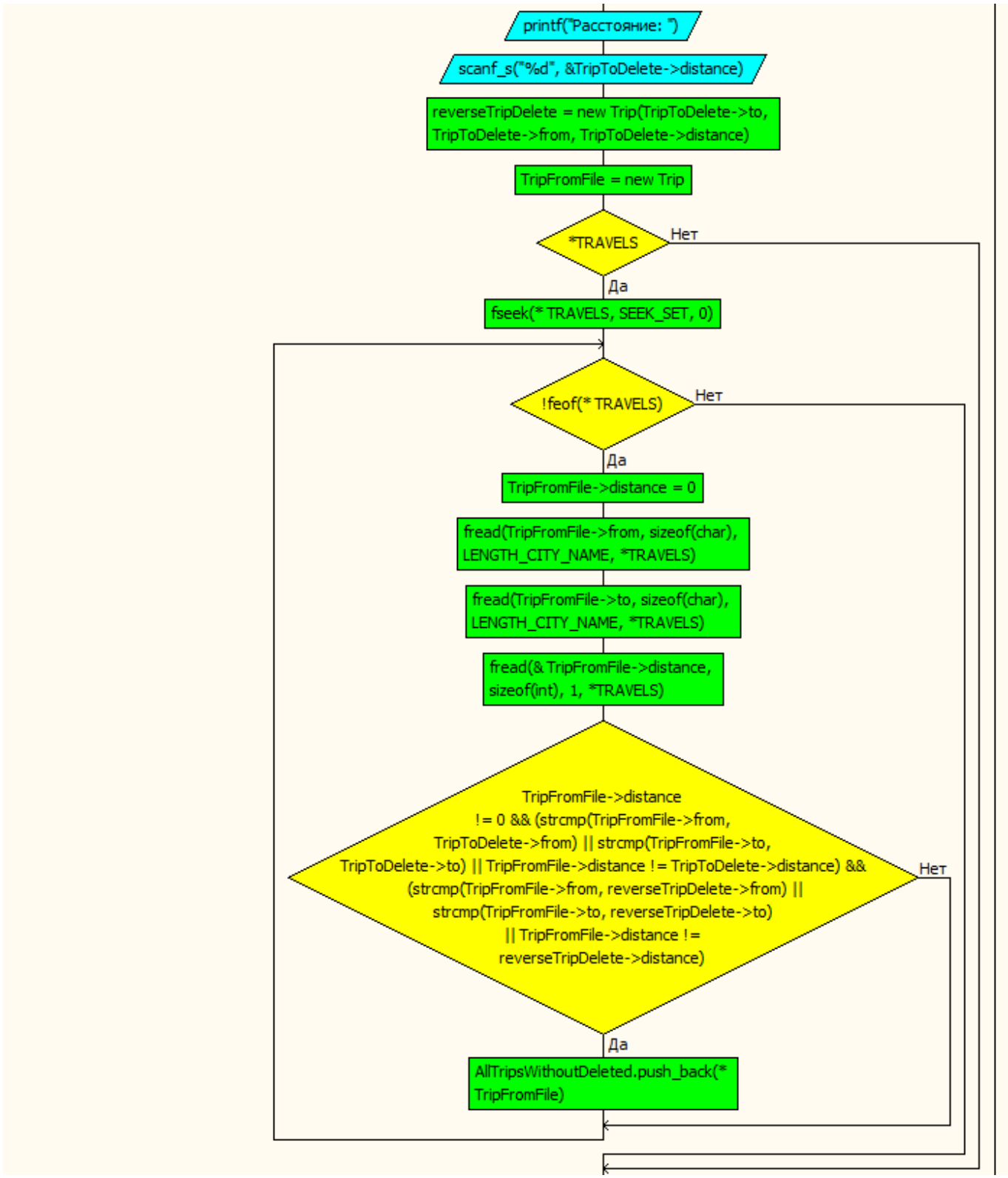

Начало
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)
system("cls")
*TRAVELS = new FILE *
!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "rb+")
Нет
Да
SetConsoleTextAttribute(hConsole, (WORD)Yellow | (WORD)Black)
printf("----------\n")
printf("---------Добавление-записи--------\n")
printf("----------\n\n")
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)
printf("Введите данные добавляемой записи:\n\n")
AddedTrip = new Trip( )
printf("Пункт отправления: ")
gets_s(AddedTrip->from, LENGTH_CITY_NAME)
printf("Пункт Назначения: ")
gets_s(AddedTrip->to, LENGTH_CITY_NAME)
printf("Расстояние: ")
scanf_s("%d", &AddedTrip->distance)
AllTrips.push_back(* AddedTrip)
ReverseAddedTrip = new Trip(AddedTrip->to, AddedTrip->from, AddedTrip->distance)
AllTrips.push_back(* ReverseAddedTrip)
TripFromFile = new Trip

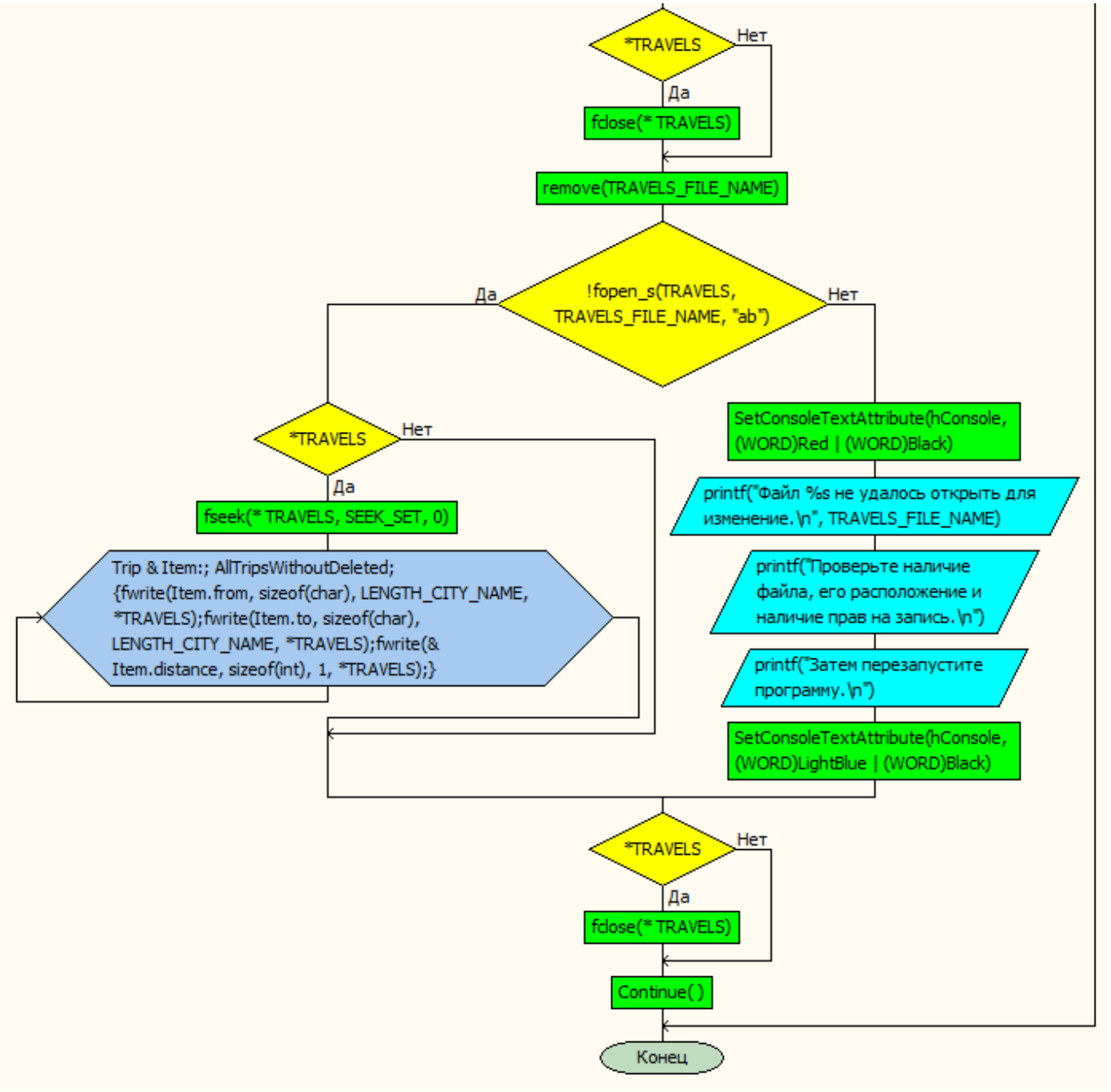

*TRAVELS
Нет
Да
fseek(* TRAVELS, SEEK_SET, 0)
!feof(* TRAVELS)
Нет
Да
TripFromFile->distance = 0
fread(TripFromFile->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(TripFromFile->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(& TripFromFile->distance, sizeof(int), 1, *TRAVELS)
TripFromFile->distance != 0
Нет
Да
AllTrips.push_back(* TripFromFile)
std::sort(begin(AllTrips), end(AllTrips), [](const Trip & Obj1, const Trip & Obj2){if(strcmp(Obj1.from, Obj2.from) == 0)return strcmp(Obj1.to, Obj2.to) == -1;else return strcmp(Obj1.from, Obj2.from) == -1;})


fclose(* TRAVELS)
remove(TRAVELS_FILE_NAME)
!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "wb")
Да
Нет
*TRAVELS
Нет
Да
fseek(* TRAVELS, SEEK_SET, 0)
Trip & trip:; AllTrips; {fwrite(trip.from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);fwrite(trip.to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);fwrite(& trip.distance, sizeof(int), 1, *TRAVELS);}
SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black)
printf("Файл %s не удалось открыть для дозаписи.\n", TRAVELS_FILE_NAME)
printf("Проверьте наличие файла, его расположение и наличие прав на запись.\n")
printf("Затем перезапустите программу.\n")
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)
*TRAVELS
Нет
Да
fclose(* TRAVELS)
Continue( )
Конец

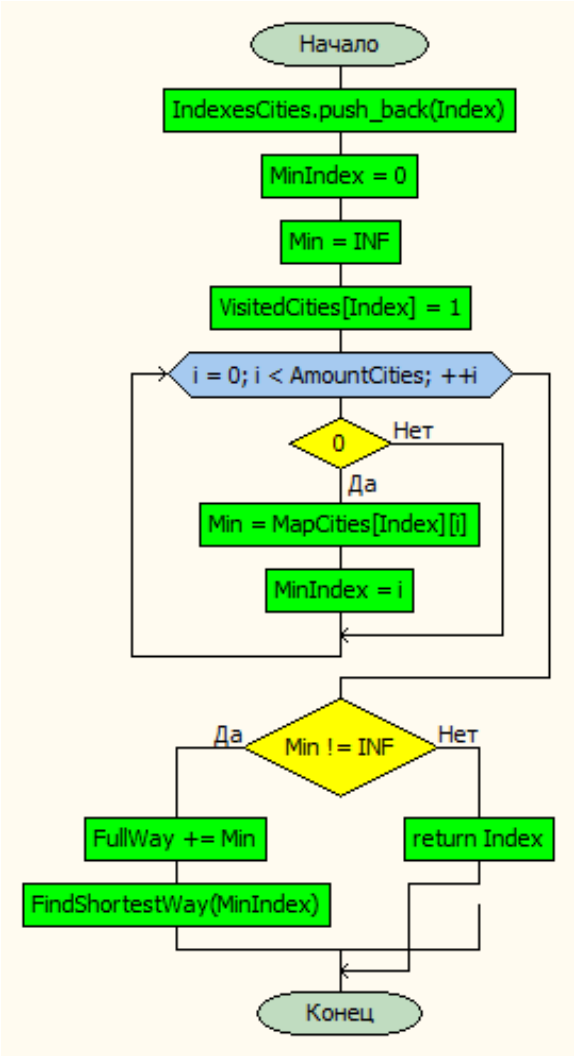
4. delWay


Начало
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)
system("cls")
*TRAVELS = new FILE *
!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "rb")
Нет
Да
SetConsoleTextAttribute(hConsole, (WORD)Yellow | (WORD)Black)
printf("----------\n")
printf("----------Удаление-записи----------\n")
printf("----------\n\n")
SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black)
printf("Введите данные записи, которую требуется удалить:\n\n")
TripToDelete = new Trip
printf("Пункт отправления: ")
gets_s(TripToDelete->from, LENGTH_CITY_NAME)
printf("Пункт назначения: ")
gets_s(TripToDelete->to, LENGTH_CITY_NAME)


printf("Расстояние: ")
scanf_s("%d", &TripToDelete->distance)
reverseTripDelete = new Trip(TripToDelete->to, TripToDelete->from, TripToDelete->distance)
TripFromFile = new Trip
*TRAVELS
Нет
Да
fseek(* TRAVELS, SEEK_SET, 0)
!feof(* TRAVELS)
Нет
Да
TripFromFile->distance = 0
fread(TripFromFile->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(TripFromFile->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS)
fread(& TripFromFile->distance, sizeof(int), 1, *TRAVELS)
TripFromFile->distance != 0 && (strcmp(TripFromFile->from, TripToDelete->from) || strcmp(TripFromFile->to, TripToDelete->to) || TripFromFile->distance != TripToDelete->distance) && (strcmp(TripFromFile->from, reverseTripDelete->from) || strcmp(TripFromFile->to, reverseTripDelete->to) || TripFromFile->distance != reverseTripDelete->distance)
Нет
Да
AllTripsWithoutDeleted.push_back(* TripFromFile)



5. FindShortestWay



Общие характеристики входной и выходной информации

Общие характеристики входной информации представлены в таблице 3.

Табл. 3 Входная информация

Данные	Имя	Тип	Описание
Список городов	CitiesToVisit	char*	Все города через запятую, вводимые пользователем

Общие характеристики выходной информации представлены в таблице 4.

Табл. 4 Выходная информация

Данные	Имя	Тип	Форма	Описание
Город	Cities[Indexes Cities[needIndex * AmountCities +j]]	Std::string	Слово	Город, в который нужно проехать
Длина маршрута	AllLengths[needIndex]	int	Число	Длина всего рассчитанного маршрута через города, которые ввёл пользователь

Набор тестовых примеров с результатами их выполнения

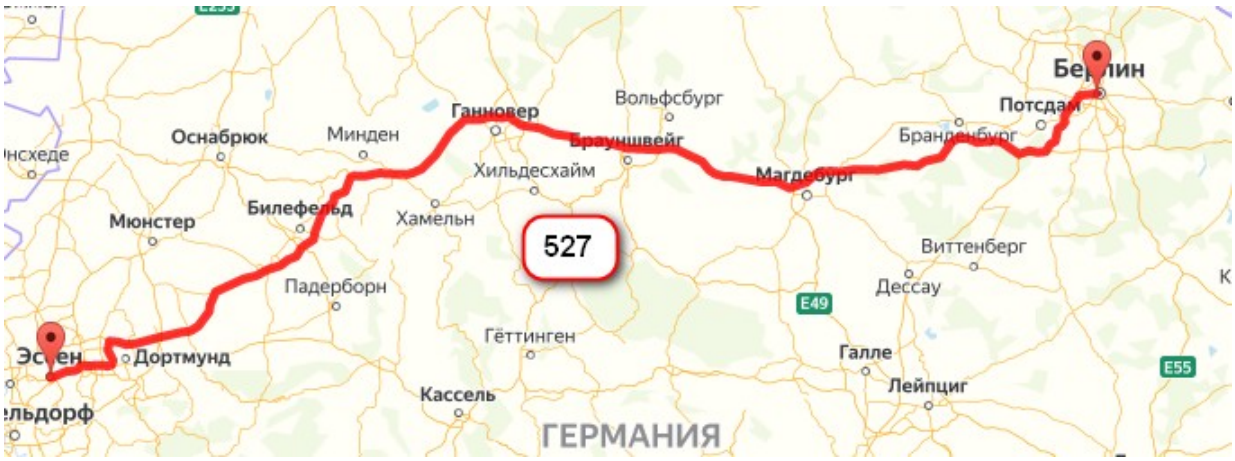
Будем брать города Германии, которые использовались для примеров ранее.

1. Берлин, Эссен.

   Самый простой и очевидный пример. Понятно, что такой путь один.

   Путь: Берлин -> Эссен -> Берлин

   Длина рассчитанного пути: 1054



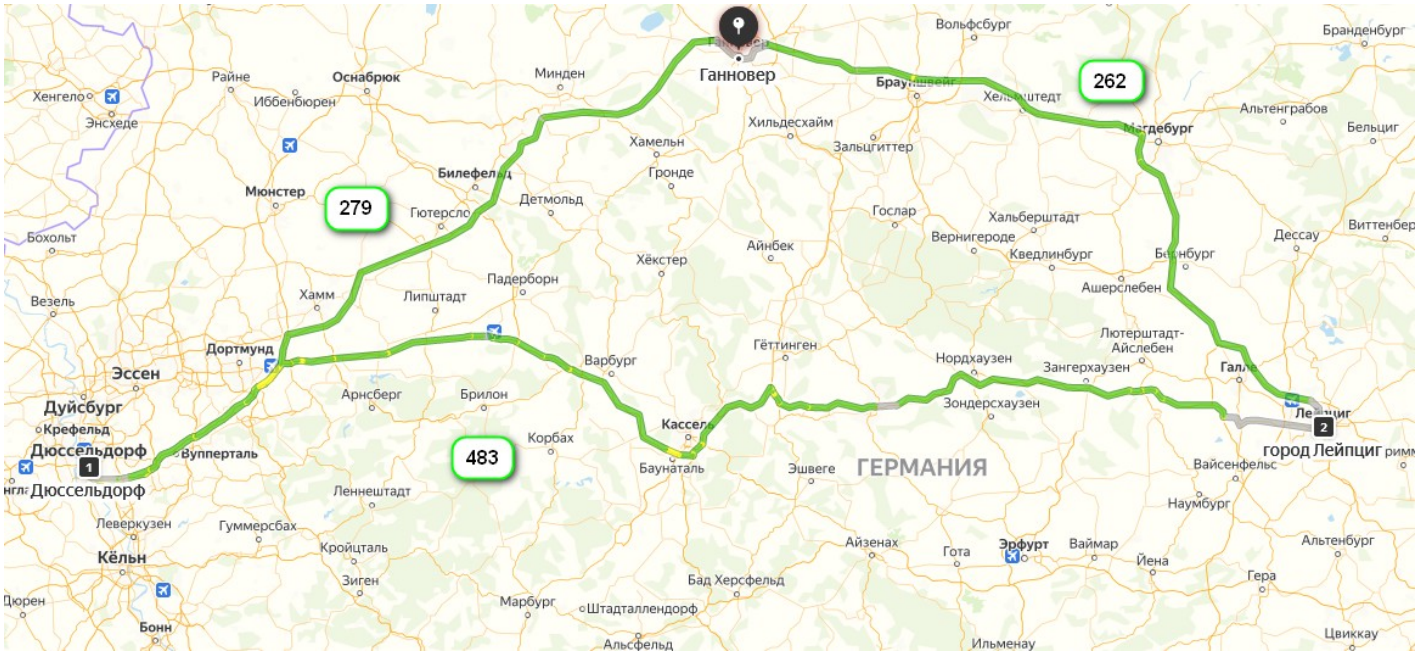
Расстояние от Берлина до Эссена 527, плюс обратный путь тоже 527, итого 1054.

2. Лейпциг, Дюссельдорф, Ганновер.

   Путь: Ганновер -> Лейпциг -> Дюссельдорф -> Ганновер

Также достаточно очевиден, другого пути быть просто не может, кроме как подобного треугольника.

Длина рассчитанного пути: 1024

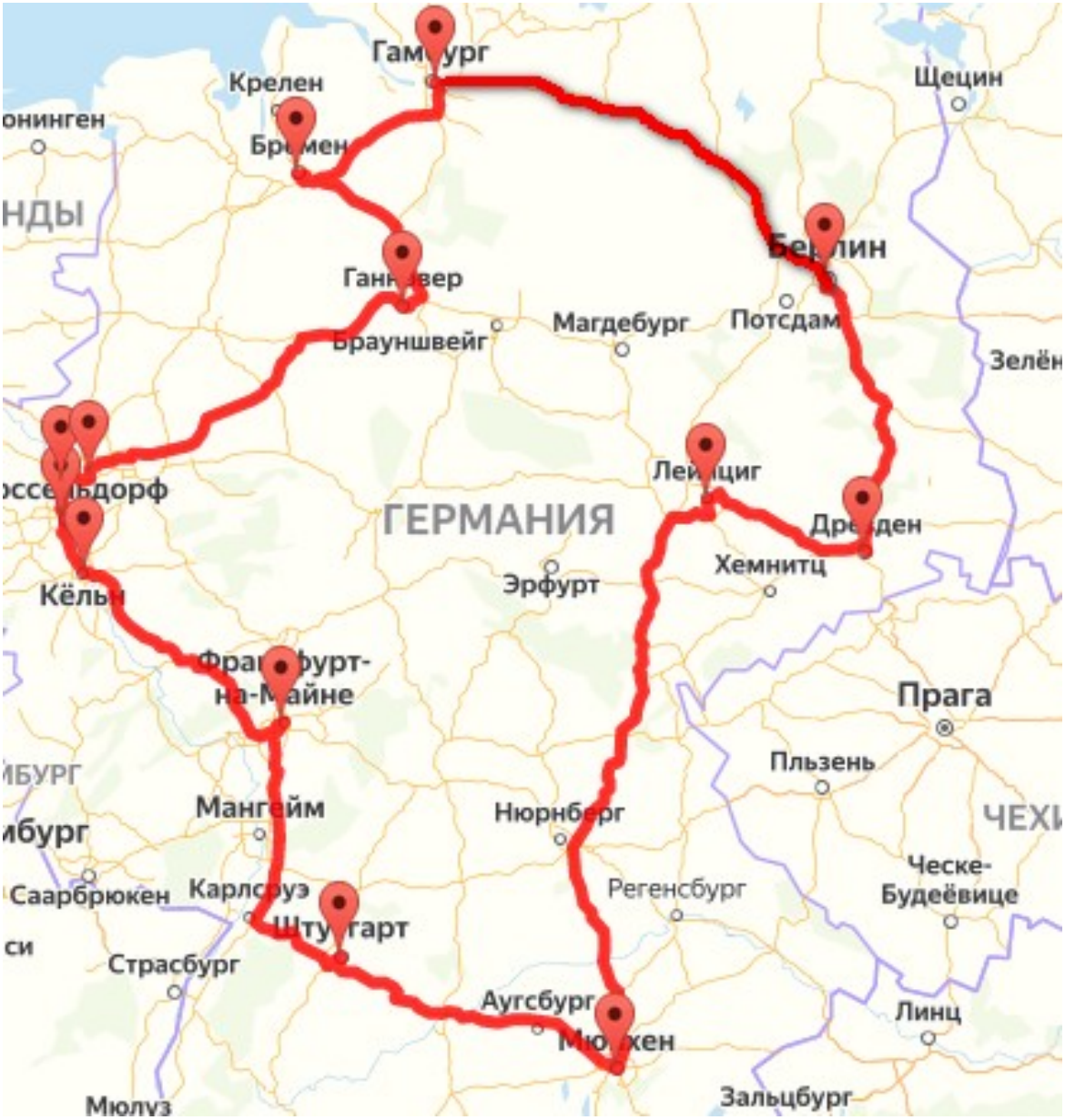


3. Все города, внесённые в тестовый пример.

   Путь: Берлин -> Лейпциг -> Дрезден -> Нюрнберг -> Мюнхен -> Штутгарт -> Франкфурт-на-Майне -> Кёльн -> Дюссельдорф -> Дуйсбург -> Эссен -> Ганновер -> Бремен -> Гамбург -> Берлин

Вообще программа неплохо работает со всеми маршрутами, которые являются очень похожими на кольцевые. Жадный алгоритм сработает неверно, если какой-либо город будет сильно в стороне от других.

Длина рассчитанного пути: 2272



Выше показан самый оптимальный маршрут, однако наш маршрут от Берлина отклонился до Лейпцига, так расстояние до него ближе, чем до Дрездена. Данная неточность безусловно является следствием работы жадного алгоритма.

В данном случае ошибка на участке Берлин, Лейпциг, Дрезден, Нюрнберг. Есть два возможных варианта:

1. Берлин -> Лейцпиг -> Дрезден -> Нюрнберг (188+112+312 = 612)
2. Берлин -> Дрезден -> Лейпциг -> Нюрнберг (192+112+283 = 587)

Совершенно ясно, что второй путь лучше, однако алгоритм не может этого оценить, двигаясь в Лейпциг, как в ближайший пункт.

В результате этого мы потеряли 25 км.

Руководство пользователя

Инсталляция

Программа не требует установки.

Для работы программы необходимы следующие файлы:

1. TRAVELLING_SALESMAN.exe – основной исполняемый модуль программы.
2. travels.bin – файл, содержащий в себе расстояния между городами (файл в начале может быть пустым, однако всегда должен присутствовать в одной директории с исполняемой программой).

Запуск

Запустите файл TRAVELLING_SALESMAN.exe.

Инструкция по работе с программой

После запуска программы на экране вы увидите основное меню программы (рис.1).

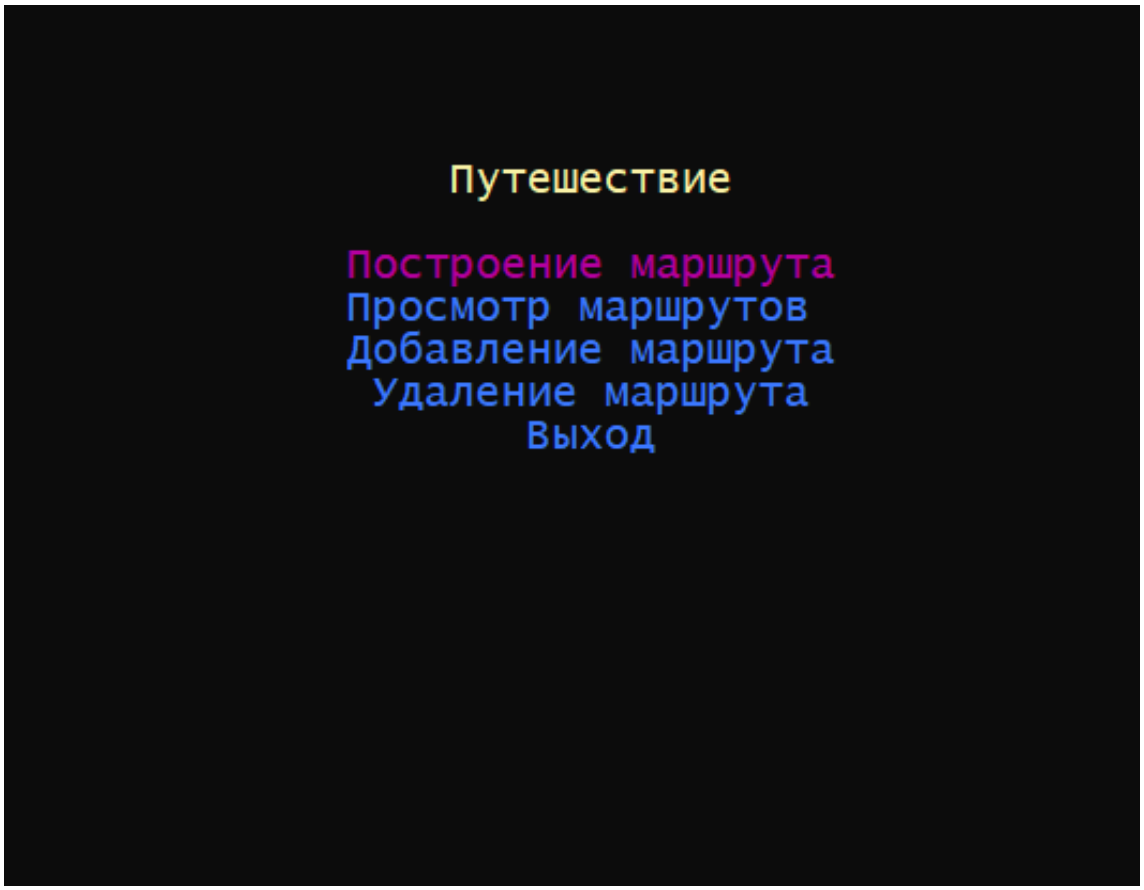


Рис1. Меню программы.

Из названия пунктов понятно их предназначение.

1. Построение маршрута.

Пользователю требуется ввести список городов, которые точно содержатся в файле travels.bin, без пробелов через запятую (программа определяет именно запятую как разделитель). Затем программой будет произведён расчёт и выдан ответ.

Для демонстрации этого и последующих пунктов в файл travels.bin были заранее записаны маршруты между 14 крупными городами Германии: Гамбург, Дрезден, Кёльн, Лейпциг, Мюнхен, Эссен, Берлин, Нюрнберг, Ганновер, Штутгарт, Дуйсбург, Франкфурт-на-Майне, Дюссельдорф, Бремен.

Примечание: расстояния между городами были взяты с https://www.avtodispetcher.ru/distance/, взяты значения по трассе, поскольку в условии задачи говорится о карте автомобильных дорог некоторого региона.

Пример вывода маршрута для всех 14 городов показан на рисунке 2.

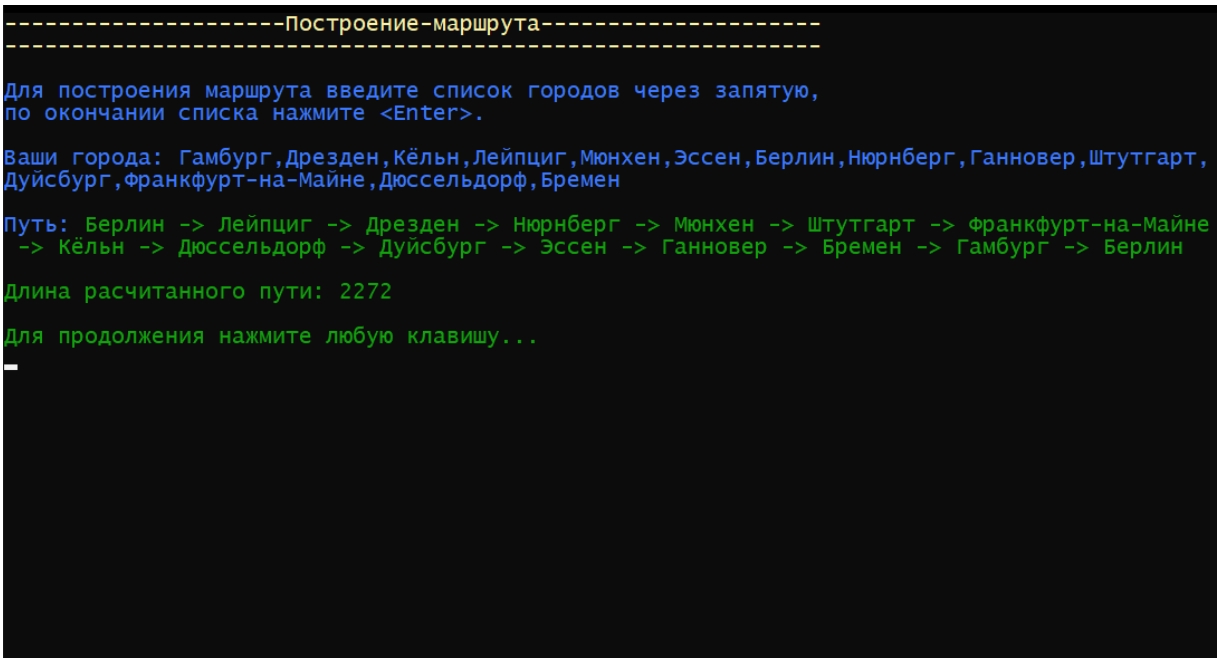


Рис.2 Пример вывода программы для объезда всех 14 городов.

2. Просмотр маршрутов.

Пример вывода показан на рисунке 3. Выводятся расстояния от одного города до всех остальных городов.

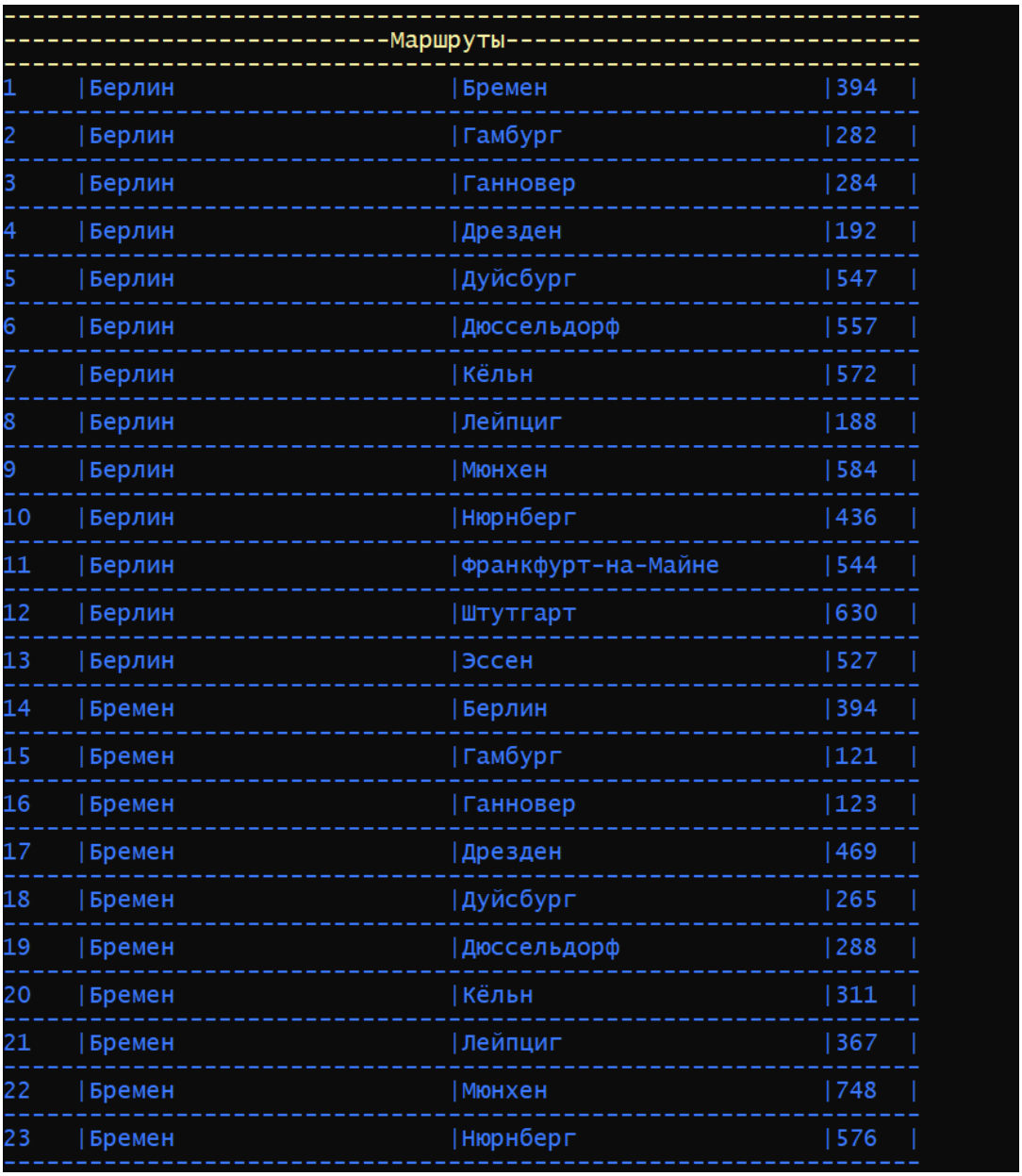


Рис.3 Пример вывода расстояний между городами.

3. Добавление маршрута.

Пользователь вводит пункт отправления, пункт назначения и расстояние между этими пунктами. В файл добавляется запись, которую ввёл пользователь, а также запись, обратная этой (пункты меняются местами, расстояние прежнее).

4. Удаление маршрута.

Порядок действий пользователя такой же, как и пункте 3, только записи наоборот удаляются.

5. Выход.

Окно приложения закрывается.

Описание исходных модулей программы

1. Main.cpp

Содержит точку входа в программу, отрисовку меню, функции SetLanguage и SetTypeLucidaConsole.

2. Headers.h

Содержит все подключаемые заголовочные файлы.

3. Trip.h и Trip.cpp

Содержат соответственно интерфейс и реализацию структуры Trip, которая используется для описания поездки.

4. Colors.h

Содержит Перечисляемые константы для 16-ти консольных цветов.

5. Constants.h

Содержит основные константы, используемые в программе.

6. MenuItems.h и MenuItems.cpp

Содержат соответственно прототипы и реализации функций-пунктов меню. Кроме того, файл MenuItems.cpp содержит функции FindShorterstWay и Continue.

Код

1. Содержимое файла Main.cpp

```
// Файл Main.cpp содержит точку входа в программу - здесь начинается и заканчивается выполнение программы.

/*
* Программа компилировалась в операционной система Windows 10 x64, IDE - Microsoft Visual Studio 2019.
*/

/*
* 6. Тема: Разработка программы определения оптимального маршрута.
*
* Задача выполнена при помощи алгоритма ближайшего соседа.
* Данный алгоритм является жадным (означает не всегда точное определение кратчайшего маршрута).
*
* Алгоритм был выбран из-за относительной простоты реализации и формулировки задачи (требуется обходить каждый город в списке),
* в то время как большинство других алгоритмов (например, алгоритм Дейкстры или алгоритм Беллмана-Форда) находят либо кратчайший путь от точки до точки,
* либо кратчайший путь от точки до всех других точек (хотя их тоже можно приспособить).
*
* По факту суть задачи состоит в нахождении гамильтонова цикла - замкнутового маршрута, проходящего через каждую вершину только один раз.
*
```

```
* Алгоритм является достаточно быстрым, неплохо работает на симметричной матрице городов.
* Данная реализация содержит небольшое улучшение, которое позволяет алгоритму определять маршрут ближе к идеальному,
* однако это заметно лишь на достаточно большом количестве городов.
* Объяснение работы алгоритма и улучшения содержится в файле MenuItems.cpp.
*
* Пример, расположенный в файле travels.bin, взят с https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2%D0%BE%D1%8F%D0%B6%D1%91%D1%80%D0%B0
* Главный пример: Гамбург,Дрезден,Кёльн,Лейпциг,Мюнхен,Эссен,Берлин,Нюрнберг,Ганновер,Штутгарт,Дуйсбург,Франкфурт-на-Майне,Дюссельдорф,Бремен
* В примере выше городов 14 (на картинке в Википедии не удалось найти 15 городов), маршрут, построенный программой, похож на маршрут на картинке.
* Расстояния были взяты с https://www.avtodispetcher.ru/distance/, взяты расстояния по трассе (в задаче карта автомобильных дорог).
*/

#include "Headers.h" // Содержит все подключаемые заголовочные файлы.

extern HANDLE hConsole; // Объявляем как extern для использования в других местах.

void SetLanguage() // Установка кодировки Windows-1251 в поток ввода и вывода для отображения русского текста.
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
}

void SetTypeLucidaConsole() // Ставим шрифт Lucida Console для работы с кодировкой Windows-1251.
{
	HANDLE hCon = CreateConsoleScreenBuffer(GENERIC_READ | GENERIC_WRITE, 0, NULL, CONSOLE_TEXTMODE_BUFFER, NULL);
	CONSOLE_FONT_INFOEX cfi;
	if (hCon != INVALID_HANDLE_VALUE) {
		GetCurrentConsoleFontEx(hConsole, TRUE, &cfi);
		cfi.cbSize = sizeof(cfi);
		cfi.nFont = 0;
		cfi.dwFontSize.X = 0;
		cfi.dwFontSize.Y = 20;
		cfi.FontFamily = FF_DONTCARE;
		cfi.FontWeight = FW_NORMAL;
		wcscpy_s(cfi.FaceName, L"Lucida Console");
		SetCurrentConsoleFontEx(hConsole, FALSE, &cfi);
	}
}

int main()
{
	SetLanguage();				// Установка кодировки Windows-1251 для отображения русского текста.
	SetTypeLucidaConsole();			// Ставим шрифт Lucida Console для работы с кодировкой Windows-1251.
```

```
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); // Получение информации о консольном окне.
	HWND hcon = GetConsoleWindow();			// Получение информации о консольном окне.
	MoveWindow(hcon, 100, 10, 1280, 1024, FALSE);	// Окно распологаем по координатам (100;10) от левого верхнего угла, размеры 1280X1024.

	COORD position[NUMBER_MENU_ITEMS]; // Координаты пунктов меню.
	int selectedItem = 0;		// Выбранный пункт меню.
	do
	{
		int xmax, ymax;

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

		// Определяем положение пунктов меню.
		int y0 = 7;
		for (int i = 0; i < NUMBER_MENU_ITEMS; ++i)
		{
			position[i].X = (xmax - strlen(ITEMS[i])) / 2;
			position[i].Y = y0 + i;
		}
		system("cls");
		SetConsoleTextAttribute(hConsole, (WORD)Yellow| (WORD) Black);

		COORD pos;
		pos.X = (xmax - strlen(TITLE)) / 2;
		pos.Y = 5;
		SetConsoleCursorPosition(hConsole, pos);
		puts(TITLE);
		SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
		for (int i = 0; i < NUMBER_MENU_ITEMS; ++i)
		{
			SetConsoleCursorPosition(hConsole, position[i]);
			puts(ITEMS[i]);
		}
		SetConsoleTextAttribute(hConsole, (WORD)Magenta | FOREGROUND_INTENSITY | (WORD)Black);
		SetConsoleCursorPosition(hConsole, position[selectedItem]);			// выделение текущего пункта ярким цветом
		puts(ITEMS[selectedItem]);
		unsigned char pressedKey;
		do
		{
			pressedKey = _getch();
			if (pressedKey == 224)
			{
```

```
                pressedKey = _getch();
                switch (pressedKey)
                {
                case 72:
                        SetConsoleCursorPosition(hConsole, position[selectedItem]);
                        SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
                        puts(ITEMS[selectedItem]);
                        --selectedItem;
                        if (selectedItem < 0) selectedItem = 4;
                        SetConsoleTextAttribute(hConsole, (WORD)Magenta |
FOREGROUND_INTENSITY | (WORD)Black);
                        SetConsoleCursorPosition(hConsole, position[selectedItem]);
                        puts(ITEMS[selectedItem]); break;
                case 80:
                        SetConsoleCursorPosition(hConsole, position[selectedItem]);
                        SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
                        puts(ITEMS[selectedItem]);
                        ++selectedItem;
                        if (selectedItem > 4) selectedItem = 0;
                        SetConsoleTextAttribute(hConsole, (WORD)Magenta |
FOREGROUND_INTENSITY | (WORD)Black);
                        SetConsoleCursorPosition(hConsole, position[selectedItem]);
                        puts(ITEMS[selectedItem]); break;
                }
            }
        } while (pressedKey != 13);
        switch (selectedItem)
        {
        case 0:
            getDirections(); // Построение маршрута.
            break;
        case 1:
            showWays();      // Показать маршруты.
            break;
        case 2:
            addWay();        // Добавить маршрут.
            break;
        case 3:
            delWay();        // Удалить маршрут.
        }
    } while (selectedItem != 4);

    return 0;
}
```

2. Содержимое файла Headers.h

```
// Содержит все подключаемые заголовочные файлы.

#pragma once
```

```
#include <cstdio>
#include <conio.h>
#include <vector>
#include <windows.h>
#include <algorithm>
#include <sstream>
#include <stdint.h>

#include "Colors.h"
#include "Constants.h"
#include "MenuItems.h"
#include "Trip.h"
```

3. Содержимое файла Constants.h

```
// Содержит основные константы.

#pragma once

static const char* TRAVELS_FILE_NAME = "travels.bin";          // Имя файла с маршрутами.
static const char* TITLE             = "Путешествие";          // Заголовок.
    const int   NUMBER_MENU_ITEMS = 5;                                  // Количество пунктов меню.
    const int   LENGTH_CITY_NAME  = 20;                  // Максимальная длина имени города.
    const char  ITEMS[5][20]      = { "Построение маршрута", // Наименования пунктов меню.
"Просмотр маршрутов", "Добавление маршрута", "Удаление маршрута", "Выход" };
```

4. Содержимое файла Colors.h

```
// Содержит цветовую гамму из 16 базовых консольных цветов.

#pragma once

enum ConsoleColor
{
        Black = 0,
        Blue = 1,
        Green = 2,
        Cyan = 3,
        Red = 4,
        Magenta = 5,
        Brown = 6,
        LightGray = 7,
        DarkGray = 8,
        LightBlue = 9,
        LightGreen = 10,
        LightCyan = 11,
        LightRed = 12,
        LightMagenta = 13,
        Yellow = 14,
        White = 15
};
```

5. Содержимое файла Trip.h

```
// Содержит структуру поездки.

#pragma once

#include <string>      // Для strcpy_s в Trip.cpp.

#include "Constants.h" // Константы, здесь требуется LENGHT_CITY_NAME.

struct Trip // Информация о поездке.
{
	Trip();                          // Конструтор по умолчанию.
	Trip(char* f, char* t, int d); // Конструктор с параметрами.
	Trip(const Trip& Obj);       // Конструктор копирования.

	char from[LENGTH_CITY_NAME]; // Пункт отправления.
	char to[LENGTH_CITY_NAME];     // Пункт назначения.
	int distance;                          // Расстояние между пунктами.
};
```

6. Содержимое файла Trip.cpp

```
// Содержит описание методов структуры Trip.

#include "Trip.h" // Структура поездки.

Trip::Trip() // Конструтор по умолчанию.
{
	strcpy_s(from, LENGTH_CITY_NAME, "");
	strcpy_s(to, LENGTH_CITY_NAME, "");
	distance = 0;
}

Trip::Trip(char* f, char* t, int d) // Конструктор с параметрами.
{
	strcpy_s(from, LENGTH_CITY_NAME, f);
	strcpy_s(to, LENGTH_CITY_NAME, t);
	distance = d;
}

Trip::Trip(const Trip& Obj) // Конструктор копирования.
{
	strcpy_s(from, LENGTH_CITY_NAME, Obj.from);
	strcpy_s(to, LENGTH_CITY_NAME, Obj.to);
	distance = Obj.distance;
}
```

7. Содержимое файла MenuItems.h

```
// Содержит функции - пункты меню.

#pragma once

void getDirections(); // Построить маршрут.
void showWays();      // Показать маршруты.
void addWay();        // Добавить маршрут.
void delWay();        // Удалить маршрут.
```

8. Содержимое файла MemuItems.cpp

```
#include "Headers.h"

HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE); // Для установки цветов

constexpr auto   INF = INT16_MAX;  // Числовой аналог бесконечности.
int**                MapCities;       // Матрица с расстояниями городов, введённых пользователем.
int            FullWay = 0;          // Длина каждого расчитанного пути.
bool*            VisitedCities;    // Матрица с метками посещённых городов.
int            AmountCities = 0; // Количество городов.
std::vector<int> IndexesCities;    // Индексы городов в порядке их посещения.

/*
* Принцип работы алгоритма:
* В каждой строке сформированной матрицы, начиная с нулевой, находим кратчайшее расстояние.
* Если город с таким индексом ещё не был посещён, едем в него, отметив, что мы его посетили.
* Вместе с этим прибавили кратчайшее расстояние в строке к общей длине рассчитываемого пути
* Пока не закончатся все города для посещения, мы ездим по городам и прибавляем расстояния.
* В конце прибавляем расстояние от оставшегося города до города, с которого вы начали путешествие,
так нужно в него вернуться (по логике задачи коммивояжера).
* Получаем несколько возможных вариантов, из которого выбираем наименьшую длину маршрута.
*
* Небольшой пример работы улучшения:
* Программе был дан следующий список:
Берлин,Лейпциг,Ганновер,Гамбург,Эссен,Дюссельдорф,Дрезден
* Если действовать строго по жадному алгоритму, длина маршрута составит 1773 км (Берлин ->
Лейпциг -> Дрезден -> Ганновер -> Гамбург -> Эссен -> Дюссельдорф -> Берлин).
* Однако программа нашла путь длиной 1508 км (Берлин -> Гамбург -> Ганновер -> Эссен ->
Дюссельдорф -> Лейпциг -> Дрезден -> Берлин).
*/

int FindShortestWay(int Index) // Основной рекурсивный алгоритм нахождения пути.
{
        // На вход принимает индекс города, он же является индексом строки,
        // в которой ведётся поиск наименьшего кратчайшего расстояния.
        IndexesCities.push_back(Index);
        int MinIndex = 0;                       // Следующий индекс города, в который направлен
минимальный путь.
        int Min = INF;                  // Для нахождения минимума сначала приравняем к большому числу.
```

```
	VisitedCities[Index] = 1;	// Отмечаем, что в городе с данным индексом мы побывали.
	for (int i = 0; i < AmountCities; ++i) // Начинаем цикл по строке с полученным индексом.
		// Если ячейка матрицы в данной строке больше 0, не бесконечна, мы ещё не бывали в этом городе,
		// и расстояние до него меньше ранее рассчитанного минимального.
		if (MapCities[Index][i] != INF && MapCities[Index][i] < Min && VisitedCities[i] != 1 && MapCities[Index][i] > 0)
		{
			// То он может рассматриваться, как город, до которого путь наименьший.
			Min = MapCities[Index][i]; // Запомнили минимальное расстояние.
			MinIndex = i;		// Запомнили индекс города, до которого в строке наименьшее расстояние.
		}
	// К этому моменту получили минимальное расстояние и индекс нужного города.
	// Пока остаётся хоть один город, который мы не посетили, переменная Min не будет равна бесконечности.
	if (Min != INF)
	{
		FullWay += Min;				// Прибавляем полученное минимальное расстояние.
		FindShortestWay(MinIndex); // И снова вызываем функцию, передав ей индекс города, расстояние до которого здесь было минимальным.
	}
	else
	{
		// Блок выполниться, если все города мы уже прошли.
		return Index; // Возвращаем индекс последнего посещённого города.
		// Здесь рекурсия заканчивается.
	}
}

void Continue() // Функция выступает заглушкой в конце выполнения каждого из пунктов меню.
{
	SetConsoleTextAttribute(hConsole, (WORD)Green | (WORD)Black);
	printf("\nДля продолжения нажмите любую клавишу...\n");
	SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
	char* Continue = new char;
	*Continue = getchar();
	delete Continue;
}

void getDirections() // Построить маршрут
{
	SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

	system("cls");
	FILE** TRAVELS = new FILE*;
	if (!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "rb"))
	{
		SetConsoleTextAttribute(hConsole, (WORD)Yellow | (WORD)Black);
		printf("------------------------------------------------------------\n");
		printf("----------------------Построение-маршрута---------------------\n");
		printf("------------------------------------------------------------\n\n");
		SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
```

```
		printf("Для построения маршрута введите список городов через запятую, \nпо окончании списка нажмите <Enter>.\n\n");
		printf("Ваши города: ");
		char* CitiesToVisit = new char[LENGTH_CITY_NAME * 50];
		std::vector<std::string> Cities; // Все города, которые введёт пользователь.
		// Список всех городов получаем в строку через запятую, пробелы или другие знаки недопустимы.
		gets_s(CitiesToVisit, LENGTH_CITY_NAME * 50);

		// Разбиваем строку на отдельные города.
		std::stringstream ss(CitiesToVisit);
		std::string City;
		while (ss.good())
		{
			getline(ss, City, ',');
			Cities.push_back(City);
		}
		delete[] CitiesToVisit;

		// Формируем список доступных городов.
		// Читаем из файла часть записи "Откуда" и, если её нет в векторе, заталкиваем туда.
		std::vector<std::string> AvailableCities;
		Trip* FromFile = new Trip;
		if (*TRAVELS)
		{
			while (!feof(*TRAVELS))
			{
				fread(FromFile->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
				fread(FromFile->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
				fread(&FromFile->distance, sizeof(int), 1, *TRAVELS);

				// Находим, есть ли такая же запись в векторе.
				// За контейнер можно было взять и множество, но по нему нельзя произвольно индексироваться.
				auto result = std::find(begin(AvailableCities), end(AvailableCities), FromFile->from);
				if (strcmp(FromFile->from, " ") && result == end(AvailableCities))
					AvailableCities.push_back(FromFile->from);
			}
		}

		// Сортируем введённый пользователем список городов по алфавиту быстрой сортировкой из STL.
		std::sort(begin(Cities), end(Cities), [](std::string One, std::string Two)
			{
				return One < Two;
			});

		// Проверяем, всё ли города из введённого списка доступны (есть ли записи о них).
		for (const auto& Item : Cities)
		{
			auto result = std::find(begin(AvailableCities), end(AvailableCities), Item);
```

```
                if (!(result != end(AvailableCities)))
                {
                        SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black);
                        printf("\nОшибка: В списке найден город, который отсутствует в перечне
маршрутов.\n");
                        Continue();
                        return;
                        SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
                }
        }

        AmountCities = Cities.size();
        MapCities = new int* [AmountCities];      // Будущая матрица расстояний.
        for (int i = 0; i < AmountCities; ++i)
                MapCities[i] = new int[AmountCities];
        VisitedCities = new bool[AmountCities];   // Отметки о посещённых городах.

        // Формируем матрицу расстояний, сверяя имена городов в списке пользователя и в
считанных записях из файла.
        fseek(*TRAVELS, SEEK_SET, 0);
        if (*TRAVELS)
        {
                for(int i = 0; i<AmountCities;++i)
                        for (int j = 0; j < AmountCities; ++j)
                        {
                                if (i == j)
                                {
                                        MapCities[i][j] = 0; // По диагонали все элементы нулевые
(недостижимые).
                                        continue;
                                }
                                while (!feof(*TRAVELS))
                                {
                                        fread(FromFile->from, sizeof(char), LENGTH_CITY_NAME,
*TRAVELS);
                                        fread(FromFile->to, sizeof(char), LENGTH_CITY_NAME,
*TRAVELS);
                                        fread(&FromFile->distance, sizeof(int), 1, *TRAVELS);

                                        if (!strcmp(Cities[i].c_str(), FromFile->from) &&
                                                !strcmp(Cities[j].c_str(), FromFile->to))
                                        {
                                                MapCities[i][j] = FromFile->distance; // Вносим
расстояние в матрицу.
                                                break;
                                        }
                                }
                        }
        }
        delete FromFile;

        printf("\nПуть: ");      // Рядом будут выведены имена городов, образующих путь.
```

```
		/*
		* Алгоритм ближайшего соседа жадный, но с небольшим улучшением.
		* Ниже перебираются все "жадные" пути из всех городов (начальная точка каждый раз меняется).
		* Затем выбирается наилучший путь.
		*/

		std::vector<int> AllLengths;		// Длины всех маршрутов, которые нашёл алгоритм, каждый раз меняя точку отправления.
		for (int i = 0; i < AmountCities; ++i)	// Стартует цикл, перебирающий все возможные точки отправления.
		{
			int lastIndex = FindShortestWay(i);	// Передаём каждую точку в алгоритм, и принимаем индекс последнего посещенного города.
			FullWay += MapCities[lastIndex][i];	// Прибавляем путь на возрат (маршрут кольцевой).
			IndexesCities.push_back(i);		// Вносим в вектор индекс города, к которому надо вернуться. (В самом векторе все маршруты идут подряд).
			AllLengths.push_back(FullWay);		// Запись о длине рассчитанного пути вносим в вектор.
			FullWay = 0;				// Обнуляем путь.
			for (int j = 0; j < AmountCities; ++j) // Обнуляем метки посещенных городов, поскольку пойдём их ещё раз.
				VisitedCities[j] = 0;
		}

		int minLength = INF; // Найдём наименьшую длину.
		int needIndex = 0;	// Сюда впишем индекс маршрута, с которого начинаются нужные нам города в векторе IndexesCities (маршрут здесь - несколько городов).
		for (int i = 0; i < AmountCities; ++i)
		{
			if (AllLengths[i] < minLength)
			{
				minLength = AllLengths[i];
				needIndex = i;		// Здесь находим индекс записи, с которого начинаются нужные нам города.
			}
		}

		SetConsoleTextAttribute(hConsole, (WORD)Green | (WORD)Black);

		// Выводим маршрут и его длину
		for (int i = 0; i < AmountCities; ++i)
		{
			if (i == needIndex) // Если мы нашли позицию, с которой начинаются нужные нам индексы городов.
			{
				for (int j = needIndex; j < AmountCities + needIndex; ++j)			// То читаем их индексы и сопоставляем их с именами, затем выводим на экран.
					printf("%s -> ", Cities[IndexesCities[needIndex * AmountCities + j]].c_str());
				printf("%s", Cities[IndexesCities[needIndex * (AmountCities + 1)]].c_str());	// Выводим город, с которого следует отправление, чтобы закольцевать маршрут.
```

```
                                printf("\n\nДлина расчитанного пути: %d\n",
AllLengths[needIndex]);                 // Выводим длину маршрута
                                break;
                        }
                }

                SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

                // Очишаем всё.
                AllLengths.clear();
                Cities.clear();
                AvailableCities.clear();
                IndexesCities.clear();
                delete [] VisitedCities;
                for (int i = 0; i < AmountCities; ++i)
                        delete MapCities[i];
                delete[] MapCities;
        }
        else
        {
                SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black);
                printf("Файл %s не удалось открыть для чтения.\n", TRAVELS_FILE_NAME);
                printf("Проверьте наличие файла, его расположение и наличие прав на запись.\n");
                printf("Затем перезапустите программу.\n");
                SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
        }

        fclose(*TRAVELS);
        delete TRAVELS;

        Continue();
}

void showWays() // Показать маршруты
{
        SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

        system("cls");
        FILE** TRAVELS = new FILE*;
        if (!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "rb"))
        {
                SetConsoleTextAttribute(hConsole, (WORD)Yellow | (WORD)Black);
                printf("------------------------------------------------------------\n");
                printf("--------------------------Маршруты--------------------------\n");
                printf("------------------------------------------------------------\n");
                SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

                // Читаем из файла все записи и выводим в консоль
                int numberRecord = 1;
                Trip* ShowTrip = new Trip;
                if (*TRAVELS)
                {
                        while (!feof(*TRAVELS))
```

```
                {
                        ShowTrip->distance = 0;

                        fread(ShowTrip->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
                        fread(ShowTrip->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
                        fread(&ShowTrip->distance, sizeof(int), 1, *TRAVELS);

                        if (ShowTrip->distance != 0)
                        {
                                printf("%-5d|", numberRecord);
                                ++numberRecord;
                                printf("%-25s|", ShowTrip->from);
                                printf("%-25s|", ShowTrip->to);
                                printf("%-5d|\n", ShowTrip->distance);
                                printf("---------------------------------------------------------------\n");
                        }
                }
        }

        delete ShowTrip;
    }
    else
    {
        SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black);
        printf("Файл %s не удалось открыть для чтения.\n", TRAVELS_FILE_NAME);
        printf("Проверьте наличие файла, его расположение и наличие прав на запись.\n");
        printf("Затем перезапустите программу.\n");
        SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
    }
    if (*TRAVELS) fclose(*TRAVELS);
    delete TRAVELS;

    Continue();
}

void addWay() // Добавить маршрут.
{
    SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

    system("cls");
    FILE** TRAVELS = new FILE*;
    if (!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "rb+"))
    {
        SetConsoleTextAttribute(hConsole, (WORD)Yellow | (WORD)Black);
        printf("---------------------------------\n");
        printf("---------Добавление-записи--------\n");
        printf("---------------------------------\n\n");
        SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

        std::vector<Trip> AllTrips; // Сюда запишем все поездки из файла.
        printf("Введите данные добавляемой записи:\n\n");
        // Запросим добавляемую запись.
        Trip* AddedTrip = new Trip();
```

```
		printf("Пункт отправления: ");
		gets_s(AddedTrip->from, LENGTH_CITY_NAME);
		printf("Пункт Назначения: ");
		gets_s(AddedTrip->to, LENGTH_CITY_NAME);
		printf("Расстояние: ");
		scanf_s("%d", &AddedTrip->distance);

		// Добавим запись, которую ввёл пользователь, и обратную ей (предполагается, что пути
симметричны).
		AllTrips.push_back(*AddedTrip);
		Trip* ReverseAddedTrip = new Trip(AddedTrip->to, AddedTrip->from, AddedTrip->distance);
		AllTrips.push_back(*ReverseAddedTrip);

		delete AddedTrip;
		delete ReverseAddedTrip;

		// Читаем из файла все поездки.
		Trip* TripFromFile = new Trip;
		if (*TRAVELS)
		{
			fseek(*TRAVELS, SEEK_SET, 0);

			while (!feof(*TRAVELS))
			{
				TripFromFile->distance = 0;

				fread(TripFromFile->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
				fread(TripFromFile->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
				fread(&TripFromFile->distance, sizeof(int), 1, *TRAVELS);

				if (TripFromFile->distance != 0) AllTrips.push_back(*TripFromFile);
			}
		}

		delete TripFromFile;

		// При помощи быстрой сортировки из STL сортируем вектор поездок в
лексикографическом порядке возрастания.
		std::sort(begin(AllTrips), end(AllTrips), [](const Trip& Obj1, const Trip& Obj2)
			{
				if (strcmp(Obj1.from, Obj2.from) == 0)
					return strcmp(Obj1.to, Obj2.to) == -1;
				else
					return strcmp(Obj1.from, Obj2.from) == -1;
			});

		// Открываем теперь уже чистый файл для записи всех поездок вместе с добавленной.
		fclose(*TRAVELS);
		remove(TRAVELS_FILE_NAME);
		if (!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "wb"))
		{
			if (*TRAVELS)
			{
```

```
                                fseek(*TRAVELS, SEEK_SET, 0);

                                for (const Trip& trip : AllTrips)
                                {
                                        fwrite(trip.from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
                                        fwrite(trip.to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
                                        fwrite(&trip.distance, sizeof(int), 1, *TRAVELS);
                                }
                        }
                }
        }
        else
        {
                SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black);
                printf("Файл %s не удалось открыть для дозаписи.\n", TRAVELS_FILE_NAME);
                printf("Проверьте наличие файла, его расположение и наличие прав на запись.\n");
                printf("Затем перезапустите программу.\n");
                SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
        }

        if(*TRAVELS) fclose(*TRAVELS);
        delete TRAVELS;

        Continue();
}

void delWay() // Удалить маршрут.
{
        SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

        system("cls");
        FILE** TRAVELS = new FILE*;

        if (!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "rb"))
        {
                // Запрашиваем данные записи для удаления.
                SetConsoleTextAttribute(hConsole, (WORD)Yellow | (WORD)Black);
                printf("------------------------------------------------\n");
                printf("-----------------Удаление-записи----------------\n");
                printf("------------------------------------------------\n\n");
                SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);

                printf("Введите данные записи, которую требуется удалить:\n\n");
                Trip* TripToDelete = new Trip;
                printf("Пункт отправления: ");
                gets_s(TripToDelete->from, LENGTH_CITY_NAME);
                printf("Пункт назначения: ");
                gets_s(TripToDelete->to, LENGTH_CITY_NAME);
                printf("Расстояние: ");
                scanf_s("%d", &TripToDelete->distance);
                // Создаём запись, обратную введённой, её тоже удалим из списка поездок, поскольку ранее добавлялись обе.
```

```
		Trip* reverseTripDelete = new Trip(TripToDelete->to, TripToDelete->from, TripToDelete->distance);

		std::vector<Trip>AllTripsWithoutDeleted; // Сюда поместим все записи, не включая удаляемые, их просто пропустим.
		Trip* TripFromFile = new Trip;
		if (*TRAVELS)
		{
			fseek(*TRAVELS, SEEK_SET, 0);

			while (!feof(*TRAVELS))
			{
				TripFromFile->distance = 0;

				fread(TripFromFile->from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
				fread(TripFromFile->to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
				fread(&TripFromFile->distance, sizeof(int), 1, *TRAVELS);

				// Сравниваем запись из файла с записями, которые введены пользователем для удаления.
				// Если записи не одинаковые, заталкиваем её в вектор.
				if (TripFromFile->distance != 0 &&
					(strcmp(TripFromFile->from, TripToDelete->from) || strcmp(TripFromFile->to, TripToDelete->to) || TripFromFile->distance != TripToDelete->distance) &&
					(strcmp(TripFromFile->from, reverseTripDelete->from) || strcmp(TripFromFile->to, reverseTripDelete->to) || TripFromFile->distance != reverseTripDelete->distance))
						AllTripsWithoutDeleted.push_back(*TripFromFile);
			}
		}

		// Переписываем вновь открытый файл, добавляя все записи, кроме удалённых
		if (*TRAVELS) fclose(*TRAVELS);
		remove(TRAVELS_FILE_NAME);

		if (!fopen_s(TRAVELS, TRAVELS_FILE_NAME, "ab"))
		{
			if (*TRAVELS)
			{
				fseek(*TRAVELS, SEEK_SET, 0);
				for (const Trip& Item : AllTripsWithoutDeleted)
				{
					fwrite(Item.from, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
					fwrite(Item.to, sizeof(char), LENGTH_CITY_NAME, *TRAVELS);
					fwrite(&Item.distance, sizeof(int), 1, *TRAVELS);
				}
			}
		}
	}
	else
	{
		SetConsoleTextAttribute(hConsole, (WORD)Red | (WORD)Black);
		printf("Файл %s не удалось открыть для изменение.\n", TRAVELS_FILE_NAME);
		printf("Проверьте наличие файла, его расположение и наличие прав на запись.\n");
```

```
            printf("Затем перезапустите программу.\n");
            SetConsoleTextAttribute(hConsole, (WORD)LightBlue | (WORD)Black);
        }

        if (*TRAVELS) fclose(*TRAVELS);
        delete TRAVELS;

        Continue();
}
```