**Содержание**

[Введение 3](#_Toc197710283)

[Постановка задачи «Шары» 4](#_Toc197710284)

[Ход решения задачи «Шары» 5](#_Toc197710285)

[Постановка задачи «Расстояние между городами» 8](#_Toc197710286)

[Ход решения задачи «Расстояние между городами» 9](#_Toc197710287)

[Постановка задачи «Минное поле» 12](#_Toc197710288)

[Ход решения задачи «Минное поле» 13](#_Toc197710289)

[Постановка задачи «Прямоугольник» 17](#_Toc197710290)

[Ход решения задачи «Прямоугольник» 18](#_Toc197710291)

[Постановка задачи «Перевозки» 21](#_Toc197710292)

[Ход решения задачи «Перевозки» 22](#_Toc197710293)

[Заключение 25](#_Toc197710294)

[Список литературы 26](#_Toc197710295)

**Введение**

В современном мире информационные технологии играют ключевую роль во многих сферах деятельности. Разработка программного обеспечения является одним из наиболее востребованных направлений, а язык программирования C# занимает лидирующие позиции среди разработчиков.

C# – объектно-ориентированный язык программирования, разработанный компанией Microsoft и используемый для создания приложений, работающих на платформе .NET. Он сочетает в себе простоту и мощь, что делает его идеальным инструментом для решения разнообразных задач.

Целью данной расчётно-графической работы является изучение основ и продвинутых функций языка программирования C#, а также применение его на практике при решении олимпиадных задач. В ходе работы будут рассмотрены основные понятия и концепции C#, такие как работа со строками и числами, использование логических конструкций и функций. Также будет уделено внимание тестированию, отладке и оптимизации программного кода.

**Постановка задачи «Шары»**

В пространстве расположен шар, заданный координатами своего центра **XN**, **YN**, **ZN** и радиусом **RN**.

Предполагается добавить в пространство еще максимум **M** шаров в заданной последовательности. Каждый добавляемый шар также характеризуется координатами своего центра **Xi**, **Yi**, **Zi** и радиусом **Ri** (1 ≤ **i** ≤ **M**).

Добавление шаров осуществляется до возникновения ситуации, когда в пространстве каждый шар будет пересекаться с каким-нибудь другим.

Требуется определить, после добавления какого шара каждый шар будет пересекаться с каким-нибудь другим.

**Примечание**:

* взаимное пересечение абсолютно всех шаров не требуется, т.е. решением является и наличие непересекающихся групп пересекающихся шаров;
* касание шаров не является пересечением.

**Входной файл**

Первая строка содержит четыре вещественных числа **XN**, **YN**, **ZN**, **RN**, разделенных пробелами - параметры исходного шара (|**XN**, **YN**, **ZN**| ≤ 30000; 0 < **RN** ≤ 30000).

Вторая строка содержит целое число **M** - количество шаров, которые предполагается добавить (1 ≤ **M** ≤ 5000).

Следующие **M** строк содержат по четыре вещественных числа **Xi**, **Yi**, **Zi**, **Ri** - параметры **i**-го шара (1 ≤ **i** ≤ **M**; |**Xi**, **Yi**, **Zi**| ≤ 30000; 0 < **Ri** ≤ 30000). Информация по добавляемым шарам расположена в порядке очередности добавления шара.

**Выходной файл** должен содержать номер шара, после добавления которого в пространстве не останется ни одного не пересекающегося с другими шара. Если во время добавления шаров такой ситуации не произошло, и после добавления всех шаров в пространстве остался хотя бы один шар, не пересекающийся с другими, то выводится **0** (ноль).

**Ход решения задачи «Шары»**

Для решения задачи требуется определить, после добавления какого шара все шары в пространстве будут пересекаться хотя бы с одним другим шаром. При этом касание не считается пересечением.

Первым шагом является чтение данных из входного файла. Первая строка содержит параметры исходного шара: его координаты XN YN ZN и радиус RN. Затем считывается число M – количество шаров, которые предполагается добавить. Далее следуют M строк, каждая из которых содержит параметры очередного шара Xi Yi Zi Ri.

После получения всех начальных данных начинается процесс последовательного добавления новых шаров. После добавления каждого нового шара проверяется состояние системы: нужно определить, не стал ли каждый шар пересекаться хотя бы с одним другим. Для этого необходимо уметь вычислять расстояние между центрами двух шаров и сравнивать его с суммой их радиусов. Если расстояние меньше этой суммы – шары пересекаются.

Каждый новый шар добавляется в список существующих. После этого проводится проверка на наличие изолированных шаров – то есть таких, которые не пересекаются ни с одним из остальных. Как только такой изолированный шар находится, можно сразу перейти к добавлению следующего. Если же после добавления текущего шара все шары оказались взаимосвязаны (каждый имеет хотя бы одно пересечение), то именно этот шар является искомым и мы возвращаем его порядковый номер.

Процесс останавливается либо при выполнении условия полной связанности, либо когда все M шаров добавлены, но условие так и не было достигнуто – в таком случае выводится 0.

Таким образом, реализация сводится к последовательному добавлению шаров, поддержанию списка уже добавленных объектов и регулярной проверке наличия изолированных элементов. Основное внимание уделяется эффективности вычисления связности, чтобы программа могла справиться с максимальным количеством шаров за приемлемое время.

Итоговый код программы представлен на рисунке 1.1.



Рисунок №1.1 – Код программы.

Проверим правильность работы программы на тестах (рис 1.2).

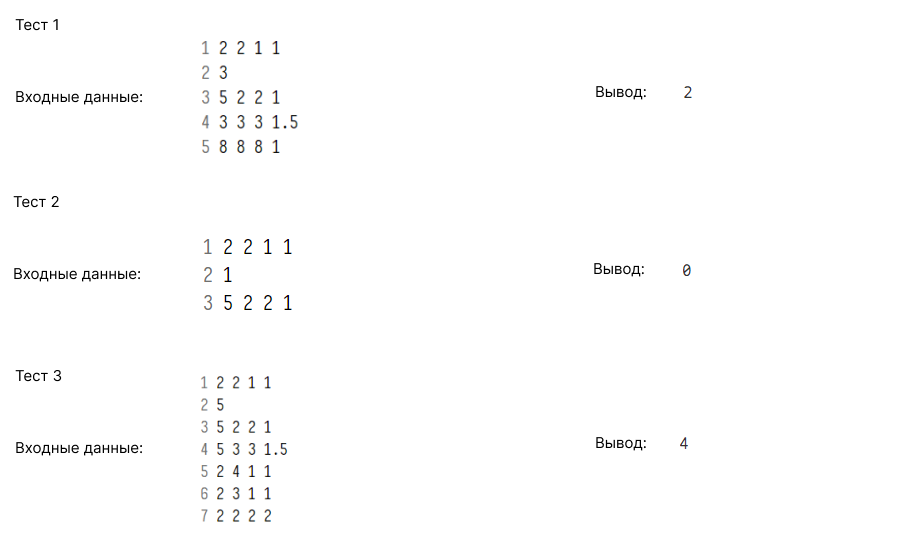


Рисунок №1.2 – Тесты.

**Постановка задачи «Расстояние между городами»**

На поверхности некоторой планеты, представляющей собой идеальный шар радиуса **R**, заданы координаты двух городов в виде двух чисел – широты и долготы.

Требуется определить минимальное расстояние по поверхности планеты между этими городами.

**Примечание**

Пары (широта, долгота) уникальны.

**Входной файл**

Первая строка содержит два целых числа **S1** и **D1** – широту и долготу первого города в градусах (-90 ≤ **S1** ≤ 90; 0 ≤ **D1** ≤ 359).

Вторая строка содержит два целых числа **S2** и **D2** – широту и долготу второго города в градусах (-90 ≤ **S2** ≤ 90; 0 ≤ **D2** ≤ 359).

Третья строка содержит целое число **R** - радиус планеты (1 ≤ **R** ≤ 30000).

**Выходной файл**

Должен содержать одно вещественное число – минимальное расстояние между городами по поверхности планеты, выведенное с тремя знаками после запятой.

**Ход решения задачи «Расстояние между городами»**

Хорошо, вот цельный текст без каких-либо списков и форматирования:

Для решения задачи необходимо вычислить минимальное расстояние между двумя точками на поверхности сферы по заданным широте и долготе этих точек. Города расположены на идеальной сфере радиуса R поэтому расстояние между ними определяется как длина кратчайшей дуги большого круга проходящей через эти две точки.

Первым шагом является чтение входных данных. Входной файл содержит две строки с координатами городов в виде широты и долготы выраженных в градусах, а также третью строку, содержащую радиус планеты. Эти данные преобразуются в числа, которые используются для дальнейших вычислений.

Далее координаты широты и долготы переводятся из градусной меры в радианную поскольку тригонометрические функции в большинстве языков программирования работают именно с радианами. После этого точки представляются как углы на сфере и рассчитывается угол между ними.

Для вычисления угла используется формула из сферической тригонометрии: косинус угла между двумя точками равен произведению синусов широт плюс произведение косинусов широт на косинус разности долгот. Полученное значение может выходить за пределы допустимого диапазона из-за ошибок округления поэтому его необходимо ограничить значениями от -1 до 1.

После получения корректного значения косинуса угла находится сам угол как арккосинус этого значения. Умножая этот угол на радиус планеты получаем искомое расстояние – длину дуги по поверхности планеты между двумя городами.

В конце результат выводится с тремя знаками после десятичной точки что соответствует требованиям задачи.

Итоговый код программы представлен на рисунке 2.1.

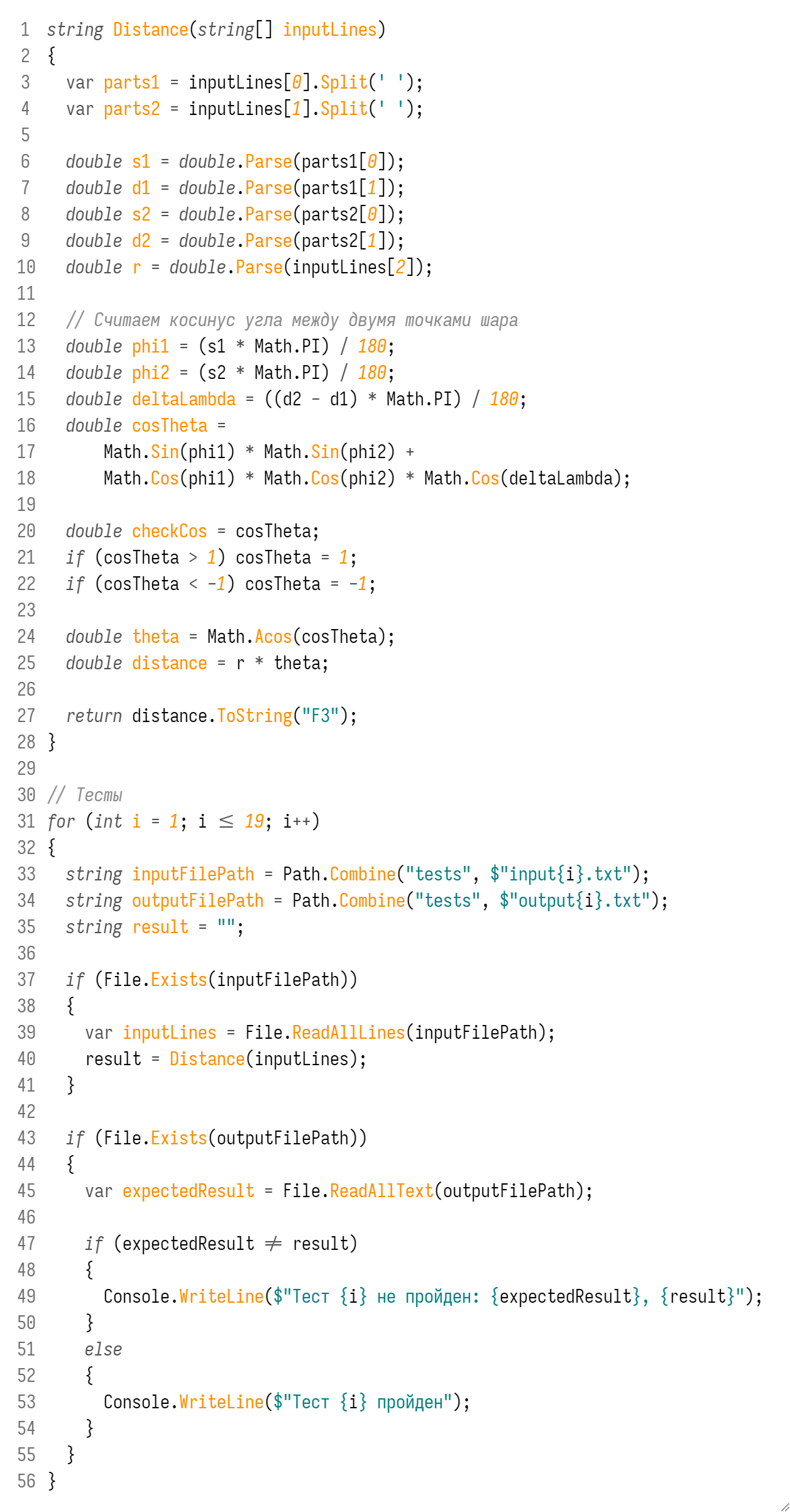


Рисунок №2.1 – Код программы.

Проверим правильность работы программы на тестах (рис 2.2).

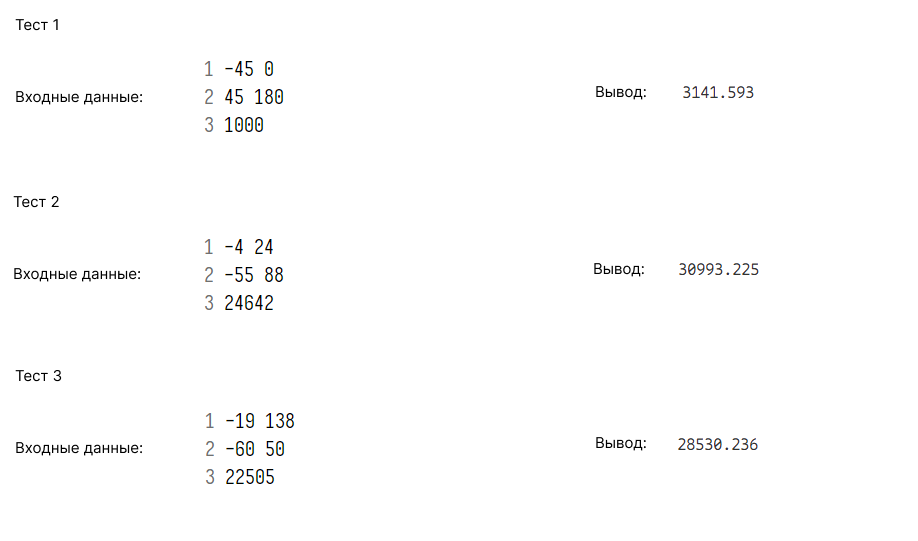


Рисунок №2.2 – Тесты.

**Постановка задачи «Минное поле»**

Минное поле представляет собой прямоугольник, разделенный на квадратные зоны одинакового размера. С помощью средств современной разведки было выяснено, сколько мин заложено в каждой зоне. Для разминирования каждой мины требуется одинаковое время. Для прохода по зоне, необходимо разминировать все мины на ней.

Требуется найти один из вариантов прохода в минном поле, требующий минимального времени для разминирования.

**Примечания:**

* размер поля **N**\***M**, 0 < **N** ≤ 200, 0 < **M** ≤ 400;
* количество мин в каждой зоне от 0 до 100.

**Входной файл** содержит:

* в первой строке числа **N** и **M**, разделенные пробелом (количество рядов на минном поле и зон в ряде соответственно);
* в следующих **N** строках содержится по **M** целых чисел, разделенных пробелом – количество мин в зонах **i**-го ряда (1 ≤ **i** ≤ **N**).

**Выходной файл** должен содержать **N** строк. В каждой строке указывается номер зоны в **i**-м ряду (1 ≤ **i** ≤ **N**), через которую лежит найденный проход.

**Ход решения задачи «Минное поле»**

Для решения задачи нужно найти путь через минное поле, который требует минимального времени на разминирование. Минное поле представляет собой прямоугольник размером N на M, где в каждой ячейке указано количество мин. Движение осуществляется по рядам сверху вниз, и разрешено переходить из текущей ячейки в следующий ряд либо прямо, либо по одной из двух диагоналей – влево-вниз или вправо-вниз. При этом необходимо пройти от какой-либо точки первого ряда до какой-либо точки последнего ряда так, чтобы суммарное количество мин, которые придётся разминировать на пути, было минимальным.

Входной файл содержит сначала размеры поля N и M, затем N строк по M чисел – количество мин в каждой зоне. Выходной файл должен содержать N строк, в каждой из которых указан номер зоны в соответствующем ряду, через которую проходит один из кратчайших по количеству мин путей.

Решение основано на методе динамического программирования. Создается двумерный массив minSums того же размера, что и исходное поле, где каждое значение minSums[i][j] будет хранить минимальное количество мин, необходимое для достижения этой ячейки. Также создается массив prevLine, в котором сохраняются индексы предыдущих ячеек для восстановления пути.

Первый шаг – заполнение начального ряда: для каждой ячейки первого ряда (i = 0) минимальная сумма равна просто количеству мин в ней самой, так как других вариантов прийти туда нет.

Затем для каждого следующего ряда пересчитывается значение minSums[i][j], основываясь на значениях предыдущего ряда. Для этого проверяются все допустимые направления движения в текущую ячейку и выбирается то, которое даёт минимальную сумму. Одновременно записывается информация о том, из какой ячейки был сделан переход, чтобы в дальнейшем восстановить путь.

После обработки всех рядов находится ячейка в последнем ряду с минимальным значением суммы мин. С неё начинается обратный проход по массиву prevLine для восстановления оптимального пути от последнего ряда к первому.

Наконец, результат выводится в виде N строк, где в каждой строке указан номер зоны (начиная с единицы), через которую прошёл найденный путь.

Итоговый код программы представлен на рисунке 3.1.



Рисунок №3.1 – Код программы.

Проверим правильность работы программы на тестах (рис 3.2).

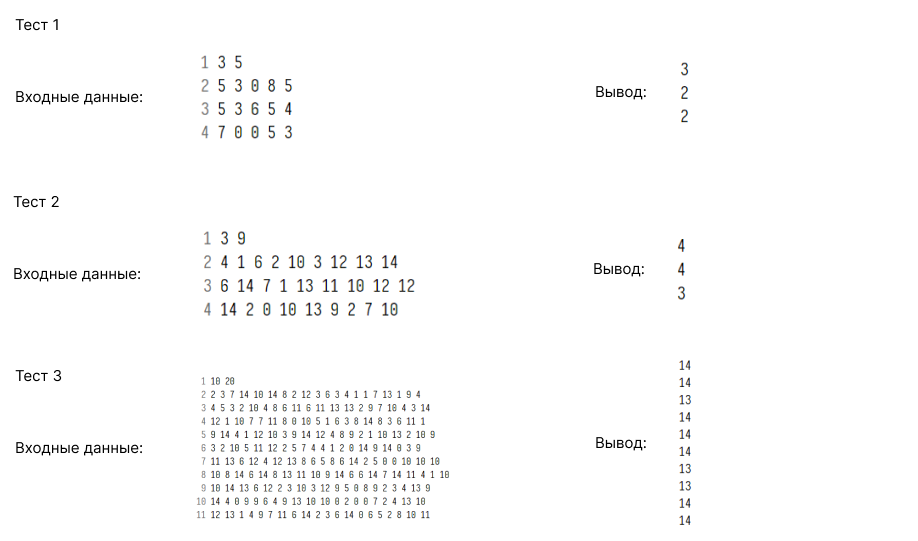


Рисунок №3.2 – Код программы.

**Постановка задачи «Прямоугольник»**

Имеется прямоугольная область размером **N**×**M** клеток, некоторые из которых закрашены.

Требуется найти прямоугольник максимальной площади, не содержащий закрашенные клетки. Стороны прямоугольника должны быть параллельны краям прямоугольной области.

**Входной файл** содержит:

* в первой строке два числа **N** и **M** – число столбцов и строк прямоугольной области соответственно (1 ≤ **N**, **M** ≤ 250);
* во второй строке число **K** – количество закрашенных клеток (0 ≤ **K**  ≤ **(N**\***M**-1));
* следующие **K** строк содержат по 2 числа **X** и **Y** – координаты (столбец и строка соответственно) закрашенной клетки.

**Выходной файл** должен содержать площадь найденного прямоугольника.

**Ход решения задачи «Прямоугольник»**

Для решения задачи необходимо найти прямоугольник максимальной площади, который может быть размещен внутри заданной прямоугольной области, не включающий ни одной закрашенной клетки. При этом стороны прямоугольника должны быть параллельны краям области.

Входные данные содержат размеры области N (количество столбцов) и M (количество строк), количество закрашенных клеток K, а также координаты этих клеток. Закрашенные клетки помечаются на двумерном массиве, представляющем собой булеву матрицу размером M×N, где true означает наличие закрашенной клетки.

Программа перебирает все возможные прямоугольники, задавая их левый верхний и правый нижний углы. Для каждого такого прямоугольника проверяется, содержит ли он хотя бы одну закрашенную клетку. Если таких клеток нет, вычисляется его площадь и если она больше ранее найденной, значение максимальной площади обновляется.

Таким образом, алгоритм сводится к полному перебору всех возможных прямоугольников и проверке для каждого из них наличия закрашенных клеток внутри. После завершения перебора выводится максимальная найденная площадь такого прямоугольника, не содержащего закрашенных ячеек.

Итоговый код программы представлен на рисунке 4.1.

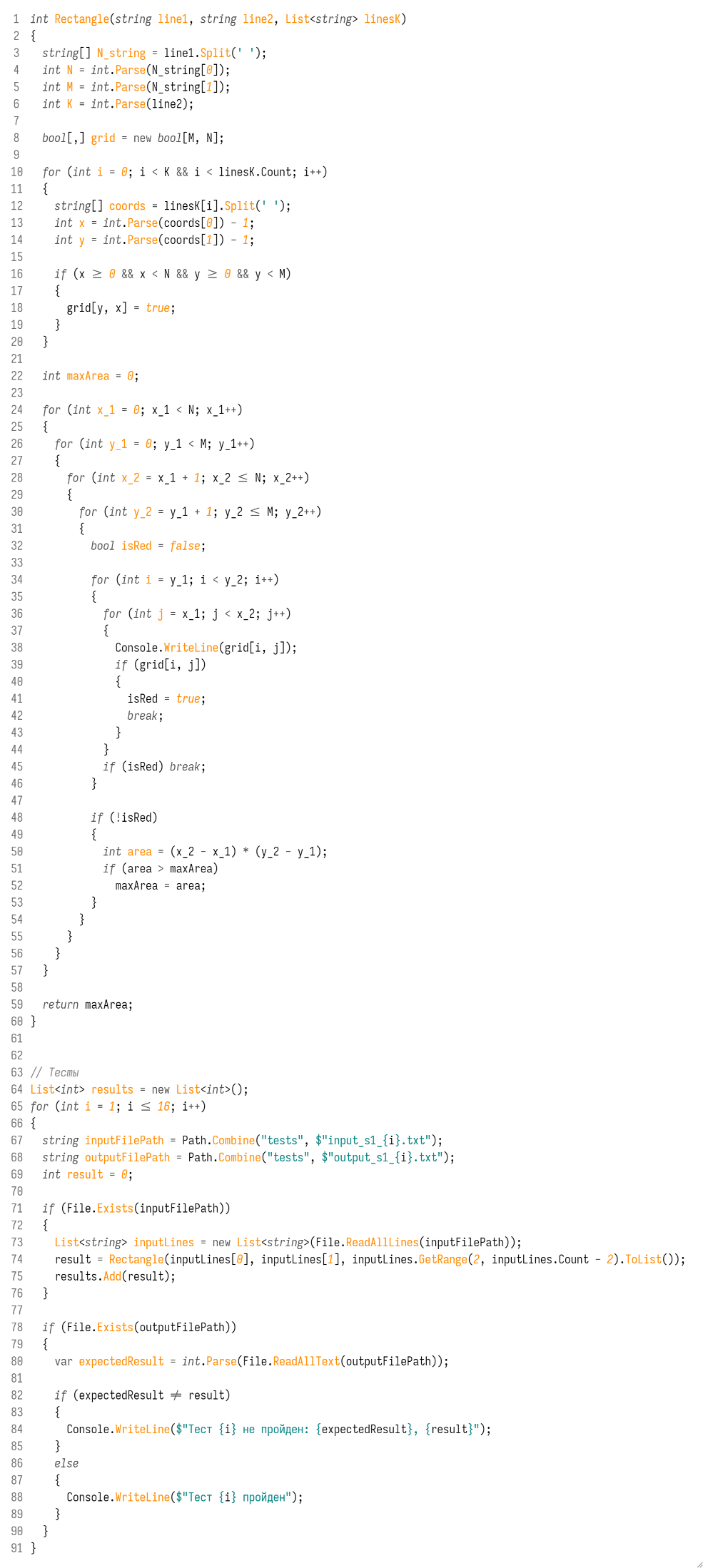


Рисунок №4.1 – Код программы.

Проверим правильность работы программы на тестах (рис.4.2).

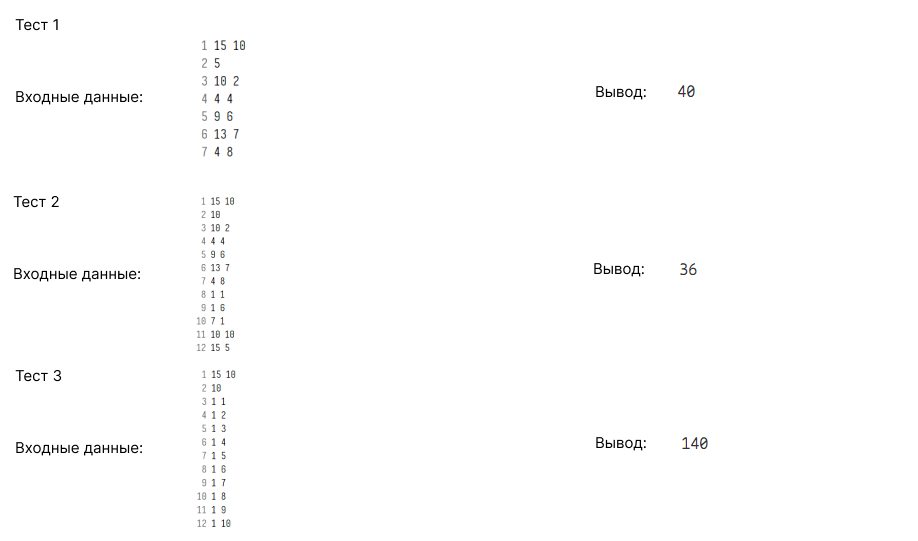


Рисунок №4.2 – Тесты.

**Постановка задачи «Перевозки»**

Таблица стоимости перевозок устроена следующим образом: числа, стоящие на пересечении строк и столбцов таблиц, обозначают стоимость перевозок между соответствующими соседними станциями. Если пересечение столбца и строки пусто, то станции не являются соседними. Стоимость перевозки по маршруту складывается из стоимостей перевозок между соседними станциями. Перевозки между населенными пунктами осуществляют 3 компании, представившие стоимость своих услуг в табличной форме. Определить компанию, которая обеспечивает минимальную стоимость перевозки между заданными станциями.

**Формат входных данных**

Во входном файле записано число N (1<=N<=100), определявшее количество станций. Затем идет описание стоимости перевозки по транспортной сети для каждой компании, где каждое соединение задается 5 числами - номерами узлов, которые она соединяет и три числа, определяющие стоимость перевозки для каждой компании. Станции, между которыми необходимо определить минимальную стоимость перевозки, задается пользователем.

**Формат выходных данных**

На экран вывести сообщение о стоимости перевозки и компании, которая гарантирует эту стоимость

**Ход решения задачи «Перевозки»**

Для решения задачи необходимо определить, какая из трёх транспортных компаний обеспечивает минимальную стоимость перевозки между двумя заданными станциями. Стоимость перевозки по маршруту складывается из стоимостей отдельных участков между соседними станциями. Для каждой компании построена своя сеть соединений и свои цены на каждом участке.

Входные данные включают количество станций N, описание транспортной сети в виде списка соединений между парами станций и три стоимости перевозки для каждой компании. Также задаются начальная и конечная станции, между которыми нужно найти маршрут с минимальной стоимостью.

Алгоритм решения основан на использовании алгоритма Дейкстры для поиска кратчайшего пути во взвешенном графе. Для каждой компании строится собственная матрица смежности, где хранится минимальная стоимость перевозки между станциями. Если между станциями существует несколько соединений от разных участков, выбирается то, которое имеет наименьшую стоимость для конкретной компании.

После построения матриц для всех трёх компаний запускается алгоритм Дейкстры от начальной станции до конечной для каждой из компаний. В результате получается три значения – минимальные стоимости перевозок для каждой из компаний.

Далее среди этих значений находится минимальное. Компания, которой оно принадлежит, считается победителем, а соответствующая стоимость выводится на экран.

Итоговый код программы представлен на рисунке 5.1.



Рисунок №5.1 – Код программы.

Проверим правильность работы программы на тестах (рис.5.2).

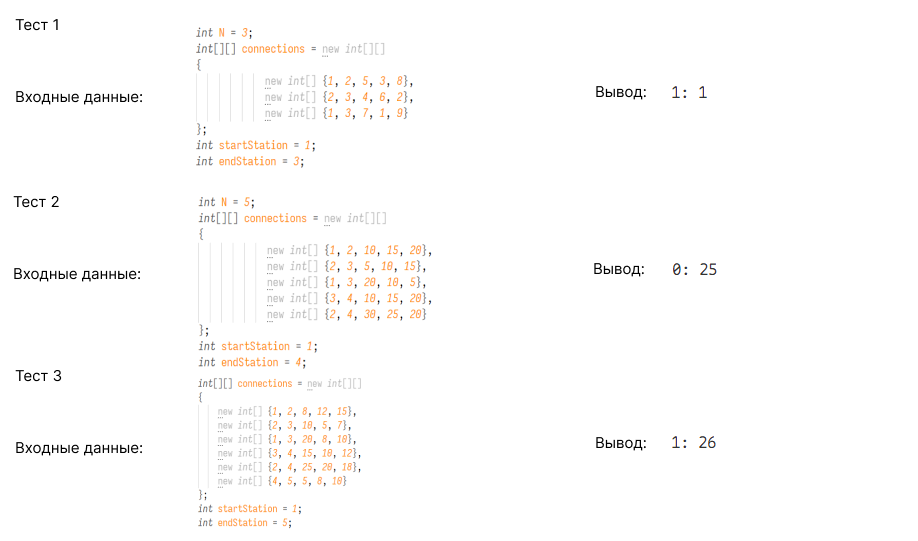


Рисунок №5.2 – Тесты.

**Заключение**

В ходе выполнения работы были изучены основы и продвинутые функции языка программирования C# и применены полученные знания для решения поставленных задач. Были рассмотрены основные концепции данного языка программирования, проведена работа с классами, объектами и методами. Также были изучены возможности использования стандартных библиотек и работа в среде Visual Studio.

В процессе выполнения работы были созданы несколько программ, демонстрирующих различные аспекты применения языка C# и его возможностей. Были проведены тестирование и отладка созданных программ, что позволило выявить и устранить ошибки и недочёты.

В результате выполнения работы были получены навыки программирования на языке C#, а также опыт разработки и тестирования программного обеспечения. Полученные знания и умения могут быть применены в дальнейшей профессиональной деятельности или при выполнении других проектов.

**Список литературы**

1. Флёнов М. Е. Библия C#. 6-е издание. – БХВ, Санкт-Петербург, 2024 – 512 с.
2. Албахари Д. C#12. Справочник. Полное описание языка. – Диалектика, Санкт-Петербург, 2024 – 272 с.
3. Полное руководство по языку программирования С# 13 и платформе .NET 9: <https://metanit.com/sharp/tutorial/> (дата обращения: 14.11.2024).
4. Что нужно знать при написании алгоритмов на .NET: <https://habr.com/ru/articles/863950/> (дата обращения: 25.10.2024).
5. Особенности строк в .NET: <https://habr.com/ru/articles/172627/> (дата обращения: 05.12.2024).