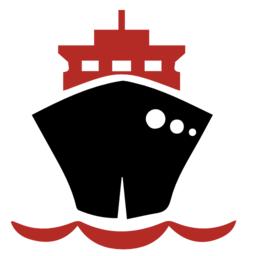
# StrawberryShip



# Методы оптимизации

Задача поиска маршрута с близкой к минимальной цене

Подготовили студенты группы ИВТ-11М

Григорьев А.В.

Зиновьев Н.А.

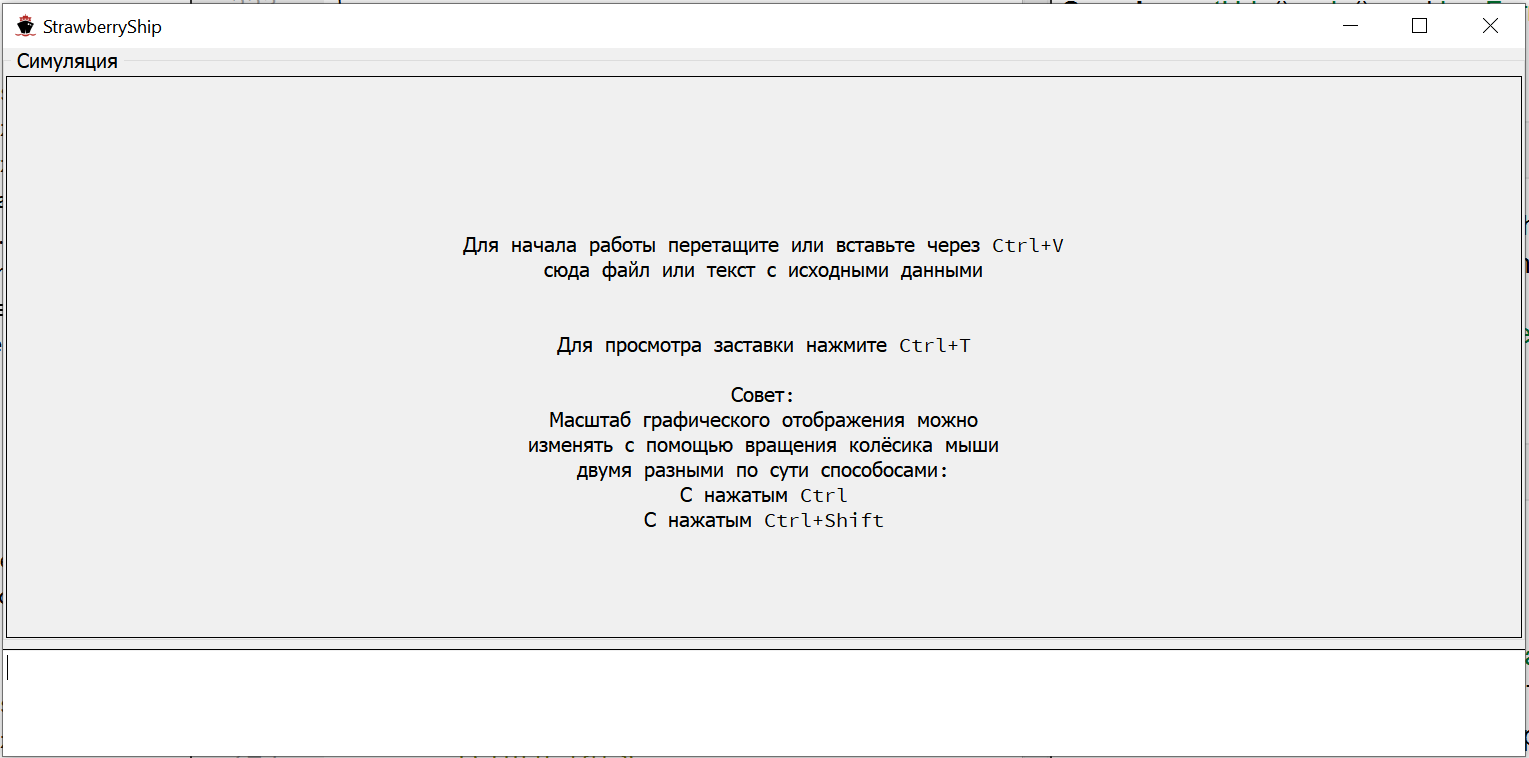
Королёва В.А.

Петров А.А.

# Проверка исходных данных

Исходные данные в данной реализации программы имеют три этапа проверки.

Но перед их описанием стоит отметить, как исходные данные могут поступить в программу. Для этого реализованы два способа. Во-первых, программа принимает данные через механизм перетаскивания (drag&drop), во-вторых, через буфер обмена (событие на Ctrl+V). Исходной информацией же может служить как файл (mime: text/uri-list), так и текст (mime: text/plain). Итого 4 способа, но они внутри сводятся к одному – к построчному считыванию и обработке информации.



## Первый этап

Во время первого этапа происходит считывание файла. Здесь происходят базовые проверки корректности структур данных.

На первом этапе проверяется:

На уровне между блоками данных:

* Наличие повторных объявлений блока (например, два раза встречается блок SHIP и т.д.). Это ошибка.
* Наличие излишних символов завершения блока без предшествующего начала блока (его объявления). Это ошибка.
* Наличие строк вне блока, не относящихся к названию блока (например, вместо SHIP встретили SHIPPS). Это ошибка.
* Отсутствие обязательных блоков (trac, mone, ship). Это ошибка.
* Отсутствие необязательных блоков (icee). Предупреждение.

На уровне содержимого каждого блока проверяется:

* Цифры там, где они должны находиться по ожидаемой структуре должны быть конвертируемы в формат int. Иначе ошибка.
* Число аргументов (считается через regexp по разделению (split) по ‘\\s+’) должно совпадать с ожидаемым. Иначе ошибка.
* Встреча в текущей строке начала следующего блока (то есть пропуск символа завершения блока). Это ошибка.
* Встреча в текущей строке конца файла. Это ошибка.

На уровне блока trac дополнительных проверок не предусмотрено.

На уровне блока ship дополнительно проверяется:

* Тип корабля равен ‘H’ или ‘S’. Иначе ошибка.
* Количество аргументов в строке соотносится с типом корабля (у ‘H’ их больше). Иначе ошибка.

На уровне блока mone дополнительно проверяется:

* Первая значимая строка содержит один аргумент – цифру. Иначе ошибка.
* Далее идут строки с кораблями. Для них проверятся тип (см выше). При некорректном типе ошибка.

На уровне блока icee дополнительных проверок не предусмотрено.

На уровне блока path куча всего проверяется:

* В момент времени, когда ожидается (пояснить смысл слова ожидается сложно) запись о корабле встречена запись о пути корабля. И обратная ситуация. Это ошибка.
* Проверка типа корабля (один из двух доступных). Иначе ошибка.
* В строке с именем корабля должна находиться положительная цифра о последующем числе записей. Иначе ошибка.

## Второй этап

Второй этап проходит над уже считанными данными, если в течение первого этапа не было найдено ошибок.

Проверки второго этапа тоже разделены на логические блоки.

Проверка пределов:

* Число трасс не более 1000
* Шаг раскладки датчиков положительный
* Скорость кораблей должна быть положительной
* Число датчиков на укладчике должно быть положительным
* Число датчиков на укладчике не должно превышать 20000
* Стоимость аренды датчика должна быть положительной
* Стоимость аренды корабля должна быть положительной
* Время закрытия трассы должно быть менее времени открытия
* Номер трассы в ice должен соответствовать реальной трассе (индексация от 1).
* У кораблей в path должны быть те операции, которые они реально могут иметь.
* Время для операций в path не должно быть отрицательным.
* В path описаны маршруты только двух кораблей.
* В path описаны маршруты H и S типа.

Проверка отсутствия необходимых данных:

* Нет трасс. Это ошибка.
* В ship нет кораблей типа S и/или H. Ошибка.
* В блоке mone не было данных и поэтому сейчас у нас нет информации о цене датчиков. Это ошибка.

Проверка на ошибки с именами кораблей:

* В каждом из блоков ship, mone и path проверяется (независимо друг от друга), что имена кораблей в совокупности с их типом являются уникальными в рамках данного блока.
* Проверяется, что в mone есть все корабли из ship и в ship есть все корабли из mone. Что все корабли из path есть в ship.

Производится проверка на пересечение по некоторой длине одной трассы другой. То есть, если трассы пересекаются и это пересечение имеет ненулевую длину, то это считается ошибкой. Это вполне логичная проверка.

## Этап два с половиной

До этого для проверок использовались простые структуры данных с избыточной информацией (данные о кораблях, к примеру, были в двух блоках и частично пересекались). После прохождения второго этапа создаются уже более подходящие для дальнейшей работы структуры данных. И здесь уже удобно осуществить ещё одну проверку – что хотя бы один из обработчиков имеет столько датчиков, что сможет полностью обработать самую требовательную к датчикам трассу.

## Третий этап

Если в исходных данных есть блок path и предыдущие проверки не выявили ошибки, то проверяется корректность для path. Если ранее проверки осуществились над сырыми структурами с данными, на третьем этапе используются уже подготовленные данные.

Проверяются ошибки также по категориям.

Проверка на ошибки начала/конца:

* Судно должно начинать маршрут из 0,0. Иначе ошибка.
* Судно должно заканчивать маршрут в 0,0. Иначе ошибка.
* Судно должно заканчивать маршрут с действием 0 (простой). Иначе ошибка.

Проверка на ошибки времени/скорости:

* Время в каждой следующей записи не уменьшается. Иначе ошибка.
* При простое следующая запись имеет те же координаты. Иначе ошибка.
* При простое в течение нулевого времени появится предупреждение (уже поздно, но стоит отметить, что предупреждения сами по себе не относятся к категории ошибок).
* Судно не должно изменять свои координаты, если между записями прошло 0 часов. Иначе ошибка.
* Судно не должно превышать свою максимальную скорость. Иначе ошибка.
* Судно должно проходить расстояния за минимальное количество часов (с округлением вверх до целого). Иначе ошибка.

Проверка на ошибки с трассами:

* Если происходит активное действие (не простое перемещение и простой), то это действие должно происходить по координатам, которые образуют реальную трассу. Иначе ошибка.

Проверка на ошибки обработки трасс:

* Над каждой существующей трассой должно быть произведено три активных действия со стороны кораблей. Иначе ошибка.
* Активные действия, производимые над трассой должны включать три разных доступных типа действия. Иначе ошибка.
* Время начала прострела должно быть не менее времени завершения раскладки. Иначе ошибка.
* Время начала сбора датчиков должно быть не менее времени завершения прострела. Иначе ошибка.
* При выполнении активных операций над трассой она всё время должна быть доступной (то есть без ограничений от icee). Иначе ошибка.

Также проверяется, что в течение всего времени работы укладчика его баланс датчиков неотрицательный.

# Результаты тестирования

Из первого этапа

Для файла Ship1.txt вывод программы:

Строка 11. Ошибка при чтении в TRAC: строка имеет некорректное ожидаемому число аргументов (получено 4, ожидалось 5). Данная строка не будет учтена. Исходный текст строки: '868 253 711 141'

Задан некорректный (неположительный) шаг раскладки датчиков в TRAC для пути 169 500 334 467 ( 0 )

Задано некорректное (неположительное) максимальное число датчиков в SHIP на укладчике 'H' "BSHIP2" ( 0 )

В ICEE задан номер трассы, которого нет в TRAC ( 10 ). Данные трассы: 10 393 441

В ICEE задан номер трассы, которого нет в TRAC ( 10 ). Данные трассы: 10 118 200

В ICEE задан номер трассы, которого нет в TRAC ( 10 ). Данные трассы: 10 945954

Последние три строчки появились потому, что в 11 строке не был учён путь из-за ошибки числа аргументов

Для файла Ship2.txt вывод программы:

Строка 52. При чтении блока MONE встречено начало следующего блока ICEE - вероятно пропущен символ разделения блоков. Следующие строки будут учитываться как данные блока ICEE

Задана некорректная (неположительная) стоимость аренды датчиков в MONE ( -8 )

Задана некорректная (неположительная) стоимость аренды судна в MONE 'S' "ASHIP5" ( 0 )

Для файла Ship3.txt вывод программы:

Задано некорректное время закрытия и открытия трасс в ICEE (закрытие должно происходить раньше, чем открытие) для номера 1 закрытие: 28 открытие: 28

Для файла Ship4.txt вывод программы:

Задано некорректное время закрытия и открытия трасс в ICEE (закрытие должно происходить раньше, чем открытие) для номера 6 закрытие: 509 открытие: 509

Задано некорректное время закрытия и открытия трасс в ICEE (закрытие должно происходить раньше, чем открытие) для номера 8 закрытие: 166 открытие: 166

Задано некорректное время закрытия и открытия трасс в ICEE (закрытие должно происходить раньше, чем открытие) для номера 3 закрытие: 189 открытие: 189

В MONE пропущены описанные в SHIP корабли: H BSHIP7, H BSHIP5, H BSHIP6

Для файла Ship5.txt вывод программы:

-- ошибок не обнаружено

Для файла Ship6.txt вывод программы:

Предупреждение. В файле отсутствует блок данных SHIP

Предупреждение. В файле отсутствует блок данных ICEE

В SHIP отсутствуют корабли типа: укладчик и шутер

В MONE встречены неописанные в SHIP корабли: H BSHIP6, H BSHIP13, H BSHIP14, S ASHIP16, S ASHIP7, H BSHIP7, H BSHIP9, S ASHIP9, S ASHIP2, S ASHIP14, H BSHIP10, H BSHIP15, S ASHIP1, S ASHIP11, H BSHIP16, H BSHIP3, H BSHIP11, S ASHIP4, H BSHIP12, S ASHIP18, H BSHIP5, H BSHIP17, S ASHIP3, S ASHIP12, H BSHIP8, S ASHIP15, S ASHIP6, S ASHIP13, H BSHIP2, H BSHIP4, S ASHIP8, S ASHIP10, H BSHIP1, H BSHIP18, S ASHIP17, S ASHIP5

Для файла Ship7.txt вывод программы:

В TRAC отсутствуют трассы. Нечего делать

В SHIP отсутствуют корабли типа: укладчик и шутер

В MONE нет информации о стоимости датчиков

Для файла Ship8.txt вывод программы:

Строка 17. Ошибка при чтении в TRAC: строка имеет неконвертируемую в int запись. Данная строка не будет учтена. Аргумент по номеру 4: 'A'. Исходный текст строки: '223 833 496 305 A'

Строка 91. Ошибка при чтении в SHIP: корабль имеет некорректный тип 'W'. Данная строка не будет учтена. Исходный текст строки: 'W ASHIP8 1'

Строка 149. Ошибка при чтении в MONE: корабль имеет некорректный тип 'Q'. Данная строка не будет учтена. Исходный текст строки: 'Q ASHIP15 61000'

В ICEE задан номер трассы, которого нет в TRAC ( 0 ). Данные трассы: 0 405 470

Задано некорректное время закрытия и открытия трасс в ICEE (закрытие должно происходить раньше, чем открытие) для номера 3 закрытие: 837 открытие: 837

Задано некорректное время закрытия и открытия трасс в ICEE (закрытие должно происходить раньше, чем открытие) для номера 9 закрытие: 467 открытие: 467

Задано некорректное время закрытия и открытия трасс в ICEE (закрытие должно происходить раньше, чем открытие) для номера 5 закрытие: 376 открытие: 376

В MONE встречены неописанные в SHIP корабли: S ASHIP8

В MONE пропущены описанные в SHIP корабли: S ASHIP15

Для файла Ship9.txt вывод программы:

Строка 91. Ошибка при чтении в TRAC: строка имеет неконвертируемую в int запись. Данная строка не будет учтена. Аргумент по номеру 4: 'Q'. Исходный текст строки: '409 693 991 212 Q'

Строка 110. Ошибка при чтении в SHIP: строка имеет некорректное ожидаемому число аргументов (получено 2, ожидалось 3 или 4). Данная строка не будет учтена. Исходный текст строки: 'S 2'

Строка 147. Ошибка при чтении в SHIP: корабль имеет несоответствие данных: укладчик с 3 входными аргументами. Данная строка не будет учтена. Исходный текст строки: 'H 2 7000'

Предупреждение. В файле отсутствует блок данных ICEE

Задан некорректный (неположительный) шаг раскладки датчиков в TRAC для пути 323 155 955 500 ( -30 )

В MONE встречены неописанные в SHIP корабли: H BSHIP27, S ASHIP17

Из второго этапа

Для файла Ship1.txt вывод программы:

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 2 )

Задана некорректная операция действия в пути шутера 'S' "BSHIP1" ( 3 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

Задана некорректная операция действия в пути обработчика 'H' "ASHIP1" ( 4 )

В MONE встречены неописанные в SHIP корабли: S BSHIP1, S BSHIP2, S BSHIP3

В MONE пропущены описанные в SHIP корабли: H BSHIP1, H BSHIP2, H BSHIP3

В PATH встречены неописанные в SHIP корабли: S BSHIP1, H ASHIP1

Для файла Ship1b.txt вывод программы:

-- ошибок не обнаружено

-- стоимость аренды по маршруту из исходных данных равна 3.460.000 [101 ч / 5 дн.]

Для файла Ship1b1.txt вывод программы:

Строка 75. Ошибка при чтении строки в PATH: Встречена запись, похожая на начало маршрута следующего корабля, но для текущего маршрута судна BSHIP1 ожидалось получить ещё 1 строк. Действие по умолчанию - начать считывание следующего маршрута. Входная строка ( S ASHIP1 42 )

Строка 117. Ошибка при чтении точки в PATH: Встречен конец блока, хотя ожидалось получить ещё 1 строк для корабля ASHIP1

Для файла Ship1b2.txt вывод программы:

В PATH между записями ( 19912 25667 76 2 ) и ( 26299 17035 80 3 ) судно двигалось ( 4 ч ), но оно может пройти данное расстояние за ( 2 ч ); скорость корабля ( 8000 м/ч )

В PATH ожидалось, что время не будет уменьшаться ( 26299 17035 80 3 ) и ( 26299 17035 78 0 )

В PATH после записи о простое ( 26299 17035 78 0 ) встречена запись с отличными от предыдущих координатами ( 19912 25667 82 1 )

В PATH между записями ( 26299 17035 82 1 ) и ( 28703 23811 85 4 ) судно двигалось ( 3 ч ), но оно может пройти данное расстояние за ( 1 ч ); скорость корабля ( 8000 м/ч )

В PATH ожидалось, что время не будет уменьшаться ( 28703 23811 85 4 ) и ( 28703 23811 83 0 )

В PATH после записи о простое ( 28703 23811 83 0 ) встречена запись с отличными от предыдущих координатами ( 31322 30333 86 0 )

В PATH активное действие ( 3 ) происходит не по существующей трассе из TRAC между записями ( 26299 17035 80 3 ) и ( 26299 17035 78 0 )

В PATH активное действие ( 4 ) происходит не по существующей трассе из TRAC между записями ( 28703 23811 85 4 ) и ( 28703 23811 83 0 )

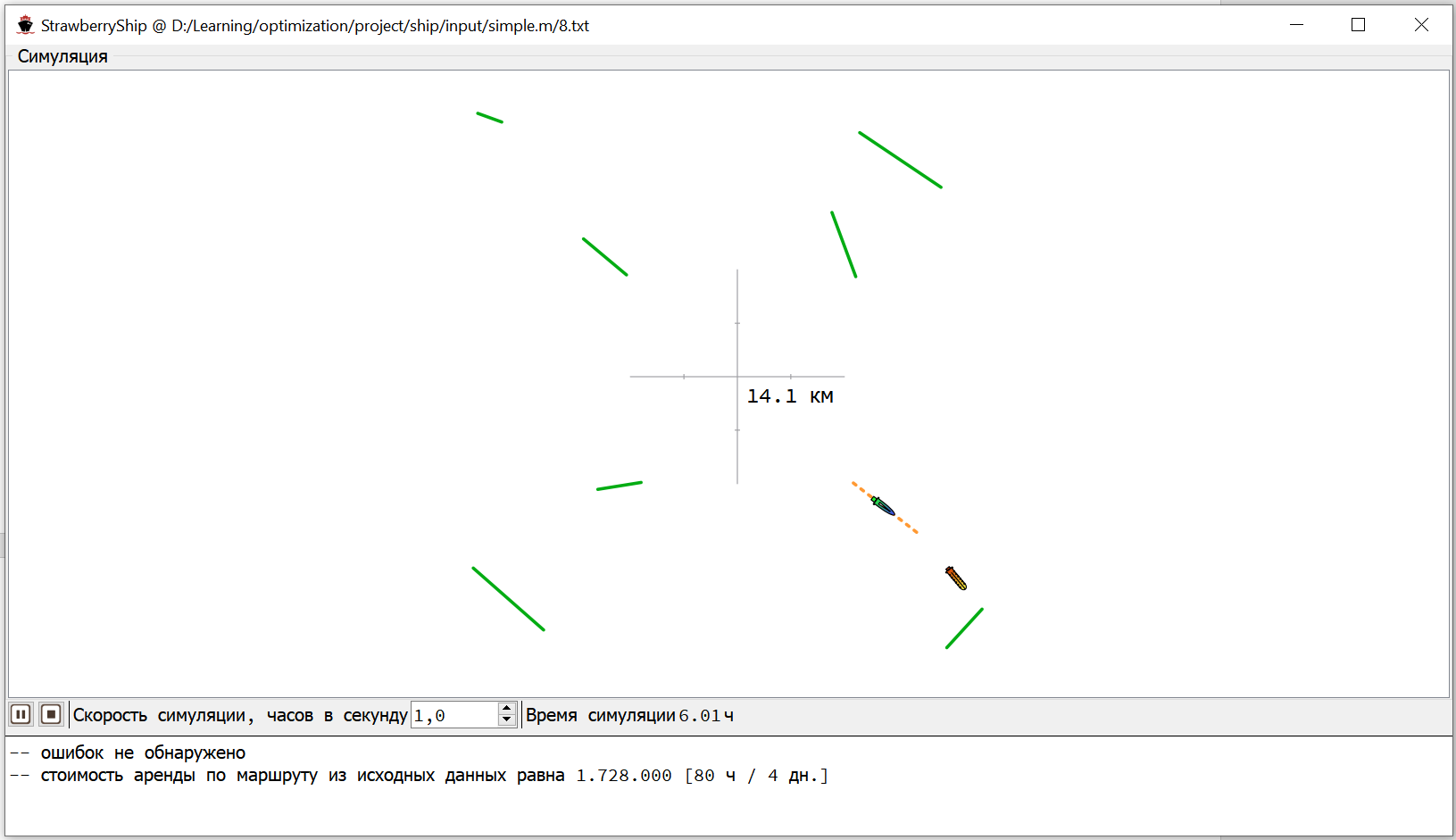
В PATH для трассы ( 19912 25667 26299 17035 ) ожидалось увидеть 3 взаимодействия с кораблями, но встречено ( 2 ), это действия ( 2 , 4 )

В PATH для трассы ( 19912 25667 26299 17035 ) ожидалось увидеть раскладку(2), прострел(4) и сбор(3), но встречены только действия ( "2 , 4" )

В PATH для трассы ( 28703 23811 31322 30333 ) ожидалось увидеть 3 взаимодействия с кораблями, но встречено ( 2 ), это действия ( 2 , 3 )

В PATH для трассы ( 28703 23811 31322 30333 ) ожидалось увидеть раскладку(2), прострел(4) и сбор(3), но встречены только действия ( "2 , 3" )

# Графическая симуляция



Симуляция становится доступной, если пройдены первые два этапа (если path нет, то корабли не движутся и можно просто пронаблюдать за положением трасс).

Если path не было, то маршрут создастся (со временем…), если был, то будет использован существующий.

Симуляция построена в виде независимого отображения во времени (то есть для построения текущего момента не важно, какой момент времени был предыдущим – мы не зависим от истории). То есть в любой момент можно задать любое время и за константное ожидание отобразится состояние, которое должно быть в это время. Благодаря этому скорость симуляции может быть любой, даже очень большой.

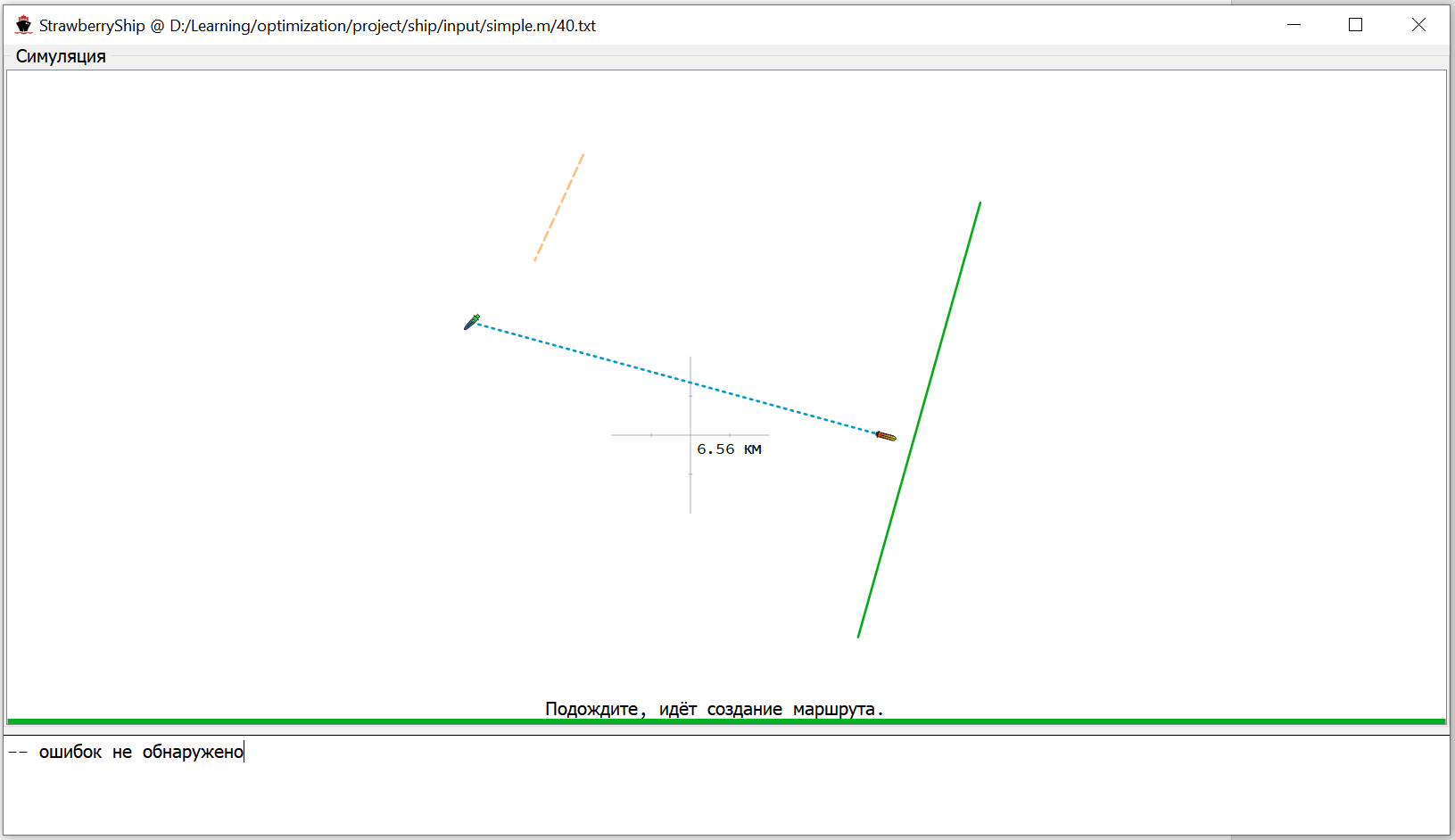
Реализована за счёт двух объектов: корабля и трассы.

При подаче в трассу конкретного времени она меняет своё состояние (цвет) в зависимости от доступности (icee) и от того, какое состояние обработки она имеет (константы с часами вносятся в этот элемент при его создании).

Корабль также через бинарный поиск по времени получает номер текущей трассы, на котором он должен находиться. И на основе некоторых параметров смещается от её начала и получает некоторый угол поворота (для плавности).

Вычисление основного маршрута происходит в отдельном потоке/потоках, поэтому графическое окно не зависает.

Вот так приложение выглядит при создании маршрута:



При вычислении маршрута показывается бесконечная анимация, где создаются три случайных трассы, а дальше по алгоритму перебора вариантов к ним строится маршрут и отображается, как он проходится. Перебор для трёх трасс не требует много вычислений и происходит в потоке графического окна. Поэтому ожидать вычисления основного маршрута становится не так скучно.

# Построение маршрута

## Целевая функция

Целевой функцией является функция, получающая два корабля и некоторый маршрут, возвращающая цену по полученным данным.

В рамках задачи добавлена вспомогательная целевая функция, принимающая заранее два корабля и высчитывающая именно часы.

В качестве аргументов целевой функции используются два массива, каждый из элементов которого содержит номер трассы (short) и сторону обхода (bool). Тогда для задания маршрута укладчика при N трассах нужно 2N элементов, где каждый номер трассы повторяется 2 раза. Для шутера же требуется N элементов.

Полученная функция довольно просто вычисляется – тысячи вычислений в секунду даже при 1000 трасс, а при малом их числе скорость доходит до нескольких миллионов вычислений.

При этом во входных данных функции возможны некорректные комбинации. Например, когда укладчик с самого начала хочет обработать трассу 1 два раза, а шутер хочет обработать трассу 2 – тогда функция сообщит о некорректности данных. Но вообще, почти везде далее будет использоваться маршрут шутера, являющийся повторением маршрута укладчика. Тогда бессмысленных вычислений не будет. Хотя это потенциально и лишает нас возможности получения самого оптимального маршрута.

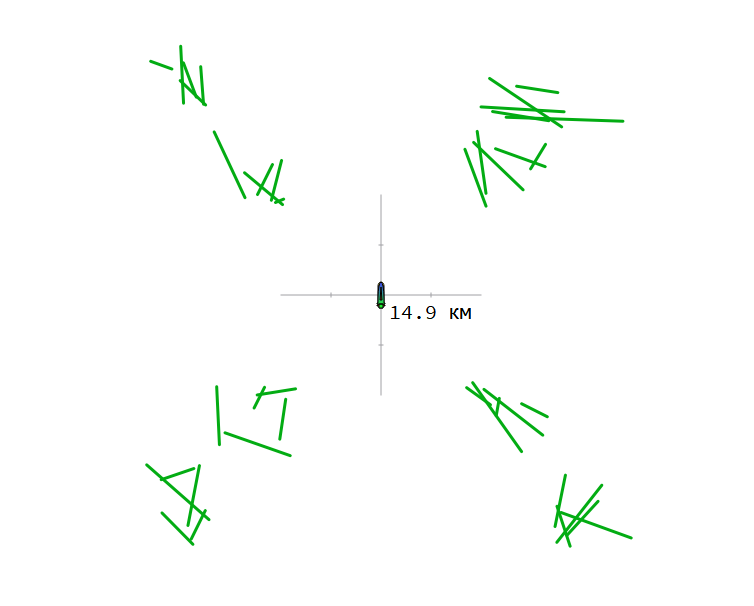
Если в процессе вычисления у обработчика окажется отрицательный баланс, то функция тоже сообщит о некорректных данных. (просто вернёт -1 час, что легко проверяется извне).

Но такая придуманная функция довольна сложна к анализу. Сложно применить к входным данным понятие направления и подобное.

Преимущество же функции в том, что она сразу учитывает ограничения (icee) при вычислении времени. То есть в целом на эти ограничения внимание больше обращаться не будет.

## Группировка трасс

Некоторые трассы могут быть скучены рядом друг с другом, поэтому желательно, чтобы они стояли последовательно (это может улучшить работу дальнейших алгоритмов). Далее на рисунке приведён пример подобной ситуации.



Для их разделения на группы предусмотрен довольно быстрый, но имеющий свои недостатки алгоритм. В любом случае, хуже он не сделает, а лучше потенциально может.

Вводится функция, считающая меру разрозненности трасс как среднее расстояние между всеми конечными точками трасс к средней длине трасс. То есть чем больше величина функции, тем дальше трассы располагаются друг от друга в отношении к их длине.

Если параметр больше, чем 1.2, то производится дальнейшая группировка. Но сложно что-то толковое придумать, поэтому вводятся два послабления.

Упрощение первое. Вместо конкретной трассы используется координата её центра.

Упрощение второе. Вместо достоверного подбора групп по разрозненности используется принцип исключения из минимального остовного дерева самых длинных рёбер.

То есть сначала мы получаем из трасс граф с вершинами – центрами отрезков. Этот граф полный. Далее по алгоритму Крускала для него ищется минимальное остовное дерево. Потом из него последовательно извлекают по одному самому длинному ребру для получения нескольких подграфов. Для каждого подграфа считается мера разрозненности. И такие исключения производятся несколько раз (для ясности порядка – около 10 раз).

Потом из всего этого выбирается наиболее удачный вариант, который удачно уменьшает меру разрозненности и при этом не разбивает граф на слишком большое число подграфов.

## Исключение лишних кораблей

Из списка кораблей убираем те, которые достоверно не лучше остальных:

* Убираются корабли-дубли с полностью одинаковыми характеристиками относительно уже отобранных кораблей
* Убираются корабли, которые точно хуже какого-либо другого (например, убирается шутер, если его скорость ниже, чем у другого, а цена при этом не меньше; с раскладчиками сложнее – исключения могут происходить только при одинаковом числе датчиков)
* Убираются обработчики, у которых число датчиков меньше, чем необходимо для обработки одной трассы с самым большим числом датчиков

## Основная функция построения

В качестве основного подхода был выбран метод имитации отжига. Ведь чтобы прийти к какому-то выгодному решению, часто нужно сделать сразу несколько перестановок в входных данных целевой функции.

В ходе работы алгоритма изменяется массив с номерами и направлениями трасс для укладчика. Шутер же каждый раз получает урезанную копию маршрута укладчика – все номера трасс встречаются по одному разу, а не по два.

Для изменения реализовано несколько функций:

* Множественное разворачивание(1) – берётся два случайных индекса и по ним делается зеркальное отображение (std::reverse) для участка массива, при этом для этого участка также меняется направление обхода. Чем меньше температура, тем ближе два случайных индекса друг к другу.
* Единичная перестановка(2) – берётся два случайных индекса (чем ниже температура, тем они ближе) и в массиве по ним меняются номера трасс с направлением обхода.
* Единичная смена направления(3) – берётся случайный индекс и по нему в массиве меняется направление обхода трассы.

Согласно принципу имитации отжига, описанные выше функции могут применить с некоторым шансом (зависящим от температуры и времени по сравнению к предыдущему решению) даже невыгодные изменения.

Отметим, что реализованные алгоритмы довольно несовершенны и зачастую не могут воспользоваться преимуществом большого числа датчиков на судне (изначально дана последовательность, что укладчик сразу собирает разложенные датчики и уже к этой последовательности применяются случайные изменения). Поэтому нет смысла считать большое число датчиков за преимущество.

Отметим также, что в ходе работы алгоритма имитации отжига мы периодически возвращаемся к самому оптимальному решению из найденных (но не слишком часто). Это сделано для того, чтобы слишком сильно не заходить в невыгодные состояния.

## Поэтапно

1. Создаётся исходная последовательность номеров трасс передвижения для укладчика: 0,0,1,1,2,2,3,3… при этом направление обхода каждой следующей трассы – случайно. Далее полным перебором по всем кораблям (которые остались после исключения лишних кораблей, см. выше) определяется самая дешёвая пара по входным условиям.
2. Если трассы разрознены (ранее введённый коэффициент больше 1.2), то будут следующие действия:
   1. Вычисляется лучшая группировка трасс, что повысит локальную скученность для каждой подгруппы.
   2. Нужно определить, в каком порядке следует отсортировать полученные группы трасс из этой лучшей группировки. Для этого используются перестановки (std::next\_permutation) для массива с индексами групп и для каждой перестановки высчитывается время. Берётся лучший по времени результат. (Всего делается не более 100к перестановок – явно введённое ограничение).
   3. Происходит отработка алгоритма имитации отжига с единичными перестановками(2) и единичной сменой направления(1), причем единичные перестановки происходят в пределах одной группы. Это долгая операция, занимает ориентировочно треть всего времени вычисления.
   4. На всякий случай ещё раз перебираем по всем комбинациям кораблей, вдруг с новой расстановкой трасс что-то изменилось. Это быстрая операция.
3. Происходит отработка алгоритма имитации отжига с множественными перестановками(3). Это долгая операция, занимает ориентировочно треть всего времени вычисления.
4. Происходит отработка алгоритма имитации отжига с перестановками(2) и единичной сменой направления(1). Это долгая операция, занимает ориентировочно треть всего времени вычисления.

По данному алгоритму создаётся несколько потоков (равно числу логических ядер процессора), где каждый проделывает всю работу полностью (можно было бы и оставить начальные пункты только одному потоку, но основное время вычислений идёт на имитацию отжига, поэтому некритично).

В конце среди всех конечных результатов (каждый поток даёт один ответ) выбирается лучший. По нему в графический интерфейс выдаётся маршрут и стоимость. В папке с приложением в файле out.txt создаётся маршрут (вместе с исходными данными других блоков).

# Анализ полученного алгоритма

Так как мы находимся в условиях недружелюбных плюсов, строить графики затруднительно.

Для определённости будем анализировать работу программы для файла Ship1b.txt из Этапа 2, из которого исключим готовое решение (тогда создастся наше).

Изначальное решение, которое нам предлагается, имеет следующие характеристики:

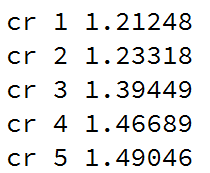
-- стоимость аренды по маршруту из исходных данных равна 3.460.000 [101 ч / 5 дн. @ H: BSHIP1, S: ASHIP1 ]

Если запустить его в нашей программе, то видно, что над этим решением не предпринималось никаких оптимизаций.

Наша программа же после оптимизации выдаёт следующий результат:

-- стоимость аренды по созданному маршруту равна 1.185.000 [72 ч / 3 дн. @ H: BSHIP2, S: ASHIP1 ]

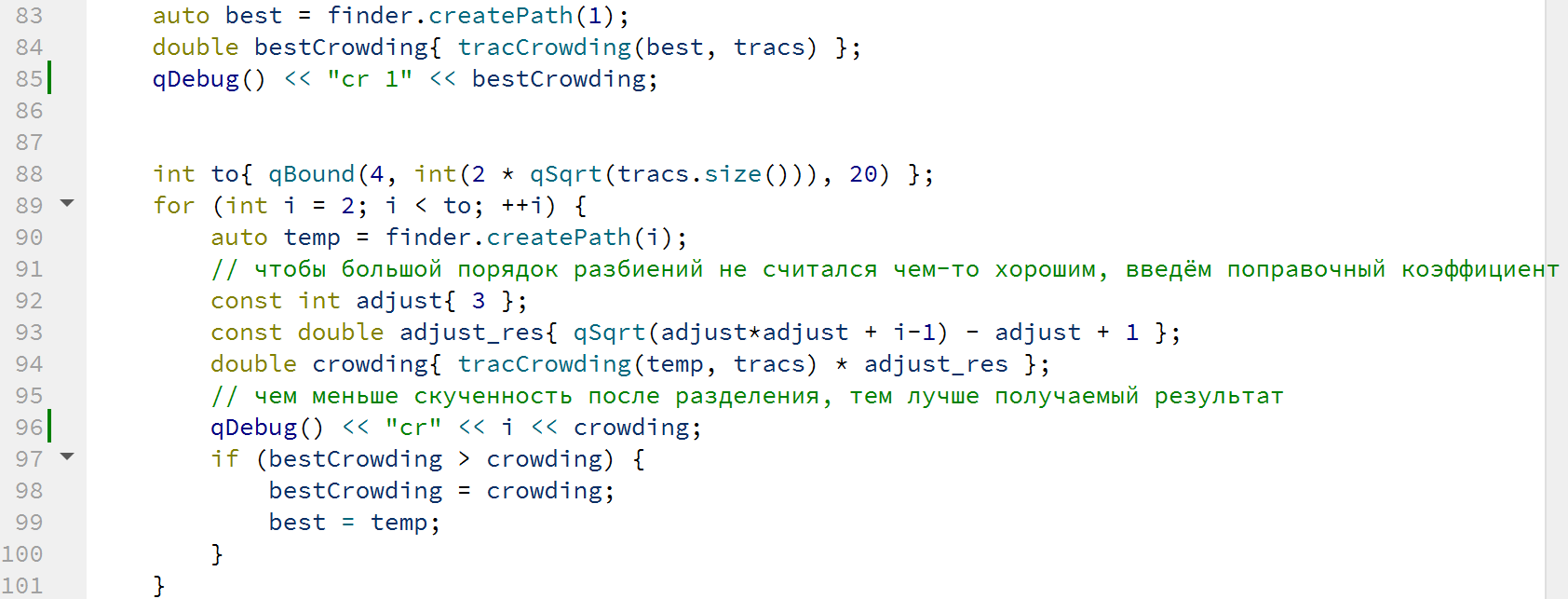
Для обозначенных трасс мера разрозненности составляет 1.21248. То есть мы пытаемся найти выгодную группу. И разбиения выдают такую меру разрозненности:



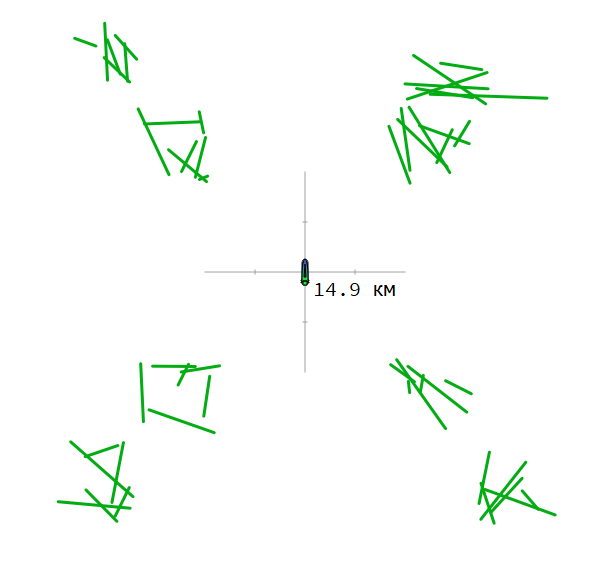
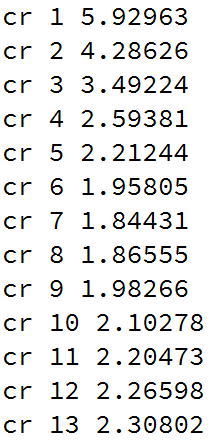
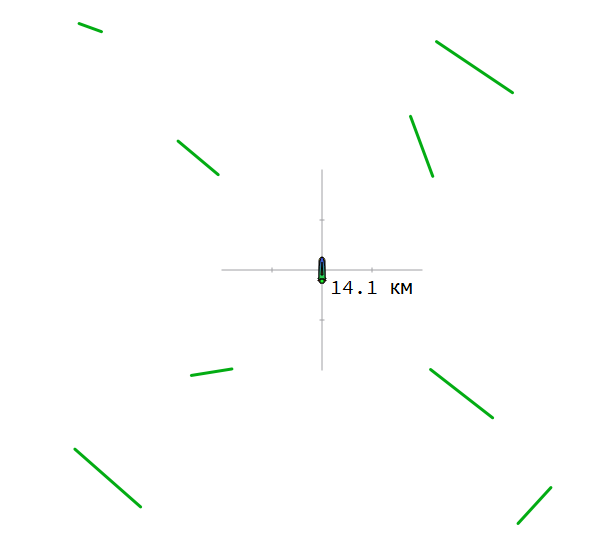
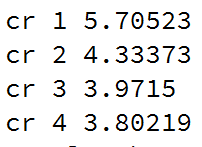
Где после cr идёт количество подграфов. Так как мы ввели поправочный коэффициент при увеличении количества разбиений, то в итоге останавливаемся на том варианте, что разбиений не будет. (После разбиений разрозненность убывает, чем она ниже, тем лучше) По расположению трасс видно, что на группы по скученности и не разбить ничего:



Для справки, вот так вычисляются эти cr:



Сразу приведу пример, где скученность явно выражена:

  или вот  

На подобных примерах польза данного метода очевидна. Впрочем, эти примеры довольно искусственны.

Теперь кратко выведу расчётное время после каждого основного этапа работы алгоритма:



Пояснение:

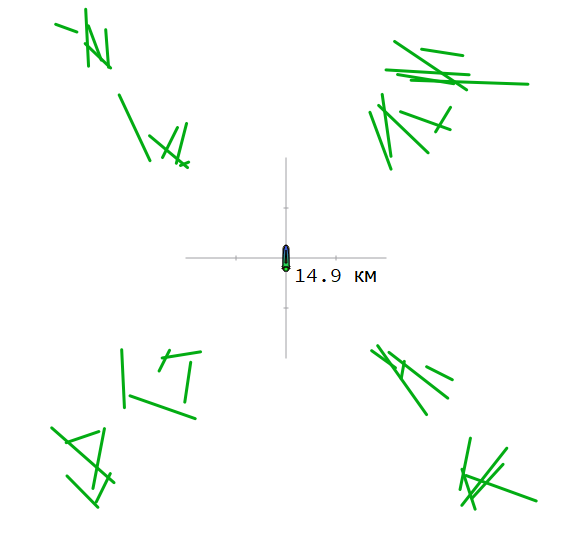
* первая цифра обозначает время без оптимизаций (0,0,1,1,2,2,3,3…) [результат этапа 1]
* вторая – время после группировки трасс и оптимального размещения этих групп (если группировки не было, выводится ноль) [результат этапа 2.2]
* третья – после работы алгоритма внутри групп [результат этапа 2.3]
* четвёртая – после множественных перестановок(3) [результат этапа 3]
* пятая – после единичных перестановок(2) и …(1) [результат этапа 4]

В качестве эксперимента уберём из алгоритма этап 3:



Более точных характеристик для анализа с графиками сделать не получится – выделенный бюджет на исходе. Будем работать с тем, что есть.

Приведём ещё несколько примеров конфигураций трасс и получаемых данных (с третьим этапом и без него, с включённой возможностью группировки и без неё):

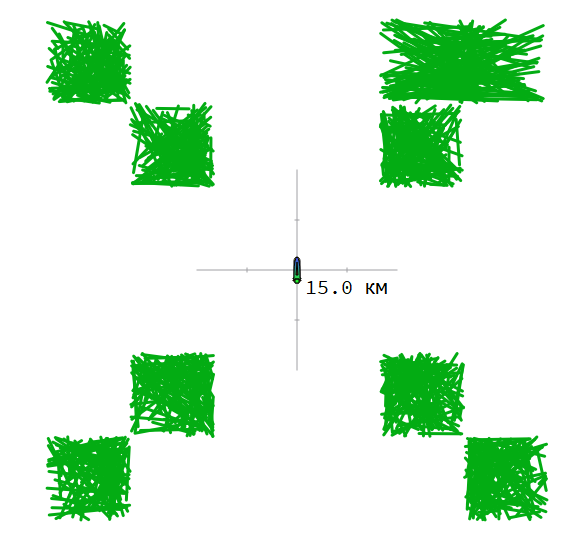
40 трасс







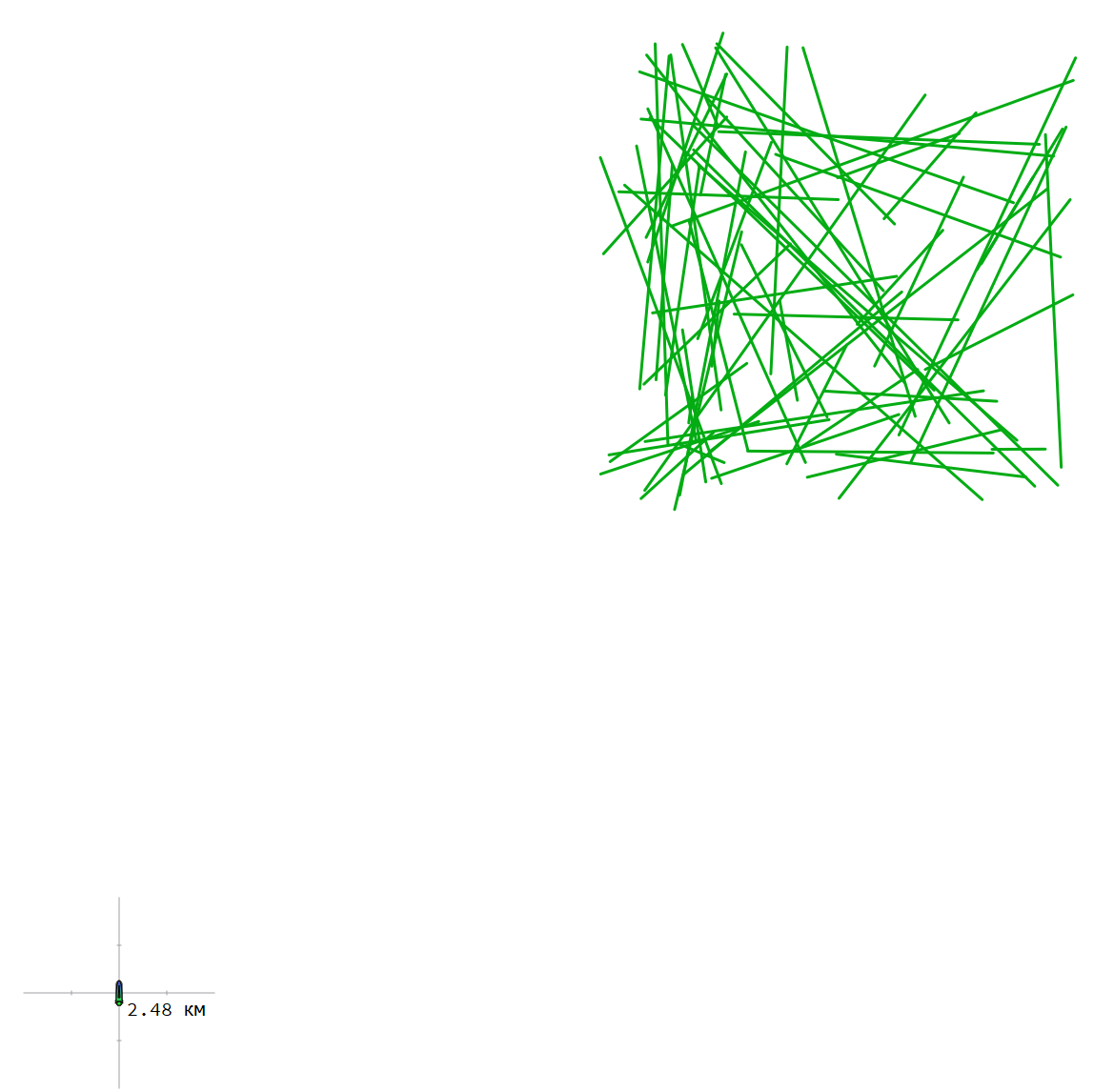
(по последнему результату видно, что без этой группировки (группировка была напрямую выключена) основной алгоритм даже для 40 трасс уже ничего особо не может сделать, но весь бюджет уже распилен, ничего не поделаешь…)

1000 трасс







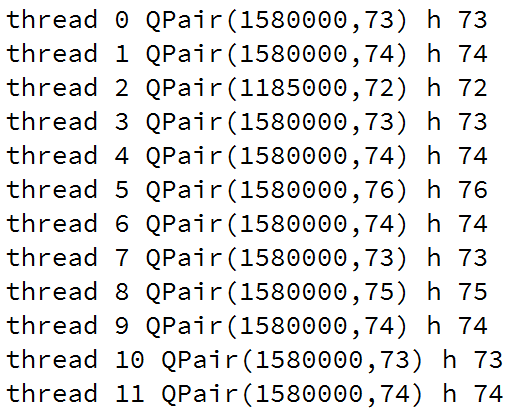
70 трасс



## Многопоточное вычисление

В подобного плана задаче было бы сложно подобрать оптимальную схему распараллеливания всего процесса вычисления. Ведь здесь в основном совершаются мелкие и быстрые операции над изменяемыми данными, постоянная синхронизация убила бы всю параллельность.

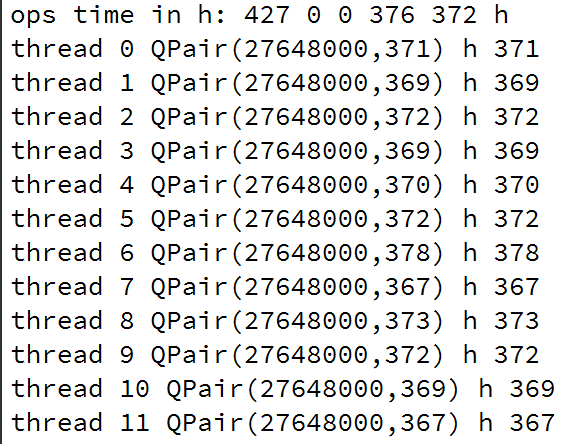
Потенциально можно было бы разделить данные, выполнить параллельно алгоритм случайного поиска на основе имитации отжига в размере десятков тысяч операций, а потом объединять потоки, взяв лучший из результатов за новую отправную точку и так итеративно идти, но на текущий проект и так было потрачено слишком много усилий при низком бюджете. Поэтому остановились на более простом варианте – каждый поток проделывает всю работу от начала до конца, а так как алгоритмы работают на основе случайных перемещений, то дают разные результаты (в каждый поток устанавливается своё семя для рандомайзера, а функция получения случайного числа для каждого потока своя – гонки не возникает). В результате вычислений потоки дают разные ответы:

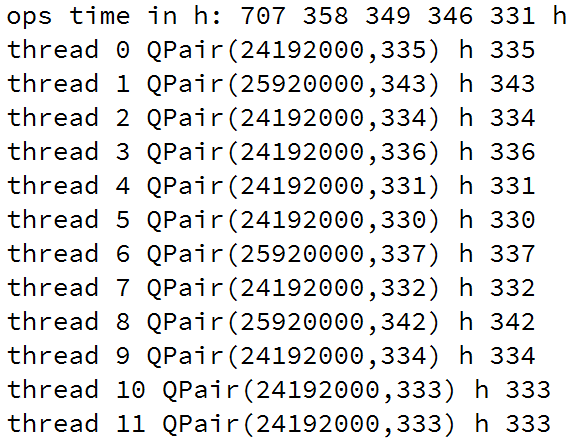


Пояснение к рисунку: каждая строчка описывает результат вычислений в конкретном потоке. QPair содержит в себе общую цену и количество часов (это для того, чтобы при одинаковой цене выводился результат с минимальным числом часов). По рисунку видно, что тестовая машина содержит 12 логических ядер (AMD Ryzen 5 5600H). Да, это тот ноутбук, который пришлось купить, чтобы делать лабораторные по методам оптимизации.

Также видно, что весь процессор нагружается не зря – мы смогли найти более оптимальный вариант, что занимает по времени 3 дня, а не 4.

Ещё примеры многопоточных вычислений:





В целом не такая большая разница, но она всё же есть.