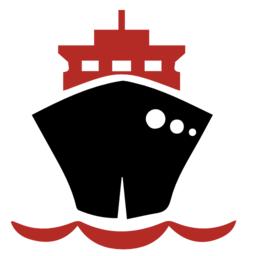
# StrawberryShip



# Методы оптимизации

Задача поиска маршрута с близкой к минимальной цене

Подготовили студенты группы ИВТ-11М

Григорьев А.В.

Зиновьев Н.А.

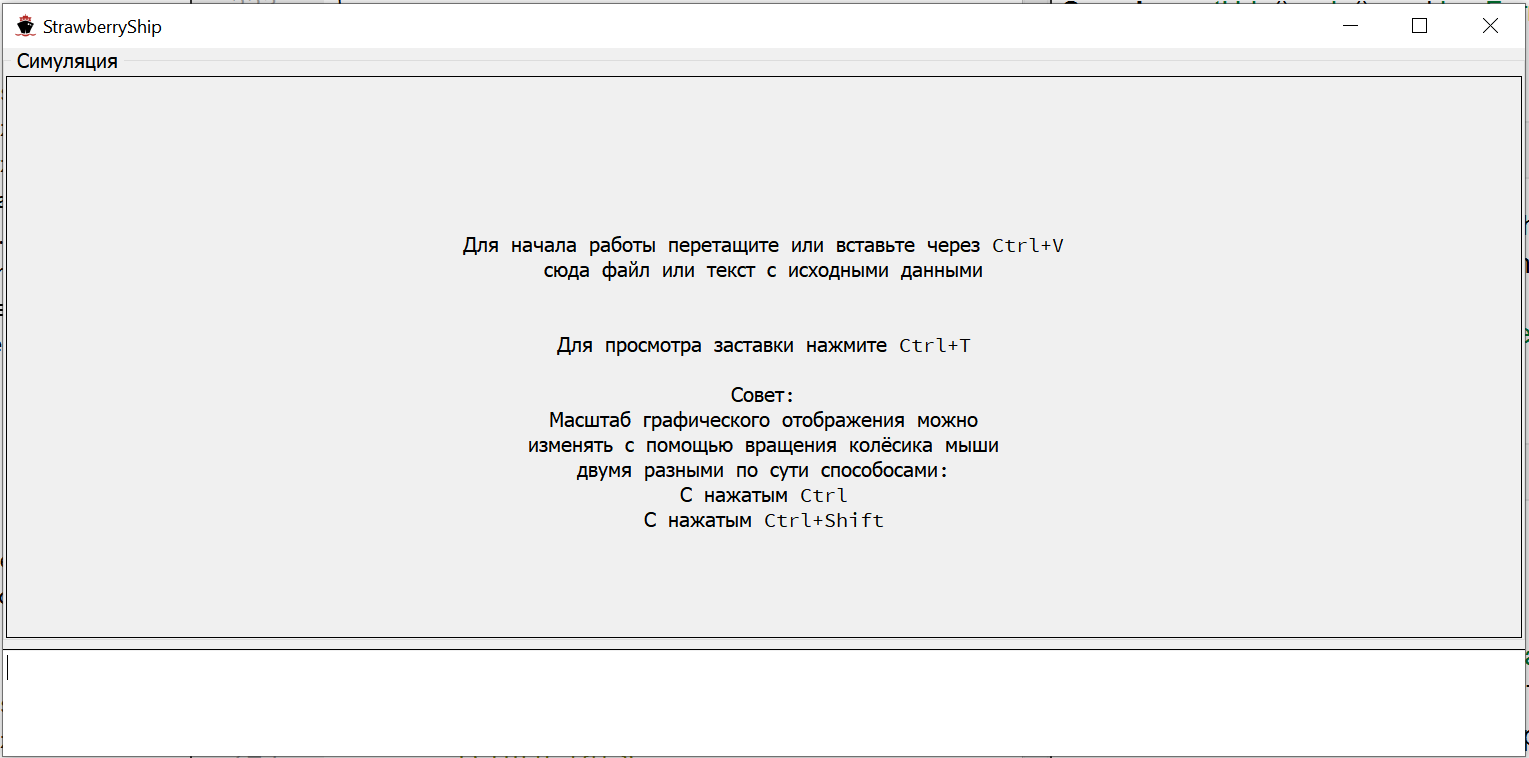
Королёва В.А.

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:

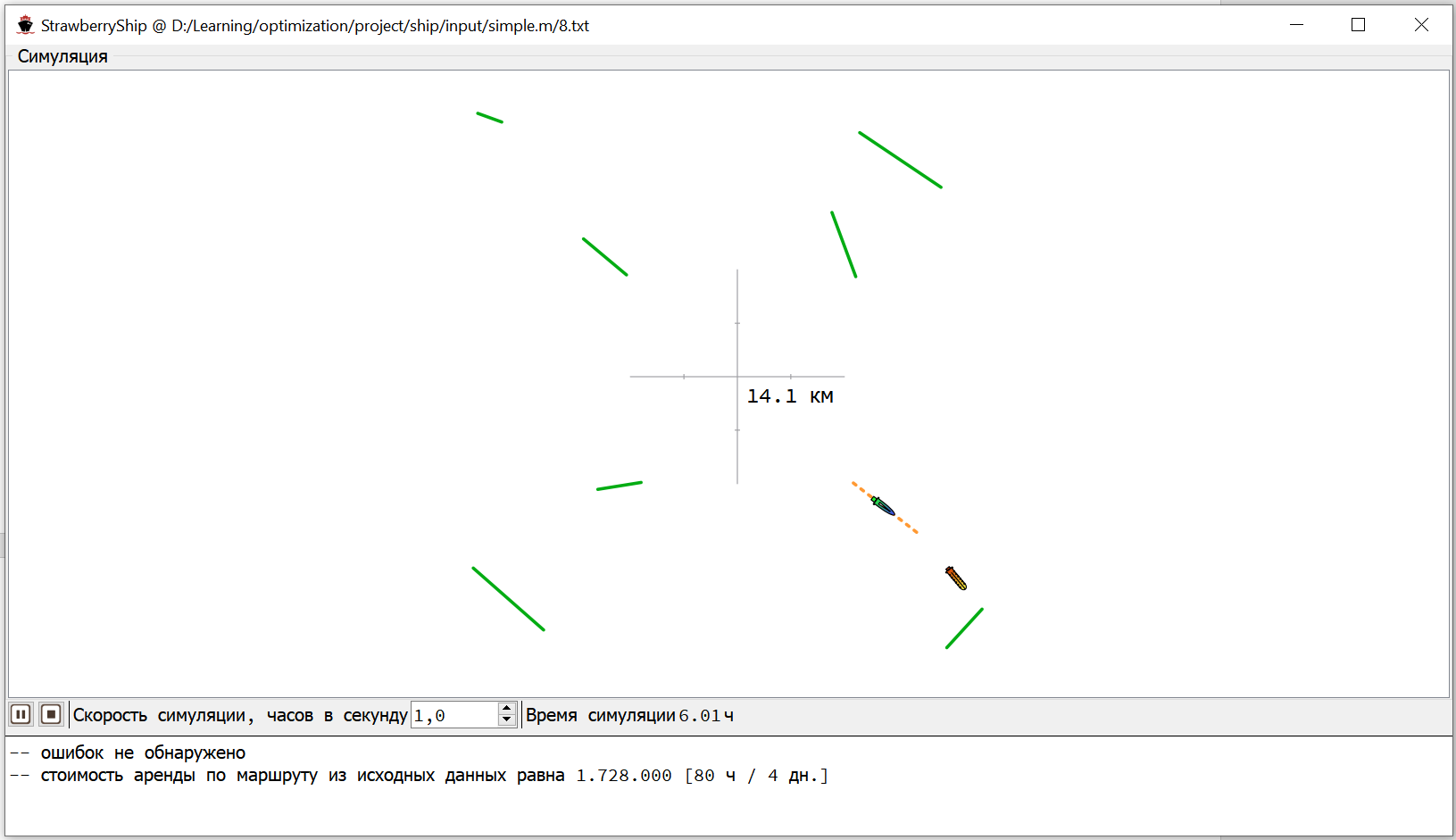
* Если происходит активное действие (не простое перемещение и простой), то это действие должно происходить по координатам, которые образуют реальную трассу. Иначе ошибка.

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.

# 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.

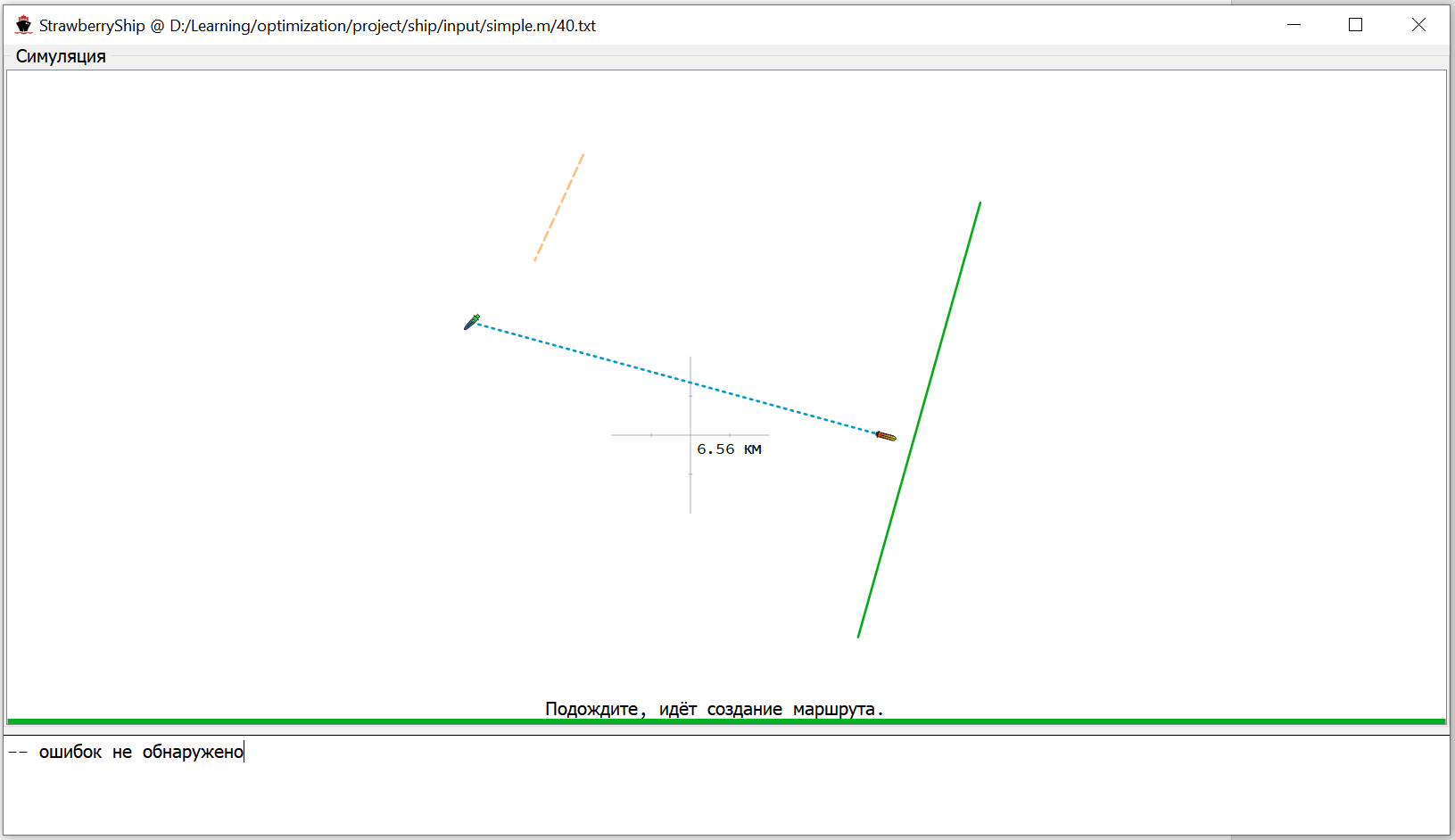
Реализована за счёт двух объектов: корабля и трассы.

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.

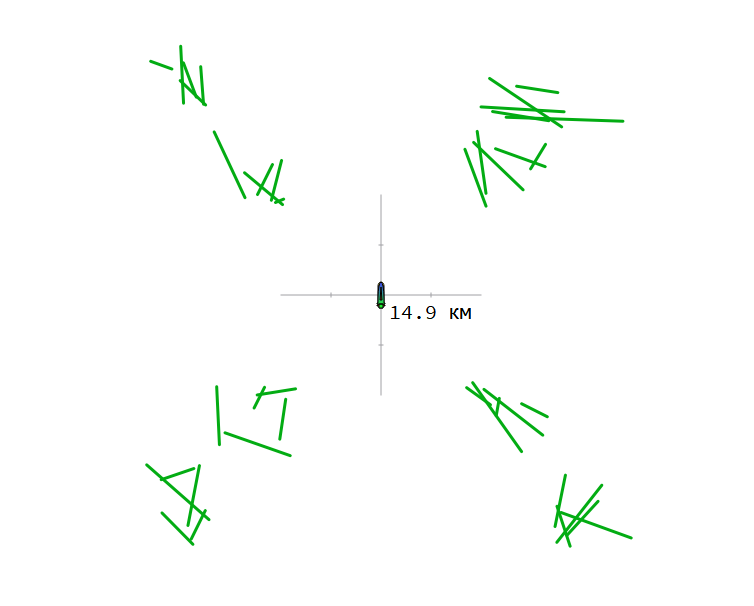
Если в процессе вычисления у обработчика окажется отрицательный баланс, то функция тоже сообщит о некорректных данных. (просто вернёт -1 час, что легко проверяется извне).

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).