Содержание

Введение

Существует множество различных алгоритмических задач, которые на первый взгляд выглядят не очень практически применимо, однако имеют под собой серьёзное обоснование и сложное решение. Одной из таких задач является задача о *максимальном пустом прямоугольнике*(МПП). Её суть состоит в следующем: требуется найти прямоугольник максимального размера, который следует разместить среди препятствий на плоскости. Задачи этого типа могут возникать при автоматизации проэктирования электроники, в разработке и компоновке интегральных микросхем. Более частным случаем задачи о МПП является другая: допустим, что дан прямоугольник A, содержащий в себе n точек, нужно найти прямоугольник наибольшей площади, стороны которого параллельны прямоугольнику A, лежащий в прямоугольнике A и не содержащий какую-либо из данных точек. Именно один из частных случаев этой задачи и будет рассматриваться в данной работе.

1 Постановка задачи курсовой работы

Разработка модульной программы на языке C++ для решения следующей задачи. Имеется прямоугольное поле — это участок земли, на котором располагаются прямоугольные здания, параллельные сторонам внешнего участка. Кроме того, на этом участке посажены деревья. Требуется выбрать прямоугольный участок максимальной площади для здания так, чтобы он был построен не ближе чем на метр ко всем препятствиям, а его стороны также располагались параллельно сторонам участка.

Введём несколько понятий для данной задачи:

A — участок земли, представленный левой нижней точкой A_{lb} и правой верхней точкой A_{rt}, каждая точка описывается своими координатами (X_i, Y_i).

T — множество n деревьев, каждое дерево — $T_i, i = 1, 2, ...n$, также представляет из себя точку, определённую своими координатами.

B — множество всех k зданий, каждое здание B_i описывается парой координат нижней левой и правой верхней точек $B_{i_{lb}}, B_{i_{rt}}, i = 1, 2, ...k$, и каждая из которых в свою очередь также определяется своими координатами.

Формат входных данных:

Координаты являются целыми числами, один метр равен евклидовому расстоянию между точками $(0, 0)$ и $(0, 1)$ или же между $(0, 0)$ и $(1, 0)$. В первой строке входных данных записано 4 числа — X и Y координаты точек A_{lb} и A_{rt} по очереди, сперва координаты первой, затем второй.

Во второй строке записано число n, равное количеству деревьев на участке. В следующих n строках идут пары координат X и Y деревьев.

В строке после этого записано число k, которое определяет количество зданий на участке. И в дальнейших k строках записаны по две пары чисел - X и Y координаты левой нижней и правой верхней точек по очереди, сперва координаты первой, затем второй.

Поток ввода — стандартный ввод или input.txt.

Формат вывода данных:

В строке записаны две пары чисел — X и Y координаты левой нижней и правой верхней точек искомого прямоугольника по очереди, сперва координаты первой, за-

тем второй.

Поток вывода — стандартный вывод или output.txt.

Кроме того, в результате выполнения программы будет производиться открытие нового окна с графической визуализаций и представлением участка земли, как одноцветного многоугольника, деревьев - в виде точек, зданий - в виде контуров прямоугольников, а искомый МПП будет представлять из себя закрашенный многоугольник, но уже другого цвета. [1]

[1] Будет добавлено дополнение по поводу графической визуализации, возможна сетка

2 Описание решения задачи

2.1 Переход к задаче о МПП

Для того, чтобы из данной задачи получить более общую, воспользуемся следующим приёмом. Множество всех точек, которое можно получить из деревьев и зданий, мы переведём в новое множество точек, используя условие того, что наш многоугольник должен располагаться на расстоянии метра от данных препятствий. Для этого построим функцию $f(A, B, T, 1) : \{B, T\} \to \{S\}$. Таким образом, данную задачу мы свели к задаче о МПП, а следовательно, если решим её, то решим и исходную.

2.2 Решение задачи о МПП

Простейшим решением, которое лежит на поверхности, является полноперебор-ный алгоритм. Сперва сформируем множество всех точек, из которых мы можем начать построение нашего многоугольника. Затем, будем по очереди увеличивать наш многоугольник — сперва по диагонали, увеличивая как ширину, так и длину, на каждом шаге проверяя, не входит ли в территорию нашего многоугольника какая-либо точка, и если такая есть, то возвращаемся на шаг назад. Далее, мы будем делать схожее действие, однако увеличивая нашу фигуру сперва в длину, запоминая максимальную площадь и координаты, на которых прервётся данный цикл, если полученная площадь больше, чем сохранённая ранее, а затем в ширину, также запоминая оптимальные параметры. После перебора всех возможных пустых прямоугольников, получим искомое решение.

Несмотря на свою простоту, данное решение не может похвастаться скоростью работы, очевидно, что этот алгоритм крайне неэффективен даже для переборных алгоритмов. Таким образом, следует изменить подход к данной задаче.

Назовём данный *прямоугольник P ограниченным* (далее ОП), если для него выполняются условия:

- Прямоугольник P полностью содержится в прямоугольнике A.
- P не содержит в себе никаких точек.

- Каждая сторона прямоугольника P либо содержит в себе точку, либо прилегает к стороне прямоугольника A.

Всего можно выделить 4 типа ОП:

1. В которых противоположные стороны прилегают к сторонам прямоугольника A.
2. В которых две смежные стороны прилегают к сторонамм прямоугольника A.
3. В которых только одна сторона прилегает к стороне прямоугольника A.
4. В которых каждая сторона содержит в себе как минимум одну точку.

Сперва, получив максимальное расстояние $MGAP$ между x-овыми координатами всех точек, перемножим его на расстояние между y-ковыми координатами верхней и нижней сторон прямоугольника , сохранив это произведение как $MAXR$, а также сохранив соответствующие координаты полученного прямоугольника. Таким образом, данным шагом мы отобрали наибольший МПП среди тех ОП, которые представляют собой тип №1, и в которых сторонами касания являются верхняя и нижняя.
Затем, мы отсортируем все точки по y-ковым координатам в порядке невозрастания, и будем попарно перебирать каждую пару точек единожды (т.е не учитывая порядок точек в паре). При новом выборе каждой точки мы будем сохранять границы текущего поиска по x как T_L и T_R. Если вторая точка из пары принадлежит данному интервалу, тогда мы посчитаем максимальную площадь прямоугольника, у которого y-ковые координаты будут представлять из себя соответствующие координаты выбранных двух точек, а в качестве x-овых будут идти T_L и T_R. Если полученная площадь больше, чем найденная ранее $MAXR$, сохраним её и искомые координаты. Если вторая точка из выбранной пары больше, чем первая, то сохраним её как левую границу T_L, иначе как правую T_R. Этими действиями мы определили остальные ОП 1-го типа — стороны которых прилегают в к левым и правым сторонам прямоугольника A, ОП типа №2 и №4, а также типа №3, чьи левые или правые стороны прилегают к соответствующим сторонами A.
Кроме того, на каждом шаге этого перебора, для каждой точки из первой пары мы

посчитаем площадь, беря её y-ковую координату и y-ковую координату нижней стороны A, а в качестве x-овых координат будем брать T_L и T_R, также сохраняя $MAXR$ и координаты в случае необходимости. Этим мы перебрали ОП типа №3, чья нижняя сторона прилегает к нижней стороне прямоугольника A.

Нам осталось посчитать только площадь тех ОП типа №3, чья верхняя сторона прилегает к верхней стороне прямоугольника A. Для этого мы будем брать каждую точку, находить правую координату R, которую будем определять как минимальную среди всех x-овых координат точек в обьединении с координатой правой стороны A, таких, что её y-ковая координата больше выбранной точки, а x-овая координата также больше соответствующей координаты выбранной точки. Соответственно, левую координату L мы будем брать как максимальную координату среди всех точек, чья y-ковая координата больше координаты этой точки, а x-овая координата меньше этой точки. Посчитаем для прямоугольника с x-овыми координатами R и L, а y-ковыми координату верхней стороны прямоугольника A и y-ковую координату для выбранной точки. В очередной раз сохраняем координаты и $MAXR$, если оно получилось максимальным.

Теперь мы точно можем сказать, что задача о МПП может быть решена с помощью алгоритма, чья временная сложность оценивается как $O(n^2)$, где n — число точек в прямоугольнике.

3 Структуры данных

Для решения задачи воспользуемся следующими структурами данных:

- Point — данная структура представляет из себя простейшую СД, на основе которой будут строиться все дальнейшие СД. Она состоит из двух чисел, которые характеризуют x и y координаты точки.

```
struct point {
  int x;
  int y;
};
```

- Rectangle — эта СД служит представлением для прямоугольника. В неё хранятся левая нижняя и правая верхня вершины прямоугольника.

```
struct rectangle {
  point lb;
  point rt;
};
```

- Следущая структура, input_data, представляет из себя набор входных данных для данной задачи. То есть, в ней содержится представление участка земли, в виде прямоугольника, массива деревьев, как массива точек, и массива зданий как массива прямоугольников.

```
struct  input_data {
  rectangle A;
  std::vector<point> T;
  std::vector<rectangle> B;
};
```

- Последняя СД, которая необходима в данной задаче — входные данные для задачи о МПП, MERdata. В данной структуре хранится внешняя плоскость в

форме прямоугольника, а также массив всех точек.

```
struct MERdata {
  rectangle A;
  std::vector<point> S;
};
```

4 Блок-схемы/Функции/Методы/Спецификации(Думаю, как назвать)

Reader

- Заголовок: Reader(std::istream& input_stream);
  Назначение: конструктор, инициализация объекта класса Reader, запись потока ввода std::istream& input_stream в атрибут класса std::istream& is.

- Заголовок: input_data read();
  Назначение: частный метод, возвращает введёную информацию типа input_data из потока ввода, сохранённого в атрибуте класса std::istream& is.

- Заголовок: input_data read_input_data();
  Назначение: публичный метод, интерфейс для класса, возвращает введёную информацию типа input_data из потока ввода.

Writer

- Заголовок: Writer(std::ostream& output_stream);
  Назначение: конструктор, инициализация объекта класса Writer, запись потока вывода std::ostream& outputs_stream в атрибут класса std::ostream& os.

- Заголовок: void write(rectangle MER);
  Назначение: частный метод, записывает в поток вывода, сохранённый в атрибуте класса std::ostream& os, координаты прямоугольника rectangle MER.

- Заголовок: void write_output_data(rectangle MER);
  Назначение: публичный метод, записывает в поток вывода координаты прямоугольника rectangle MER.

Converter

- Заголовок: Converter();
  Назначение: конструктор, инициализация объекта класса Converter

- Заголовок: MERdata convert(input_data d, int dist);
  Назначение: частный метод, возвращает информацию MERdata для задачи о МПП, преобразовывая в неё информацию input_data о задаче поиска прямоугольника максимальной площади отдалённого от всхе обьектов на расстояние dist.

- Заголовок: MERdata get_converted_data(input_data d, int dist);
  Назначение: публичный метод, интерфейс для класса, возвращает информацию MERdata для задачи о МПП, преобразовывая в неё информацию input_data о задаче поиска прямоугольника максимальной площади отдалённого от всхе обьектов на расстояние dist.

MERSolver

- Заголовок: MERSolver();
  Назначение: конструктор, инициализация объекта класса MERSolver

- Заголовок: void MGAP_x(std::vector<point> arr, int* x_left_best, int* x_right_best)
  Назначение: частный метод, записывает по адрес x_left_best и x_right_best координаты по x, между которыми наибольшее расстояние по этой же координате, и нет таких точек, что их x-овая координата будет лежать между ними.

- Заголовок: void sort_by_coord(std::vector<point>& arr, bool x);
  Назначение: частный метод, сортирует массив точек std::vector<point> arr по координате x, если bool x является истинной, иначе сортирует по координате y.

- Заголовок: rectangle solve(MERdata md);

  Назначение: частный метод, возвращает координаты МПП rectangle, для данных MERdata md.

- Заголовок: rectangle get_solution(MERdata md);

  Назначение: публичный метод, интерфейс класса, возвращает координаты МПП rectangle, для данных MERdata md.

Render

- Заголовок: Render(rectangle l_button, rectangle r_button);;

  Назначение: конструктор, инициализация объекта класса Render(), запись в атрибуты класса rectangle l_b и r_b координаты левой и правой кнопок соответственно структурами l_button и r_button.

- Заголовок: void init();

  Назначение: частный метод, инициализирует атрибуты класса sf::Font font фоном из файла "font.ttf"и sf::Window window;

- Заголовок: void start();

  Назначение: публичный метод, отрисовывает кнопки для выбора потока ввода в открытом окне атрибуте sf::Window window.

- Заголовок: void end();

  Назначение: публичный метод, отрисовывает кнопки для выбора потока вывод в открытом окне атрибуте sf::Window windowa.

- Заголовок: void close();

  Назначение: публичный метод, закрывает окно в атрибуте sf::Window window.

- Заголовок: bool is_window_open();

  Назначение: публичный метод, возвращает значение истина, если окно атрибут sf::Window window открыто, иначе возвращает значение ложь.

- Заголовок: bool poll_event(sf::Event& e);

  Назначение: публичный метод, возвращает значение истина, если из стека для

окна атрибута sf::Window window было записано событие в sf::Evenet& e, иначе возвращает значение ложь.

- Заголовок: sf::Vector2i get_mouse_position();
  Назначение: публичный метод, возвращает координаты sf::Vector2i текущей позиции компьютерной мыши.

- Заголовок: void solution(input_data d, rectangle solution);
  Назначение: публичный метод, отрисовывает решение задачи с инфомрацией input_data d о поиске прямоугольника с максимальной площадью с координатами rectangle solution в открытом окне атрибуте sf::Window window.

Controller

- Заголовок: Controller();
  Назначение: конструктор, инициализация объекта класса Controller.

- Заголовок: void run();
  Назначение: публичный метод, считавает данные из выбранного потока ввода, решает задачу о поиске прямоугольника с максимальной площадью, удалённого от всех объектов — деревьев, зданий, на 1 метр, записывает решение в выбранный поток вывода, визуализирует решение графически в окне.

5 Разбиение на модули

Данную задачу можно разбить на несколько классов:

1. Reader — класс, отвечающий за получение входных данных.
2. Writer — класс, отвечающий за вывод выходных данных.
3. Класс Converter предназначен для преобразования данных исходной задачи в данные задачи о МПП.
4. MERSolver представляет собой класс, который получает данные и решает задачу о МПП, в результате чего формируются выходные данные.
5. Render — класс, отвечающий за графическую визуализацию.
6. И последний класс, Controller, является управляющим классом. Он взаимодействует с пользователем, управляя классом для визуализации Render и получая режим ввода входных данных и режима вывода выходных, получает входные данные от класса Reader, передаёт их в Converter. Новые, преобразованные данные, он даёт классу MERSolver, полученный результат же Controller отдаёт в класс Writer, а затем заставляет класс Render отобразить исходную задачу в окне приложения.

Список литературы

[1] A. Naamad, D.T. Lee and W.L. Hau, On the maximum empty rectangle problem, Discrete Applied Mathematics 8, 267-277, (1984).

[2] S.C. Nandy and B.B. Bhattachary, Maximal Empty Cuboids Among Points and Blocks, Computers Math. Applic. Vol. 36, No. 3, pp. 11-20, 1998.

[3] S.C. Nandy and B.B. Bhattachary, On finding an empty staircase polygon of largest area (width) in a planar point-set, Computational Geometry 26 (2003) 143–171.

[4] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 3-е издание = Introduction to Algorithms, Third Edition. — М.: «Вильямс», 2013. — 1328 с.