# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Лабораторная работа №8 по курсу «Дискретный анализ»**


Студент: А. Ю. Голов
Преподаватель: С. А. Михайлова
Группа: М8О-301Б-21

Дата:
Оценка:
Подпись:


**Москва, 2023**

# 1 Поставленная задача

**Формулировка задания**: На координатной прямой даны несколько отрезков с координатами $[L_i, R_i]$. Необходимо выбрать минимальное количество отрезков, которые бы полностью покрыли интервал $[0, M]$.

**Формат ввода**: На первой строчке располагается число $N$, за которым следует $N$ строк на каждой из которой находится пара чисел $L_i$, $R_i$ ; последняя строка содержит в себе число $M$.

**Формат вывода**: На первой строке число $K$ выбранных отрезков, за которым следует $K$ строк, содержащих в себе выбранные отрезки в том же порядке, в котом они встретились во входных данных. Если покрыть интервал невозможно, нужно распечатать число 0.

# 2 Описание

Жадные алгоритмы предназначены для решения задач оптимизации. Они обычно представляют собой последовательность шагов, на каждом из которых предоставляется некоторое множестов выборов. В жадном алгоритме всегда делается выбор, который кажется самым лучшим в данный момент, т.е. проводится локально оптимальный выбор в надежде, что он приведет к оптимальному решению глобальой задачи. Важно отметить, что жадные алгоритмы не всегда приводят к оптимальным решениям, но во многих задачах дают нужный результат.

Применим жадный алгортим к задаче выбора отрезков. Сохраним наши пары границ отрезков в массив данных и отсортируем их по правой границе, т.е. при обращении к данному массиву в начале будут лежать отрезки, которые заканчиваются правее всех. Также при сохранении отрезков запомним их изначальный индекс для того чтобы, при выводе восстановить исходный порядок.

Теперь будем считать, что в точке 0, т.е. в начале покрываемого интервала, лежит грань, котрорую мы будем двигать по мере того, как будет происходить покрытие интервала. Эта грань будет означать самую правую точку покрытой части (в начале это точка 0). Пока эта точка не станет $\geq M$, мы будем считать наш интервал непокрытым.

Затем будем в цикле проходиться по нашему отсортированному масиву и брать первый отрезок, левая граница которого лежит левее или равна текущей грани, а правая граница лежит обязательно правее текущей грани. Это необходимые условия, т.к. мы ищем минимальное число отрезков, иначе есть шанс взять отрезки, котрые не подвинут нашу грань. Если же на каком-то этапе прохода мы не нашли такого отрезка, это говорит о том, что мы не пожем покрыть наш интервал данной выборкой отрезков.

Подходящие же отрезки будем сохранять в новый массив, который в случае порытия интервала отсортируем по изначальному индексу и выведем в качестве ответа. Докажем верность данного алгоритма. Благодаря тому, что на каждом шаге мы берем отрезок с самой правой границей, то это будет гаратировать минимальность количества выбранных отрезков. Здесь важно отметить, что жадные алгоритмы да- ют нам лишь одно оптимальное решение, которых может быть несколько. Следо- вательно, допустим, что мы имеем оптимальное решение задачи выбора отрезков и на каком-то этапе мы решаем добавить наш отрезок в это решение. У нас есть два варианта:

- Этот отрезок лежит в этом решении, тогда все ок, просто перейдем к следующй подзадаче;
- Этот отрезок не лежит в этом решении, но т.к. согласно нашему алгоритму на дан- ном этапе мы добавляем отрезок с самой правой границей, то он сдвинет грань еще дальше и обязательно будет иметь пересечние со следующим отрезком в оптималь- ном решении, значит мы можем просто исключить отрезок

лежащий в оптимальном решении на выбраннный.

# 3 Исходный код

```
#include <bits/stdc++.h>

class TSegment {
  public:
  TSegment(int left, int right, int idx){
    this->left = left;
    this->right = right;
    this->idx = idx;
  }
  TSegment() = default;

  int GetLeft() const {
    return this->left;
  }

  int GetRight() const {
    return this->right;
  }

  int GetIdx() const {
    return this->idx;
  }

  void SetLeft(int value){
    this->left = value;
  }

  void SetRight(int value){
    this->right = value;
  }

  void SetIdx(int value){
    this->idx = value;
  }

  private:
  int left;
  int right;
  int idx;

  protected:

```

```
};

using psb = std::pair<std::vector<TSegment>, bool>;

bool operator<(TSegment const &seg1, TSegment const &seg2){
  return seg1.GetIdx() < seg2.GetIdx();
}

bool Select(std::vector<TSegment> &data, int m, std::vector<TSegment> &result)
{
  TSegment curSegment(-1, -1, -1);

  for (TSegment &segment : data) {
    if (segment.GetRight() <= result.back().GetRight()) {
      continue;
    }

    if (segment.GetLeft() <= result.back().GetRight())
    {
      if (curSegment.GetRight() < segment.GetRight())
      {
        curSegment = segment;

        if (m <= curSegment.GetRight())
        {
          result.push_back(curSegment);
          return true;
        }
      }
    }
    else
    {
      if (segment.GetLeft() <= curSegment.GetRight())
      {
        result.push_back(curSegment);
        curSegment = segment;

        if (m <= curSegment.GetRight())
        {
          result.push_back(curSegment);
          return true;
        }
      }
      else {
        return false;
      }
    }
  }

```

```
92    return false;
93  }
94
95  std::istream& operator >> (std::istream& in, std::vector<TSegment> &data) {
96    for (int i = 0; i < (int)data.size(); ++i)
97    {
98      data[i].SetIdx(i);
99
100     int left, right;
101     in >> left >> right;
102
103     data[i].SetLeft(left);
104     data[i].SetRight(right);
105   }
106
107   return in;
108 }
109
110 bool Comparator(const TSegment &a, const TSegment &b)
111 {
112   int aLeft = a.GetLeft();
113   int aRight = a.GetRight();
114   int aIdx = a.GetIdx();
115
116   int bLeft = b.GetLeft();
117   int bRight = b.GetRight();
118   int bIdx = b.GetIdx();
119
120   return std::tie(aLeft, aRight, aIdx) < std::tie(bLeft, bRight, bIdx);
121 }
122
123 int main()
124 {
125   int n, m;
126   std::cin >> n;
127
128   std::vector<TSegment> data(n);
129
130   std::cin >> data;
131   std::cin >> m;
132
133   std::sort(data.begin(), data.end(), Comparator);
134
135   std::vector<TSegment> resData = {{-1, 0, -1}};
136   bool found = Select(data, m, resData);
137
138   std::sort(resData.begin(), resData.end());
139
140   if (found) {
```

```
141     std::cout << resData.size() - 1 << '\n';
142
143     for (int i = 1; i < (int)resData.size(); ++i){
144       std::cout << resData[i].GetLeft() << ' ' << resData[i].GetRight() << '\n';
145     }
146   }
147   else {
148     std::cout << "0\n";
149   }
150
151   return 0;
152 }
```

# 4 Тест производительности

В качестве тестирующего инструмента были выбраны гугл-тесты, интегрированные в программу при помощи CMake. Применён метод «table-driven tests».

# 5 Выводы

По итогам выполнения восьмой лабораторной работы по курсу «Дискретный анализ», стало очевидно, что грамотный практический подход к задаче ведёт к оптимизации производительности программы посредством применения верного принципа разработки программы.

# Список литературы


[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008