Лабораторная работа №8 по курсу дискретного анализа: жадные алгоритмы

Выполнил студент группы М8О-308Б-20 МАИ *Зубко Дмитрий.*

Условие

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

Метод решения

Создаём вектор из n отрезков. Сортируем его по неубыванию левой границы отрезков. Используем две вспомогательные переменные: точку A, которую мы должны покрыть и точку B, до которой мы все покрыли. Проходим по отрезкам: если левая граница отрезка больше A, то добавляем в ответ отрезок, покрывающий B, приравниваем A к B. Если левая граница меньше или равна A и правая граница отрезка больше B, то запоминаем этот отрезок. В конце сортируем вектор по номеру входных данных.

Описание программы

В моей пр .

```
#include <iostream>
#include <algorithm>
#include <vector>


struct Segment {
    int l;
    int r;
    int idx;

    explicit Segment(int l=0, int r=0, int idx=0): l(l), r(r), idx(idx) {}
};


bool select_segments(std::vector<Segment>& segments, std::vector<Segment>& ans, int m) {
    int need_point = 0;
    int max_point = 0;
    Segment best_segment_to_cover_need_point;
    for (Segment &segment: segments) {
        if (segment.l > need_point) {
            ans.push_back(best_segment_to_cover_need_point);
            need_point = max_point;
            if (need_point >= m)
                break;
```

```
        }
        if (segment.l <= need_point and segment.r > max_point) {
            max_point = segment.r;
            best_segment_to_cover_need_point = segment;
        }
    }
    if (need_point < m) {
        ans.push_back(best_segment_to_cover_need_point);
        need_point = max_point;
    }
    return need_point >= m;
}


int main() {
    int n, m;
    std::cin >> n;
    std::vector<Segment> segments(n);
    std::vector<Segment> ans;
    for (int i = 0; i < n; ++i) {
        int l, r;
        std::cin >> l >> r;
        segments[i] = Segment(l, r, i);
    }
    std::cin >> m;
    std::sort(segments.begin(), segments.end(), [](Segment a, Segment b) {return a.l < b.l;});
    bool solution_exist = select_segments(segments, ans, m);

    if (solution_exist) {
        std::sort(ans.begin(), ans.end(), [](Segment a, Segment b) { return a.idx < b.idx; });
        std::cout << ans.size() << '\n';
        for (Segment &seg: ans) {
            std::cout << seg.l << ' ' << seg.r << '\n';
        }
    } else {
        std::cout << 0 << '\n';
    }
}
```

Дневник отладки

Забыл отсортировать массив по порядку во входных данных, не выводил количество отрезков в случае успешного покрытия. После исправления получил ОК.

Тест производительности

Померить время работы кода лабораторной и теста производительности на разных объёмах входных данных. Сравнить результаты. Проверить, что рост времени работы приувеличении объема входных данных согласуется с заявленной сложностью.

Время работы алгоритма для разных n:

1. n = 1000 => 0m0.008s
2. n = 10000 => 0m0.009s
3. n = 100000 => 0m0.015s
4. n = 1000000 => 0m0.283s
5. n = 10000000 => 0m0.443s
6. n = 100000000 => 0m6.345s

Можно сделать вывод, что моя асимптотическая оценка алгоритма абсолютно верна.

Недочёты

Программа работает только для корректных входных данных.

Выводы

При выполнении данной лабораторной работы я ознакомился с жадными алгоритмами и смог решить задачу покрытия отрезка отрезками. Сложность решения составила O(n * log(n)). Если бы входные отрезки были отсортированы по неубыванию левой границы, то сложность была бы O(n).