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: С. А. Красоткин
Преподаватель: Н. С. Капралов
Группа: М8О-208Б
Дата:
Оценка:
Подпись:

Москва, 2021

0.0 Лабораторная работа №8

Задача: Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Вариант задания: 2

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

Входные данные: На первой строке 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.

Выходные данные: 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.

0.1 Описание

Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным[2].

Задача похожа на поиск маскимального числа непересекающих отрезков на сегменте [1], но тут нам важен не правый, но левый край.

Мой жадник проверяет, что правый край последнего отрезка в векторе ответов меньше заданного M. Если условие выполняется, то прохожу за линию по тем отрезкам, левый конец, которого входит в покрытие, а правый наибольший. Добавляю его в вектор ответов.

Когда такой отрезок не находится, то ответ ноль.

После этого нужно вывести отсортированый вектор ответов.

Почему же жадник работает корректно. Пусть на каком шаге k непокрыто I чисел. Если S - минимальное покрытие, то верно $I + 1 \leq I - \frac{I}{S}$. Пусть есть другое покрытие M, тогда оно закроет $M * \frac{I}{M}$, но получаю противоречие, так как $M \leq S$.

Итоговая сложность складывается из поиска покрытия $O(n)$ и сортировки $O(n * \log n)$, которую вприниципе можно занизить до линии сортировкой подсчётом.

Оценка по памяти линейная, так как хранятся отрезки, идущие со входа.

0.2 Исходный код

Для хранения отрезков создаю структуру с полями концов, а также индексом порядка встречи во входном потоке. Конструктор по умолчанию вырождает отрезок в исходную точку с отрицательным индексом, так как такой отрезок не может быть задан.

За поиск оптимального покрытия отвечает функция *SeekSegments*(). Ответ хранится в векторе, в который сначала запихиваю нулевую точку для удобства реализации.

Когда ответ найден, его нужно отсортировать по индексам, для этого использую лямбда-компаратор.

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

const int ZERO = 0;

struct TSegment {
    int left;
    int right;
    int index;
    TSegment()
    {
        left = 0;
        right = 0;
        index = -1;
    }
};

void SeekSegments(std::vector<TSegment> &segments, int M) {
    std::vector<TSegment> answer;
    TSegment plug;
    answer.push_back(plug);

    while(answer.back().right < M) {
        int rMax = 0;
        int segInd = -1;
        for(int i = 0; i < segments.size(); i++) {
            if(segments[i].left <= answer.back().right &&
               segments[i].right > answer.back().right ) {
                if(segments[i].right > rMax) {
                    rMax = segments[i].right;
                    segInd = i;
                }
            }
        }
        if(segInd == -1) {
            std::cout<<"0\n";
            return;
        }
        else {
```

```
            answer.push_back(segments[segInd]);
        }
    }

    std::sort(answer.begin(), answer.end(), [](const TSegment &lhs, const TSegment &rhs) {
        return lhs.index < rhs.index;
    });
    std::cout<<answer.size()-1<<"\n";
    for(int i = 1; i < answer.size(); i++) {
        std::cout<<answer[i].left<<" "<<answer[i].right<<"\n";
    }
}

int main() {
    int N, M, L, R;

    std::cin>>N;
    std::vector<TSegment> segments(N);
    for(int i = 0; i < N; i++) {
        std::cin>>L>>R;
        segments[i].left= L;
        segments[i].right = R;
        segments[i].index = i;
    }
    std::cin>>M;

    SeekSegments(segments, M);

    return 0;
}
```

0.3 Консоль

```
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  cat ../../LR-s/8/Tests/zero.txt
3
-1 0
-5 -3
2 5
1
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  ./gSS < ../../LR-s/8/Tests/zero.txt
0
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  cat ../../LR-s/8/Tests/1.txt
6
-2 0
-2 1
-2 2
-2 3
-2 4
-1 6
4
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  ./gSS < ../../LR-s/8/Tests/1.txt
1
-1 6
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  cat ../../LR-s/8/Tests/2.txt
6
3 4
-2 1
-2 0
-2 3
-2 2
-2 3
4
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  ./gSS < ../../LR-s/8/Tests/2.txt
2
3 4
-2 3
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  cat ../../LR-s/8/Tests/3.txt
3
2 5
-5 -3
1 4
5
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  ./gSS < ../../LR-s/8/Tests/3.txt
0
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  cat ../../LR-s/8/Tests/4.txt
8
0 2
4 6
1 2
1 3
```

```
3 6
3 4
2 3
2 4
4
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  ./gSS < ../../LR-s/8/Tests/4.txt
2
0 2
2 4
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  cat ../../LR-s/8/Tests/5.txt
4
2 5
-5 -3
1 4
0 4
5
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  ./gSS < ../../LR-s/8/Tests/5.txt
2
2 5
0 4
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  cat ../../LR-s/8/Tests/6.txt
4
-1 0
-5 -3
2 5
0 1
1
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/8  ./gSS < ../../LR-s/8/Tests/6.txt
1
0 1
```

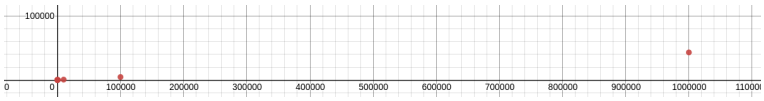
0.4 Дневник отладки

1. Первая проблема была на третьем тесте. Я выводил отрезки в порядке возрастания правой границы, а надо было так, как они встречались во входных данных.

2. Вторая проблема была на четвёртом тесте. Если отрезок в покрытие не находился надо было немедленно выходить из функции.

0.5 Тест производительности

В тесте решил удостовериться, что программа укладывается в свою сложность. Для этого сгенерировал большие тесты.



Выглядит не очень, но явно не квадрат. Это ниже $n \log n$, потому что в больших тестах покрытие одним отрезком встречается довольно часто.

0.6 Выводы

Когда выполнил 8-ю лабораторную работу, то понял разницу между жадным алгоритмом и динамическим программированием.

Динамическое программирование подразумевает анализ решений подзадач и выбор из него оптимального. В жадном же алгоритме выбор происходит каждый шаг в надежде, что он оптимален.

Жадный алгоритм выигрывает у динамического программирования, что он не требует мемоизации состояний, а, значит, много памяти у нас остаётся свободной.

Но он же и проигрывает в некоторых задачах. Например, для дискретного рюкзака жадный алгоритм не всегда даёт оптимальный результат, в отличие от непрерывного. Другой пример, представим граф с 7 вершинами. Между первой и седьмой есть путь длины 5, остальные вершины соединены единичными рёбрами от первой до седьмой. Жадность в выборе миниального ребра тут приведёт к неверному результату.

Литература

[1] Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн *Алгоритмы Построение и анализ* — . Третье издание, Москва, Санкт-Петербург, Киев, 2013, 1324 с.

[2] *Жадный алгоритм — Википедия.*
URL: `https://ru.wikipedia.org/wiki/?curid=171223` (дата обращения: 11.05.2021).