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: К. М. Воронов
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №8

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

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат входных и выходных данных описан в варианте 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 входных данных

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.

Формат результата

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.

1 Описание

Требуется решить задачу с использованием жадного алгоритма.

Как сказано в [1]: «Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным.» В данной задаче можно поступить таким образом: сначала нужно отсортировать все отрезки по левой границе. Далее, мы будем смотреть все отрезки, левые границы которых не заходят вправо за ту часть, которую мы покрыли, и выбирать из них тот, который заканчивается правее всего. После этого обновлять покрытую часть - она станет правой границей выбранного отрезка. Если такой отрезок найти не удаётся - значит покрытие невозможно. Получается, мы в данный момент выбираем самое оптимальное решение - отрезок, который заканчивается правее всего. Чтобы доказать оптимальность данного алгоритма, надо сравнить его решение с неким оптимальным решением. Допустим, на каком-то этапе отрезок, выбранный жадным алгоритмом не совпадает с отрезком из оптимального. Тогда в оптимальном решении будет другой отрезок, покрывающий данную точку, и мы можем заменить его на отрезок из жадного решения. Решение останется решением, ведь все точки до этого были покрыты. Продолжая таким образом, мы дойдём до конца отрезка. Сложность жадного алгоритма - O(n), так как мы один раз проходимся по всем отрезкам.

2 Исходный код

Для хранения отрезков создадим структуру TEdge, в которой будет два параметра: левая и правая граница. Так же напишем там оператор сравнения по левой границе для сортировки.

```
##include<iostream>
#include<vector>
#include<algorithm>
#include<unistd.h>

using namespace std;

struct TEdge{
    int L;
    int R;

    bool operator<(const TEdge &t) {
        return L < t.L;
    }
};

int main() {
    int n, m;
    cin >> n;
    vector<TEdge> v;
    for (int i = 0; i < n; ++i) {
        int l, r;
        cin >> l >> r;
        v.push_back({l, r});
    }
    vector<TEdge> begin = v;
    sort(v.begin(), v.end());
    cin >> m;
    vector<int> res;

    int empty = 0;
    bool zero = false;
    int j = -1;

    while (empty < m && !zero) {
        int index = j + 1;
        int right = empty;
        for (long unsigned int i = index; i < v.size(); ++i) {
            if(v[i].L > empty) {
                break;
            }
            if (v[i].R > right) {
                j = i;
```

```
44                  right = v[j].R;
45              }
46          }
47          if (j == index - 1) {
48              zero = true;
49              break;
50          }
51          empty = v[j].R;
52          for (long unsigned int k = 0; k < begin.size(); ++k) {
53              if (v[j].L == begin[k].L && v[j].R == begin[k].R) {
54                  res.push_back(k);
55                  break;
56              }
57          }
58      }
59      sort(res.begin(), res.end());
60      if (zero) {
61          cout << "0" << endl;
62      } else {
63          cout << res.size() << endl;
64          for (long unsigned int i = 0; i < res.size(); ++i) {
65              cout << begin[res[i]].L << " " << begin[res[i]].R << endl;
66          }
67      }
68  }
```

3 Консоль

```
kirill@kirill-G3-3779:~/DA/lab8$ cat test.txt
4
0 2
2 4
4 5
0 4
5
kirill@kirill-G3-3779:~/DA/lab8$ ./s* <test.txt
2
4 5
0 4
```

4 Тест производительности

Тест производительности представляет из себя следующее: сравнение жадного алгоритма и наивного. Размер тестов - 15 и 30 отрезков.

```
kirill@kirill-G3-3779:~/DA/lab8$ ./a* <tests/1.in
Наивный алгоритм: 3.802ms
kirill@kirill-G3-3779:~/DA/lab8$ ./s* <tests/1.in
Жадный алгоритм: 0.020ms
```

```
kirill@kirill-G3-3779:~/DA/lab8$ ./a* <tests/1.in
Наивный алгоритм: 63489.423ms
kirill@kirill-G3-3779:~/DA/lab8$ ./s* <tests/1.in
Жадный алгоритм: 0.031ms
```

Как видно, наивный алгоритм, который основан на переборе, работает намного дольше даже на маленьких тестах, так как его сложность $O(2^n)$.

5 Выводы

Выполнив восьмую лабораторную работу по курсу «Дискретный анализ», я научился решать задачи с использованием жадных алгоритмов. У данных алгоритмов есть недостатки: выбирать всегда наилучший вариант в данной ситуации не всегда приводит к оптимальному решению. Иногда не стоит быть жадным. Эти алгоритмы проводят хорошую аналогию на решении задач с жизнью, заставляя задуматься над своими действиями в той или иной ситуации. Прежде чем применять такой метод, следует удостовериться в его эффективности.

Список литературы

[1] Жадный алгоритм
https://ru.wikipedia.org/wiki/Жадный_алгоритм

[2] Лекции Н.К.Макарова, Московский авиационный институт.