Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 8 по курсу дискретного анализа: Жадные алгоритмы

Студент: А. О. Тояков
Преподаватель: А. Н. Ридли
Группа: М8О-307Б-18
Дата:
Оценка:
Подпись:

Москва
2021

Условие

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 [Li, Ri]. Необходимо выбрать минимальное количество отрезков, которые бы полностью покрыли интервал [0, M].

Метод решения

Для начала мне понадобится структура Segment, чтобы хранить отрезки. В ней будут хранится 3 числа: left, right , ind, которые будут обозначать левый конец отрезка, правый конец отрезка и индекст отрезка соответственно.

Для реализации поставленной задачи понадобится вектор. Проходя по всем отрезкам, мы будем добавлять в вектор такой отрезок, чтобы его левый конец покрывал уже пройденную область, а правый был максимальным. Будем выполнять этот алгоритм, пока не покроем всю область, иначе выведем 0 (по усл.).

Описание программы

Реализованный жадный алгоритм очень прост. В функции main мы вводим исходные значения, а затем запускаем функцию Selection, которая и решает поставленную задачу. В ней мы инициализируем вектор answer, в котором мы будем хранить отрезки, покрывающие область. Затем запускается цикл, в котором проверяется условие, что правый край крайнего отрезка в векторе меньше введённого числа m, и тогда мы проходим по всем отрезкам, находя отрезок оптимальной длины, т. е. такой, что его левый край покрывает уже пройденный путь, а правый является максимальным. Если мы не нашли такой отрезок, то выходим из функции и печатаем 0, иначе мы заносим его в вектор answer. На выходе из цикла вектор сортируется по индексам с помощью компаратора и результат печатается на выход.

Исходный код

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

using namespace std;

struct TSegment {
int left;
```

```
int right;
int ind;
};

bool Cmp (TSegment &a, TSegment &b) {
return a.ind < b.ind;
}

void Selection(vector<TSegment> &segs, int m) {
vector<TSegment> answer;
TSegment begin;
begin.left = 0;
begin.right = 0;
answer.push_back(begin);
while (answer.back().right < m) {
int max = 0;
int index = -1;
for (int i = 0; i < segs.size(); i++) {
if (segs[i].left <= answer.back().right && segs[i].right > answer.back().right) {
if (segs[i].right > max) {
max = segs[i].right;
index = i;
}
}
}
if (index == -1) {
cout << "0\n";
return;
} else {
answer.push_back(segs[index]);
}
}

sort(answer.begin() + 1, answer.end(), Cmp);
cout << answer.size() - 1 << '\n';
for (int i = 1; i < answer.size(); i++) {
cout << answer[i].left << " " << answer[i].right << '\n';
}
return;

}
```

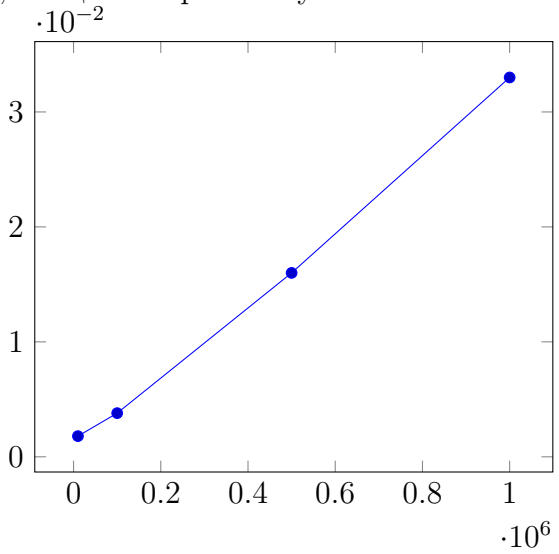
```
int main() {
int n, m;
cin >> n;
vector<TSegment> segs(n);
for (int i = 0; i < n; i++) {
segs[i].ind = i;
cin >> segs[i].left >> segs[i].right;
}
cin >> m;
Selection(segs, m);
return 0;
}
```

Дневник отладки

Неправильный ответ на тесте 3: итоговый массив не сортировался, поэтому отрезки могли выводиться в неправильном порядке. Была добавлена сортировка с помощью функции std::sort и компаратора Cmp, который сортирует элементы по индексам.

Тест производительности

Тесты представляют из себя файлы, в которых сгенерировано различное число отрезков, концы которых могут быть числами от -10000 до 10000.



Пояснения к графику: Ось y - время в секундах. Ось x - количество отрезков.

Выводы

Жадные алгоритмы, как и динамическое программирование основываются на свойстве оптимальности подзадач. В жадном алгоритме мы принимаем локальные решения, до-

пуская, что конечное решение также окажется оптимальным. Это работает не всегда и в данной лабороторной сложность меняется от O(n) до $O(n^2)$ в зависимости от входных данных. Однако, если данное свойство выполняется, то реализовать жадный алгоритм будет проще, чем наивный, и также мы получим выигрыш в скорости.

Проверить это можно с помощью так называемого матроида. Он представляет из себя конечный автомат с двумя состояниями (X, I). В свою очередь для него также должны выполнятся 3 свойства, а именно: множество I не может быть пустым, любое подмножество любого элемента из I также будет элементом этого множества и если A и B - множества, принадлежащие I, причём $A < B$, то какой нибудь элемент x из B, не принадлежащий A, в объединении с A будет элементом множества I.

Если доказать, что наш объект это матроид, то жадный алгоритм будет работать корректно, иначе придётся пользоваться, например, динамическим программированием или даже наивными алгоритмами.