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. О. Дубинин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

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++, соответствующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

Вариант 2 (Выбор отрезков): На координатной прямой даны несколько отрезков с координатами [Li, Ri]. Необходимо выбрать минимальное количество отрезков, которые бы полностью покрыли интервал [0, M]. Формат входных данных: на первой строчке располагается число N, за которым следует N строк на каждой из которой находится пара чисел Li, Ri ; последняя строка содержит в себе число M . Формат выходных данных: 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.

Пример:

Входной файл	**Выходной файл**
3 -1 0 -5 -3 2 5 1	0

Входной файл	**Выходной файл**
2 -1 0 0 1 1	1 0 1

1 Описание

Согласно[1], суть жадных алгоритмов заключается в принятии на каждом этапе локально оптимальных решений, допуская, что конечное решение также окажется оптимальным. Чтобы задачу можно было решить жадным алгоритмом, последовательность локально оптимальных выборов должна давать глобально оптимальное решение.

Наша задача похожа на классическую задачу о выборе процессов[1]. Для её решения время процессов сортировалась по окончанию процессов, брался первый процесс и далее в цикле выбирался первый подходящий и тд. Для нашей задачи всё наоборот, мы сортируем наши отрезки по началу отрезка, выбираем последний подходящий к началу нашего отрезка [0, M], чтобы он покрывал начальную точку 0. Далее в цикле выбираем остальные точки, по тому же принципу, рассматривая конец последнего взятого отрезка, а не точку 0.

Сложность программы состоит из двух частей: из самого выбора отрезков за $O(n)$, так как каждый отрезок мы рассматриваем максимум два раза, и из сортировки отрезковов, стандартная функция c++ sort выполняет сортировку за $O(n * logn)$. Т.е. итоговая сложность программы $O(n * logn)$.

2 Исходный код

main.cpp

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

class TSectionItem {
public:
    int start;
    int finish;
    int index;

    TSectionItem &operator=(const TSectionItem &rhs) = default;
};

bool Cmp(TSectionItem &lhs, TSectionItem &rhs) {
    return (lhs.start < rhs.start);
}

bool CmpIndex(TSectionItem &lhs, TSectionItem &rhs) {
    return (lhs.index < rhs.index);
}

void SegmentSelection(std::vector<TSectionItem> &points, int m) {
    std::vector<TSectionItem> ans;

    // sort by left(start) point
    std::sort(points.begin(), points.end(), Cmp);

    int index = 0;
    // pick first correct section
    int lastCorrectSection = -1;
    for (; index < points.size(); ++index) {
        if (points[index].start <= 0 && points[index].finish >= 0) {
            lastCorrectSection = index;
        } else if (points[index].start > 0) {
            break;
        }
    }
    // fail
    if (lastCorrectSection == -1) {
        std::cout << 0 << std::endl;
        return;
    }
    //else
    ans.push_back(points[lastCorrectSection]);

    //pick the remaining sections
```

```
47      bool complete = false;
48      lastCorrectSection = -1;
49      while (index < points.size()) {
50          // find next correct section
51          for (; index < points.size(); ++index) {
52              if (points[index].start <= ans.back().finish && points[index].finish >= ans
                    .back().finish) {
53                  lastCorrectSection = index;
54                  // if complete
55                  if (points[index].finish >= m) {
56                      complete = true;
57                      break;
58                  }
59              } else if (points[index].start > ans.back().finish) { // last section was
                    best
60                  break;
61              }
62          }
63          //fail
64          if (lastCorrectSection == -1) {
65              std::cout << 0 << std::endl;
66              return;
67          }
68          //else
69          ans.push_back(points[lastCorrectSection]);
70          if (m <= ans.back().finish || complete) {
71              //complete
72              break;
73          }
74          lastCorrectSection = -1;
75      }
76
77
78      if (m <= ans.back().finish) {
79          std::cout << ans.size() << std::endl;
80          // sort by index
81          std::sort(ans.begin(), ans.end(), CmpIndex);
82          for (auto j : ans) {
83              std::cout << j.start << " " << j.finish << std::endl;
84          }
85      } else {
86          std::cout << 0 << std::endl;
87      }
88  }
89
90  int main() {
91
92
93      size_t n;
```

```
94      std::cin >> n;
95
96      std::vector<TSectionItem> points(n);
97
98      for (int i = 0; i < n; ++i) {
99          points[i].index = i;
100         std::cin >> points[i].start >> points[i].finish;
101     }
102
103     int m;
104     std::cin >> m;
105
106     SegmentSelection(points, m);
107
108     return 0;
109 }
```

3 Консоль

```
art@mars:~/study/DA/labs/lab_8$ ./main
8
-3 3
6 13
7 10
-2 0
-1 4
4 6
3 5
5 7
10
3
6 13
-1 4
4 6
```

4 Выводы

Жадное программирование очень удобно в большинстве задач, которые возникают при решении каких либо реальных проблем, поскольку случай когда нужно на каждом шаге делать локально наилучший выбор в надежде, что итоговое решение будет оптимальным, довольно простой и, даже, очевидный концепт для человека и следовательно его легко применять в жизни.

Разница между динамичесским и жадным программированием заключается в том, что в динамическом программировании на каждом шаге можно рассмотреть несколько решений в поисках оптимального, то для жадных алгоритмов такой путь будет всего один – предположительно, оптимальный. Разница лишь в том, что в динамическом программировании подзадачи решаются до выполнения первого выбора, а жадный алгоритм делает первый выбор до решения подзадач.

Однако такая простота применения, в точных научных задачах, либо высокоточных системах выливается в сложное доказательство применимости жадного алгоритма для её решения.

Конкретным примером жадных алгоритмов являются: алгоритм Хаффмана для адаптивного кодирования алфавита, алгоритм Крускала (поиск остовного дерева минимального веса в графе), алгоритм Прима (поиск остовного дерева минимального веса в связном графе).

Список литературы

[1] Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн *Алгоритмы: построение и анализ, 3-е изд.* — Издательский дом « Вильямс», 2013. Перевод с английского: ООО «И. Д. Вильямс». — 1328 с. (ISBN 978-5-8459-1794-2 (рус.))