Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студентка: А. А. Довженко
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2018

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 4: Откорм бычков. Бычкам дают пищевые добавки, чтобы ускорить их рост. Каждая добавка содержит некоторые из N действующих веществ. Соотношения количеств веществ в добавках могут отличаться. Воздействие добавки определяется как

$c_1a_1 + c_2a_2 + \ldots + c_Na_N$,

где a_i — количество i-го вещества в добавке, c_i — неизвестный коэффициент, связанный с веществом и не зависящий от добавки. Чтобы найти неизвестные коэффициенты c_i, Биолог может измерить воздействие любой добавки, использовав один её мешок. Известна цена мешка каждой из M ($M \geq N$) различных добавок. Нужно помочь Биологу подобрать самый дешевый наобор добавок, позволяющий найти коэффициенты c_i. Возможно, соотношения веществ в добавках таковы, что определить коэффициенты нельзя.

Входные данные: в первой строке текста — целые числа M и N ; в каждой из следующих M строк записаны N чисел, задающих соотношение количеств веществ в ней, а за ними — цена мешка добавки. Порядок веществ во всех описаниях добавок один и тот же, все числа — неотрицательные целые не больше 50. Выходные данные: -1, если определить коэффциенты невозможно, иначе набор добавок (и их номеров по порядоку во входных данных). Если вариантов несколько, вывести какой-либо из них.

1 Описание

Жадный алгоритм [1] 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. К тому же, оптимальное решение задачи должно содержать в себе оптимальные решения подзадач. Для теоретических изысканий иногда используются матроиды: если показать, что объект является матроидом, то жадный алгоритм будет работать корректно. Свойства матроида приблизительно совпадают с описанными выше неформальными свойствами.

Этапы построения жадного алгоритма:

- Привести задачу оптимизации к виду, когда после сделанного выбора остаётся решить только одну подзадачу.
- Доказать, что всегда существует такое оптимальное решение исходной задачи, которое можно получить путём жадного выбора, так что такой выбор всегда допустим.
- Продемонстрировать оптимальную структуру, показав, что после жадного выбора остаётся подзадача, обладающая тем свойством, что объединение оптимального решение подзадачи со сделанным жадным выбором приводит к оптимальному решению исходной задачи.

2 Исходный код

Для решения приводим исходную матрицу к ступенчатому виду методом Гаусса. В отличие от метода Гаусса, нам не нужно искать решение системы, что упрощает задачу. Если на каком-то шаге не удается найти строку, содержащую ненулевой элемент, и привести матрицу к ступенчатому виду, то система несовместна, и её решения не существует.

Алгоритм выглядит следующим образом: среди элементов первого столбца матрицы выбираем ненулевой, чья строка имеет наименьшую стоимость, перемещаем его на крайнее верхнее положение перестановкой строк и вычитаем получившуюся после перестановки первую строку из остальных строк, домножив её на величину, равную отношению первого элемента каждой из этих строк к первому элементу первой строки, обнуляя тем самым столбец под ним. После того, как указанные преобразования были совершены, проделываем тоже самое с остальными столбцами, только теперь перестановка строк и арифметические операции осуществляются со следующих по счёту строк сверху.

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

int main(void)
{
    bool hasSolution = true;
    int m = 0, n = 0, cost = 0, numRow = 0;
    std::cin >> m >> n;
    cost = n;
    numRow = n + 1;

    std::vector<int> res;
    std::vector<std::vector<double> > elems;
    elems.resize(m);
    for (int i = 0; i < m; ++i) {
        elems[i].resize(n + 2);
    }

    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n + 1; ++j) {
            std::cin >> elems[i][j];
        }
        elems[i][numRow] = i + 1;
    }


    for (int col = 0; col < n; ++col) {
        int minRow = -1;
        int minCost = 51;
```

```
        for (int row = col; row < m; ++row) {
            if (elems[row][col] != 0 && elems[row][cost] < minCost) {
                minRow = row;
                minCost = elems[row][cost];
            }
        }

        if (minRow == -1) {
            hasSolution = false;
            std::cout << "-1" << std::endl;
            break;
        }

        elems[col].swap(elems[minRow]);
        res.push_back(elems[col][numRow]);
        for (int row = col + 1; row < m; ++row) {
            double c = elems[row][col] / elems[col][col];
            for (int i = col; i < n; ++i) {
                elems[row][i] -= elems[col][i] * c;
            }
        }
    }

    std::sort(res.begin(), res.end());
    if (hasSolution) {
        for (int i = 0; i < res.size(); ++i) {
            std::cout << res[i];
            if (i != res.size() - 1) {
                std::cout << " ";
            }
        }
        std::cout << std::endl;
    }

    return 0;
}
```

3 Консоль

```
karma@karma:~/mai_study/DA/lab8$ make
g++ -std=c++14 -O2 -pedantic -lm -o da8 main.cpp
karma@karma:~/mai_study/DA/lab8$ cat test
5 3
1 1 1 1
4 2 1 3
4 2 1 5
1 2 3 1
4 6 8 2
karma@karma:~/mai_study/DA/lab8$ cat test | ./da8
1 2 4
```

4 Тест производительности

Тест производительности представляет собой сравнение с наивным решением этой задачи, в котором перебираются все возможные подсистемы уравнений, а потом выбирается решение с наименьшей стоимостью.

Генерирую 4 файла, содержащие 10 (input10), 20 (input20), 30 (input30) и 40 (input40) уравнений с 10, 20, 30 и 40 неизвестными коэффициентами соответственно.

```
karma@karma:~/mai_study/DA/lab8$ cat input10 | ./da8
Greed time: 7.8e-05 sec.
karma@karma:~/mai_study/DA/lab8$ cat input10 | ./naive
Naive time: 9.9e-05 sec.
karma@karma:~/mai_study/DA/lab8$ cat input20 | ./da8
Greed time: 0.000228 sec.
karma@karma:~/mai_study/DA/lab8$ cat input20 | ./naive
Naive time: 0.002703 sec.
karma@karma:~/mai_study/DA/lab8$ cat input30 | ./da8
Greed time: 0.00017 sec.
karma@karma:~/mai_study/DA/lab8$ cat input30 | ./naive
Naive time: 0.000862 sec.
karma@karma:~/mai_study/DA/lab8$ cat input40 | ./da8
Greed time: 0.001095 sec.
karma@karma:~/mai_study/DA/lab8$ cat input40 | ./naive
Naive time: 29.5344 sec.
```

Как видим, на малых тестах разница почти неощутима, но начиная с 40 уравнений, жадное решение значительно выигрывает у наивного. Предполагаемая сложность наивного алгоритма – $O(m * 2^n)$, а жадного – $O(m * n^2)$. Результаты тестирования это подтверждают. Также, наивное решение затрачивает больше памяти, потому что приходится хранить исходную систему уравнений и обрабатываемую на текущем шаге подсистему. Категорически недопустимые затраты для подобной задачи.

5 Выводы

Стоит сразу же заметить, что если глобальная оптимальность алгоритма имеет место практически всегда, его обычно предпочитают другим методам, таким как динамическое программирование. К тому же очевидно, что ситуация встречается довольно часто: жадным является любой алгоритм, который на каждом шаге делает локально наилучший выбор в надежде, что итоговое решение будет оптимальным. В некотором смысле можно считать, что жадные алгоритмы – частный случай алгоритмов динамического программирования: в самом деле, если 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.

Немотря на простоту применения, для каждой задачи требуется подчас весьма сложное доказательство применимости жадного алгортма для её решения. Область применения жадных алгоритмов широка: кодирование Хаффмана для сжатия данных, алгоритмы аллокации в операционных системах, многие алгоритмы на графах et cetera.

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.))