Национальный исследовательский университет
"Московский авиационный институт"
Факультет No8 "Информационные технологии и прикладная математика"
Кафедра 806 "Вычислительная математика и программирование"

ЛАБОРАТОРНАЯ РАБОТА №8
ПО КУРСУ “ДИСКРЕТНЫЙ АНАЛИЗ”
4 СЕМЕСТР

Выполнил студент: Поляков А.И.
Группа: М80-208Б-19
Оценка:
Подпись:

Москва 2022

Лабораторная работа №8

Вариант №04.

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.

Вариант алгоритма:

Бычкам дают пищевые добавки, чтобы ускорить их рост. Каждая добавка содержит некоторые из N действующих веществ. Соотношенияколичеств веществ в добавках могут отличаться. Воздействие добавки определяется как $c_1a_1 + c_2a_2 + \ + c_Na_N$, 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.

Описание

Жадный алгоритм 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.

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.

Программный код состоит из двух файлов: *lib.h*, в котором реализованны функция Гаусса для приведения матрицы к ступенчатому виду, и *main.cpp* - основной файл программы с реализацией ввода.

Заголовочный файл lib.h

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

typedef std::vector<double> doub_vector;
typedef std::vector<double> int_vector;

void GaussAlgorythm(std::vector<doub_vector>& vec,
doub_vector& res, int& N, int& M){
    int cost = N;
    for(int i = 0; i < N; i++){
        int rowNum, row = -1;
        int minCost = 51;
        for(int j = i; j < M; j++){
            if(vec[j][i] != 0 && vec[j][cost] < minCost){
                minCost = vec[j][cost];
                row = j;
            }
        }

        if(row == -1){
            res.clear();
```

```
            return;
        }

        rowNum = vec[row][cost+1];

        res.push_back(rowNum);
        vec[i].swap(vec[row]);

        for(int j = i + 1; j < M; j++){
            double k = vec[j][i] / vec[i][i];
            for(int l = i; l < N; l++){
                vec[j][l] -= vec[i][l] * k;
            }
        }
    }
}

std::pair<int_vector, int> NaiveAlgorythm(int_vector freepos,
int N, int start, std::vector<doub_vector>& arr){
    std::pair<int_vector, int> pairs;
    int_vector currentFree, result, currentResult;
    int sum = 100000;
    if(start == N){
        return std::make_pair(result, 0);
    }
    for(int j = 0; j < freepos.size(); j++){
        if(arr[freepos[j]][start]){

            currentFree = freepos;
            int_vector::iterator it = currentFree.begin()+j;
            currentFree.erase(it);
            pairs = NaiveAlgorythm(currentFree, N, start+1, arr);
            pairs.first.push_back(freepos[j]);

            if(sum > pairs.second + arr[freepos[j]][N]){
                sum = pairs.second + arr[freepos[j]][N];
                result = pairs.first;

            }
        }
    }
    return std::make_pair(result, sum);
}
```

Файл main.cpp

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

#include "lib.h"

int main() {
    int M, N;
    int p;
    std::cin >> M >> N;
    std::vector<doub_vector> coefficient;
    doub_vector current;

        // Fill vectors with input values
    for (int i = 0; i < M; i++) {
           for (int j = 0; j < N + 1; j++) {
                  std::cin >> p;
                  current.push_back(p);
           }
           current.push_back(i + 1);
           coefficient.push_back(current);
           current.clear();
    }

        //
    GaussAlgorythm(coefficient, current, N, M);

        // Print results
    if (current.empty()) {
           std::cout << -1 << std::endl;

    } else {
           std::sort(current.begin(), current.end());
           std::cout << *current.begin();

           for (doub_vector::iterator it = current.begin() + 1;
           it != current.end(); it++) {
                  std::cout << " " << *it;
           }

           std::cout << std::endl;
```

```
    }
}
```

Тест производительности

Консольный вывод

```
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ make
g++ -std=c++14 -O2 -pedantic -lm -o da8 main.cpp
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/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
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat test | ./da8
1 2 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.

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

```
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input10 | ./da
Greed time: 7.8e-05 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input10 | ./na
Naive time: 9.9e-05 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input20 | ./da
Greed time: 0.000228 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input20 | ./na
Naive time: 0.002703 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input30 | ./da
Greed time: 0.00017 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input30 | ./na
Naive time: 0.000862 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input40 | ./da
Greed time: 0.001095 sec.
user@AN-LAP-1110:/mnt/c/Users/Andrew/Desktop/ДА 4/lab8$ cat input40 | ./na
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.

Вывод

В данной лабораторной работе я научился решать задачи с применением жадных алгоритмов. 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 сложное доказательство применимости жадного алгортма для её решения.

Литература

[1] Классические алгоритмы поиска образца в строке, Андрей Калинин. 28 октября 2011 г.

[2] Дэн Гасфилд. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология. — Издательский дом «Невский Диалект», 200ц. Перевод с английского: И. В. Романовского. — 654с. (ISBN 5-7940-0103-8 (рус.))