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: М. С. Гаврилов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

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.

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 c1a1 + c2a2 +· · ·+cNaN, где ai количество i-го вещества в добавке, ci — неизвестный коэффициент, связанный с веществом и не зависящий от добавки. Чтобы найти неизвестные коэффициенты ci, Биолог может измерить воздействие любой добавки, использовав один её мешок. Известна цена мешка каждой из M (M N) различных добавок. Нужно помочь Биологу подобрать самый дешевый наобор добавок, позволяющий найти коэффициенты ci. Возможно, соотношения веществ в добавках таковы, что определить коэффициенты нельзя.

Формат входных данных
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.

Формат результата
-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 и исходный код

Заметим, что запись из составов добавок и сопоставленных им воздействий представляет собой систему линейных уравнений. Чтобы с помощью набора добавок можно было определить воздействие каждого компонента, необходимо и достаточно чтобы система имела решение. Таким образом, наша задача состоит в том чтобы составить из данных добавок наиболее дешевый набор, образующий систему линейных уравнений, имеющую решение. Чтобы система имела решение необходимо, чтобы ее матрица имела ранг равный количеству строк, а количество столбцов не превосходило количество строк. Для определения ранга будем использовать ступенчатый вид. Функция эшелонизации сразу проверяет, является система линейно независимой (т.е. равен ли ранг количеству строк).
Сам жадный алгоритм будет действовать следующим образом:
1. Выбирается самая дешевая из оставшихся добавок и добавляется в матрицу проверки.
2. Проверяется линейная независимость матрицы. Если она ЛЗ, то строка удаляется, алгоритм переходит в п.1.
3. Если система из строк матрицы ЛНЗ, то проверяется, равно ли число строк матрицы числу столбцов (неизвестных в системе уравнений). Если число строк меньше числа столбцов, то переходим в п.1. иначе – полученный набор строк – оптимальный.

Класс матрицы и функции ее заполнения и приведения к ступенчатому виду:

```
1
2  struct TMatrix {
3      std::vector<std::vector<double>> Body;
4      int M, N;
5      TMatrix(int m, int n) : M(m), N(n)
6      {
7          Body = std::vector<std::vector<double>>(m);
8          for (int i = 0; i < m; ++i) {
9              Body[i] = std::vector<double>(n , 0);
10         }
11     }
12     void FillMatrix(std::multimap<double, int> &inter/*   */) {
13         for (int i = 0; i < M; ++i) {
14             for (int j = 0; j < N; ++j) {
15                 std::cin >> Body[i][j];
16             }
17             double intV;
18             std::cin >> intV;
19             inter.emplace(std::pair<double, int>(intV, i));
20         }
21     }
22     int Echelonise() {
```

```
        int startRow = 0;
        for (int k = 0; k < N && startRow < M; ++k) {
            if (Body[startRow][k] == 0) {// ,    0
                for (int j = startRow; j < M; ++j) {
                    if (Body[j][k] != 0) {
                        std::vector<double> f = Body[startRow];
                        Body[startRow] = Body[j];
                        Body[j] = f;
                        break;
                    }
                }
            }
            if (Body[startRow][k] == 0) {
                //,  startRow
                if (DEBUG) std::cout << "[k|sc: " << k << " | " << startRow << "
                    continuing\n";
                continue;
            }
            for (int i = startRow + 1; i < M; i++) { //
                double factor = Body[i][k] / Body[startRow][k];
                if(DEBUG) std::cout << "[i|k|sc: " << i << " | " << k << " | " <<
                    startRow << " factor: " << factor << " sizes: " << M << " " << N <<
                    std::endl;
                for (int j = k; j < N; ++j) {
                    Body[i][j] -= Body[startRow][j] * factor;
                }
            }

            startRow++;
        }
        int lastRow = M - 1;
        for (int i = 0; i < N; ++i) {
            if (Body[lastRow][i] > 0.00001 || Body[lastRow][i] < -0.00001) {
                //
                return 0;
            }
        }
        //
        return 1;
    }
};
```

Основная функция, реализующая алгоритм

```
int main() {

    int m, n;
    std::cin >> m >> n;

    std::multimap<double, int> costs;
```

```
7       std::map<int, int> res;
8       TMatrix test(m,n);
9
10      test.FillMatrix(costs);
11
12      /*
13          1.     .
14          2.          .
15          3.       .
16          4.     ,     .     .
17      */
18
19      long long cost = 0;
20      TMatrix resMatr = TMatrix(0, test.N);
21
22      if (DEBUG) {
23          printf("prev matr: \n");
24          for (int i = 0; i < test.M; ++i) {
25              for (int j = 0; j < test.N; ++j) {
26                  printf("%lf ", test.Body[i][j]);
27              }
28              printf("\n");
29          }
30          printf("\nsumm\n");
31      }
32
33      if (test.N == 0) {
34          std::cout << "-1" << std::endl;
35          return 0;
36      }
37
38      for (std::pair<int, double> iter : costs) {
39          //
40
41          resMatr.Body.push_back(test.Body[iter.second]);
42          resMatr.M++;
43          //
44
45
46          if (DEBUG) {
47              printf("attempting to add: \n");
48              for (int i = 0; i < test.Body[iter.second].size(); ++i) {
49                  printf("%lf ", test.Body[iter.second][i]);
50              }
51              printf("\n\n");
52              for (int i = 0; i < resMatr.M; ++i) {
53                  for (int j = 0; j < resMatr.N; ++j) {
54                      printf("%lf ", resMatr.Body[i][j]);
55                  }
```

```
56                 printf("\n");
57             }
58         }
59 
60 
61         TMatrix checkMatr = resMatr;
62         int LinDep = checkMatr.Echelonise();
63 
64 
65         if (DEBUG) {
66             printf("echelonized:\n");
67             for (int i = 0; i < checkMatr.M; ++i) {
68                 for (int j = 0; j < checkMatr.N; ++j) {
69                     printf("%lf ", checkMatr.Body[i][j]);
70                 }
71                 printf("\n");
72             }
73         }
74 
75         if (LinDep == 0) {
76             //    ,
77             cost += iter.first;
78             res.emplace(std::pair<int, int>(iter.second, iter.second));
79         }
80         else {
81             // ,
82             resMatr.M--;
83             resMatr.Body.pop_back();
84         }
85 
86 
87         if (resMatr.M == resMatr.N) {
88             //    . ,  .
89             break;
90         }
91     }
92 
93     if (resMatr.M != resMatr.N) {
94         std::cout << "-1" << std::endl;
95         return 0;
96     }
97 
98     for (auto iter : res) {
99         std::cout << iter.first + 1 << " ";
100    }
101    std::cout << std::endl;
102 }
```

2 Досказательство корректности и Тест эффективности

Взглянем на решение с несколько другой стороны. Если разбить все строки на наборы подобных (попарно линейно зависимых) строк, то задача сведется к выбору оптимальной, то есть самой дешевой строки из каждого набора. Таким образом мы доказали, что с точки зрения решаемости системы нет разницы, какую из подобных строк выбрать для записи в нее. С другой стороны, если строка представляет собой линейную комбинацию других N строк, то любые N из этих ЛЗ N + 1 строк будут линейно независимы, а значит, для решаемости системы также нет разницы, в каком порядке добавлять эти строки в нее. Так что добавляя в итоговую матрицу строки таким образом, чтобы она оставалась ЛНЗ, мы не закрываем пути к оптимальному решению.
После совершения локально оптимального выбора (добавления в матрицу наиболее дешевой не приводящей ее к линейной зависимости строки) мы также как и на прошлом шаге имеем матрицу и набор из строк для поиска в нем оптимальной для добавления, а значит, задача на следующем шаге эквивалентна задаче на предыдущем. Таким образом корректность жадного алгоритма доказана.

Выполним оценку эффективности по времени:
1. Эшелонизация выполняется за O(m*n*n), где m и n – размеры матрицы.
2. в ходе работы основного алгоритм в худшем случае выполняется M попыток добавить строку в матрицу, где M – число различных добавок.
3. Эшелонизация выполняется один раз за попытку, так что имеем предполагаемую сложность O(M*m*n*n) или O(M^2*N^2*1/4)

Выполним оценку эффективности по памяти:
В каждый момент существует лишь две матрицы: основная и тестовая (нужна для проверки ЛЗ) O(m*n)

```
max@max-Swift:~/Рабочий стол/ДА/lab8/DA_lab_8-main$ ./gen.exe 100 >testx
max@max-Swift:~/Рабочий стол/ДА/lab8/DA_lab_8-main$ ./lab8.exe <testx >resx
test complete| 172ms
max@max-Swift:~/Рабочий стол/ДА/lab8/DA_lab_8-main$ ./gen.exe 200 >testx
max@max-Swift:~/Рабочий стол/ДА/lab8/DA_lab_8-main$ ./lab8.exe <testx >resx
test complete| 2653ms
max@max-Swift:~/Рабочий стол/ДА/lab8/DA_lab_8-main$ ./gen.exe 400 >testx
max@max-Swift:~/Рабочий стол/ДА/lab8/DA_lab_8-main$ ./lab8.exe <testx >resx
test complete| 42156ms
```

```
max@max-Swift:~/Рабочий стол/ДА/lab8/DA_lab_8-main$
```

Видим увеличение времени работы в 16 раз при увеличении размеров матрицы в 2 раза. Сложность X^4.

3 Выводы

В ходе выполнения этой лабораторной работы я получил опыт применения жадных алгоритмов при решении практических задач. Также я реализовал некоторые методы работы с матрицами. В целом это была достаточно несложная лабораторная работа, основные проблемы в которой я испытал не сразу поняв что вместо map для записи стоимостей добавок надо использовать multimap.

Список литературы

[1] *Жадный алгоритм — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Жадный_алгоритм`
(дата обращения: 11.05.2021)

[2] *Жадный алгоритм — Хабр.*
URL: `https://habr.com/ru/post/120343/`
(дата обращения: 12.05.2021)