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. А. Почечура
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №8

Задача: 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 $c_1a_1 + c_2a_2+...+c_{NaN}$, 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 ($M \leq N$) 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.

1 Описание

Требуется реализовать жадный алгоритм, решающий поставленную задачу: найти набор добавок минимальной стоимости. Для решения этой задачи используются методы из линейной алгебры. Нам требуется найти минимальную систему линейно независимых уравнений, которая имела бы одно решение. Для этого нам требуется среди всех уравнений найти те, которые бы были друг другу линейно независимы. Реализуется это также, как и в линейной алгебре: путём вычетов строчек матрицы. Главная идея заключается в том, что в начале мы сортируем все уравнения в порядке возрастания цены и каждый раз берём уравнения с наименьшей ценой, которое ещё не было использовано. Если оно будет линейно независимо для уже составленной системы, то мы его берём и продолжаем поиск нового уравнения. Делать мы это можем потому, что заменить линейно независимое уравнение в системе мы можем только линейно зависимым ему же, значит можно спокойно брать самые дешёвые уравнения.

2 Исходный код

Как уже было сказано выше, изначально мы сортируем все добавки по возрастанию цены. В векторе *discharge* на позиции i находится номер уравнения из вектора *preAns*, который отвечает за "ступеньку" матрицы, которая находится в i-том столбце. Получая новую строку, ищем в ней первый ненулевой элемент. Если на этой позиции в матрице ещё нет "ступеньки" , то добавляем уравнение в вектор *preAns* и добавляем его номер в *discharge*. Иначе, с помощью вычетов ищем место для текущего уравнения, на котором ещё нет "ступеньки". Если такое место найдено не было, то уравнение для текущей системы является линейно зависимым и оно нам не подходит. Подбор уравнений продолжается до тех пор, пока количество уравнений не равно количеству неизвестных или пока все уравнения не просмотрены.

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

using namespace std;

double eps = 1;

void CalculateMachineEps() {
    while (1 + eps > 1) {
        eps = eps / 2;
    }
}

vector<double> Difference(vector<double> v, vector<double> v1, int tmp) { //
    double coef = v[tmp] / v1[tmp];
    for (int i = tmp; i < v.size();i++) {
        v[i] = v[i] - v1[i] * coef;
    }
    return v;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr); cout.tie(nullptr);
    int n, m;
    CalculateMachineEps();
    cin >> n >> m;
    vector<vector<double>> input(n, vector<double>(m));
    vector<pair<int, int>> price(n);
    for (int i = 0;i < n;i++) {
        for (int j = 0;j < m;j++) {
            cin >> input[i][j];
        }
        cin >> price[i].first;
```

```
        price[i].second = i;
    }
    sort(price.begin(), price.end());
    vector<int> discharge(m, -1); //
    vector<double> v(m);
    int tmp;
    vector<vector<double>> preAns;
    for (int i = 0;i < n;i++) { //
        v = input[price[i].second];
        tmp = -1;
        for (int j = 0;j < m;j++) {
            if (v[j] > eps) {
                tmp = j;
                break;
            }
        }
        if (tmp == -1) { //
            price[i] = { 1e3, 1e3 };
        }
        else if (discharge[tmp] == -1) {
            discharge[tmp] = preAns.size();
            preAns.push_back(v);
        }
        else {
            while (tmp != -1 && discharge[tmp] != -1) {
                v = Difference(v, preAns[discharge[tmp]], tmp);
                tmp = -1;
                for (int j = 0;j < m;j++) {
                    if (v[j] > eps || v[j] < -eps) {
                        tmp = j;
                        break;
                    }
                }
            }
            if (tmp == -1) { //     ,
                price[i] = { 1e3, 1e3 };
            }
            else { //
                discharge[tmp] = preAns.size();
                preAns.push_back(v);
            }
        }
        if (preAns.size() == m) { //
            break;
        }
    }
    if (preAns.size() != m) {
        cout << -1;
    }
```

```
85      else {
86          vector<int> ans;
87          sort(price.begin(), price.end());
88          for (int i = 0;i < m;i++) { //
89              ans.push_back(price[i].second + 1);
90          }
91          sort(ans.begin(), ans.end());
92          for (int i = 0;i < m;i++) {
93              cout << ans[i] << ' ';
94          }
95      }
96  }
```

3 Консоль

```
root@DESKTOP-5HM2HTK:~# cat <test
3 3
1 0 2 3
1 0 2 4
2 0 1 2
root@DESKTOP-5HM2HTK:~# g++ lab8.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <test
-1
root@DESKTOP-5HM2HTK:~#
```

4 Тест производительности

```
root@DESKTOP-5HM2HTK:~# g++ generator.cpp
root@DESKTOP-5HM2HTK:~# ./a.out tests
root@DESKTOP-5HM2HTK:~# g++ benchmark.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <tests/1.t
N is: 1000
M is: 1000
DP algorithm time: 337ms
Simple algoritm time: 2152ms
root@DESKTOP-5HM2HTK:~#
```

Как видно, жадный алгоритм работает значительно быстрее, чем наивный. Это неудивительно, ведь в жадном алгоритме каждое уравнение берётся ровно один раз, а в наивном подбираются все возможные варианты построения системы.

5 Выводы

Выполнив восьмую лабораторную работу по курсу «Дискретный анализ», я потренировался решать задачи методом жадных алгоритмов и обрёл практику написания алгоритмов из линейной алгебры. Данная работа была для меня интересной и полезной для решения будущих задач.

Список литературы

[1] *Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е издание. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.)).*

[2] *Линейная независимость - Википедия.*
URL: `https://ru.wikipedia.org/wiki/Линейная_независимость` (дата обращения: 19.10.2022).