Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: Е. А. Кондратьев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №8

Задача: Вариант №4 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

Жадный алгоритм заключается в принятии на каждом этапе локально оптимальных решений, предполагая, что конечное решение также окажется оптимальным. Чтобы жадный алгоритм работал, необходимо, чтобы последовательные локальные оптимальные выборы по итогу давали оптимальное глобальное решение. К тому же, оптимальное решение задачи должно содержать в себе оптимальные решения подзадач. Для теоретических выкладок иногда используются матроиды: если показать, что объект является матроидом, то можно показать, что жадный алгоритм будет работать корректно. Свойства матроида примерно пересекаются с изложенными выше свойствами.

2 Исходный код

Заметим, что данная задача — не что иное, как нахождение решения системы линейных алгебраических уравнений, но в более упрощенном виде. Приведем матрицу к ступенчатому виду с помощью метода Гаусса.

Код: main.cpp

```cpp
#include <bits/stdc++.h>

using namespace std;
#define fastIO \
    ios_base::sync_with_stdio(0); \
    cin.tie(0); \
    cout.tie(0)
#define fori(k, n) for (int i(k); i < (n); ++i)
#define forj(h, m) for (int j(h); j < (m); ++j)
#define foru(l, p) for (int u(l); u < (p); ++u)
typedef unsigned long long ull;
typedef long double ld;
typedef long long ll;

typedef vector<long long> vll;
typedef vector<int> vi;
typedef pair<ll, ll> pl;

#define MAX_N 100
//ull MOD = 1000000007LL;
//vector<int> G[MAX_N];
//map<ull, ull> m;

void solve() {
#include <bits/stdc++.h>

using namespace std;
#define fastIO \
    ios_base::sync_with_stdio(0); \
    cin.tie(0); \
    cout.tie(0)
#define fori(k, n) for (int i(k); i < (n); ++i)
#define forj(h, m) for (int j(h); j < (m); ++j)
#define foru(l, p) for (int u(l); u < (p); ++u)
typedef unsigned long long ull;
typedef long double ld;
typedef long long ll;

typedef vector<long long> vll;
typedef vector<int> vi;
typedef pair<ll, ll> pl;
static double eps = numeric_limits<double>::epsilon();
#define MAX_N 50
//ull MOD = 1000000007LL;
//vector<int> G[MAX_N];
//map<ull, ull> m;

struct Row: vector<double>{
    ll num;
    ll cost;
};

void solve() {
```

```
    ll m, n;
    cin >> m >> n;
    vector<Row> matr(m);
    set<ll> ans;

    fori(0, matr.size()) {
        Row& row = matr[i];
        row.num = i+1;
        row.resize(n);
        for(double &el : row) cin >> el;
        cin >> row.cost;
    }

    fori(0, n){
        ll min_cost = MAX_N + 1;
        ll min_idx;
        forj(i, m) {
            if(abs(matr[j][i]) > eps and matr[j].cost < min_cost){
                min_idx = j;
                min_cost = matr[j].cost;
            }
        }
        if(min_cost == MAX_N + 1){
            cout << -1 << '\n';
            exit(0);
        }

        ans.insert(matr[min_idx].num);

        swap(matr[i], matr[min_idx]);

        forj(i + 1, m) { // change all bottom rows
            double coefficient = matr[j][i] / matr[i][i]; // coefficient for current
                row
            foru(i, n) { // start from first not calc column
                matr[j][u] -= matr[i][u] * coefficient;
            }
        }
    }
    for (const ll& elem: ans)
        cout << elem << ' ';
}

signed main() {
#ifdef _ONPC_
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#define TIMEIT
    timeit
#endif // _ONPC_

    fastIO;
    int t(1);
    //cin >> t; // UNCOMMENT IF WITH TESTS //
    fori(0, t) solve();
    cout << '\n';
#ifdef TIMEIT
    endtimeit
#endif // TIMEIT
    return 0;
}
```

3 Консоль

```
C:/Users/egork/Desktop/space/cmake-build-debug/space.exe

3 3

1 0 2 3

1 0 2 4

2 0 1 2


-1
```

4 Тест производительности

Тест производительности представляет собой сравнение с наивным решением этой задачи. 20x20 40x40

```
C:/Users/egork/Desktop/space/cmake-build-debug/space.exe
20x20
Greed: 0.00042 sec
Naive: 0.00538 sec

40x40
Greed: 0.00341 sec
Naive: 19.38262 sec
```

На 20x20 время работы жадного алгоритма сравнимо с временем работы наивного алгоритма.

Однако, на 40x40 уравнений наивный алгоритм начинает катастрофически отставать по времени работы от жадного.

Сложность наивного алгоритма O(m $2^m * n^2$)

Также хочется отметить, что наивный алгоритм затрачивает гораздо больше памяти, чем жадный алгоритм, поскольку ему необходимо хранить исходную систему уравнений и обрабатываемую на текущем шаге подсистему.

5 Выводы

Стоит сразу отметить, что если глобальная оптимальность алгоритма имеет место практически всегда, его обычно предпочитают другим методам, таким как динамическое программирование. К тому же, очевидно, что ситуация, когда жадный алгоритм это такой алгоритм, который на каждом шаге делает локально наилучший выбор в надежде, что итоговое решение будет оптимальным – частая. Примечательно, что, в каком-то смысле, жадные алгоритмы – частный случай динамического программирования, за исключением того, что в жадном алгоритме выбор делается сразу – до решения подзадач –, а в динамическом программировании наоборот – выбор происходит после решения подзадач. В общем и целом жадные алгоритмы, как и динамическое программирование, упрощает жизнь практикам, которые эти алгоритмы придумывают, т.к. в основном такие алгоритмы довольно просты в реализации.

Список литературы

[1] *Поисковик - Google.*
URL: `https://www.google.com/`

[2] *Сайт с подробной документацией библиотек C++*
URL: `https://en.cppreference.com/`