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. Боглаев
Преподаватель:	А. А. Кухтичев
Группа:	М8О-306Б-22
Дата:	
Оценка:	
Подпись:	

Москва, 2024

Лабораторная работа №8

Задача: Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объем затрачиваемой памяти.

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат 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

$$c_1 a_1 + c_2 a_2 + \ + c_N a_N,$$

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 ($M <= N$) различных добавок. Нужно помочь Биологу подобрать самый дешевый наобор добавок, позволяющий найти коэффициенты ci. Возможно, соотношения веществ в добавках таковы, что определить коэффициенты нельзя.

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 порядку во входных данных). Если вариантов несколько, вывести какой-либо из них.

1 Описание

Согласно [1]: «Жадный алгоритм позволяет получить оптимальное решение задачи путем осуществления ряда выборов. В каждой точке принятия решения в алгоритме делается выбор, который в данный момент выглядит самым лучшим. Эта эвристическая стратегия не всегда дает оптимальное решение, но все же решение может оказаться и оптимальным.»

Свойство жадных алгоритмов: глобальное оптимальное решение можно получить, делая локальный оптимальный (жадный) выбор.

Процесс построения жадных алгоритмов:

1. Привести задачу оптимизации к виду, когда после сделанного выбора остается решить только одну подзадачу.

2. Доказать, что всегда существует такое оптимальное решение исходной задачи, которое можно получить путем жадного выбора, так что такой выбор всегда допустим.

3. Показать, что после жадного выбора остается подзадача, обладающая тем свойством, что объединение оптимального решения подзадачи со сделанным жадным выбором приводит к оптимальному решению исходной задачи.

Для решения задачи нам потребуется вспомнить теорию из курса Линейной алгебры. А точнее, что такое линейная независимость строк и столбцов матрицы, как приводить матрицу к ступенчатому виду и находить ее ранг.

Для нахождения коэффициентов в формуле для расчета воздейсвтвия добавки, необходимо составить матрицу из значений, поступающих на ввод программе. Нужно найти N коэффициентов, это говорит нам о том, что нужно найти ранг матрицы и проверить, равен ли он числу столбцов.

Ранг — это количество не нулевых строк матрицы. Если ранг не будет равен числу столбцов, значит найти коэффициенты нельзя.

Ранг можно найти путём приведения матрицы к ступенчатому виду. Еще одно определение ранга матрицы: это максимальное число ее линейно независимых строк(столбцов). Самое главное — ранг не может быть больше, чем $min(m, n)$.

Если расположить строки матрицы по возрастанию цены, сверху вниз, то можно найти набор добавок с наименьшей ценой.

Данный подход и будет являться жадным алгоритмом, так как ненулевые строки, составляющие ранг матрицы, будут иметь наименьшую цену. И вместо перебора всевозможных линейно независимых строк с наименьшей ценой, будет рассмотрено N первые линейно независимые строки.

2 Исходный код

Для реализации алгоритма напишем функцию *IsLinearIndependent* для поиска N линейно назавимых строк. Сначала матрица приводится к ступенчатому виду при помощи метода Гаусса, далее проверяется, равен ли ранг матрицы количеству столбцов и возвращается результат сравнения.

Чтобы не создавать дополнительных переменных и структур, увеличим количетсво столбцов матрицы для сохранения цену мешка добавок и номер строки во входный данных.

Чтобы расположить строки матрицы по возрастанию цены мешка добавки, отсортируем строки по цене, предворительно, написав функцию *ComporatorCost* для сравения векторов по предпоследнему элементу.

Если в мы смогли найти N линейно независимых строк в матрице, то нужно сохранить их исходные номера в вектор и отсортировать его.

Далее выводим отсортированный вектор.

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

int GetRank(std::vector<std::vector<double>> &matr) {
    int rows = matr.size();
    int cols = matr[0].size() - 2;
    int rank = 0;

    for (int col = 0; col < cols; col++) {
        int current_row = -1;
        for (int row = rank; row < rows; row++) {
            if (matr[row][col] != 0) {
                current_row = row;
                break;
            }
        }

        if (current_row == -1) {
            continue;
        }

        std::swap(matr[rank], matr[current_row]);

        for (int row = 0; row < rows; row++) {
            if (row != rank) {
                double factor = matr[row][col] / matr[rank][col];
                for (int j = col; j < cols; j++) {
                    matr[row][j] -= factor * matr[rank][j];
                }
            }
        }
```

```
        rank++;
    }

    return rank;
}

bool ComporatorCost(const std::vector<double> &a, const std::vector<double> &b) {
    return a[a.size() - 2] < b[b.size() - 2];
}

int main() {
    int m, n;
    std::cin >> m >> n;

    std::vector<std::vector<double>> matrix(m, std::vector<double>(n + 2));

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

    std::sort(matrix.begin(), matrix.end(), ComporatorCost);

    std::vector<int> selected_indices;
    std::vector<std::vector<double>> current_matrix;

    for (const auto &additive : matrix) {
        current_matrix.push_back(additive);
        int rank = GetRank(current_matrix);

        if (rank == int(selected_indices.size()) + 1) {
            selected_indices.push_back(additive.back() + 1);
        }

        if (int(selected_indices.size()) == n) {
            break;
        }
    }

    if (int(selected_indices.size()) == n) {
        std::sort(selected_indices.begin(), selected_indices.end());
        for (const int &index : selected_indices) {
            std::cout << index << " ";
        }
        std::cout << std::endl;
    } else {
        std::cout << -1 << std::endl;
```

```
    }

    return 0;
}
```

3 Консоль

```
alex@wega:~/$ cat tests/01.t
2 2
45 17 21
50 14 38
alex@wega:~/$ ./main <tests/01.t
1 2
alex@wega:~/$ cat tests/01.t
3 2
34 4 48
50 19 33
5 23 50
alex@wega:~/$ ./main <tests/01.t
1 2
alex@wega:~/$ cat tests/01.t
7 6
35 39 0 11 0 39 23
4 9 36 12 31 38 5
23 43 41 28 27 23 2
26 9 49 41 49 26 32
29 1 42 18 32 24 17
45 0 49 12 48 11 35
22 1 3 34 38 6 45
alex@wega:~/$ ./main <tests/01.t
1 2 3 4 5 6
alex@wega:~/$ cat tests/01.t
14 14
29 43 3 21 21 32 17 14 3 37 21 4 39 40 27
41 13 2 20 21 3 37 22 47 9 50 10 42 29 37
16 38 47 31 39 23 30 32 30 25 1 0 41 17 31
44 32 25 34 17 1 40 21 8 0 45 29 46 2 34
13 39 0 37 4 24 31 20 10 0 30 20 16 20 34
1 50 46 22 8 2 5 32 25 22 47 15 24 0 49
46 35 19 45 32 40 37 49 26 20 2 40 10 25 5
3 39 44 45 21 38 35 6 20 6 43 11 34 21 35
33 39 17 46 22 48 38 16 28 23 26 47 40 12 34
2 34 24 37 32 10 16 50 10 16 17 4 9 23 5
7 26 36 16 39 17 32 27 49 13 42 27 47 41 47
24 15 33 5 47 41 26 32 13 14 36 40 5 26 5
2 44 13 50 41 45 34 48 40 13 42 18 32 7 33
```

```
6 28 40 38 10 5 20 26 45 4 31 8 49 31 9
alex@wega:~/$ ./main <tests/01.t
1 2 3 4 5 6 7 8 9 10 11 12 13 14
alex@wega:~/$ cat tests/01.t
3 5
37 5 24 41 8 10
37 45 50 9 44 37
1 12 43 17 38 47
alex@wega:~/$ ./main <tests/01.t
-1
```

4 Тест производительности

Сравним алгоритм с наивным решение этой задачи.

Для тестирования возьмем наборы содержащие: 10 добавок, 15 добавок, 20 добавок, 25 добавок.

```
alex@wega:~/$ ./becnh.sh
Greedy: 0.008 ms
Naive: 1.554 ms
alex@wega:~/$ ./becnh.sh
Greedy: 0.013 ms
Naive: 1.448 ms
alex@wega:~/$ ./becnh.sh
Greedy: 0.025 ms
Naive: 1571.183 ms
alex@wega:~/$ ./becnh.sh
Greedy: 0.032 ms
Naive: 2802.912 ms
```

Можем заметить, что жадный алгоритм значительно быстрее наивного. Это пороисходит потому, что сложность наивного алгоритма — $O(2^m * n^3)$ (перебор всех комбинаций $O(2^m)$, метод Гаусса $O(n^3)$), а сложность жадного алгоритма — $O(m * n^2)$. Стоит отметить, что жадный алгоритм затрачивает меньше памяти, так как не хранит исходную систему уравнений.

5 Выводы

Выполнив восьмую лабораторную работу по курсу «Дискретный анализ», я ознакомился с классическими задачами, которые решаются при помощи жадных алгоритмов, решил задачу для своего варианта, используя жадный алгоритм.

Жадные алгоритмы делают выбор на основе локальной оптимальности и заранее определенных правил, что может привести к оптимальному решению в некоторых случаях, но не всегда. Динамическое программирование, в свою очередь, рассматривает все возможные варианты и гарантирует нахождение оптимального решения, что делает его более универсальным, но и более сложным в реализации.

Жадные алгоритмы могут эффективно решать определенные классы задач, обеспечивая быстрое получение решений. Благодаря своей простоте и скорости, жадные алгоритмы остаются важным инструментом в арсенале алгоритмических решений, особенно в задачах, где требуется быстрое приближение к оптимальному решению.

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] *Жадные алгоритмы — Яндекс образование*
URL: `https://education.yandex.ru/handbook/algorithms/article/zhadnye-algoritmy`
(дата обращения: 12.11.2024).