Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Дискретный анализ" №8

Студент: Шипилов К. Ю.

Преподаватель: Макаров Н. К.

Группа: М8О-303Б-22

Дата: 02.11.2024

Оценка:

Подпись:

Москва 2024 г.

Оглавление

Постановка задачи

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 $c_1a_1 + c_2a_2 + \cdot\cdot\cdot + c_{NaN}$, 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 ($M \leqslant N$) 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.

Формат ввода

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 вывода

Вывести -1 если определить коэффциенты невозможно, иначе набор добавок (их номеров по порядоку во входных данных). Если вариантов несколько, вывести какой-либо из них.

Алгоритм решения

В задаче дана система из M уравнений с N переменных. Биолог может для каждого уравнения определить свободный член. Для того, чтобы у уравнения с N переменных существовало единственное решение необходимо N линейно независимых уравнений. Линейную независимость уравнений можно проверить, используя метод Гаусса, если записать систему в виде матрицы и привести ее к ступенчатому виду.

Так как необходимо минимизировать стоимость добавок, то при приведении матрицы к ступенчатому виду необходимо поднимать наверх те строки, у которых в столбце, соответствующем текущей переменной, ненулевой коэффициент, а цена минимальная.

Сначала матрица сохраняется в двумерный массив. Также в конце каждой строки сохраняется цена добавки и ее номер.

В цикле по всем компонентам добавок, в столбце с текущим компонентом добавки ищется ненулевой элемент, с минимальной ценой. Если такой элемент не найден, программа завершает свою работу с выводом -1, так как невозможно определить коэффициент этого вещества.

Такой компонент найден, то строка, в которой он содержится меняется со строкой, с индексом текущей итерации цикла. И из всех строк ниже строки с этим индексом вычитается найденная строка так, чтобы в столбце с текущим компонентом остались только 0.

После того как сделано N итераций, N первых строк матрицы соответствуют системе из N линейно независимых уравнений с N переменными, и общая цена добавок, соответствующих этим уравнениям, минимальна. Номера добавок из N первых строк сортируются в порядке возрастания и выводятся.

Исходный код

main.cpp

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


std::vector<std::vector<double>> inputAdditivesMatrix(size_t n,
size_t m) {
  std::vector<std::vector<double>> additives(n,
std::vector<double>(m + 2, 0));
  for (size_t i = 0; i < n; ++i) {
    for (size_t j = 0; j < m + 1; ++j) {
      std::cin >> additives[i][j];
    }
    additives[i][m + 1] = i + 1;
  }
  return additives;
}


int64_t findComponentMinPrice(size_t startRow, size_t
component,
                              std::vector<std::vector<double>>
additives,
                              size_t priceColumn) {
  double minPrice = 51;
  int64_t minPriceIndex = -1;
  for (size_t i = startRow; i < additives.size(); ++i) {
    if (additives[i][component] != 0 && additives[i]
[priceColumn] < minPrice) {
      minPrice = additives[i][priceColumn];
      minPriceIndex = i;
    }
```

```
  }
  return minPriceIndex;
}

void substractRow(const std::vector<double>& row,
                  std::vector<std::vector<double>>& additives,
size_t startRow,
                  size_t startColumn, size_t componentsCount) {
  for (size_t i = startRow; i < additives.size(); ++i) {
    double coef = additives[i][startColumn] / row[startColumn];
    for (size_t j = startColumn; j < componentsCount; ++j) {
      additives[i][j] -= row[j] * coef;
    }
  }
}

bool gaussianElimination(std::vector<std::vector<double>>&
additives, size_t n,
                         size_t m) {
  size_t j = 0;
  for (size_t i = 0; i < n; ++i) {
    int64_t rowIndex = findComponentMinPrice(i, j, additives,
m);
    if (rowIndex < 0) {
      break;
    }
    std::swap(additives[i], additives[rowIndex]);
    substractRow(additives[i], additives, i + 1, j, m);
```

```
    if (++j >= m) {
      return true;
    }
  }
  return false;
}

std::vector<size_t> countingSort(const std::vector<size_t>&
numbers,
                                 size_t maxKey) {
  std::vector<size_t> sortedNumbers(numbers.size(), 0);
  std::vector<size_t> countingArray(maxKey + 1, 0);
  for (size_t i = 0; i < numbers.size(); ++i) {
    ++countingArray[numbers[i]];
  }
  for (size_t i = 1; i <= maxKey; ++i) {
    countingArray[i] += countingArray[i - 1];
  }
  for (int i = numbers.size() - 1; i >= 0; --i) {
    sortedNumbers[countingArray[numbers[i]] - 1] = numbers[i];
    --countingArray[numbers[i]];
  }
  return sortedNumbers;
}

int main() {
  size_t N, M;
  std::cin >> M >> N;
```

```
  std::vector<std::vector<double>> additives =
inputAdditivesMatrix(M, N);
  if (!gaussianElimination(additives, M, N)) {
    std::cout << -1 << std::endl;
  } else {
    std::vector<size_t> additivesNumbers;
    for (size_t i = 0; i < N; ++i) {

additivesNumbers.push_back(static_cast<size_t>(additives[i][N +
1]));
    }
    for (size_t number : countingSort(additivesNumbers, M)) {
      std::cout << number << " ";
    }
    std::cout << std::endl;
  }
}
```

Тесты

Test 1

Input	Output
3 3 1 0 2 3 1 0 2 4 2 0 1 2	-1

Test 2

Input	Output
4 3 1 1 1 5 1 0 0 2 0 1 0 3 0 0 1 4	2 3 4

Test 3

Input	Output
4 2 1 1 5 1 1 4 1 1 6 1 1 3	-1

Test 4

Input	Output
2 3 1 0 1 5 1 2 3 4 -1	-1

Вывод

Выполняя лабораторную работу я изучил жадные алгоритмы и их применение к задачам оптимизации. Жадные алгоритмы представляют собой эффективный метод решения, когда оптимальное решение задачи можно построить, выбирая

на каждом этапе локально оптимальный вариант. Эти алгоритмы просты, эффективны и часто дают оптимальный результат, но требуют анализа применимости к каждой конкретной задаче.