Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Программирование на языках высокого уровня

Тема: Венгерский алгоритм

Выполнил
студент гр. 3331506/70401 Ляпцев И.А.
Преподаватель Ананьевский М. С.

« »________2020 г.

Санкт-Петербург

2020 г.

Оглавление

1. Введение

1.1 Формулировка задачи, которую решает алгоритм

В комбинаторике достаточно часто возникает необходимость решить задачу о назначениях- распределить исполнителей с неодинаковыми затратами на определенные работы с минимальными затратами. Существует несколько способов решения этой задачи, один из них я рассмотрю в этой работе.

В данной работе рассмотрен Венгерский алгоритм, приведена его реализация на языке программирования, произведена оценка сложности и численный анализ алгоритма, описано его применение.

Венгерский алгоритм - алгоритм оптимизации, решающий описанную выше задачу за полиномиальное время. Разработан Гарольдом Куком в 1955, название алгоритма выбрано из-за того, что автор во многом основывался на работах двух венгерских математиков (Кёнига и Эгервари).

1.2 Словесное описание алгоритма

Дана неотрицательная матрица C размера $n \times n$, где элемент в i-й строке и j-м столбце соответствует стоимости выполнения j-го вида работ i-м работником. Нужно найти такое соответствие работ работникам, чтобы расходы на оплату труда были наименьшими.

1-й шаг. Цель данного шага — получение максимально возможного числа нулевых элементов в матрице C. Для этого из всех элементов каждой строки вычитаем минимальный элемент соответствующей строки, а из всех элементов каждого столбца вычитаем минимальный элемент соответствующего столбца.

2-й шаг. Если после выполнения 1-го шага в каждой строке и каждом столбце матрицы C можно выбрать по одному нулевому элементу, то полученное решение будет оптимальным назначением.

3-й шаг. Если допустимое решение, состоящее из нулей, не найдено, то проводим минимальное число прямых через некоторые столбцы и строки так, чтобы все нули оказались вычеркнутыми. Выбираем наименьший невычеркнутый элемент. Этот элемент вычитаем из каждого невычеркнутого элемента и прибавляем к каждому элементу, стоящему на пересечении проведенных прямых.

Если после проведения 3-го шага оптимальное решение не достигнуто, то процедуру проведения прямых следует повторять до тех пор, пока не будет получено допустимое решение.

2. Реализация алгоритма

Алгоритм был реализован на языке C++.

Функция, реализующая Венгерский алгоритм, представлена на рисунках 1-2.

```cpp
/*
 * Решает задачу о назначениях Венгерским методом.
 * matrix: прямоугольная матрица из целых чисел (не обязательно положительных).
 *         Высота матрицы должна быть не больше ширины.
 * Возвращает: Список выбранных элементов, по одному из каждой строки матрицы.
 */
VPInt hungarian(const VVInt& matrix) {

    // Размеры матрицы
    int height = matrix.size();
    int width = matrix[0].size();

    // Значения, вычитаемые из строк (u) и столбцов (v)
    VInt u(height, 0), v(width, 0);

    // Индекс помеченной клетки в каждом столбце
    VInt markIndices(width, -1);

    // Будем добавлять строки матрицы одну за другой
    for (int i = 0; i < height; i++) {
        VInt links(width, -1);
        VInt mins(width, inf);
        VInt visited(width, 0);

        // Разрешение коллизий (создание "чередующейся цепочки" из нулевых элементов)
        int markedI = i, markedJ = -1, j;
        while (markedI != -1) {
            // Обновим информацию о минимумах в посещенных строках непосещенных столбцов
            // Заодно поместим в j индекс непосещенного столбца с самым маленьким из них
            j = -1;
            for (int j1 = 0; j1 < width; j1++)
                if (!visited[j1]) {
                    if (matrix[markedI][j1] - u[markedI] - v[j1] < mins[j1]) {
                        mins[j1] = matrix[markedI][j1] - u[markedI] - v[j1];
                        links[j1] = markedJ;
                    }
                    if (j == -1 || mins[j1] < mins[j])
                        j = j1;
                }
```

Рисунок 1 –Венгерский алгоритм, часть 1

```
            // Теперь нас интересует элемент с индексами (markIndices[links[j]], j)
            // Произведем манипуляции со строками и столбцами так, чтобы он обнулился
            int delta = mins[j];
            for (int j1 = 0; j1 < width; j1++)
                if (visited[j1]) {
                    u[markIndices[j1]] += delta;
                    v[j1] -= delta;
                }
                else {
                    mins[j1] -= delta;
                }
            u[i] += delta;

            // Если коллизия не разрешена - перейдем к следующей итерации
            visited[j] = 1;
            markedJ = j;
            markedI = markIndices[j];
        }

        // Пройдем по найденной чередующейся цепочке клеток, снимем отметки с
        // отмеченных клеток и поставим отметки на неотмеченные
        for (; links[j] != -1; j = links[j])
            markIndices[j] = markIndices[links[j]];
        markIndices[j] = i;
    }

    // Вернем результат в естественной форме
    VPInt result;
    for (int j = 0; j < width; j++)
        if (markIndices[j] != -1)
            result.push_back(PInt(markIndices[j], j));
    return result;
}
```

Рисунок 2 –Венгерский алгоритм, часть 2

3. Анализ алгоритма

3.1 Анализ сложности алгоритма

Во внешнем цикле мы добавляем в рассмотрение строки матрицы одну за другой. Каждая строка обрабатывается за время $O(n^2)$, поскольку при этом могло происходить лишь $O(n)$ пересчётов потенциала (каждый — за время $O(n)$, для чего за время $O(n^2)$ поддерживается массив $mins[]$; суммарно алгоритм отработает за время $O(n^2)$ (поскольку он представлен в форме $O(n)$ итераций, на каждой из которых посещается новый столбец).

Итоговая асимптотика составляет $O(n^3)$.

3.2 Численный анализ алгоритма

Посчитаем время нахождения решения для массивов разных размеров, пусть число работ и работников одинаково.

На рисунке 3 приведен график зависимости времени выполнения от количества элементов (время указано в мс).

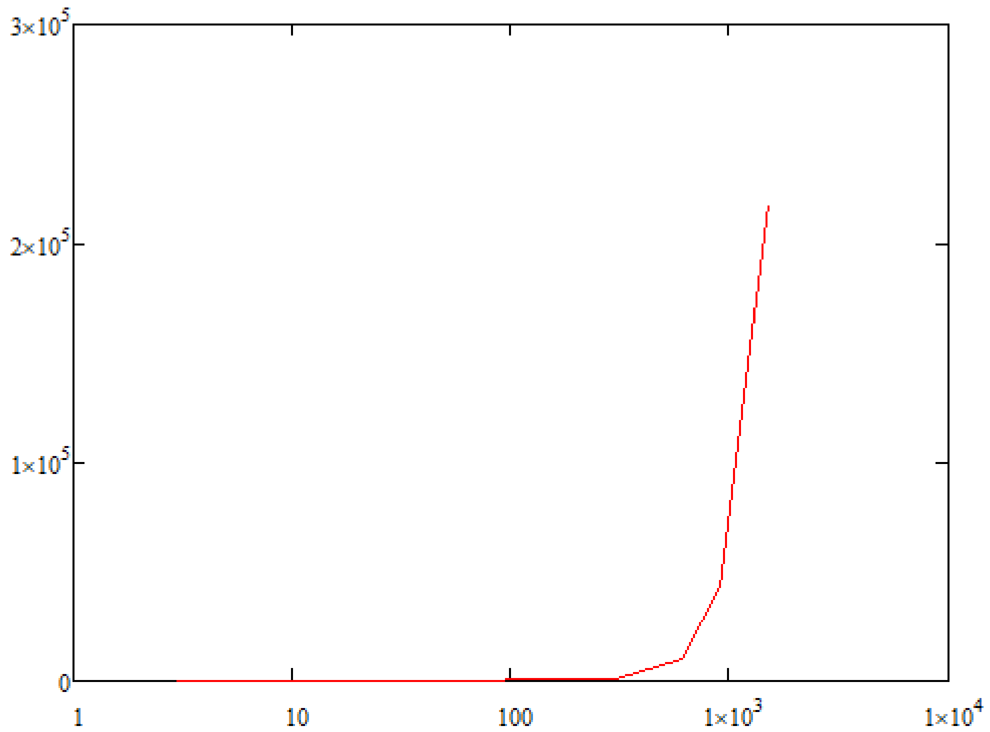

Рисунок 3 – Численный анализ

Численные значения приведены в таблице 1.

Таблица 1 – Численный анализ

Число работ	3	30	300	600	900	1500
Время выполнения, мс	<1	1	471	9680	42660	217324

Как видно из графика, даже для третьего случая (по 300 работ и 300 работников) время приблизительно равно 1 мс. С последующим увеличением количества элементов время начинает резко возрастать. Таким образом, для задачи с 1500 элементами время выполнения составляет уже 3,5 минут. Такие затраты времени неприемлемы. Для генерации перестановок с большим количеством элементов потребуются более производительные мощности.

4. Применение алгоритма

Основным применением Венгерского алгоритма является подсчет самого оптимального назначения.

На рисунке 4 представлена функция, которая реализует это применение.

```
void hungarian_app(VVInt array, int n, int m)
{
    VPInt Matrix = hungarian(array);
    for (int j = 0; j < n; j++) // Цикл, который идёт по элементам
        cout << "Job number" << (Matrix[j].second + 1) << ' ' << "performs workers number" << ' ' << (Matrix[j].first + 1) << ' ' << endl;
    cout << endl;
}
```

Рисунок 4 – Применение Венгерского алгоритма

Таблица 2 – Пример задачи о назначениях

	A	B	C
Иван	10.000 руб.	20.000 руб.	30.000 руб.
Пётр	30.000 руб.	30.000 руб.	30.000 руб.
Андрей	30.000 руб.	30.000 руб.	20.000 руб.

Результат выполнения данной функции для таблицы 2 представлен на рисунке 5.

```
Input number of jobs and then number of workers
3
3
10
20
30
30
30
30
30
30
20
Job number 1 performs workers number 1
Job number 2 performs workers number 2
Job number 3 performs workers number 3
```

Рисунок 4 – Результат

5. Заключение

В ходе выполнения работы было рассмотрены принцип работы Венгерского алгоритма, его реализация на языке программирования C++, проведен анализ сложности и численный анализ алгоритма, а также рассмотрено его применение для решения определенных задач.

Таким образом, Венгерский алгоритм является очень простым и эффективным при решении задачи о подстановках за полиномиальное время.

Список литературы

1. Harold W. Kuhn, «The Hungarian Method for the assignment problem», *Naval Research Logistics Quarterly*, **2**:83—97, 1955.
2. Harold W. Kuhn, «Variants of the Hungarian method for assignment problems», *Naval Research Logistics Quarterly*, **3**: 253—258, 1956.