Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентрованное программирование»

Выполнил
студент группы 3331506/80401 _____ Э. Ф. Хуснутдинова

Руководитель _____ Е. М. Кузнецова

«__»_____2021 г

Санкт-Петербург
2021

Оглавление

1 ВВЕДЕНИЕ

В работе будет рассмотрен венгерский алгоритм. Это алгоритм оптимизации, решающий задачу о назначениях за полиномиальное время. Задача о назначениях – это задача о наилучшем распределении некоторого числа работ между таким же числом исполнителей. При ее решении ищут оптимальное назначение из условия максимума общей производительности, которая равна сумме производительности, которая равна сумме производительности исполнителей.

Случаи, в которых применим этот алгоритм:

- Задача о назначении работников на должности;
- Назначение машин на производственные секции;
- Выбор кандидатов на разные вакансии по оценкам.

2 ИДЕЯ АЛГОРИТМА

Задача:

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.

Алгоритм:

- Вычитаем из каждой строки значение ее минимального элемента. Соответственно теперь в каждой строке есть хотя бы один нулевой элемент.
- Вычитаем из каждого столбца значение его минимального элемента. Соответственно теперь в каждом столбце есть хотя бы один нулевой элемент.
- Ищем в каждой строке и каждом столбце матрицы C по одному нулевому элементу:
    - Если они найдены, то полученное решение будет оптимальным назначением, алгоритм закончен.
    - Если допустимое решение не найдено, то проводим минимальное число прямых через некоторые столбцы и строки так, чтобы все нули оказались вычеркнутыми. Выбираем наименьший невычеркнутый элемент и прибавляем к каждому элементу, стоящему на пересечении проведенных прямых. Повторять до тех пор, пока не получим допустимое решение.

3 ЭФФЕКТИВНОСТЬ

Оценим время работы 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 $O(n^3)$.

4 РЕЗУЛЬТАТЫ РАБОТЫ АЛГОРИТМА

В качестве входных данных будем подавать массивы разных размеров, число работ и работников одинаково. Замерим время выполнения для каждого случая (результаты представлены в таблице 1).

Таблица 1 – Результаты измерения

Число работ	3	30	300	600	900	1500	3000
Время выполнения, нс	11100	237500	137760300	2278707800	8153355500	58113184400	364265281700

По таблице 1 построен график зависимости времени выполнения от количества элементов (рисунок 4.1).

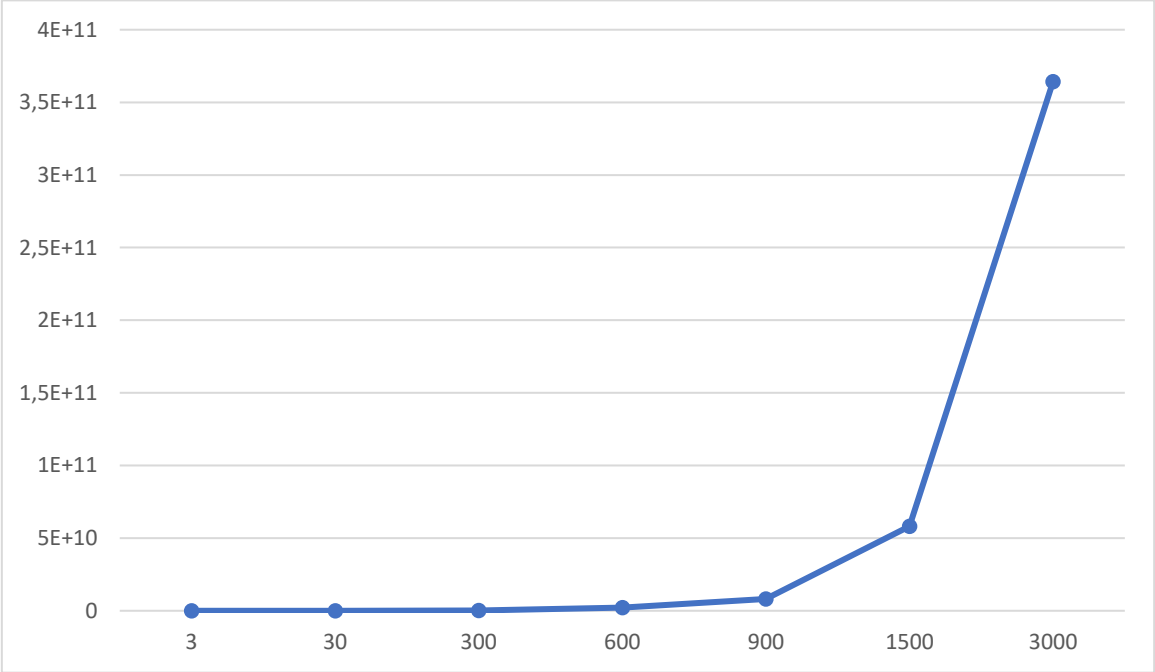

Рисунок 4.1 – График зависимости времени от количества элементов

5 СПИСОК ЛИТЕРАТУРЫ

1) Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е. — М.: Вильямс, 2005. — 1296 с.
2) Готтшлинг П. Современный C++ для программистов, инженеров и ученых. Серия «C++ In-Depth» = Discovering Modern C++: A Concise Introduction for Scientists and Engineers (C++ In-Depth). — М.: Вильямс, 2016. — 512 с.
3) Вирт Н. Алгоритмы и структуры данных — М.: Мир, 1989. — 360 с.

6 ПРИЛОЖЕНИЕ

Приложение 1

```
#include <iostream>
#include <vector>
#include <limits>
#include <stdlib.h>
#include <chrono>
using namespace std;

typedef pair<int, int> PInt;
typedef vector<int> VInt;
typedef vector<VInt> VVInt;
typedef vector<PInt> VPInt;

const int inf = numeric_limits<int>::max();

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);

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

        // Создание "чередующейся цепочки" из нулевых элементов
        int markedI = i, markedJ = -1, j;
        while (markedI != -1) {
            // Обновим информацию о минимумах в посещенных строках
непосещенных столбцов
            // 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;
                }

            //Обнуляем элемент с индексами (markIndices[links[j]], j)
            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;

            // Переход 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;
    }

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

void hungarian_app(VVInt array, int n, int m)
{
    VPInt Matrix = hungarian(array);
}

int main()
{
    int array[] = { 3, 30, 300, 600, 900, 1500,3000};

    for (int f = 0; f < 7; f++)
    {
        int Size = array[f];
        vector < vector <int> > a(Size, vector <int>(Size));
        VVInt NewMatrix = a;
        for (int i = 0; i < Size; i++)      // Цикл, который идёт по строкам
        {
            for (int j = 0; j < Size; j++) // Цикл, который идёт по элементам
            {
                NewMatrix[i][j] = rand() % 100;; // Заполнение вектора или
массива (в данном случае ввод)
            }
        }
        auto begin = std::chrono::steady_clock::now();
        hungarian_app(NewMatrix, Size, Size);
        auto end = std::chrono::steady_clock::now();
        auto elapsed_ms =
std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin);
        std::cout << "The time: " << elapsed_ms.count() << " ns\n";
    }
    return 0;
}
```