Жуковский Павел 3 курс 12 группа

Лабораторная работа №3

Вариант 8

Описание задачи

Задача 8. Станки

Конвейер состоит из N различных станков. Есть N рабочих. Известна матрица C размера N × N, где элемент Cij задаёт производительность i-го рабочего на j-м станке. Необходимо определить, каким должно быть распределение рабочих по станкам (каждый рабочий может быть назначен только на один станок, на каждом станке может работать только один рабочий), чтобы производительность всего конвейера была максимальной. Производительность конвейера при некотором распределении рабочих по станкам равна минимальной производительности рабочих на назначенных им на конвейере станках. Если решение не единственно, вывести решение, первое в лексикографическом порядке среди всех решений.

Формат входных данных

Первая строка содержит число рабочих (станков) N ($1 \le N \le 500$). Затем идут N строк файла, которые задают матрицу C производительностей ($1 \le Cij \le 1000$).

Формат выходных данных

В первой строке выведите максимальную возможную производительность конвейера. Во второй строке – номера станков, на которые должны быть распределены рабочие 1, 2, …, N соответственно.

Решение

Если перевести нашу задачу на язык теории графов, то мы имеем двудольный граф, где каждая вершина из одной доли (рабочие) является смежной для каждой вершины из другой доли (станки). Наша задача – найти такое полное паросочетание этого графа (т.е. множество ребёр, покрывающее все вершины), чтобы его суммарный вес был минимален.

Данную задачу можно свести к классу задач, которые известны под названием «задачи о назначениях». Есть несколько алгоритмов, которые решают данную задачу, но одним из самых популярных является алгоритм Куна или «венгерский алгоритм». Данный алгоритм работает в худшем случае за $O(n^4)$, хотя на сегодняшний день существует его модификация, которая работает за $O(n^3)$.

Общая пошаговая идея алгоритма:

1) Среди всех элементов матрицы C находим максимальным элемент

2) Умножаем все элементы нашей матрицы C на (-1), нам нужно так поступить, так как в нашей задаче мы ищем максимальную эффективность (в оригинальном алгоритме акцент был на минимальную стоимость, т.е. мы должны построить матрицу, эквивалентную исходной)

3) Прибавляем ко всем элементам матрицы модуль максимального, чтобы полностью избавиться от отрицательных элементов и получить эквивалентную матрицу (по итогу на месте максимального по модулю элемента (или элементов) появятся нули).

4) Пробуем выбрать на всей матрице N нулей так, чтобы при проведении через все нули линий по горизонтали и вертикали ни одна линия не пересекалась. Если получилось выбрать N нулей таким образом, значит, мы нашли оптимальное решение нашей задачи. Каждому работнику достаточно выбрать в соответствие станок, который соответствует выбранному нулю (у нас N нулей и соответственно будет N выбранных клеток для каждого работника и его станка).

5) Обычно, на этом этапе не всегда возможно выбрать N нулей так, чтобы линии, проведённые через нули по горизонтали и вертикали, не пересекались (далее будем говорить, чтобы «нули не пересекались»), поэтому редуцируем всю матрицу по строкам (т.е. в каждой строке находим минимальный элемент и отнимаем его от всех остальных

элементов в строке, чтобы в строке появился по крайней мере один нуль). Возвращаемся к шагу 4 и пробуем найти решение.

6) Если после шага 5) также невозможно выбрать N непересекающихся нулей, то проводим операцию редуцирования матрицы снова, но уже не со строками, а со столбцами (таким образом в каждом столбце появится по крайней мере один нуль). Снова возвращаемся к шагу 4) и пробуем найти решение.

7) Бывает и так, что редуцирования по строкам и по столбцам недостаточно, чтобы стало возможным выбрать N непересекающихся нулей. В этом случае определяем, каково наименьшее количество вычеркиваний, которое покрывает все имеющиеся нули и выбираем данный способ вычёркивания (т.е. выбираем все элементы, которые покрываются этими вычеркиваниями). После этого у нас останутся какие-то невыбранные ненулевые элементы, среди них мы определяем наименьший и отнимаем от всех остальных. Таким образом, у нас появится по крайней мере один новый ноль. Далее возвращаемся к шагу 4) – выбираем ноль (нули) так, чтобы покрыть вычеркнутыми линиями всю матрицу. Если это снова невозможно, то повторяем этот шаг, пока это не станет возможным. (а это, согласно алгоритму, когда-нибудь должно стать возможным).

Реализация

В этой задаче нам важна скорость и время выполнения, поэтому в качестве инструмента для реализации я выбрал язык программирования C++. Ещё одной причиной выбора данного языка стало наличие контейнеров SQL, с которым удобно работать при решении подобных задач. Так, я задействовал в своей программе такой контейнер, как vector.

В матрице gg будем хранить матрицу работников и их эффективности на станках (она размера 502x502, т.е. с запасом, чтобы у нас не было проблем с переполнением во время выполнения).

Все остальные контейнеры нужны для работы алгоритма.

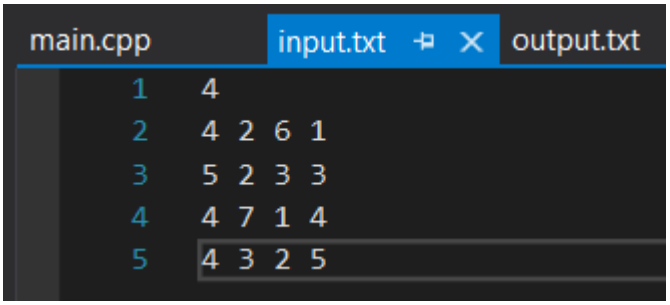
```cpp
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

ifstream fin("input.txt");
ofstream fout("output.txt");

int n;
int gg[502][502];
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;
int border;
vector<bool> block;
int e[1003][1003];
int ans[503];
int parent[1003];

bool try_kuhn(int v)
{
    if (used[v])  return false;
    used[v] = true;
    for (unsigned int i = 0; i < g[v].size(); i++)
    {
        int to = g[v][i];
        if (mt[to] == -1 || try_kuhn(mt[to]))
        {
            mt[to] = v;
            return true;
        }
    }
    return false;
}

bool f(int m)
{
    g.clear();

    for (int i = 0; i < n; i++)
    {
        vector<int> cur;
        for (int j = 0; j < n; j++)
        {
            if (gg[i][j] >= m) cur.push_back(j);
        }
        g.push_back(cur);
    }

    mt.assign(n, -1);
    for (int v = 0; v < n; ++v)
    {
        used.assign(n, false);
        try_kuhn(v);
    }
    int ans = 0;
    for (int i = 0; i < n; ++i)
        if (mt[i] != -1) ans++;
```

```
    if (ans == n)
    {
        return true;
    }
    else
    {
        return false;
    }
}

void dfs(int v, int p)
{
    if (used[v]) return;
    used[v] = true;
    parent[v] = p;
    for (int i = 0; i < 2 * n; i++)
    {
        if (e[v][i] == 1 && !block[i]) dfs(i, v);
    }
}

void findLex(int v)
{
    for (int i = 0; i < n; i++) e[i][ans[i] + n] = 1;
    used.assign(2 * n, false);
    for (int i = 0; i < n; i++) parent[i] = -1;
    dfs(v, v);
    for (int i = 0; i < n; i++) e[i][ans[i] + n] = 0;
    for (int i = n; i < 2 * n; i++)
    {
        if (used[i] && !block[i] && gg[v][i - n] >= border)
        {
            block[i] = true;
            ans[v] = i - n;
            int u = parent[i];
            while (u != parent[u])
            {
                ans[u] = parent[u] - n;
                u = parent[parent[u]];
            }
            break;
        }
    }
    block[v] = true;
}

void lex()
{
    block.assign(2 * n, false);
    for (int i = 0; i < n; i++) ans[mt[i]] = i;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (gg[i][j] >= border) e[n + j][i] = 1;
        }
    }
    for (int i = 0; i < n; i++)
    {
```

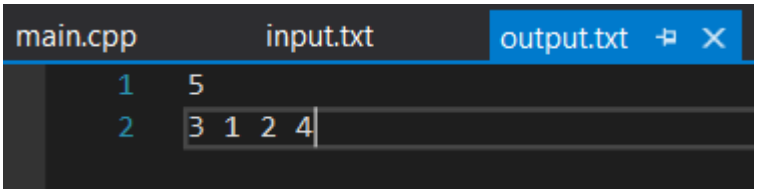
```
            findLex(i);
        }
        for (int i = 0; i < n - 1; i++) fout << ans[i] + 1 << " ";
        fout << ans[n - 1] + 1;
}

int main()
{
    fin >> n;
    int l = 1, r = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            fin >> gg[i][j];
            if (gg[i][j] > r) r = gg[i][j];
        }
    }
    while (r - l > 1)
    {
        int mid = (l + r) / 2;
        if (f(mid))
        {
            l = mid;
        }
        else
        {
            r = mid;
        }
    }
    if (f(r))
    {
        border = r;
    }
    else
    {
        f(l);
        border = l;
    }
    fout << border << endl;
    lex();
      }
```

Результаты

На следующих входных данных (из условия):



Программа выдала следующие результаты:



Выводы

Исходя из результатов, полученных программой, можно сделать вывод, что при N = 4 работниках и N = 4 станках и следующей матрице C:

$$C = \begin{bmatrix} 4 & 2 & 6 & 1 \\ 5 & 2 & 3 & 3 \\ 4 & 7 & 1 & 4 \\ 4 & 3 & 2 & 5 \end{bmatrix}$$

Самым оптимальным решением будет:

1) Поставить 1-ого работника на 3-ий станок (его эффективность будет равна 6)

2) Поставить 2-ого работника на 1-ый станок (его эффективность будет равна 5)

3) Поставить 3-его работника на 2-ой станок (его эффективность будет равна 7)

4) Поставить 4-ого работника на 4-ый станок (его эффективность будет равна 5)

При этом производительность всего конвейера будет равна 5, т.к. минимальным значением эффективности среди 6, 5, 6, 5 действительно является значение 5.