МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

(ФГБОУ ВО «ВятГУ»)

Факультет автоматики и вычислительной техники

Кафедра ЭВМ

Отчёт

Лабораторная работа № 4 по дисциплине

«Исследование операций»

Выполнил студент группы ИВТб-2301-04-00 / Жеребцов К. А.

Проверил преподаватель / Коржавина А. С.

Киров 2022

1. Цель работы: закрепить на практике знания о методах решения транспортных задач линейного программирования и получить навыки их программной реализации.
2. Задание: написать программу, решающую задачу о назначениях венгерским методом.
3. Описание алгоритмов:

Для {\displaystyle n} работников и работ, дана матрица *n*×*n*, задающая стоимость выполнения каждой работы каждым работником. Найти минимальную стоимость выполнения работ, такую что каждый работник выполняет ровно одну работу, а каждую работу выполняет ровно один работник.

В дальнейшем мы под **назначением** понимаем соответствие между работниками и работами, имеющее нулевую стоимость, после того как мы произвели трансформации, влияющие лишь на общую стоимость работ.

**Шаг 1**

Уменьшаем элементы построчно. Находим наименьший из элементов первой строки (а1, а2, а3, а4), и вычитаем его из всех элементов первой строки. При этом хотя бы один из элементов первой строки обнулится. То же самое выполняем и для всех остальных строк. Теперь в каждой строке матрицы есть хотя бы один ноль. Иногда нулей уже достаточно, чтобы найти назначение. Пример показан в таблице. Красные нули обозначают назначенные работы.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a2' | 0 | a4' |
| b1' | b2' | b3' | 0 |
| 0 | c2' | c3' | c4' |
| d1' | 0 | d3' | d4' |

При большом количестве нулей для поиска назначения (нулевой стоимости) можно использовать алгоритм нахождения максимального паросочетания двудольных графов, например, алгоритм Хопкрофта — Карпа. Кроме того, если хотя бы в одном столбце нет нулевых элементов, то назначение невозможно.

**Шаг 2**

Часто на первом шаге нет назначения, как, например, в следующем случае:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a2' | a3' | a4' |
| b1' | b2' | b3' | 0 |
| 0 | c2' | c3' | c4' |
| d1' | 0 | d3' | d4' |

Задача 1 может быть эффективно (за нулевую стоимость) выполнена как работником a, так и работником c, зато задача 3 не может быть эффективно выполнена никем.

В таких случаях мы повторяем шаг 1 для столбцов и вновь проверяем, возможно ли назначение.

**Шаг 3**

Во многих случаях мы достигнем желаемого результата уже после шага 2. Но иногда это не так, например:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a2' | a3' | a4' |
| b1' | b2' | b3' | 0 |
| 0 | c2' | c3' | c4' |
| d1' | 0 | 0 | d4' |

Если работник d выполняет работу 2, некому выполнять работу 3, и наоборот.

В таких случаях мы выполняем процедуру, описанную ниже.

Сначала, используя любой алгоритм поиска максимального паросочетания в двудольном графе, назначаем как можно больше работ тем работникам, которые могут их выполнить за нулевую стоимость. Пример показан в таблице, назначенные работы выделены красным.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | a2' | a3' | a4' |
| b1' | b2' | b3' | 0 |
| 0 | c2' | c3' | c4' |
| d1' | 0 | 0 | d4' |

Отметим все строки без назначений (строка 1). Отметим все столбцы с нулями в этих строках (столбец 1). Отметим все строки с «красными» нулями в этих столбцах (строка 3). Продолжаем, пока новые строки и столбцы не перестали отмечаться.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| × |  |  |  |  |
| 0 | a2' | a3' | a4' | × |
| b1' | b2' | b3' | 0 |  |
| 0 | c2' | c3' | c4' | × |
| d1' | 0 | 0 | d4' |  |

Теперь проводим линии через все отмеченные столбцы и **неотмеченные** строки.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| × |  |  |  |  |
| 0 | a2' | a3' | a4' | × |
| b1' | b2' | b3' | 0 |  |
| 0 | c2' | c3' | c4' | × |
| d1' | 0 | 0 | d4' |  |

Все эти действия преследовали лишь одну цель: провести наименьшее количество линий (вертикалей и горизонталей) так, чтобы покрыть все нули. Можно было воспользоваться любым другим методом вместо описанного.

**Шаг 4**

Из непокрытых линиями элементов матрицы (в данном случае это a2', a3', a4', c2', c3', c4') найти наименьший. Вычесть его из всех не отмеченных строк и прибавить ко всем пересечениям отмеченных строк и столбцов. Так, например, если наименьший элемент из перечисленных равен а2', мы получим

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| × |  |  |  |  |
| 0 | 0 | a3'-а2' | a4'-a2' | × |
| b1'+a2' | b2' | b3' | 0 |  |
| 0 | c2'-а2' | c3'-а2' | c4'-а2' | × |
| d1'+a2' | 0 | 0 | d4' |  |

Повторять процедуру (шаги 1-4) до тех пор, пока назначение не станет возможным.

1. Листинг программы:

#include <iostream>

#include <fstream>

#include <string>

#include <algorithm>

#include <vector>

using namespace std;

using std::vector;

const int INF = 1000 \* 1000 \* 1000;

const int inf = numeric\_limits<int>::max();

typedef pair<int, int> PInt;

typedef vector<int> VInt;

typedef vector<VInt> VVInt;

typedef vector<PInt> VPInt;

void printMatrVector(VVInt const& mat)

{

for (VInt row : mat)

{

for (int val : row) {

cout << "|" << val << "\t";

}

cout << "\n";

}

cout << "\n";

}

void printVector(VPInt const& v)

{

cout << "|Str\t|Col\n";

for (size\_t i = 0; i < size(v); i++)

{

cout << "|" << v[i].first << "\t|" << v[i].second << "\n";

}

cout << "\n";

}

VPInt hungarian(const VVInt& matrix)

{

cout << "Start: \n";

int height = matrix.size(), width = matrix[0].size();

VInt u(height, 0), v(width, 0);

VInt markIndices(width, -1);

for (int i = 0; i < height; i++)

{

VInt links(width, -1);

VInt mins(width, inf);

VInt visited(width, 0);

int markedI = i, markedJ = -1, j;

while (markedI != -1)

{

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;

}

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;

visited[j] = 1;

markedJ = j;

markedI = markIndices[j];

}

for (;links[j] != -1; j = links[j])

markIndices[j] = markIndices[links[j]];

markIndices[j] = i;

for (int i = 0; i < height; i++)

{

for (int j = 0; j < width; j++)

{

cout << "|" << matrix[i][j] - u[i] - v[j] << "\t";

}

cout << "\n";

}

cout << "\n----------------------------------------\n\n";

}

VPInt result;

for (int j = 0; j < width; j++)

if (markIndices[j] != -1)

result.push\_back(PInt(markIndices[j], j));

return result;

}

int Res(VVInt const& matr, VPInt const& answer)

{

int result = 0;

for (size\_t i = 0; i < answer.size(); i++)

{

result += matr[answer[i].first][answer[i].second];

}

return result;

}

int main()

{

VVInt a = {

{9, 3, 9, 6, 9},

{9, 14, 9, 4, 9},

{14, 11, 17, 8, 8},

{13, 14, 16, 7, 7},

{7, 11, 9, 8, 11}

};

/\*VVInt a =

{

{ 1,3,3,3,7 },

{ 1,15,9,9,9 },

{ 1,11,17,18,18 },

{ 1,22,16,17,17 },

{ 1,11,19,18,19},

};\*/

printMatrVector(a);

VPInt ans = hungarian(a);

printVector(ans);

cout << "Sum: " << Res(a, ans);

return 0;

}

1. Вывод:

В ходе выполнения лабораторной работы были закреплены на практике знания о венгерском методе решения задач назначения. Также получены навыки его программной реализации.