Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа №6
«Задача о назначениях»

Выполнил: ст. гр.953503
Басенко К.А.
Проверил: Дугинов О. И.

Минск 2022

Постановка задачи

(теоретико-графовая формулировка)

Задан сбалансированный полный двудольный граф G с долями V1 и V2 с весами на ребрах w: E(G) → {0, 1, ...}. Требуется найти в G совершенное паросочетание, сумма весов ребер которого минимальна.

(матричная формулировка)

Задана квадратная матрица C размера n×n, состоящая из целых неотрицательных чисел. Требуется выбрать в матрице C элементы так, что в каждой строке и каждом столбце выбран ровно один элемент и сумма выбранных элементов минимальна.

Определения

Двудольный граф G с долями V1 и V2 называется полным, если каждая вершина из доли V1 соединена ребром с каждой вершиной доли V2.

Полный двудольный граф с долями V1 и V2 такими, что число вершин V1 равно числу вершин V2, называется сбалансированным.

Паросочетание в сбалансированном полном двудольном графе называется совершенным, если оно покрывает каждую вершину графа.

Описание алгоритма метода

G сбалансированный полный двудольный граф с долями V1,V2

$n=|V1|=|V2|$

$V1 = \{u_1, u_2, ..., u_n\}$

$V2 = \{v_1, v_2, ..., v_n\}$

C - квадратная матрица размера n×n

$c_{ij} = w(\{u_i ; v_j\})$ - элемент матрицы C, находящийся на пересечении i-ой строки и j-ого столбца.

Вход: матрица C

Выход:совершенное паросочетание M графа G, сумма весов ребер которого минимальна

1. $\forall i \in \{1, 2, ..., n\}\ \alpha_i = 0$
   $\forall i \in \{1, 2, ..., n\}\ \forall j \in \{1, 2, ..., n\}\ \beta_j = \min(c_{ij})$
2. $J = =\{(i, j): \alpha_i + \beta_j = c_{ij}\}$

3. Строим двудольный граф G’ с долями V1’=V1 и V2’=V2 и ребрами из J = . Находим в графе G’ наибольшее паросочетание M.

4. Если |M|=n, то STOP, M - совершенное паросочетание, сумма весов ребер которого минимальна. Иначе переходим на шаг 5.

5. Ориентируем граф G’. Те ребра, что принадлежат паросочетанию M, направляем от вершины из V2’ к вершине из V1’.Те ребра, что не принадлежат паросочетанию M, направляем от вершины из V1’ к вершине из V2’.

6. Все вершины из V1’, не покрытые паросочетанием объявляем стартовыми.

7. Все вершины графа G’, которые достижимы из стартовых, помечаем звездочкой(стартовые вершины по умолчанию достижимы из самих себя).

8. Формируем множество I*, в которое помещаем вершины из V1’, помеченные звездочкой, и множество J*, в которое помещаем вершины из V2’, помеченные звездочкой.

9.

$$\forall i \in I^* \ \overline{\alpha}_i = 1$$
$$\forall i \notin I^* \ \overline{\alpha}_i = -1$$
$$\forall j \in J^* \ \overline{\beta}_j = -1$$
$$\forall j \notin J^* \ \overline{\beta}_i = 1$$

10.

$$\theta = \min_{\substack{i \in I^* \\ j \notin J^*}} \frac{c_{ij} - \alpha_i - \beta_j}{2}$$

11. Обновляем значения переменных:

$$\forall i \in \{1, 2, \ldots, n\} \ \alpha_i \leftarrow \alpha_i + \theta\overline{\alpha}_i$$
$$\forall j \in \{1, 2, \ldots, n\} \ \beta_j \leftarrow \beta_j + \theta\overline{\beta}_j$$

12. Переходим на шаг 2.

Результат работы

Тест 1

$$C = \begin{pmatrix} 7 & 2 & 1 & 9 & 4 \\ 9 & 6 & 9 & 5 & 5 \\ 3 & 8 & 3 & 1 & 8 \\ 7 & 9 & 4 & 2 & 2 \\ 8 & 4 & 7 & 4 & 8 \end{pmatrix}$$

```
C:
[[7 2 1 9 4]
 [9 6 9 5 5]
 [3 8 3 1 8]
 [7 9 4 2 2]
 [8 4 7 4 8]]
Результат: [[2, 4], [3, 1], [4, 5], [5, 2], [1, 3]]
```

Тест 2

$$\begin{pmatrix} 50 & 19 & 22 & 24 \\ 23 & 21 & 23 & 20 \\ 18 & 21 & 23 & 33 \\ 19 & 17 & 15 & 24 \end{pmatrix}$$

```
C:
[[50 19 22 24]
 [23 21 23 20]
 [18 21 23 33]
 [19 17 15 24]]
Результат: [[1, 2], [2, 4], [3, 1], [4, 3]]
```

Тест 3

$$\begin{vmatrix} 4 & 2 & 4 \\ 3 & 1 & 2 \\ 4 & 4 & 4 \end{vmatrix}$$

```
C:
[[4 2 4]
 [3 1 2]
 [4 4 4]]
Результат: [[3, 1], [1, 2], [2, 3]]
```

Код программы

```
import numpy as np
from l5 import max_matching



def assignment_problem(c):
    a = [0] * len(c)
    b = [0] * len(c)
    for i in range(len(b)):
        b[i] = min(np.array(c)[0:len(c), i])
    V1 = [f'x{i}' for i in range(len(c))]
    V2 = [f'y{i}' for i in range(len(c))]
    while True:
        J_equal = []
        for i in range(len(c)):
            for j in range(len(c)):
                if a[i] + b[j] == c[i][j]:
                    J_equal.append([f'x{i}', f'y{j}'])
        M = max_matching(V1, V2, J_equal.copy())
        # print(M)
        if (len(M) == len(c)):
            # STOP
            result = []
            for i in M:
                result.append([int(i[0][1]) + 1, int(i[1][1]) + 1])
            return result
        A = []
        for i in J_equal:
            if (i in M):
                A.append([i[1], i[0]])
            else:
                A.append(i)
        start = V1.copy()
        for i in M:
            if i[0] in start:
                start.remove(i[0])
        I_star = start.copy()
        J_star = []
        V = V1 + V2
        labels = {i: None for i in V}
        for i in start:
            labels[i] = -1  # star
        Q = start.copy()
        while len(Q) != 0:
            point = Q.pop(0)
            out_arcs = [i for i in A if i[0] == point]
            for i in out_arcs:
                if labels[i[1]] == None:
                    labels[i[1]] = -1
                    Q.append(i[1])
                    if i[1][0] == 'x':
```

```
                    I_star.append(i[1])
                else:
                    J_star.append(i[1])
        _a = [-1] * len(a)
        _b = [1] * len(b)
        for i in I_star:
            _a[int(i[1])] = 1
        for j in J_star:
            _b[int(j[1])] = -1
        teta = min(
            [(c[int(i[1])][int(j[1])] - a[int(i[1])] -
             b[int(j[1])]) / 2
             for i in I_star
             for j in V2 if j not in J_star
             ]
        )
        for i in range(len(a)):
            a[i] = a[i] + teta * _a[i]
        for i in range(len(b)):
            b[i] = b[i] + teta * _b[i]


if __name__ == '__main__':
    c = np.array([
        [7, 2, 1, 9, 4],
        [9, 6, 9, 5, 5],
        [3, 8, 3, 1, 8],
        [7, 9, 4, 2, 2],
        [8, 4, 7, 4, 8],
    ])
    print("C:")
    print(c)
    result = assignment_problem(c.copy())
    print("Результат:", result)
    c = np.array([
        [50, 19, 22, 24],
        [23, 21, 23, 20],
        [18, 21, 23, 33],
        [19, 17, 15, 24]
    ])
    print()
    print("C:")
    print(c)
    result = assignment_problem(c.copy())
    print("Результат:", result)
    c = np.array([
        [4, 2, 4],
        [3, 1, 2],
        [4, 4, 4]
    ])
    print()
```

```
print("C:")
print(c)
result = assignment_problem(c.copy())
print("Результат:", result)
```