|  |  |  |  |
| --- | --- | --- | --- |
| **Zagadnienie przydziału – algorytm / metoda węgierska** | | | |
| Skład zespołu: | Grupa: | | Środowisko uruchomienia: |
| Arkadiusz Kowalski  Kacper Komenda  Piotr Kur | Laboratoryjna 2 | Środa 16:45 | Python 3.12 |

1. Zadanie 1

Kod programu:

import numpy as np

def metoda\_wegierska(koszty):

    #1: Redukcja całkowita

    koszty = redukcja\_calkowita(koszty)

    print (koszty)

    #2: Inicjalizacja

    n = len(koszty)

    count\_zero\_lines = 0

    #3: Wykonanie iteracji, dopóki nie zostaną wybrane wszystkie elementy

    while count\_zero\_lines < n:

        zaznaczone\_wiersze, zaznaczone\_kolumny = wykreslanie\_zer\_min\_linii(koszty)

        print (zaznaczone\_wiersze)

        print (zaznaczone\_kolumny)

        count\_zero\_lines = len(zaznaczone\_wiersze) + len(zaznaczone\_kolumny)

        if count\_zero\_lines < n:

            koszty = adjust\_matrix()

        else:

            rozwiazanie = znajdz\_zerowy\_element(koszty)

    return rozwiazanie

def redukcja\_calkowita(matrix):

    # Zmniejszenie macierzy szukając najmniejszej wartości w wierszu nastepnie w kolumnie

    # Zmniejszenie wiersza

    row\_mins = matrix.min(axis=1).reshape(-1, 1)

    row\_reduced = matrix - row\_mins

    # Obrót macierzy i redukcja kolumn

    col\_mins = row\_reduced.min(axis=0)

    col\_reduced = (row\_reduced.T - col\_mins).T

    return col\_reduced

def znajdz\_zerowy\_element(zero\_mat):

    mark\_zero = []

    # Iterujemy, dopóki istnieją zera w macierzy

    while np.count\_nonzero(zero\_mat) > 0:

        min\_row = [99999, -1]

        # Znajdujemy wiersz z najmniejszą ilością zer

        for row\_num in range(zero\_mat.shape[0]):

            count\_zeros = np.sum(zero\_mat[row\_num] == 0)

            if count\_zeros > 0 and min\_row[0] > count\_zeros:

                min\_row = [count\_zeros, row\_num]

        # Jeśli znajdujemy wiersz z zerami

        if min\_row[1] != -1:

            zero\_index = np.where(zero\_mat[min\_row[1]] == 0)[0][0]

            mark\_zero.append((min\_row[1], zero\_index))

            # Oznaczamy wiersz i kolumnę zawierającą zero jako użyte

            zero\_mat[min\_row[1], :] = True

            zero\_mat[:, zero\_index] = True

        else:

            break  # Jeśli nie ma już zer do wyznaczenia, przerywamy pętlę

    return mark\_zero[0], mark\_zero

def adjust\_matrix(mat, cover\_rows, cover\_cols):

    cur\_mat = np.copy(mat)

    non\_zero\_elements = []

    # Znajdujemy najmniejszą wartość elementu nieoznaczonego w zaznaczonych wierszach/kolumnach

    for row in range(cur\_mat.shape[0]):

        if row not in cover\_rows:

            for col in range(cur\_mat.shape[1]):

                if col not in cover\_cols:

                    non\_zero\_elements.append(cur\_mat[row, col])

    min\_num = min(non\_zero\_elements)

    # Odejmujemy od wszystkich wartości nieoznaczonych wierszy/kolumn

    for row in range(cur\_mat.shape[0]):

        if row not in cover\_rows:

            for col in range(cur\_mat.shape[1]):

                if col not in cover\_cols:

                    cur\_mat[row, col] -= min\_num

    # Dodajemy do wszystkich wartości oznaczonych wierszy/kolumn

    for row in cover\_rows:

        for col in cover\_cols:

            cur\_mat[row, col] += min\_num

    return cur\_mat

def wykreslanie\_zer\_min\_linii(koszty):

    # Krok 1: Znalezienie minimum w każdym wierszu

    min\_wierszy = [min(wiersz) for wiersz in koszty]

    # Krok 2: Znalezienie minimum w każdej kolumnie

    min\_kolumn = [min(koszty[i][j] for i in range(len(koszty))) for j in range(len(koszty[0]))]

    # Krok 3: Inicjalizacja tablicy zaznaczeń dla wierszy i kolumn

    zaznaczone\_wiersze = [False] \* len(koszty)

    zaznaczone\_kolumny = [False] \* len(koszty[0])

    # Krok 4: Zaznaczanie zer, które pokrywają się z minimalnymi wartościami z Kroków 1 i 2

    for i in range(len(koszty)):

        for j in range(len(koszty[i])):

            if koszty[i][j] == min\_wierszy[i] and koszty[i][j] == min\_kolumn[j]:

                zaznaczone\_wiersze[i] = True

                zaznaczone\_kolumny[j] = True

    # Krok 5: Inicjalizacja tablicy rzędów i kolumn

    rzedy\_do\_wykreslenia = [i for i, zaznaczony in enumerate(zaznaczone\_wiersze) if not zaznaczony]

    kolumny\_do\_wykreslenia = [j for j, zaznaczony in enumerate(zaznaczone\_kolumny) if zaznaczony]

    return rzedy\_do\_wykreslenia, kolumny\_do\_wykreslenia

# Przykładowe użycie:

koszty = np.array([

    [0,0,3,4],[3,0,7,4],[6,1,0,0],[1,0,4,10]

])

rozwiazanie = metoda\_wegierska(koszty)

print(rozwiazanie)

1. Zadanie 2

Dla macierzy:

np.array([[2, 6,  6,  2,  9,  11, 7],

                             [2, 3,  2,  2,  5,  5,  4],

                             [2, 3,  3,  9,  1,  2,  5],

                             [2, 5,  2,  3,  1,  1,  3],

                             [2, 4,  1,  2,  3,  1,  2],

                             [2, 3,  4,  1,  4,  3,  1],

                             [2, 1,  5,  15, 1,  2,  3]])

Wynik:

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

1. Zadanie 3

- Czy wynik redukcji jest zależny od kolejności (wiersze-kolumny/ kolumny-wiersze) – uzyskamy zawsze tą samą macierz / sumaryczną wielkość redukcji?

Wynik końcowy nie jest zależny od kolejności redukcji – otrzymamy takie samo optymalne rozwiązanie, lecz kroki (macierze) pomiędzy mogą się różnić.

- Jak jest możliwa minimalna / maksymalna liczba zer niezależnych w macierzy zredukowanej?

Minimalna liczba zer niezależnych -

Maksymalna liczba zer niezależnych – n dla macierzy nxn.

- Czy wykreślanie zer macierzy jest prawidłowa (zawsze) jeśli będziemy wykreślać kolejno linie (wiesz/kolumna) z największą liczbą nieskreślonych zer?

- Jak się ma minimalna liczba linii (wykreślająca zera) i liczba (maksymalna) zer niezależnych?

Liczba zer niezależnych jest równa liczbie linii w optymalnym rozwiązaniu.

- Czy procedura zwiększająca liczbę zer niezależnych zawsze jest skuteczna / o ile może zmienić się ich liczba?