Sprawozdanie

June 10, 2021

1 Metoda podziału i ograniczeń – algorytm Little'a dla zagadnienia komiwojażera

Grupa: 4a, czwartek 14:30 – 16:00

Data zajęć: 27.05.2021

Skład zespołu:

-Zuzanna Zielińska

-Zofia Lenarczyk

-Maciej Kucharski

-Łukasz Rams

1.0.1 Importy potrzebnych bibliotek

[1]:
```
import numpy as np
from typing import Tuple, Optional, List
```

1.1 Krok 1 - Redukcja macierzy i wyznaczenie najmniejszego kosztu redukcji

[2]:
```
def reduce_matrix(matrix: np.ndarray) -> Optional[Tuple[np.ndarray, int]]:
    """
    Funkcja realizuje redukcję macierzy kwadratowej.
    W każdym wierszu wyszukiwany jest element najmniejszy, który następnie jest␣
 ↪dodawany do kosztu redukcji oraz
    odejmowany (redukowany) od wszystkich elementów w wierszu.
    Analogiczne działanie dla kolumn macierzy.

    :param matrix: macierz wejściowa
    :return: zredukowana macierz, koszt redukcji
    """

    # Sprawdzenie czy macierz jest kwadratowa:
    if matrix.shape[0] != matrix.shape[1]:
        print('Niepoprawne wymiary macierzy wejściowej')
        return None
```

```
    reduction_cost: int = 0  # Koszt redukcji
    reduced_rows_matrix: np.ndarray = np.zeros(matrix.shape)  # Macierz z␣
↪zredukowanymi wierszami
    reduced_matrix: np.ndarray = np.zeros(matrix.shape)  # Zredukowana macierz

   # Redukcja wierszy
    for i in range(matrix.shape[0]):
        if np.array_equal(matrix[i], matrix.shape[0]*[np.inf]) is False:
            minimum = matrix[i].min()
            reduction_cost += minimum
            row_reduced = [x - minimum for x in matrix[i]]
            reduced_rows_matrix[i] = row_reduced
        else:
            reduced_rows_matrix[i] = matrix[i]

#     print(f'Macierz ze zredukowanymi wierszami:\n {reduced_rows_matrix}')
#     print(f'Koszt redukcji wierszy: {reduction_cost}')

    # Redukcja kolumn
    for i in range(matrix.shape[0]):
        if np.array_equal(matrix[:, i], matrix.shape[0]*[np.inf]) is False:
            minimum = reduced_rows_matrix[:, i].min()
            reduction_cost += minimum
            col_reduced = [x - minimum for x in reduced_rows_matrix[:, i]]
            reduced_matrix[:, i] = col_reduced
        else:
            reduced_matrix[:, i] = matrix[:, i]

#     print(f'Zredukowana macierz:\n {reduced_matrix}')
#     print(f'Całkowity koszt redukcji macierzy: {reduction_cost}')

    return reduced_matrix, reduction_cost


def get_cost(row: int, col: int, A: np.ndarray) -> int:
    '''
    Funkcja obliczająca koszt wyłączenia odcinka o danych współrzędnych (row,␣
↪col).
    Koszt to suma najmniejszego elementu w kolumnie col oraz najmniejszego␣
↪elementu w wierszu row, nie                \
    uwzględniając elementu (row, col)

    :param row: numer wiersza
    :param col: numer kolumny
    :param A: macierz
    :return: koszt wyłączenia odcinka o danych współrzędnych
```

```
    '''

    X, Y = A.shape
    row_without_zero = []
    col_without_zero = []

    for i in range(Y):
        if i != col:
            row_without_zero.append(A[row][i])

    for i in range(X):
        if i != row:
            col_without_zero.append(A[i][col])

    cost = np.min(np.array(row_without_zero)) + np.min(np.
↪array(col_without_zero))

    return cost
```

1.2 Krok 2 - Wyznaczenie odcinka o maksymalnym otymistycznym koszcie wyłączenia

[3]:
```
def get_new_coords(A: np.ndarray) -> Tuple[Tuple[int, int], int]:
    '''
    Funkcja wyznaczająca współrzędne nowego odcinka
    PRZYJĘTE OZNACZENIE PRZEJŚĆ ZABRONIONYCH JAKO NP.INF

    :param A: Zredukowna macierz kosztów
    :return: dwuelementowa krotka zawierająca współrzędne początka oraz końca␣
↪odcinka <i*j*>,
             max optymalny koszt wyłączenia tego odcinka
    '''

    print("\nKrok 2\n")

    X, Y = A.shape
    cost: int = -1

    #  współrzędne nowego odcinka
    new_row: int = 0
    new_col: int = 0

    for i in range(X):
        for j in range(Y):
            # Szukam zer w zredukowanej macierzy
            if A[i][j] != np.inf:
```

```
                if A[i][j] == 0:
                    #  koszt wyłączenia - suma elementów minimalnych w wierszu␣
↪'i' oraz kolumnie 'j' bez                                    rozważanego␣
↪elementu zerowego
                    suma = get_cost(i, j, A)
                    #  Wybieramy te współrzędne, dla których koszt wyłączenia␣
↪jest największy
                    if suma > cost:
                        new_row, new_col = i, j
                        cost = suma

    print(f'Wyznaczony odcinek: {new_row, new_col}, koszt wyłączenia: {cost}')
    return (new_row, new_col), cost
```

1.3 Krok 3 - Podział problemu na dwa podproblemy

[4]:
```
def subproblems(data: np.ndarray, path: List[Tuple[int, int]], section:␣
↪Tuple[int, int], LB: int):
    """
    funkcja przyjmuje jako argumenty:
    data: Macierz z danymi
    path: dotychczas wyznaczone odcinki
    section: nowo wyznaczony odcinek
    LB: dotychczasowe LB

    zwraca:
    red_first_sub - zredukowany pierwszy podproblem zawierający odcinek <i*j*>
    new_LB - nowo wyznaczone LB dla red_first_sub
    red_second_sub - zredukowany drugi podproblem nie zawierający odcinka <i*j*>
    new_LB2 - nowo wyznaczone LB dla red_second_sub
    """

    print("\nKrok 3")

    # PODPROBLEM PIERWSZY
    first_sub: np.ndarray = data.copy()

    # Wykreślenie i-tego wiersza oraz j-tej kolumny
    for i in range(first_sub.shape[0]):
        if i == section[0]:
            first_sub[i] = [np.inf] * len(first_sub[i])
        else:
            first_sub[i][section[1]] = np.inf

    # zabronienie podcyklu
```

```
    path.append(section)
    subcircles(first_sub, path)

    # dodatkowa redukcja i nowe LB
    red_first_sub, cost = reduce_matrix(first_sub)
    new_LB = LB + cost

    print("\nMacierz zredukowana:\n", red_first_sub, f"\nNowe LB = {LB} +␣
↪{cost} = {new_LB}")


    # PODPROBLEM DRUGI
    # zabronienie <i*j*>
    second_sub: np.ndarray = data.copy()
    second_sub[section[0]][section[1]] = np.inf

    # redukcja
    red_second_sub, cost = reduce_matrix(second_sub)

    # LB
    new_LB2 = LB + cost

    print("\nMacierz zredukowana:\n", red_second_sub, f"\nNowe LB = {LB} +␣
↪{cost} = {new_LB2}")

    return [(red_first_sub, new_LB), (red_second_sub, new_LB2)]
```

[5]:
```
def subcircles(data: np.ndarray, section: List[Tuple[int, int]]) -> None:
    """
    data: macierz z danymi
    section: dotychczas wyznaczone odcinki
    Funkcja zabrania powstawania podcykli
    """
    if len(section) < data.shape[0]:
        p = section[-1][0]
        k = section[-1][1]
        begins = [elem[0] for elem in section]
        ends = [elem[1] for elem in section]

        # znalezienie początku i końca odcinka
        while True:
            if p in ends:
                p = section[ends.index(p)][0]
            else:
                break
```

```
        while True:
            if k in begins:
                k = section[begins.index(k)][1]
            else:
                break

        # zabronienie podcyklu
        data[k][p] = np.inf
```

1.4 Krok 4 - Analiza podproblemów i kryterium zakończenia obliczeń

[6]:
```
#Funkcje obsługi pojedyńczych podproblemów
#-------------------------------------------------------------------------------------------
#Jeśli macierz jest wypełniona nieskończonościami => zamknij podproblem
def KZ1(cost: float) -> bool:

    """
    cost - najmniejszy koszt redukcji w obecnej pętli
    return: True - problem zamknięty
    return: False - problem niezamknięty
    """

    if cost == np.inf:
        return True
    else:
        return False

    pass

def KZ2(LB, minim):

    if LB > minim:
        return True
    else:
        return False

#Jeśli wszystkie wierzchołki zostały odwiedzone => zamknij podproblem
def KZ3(matrix: np.ndarray, path: List[Tuple[int, int]]) -> bool:

    """
    matrix - macierz po podziale na podproblemy
    path - krotka ze współrzędnymi odwiedzonych wierzchołków
    return: True - problem zamknięty
    return: False - problem niezamknięty
```

```
    """

    if len(matrix) == len(path):
        return True
    else:
        return False

    pass

#Funkcja zamykania wszystkich podproblemów
#-------------------------------------------------------------------------------------------
#Jeśli podproblem został rozwiązany -> wyrzuć go z listy podproblemów
# Jeśli lista podproblemów jest pusta -> zakończ cały algorytm - do␣
 ↪zaimplementowania w funkcji zewnętrznej
# Jeśli lista podproblemów nie jest pusta -> przejdź do kroku 5 - do␣
 ↪zaimplementowania w funkcji zewnętrznej
def closing_subproblems(list_of_subproblems: List[Tuple[np.ndarray, float]],␣
 ↪path: List[Tuple[int, int]]) -> Tuple[bool, List[Tuple[np.ndarray, float]]]:

    """
    list_of_subproblems = List[matrix, LB] - lista podproblemów
    matrix - macierz po redukcji
    LB - najmniejszy koszt redukcji w obecnej pętli
    path - krotka ze współrzędnymi odwiedzonych wierzchołków
    return: list_copy_copy - lista niezamkniętych podproblemów
    """

    print("\nKrok 4\n")

    list_copy = list_of_subproblems.copy()
    n = len(list_of_subproblems)

    minim = np.inf
    for j in range(n): #Liczenie wartości odcinającej
        minim = min(minim, list_of_subproblems[j][1])

    print("Lista zamkniętych podproblemów:")
    count = 0

    for i in range(n):

        matrix, LB = list_of_subproblems[i]
        kz1 = KZ1(LB)
        kz2 = KZ2(LB, minim)
        kz3 = KZ3(matrix, path)

        if kz1 == True or kz2 == True or kz3 == True:
```

```
            count = count+1
            print(f"\nPodproblem {count} :\n{list_copy[i][0]} \nLB =␣
↪{list_copy[i][1]}")

            if kz1 == True:
                print("Zamknięte przez KZ1 - skończyły się nieodwiedzone␣
↪wierzchołki")
            elif kz2 == True:
                print("Zamknięte przez KZ2 - jest lepsze rozwiązanie")
            elif kz3 == True:
                print("Zamknięte przez KZ3 - skończyła się ścieżka")

            list_copy[i] = None

    print("\nLista niezamkniętych podproblemów:\n")
    count = 0

    list_copy_copy = []

    for i in list_copy:
        if i != None:
            count = count+1
            print(f"Podproblem {count} :\n{i[0]} \nLB = {i[1]}\n")
            list_copy_copy.append(i)

    if list_copy_copy == []:
        return True, list_copy_copy, minim
    else:
        return False, list_copy_copy, minim
    pass
```

1.5 Krok 5 - Wybór podproblemu o minimalnej wartości kosztu redukcji

[7]:
```
def choose_subproblem(list_of_subproblems: List[Tuple[np.ndarray, float]],␣
↪LB_min: float) -> Tuple[np.ndarray, float]:

    print("Krok 5\n")


    for i in list_of_subproblems:
        if i[1] == LB_min:
            print("Najmniejsze LB:", i[1])
            return i

    pass
```

2 Algorytm Little'a

[ ]:
```
def Little_algorithm (matrix: np.ndarray):

    print("Macierz wejściowa:\n", matrix, '\n')

    path = []
    stop = False
    count = 0
    list_of_subproblems = []

    print("Krok 1\n")

    reduced_matrix, reduction_cost = reduce_matrix(matrix) #Krok 1
    print(f'Zredukowana macierz:\n {reduced_matrix}')
    print(f'Całkowity koszt redukcji macierzy: {reduction_cost}')

    while stop == False:

        count = count+1

        print("\n\n-------------------------------------------------------------------------------")
        print("Numer pętli:", count)

        new_vertex, LB = get_new_coords(reduced_matrix) #Krok 2


#         subproblem1, subproblem2 = subproblems(reduced_matrix, path, new_vertex, LB) #Krok 3
#         list_of_subproblems.append(subproblem1)
#         list_of_subproblems.append(subproblem2)
        list_of_subproblems = subproblems(reduced_matrix, path, new_vertex, LB) #Krok 3


        stop, list_of_subproblems, LB_min = closing_subproblems(list_of_subproblems, path) #Krok 4
        if stop == True:
            break

        subproblem = choose_subproblem(list_of_subproblems, LB_min) #Krok 5
        reduced_matrix = subproblem[0]

        print("Droga:", path)

    return path
```

```
matrix = np.array([[5, 8, 9, 2, 1, 5, 8, 6, 4, 2],
                   [3, 4, 5, 8, 7, 5, 2, 1, 6, 5],
                   [9, 8, 7, 5, 9, 2, 3, 6, 5, 9],
                   [8, 7, 5, 6, 5, 7, 8, 9, 5, 6],
                   [2, 1, 4, 5, 7, 8, 6, 3, 2, 1],
                   [4, 7, 5, 8, 9, 6, 5, 2, 1, 4],
                   [7, 9, 5, 6, 3, 2, 1, 4, 5, 4],
                   [8, 2, 1, 5, 3, 2, 1, 2, 3, 6],
                   [5, 7, 5, 6, 2, 7, 9, 3, 4, 6],
                   [7, 5, 6, 4, 6, 7, 9, 2, 1, 7]])

path = Little_algorithm(matrix)
print(f"Przebyta trasa: {path}")
```

```
Macierz wejściowa:
 [[5 8 9 2 1 5 8 6 4 2]
 [3 4 5 8 7 5 2 1 6 5]
 [9 8 7 5 9 2 3 6 5 9]
 [8 7 5 6 5 7 8 9 5 6]
 [2 1 4 5 7 8 6 3 2 1]
 [4 7 5 8 9 6 5 2 1 4]
 [7 9 5 6 3 2 1 4 5 4]
 [8 2 1 5 3 2 1 2 3 6]
 [5 7 5 6 2 7 9 3 4 6]
 [7 5 6 4 6 7 9 2 1 7]]

Krok 1

Zredukowana macierz:
 [[3. 7. 8. 0. 0. 4. 7. 5. 3. 1.]
 [1. 3. 4. 6. 6. 4. 1. 0. 5. 4.]
 [6. 6. 5. 2. 7. 0. 1. 4. 3. 7.]
 [2. 2. 0. 0. 0. 2. 3. 4. 0. 1.]
 [0. 0. 3. 3. 6. 7. 5. 2. 1. 0.]
 [2. 6. 4. 6. 8. 5. 4. 1. 0. 3.]
 [5. 8. 4. 4. 2. 1. 0. 3. 4. 3.]
 [6. 1. 0. 3. 2. 1. 0. 1. 2. 5.]
 [2. 5. 3. 3. 0. 5. 7. 1. 2. 4.]
 [5. 4. 5. 2. 5. 6. 8. 1. 0. 6.]]
Całkowity koszt redukcji macierzy: 18.0


--------------------------------------------------------------------------------
Numer pętli: 1

Krok 2
```

```
Wyznaczony odcinek: (1, 7), koszt wyłączenia: 2.0

Krok 3

Macierz zredukowana:
 [[ 3.  7.  8.  0.  0.  4.  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 6.  6.  5.  2.  7.  0.  1. inf  3.  7.]
 [ 2.  2.  0.  0.  0.  2.  3. inf  0.  1.]
 [ 0.  0.  3.  3.  6.  7.  5. inf  1.  0.]
 [ 2.  6.  4.  6.  8.  5.  4. inf  0.  3.]
 [ 5.  8.  4.  4.  2.  1.  0. inf  4.  3.]
 [ 6. inf  0.  3.  2.  1.  0. inf  2.  5.]
 [ 2.  5.  3.  3.  0.  5.  7. inf  2.  4.]
 [ 5.  4.  5.  2.  5.  6.  8. inf  0.  6.]]
Nowe LB = 2.0 + 0.0 = 2.0

Macierz zredukowana:
 [[ 3.  7.  8.  0.  0.  4.  7.  4.  3.  1.]
 [ 0.  2.  3.  5.  5.  3.  0. inf  4.  3.]
 [ 6.  6.  5.  2.  7.  0.  1.  3.  3.  7.]
 [ 2.  2.  0.  0.  0.  2.  3.  3.  0.  1.]
 [ 0.  0.  3.  3.  6.  7.  5.  1.  1.  0.]
 [ 2.  6.  4.  6.  8.  5.  4.  0.  0.  3.]
 [ 5.  8.  4.  4.  2.  1.  0.  2.  4.  3.]
 [ 6.  1.  0.  3.  2.  1.  0.  0.  2.  5.]
 [ 2.  5.  3.  3.  0.  5.  7.  0.  2.  4.]
 [ 5.  4.  5.  2.  5.  6.  8.  0.  0.  6.]]
Nowe LB = 2.0 + 2.0 = 4.0

Krok 4

Lista zamkniętych podproblemów:

Podproblem 1 :
[[ 3.  7.  8.  0.  0.  4.  7.  4.  3.  1.]
 [ 0.  2.  3.  5.  5.  3.  0. inf  4.  3.]
 [ 6.  6.  5.  2.  7.  0.  1.  3.  3.  7.]
 [ 2.  2.  0.  0.  0.  2.  3.  3.  0.  1.]
 [ 0.  0.  3.  3.  6.  7.  5.  1.  1.  0.]
 [ 2.  6.  4.  6.  8.  5.  4.  0.  0.  3.]
 [ 5.  8.  4.  4.  2.  1.  0.  2.  4.  3.]
 [ 6.  1.  0.  3.  2.  1.  0.  0.  2.  5.]
 [ 2.  5.  3.  3.  0.  5.  7.  0.  2.  4.]
 [ 5.  4.  5.  2.  5.  6.  8.  0.  0.  6.]]
LB = 4.0
Zamknięte przez KZ2 - jest lepsze rozwiązanie
```

```
Lista niezamkniętych podproblemów:

Podproblem 1 :
[[ 3.  7.  8.  0.  0.  4.  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 6.  6.  5.  2.  7.  0.  1. inf  3.  7.]
 [ 2.  2.  0.  0.  0.  2.  3. inf  0.  1.]
 [ 0.  0.  3.  3.  6.  7.  5. inf  1.  0.]
 [ 2.  6.  4.  6.  8.  5.  4. inf  0.  3.]
 [ 5.  8.  4.  4.  2.  1.  0. inf  4.  3.]
 [ 6. inf  0.  3.  2.  1.  0. inf  2.  5.]
 [ 2.  5.  3.  3.  0.  5.  7. inf  2.  4.]
 [ 5.  4.  5.  2.  5.  6.  8. inf  0.  6.]]
LB = 2.0

Krok 5

Najmniejsze LB: 2.0
Droga: [(1, 7)]


--------------------------------------------------------------------------------
Numer pętli: 2

Krok 2

Wyznaczony odcinek: (2, 5), koszt wyłączenia: 2.0

Krok 3

Macierz zredukowana:
 [[ 3.  7.  8.  0.  0. inf  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 2.  2.  0.  0.  0. inf  3. inf  0.  1.]
 [ 0.  0.  3.  3.  6. inf  5. inf  1.  0.]
 [ 2.  6. inf  6.  8. inf  4. inf  0.  3.]
 [ 5.  8.  4.  4.  2. inf  0. inf  4.  3.]
 [ 6. inf  0.  3.  2. inf  0. inf  2.  5.]
 [ 2.  5.  3.  3.  0. inf  7. inf  2.  4.]
 [ 5.  4.  5.  2.  5. inf  8. inf  0.  6.]]
Nowe LB = 2.0 + 0.0 = 2.0

Macierz zredukowana:
 [[ 3.  7.  8.  0.  0.  3.  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 5.  5.  4.  1.  6. inf  0. inf  2.  6.]
 [ 2.  2.  0.  0.  0.  1.  3. inf  0.  1.]
```

```
 [ 0.  0.  3.  3.  6.  6.  5. inf  1.  0.]
 [ 2.  6.  4.  6.  8.  4.  4. inf  0.  3.]
 [ 5.  8.  4.  4.  2.  0.  0. inf  4.  3.]
 [ 6. inf  0.  3.  2.  0.  0. inf  2.  5.]
 [ 2.  5.  3.  3.  0.  4.  7. inf  2.  4.]
 [ 5.  4.  5.  2.  5.  5.  8. inf  0.  6.]]
Nowe LB = 2.0 + 2.0 = 4.0

Krok 4

Lista zamkniętych podproblemów:

Podproblem 1 :
[[ 3.  7.  8.  0.  0.  3.  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 5.  5.  4.  1.  6. inf  0. inf  2.  6.]
 [ 2.  2.  0.  0.  0.  1.  3. inf  0.  1.]
 [ 0.  0.  3.  3.  6.  6.  5. inf  1.  0.]
 [ 2.  6.  4.  6.  8.  4.  4. inf  0.  3.]
 [ 5.  8.  4.  4.  2.  0.  0. inf  4.  3.]
 [ 6. inf  0.  3.  2.  0.  0. inf  2.  5.]
 [ 2.  5.  3.  3.  0.  4.  7. inf  2.  4.]
 [ 5.  4.  5.  2.  5.  5.  8. inf  0.  6.]]
LB = 4.0
Zamknięte przez KZ2 - jest lepsze rozwiązanie

Lista niezamkniętych podproblemów:

Podproblem 1 :
[[ 3.  7.  8.  0.  0. inf  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 2.  2.  0.  0.  0. inf  3. inf  0.  1.]
 [ 0.  0.  3.  3.  6. inf  5. inf  1.  0.]
 [ 2.  6. inf  6.  8. inf  4. inf  0.  3.]
 [ 5.  8.  4.  4.  2. inf  0. inf  4.  3.]
 [ 6. inf  0.  3.  2. inf  0. inf  2.  5.]
 [ 2.  5.  3.  3.  0. inf  7. inf  2.  4.]
 [ 5.  4.  5.  2.  5. inf  8. inf  0.  6.]]
LB = 2.0

Krok 5

Najmniejsze LB: 2.0
Droga: [(1, 7), (2, 5)]
```

```
Numer pętli: 3

Krok 2

Wyznaczony odcinek: (4, 0), koszt wyłączenia: 2.0

Krok 3

Macierz zredukowana:
 [[inf  5.  8.  0. inf inf  7. inf  3.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf  0.  0.  0.  0. inf  3. inf  0.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf  4. inf  6.  8. inf  4. inf  0.  2.]
 [inf  6.  4.  4.  2. inf  0. inf  4.  2.]
 [inf inf  0.  3.  2. inf  0. inf  2.  4.]
 [inf  3.  3.  3.  0. inf  7. inf  2.  3.]
 [inf  2.  5.  2.  5. inf  8. inf  0.  5.]]
Nowe LB = 2.0 + 3.0 = 5.0

Macierz zredukowana:
 [[ 1.  7.  8.  0.  0. inf  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0.  2.  0.  0.  0. inf  3. inf  0.  1.]
 [inf  0.  3.  3.  6. inf  5. inf  1.  0.]
 [ 0.  6. inf  6.  8. inf  4. inf  0.  3.]
 [ 3.  8.  4.  4.  2. inf  0. inf  4.  3.]
 [ 4. inf  0.  3.  2. inf  0. inf  2.  5.]
 [ 0.  5.  3.  3.  0. inf  7. inf  2.  4.]
 [ 3.  4.  5.  2.  5. inf  8. inf  0.  6.]]
Nowe LB = 2.0 + 2.0 = 4.0

Krok 4

Lista zamkniętych podproblemów:

Podproblem 1 :
[[inf  5.  8.  0. inf inf  7. inf  3.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf  0.  0.  0.  0. inf  3. inf  0.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf  4. inf  6.  8. inf  4. inf  0.  2.]
 [inf  6.  4.  4.  2. inf  0. inf  4.  2.]
 [inf inf  0.  3.  2. inf  0. inf  2.  4.]
 [inf  3.  3.  3.  0. inf  7. inf  2.  3.]
```

```
 [inf  2.  5.  2.  5. inf  8. inf  0.  5.]]
LB = 5.0
Zamknięte przez KZ2 - jest lepsze rozwiązanie

Lista niezamkniętych podproblemów:

Podproblem 1 :
[[ 1.  7.  8.  0.  0. inf  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0.  2.  0.  0.  0. inf  3. inf  0.  1.]
 [inf  0.  3.  3.  6. inf  5. inf  1.  0.]
 [ 0.  6. inf  6.  8. inf  4. inf  0.  3.]
 [ 3.  8.  4.  4.  2. inf  0. inf  4.  3.]
 [ 4. inf  0.  3.  2. inf  0. inf  2.  5.]
 [ 0.  5.  3.  3.  0. inf  7. inf  2.  4.]
 [ 3.  4.  5.  2.  5. inf  8. inf  0.  6.]]
LB = 4.0

Krok 5

Najmniejsze LB: 4.0
Droga: [(1, 7), (2, 5), (4, 0)]


--------------------------------------------------------------------------------
Numer pętli: 4

Krok 2

Wyznaczony odcinek: (4, 1), koszt wyłączenia: 2.0

Krok 3

Macierz zredukowana:
 [[ 1. inf  8.  0.  0. inf  7. inf  3.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0. inf  0.  0.  0. inf  3. inf  0.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0. inf inf  6.  8. inf  4. inf  0.  2.]
 [ 3. inf  4.  4.  2. inf  0. inf  4.  2.]
 [ 4. inf  0.  3. inf inf  0. inf  2.  4.]
 [ 0. inf  3.  3.  0. inf  7. inf  2.  3.]
 [ 3. inf  5.  2.  5. inf  8. inf  0.  5.]]
Nowe LB = 2.0 + 1.0 = 3.0

Macierz zredukowana:
```

```
 [[ 1.  5.  8.  0.  0. inf  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0.  0.  0.  0.  0. inf  3. inf  0.  1.]
 [inf inf  3.  3.  6. inf  5. inf  1.  0.]
 [ 0.  4. inf  6.  8. inf  4. inf  0.  3.]
 [ 3.  6.  4.  4.  2. inf  0. inf  4.  3.]
 [ 4. inf  0.  3.  2. inf  0. inf  2.  5.]
 [ 0.  3.  3.  3.  0. inf  7. inf  2.  4.]
 [ 3.  2.  5.  2.  5. inf  8. inf  0.  6.]]
Nowe LB = 2.0 + 2.0 = 4.0

Krok 4

Lista zamkniętych podproblemów:

Podproblem 1 :
[[ 1.  5.  8.  0.  0. inf  7. inf  3.  1.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0.  0.  0.  0.  0. inf  3. inf  0.  1.]
 [inf inf  3.  3.  6. inf  5. inf  1.  0.]
 [ 0.  4. inf  6.  8. inf  4. inf  0.  3.]
 [ 3.  6.  4.  4.  2. inf  0. inf  4.  3.]
 [ 4. inf  0.  3.  2. inf  0. inf  2.  5.]
 [ 0.  3.  3.  3.  0. inf  7. inf  2.  4.]
 [ 3.  2.  5.  2.  5. inf  8. inf  0.  6.]]
LB = 4.0
Zamknięte przez KZ2 - jest lepsze rozwiązanie

Lista niezamkniętych podproblemów:

Podproblem 1 :
[[ 1. inf  8.  0.  0. inf  7. inf  3.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0. inf  0.  0.  0. inf  3. inf  0.  0.]
 [inf inf inf inf inf inf inf inf inf inf]
 [ 0. inf inf  6.  8. inf  4. inf  0.  2.]
 [ 3. inf  4.  4.  2. inf  0. inf  4.  2.]
 [ 4. inf  0.  3. inf inf  0. inf  2.  4.]
 [ 0. inf  3.  3.  0. inf  7. inf  2.  3.]
 [ 3. inf  5.  2.  5. inf  8. inf  0.  5.]]
LB = 3.0

Krok 5

Najmniejsze LB: 3.0
```

```
Droga: [(1, 7), (2, 5), (4, 0), (4, 1)]


--------------------------------------------------------------------------------
Numer pętli: 5

Krok 2

Wyznaczony odcinek: (6, 6), koszt wyłączenia: 2.0

Krok 3
```