LAB9: Problem szeregowania zadań – algorytm Johnsona

Zadanie 1

Implementacja algorytmu Johnsona dla problemu 2 maszyn

In [10]:
```
from copy import deepcopy
inf = float('inf')

def johnson(tasks):
    #algorytm przyjmuje listę 2xN, gdzie 2 to liczba maszyn a N liczba zadań    inf = float('inf')
    cp = deepcopy(tasks) #utowrzenie kopii listy zadań
    # utworzenie pustej listy 2xN do umieszczania danych wyjściowych
    output = [[None for _ in range(len(tasks[0]))] for _ in range(2)]
    #index wstawiania zadań na początek
    start_idx = 0
    #index wstawiania zadań na koniec
    end_idx = len(tasks[0])-1
    min_elem = []
    min_idx = []

    #algorytm wykonuje się dopóki indeksy wstawiania zadań nie będą równe
    while start_idx <= end_idx:
        #iteracja przez wiersze i kolumny w poszukiwaniu minimum
        for row_id, row in enumerate(cp):
            #dodwanie minimum z danego wierwsza oraz indeksu danego minima
            min_elem.append(min(row))
            min_idx.append(cp[row_id].index(min_elem[row_id]))
        #jeżeli minimum jest w wierszu 0 lub są równe
        if min_elem[0] < min_elem[1] or min_elem[0] == min_elem[1]:
            #uzupełnienie wyjściowej listy o zadanie z obecnym minimum
            output[0][start_idx] = cp[0][min_idx[0]]
            output[1][start_idx] = cp[1][min_idx[0]]
            #zwiększenie indeksu oraz ustawienie przetworzonego zadania na inf
            start_idx += 1
            cp[0][min_idx[0]] = inf
            cp[1][min_idx[0]] = inf
        #jeżeli minimum jest w wierszu 1 lub są równe
        elif min_elem[0] > min_elem[1] or min_elem[0] == min_elem[1]:
            #uzupełnienie wyjściowej listy o zadanie z obecnym minimum
            output[0][end_idx] = cp[0][min_idx[1]]
            output[1][end_idx] = cp[1][min_idx[1]]
            #zmniejszenie indeksu oraz ustawienie przetworzonego zadania na inf
            end_idx -= 1
            cp[0][min_idx[1]] = inf
            cp[1][min_idx[1]] = inf
        #zerowanie
        min_elem = []
        min_idx = []
    return output

def ending_times(tasks):
    cp = deepcopy(tasks)
    for idx, elem in enumerate(cp[0][:-1]):
        cp[0][idx+1] += cp[0][idx]
        prev = cp[1][idx-1] if idx > 0 else -inf
        cp[1][idx] += max([cp[0][idx], prev])
    cp[1][-1] += cp[0][-1]
    return cp
```

Zadanie 2

Przykład uszeregowania dla 2 maszyn i 10 zadań wraz z zaprezentowaniem uporządkowania początkowego, końcowego oraz czasów końcowych.

In [12]:
```
# tasks = [
#     [9, 6, 8, 7, 12, 3],
#     [7, 3, 5, 10, 4, 7]
# ]

tasks = [
    [9, 6, 8, 7, 12, 3, 14, 15, 4, 5],
    [7, 3, 5, 10, 4, 6, 11, 20, 5, 2]
]
print(f'Uszeregowanie początkowe:\n{tasks[0]}\n{tasks[1]}\n')
tasks_ordered = johnson(tasks)
print(f'Uszeregowanie końcowe:\n{tasks_ordered[0]}\n{tasks_ordered[1]}\n')
end_times = ending_times(tasks_ordered)
print(f'Czasy zakończeń:\n{end_times[0]}\n{end_times[1]}\n')
```

```
Uszeregowanie początkowe:
[9, 6, 8, 7, 12, 3, 14, 15, 4, 5]
[7, 3, 5, 10, 4, 6, 11, 20, 5, 2]

Uszeregowanie końcowe:
[3, 4, 7, 15, 14, 9, 8, 12, 6, 5]
[6, 5, 10, 20, 11, 7, 5, 4, 3, 2]

Czasy zakończeń:
[3, 7, 14, 29, 43, 52, 60, 72, 78, 83]
[9, 14, 24, 49, 60, 67, 72, 76, 81, 85]
```

Zadanie 3

- Powyższy problem posiada następujące oznaczenie w notavcji Grahama: F2||Cmax. Jest to problem przepływowy flow-shop, w którym wszytskie zadania wyknywane są według tej samej sekwencji, a każda operacja z sekwencji wyknywana jest na innej maszynie. W tym przypadku liczba maszyn wynosi 2. Pole || oznacza brak dodatkowych ograniczeń technologicznych. Cmax to funckja celu biorąca pod uwagę maksymalny czas wykonania się całej sekwencji, który jest minimalizowany.
- Przy 2 takich samych minimach wybieramy np. pierwsze z nich a czas dla takiego alternatywnego uszeregowania będzie taki sam.
- W tym przypadku nie ma dodatkowych warunków. W algorytmie Johnsona dla 3 maszyn musi być spełniony warunek taki, że najmniejszy koszt dla maszyny 1 musi być większy lub równy od największego kosztu maszyny 2 lub największy koszt maszyny 2 musi być mniejszy bądź równy od od najmniejszego kosztu maszyny 3. Dodatkowo algorytm Johnsona dla 2 maszyn jest algorytmem ścisłym podczas gdy algorytm CDS zwraca rozwiązanie przybliżone.
- Złożoność obliczeniowa tego algorytmu to O(n*logn)