Sprawozdanie Problem szeregowania zadań – algorytm Johnsona

Jakub Ochman grupa 3. AiR

Zadanie 1

Implementacja algorytmu Johnsona dla dwóch maszyn oraz algorytmu CDS:

In [43]:
```
def johnson_2(tasks): # Algorytm Johnsona dla dwóch maszyn
    # tasks - lista krotek: (index, czas maszyny 1, czas maszyny 2)
    n = len(tasks) # liczba zadań
    start = 0 # indeks pierwszego zadania
    end = n - 1 # indeks ostatniego zadania
    order = [None] * n # lista do przechowywania kolejności zadań
    details = tasks[:] # tymczasowa lista zadań

    while details: # dopóki lista details nie jest pusta
        min_detail = min(details, key=lambda x: min(x[1], x[2])) # wyszukiwanie zadania z najmniejszym czasem
        i, t1, t2 = min_detail # rozpakowanie krotki

        if t1 <= t2: #  jeśli czas znajduje się w pierwszym rzędzie
            order[start] = i # to ląduje na początku sortowanej listy
            start += 1  # inkrementacja indeksu początku
        else: # jeśli jest w drugim rzędzie
            order[end] = i  # to ląduje na końcu
            end -= 1 # dekrementacja indeksu końca

        details.remove(min_detail) # usuwanie zadania z tymczasowej listy

    return order # zwracanie kolejności (bez czasów)

def cds_algorithm(times): # algorytm CDS
    # times - macierz n x m - zadania x maszyny
    n = len(times)    # ilość zadań
    m = len(times[0]) # ilość maszyn
    best_order = None # tymczasowa zmienna do przechowywania najlepszej kolejności
    best_times = None # zmienna przechowująca najlepszą macierz czasów

    best_makespan = float('inf') # najlepszy minimalny czas realizacji
    for r in range(1, m): # dla r w zakresie 1, m-1
        times_2d = [] # tymczasowa lista przekazywana do algorytmu johnsona
        for i in range(n): # dla każdego zadania
            time_a = sum(times[i][:r]) # czas maszyny 1
            time_b = sum(times[i][-r:])  # czas maszyny 2
            times_2d.append((i, time_a, time_b)) # uzupełnianie tymczasowej listy

        order = johnson_2(times_2d) #  tworzenie pomocniczego zadania dla 2 maszyn

        # wyznaczanie czasu realizacji
        end_times = [[0]*m for _ in range(n)] # pomocnicza macierz

        for i, task in enumerate(order):#
            for j in range(m):          # iteracja po elementach macierzy
                if i == 0 and j == 0: # pierwszy element
                    end_times[i][j] = times[task][j] # czas realizacji to jego czas
                elif i == 0: # element w pierwszym rzędzie, czas realizacji to czas elemenu z
                    end_times[i][j] = end_times[i][j-1] + times[task][j] #  poprzedniej kolumny i poprzedniego
                    #wiersza oraz jego czas realizacji
                elif j == 0: # element w pierwszej kolumnie, jego wartość to czas elementu w
                    # poprzednim wierszu i jego czas realizacji
                    end_times[i][j] = end_times[i-1][j] + times[task][j]
                else: # pozostałe elementy, wyznaczana jest maksymalna wartość z elementu po
                    # prawej w macierzy i powyżej, do tego dodaje się czas realizcji
                    end_times[i][j] = max(end_times[i-1][j], end_times[i][j-1]) + times[task][j]

        # całkowity czas realizacji jako kryterium sortowania
        current_makespan = end_times[-1][-1] # ostatni element to całkowity czas
        if current_makespan < best_makespan: # jak jest mniejszy to zastępuje najlepszy
            best_makespan = current_makespan
            best_order = order
            best_times = end_times

    return best_order, best_makespan, best_times
```

Zadanie 2

Poniżej przykład działania algorytmu dla liczby zadań n=10 oraz liczby maszyn m=5

In [42]:
```
def print_times_table(times, order=None, bold_last = False): # funkcja do wyświetlania zadań
    col_width = 6 # szerokość kolumny
    num_tasks = len(times) # lizzba zadań
    num_machines = len(times[0]) # liczba maszyn
    # zmiana kolejności
    if order:
        reordered_times = [times[i] for i in order]
        header_labels = [f"Z{idx}" for idx in order]
```

```python
    else:
        reordered_times = times
        header_labels = [f"Z{i}" for i in range(num_tasks)]

    # Transponowanie: maszyny jako wiersze
    transposed = list(zip(*reordered_times))

    # Nagłówki kolumn
    header = f"{'':<{col_width}}" + "".join(f"{label:>{col_width}}" for label in header_labels)
    print(header)
    print("-" * len(header))

    # Wiersze maszyn M1 ... Mm
    for m_idx, row in enumerate(transposed):
        row_str = f"{'M'+str(m_idx+1):<{col_width}}"
        for j, val in enumerate(row):
            # Pogrubienie ostatniego wiersza i ostatniej kolumny
            if bold_last and m_idx == len(transposed) - 1 and j == len(row) - 1:
                val_str = f"\033[1m{val:>{col_width}}\033[0m"
            else:
                val_str = f"{val:>{col_width}}"
            row_str += val_str
        print(row_str)


tasks = [
 [13, 13,  4, 13, 18],
 [14,  5,  7,  6, 13],
 [18, 18,  3,  5, 13],
 [ 9,  1, 20, 16,  2],
 [19,  4,  5,  7,  8],
 [ 7, 10,  5, 13, 11],
 [ 8,  2, 17,  3, 13],
 [14, 17, 19, 13,  7],
 [11, 18, 19, 20, 14],
 [ 6, 20, 13,  6,  9]
]
print("\nUszeregowanie zadań początkowe:")
print_times_table(tasks)


order, makespan, order_times = cds_algorithm(tasks)

print("\nOptymalna kolejność:", order)
print("Najmniejszy czas:", makespan)
print("\nUszeregowanie zadań końcowe:")
print_times_table(tasks, order)
print("\nCzasy zakończenia zadań:")
print_times_table(order_times, bold_last=True)
```

```
Uszeregowanie zadań początkowe:
            Z0    Z1    Z2    Z3    Z4    Z5    Z6    Z7    Z8    Z9
------------------------------------------------------------------
M1          13    14    18     9    19     7     8    14    11     6
M2          13     5    18     1     4    10     2    17    18    20
M3           4     7     3    20     5     5    17    19    19    13
M4          13     6     5    16     7    13     3    13    20     6
M5          18    13    13     2     8    11    13     7    14     9

Optymalna kolejność: [5, 1, 6, 0, 3, 8, 7, 9, 2, 4]
Najmniejszy czas: 170

Uszeregowanie zadań końcowe:
            Z5    Z1    Z6    Z0    Z3    Z8    Z7    Z9    Z2    Z4
------------------------------------------------------------------
M1           7    14     8    13     9    11    14     6    18    19
M2          10     5     2    13     1    18    17    20    18     4
M3           5     7    17     4    20    19    19    13     3     5
M4          13     6     3    13    16    20    13     6     5     7
M5          11    13    13    18     2    14     7     9    13     8

Czasy zakończenia zadań:
            Z0    Z1    Z2    Z3    Z4    Z5    Z6    Z7    Z8    Z9
------------------------------------------------------------------
M1           7    21    29    42    51    62    76    82   100   119
M2          17    26    31    55    56    80    97   117   135   139
M3          22    33    50    59    79    99   118   131   138   144
M4          35    41    53    72    95   119   132   138   143   151
M5          46    59    72    90    97   133   140   149   162   170
```

1. Jaki typ problemu rozwiązujemy (klasyfikacja Grahama)?

   Rozwiązujemy problem szeregowania zadań na wielu maszynach w układzie przepływowym, oznaczany w klasyfikacji Grahama jako F5||Cmax. Celem jest minimalizacja czasu zakończenia ostatniego zadania. Problem ten jest NP-trudny dla więcej niż dwóch maszyn.

2. Jakie czasy uzyskamy przy alternatywnych sposobach uszeregowania (takie samo min dla kilku zadań)?

   Jeśli dla kilku zadań minimalny czas wykonania występuje na tej samej pozycji, kolejność ich wykonania może wpływać na końcowy czas realizacji. Różne kolejności zadań mogą dać różne czasy zakończenia całego harmonogramu. Dlatego nawet przy takim samym minimum warto przetestować różne warianty.

3. Jakie warunki są konieczne w realizacji algorytmu / co jeśli nie będzie spełniony?

```
CDS i Johnson wymagają, aby czasy były nieujemne i znane z góry oraz aby każde zadanie przechodziło przez maszyny w tej
samej kolejności. Jeśli warunki nie są spełnione (np. brak ciągłości zadań, ujemne czasy, inne kolejności), algorytm może
dać błędne wyniki lub zawieść. Dla więcej niż dwóch maszyn CDS używa aproksymacji, więc nie gwarantuje optymalnego wyniku.
```

4. Jaka jest złożoność obliczeniowa algorytmu?

```
Dla n zadań i m maszyn, algorytm CDS wykonuje m−1 wywołań uproszczonego Johnsona, który ma złożoność O(n²). W rezultacie
całkowita złożoność CDS to O(m·n²). Jest to stosunkowo efektywne jak na typ problemu, ale dla dużych zestawów danch
algorytm osiąga bardzo dużą złożoność.
```

Źródła:

- Na podstawie materiałów z zajęć oraz wykładu
- Na podstawie opisu problemu: https://en.wikipedia.org/wiki/Job-shop_scheduling

---

Środowisko:

Jupiter Notebook w Visual Studio Code z rozszerzeniem Jupiter, Python