Problem szeregowania zadań– algorytm Johnsona

Zadanie 1

Do realizacji każdego z algorytmów potrzebna jest implementacja alg. Johnsona dla problemu dwóch maszyn, wobec tego od niej rozpoczynam rozwiązywanie zadania

```
def johnson2(t, just_ordering=False):
    # szeregownie
    optimal_t = np.zeros(t.shape)
    crossed = list()
    first = 0
    last = -1
    # zakładam, że w pierwszym wierszu znajdują się numery kolejnych zadań
    for k in range(len(t[0])):
        min_t = np.inf
        for i in range(1, 3):
            for j in range(len(t[0])):
                # znaleznienei minimum
                if (t[i, j] < min_t) and (j not in crossed):
                    min_t = t[i, j]
                    argmin_t = (i, j)

        if argmin_t[0] == 1:
            # wstawienie na początek
            optimal_t[0][first] = t[0][argmin_t[1]]
            optimal_t[1][first] = t[1][argmin_t[1]]
            optimal_t[2][first] = t[2][argmin_t[1]]
            first += 1
            crossed.append(argmin_t[1])

        elif argmin_t[0] == 2:
            # wstawienie na koniec
            optimal_t[0][last] = t[0][argmin_t[1]]
            optimal_t[1][last] = t[1][argmin_t[1]]
            optimal_t[2][last] = t[2][argmin_t[1]]
            last -= 1
            crossed.append(argmin_t[1])

    # zwrócenie jedynie kolejności zadań
    # do metody johnsona dla 3 maszyn
    if just_ordering:
        return optimal_t[0]

    # obliczanie teminów
    optimal_t[2, 0] += optimal_t[1, 0]
    for j in range(len(optimal_t[0])):
        if j != 0:
            optimal_t[1, j] += optimal_t[1, j-1]
            optimal_t[2, j] += max(optimal_t[1, j], optimal_t[2, j-1])

    return optimal_t
```

Następnie nieco z rozpędu i niepotrzebnie zaimplementowałem alg. Johnsona dla trzech maszyn, jednak przydatna będzie później stworzona na potrzeby tej implementacji funkcja order(), która oblicza terminy na podstawie zwróconej sekwencji problemu pomocniczego.

```
def johnson3(t):
    if max(t[1]) < max(t[2]) and max(t[3]) < max(t[2]):
        raise ValueError("Johnson rule's condition not satisfied!")

    t_temp = np.array([t[0], t[1]+t[2], t[2]+t[3]])
    ordering = johnson2(t_temp, just_ordering=True)

    return order(t, ordering)

def order(t, ordering):
    optimal_t = t.copy()
    i = 0
    for task in ordering:
        id = int(np.argwhere(t[0] == task))
        optimal_t[:, i] = t[:, id]
        i += 1

    # obliczanie teminów
    for i in range(2, optimal_t.shape[0]):
        optimal_t[i, 0] += optimal_t[i-1, 0]

    for j in range(len(optimal_t[0])):
        if j != 0:
            optimal_t[1, j] += optimal_t[1, j - 1]
            for i in range(2, optimal_t.shape[0]):
                optimal_t[i, j] += max(optimal_t[i-1, j], optimal_t[i, j - 1])

    return optimal_t
```

Na koniec napisałem funkcję CDS() realizującą algorytm Camlbella-Dudka-Smitha

```
def CDS(t):
    min_t = np.inf
    for r in range(1, t.shape[0]):
        # utworzenie problemu pomocniczego
        temp_t = np.array([t[0], np.sum(t[1:r+1], axis=0), np.sum(t[-r:],
axis=0)])

        # wykonanie dla tego problemu algorytmu Johnsona
        candidate_ordering = johnson2(t, just_ordering=True)
        candidate_t = order(t, candidate_ordering)

        # sprawdzenie czy jest to najlepsze z dotychczasowych rozwiązań
        if candidate_t[candidate_t.shape[0]-1, candidate_t.shape[1]-1] <
min_t:
            optimal_t = candidate_t
            min_t = candidate_t[candidate_t.shape[0]-1,
candidate_t.shape[1]-1]

    return optimal_t
```

Zadanie 2

Obliczenia wykonuję na macierzy 6x12 – 5 maszyn 12 zadań, w pierwszym wierszu, zgodnie z wcześniejszym założeniem znajdują się numery zadań.

Uszeregowanie początkowe wraz z wynikiem:

```
Uszeregowanie początkowe:
[[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
 [36. 23. 74. 29. 57. 20. 75.  9. 49. 69. 15. 17.]
 [84. 59. 19. 51. 72. 77. 44. 65. 18. 26. 64. 54.]
 [38.  7. 40. 21. 43. 16.  0.  2. 22. 87. 53. 86.]
 [63. 76. 37. 62. 33. 25. 35. 60. 55.  6. 67. 79.]
 [48. 80. 11. 24. 14. 68. 66. 71. 81. 41. 10. 73.]]

Uszeregowanie końcowe:
[[  8.  11.  12.   6.   2.   4.   1.   5.   7.  10.   3.   9.]
 [  9.  24.  41.  61.  84. 113. 149. 206. 281. 350. 424. 473.]
 [ 74. 138. 192. 269. 328. 379. 463. 535. 579. 605. 624. 642.]
 [ 76. 191. 278. 294. 335. 400. 501. 578. 579. 692. 732. 754.]
 [136. 258. 357. 382. 458. 520. 583. 616. 651. 698. 769. 824.]
 [207. 268. 430. 498. 578. 602. 650. 664. 730. 771. 782. 905.]]
```

Zadanie 3

1. W notacji Grahama ten typ problemu można określić jako FP5C_{max}, czyli jako permutation flow shop o 5 maszynach produkcyjnych, nieograniczonych maszynach transportowych, z brakiem dodatkowych założeń technologicznych oraz kryterium minmaksowym.
2. Ponieważ algorytm CDS jest algorytmem przybliżonym dla alternatywnych sposobów uszeregowania można uzyskać inne wyniki, chyba, że czas wykonania każdej czynności będzie inny dla danej maszyny.
3. Jeśli mamy do czynienia z problemem typu permutation flow shop to algorytm nie nakłada dodatkowych warunków, ewentualnym warunkiem mogłaby być unikalność wcześniej wspomniana unikalność czasów dla zadań danej maszyny
4. Ponieważ musimy przeprowadzić operację obliczania terminów, która ma złożoność obliczeniową $O(n^2)$ r razy to cały algorytm ma złożoność na poziomie $O(n^2(n-1)) = O(n^3)$