

STEROWANIE PROCESAMI DYSKRETNymi				
Numer laboratorium:	I-II	Temat zajęć	Problem przepływowy	
		Data zajęć	22.02.2019, 01.03.2019	
		Termin zajęć	piątek, 18:55	
Wykonawcy			Prowadzący	Ocena
Jakub Pawłowski, 222222			mgr inż. Teodor Niżyński	
Aleksandra Rzeszowska, 234780				

## 1. Klasa Job

Celem realizacji szeregowania zadań w problemie przepływowym została utworzona klasa zawierająca:

- **time\_on\_machine** - tablica czasów wykonania zadania na poszczególnych maszynach,
- **size** - ilość maszyn, na których dane zadanie będzie wykonane,
- **time(self, machine)** - metoda zwracająca czas wykonania zadania na danej maszynie.

## 2. Rozwiązanie

Rozwiązaniem zadania jest wyznaczenie optymalnej kolejności wykonania zadań, ze względu na czas realizacji wszystkich zadań. Dla zadanej kolejności zadań funkcja **c\_max(queue, jobs=[Job])** zwraca czas wykonania  $C_{max}$ .

### 2.1. Przegląd zupełny

Przegląd zupełny polega na wyznaczeniu wartości  $C_{max}$  dla każdej z możliwych kolejności wykonania zadań. Realizacja tej metody opiera się na wygenerowaniu wszystkich permutacji (wykorzystano `list(permutations(jobs_queue))`), a następnie, dla każdej z otrzymanych permutacji, obliczono  $C_{max}$ , wyznaczając jednocześnie najmniejszy otrzymany czas oraz kolejność wykonania zadań.

### 2.2. Algorytm Johnsona

Zadaniem Algorytmu Johnsona jest znalezienie optymalnej kolejności wykonania zadań. Realizacja tego punktu ograniczyła się do wyznaczenia jej w dwóch, opisanych poniżej wariantach.

#### 2.2.1 Wariant dla dwóch maszyn

Zadania zostały podzielone na dwie grupy: te, w których czas wykonania na maszynie drugiej jest większy niż czas wykonania na maszynie pierwszej, oraz pozostałe. Zadania z grupy pierwszej, uszeregowane w kolejności rosnącej względem czasu wykonania na pierwszej maszynie, a następnie zadania z grupy drugiej, posortowane w sposób malejący, według czasu wykonania zadania na drugiej maszynie, tworzą poszukiwaną kolejność.

#### 2.2.2 Wariant dla trzech maszyn

Poszukiwanie optymalnej kolejności w przypadku, gdy zadania powinny być wykonane na trzech maszynach wymaga utworzenia dwóch wirtualnych maszyn. Czas wykonania zadania dla maszyny wirtualnej 1 to suma czasów wykonania zadania na rzeczywistej maszynie 1 oraz 2, zaś czas na wirtualnej maszynie nr 2, to suma czasów wykonania zadania na rzeczywistej maszynie 2 oraz 3. Dla otrzymanych w ten sposób czasów wykonania zadań na wirtualnych maszynach wywołana zostaje funkcja realizująca zadanie wyznaczania optymalnej kolejności za pomocą Algorytmu Johnsona dla 2 maszyn.

## 2.3. Test poprawności

Zaimplementowane funkcje poddane zostały testom poprawności. Wygenerowane zostały zadania realizowane na 2 lub 3 maszynach, a dla nich wyznaczono optymalny czas zarówno wykorzystując przegląd zupełny, jak i algorytm Johnsona. Nie wszystkie przeprowadzone testy przedstawiły ten sam czas działania. Przykładowe dane wejściowe oraz czasy i kolejności zbiera tabela poniżej:

Tabela 1: Wyniki testów

L.p.	czas na maszynie			Przegląd zupełny		Algorytm Johnsona	
	I	II	III	kolejność	$C_{max}$	kolejność	$C_{max}$
1	2	14	-	2,0,1	22	2,0,1	22
	13	5	-				
	1	2	-				
2	1	3	-	0,3,2,1	24	0,3,2,1	24
	9	3	-				
	7	8	-				
	4	8	-				
3	1	3	4	0,3,2,1	27	0,3,1,2	29
	3	5	1				
	5	1	5				
	8	9	3				
4	2	4	5	0,1,2,3	18	0,1,2,3	18
	2	5	3				
	3	4	2				
	1	2	1				
5	5	2	-	4,0,3,5,2,1	24	5,4,3,2,0,1	24
	5	1	-				
	6	2	-				
	4	3	-				
	2	6	-				
	1	7	-				
	1	2	-				

### 2.3.1 Wczytanie instancji ta000

Kolejno zaimplementowano wczytywanie danych tekstowych z pliku *data000.txt*. Dane te umieszczone zostały w pliku w formacie przedstawionym na rysunku obok. Pierwszy wiersz zawiera informacje o nazwie wczytywanej instancji, kolejny ilość zadań i ilość maszyn, oddzielone spacją. W kolejnych wierszach znajdują się czasy wykonania danego zadania na kolejnych maszynach, oddzielone spacją.

```

ta001
10 5
54 79 16 66 58
83 3 89 58 56
15 11 49 31 20
71 99 15 68 85
77 56 89 78 53
36 70 45 91 35
53 99 60 13 53
38 60 23 59 41
27 5 57 49 69
87 56 64 85 13

```

Rysunek 1: Przykładowe dane z pliku

### 2.3.2 Wygenerowanie losowych danych

Losowe dane zostały wygenerowane przez funkcję `random.randint(1,30)` z biblioteki `random`. Funkcja `test_jobs(jobs, machines)` zwraca listę `jobs` zadań wykonywanych na `machines` maszynach. Zwiększając ilość zadań i maszyn przetestowano działanie przeglądu zupełnego oraz algorytmu Johnsona. Wyniki zbiera tabela poniżej.

Tabela 2: Testy na losowych danych

ilość maszyn	ilość zadań	przegląd zupełny	algorytm Johnsona	błąd względny
		$C_{max}$	$C_{max}$	
2	2	45	45	0%
2	3	47	47	0%
2	5	80	80	0%
2	10	170	170	0%
3	2	47	47	0%
3	3	106	106	0%
3	5	130	130	0%
3	10	213	244	14,55%
4	2	76	76	0%
4	3	103	103	0%
4	5	163	167	2,45%
4	10	186	205	10,22%
5	2	87	87	0%
5	3	136	136	0%
5	5	153	173	13,07%
5	10	223	230	2,84%

## 3. Wnioski

Przegląd zupełny pozwala na wyznaczenie najkrótszego czasu wykonywania zadań, ale jest rozwiązaniem o dużej złożoności obliczeniowej ( $O(2^n)$ ). Przegląd zupełny wymaga sprawdzenia wszystkich permutacji zadań (dla  $n$  zadań oznacza to  $n!$  permutacji).

Algorytm Johnsona charakteryzuje się znacznie mniejszą złożonością obliczeniową niż przegląd zupełny, nie zawsze jednak wyznacza najlepsze rozwiązanie (Tabela 1). Błąd względny wyznaczonej wartości dla przykładu z tabeli wynosi  $b = \frac{29-27}{27} \cdot 100\% = 7,41\%$ .

Rozwiązanie generowane przez algorytm Johnsona dla trzech maszyn nie zależy od czasów wykonania kolejnych zadań na maszynie drugiej (w ogólności liczą się tylko czasy wykonania zadań na maszynie pierwszej i ostatniej). Rozumowanie prowadzące do takiego wniosku przedstawiono poniżej:

Tabela 3: Czas wykonania zadań na maszynach

zadanie	$t_{M1}$	$t_{M2}$	$t_{M3}$	$t_{VM1}$	$t_{VM2}$
1	$t_{11}$	$t_{12}$	$t_{13}$	$t_{11} + t_{12}$	$t_{12} + t_{13}$
2	$t_{21}$	$t_{22}$	$t_{23}$	$t_{21} + t_{22}$	$t_{22} + t_{23}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n$	$t_{n1}$	$t_{n2}$	$t_{n3}$	$t_{n1} + t_{n2}$	$t_{n2} + t_{n3}$

Zatem zależność między  $t_{VM1}$  a  $t_{VM2}$  nie zależy od wartości  $t_{M2}$ :

$$t_{VM1} > t_{VM2} \iff t_{M1} > t_{M3}$$

W ten sposób zadania zostają podzielone na dwie grupy niezależnie od czasów wykonania na maszynie 2. Rozumując dalej w ten sposób można wyznaczyć algorytm postępowania dla  $k$  maszyn, gdzie kolejność wykonania zadań będzie zależała od zależności między czasami wykonania kolejnych zadań na pierwszej i ostatniej maszynie.

Wykorzystanie przedstawionego powyżej pomysłu implementacji algorytmu Johnsona wykorzystano w przeprowadzaniu testów na losowych danych dla 4 oraz 5 maszyn (Tabela 2).

Nie zawsze można jednoznacznie wyznaczyć kolejność, dla której otrzymuje się minimalny czas wykonania zadań (przykład 5, Tabela 1), bowiem może istnieć wiele kolejności, dla których czas ten przyjmuje najmniejszą możliwą wartość.

Wczytanie danych z instancji<sup>1</sup> pozwoliło na sprawdzenie poprawności działania algorytmów. Po wczytaniu instancji *ta000* uzyskano czas  $C_{max} = 32$ , taki sam jaki wynika z programu *NEH.demo*, co przemawia za poprawnością zaimplementowanych algorytmów.

---

<sup>1</sup><http://mariusz.makuchowski.staff.iia.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/lab.instrukcje/lab03.neh/neh.demo.v1.2/>