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 Aleksandra Rzeszowska, 234780	mgr inż. Teodor Niżyński	

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 13 1	14 5 2	- - -	2,0,1	22	2,0,1	22
2	1 9 7 4	3 3 8 8	- - - -	0,3,2,1	24	0,3,2,1	24
3	1 3 5 8	3 5 1 9	4 1 5 3	0,3,2,1	27	0,3,1,2	29
4	2 2 3 1	4 5 4 2	5 3 2 1	0,1,2,3	18	0,1,2,3	18
5	5 5 6 4 2 1 1	2 1 2 3 6 7 2	- - - - - - -	4,0,3,5,2,1	24	5,4,3,2,0,1	24

2.3.1 Wczytanie instancji ta000

Kolejno zaimplementowano wczytywanie danych tekstowych z pliku *dane.txt*. Dane te umieszczone zostały w pliku w formacie przedstawionym na rysunku obok. W kolejnych wierszach znajdują się czasy wykonania danego zadania na kolejnych maszynach, oddzielone spacją.

```
1 3 8
9 3 5
7 8 6
4 8 7
```

Rysunek 1: Przykładowe dane z pliku *dane.txt*

2.3.2 Wygenerowanie losowych danych

Może uda mi się jeszcze jutro to ogarnąć, wtedy coś dopiszę, lub usunę punkt...

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 2: 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}	$t_{21} + t_{22}$	$t_{22} + t_{23}$
⋮	⋮	⋮	⋮	⋮	⋮
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.

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[1] 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.

[1] http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/lab.instrukcje/lab03.neh/neh.demo.v1.2/