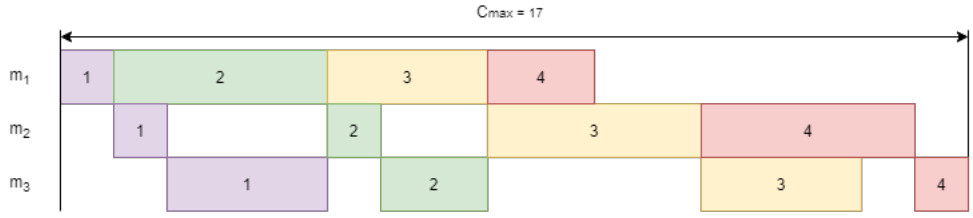
STEROWANIE PROCESAMI CIĄGŁYMI
Problem przepływowy - algorytm NEH

Prowadzący: dr inż. Paweł Drąg
Autorzy:
Emilia Szymańska 248975
Igor Zieliński 248944

30.05.2021r.

1 Opis problemu przepływowego

Problem przepływowy jest problemem wielomaszynowym. Do wykonania dostajemy zbiór zadań $J = (1, ..., n)$ na zbiorze maszyn $M = (1, ..., m)$. Każde z zadań musi po kolei przejść przez wszystkie maszyny od 1 do m tak jak jest to pokazane na rys. 1.



Rysunek 1: Przykładowe ułożenie zadań J=(1, 2, 3, 4) na maszynach M = (1, 2, 3).

Na wejściu podane są trzy parametry:

- $J = (1, ..., n)$ - zbiór zadań,
- $M = (1, ..., m)$ - zbiór maszyn
- p - macierz czasów p_{ij} wykonywania zadania J_i na maszynie M_j.

Wyjścia działania algorytmu obsługującego ten problem mogą być różnie zdefiniowane - może to być np. jest permutacja wykonania zadań π oraz moment maksymalny z możliwych terminów dostarczenia zadań C_{max}. Zaimplementowany został algorytm NEH do rozwiązania tego problemu.

2 Algorytm NEH

2.1 Opis teoretyczny, implementacja i przykład działania

Algorytm jest stosunkowo prosty i jest opisany w poniższej listy korków.

1. Posortuj zadania malejąco względem czasu ich wykonywania (po sumie czasów wszystkich operacji wymaganych do skończenia zadania na wszystkich maszynach).
2. Dopóki lista z nieposzeregowanymi zadaniami nie jest pusta:
   (a) Wybierz z listy zadanie, którego czas wykonywania jest najdłuższy.
   (b) Ułóż to zadanie na pozycję wśród zadań już wybranych tak, by całkowite C_{max} było jak najmniejsze (wymagany jest przegląd wszystkich ułożeń tego zadania i obliczenia C_{max}).

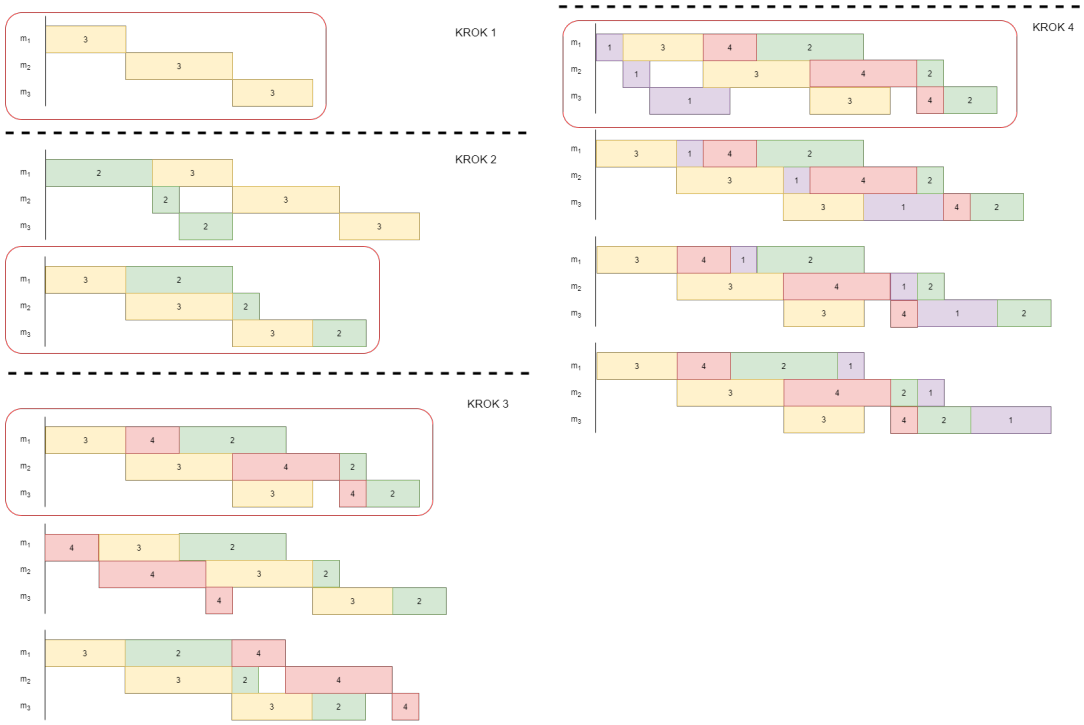
Implementacja algorytmu znajduje się poniżej.

Listing 1: Implementacja algorytmu NEH.

```
std::pair<int, std::vector<int> > NEH_algorithm(const Matrix<int>& M){
    std::vector<std::pair< int, int> > tasks;
    for(int i = 0 ; i < M.size().first; i++){
            int sum = 0;
            for(int j = 0; j < M.size().second; j++)
                    sum += M(i, j);
            tasks.push_back({-sum, i});
    }
    std::sort(tasks.begin(), tasks.end());
    std::vector<int> pi;
    int cmax = 0;
    for(int i = 0; i < M.size().first; i++){
            pi.push_back(tasks[i].second);
            int cmax_tmp = evaluate(M, pi);
            int index = pi.size() - 1;
            for(int j = pi.size() - 2; j >= 0; j--) {
                    std::swap(pi[j], pi[j + 1]);
                    int c = evaluate(M, pi);
                    if(c < cmax_tmp){
                            cmax_tmp = c;
                            index = j;
                    }
            }
            for(int j = 0; j < index; j++)
                    std::swap(pi[j], pi[j + 1]);
            cmax = cmax_tmp;
    }
    return {cmax, pi};
}
```

```
int evaluate(const Matrix<int>& M,  const std::vector<int>& pi){
    std::pair<int, int> p = M.size();
    std::vector<int> last_endings(p.first, 0);
    int cmax = 0;
    for(unsigned int i = 0; i < p.second; i++){
            int t = 0;
            for(unsigned int j = 0; j < pi.size(); j++){
                    t = std::max(t, last_endings[pi[j]]) + M(pi[j], i);
                    last_endings[pi[j]] = t;
            }
            cmax = std::max(cmax, t);
    }
    return cmax;
}
```

Na rys. 2 przedstawione zostało przykładowe działanie algorytmu NEH dla przypadku czterech zadań i trzech maszyn. W kolejnych krokach wybierane są kolejne najlepsze uszeregowania (zaznaczone czerwoną ramką).



Rysunek 2: Przykładowe działanie algorytmu NEH.

3 Wyniki i ich porównanie

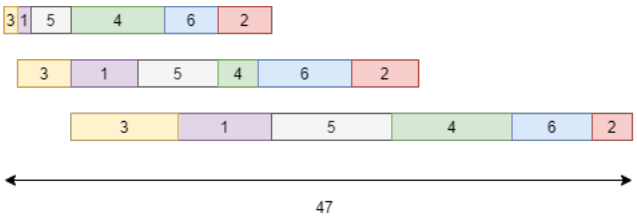
Dane testowe pobrane były ze strony `http://new.zsd.iiar.pwr.edu.pl/educ.php?lid=132&zid=NEH`.

3.1 Przykładowy wynik algorytmu

Na podstawie pierwszego zestawu testowego zostanie przedstawione działanie algorytmu opisanego w sekcji 2. Dane tekstowe przedstawiają się wówczas następująco:
6 3
1 5 7
4 5 3
1 4 8
7 3 9
3 6 9
4 7 6

Pierwsza linia zawiera dwie liczby całkowite: n oznaczającą liczbę zadań oraz m oznaczającą liczbę maszyn. Każda z następnych n linii zawiera m liczb, gdzie i-ta liczba w wierszu k oznacza czas wykonania zadania k na maszynie i.

Po przepuszczeniu przez algorytm NEH otrzymujemy poszeregowanie jak na rys. 3 odpowiadające kolejności zadań: 3 1 5 4 6 2 oraz wartości $C_{max} = 47$.



Rysunek 3: Uszeregowane zadania z pierwszego zbioru danych przez algorytm NEH.

W przypadku drugiego zestawu danych:
10 5 84 53 77 29 39
95 63 57 68 69
26 85 15 73 74
50 8 7 5 52
4 11 92 94 77
9 69 51 37 4
57 71 33 49 12
35 55 27 1 17
79 10 84 36 1
17 91 64 62 21

Algorytm NEH zwraca następującą kolejność wykonywania zadań: 5 3 2 9 10 1 7 4 6 8. W takim wypadku otrzymujemy $C_{max} = 637$.

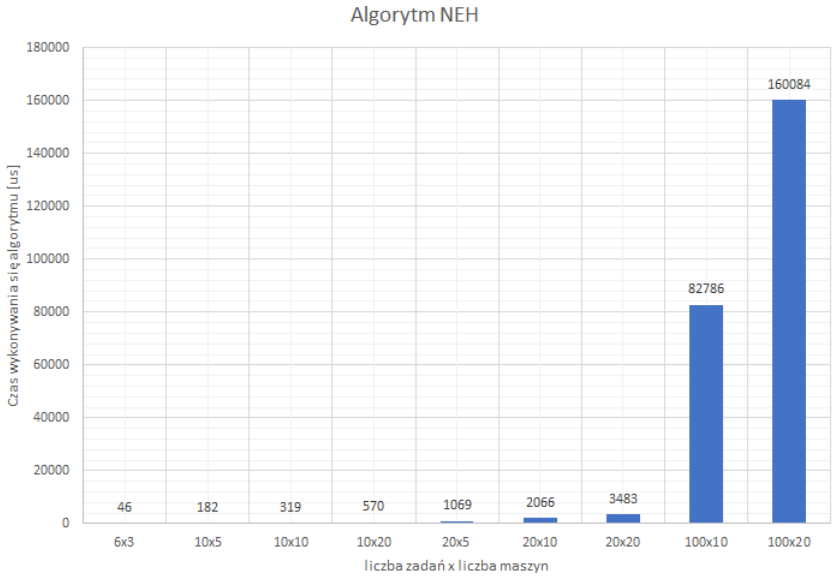
3.2 Wyniki na danych testowych

W tabeli 1 przedstawione zostały wartości otrzymanych C_{max} oraz czas wywoływania algorytmu dla poszczególnych zestawów danych, a na wykresie 4 porównano czasy wykonywania się algorytmów (z dokładnością do mikrosekund).

zestaw testowy	ilość zadań w zestawie (n x m)	C_{max}	czas działania programu [μs]
1	6x3	47	46
2	10x5	637	182
3	10x10	1163	319
4	10x20	1805	570
5	20x5	1137	1069
6	20x10	1471	2066
7	20x20	2397	3483
8	100x10	5948	82786
9	100x20	6404	160084

Tabela 1: Wynik działania algorytmu NEH.

Zestaw nr 1: 3 1 5 4 6 2
Zestaw nr 2: 10 5 2 8 1 9 6 4 3 7
Zestaw nr 3: 5 2 10 8 9 3 7 1 6 4
Zestaw nr 4: 6 2 1 4 7 9 3 8 5 10
Zestaw nr 5: 13 9 7 8 3 17 4 20 12 5 16 1 14 19 18 6 2 10 11 15
Zestaw nr 6: 6 17 4 14 13 9 5 18 1 7 12 8 3 10 2 16 15 11 19 20
Zestaw nr 7: 20 6 11 18 1 16 2 12 14 17 19 9 13 4 8 3 15 5 7 10
Zestaw nr 8: 58 30 1 6 37 61 68 26 81 77 4 22 87 56 38 83 95 57 11 62 65 69 20 71 35 96 79 52 54 40 80 18 82 46 45 78 89 17 24 51 97 19 44 98 50 91 16 34 2 100 55 53 27 15 60 13 63 94 29 76 25 36 48 42 31 3 64 33 86 28 99 10 90 39 49 32 47 59 73 67 23 75 92 66 85 74 88 93 12 72 5 41 84 21 8 7 9 14 70 43
Zestaw nr 9: 48 40 61 68 6 9 96 34 95 12 65 25 78 77 30 35 16 59 74 51 90 36 79 55 82 66 70 60 71 2 81 19 98 92 97 58 38 99 62 86 91 83 10 32 54 100 57 72 1 73 49 47 21 45 85 89 29 64 33 18 5 80 31 53 22 94 76 8 52 4 75 43 84 28 20 50 93 46 56 69 26 42 13 63 17 44 23 11 88 39 14 27 87 15 67 41 37 3 7 24



Rysunek 4: Przedstawienie czasu działania algorytmu.

4 Wnioski

- Obliczeniowa złożoność algorytmu wynosi $O(n^3)$, zatem przy używaniu go dla większej liczby maszyn oraz zadań czas wykonywania się algorytmu może być długi.
- Nie da się porównać tego algorytmu z poprzednimi ze względu na specyfikę problemu - mamy tu problem wielomaszynowy.
- Na podstawie otrzymanych wyników można stwierdzić, że x-krotne zwiększenie liczby maszyn nie wpływa na czas wykonywania się algorytmu tak jak x-krotne zwiększenie liczby zadań. Zwiększenie liczby zadań ma znacznie większy wpływ na czas wykonywania się algorytmu. Szczególnie to widać przy porównaniu czasów dla przypadków 10x5 (182 μs), 10x10 (319 μs) a 20x5 (1069 μs).