Projektowanie Efektywnych Algorytmów

Kierunek *Informatyka*	**Termin** *Piątek 13:15*
Temat *Przegląd zupełny (Brute Force)*	**Problem** *WST*
Skład grupy *230504 Jeremiasz Romejko*	**Nr grupy**
Prowadzący *Mgr inż. Radosław Idzikowski*	**data** *15 listopada 2018*

1 Opis problemu

Kryterium Ważonej Sumy Opóźnień dla jednomaszynowego problemu szeregowania zadań*(Weighted Sum Tardiness for task scheduling)*. W problemie mamy zbiór n zadań wykonywanych na jednej maszynie.

$$J = J_1, J_2, ...J_i, ..., J_n, \tag{1}$$

każde i-te zadanie składa się z trzech parametrów:

- p_i - czas wykonania,
- w_i - współczynnik kary,
- d_i - żądany termin zakończenia.

Każde zadanie musi być wykonywane nieprzerwanie przez p_i oraz powinno być ukończone przed terminem d_i , w przeciwnym wypadku zostanie naliczona kara. Naraz może być wykonywane tylko jedynie jedno zadanie. Przez π będziemy oznaczać kolejność wykonywania zadań. Dla każdego zadania należy obliczyć jego spóźnienie T_i:

$$T_i = max(C_i - d_i, 0) \tag{2}$$

gdzie C_i jest to moment zakończenia i-tego zadania, a wcześniejsze zakończenie nie jest dodatkowo premiowane. Kryterium optymalizacyjnym jest suma $w_i T_i$:

$$\sum_{i=1}^{n} w_i T_i \tag{3}$$

2 Metoda rozwiązania

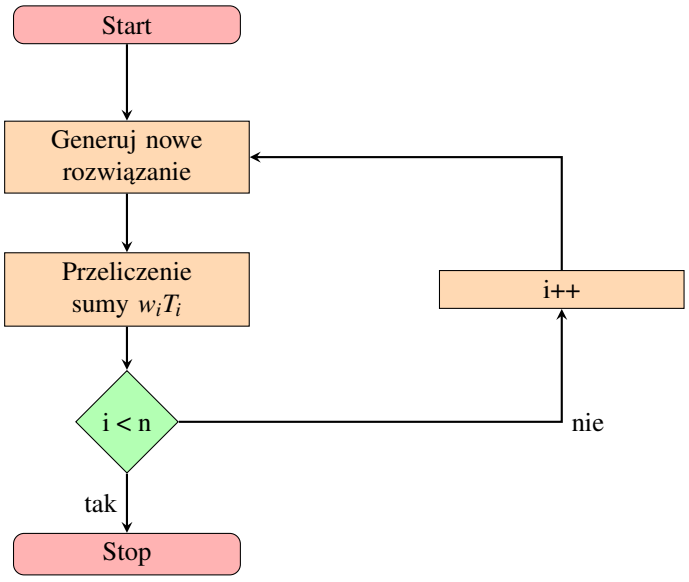
2.1 Algorytm 1

Algorytm ma złożoność obliczniową $O(n!)$ ponieważ wykorzystuje się w nim rekurencje do otrzymania kolejnych permutacji. Dla każdej kolejnej permutacji wyliczane jest kryterium optymalizacyjne i jeżeli jest ono lepsze od poprzedniego rezultatu zostaje zapisane a wynik π zostaje wyswietlony na ekranie. Poniższy kod prezentuje działanie tego algorytmu w języku $C++$.

Listing 1: Algorytm przeglądu zupełnego

```
1  int Machine::countResult(Task tasks[])
2  {
3      int result=0;
4      int time=0;
5      for (int i = 0; i < this->size; i++)//dla wszystkich zadan
6      {
7          if (tasks[i].deadLine - tasks[i].executionTime - time >= 0)
8          //jezeli zadanie wykonane w czasie
9          {
10             result += 0;
11         }
12         else
13         {
14             result+=(tasks[i].deadLine - tasks[i].executionTime - time)*
15             (-1 * tasks[i].retioPunishment);
16             //jezeli nie
17         }
```

```
18
19          time += tasks[i].executionTime;
20      }
21      return result;
22  }
23  void Machine::bruteforce(int start, int size)
24  {
25      if (start == size)//jezeli permutacja gotowa
26      {
27          if (result >= countResult(task))//jesli jest lepsza od poprzedniej
28          {
29              result = countResult(task);//zapisz wynik i wyswietl
30              showResult();
31          }
32      }
33      else
34      {
35          for (int i = start; i < size; i++)
36          {
37              swap(start, i);//zamien miejscami i-ty element ze startowym
38              bruteforce(start + 1, size);//wywolaj dla nastepnego elementu
39              swap(start, i);//zamien miejscami i-ty element ze startowym
40          }
41      }
42  }
```



Rysunek 1: Schemat algorytmu

3 Eksperymenty obliczeniowe

Obliczenia zastały wykonane na komputerze klasy PC z procesorem AMD Athlon II X4 620, kartą graficzną NVIDIA GeForce GTS 250, 8GB RAM i DYSK SSD. Dla danej instancji został zmierzony czas znalezienia optymalnego rozwiązania t.Dla każdego n eksperyment powtórzono 10 razy i wyliczono z sumy t średnią arytmetyczną:

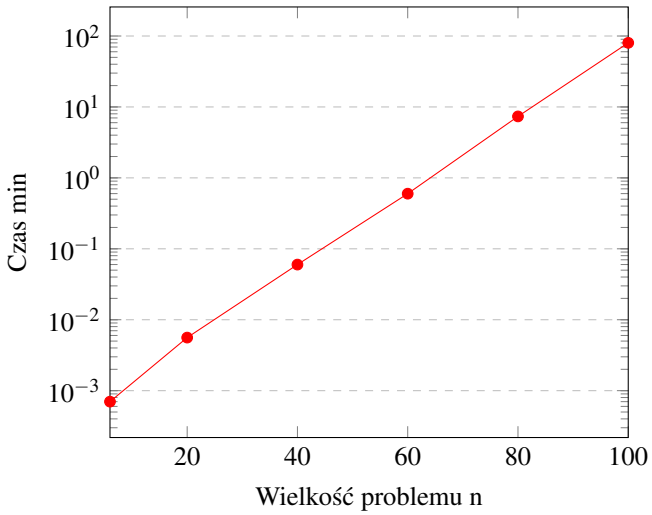
$$T = \frac{\sum_{i=1}^{n} t_i}{10} \tag{4}$$

Wszystkie wyniki zebrano i przedstawiono w tabeli nr 1 gdzie:

- n - liczba zadań,
- T - średni czas znalezienia optymalnego π

Tablica 1: Czas obliczeń dla ustalonej liczby instancji.

n	T
7	0.0007
8	0.0056
9	0.0599
10	0.5986
11	7.3530
12	119.72



Rysunek 2: Czas wykonywania się algorytmów w zależności od wielkości problemu w skali logarytmicznej

4 Wnioski

Jak można zaobserwować na rysunku 2 funkcja rośnie z złożonośćią $O(n!)$ co jest skutkiem użycia rekurencji w algorymie brute force. Wartości zadań nie mają znaczenia dla czasu wykonania jednak wielkość instancji jest kluczowa dla prędkości działania algorytmu. Algorytm jest powolny ale dokładny zanjduje wszystkie optymalne rozwiązania.