Wrocław, 29 kwietnia 2021

Sterowanie **P**rocesami **D**yskretnymi

Laboratorium 4

Opracowanie i implementacja algorytmów dla problemu $1||\sum w_i T_i$

prowadzący: *mgr inż. Radosław Idzikowski*

1 Cel laboratorium

Celem laboratorium jest zapoznanie się z podstawami teorii szeregowania zadań oraz sposobami modelowania oraz rozwiązywania podestowych problemów na przykładzie jednomaszynowego problemu z ważoną sumą opóźnień (**T**otal **W**eighted **T**ardiness) . Obejmuje to odpowiednie zdefiniowane problemu (ograniczeń i funkcji celu), danych wejściowych oraz implementacje dedykowanych algorytmów, a także interpretację wyników.

2 Przebieg zajęć

Laboratorium obejmuje zajęcia nr 8-9 (4 godziny zajęć w tym oddanie). Praca odbywa się w ramach dwuosobowych zespołów. Każdy zespół otrzymuje do opracowania ten sam problem. W tym wypadku $1||\sum w_i T_i$ oraz dedykowane dla niego algorytmy.

W trakcie zajęć nr 8 w wybranym języku `python`, `C/C++`, `Java` lub `C#` należy zaimplementować metodę generowania instancji przy użyciu załączonych generatorów dla zadanych parametrów (źródło, rozmiar, zakres). Kolejnym krokiem jest implementacja metody oceniania rozwiązania (liczenia funkcji celu). Następnie należy zacząć implementacje wszystkich algorytmów.

Podczas ostatnich zajęć z tego tematu (zajęcia nr 9) będzie czas na dokończenie rozpoczętych zadań oraz na ocenę efektu końcowego. Na ocenę `3.0` należy poprawnie zaimplementować algorytm zachłanny oraz przegląd zupełny dla badanego problemu. Na oceną `5.0` należy napisać algorytm programowania dynamicznego w wersji rekurencyjnej lub iteracyjnej dla badanego problemu. W celu uzyskania pełnej oceny `5.0` należy metodą *backtrackingu* odtworzyć kolejność wykonywania zadań. Ponadto implementacja na na niskim poziomie efektywności czasowej może mieć wpływ na ocenę.

3 Problem

Problem $1|w_j, d_j|\sum w_j T_j$ możemy zapisać w skrócie $1||\sum w_j T_j$, ponieważ parametry problemu wynikają z kryterium optymalizacyjnego. Mamy zbiór n zadań wykonywanych na maszynie.

$$\mathcal{J} = \{1, 2, \ldots, n\}, \tag{1}$$

każde j-te zadanie składa się z trzech parametrów:

- p_j - czas wykonania (*performed time*),
- w_j - waga (*weight*)/współczynnik kary,
- d_j - żądany termin zakończenia (*deadline*).

Każde zadanie musi być wykonywane nieprzerwanie przez $p_{\pi_{(j)}}$ czasu na maszynie. Naraz może wykonywać się tylko jedno zadanie. Każde zadanie powinno być ukończone przed jego żądanym terminem ukończenia $d_{\pi_{(j)}}$, w przeciwnym wypadku zostanie naliczona kara.

Musimy ustalić harmonogram. Zadania nie wymagają przygotowania. więc możemy zacząć od razu wykonywać pierwsze zdanie: $S_{\pi(1)} = 0$. W omawianym problemie zachowujemy ciągłość pracy maszyny, ponieważ wszystkie zdania są dostępne od początku. To moment rozpoczęcia kolejnego zadania jest zawsze momentem zakończenia poprzedniego:

$$S_{\pi(j)} = C_{\pi(j-1)} \tag{2}$$

Dla każdego zadania należy obliczyć jego spóźnienie $T_{\pi(j)} \geqslant 0$, pamiętając że nie premiujemy wcześniejszego wykonania zadania, więc wartośc nie może być ujemna:

$$T_{\pi(j)} = max(C_{\pi(j)} - d_{\pi(j)}, 0) \tag{3}$$

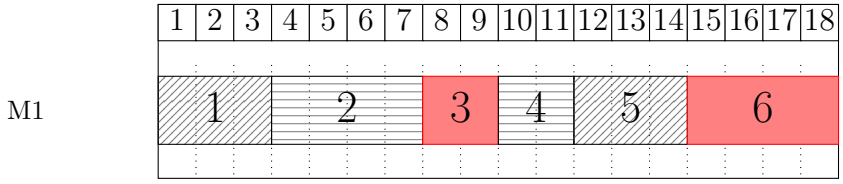
Badanym kryterium optymalizacyjnym jest ważona suma opóźnień wszystkich zadań:

$$F(\pi) = \sum_{j \in \mathcal{J}}^{n} w_{\pi(j)} T_{\pi(j)} \tag{4}$$

Szukamy takiego uszeregowania π aby wartość funkcji $F(\pi)$ byłą jak najmniejsza.

4 Przykład

Na Rysunku 1 mamy graficzne rozwiązanie problemu dla danych z Tabeli 1. Zadnie jest reprezentowane w formie bloczku. Zadania spóźnione zaznaczono na czerwono. Na schemacie dobrze widać ciągłą pracy maszyny (bez przestojów) oraz stały czas zakończenia wykonywania wszystkich zadań niezależnie od ich kolejności. Rozwiązanie w formie harmonogramu (S, C) przedstawiono w Tabeli 2.



Rysunek 1: Schemat gantta dla instancji z Tabeli 1 oraz $\pi = (1, 2, 3, 4, 5, 6)$

Tabela 1: Czasy wykonywania, wagi oraz żądane terminy zadań dla instancji o rozmiarze $n = 6$

j	p_j	w_j	d_j
1	3	3	3
2	4	2	10
3	2	1	6
4	2	2	15
5	3	4	21
6	4	2	16

Tabela 2: Harmonogramu dla instancji z Tabeli 1 oraz $\pi = (1, 2, 3, 4, 5, 6)$

j	1	2	3	4	5	6
S_j	0	3	7	9	11	14
C_j	3	7	9	11	14	18
T_j	0	0	**3**	0	0	**2**

$$F(\pi) = 0 \cdot 3 + 0 \cdot 2 + \mathbf{1} \cdot \mathbf{3} + 0 \cdot 2 + 0 \cdot 4 + \mathbf{2} \cdot \mathbf{2} = 7 \tag{5}$$

5 Sposób generowania instancji

Dla parametru n oraz ziarna Z:

1. init(Z).
2. Dla każdego $j \in \mathcal{J}$: $p_j \leftarrow \text{nextInt}(1, 29)$.
3. $A \leftarrow \sum_{j=1}^{n} p_i$.
4. Dla każdego $j \in \mathcal{J}$:$w_j \leftarrow \text{nextInt}(1, 9)$.
5. Dla każdego $j \in \mathcal{J}$:$d_j \leftarrow \text{nextInt}(1, X)$.

W ramach testu należy przyjąć najpierw $X = A$, a potem $X = 29$. W ramach oceny za implementacje algorytmów, należy sprawdzić czy zakres parametrów d_j ma wpływ na działanie algorytmów.

6 Metody rozwiązania

6.1 Przegląd zupełny

Sprawdzenie wszystkich możliwych kombinacji (*Brute Force*). Dla zbioru $\{1, 2, 3\}$ mamy $n!$ kombinacji: $(1, 2, 3)$, $(1, 3, 2)$, $(2, 1, 3)$, $(2, 3, 1)$, $(3, 1, 2)$ i $(3, 2, 1)$.

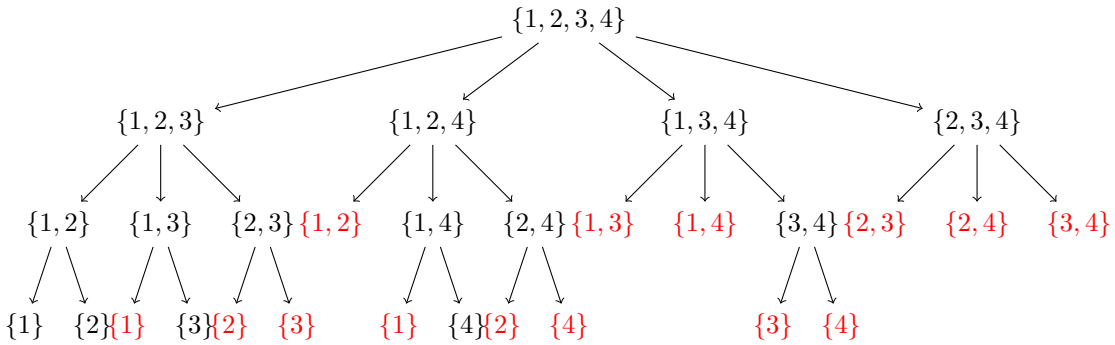
6.2 Metoda zachłanna

Sortowanie po żądanych terminach zakończenia. Złożoność zależy od zastosowanego sortowania.

6.3 Programowanie dynamiczne

Metoda polega na dzieleniu problemu na mniejsze podproblemy oraz zapamiętywaniu ich wyniku, dzięki czemu potem zostanie obliczona wartość funkcji celu dla całego problemu. Dzielenie problemów na mniejsze pokazano na Rysunku 2. Kolorem czerwonym zaznaczono powtarzające się podproblemy, dla których wartość można odczytać z pamięci.

W celu uproszczenia dostępu do wartości funkcji celu dla wcześniej policzonych poproblemów warto dobrać odpowiednią reprezentację podproblemu. Jeśli opiszmy zbiór za pomocą jednej liczby całkowitej $\mathcal{D}$, a każdy bit ustawiony na wartość "1" lub "0" odpowiednio będzie oznaczał, że zadanie zawiera się lub nie zawiera się w danym podproblemie. Przykładowo dla problemu o rozmiarze $n = 4$, mamy podproblem opisany liczbą `10` w systemie dziesiętnym, to zadanie 4 i 2 są w rozpatrywanym podproblemie, ponieważ ($10_{dec} = 1010_{bin}$). Wtedy wartości funkcji celu dla danych problemów możemy przechować w pamięci `memory` o rozmiarze równym liczbie wszystkich podproblemów $2^n - 1$. Ponadto dostęp do wybranej komórki pamięci będzie w czasie stałym $O(1)$.



Rysunek 2: Podział problemu na mniejsze podproblemy

Algorithm 1 Programowanie Dynamiczne

1: $\mathcal{D} \leftarrow \mathcal{J}$
2: Inicjalizacja tablicy `memory` o rozmiarze $2^n - 1$ wartościami -1
3: **procedure** PD($\mathcal{D}$)
4: `sum`$\leftarrow \sum_{j \in \mathcal{D}} p_j$
5: **return** $\min_{j \in \mathcal{D}}(\max\{\texttt{sum} - d_j, 0\}w_j + F(D \setminus \{j\}))$
6: **end procedure**

Problem możemy rozbijać zgodnie z Rysunkiem 2 przeglądając rekurencyjnie od góry. Ka każdym poziomie musimy w pierwszej kolejności policzyć sumę wszystkich czasów wykonywania zadań w podproblemie. Następnie trzeba kolejno przyjąć każde zadanie jako ostatnie, policzyć dla niego karę za spóźnienie oraz uwzględnić minimum z poziomu poniżej bez uwzględnienia tego zadania. Ponadto przed każdym wywołaniem rekurencji należy sprawdzić czy dla danego podproblemu nie znamy już wyniku (patrz wzór (6)). Schemat metody pokazano w Algorytmie 1

$$F(\mathcal{D}) = \begin{cases} \texttt{memory}(\mathcal{I}) & \text{jeśli } \texttt{memory}(\mathcal{D}) \neq -1, \\ \text{PD}(\mathcal{D}) & \text{w przeciwnym wypadku.} \end{cases} \tag{6}$$

Jeśli to opisu zbioru posłużono się liczbą całkowitą $\mathcal{D}$, to w łatwy sposób można napisać wersję iteracyjną algorytmu. Korzystając wiedzy na temat binarnej reprezentacji liczby, można łatwo zauważyć, że w przypadku generowania kolejnych liczb będziemy gwarancję, że wszystkie wcześniejsze podproblemy zostały już policzone. $1_{dec} = 0001_{bin}$, $2_{dec} = 0010_{bin}$, $3_{dec} = 0011_{bin}$ itd. Dla ułatwienia należy zarezerwować pamięć `memory` o rozmiarze 2^n uwzględniając wartość "0". Wersje iteracyjną Programowania Dynamicznego pokazano w Algorytmie 2.

Algorithm 2 Programowanie Dynamiczne 2

1: **procedure** PD($\mathcal{J}$)
2: Inicjalizacja tablicy `memory` o rozmiarze 2^n wartościami -0
3: **for** $\mathcal{D} = 1, 2, \ldots, 2^n$ **do**
4: `sum`$\leftarrow \sum_{j \in \mathcal{D}} p_j$
5: `memory`$(\mathcal{D}) \leftarrow \min_{j \in \mathcal{D}}(\max\{\texttt{sum} - d_j, 0\}w_j + \texttt{memory}(\mathcal{D} \setminus \{j\}))$
6: **end for**
7: **end procedure**

Poniżej rozpisano szczegółowo wszystkie kroki dla problemu o rozmiarze $n = 3$.

0. $F(000) = 0$
1. $F(001) = max\{p_1 - d_1, 0\} * w_1 + F(000)$
2. $F(010) = max\{p_2 - d_2, 0\} * w_2 + F(000)$
3. $F(011) = min \begin{cases} max\{p_1 + p_2 - d_1, 0\} * w_1 + F(010) \\ max\{p_1 + p_2 - d_2, 0\} * w_2 + F(001) \end{cases}$
4. $F(100) = max\{p_3 - d_3, 0\} * w_3$
5. $F(101) = min \begin{cases} max\{p_1 + p_3 - d_1, 0\} * w_1 + F(100) \\ max\{p_1 + p_3 - d_3, 0\} * w_3 + F(001) \end{cases}$
6. $F(110) = min \begin{cases} max\{p_2 + p_3 - d_2, 0\} * w_2 + F(100) \\ max\{p_2 + p_3 - d_3, 0\} * w_3 + F(010) \end{cases}$
7. $F(111) = min \begin{cases} max\{p_1 + p_2 + p_3 - d_1, 0\} * w_1 + F(110) \\ max\{p_1 + p_2 + p_3 - d_2, 0\} * w_2 + F(101) \\ max\{p_1 + p_2 + p_3 - d_3, 0\} * w_3 + F(011) \end{cases}$

W celu poprawienia wydajności programu warto skorzystać z operacji bitowych. Dla reprezentacji zbioru w formie jednej liczny dziesiętnej można sprawdzić w czasie stałym $O(1)$ czy dane zadanie jest w zbiorze za pomocą odpowiedniej operacji bitowej. Operacje bitowe wypisano w Tabeli 3.

Tabela 3: Operacje bitowe

operacja	opis
x `|` y	bitowa alternatywa x i y
x `^` y	bitowa różnica symetryczna x i y
x `&` y	bitowa koniunkcja x i y
x `<<` n	przesunięcie bitowe x w lewo o n bitów
x `>>` n	przesunięcie bitowe x w prawo o n bitów
`~` x	uzupełnienie bitowe x

opracował: *Radosław Idzikowski*