AISD, zadanie 7 z listy 2

Wiktor Kuchta

Sprowadzenie do wspólnego porządku wykonywania zadań przez A i B

Weźmy optymalny układ zadań. Ponumerujmy zadania w ich kolejności wykonywania przez B. Jeśli $i < j$ i zadanie j jest wykonywane bezpośrednio przed i w A, to możemy zamienić ich kolejność wykonywania przez A (na osi czasu początek i na początek j, koniec j na koniec i), otrzymując inny optymalny układ zadań. Taka operacja zmniejsza liczbę inwersji, więc po skończonej ich liczbie maszyny A i B będą wykonywać zadania w tej samej kolejności.

Czas nieaktywności maszyny B

Jeśli mamy kolejność wykonywania zadań $(1, 2, \ldots, n)$, to A może je wykonywać bez przerw — każde najwcześniej, jak to możliwe. Maszyna B niekoniecznie będzie mogła działać bez przerw. Niech X_i to czas nieaktywności maszyny B bezpośrednio przed wykonaniem zadania i. Zauważmy, że

$$X_i = \max\left\{0,\ \sum_{k=1}^{i} a_k - \sum_{k=1}^{i-1} b_k - \sum_{k=1}^{i-1} X_k\right\},$$
$$\sum_{k=1}^{i} X_k = \max\left\{\sum_{k=1}^{i-1} X_k,\ \sum_{k=1}^{i} a_k - \sum_{k=1}^{i-1} b_k\right\}.$$

Podwyrażenie $\sum_{k=1}^{i} a_k - \sum_{k=1}^{i-1} b_k$ nazwijmy K_i, wtedy rozwijając rekurencję we wzorze na sumę X_k otrzymujemy

$$\sum_{k=1}^{n} X_k = \max\{K_1, K_2, \ldots, K_n\}.$$

Optymalne rozwiązanie minimalizuje powyższą wielkość.

Zauważmy, że zamiana kolejności zadań j i $j+1$ może zmienić K_u tylko dla $u \in \{j, j+1\}$, dając pewne nowe K'_j i K'_{j+1}. Zatem sumaryczny czas wykonywania po takiej zamianie może się zwiększyć tylko, gdy

$$\begin{aligned}
0 &> \max\{K_j, K_{j+1}\} - \max\{K'_j, K'_{j+1}\} \\
&= \left(\max\{K_j, K_{j+1}\} - \sum_{i=1}^{j+1} a_i + \sum_{i=1}^{j-1} b_i)\right) - \left(\max\{K'_j, K'_{j+1}\} - \sum_{i=1}^{j+1} a_i + \sum_{i=1}^{j-1} b_i\right) \\
&= \max\{-a_{j+1}, -b_j\} - \max\{-a_j, -b_{j+1}\} = \min\{a_j, b_{j+1}\} - \min\{a_{j+1}, b_j\}.
\end{aligned}$$

Warunek na rozwiązanie optymalne

Twierdzimy, że jeśli porządek zadań $(1, 2, \ldots, n)$ spełnia

$$\min\{a_i, b_j\} \leqslant \min\{a_j, b_i\}, \quad (1 \leqslant i < j \leqslant n) \tag{1}$$

to jest optymalny. Ten porządek będzie można otrzymać podobnie jak wcześniej z dowolnego innego ciągu zadań bez utraty czasu: zamieniając sąsiednie zadania tworzące inwersję, zmniejszając liczbę inwersji. Dokładniej, jeśli $i < j$ i w jakimś porządku mamy zadanie j bezpośrednio przed i, to po ich zamianie sumaryczny czas może się zwiększyć tylko, jeśli $\min\{a_j, b_i\} - \min\{a_i, b_j\} < 0$. Ale to przeczy (**??**).

Algorytm zachłanny rozwiązujący problem

Zbudujmy listę n-elementową z wolnymi polami. Będziemy kolejno wpisywać numer zadania na pierwsze lub ostatnie wolne pole.

Weźmy najkrótszy ze wszystkich czasów zadań. Jeśli jest on dla maszyny A, to dodajmy odpowiadające mu zadanie na pierwsze wolne pole i zapomnijmy o tym zadaniu. Jeśli jest on dla maszyny B, to dodajmy to zadanie na ostatnie wolne pole i zapomnijmy o tym zadaniu. Powtarzamy to, aż skończą się zadania.

Jeśli najkrótszy czas to a_i, to zadanie i wpisujemy na pierwsze wolne pole, przed każdym później wpisanym zadaniem j i spełnione będzie $\min\{a_i, b_j\} \leqslant \min\{a_j, b_i\}$. To samo zachodzi, gdy najkrótszy czas to b_j i zadanie j wpisujemy na ostatnie pole, po każdym poźniej wpisanym zadaniu i. Zatem algorytm wpisuje każde zadanie w dobrej kolejności względem zadań wpisanych później. W każdej parze zadań któreś z nich było dodane wcześniej, więc zachodzi (**??**) i porządek zwrócony przez algorytm jest optymalny.

Praktyczna implementacja może korzystać z dwóch posortowanych rosnąco list czasów zadań A i B, gdzie każdy czas posiada informacje, jakiemu zadaniu odpowiada. Przechodzimy po tych listach podobnie jak w procedurze *merge*, wybierając mniejszą z głów dwóch list, ale musimy też sprawdzać (spamiętując w jakiejś tablicy), czy któreś zadanie już wcześniej wpisaliśmy do tablicy wynikowej i powinniśmy pominąć. Wpisywanie do tablicy wynikowej oczywiście wymaga trzymania dwóch wskaźników, do pierwszego i ostatniego wolnego w niej miejsca.

Jeśli otrzymujemy czasy zadań w kolejności rosnącej to złożoność czasowa i pamięciowa to $\Theta(n)$, w przeciwnym wypadku musimy na początku te czasy sami posortować w czasie $O(n \lg n)$.

Literatura

[1] S. M. Johnson *Optimal two- and three-stage production schedules with setup times included*