Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

Sterowanie Procesami Dyskretnymi

Oliwier Woźniak, Dzmitry Mandrukevich

kierunek studiów: Automatyka i robotyka

specjalność: Robotyka

Algorytm Schrage

Prowadzący: dr inż. Radosław Grymin

Wrocław 17 kwietnia 2024

Spis treści

1 Opis problemu

Celem zadania jest znalezienie algorytmu, znajdującego optymalną permutację zbioru zadań, dla których suma czasów wykonywania jest minimalna. Każde z zadań ma 3 parametry: r, p oraz q. Pierwszy z nich oznacza czas przygotowania zadania, p to czas przez jaki zadanie będzie wykonywane, natomiast q to czas stygnięcia, czyli ile zadanie musi czekać przed jego zakończeniem.

1.1 Algorytm Schrage

To rozwiązanie jest oparte na prostym założeniu, zadania będą ułożone optymalnie, jeśli pierwsze zostaną wykonane zadania z małym „r" i dużym „q", a ostatnie zadania z dużym „r" i małym „q". Takie podejście pozwala w sposób deterministyczny ułożyć zadania w sposób zbliżony do optymalnego. Do programowego rozwiązania tego problemu potrzebne są dwie kolejki priorytetowe, pierwsza ustawia elementy według **najmniejszego** „r", a druga według **największego** „q". W pierwszym przebiegu programu wszystkie zadania wejściowe dodawane są do pierwszej kolejki, co sortuje je według najmniejszego czasu oczekiwania. Następnie zadania gotowe do wykonania (których „r" jest większe niż obecny czas) przerzucane są do drugiej kolejki. Zatem w drugiej kolejce mamy zadania o małym „r", ułożone od największego „q", więc zgodnie z założeniem działania algorytmu, pierwszy element tej kolejki powinien być następnym zadaniem. Program powtarza ten cykl do momentu aż obie kolejki będą puste.

Dane: n – liczba zadań,
r_i – termin dostępności zadania i,
p_i – czas wykonania zadania i,
q_i – czas dostarczenia zadania i.
Szukane : π - permutacja wykonania zadań na maszynie,
C_{max} – maksymalny z terminów dostarczenia zadań.
Struktury pomocnicze :
t – chwila czasowa,
k – pozycja w permutacji π,
N – zbiór zadań nieuszeregowanych,
G – zbiór zadań gotowych do realizacji.

Algorytm **S** (Schrage)

1. $t=0, k=0, C_{max}=0, G=\emptyset, N=\{1,2,...,n\}$,
2. **Dopóki** $((G\neq\emptyset)$ **lub** $(N\neq\emptyset))$ **wykonaj**
3. **Dopóki** $((N\neq\emptyset)$ **oraz** $(\min_{j\in N} r_j \leq t))$ **wykonaj**
4. $e=\arg\min_{j\in N} r_j$, $G=G\cup\{e\}$, $N=N\setminus\{e\}$.
5. **Jeżeli** $G=\emptyset$ **wykonaj**
6. $t=\min_{j\in N} r_j$, **idź do** 3.
7. $e=\arg\max_{j\in G} q_j$, $G=G\setminus\{e\}$,
8. $k=k+1$, $\pi(k)=e$, $t=t+p_e$, $C_{max}=\max(C_{max}, t+q_e)$.

Rysunek 1: Pseudokod algorytmu Schrage

1.2 Algorytm Schrage z podziałem

Ten algorytm działa podobnie do poprzedniego, jednak tutaj pojawia się jedne dodatkowe działanie, pozwalające na dobranie bardziej optymalnego działania. Mianowicie w momencie dodawania nowego zadania na maszynę, sprawdzane jest czy nowe zadanie nie ma mniejszego czasu wykonywania niż to obecnie wykonywane na maszynie, jeśli tak jest to zadanie obecne zastępowane jest przez nowe zadanie. Wynikiem tego jest brak jednoznacznej permutacji wyjściowej, jednak wynik jest każdorazowo lepszy niż dla poprzedniego algorytmu.

Dane: n – liczba zadań,
r_i – termin dostępności zadania i,
p_i – czas wykonania zadania i,
q_i – czas dostarczenia zadania i,
UB – górne oszacowanie .
Szukane : π^* - optymalna permutacja wykonania zadań na maszynie.
Struktury pomocnicze :
t – chwila czasowa,
l – aktualnie wykonywane zadanie,
N – zbiór zadań nieuszeregowanych,
G – zbiór zadań gotowych do realizacji.

Algorytm prmt**S**

1. $t=0,\ C_{max}=0,\ G=\varnothing, N=\{1,2,\ldots,n\}, l=0, q_0=\infty$.
2. **Dopóki** $((G\neq\varnothing)$ **lub** $(N\neq\varnothing))$ **wykonaj**
3. **Dopóki** $((N\neq\varnothing)$ **oraz** $(\min_{j\in N} r_j \leq t))$ **wykonaj**
4. $e=\arg\min_{j\in N} r_j,\ G=G\cup\{e\},\ N=N\setminus\{e\},$
5. **jeżeli** $q_e>q_l$ **to** $p_l=t-r_e,\ t=r_e,$ **jeżeli** $p_l>0$ **to** $G=G\cup\{l\}$.
6. **Jeżeli** $G=\varnothing$ **wykonaj**
7. $t=\min_{j\in N} r_j$, **idź do** 3.
8. $e=\arg\max_{j\in G} q_j,\ G=G\setminus\{e\},$
9. $l=e,\ t=t+p_e,\ C_{max}=\max(C_{max},t+q_e)$.

Rysunek 2: Pseudokod algorytmu Schrage z podziałem

2 Porównanie skuteczności obu algorytmów

Ze względu na niewielkie różnice, przedstawione algorytmy dają zbliżone wyniki, jednak zdecydowanie implementacja podziałów jest opłacalna, ponieważ narzut obliczeniowy jest nieznaczny.

id	r	p	q
1	0	27	78
2	140	7	67
3	14	36	54
4	133	76	5

Rysunek 3: Przykładowe dane wejściowe

Dla przedstawionych danych wejściowych uruchomiono oba programy, dały różne wyniki zgodnie z oczekiwaniami. Schrange bez podziałów uzyskał wynik 283 dla permutacji [1 3 4 2], natomiast z podziałami dał wynik 221 dla permutacji [x x x x]. Jak widać została dokonana jedna podmiana, która zadecydowała o poprawie wyniku. Podobne obserwacje można dokonać przy pozostałych danych, jednak te zestawy są zbyt duże, aby je analizować w tym opracowaniu.

3 Analiza różnic czasu wykonania względem implementacji

Oba algorytmy można implementować na różne sposoby, często pozwala to na zmniejszenie czasu wykonywania programu. Ze względu że oba algorytmy wymagają jedynie zaimplementowania kolejki priorytetowej, można to rozwiązać na wiele sposobów. Polecenie dotyczyło dwóch implementacji na kopcach (jedna z użyciem biblioteki STL, drugi na własnej implementacji) oraz na tabeli. Wykonaliśmy wszystkie implementacje, wszystkie działają poprawnie i dają identyczne wyniki, różnią się jedynie czasami wykonania, co przedstawia poniższa tabela.

Na podstawie powyższej tabeli, można jednoznacznie stwierdzić, że najszybciej wykonywany jest program zaimplementowany na własnym kopcu. Jednak związane jest to z największym nakładem pracy w celu implementacji. Zadziwiająco następna jest zwykła kolejka priorytetowa, z zaimplementowanym algorytmem sortowania rzędu $O(n^2)$, pokrywa się to jednak z wymaganym nakładem pracy programisty. Natomiast najwolniejszy jest kopiec zaimplementowany na bibliotece STL, jednak wymaga również najmniej pracy.

Nr danych	algorytm	bez kopca [us]	kopiec STL [us]	własny kopiec [us]
Data 0	prmtS	3	12	2
	Schrange	2	5	2
Data 1	prmtS	30	76	21
	Schrange	24	57	16
Data 2	prmtS	27	81	19
	Schrange	25	56	16
Data 3	prmtS	12	69	6
	Schrange	13	54	6
Data 4	prmtS	25	80	21
	Schrange	23	59	17
Data 5	prmtS	26	69	20
	Schrange	25	65	20
Data 6	prmtS	36	98	26
	Schrange	36	63	17
Data 7	prmtS	25	92	26
	Schrange	28	75	20
Data 8	prmtS	36	102	27
	Schrange	35	87	23

Tabela 1: Porównanie trzech typów implementacji

4 Podsumowanie

Zarówno algorytm Schrange, jak i Schrange z podziałem nie są zbyt skomplikowane w założeniach, jak i implementacji. Przetestowane implementacje dają identyczne wyniki, lecz w różnych czasach. Ciekawym jest fakt, że najwolniejszym okazał się kopiec zaimplementowany na kontenerach z biblioteki STL, co daje informację, że nie jest najważniejszy algorytm sortowania, a operacje dodawania i usuwania elementów z kontenera (co przy bibliotece STL jest obarczone dużym narzutem). Wszystkie implementacje działają szybko i bez widocznych utrudnień, ze zwględu na ułożenie zadań.