STEROWANIE PROCESAMI CIĄGŁYMI
Algorytm Schrage

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

13.04.2021r.

1 Opis problemu

Ponownie będziemy się zajmować problemem RPQ. Dla przypomnienia - jest to problem jednomaszynowy, gdzie na wejściu podane są trzy parametry dla każdego zadania:

- r_i – termin dostępności zadania i,
- p_i – czas wykonania zadania i,
- q_i – czas dostarczenia zadania i.

Wyjściem działania algorytmu obsługującego problem RPQ jest permutacja wykonania zadań π oraz moment maksymalny z możliwych terminów dostarczenia zadań C_{max}. Zaimplementowane zostały dwa algorytmy - algorytm Schrage z podziałem i bez podziału.

2 Zaimplementowane algorytmy

2.1 Algorytm Schrage bez podziału

Implementacja tego algorytmu wymaga wprowadzenia zmiennych pomocniczych:

- n - liczba zadań,
- t - chwila czasowa,
- k - pozycja w permutacji π,
- N - zbiór zadań nieuszeregowanych,
- G - zbiór zadań gotowych do realizacji.

Wówczas lista kroków algorytmu Schrage bez podziału (bez przerywania zadań) prezentuje się następująco:

1. t = 0, k = 0, $C_{max} = 0$, G = $\emptyset$, N = {1, 2, ..., n},
2. dopóki (G $\neq \emptyset$ lub N $\neq \emptyset$) wykonaj:
   (a) dopóki (N $\neq \emptyset$ oraz $\min_{j \in N} r_j \leqslant$ t) wykonaj:
      i. $\arg\min_{j \in N} r_j$, G = G $\cup$ {e}, N = N \{e}.
   (b) jeżeli G = $\emptyset$ wykonaj:
      i. t = $\min_{j \in N} r_j$, idź do 3.
   (c) e = $\arg\min_{j \in G} q_j$, G = G \ {e},
   (d) k = k + 1, π(k) = e, t = t + p_e, C_{max} = max(C_{max}, t+q_e).

Algorytm Schrage zaimplementowany jak pokazano na rys. 1 ma złożoność $O(n \cdot logn)$.

```cpp
std::pair<std::vector<int>, int> schrage_algorithm(const std::vector<task>& tasks )
{
    std::priority_queue<std::pair<int, int> > N, G;
    std::vector<int> pi;
    int t = 0, k = 0, cmax = 0;

    for(unsigned int i = 0; i < tasks.size(); ++i)
        N.push({-tasks[i].r, tasks[i].index} );

    while( !G.empty() or !N.empty() )
    {
        while(!N.empty() and -N.top().first <= t)
        {
            std::pair<int, int> e = N.top();
            G.push({tasks[e.second - 1].q, e.second });
            N.pop();
        }
        if(G.empty())
        {
            t = -N.top().first;
            continue;
        }
        std::pair<int, int> e = G.top();
        G.pop();
        pi.push_back(e.second);
        k++;
        t += tasks[e.second - 1].p;
        cmax = std::max(cmax, t + e.first);
    }
    return {pi, cmax};
}
```

Rysunek 1: Algorytm Schrage bez podziału.

2.2 Algorytm Schrage z podziałem

Implementacja tego algorytmu wymaga wprowadzenia zmiennych pomocniczych:

- n - liczba zadań,
- t - chwila czasowa,
- l - aktualnie wykonywane zadanie,
- N - zbiór zadań nieuszeregowanych,
- G - zbiór zadań gotowych do realizacji.

Wówczas lista kroków algorytmu Schrage bez podziału (bez przerywania zadań) prezentuje się następująco:

1. t = 0, $C_{max} = 0$, G = $\emptyset$, N = {1, 2, ..., n}, l = 0, $q_0 = \infty$,
2. dopóki (G $\neq \emptyset$ lub N $\neq \emptyset$) wykonaj:
   (a) dopóki (N $\neq \emptyset$ oraz $\min_{j \in N} r_j \leqslant$ t) wykonaj:
      i. e = $\arg\min_{j \in N} r_j$, G = G $\cup$ {e}, N = N \{e},
      ii. jeżeli $q_e > q_l$ to $p_l = t - r_e$, $t = r_e$, jeżeli $p_l > 0$ to G = G $\cup$ {l}.
   (b) jeżeli G = $\emptyset$ wykonaj:
      i. t = $\min_{j \in N} r_j$, idź do 3.
   (c) e = $\arg\max_{j \in G} q_j$, G = G \ {e},
   (d) l = e, t = t + p_e, C_{max} = max(C_{max}, t+q_e).

Algorytm Schrage z podziałem zaimplementowany jak pokazano na rys. 2 ma złożoność $O(n \cdot logn)$.

```cpp
int schrage_algorithm(std::vector<task>& tasks )
{
    std::priority_queue<std::pair<int, int> > N, G;
    int t = 0, l = 0, cmax = 0, q = INF;
    for(unsigned int i = 0; i < tasks.size(); ++i)
        N.push({-tasks[i].r, tasks[i].index} );
    while( !G.empty() or !N.empty() )
    {
        while(!N.empty() and -N.top().first <= t)
        {
            std::pair<int, int> e = N.top();
            G.push({tasks[e.second - 1].q, e.second });
            N.pop();
            if( tasks[e.second - 1].q > q)
            {
                tasks[l - 1].p = t + e.first;
                t = -e.first;
                //printf("Przerywam %d w czasie %d na rzecz %d\n", l, t, e.second);
                if(tasks[l -1].p > 0 )
                    G.push({tasks[l -1].q, l});
            }
        }
        if(G.empty())
        {
            t = -N.top().first;
            continue;
        }
        std::pair<int, int> e = G.top();
        G.pop();
        l = e.second;
        q = e.first;
        //printf("Robie %d w czasie %d\n", e.second, t);
        t += tasks[e.second - 1].p;
        cmax = std::max(cmax, t + e.first);
    }
    return cmax;
}
```

Rysunek 2: Algorytm Schrage z podziałem.

3 Wyniki i ich porównanie

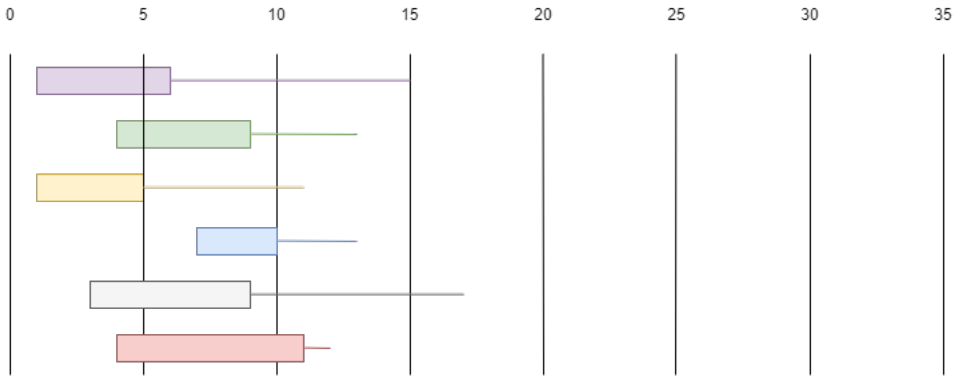
Dane testowe pobrane były ze strony `http://new.zsd.iiar.pwr.edu.pl/educ.php?lid=132&zid=SCHRAGE`.

3.1 Przykładowe działanie algorytmów

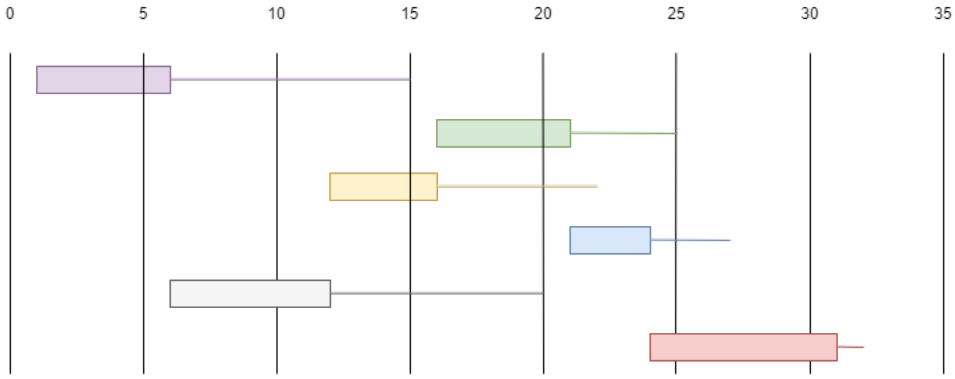
Na podstawie pierwszego zestawu testowego zostanie przedstawione działanie algorytmu opisanych w sekcji 2. Dane tekstowe przedstawiają się wówczas następująco:

6
1 5 9
4 5 4
1 4 6
7 3 3
3 6 8
4 7 1

Pierwsza liczba oznacza liczbę zadań, które mamy uszeregować. Kolejne parametry dla każdego zadania odpowiadają wartościom r_i, p_i, q_i objaśnione w opisie problemu na początku sprawozdania. Dane obrazują sytuację zaprezentowane na rysunku 3. Po przepuszczeniu przez algorytm Schrage bez podziału otrzymujemy poszeregowanie jak na rys. 4 odpowiadające kolejności zadań: 1, 5, 3, 2, 4, 6 oraz wartości $C_{max} = 32$.



Rysunek 3: Nieposzeregowane zadania z pierwszego zbioru danych.



Rysunek 4: Uszeregowane zadania z pierwszego zbioru danych przez algorytm Schrage bez podziału.

Algorytm Schrage z podziałem zwraca taką samą odpowiedź dla tego przypadku, inny wynik otrzymujemy w przypadku drugiego zestawu danych:
10
219 5 276
84 13 103
336 35 146

271 62 264
120 33 303
299 14 328
106 46 91
181 93 97
263 13 168
79 60 235

Algorytm Scharge z podziałem poniższą kolejność wykonywania zadań.

1. Przetwarzanie zadania nr 10 od chwili czasowej 79.
2. Przerwanie zadania nr 10 w chwili czasowej 120 na rzecz zadania nr 5.
3. Przetwarzanie zadania nr 5 od chwili czasowej 120.
4. Przetwarzanie zadania nr 10 od chwili czasowej 153.
5. Przetwarzanie zadania nr 2 od chwili czasowej 172.
6. Przetwarzanie zadania nr 8 od chwili czasowej 185.
7. Przerwanie zadania nr 8 w chwili czasowej 219 na rzecz zadania nr 1.
8. Przetwarzanie zadania nr 1 od chwili czasowej 219.
9. Przetwarzanie zadania nr 8 od chwili czasowej 224.
10. Przetwarzanie zadania nr 8 od chwili czasowej 185.
11. Przerwanie zadania nr 8 w chwili czasowej 263 na rzecz zadania nr 9.
12. Przetwarzanie zadania nr 9 od chwili czasowej 263.
13. Przerwanie zadania nr 9 w chwili czasowej 271 na rzecz zadania nr 4.
14. Przetwarzanie zadania nr 4 od chwili czasowej 271.
15. Przerwanie zadania nr 4 w chwili czasowej 299 na rzecz zadania nr 6.
16. Przetwarzanie zadania nr 6 od chwili czasowej 299.
17. Przetwarzanie zadania nr 4 od chwili czasowej 313.
18. Przetwarzanie zadania nr 9 od chwili czasowej 347.
19. Przetwarzanie zadania nr 3 od chwili czasowej 352.
20. Przetwarzanie zadania nr 8 od chwili czasowej 387.
21. Przetwarzanie zadania nr 7 od chwili czasowej 407.
22. Zakończenie wykonywania i stygnięcia zadań w chwili $C_{max} = 641$.

3.2 Wyniki na danych testowych

W tabelach 4 oraz 2 przedstawione zostały wartości otrzymanych C_{max} dla poszczególnych zestawów danych, a w tabeli 3 oraz na wykresie 5 porównano czasy wykonywania się algorytmów (z dokładnością tylko do mikrosekund).

zestaw testowy	C_{max}
1	32
2	687
3	1299
4	1399
5	3487
6	3659
7	6918
8	6936
9	72853

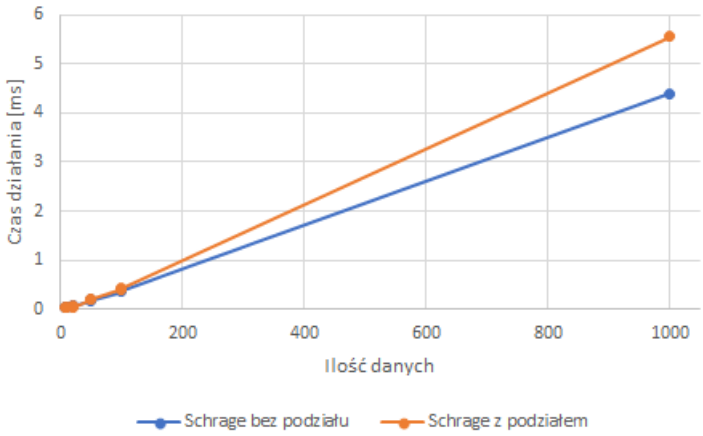
Tabela 1: Algorytm Schrage bez podziału.

zestaw testowy	C_{max}
1	32
2	641
3	1257
4	1386
5	3472
6	3617
7	6885
8	6904
9	72852

Tabela 2: Algorytm Schrage z podziałem.

zestaw testowy	ilość zadań w zestawie	alg. Schrage bez podziału	alg. Schrage z podziałem.
1	6	0.035	0.025
2	10	0.049	0.047
3	20	0.086	0.085
4	20	0.077	0.039
5	50	0.175	0.196
6	50	0.189	0.206
7	100	0.355	0.412
8	100	0.385	0.427
9	1000	4.387	5.545

Tabela 3: Porównanie czasów wykonywania się algorytmów w milisekundach.



Rysunek 5: Porównanie czasu działania algorytmów.

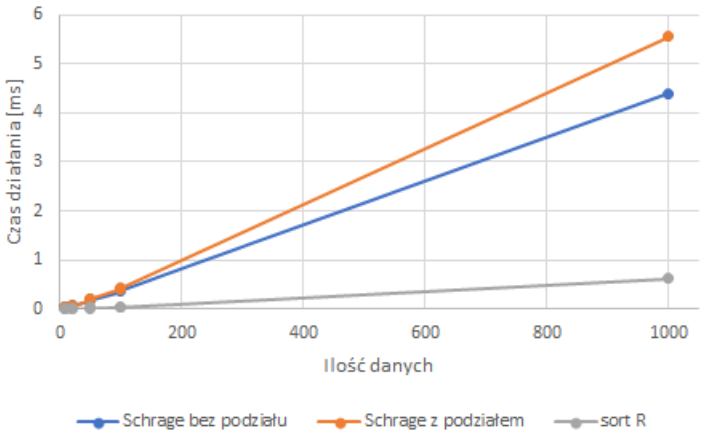
3.3 Porównanie algorytmu Schrage z poprzednimi algorytmami

Algorytm Schrage jest lepszym rozwiązaniem w porównaniu do algorytmu Jacksona ze względu na złożoność $O(n \cdot logn)$ lepszą niż $O(n^2)$ Jacksona. Ciężko jednak byłoby porównać oba algorytmy w czystych formach, gdyż w przypadku alg. Jacksona nie był podawany parametr q - należałoby je odpowiednio zmodyfikować, aby wykonywać je z tymi samymi danymi testowymi. Algorytmy można jednak porównać z poprzednim rozwiązaniem problemu RPQ - sortowaniu po r. Dla przypomnienia zamieszona poniżej tabela przedstawia wyniki C_{max} dla algorytmu sort R.

zestaw testowy	C_{max}
1	34
2	746
3	1594
4	1576
5	4013
6	4296
7	7831
8	7973
9	86648

Tabela 4: Algorytm sortowania po r.

Wyniki te są zdecydowanie gorsze niż otrzymywane przez algorytmy Schrage (zarówno z podziałem, jak i bez). Dzieje się to jednak kosztem złożoności obliczeniowej. Mimo że rząd złożoności jest podobny, to duży wpływ mają na nie stałe występujące przy nich. Na rys. 6 można zauważyć różnicę w czasowym wykonywaniu się programów.



Rysunek 6: Porównanie czasu działania algorytmów Schrage z sortowaniem po r.

4 Wnioski

- Obie wersje algorytmu są bez wątpienia lepszym rozwiązaniem (pod względem wyników) niż przedstawione w poprzednim sprawozdaniu algorytmu do rozwiązania problemu RPQ (w tym sortowanie po r).
- Algorytm Schrage bez podziału nie zwraca rozwiązania optymalnego (co można zauważyć patrząc na wyniki zwracane przez algorytm własny wykorzystujący losowość), jednak wynik może być co najwyżej dwa razy gorszy niż rozwiązanie optymalne.
- Algorytm Schrage z podziałem zwraca lepsze uszeregowanie zadań niż algorytm Schrage bez podziału, jednak jest trochę wolniejszy (mimo takiego samego rzędu złożoności obliczeniowej).