Tomasz Jankowiak 249006 1 V 2021

Sterowanie procesami dyskretnymi

Laboratorium 4

Algorytm Schrage dla $1|r_j,q_j|C_{max}$

Prowadzący:	Dr inż. Mariusz Makuchowski
Termin zajęć:	Piątek, 11.15
Numer grupy laboratoryjnej:	E12-30a

1. Opis problemu

Mamy do wykonania *n* zadań na pojedynczej maszynie. Każde zadanie *j* opisane jest trzema parametrami:

- r_j - czas dostarczenia (termin dostępności),
- p_j - czas trwania,
- q_j - czas stygnięcia.

Szukamy uszeregowania o najmniejszej długości C_{max} (najpóźniej wystygnięte zadanie).

W ogólnym przypadku zagadnienie jest NP-trudne. Najlepsze znane warianty algorytmów optymalnych są w stanie rozwiązywać w stosunkowo krótkim czasie problemy z n ≤ 1000, jednak przejawiają eksplozję obliczeń dla przykładów o większym rozmiarze [1].

2. Opis algorytmu

a) Schrage

W 1971 roku amerykański naukowiec Linus Schrage w swojej pracy [2] zaproponował algorytm, stosujący oczywistą intuicyjnie regułę, znaną jako uogólniona reguła Jacksona: jeżeli maszyna jest wolna oraz co najmniej jedno zadanie jest gotowe do wykonywania, należy skierować do wykonywania dostępne zadanie najpilniejsze. Udowodniono, że algorytm jest 2 -aproksymacyjny[3].

Algorytm buduje rozwiązanie poprzez dokładanie jeszcze nieuszeregowanych zadań na koniec bieżącej kolejności. Z zadań dostępnych (te, które już dotarły do maszyny i jeszcze nie zostały wykonane) do permutacji częściowej dodajemy zadanie o największym czasie stygnięcia.

Złożoność obliczeniowa algorytmu Schrage na kopcach wynosi ***O*(*n* log *n*)**.

Algorytm Schrage jest algorytmem przybliżonym – nie daje optymalnego rozwiązania danego problemu. Dostarcza dla problemu $1|r_j, q_j|Cmax$ uszeregowania o długości nie większej niż 2 razy długość uszeregowania optymalnego (tzw. algorytm 2-aproksymacyjny)[3]. Zatem nie jest to najlepszy algorytm, ale za to jest całkiem prosty w działaniu i implementacji.

b) Schrage z podziałem

Jeżeli maszyna jest wolna oraz co najmniej jedno zadanie jest gotowe do wykonywania, należy skierować do wykonywania dostępne zadanie posiadające najdłuższy czas stygnięcia (q_j reprezentuje priorytet zadania). Jeśli w czasie wykonywania pewnego zadania pojawi się zadanie z wyższym priorytetem niż obecnie wykonywane, należy przerwać wykonywanie bieżącego zadania, oraz zwrócić do kolejki zadań oczekujących na wykonanie pozostałą niezrealizowaną część przerwanego zadania.

Dopuszczenie przerywania wykonywania zadań jest relaksacją, która prowadzi do otrzymania problemu $1|r_j,q_j,pmtn|C_{max}$ z wielomianowym algorytmem rozwiązywania. Po niewielkiej modyfikacji może być do tego wykorzystany Algorytm Schrage. W tym przypadku dopuszczalne jest przerywanie wykonywania zadań oraz zwracanie niewykonanej jeszcze części zadania do kolejki, by później wznowić jego wykonywanie. Przerwanie następuje każdorazowo gdy w kolejce pojawia się zadanie o wyższym priorytecie niż to aktualnie wykonywane, tj. wtedy gdy ma większy czas stygnięcia. Zapis algorytmu w tym przypadku jest nieco bardziej złożony bowiem rozwiązanie nie może być reprezentowane permutacją.

Ten algorytm również posiada złożoność obliczeniową $O(n \log n)$ [3]. Udowodniono, że liczba przerwań w rozwiązaniu optymalnym jest nie większa niż n-1 [1].

---

[1] Grabowski J., Nowicki E., Smutnicki C., *„Metoda blokowa w zagadnieniach szeregowania zadań”*, Warszawa 2003, str. 62-64.
[2] Schrage L., *Obtaining optimal solution to resource constrained network scheduling problem*, unpublished manuscript, 1971.
[3] Smutnicki C., *„Algorytmy szeregowania zadań”*, Oficyna Wydawnicza PWr, Wrocław 2012, str. 147-150.

3. Kod programu

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int cmax(int n, int*R, int*P, int *Q, int *X)
{
    int m=0,c=0; // m - moment do ktorego pracuje maszyna, c - moment do ktorego siega najdluzszy ogonek q

    for(int i=0;i<n;i++){
        m = max(m,R[X[i]])+P[X[i]];
        c = max(c,m+Q[X[i]]);
    }
    return c;
}

void Schrage(int n,int*R,int*P,int*Q,int*X) // dane wejsciowe, X - proponowana kolejnosc
{
    int A[100],K[100]; // A - zbior zadan posortowanych po r, z ktorego bedziemy przenosic do kolejki K
    int a=n,k=0,x=0,t=0; // a,k - liczba zadan w A i K; x - l. uszeregowanych zadan; t-aktualny moment
    for(int i=0;i<n;i++) A[i]=i; // przypisanie kolejnosci 0,1,2,3...
    for(int i=0; i<(n-1); i++) // sortowanie malejaco po r
        for(int j=0; j<(n-1); j++)
            if( R[A[j]] < R[A[j+1]] )
                swap(A[j], A[j+1]);
    while(x!=n) // jesli jeszcze nie zakonczono szeregowania
    {
        if(a!=0){ // jesli zbior zadan jest niepusty
            if(R[A[a-1]]<=t) // jesli zadania dojechaly
            {
                K[k]=A[a-1]; // dodajemy do kolejki zadania gotowe do realizacji
                k++; a--; // aktualizacja liczby zadan w kolejce i niedojechanych
                for(int e=k-1;e>0;e--){ // wstawienie nowego zadania na odpowiednie miejsce w kolejce (wg q)
                    if( Q[K[e]] < Q[K[e-1]] )
                        swap(K[e], K[e-1]);
                }
                continue; // powrot do poczatku petli while - sprawdzenie czy nastepne zadanie tez jest gotowe
            }
        }
        if(k!=0) // jesli sa zadania w kolejce
        {
            X[x]=K[k-1]; // przypisanie do kolejnosci zadania z kolejki o najwiekszym q
            k--; x++; // zaktualizowanie liczby zadan uszeregowanych i w kolejce
            t=t+P[X[x-1]]; // aktualizacja aktualnego momentu
            continue;
        }
        if(0==k && R[A[a-1]]>t){ // jesli nie ma zadan w kolejce ani dojechanych to posuwamy czas
            t=R[A[a-1]];
        }
    }
}

int Schrage_zPodzialem(int n,int*R,int*P,int*Q)
{
    int A[100],K[100];
    int a=n,k=0,x=0,t=0,p=0,l=-1,cmaks=0; // p - czas wykonywania zadania przed przerwaniem; l - wykonywane zadanie
    for(int i=0;i<n;i++) A[i]=i;
    for(int i=0;i<(n-1);i++) for(int j=0;j<(n-1);j++) if(R[A[j]]<R[A[j+1]]) swap(A[j],A[j+1]); // sort r
    while(a!=0 || k!=0){ // kiedy sa jeszcze zadania niedojechane lub w kolejce
        if(a!=0){
            if(R[A[a-1]]<=t){
                K[k]=A[a-1];
                k++; a--;
                for(int e=k-1;e>0;e--){
                    if( Q[K[e]] < Q[K[e-1]] )
```

```
                        swap(K[e], K[e-1]);
                }
                if(l!=-1)
                    if( Q[K[k-1]] > Q[l] ){ // sprawdzamy czy robimy przerwanie
                        K[k]=l;
                        k++;
                        swap(K[k-1],K[k-2]);
                        l=-1;
                    }
                continue;
            }
        }
        if(k!=0){
            if(-1==l){
                l=K[k-1];
                k--;
            }
            if(a!=0) p = min( P[l], R[A[a-1]]-t );
            else p=P[l];
            t = t + p; // leci czas
            P[l] = P[l] - p;
            if(0==P[l]){
                cmaks=max(cmaks,t+Q[l]);
                l=-1;
            }
            continue;
        }
        if(0==k && a!=0)
            if(R[A[a-1]]>t)
                t=R[A[a-1]];
    }
    return cmaks;
}


    int main()
    {
        int n,R[100],P[100],Q[100],X[100];

        ifstream plik("data.txt");
        string s; while(s!="data.000:") plik>>s;
        plik >> n;
        for(int i=0;i<n;i++)
            plik >> R[i] >> P[i] >> Q[i];
        plik.close();

        Schrage(n,R,P,Q,X);
        cout << "schr:\n" << cmax(n,R,P,Q,X) << endl;
        for(int i=0;i<n;i++) cout << X[i]+1 << " ";

        cout << "\n\nschrpmtn = " << Schrage_zPodzialem(n,R,P,Q) << endl;

        cin.get();
        return 0;
    }
```

4. Działanie programu

Algorytmy Schrage oraz Schrage z podziałem zaimplementowano na kopcach zrobionych samemu na tablicy.

a) Bez podziału

Po wczytaniu danych z pliku i zainicjowaniu zmiennych tworzymy tablice zadań niedojechanych *A* – jeszcze niegotowe do realizacji w chwili *t* (posortowane malejąco po r), w kolejce *K* – z terminem dostępności r_j mniejszym lub równym chwili *t* (posortowane rosnąco po q) oraz już uporządkowanych *X*. Po każdym wykonanym zadaniu aktualizujemy bieżącą chwilę czasu *t* oraz kolejkę. Jeśli kolejka jest pusta, aktualizujemy bieżącą chwilę t, przesuwając ją do momentu dostępności najwcześniejszego zadania ze zbioru A i wznawiamy proces aktualizowania zbioru *K* zadań gotowych. Na końcu liczymy wartość celu C_{max} dla otrzymanej permutacji. Na wyjściu program wypisuje kolejność oraz wartość C_{max}.

b) Schrage z podziałem

W przypadku algorytmu prmtS celowo zrezygnowano z wyznaczenia permutacji wykonywania zadań, choć nie jest to trudne. Zmienna *l* przechowuje indeks zadania wykonywanego aktualnie na maszynie. W tym przypadku ostrożniej przesuwamy chwile czasu *t* – gdy jakieś zadanie dojeżdża do kolejki, sprawdzamy czy ma większy czas stygnięcia od zadania aktualnie znajdującego się na maszynie. Jeżeli tak to wykonywanie zadania jest przerywane, a pozostała część zadania, o odpowiednio krótszym czasie wykonania, dodawana jest do kolejki.

c) Testy

Działanie programu przetestowano na danych ze strony
http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/lab.instrukcje/lab04.schrage/schr.data/schr.data.txt.
Algorytm Schrage z podziałem poprawnie zwracał długość, a zwykły Schrage permutację oraz długość uszeregowania. Zatem można wyciągnąć wniosek, iż program najprawdopodobniej działa poprawnie.

5. Analiza złożoności obliczeniowej algorytmu Schrage

Kolejka *K* oraz uszeregowanie *X* są aktualizowane 2n razy – każde zadanie idzie do kolejki, a następnie do uszeregowania. Każdą z tych operacji można zrealizować na kopcach w czasie ograniczonym przez O(log n). Ponieważ liczba analizowanych chwil *t* jest rzędu O(n), zatem końcowa złożoność obliczeniowa algorytmu jest równa O(n log n).

6. Podsumowanie i wnioski

Algorytm Schrage dla problemu $1|r_j,q_j|C_{max}$ dostarcza rozwiązanie dość dalekie od optymalnego. Co prawda, nie jest ono dłuższe od 2-krotnej długości uszeregowania optymalnego, jednak nawet tak trywialne algorytmy jak sortowanie po *r* rosnąco lub po *q* malejąco, które optymalnie rozwiązują banalne problemy $1|r_j|C_{max}$ oraz $1|q_j|C_{max}$, są także algorytmami 2-aproksymacyjnymi dla problemu $1|r_j,q_j|C_{max}$[4].

Z kolei w problemie uogólnionym do $1|r_j,q_j,pmtn|C_{max}$ zmodyfikowana postać algorytmu Schrage łatwo znajduje rozwiązanie optymalne.

Efektywna implementacja algorytmu Schrage w obydwu problemach zapewnia złożoność obliczeniową $O(n \log n)$.

7. Bibliografia

1. Czesław Smutnicki – „*Algorytmy szeregowania zadań*”, Oficyna Wydawnicza PWr, Wrocław 2012, str. 147-150.
2. Linus Schrage – „*Obtaining optimal solution to resource constrained network scheduling problem*”, unpublished manuscript, 1971.
3. Józef Grabowski, Eugeniusz Nowicki, Czesław Smutnicki – „*Metoda blokowa w zagadnieniach szeregowania zadań*”, Warszawa 2003, str. 62-64,74-75, *http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/lab.instrukcje/lab04.schrage/schr.literatura/%5bMBwZSZ%5d.przerywalnosc.zadan.pdf*.
4. Mariusz Makuchowski – wykład „Algorytm Schrage dla $1|r_j, q_j|C_{max}$”, 2019, *http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl/download/courses/sterowanie.procesami.dyskretnymi/wyk.slajdy/SPD_w05_Schrage.pdf*.
5. The University of Chicago Booth School of Business – *Linus Schrage biography*, *https://www.chicagobooth.edu/faculty/emeriti/linus-schrage*.

[4] Grabowski J., Nowicki E., Smutnicki C., „*Metoda blokowa w zagadnieniach szeregowania zadań*”, Warszawa 2003, str. 74.