Uniwersytet Śląski
Instytut Matematyki, Fizyki i Chemii
w Katowicach
Kierunek Informatyka

Szeregowanie zadań o jednostkowych czasach wykonania na dowolnej liczbie identycznych procesorów.

Autorzy:
Szymon Bluma
Zbigniew Golus

Katowice
Czerwiec 2007

Spis treści:

Wstęp

Zagadnienie szeregowania zadań w popularnych systemach operacyjnych dla wielu procesorów jest procesem bardzo skomplikowanym, jednak od strony teoretycznej optymalne algorytmy dają się sprowadzić do prostego szeregowania zadań.
Projekt ten ma za zadanie zobrazować działania algorytmów szeregowania zadań niepodzielnych i zależnych na dowolnej liczbie identycznych procesorów w celu minimalizacji maksymalnego opóźnienia L_{max}.
W systemach komputerowych czas wykonania zadań jest jest znany a priori, w przeciwieństwie do wielu innych zastosowań np. szeregowania detali na maszynach. Niemniej jednak także w systemach komputerowych rozwiązanie deterministycznego problemu szeregowania ma istotne znaczenie praktyczne.

Podstawa niniejszego przedstawienia problemu jest książka [1].
Projekt został wykonany w języku JAVA firmy Sun Microsystems [2] [3] [4].

Do prawidłowego działania aplikacji wymagana jest instalacja Wirtualnej Maszyny Java (JVM - Java Virtual Machine) w wersji co najmniej 1.6.0. Plik instalacyjny został dołączony do płyty z projektem.

Opis teoretyczny

Algorytm szeregowania (ang. scheduler - planista) to algorytm rozwiązujący jedno z najważniejszych zagadnień informatyki - jak rozdzielić czas procesora i dostęp do innych zasobów pomiędzy zadania, które w praktyce zwykle o te zasoby konkurują.
Najczęściej algorytm szeregowania jest implementowany jako część wielozadaniowego systemu operacyjnego, odpowiedzialną za ustalanie kolejności dostępu zadań do procesora. Oprócz systemów operacyjnych dotyczy w szczególności także serwerów baz danych.

Parametry charakteryzujące zadanie Z_j:
- *Czas wykonywania* **-** dla procesorów identycznych jest on niezależny od maszyny i wynosi r_j .
- *Moment przybycia* r_j – czas w którym zadanie trafia do algorytmu.
- *Termin zakończenia* d_j – Oznacza czas, od którego nalicza się spóźnienie, lub termin, którego przekroczyć nie wolno dla wykonywanego zadania

Maksymalne opóźnienie zadania oznaczamy jako L_{max}.
Maksymalny termin zakończenia zadania poprzez C_{max}.

Rozpatrujemy zbiór *n* zadań $Z = \{Z_1, Z_2, Z_3, ..., Z_n\}$
i zbiór *m* procesorów $P = \{P_1, P_2, P_3, ..., P_n\}$.

Zbiory zadań z określonymi ograniczeniami kolejnościowymi przedstawia się zwykle w postaci grafów skierowanych (digrafów).

Zadania zależne:
W zbiorze zadań Z można wprowadzić ograniczenia kolejnościowe w postaci dowolnej relacji częściowego porządku. Wówczas $Z_i < Z_j$ oznacza, ze zadanie Z_j może się zacząć wykonywać dopiero po zakończeniu Z_i (bo np. Z_j korzysta z wyników pracy Z_i).
Jeśli ograniczenia te nie występują, mówimy o zadaniach niezależnych.
Zasady poprawności harmonogramu:
- w każdej chwili procesor może wykonywać co najwyżej jedno zadanie,
- w każdej chwili zadanie może być obsługiwane przez co najwyżej jeden procesor,
- zadanie Z_j wykonuje sie w całości
- spełnione są ograniczenia kolejnościowe,
- w przypadku zadań niepodzielnych każde zadanie wykonuje sie nieprzerwanie w pewnym domknieto–otwartym przedziale czasowym, dla zadań podzielnych czasy wykonania tworzą skończoną sumę rozłącznych przedziałów.
Szukamy takiego uszeregowania zadań, aby czas wykonania zadań w określonych warunkach (w zależności od liczby procesorów), był jak najmniejszy, a także opóźnienie wszystkich zadań do wykonania było jak najmniejsze.

Zakładamy także, że procesory nie wykonują w chwili startu żadnych zadań, nie przyjmują innych zadań z zewnętrznych algorytmów i nie ulegają awarii w trakcie wykonywania zadań.
Przez problem szeregowania będziemy rozumieć uporządkowany ciąg parametrów charakteryzujących dany problem, z których nie wszystkie musza mieć nadane wartości, wraz z kryterium szeregowania.
Ustalając wartości wszystkich parametrów danego problemu szeregowania , otrzymujemy konkretny problem szeregowania.
Algorytmem szeregowania dla problemu będziemy nazywać dowolną procedurę, która dla każdego konkretnego problemu znajduje uszeregowania, jeśli ono istnieje.

Algorytmy szeregowania

I. Szeregowanie na dowolnej liczbie identycznych procesorów zadań o jednostkowych czasach wykonywania tworzących graf typu anty drzewa, w celu minimalizacji L_{max}

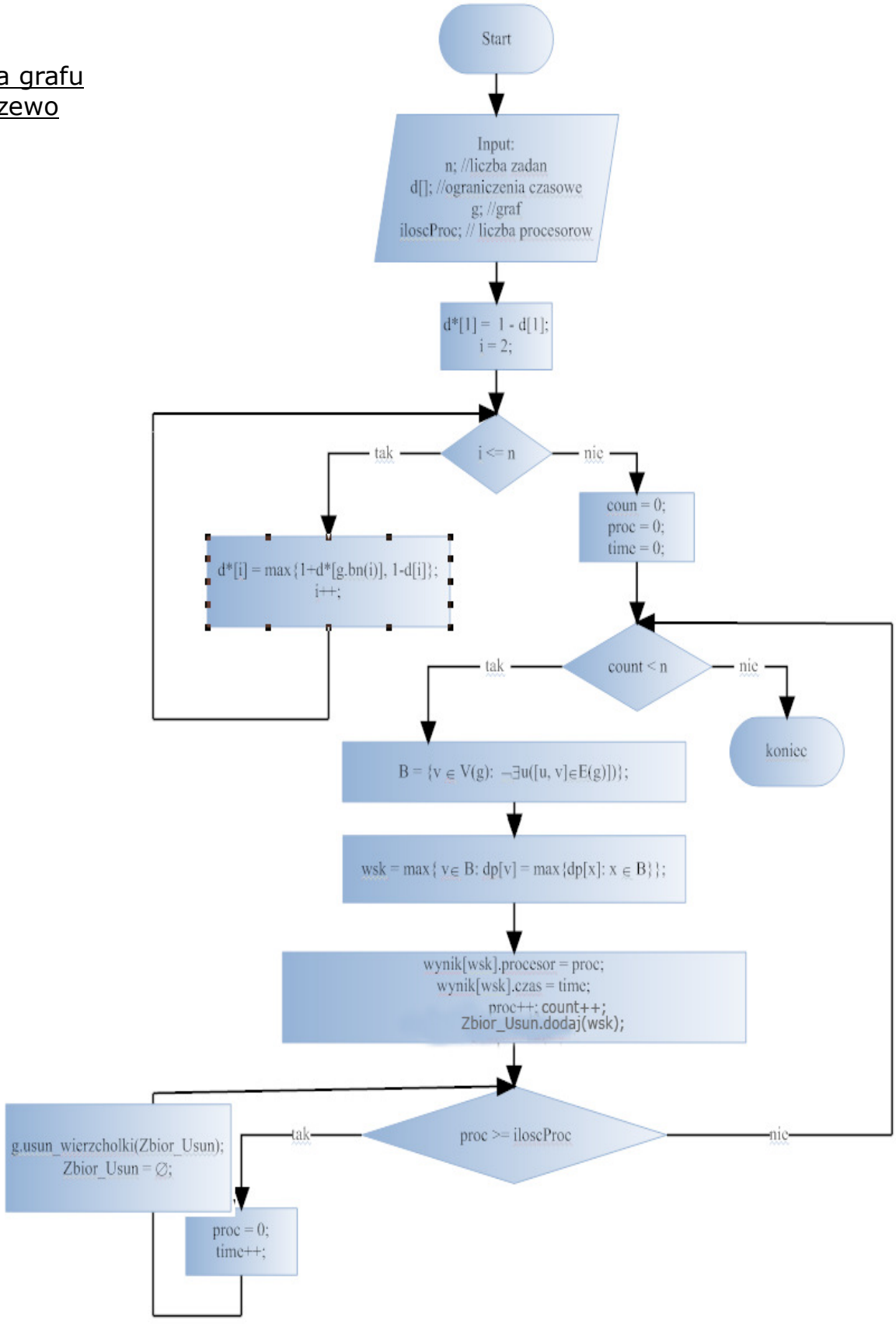
1. *Podstaw* $d_1 = 1 - d_1$
2. *Dla zadania* Z_k ($k = 2, 3, 4, \ldots n$)
   wyznacz wartości zmodyfikowanego terminu zakończenia według wzoru
   $d_k = max\{1 + d_{bn(k)}, 1 - d_k\}$
3. *Szereguj zadania w kolejności nie rosnących wartości ich zmodyfikowanych terminów zakończenia zgodnie z ograniczeniami kolejnościowymi.*

II. Szeregowanie na dwóch identycznych procesorach zadań o jednostkowych czasach wykonywania i dowolnych ograniczeniach kolejnościowych, w celu minimalizacji L_{max}.

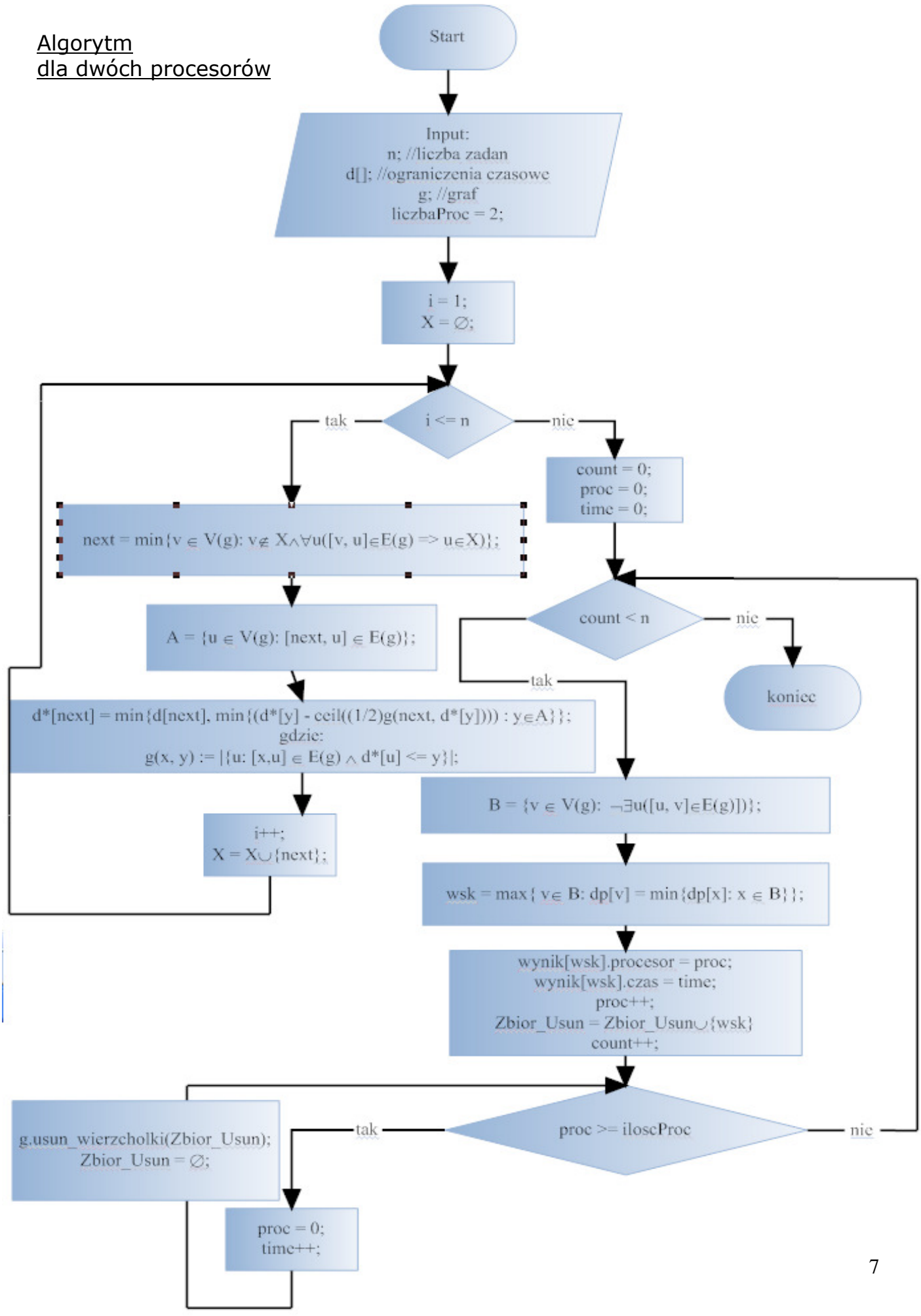
1. Wybierz zadanie Z_k, któremu nie przydzielono jeszcze zmodyfikowanego terminu zakończenia, w którego wszystkie następniki mają już wyznaczone zmodyfikowane terminy zakończenia.
2. Utwórz zbiór A_k następników zadania Z_k. Dla każdego $Z_i \in A_k$ wyznaczyć liczbę $g(k, d_i^*)$ następników zadania Z_k, mających wartości zmodyfikowanych terminów zakończenia nie większe niż d_i^*. Określ zmodyfikowany termin zakończenia wykonywania zadania Z_k w następujący sposób:
   $d_k^* = \min\{d_k, \min\{(d_i^* - [1/2g(k,d_i^*)]) : Z_i \in A_k\}\}$
   Jeśli któreś z zdań nie ma określonego zmodyfikowanego terminu zakończenia, to powróć do kroku 1.
3. Szereguj zadania w kolejności nie malejących wartości ich zmodyfikowanych terminów zakończenia i zgodnie z ograniczeniami kolejnościowymi.

Schemat blokowy

Algorytm dla grafu typu antydrzewo



Algorytm
dla dwóch procesorów



Omówienie algorytmów

```
public class UkladaczAntydrzewo implements Assigner {
    private Digraph g;      //graf zaleznosci
    private int n;                //liczba zadan
    private int liczbaProc;  //liczba procesorow
    private Zad[] dp;    //zmodyfikowane czasy zakonczenia indexowane numerami
wewnetrznymi
   public int[][] przyp;   //przyporzadkowanie tablica postaci [n][2], [i][0]
-procek, [i][1] -czas gdzie i-numer zewnetrzny
                            //procesory i czasy numerujemy od zera
    private int[] d;             //czasy zakonczenia (preferowane) indexowane
numerami wewnetrznymi wierzcholow
 private int[] numerZew; //numery zewnetrzne wierzcholkow, przed odwolaniem sie
do grafu g nalezy uzyc tej tablicy
    private int[] numerWew; //odwrotnosc permutacji numerZew :)
  private int timeMax;    //najwyzszy punkt na osi czasu w jakim jest wykonywane
zadanie (zmienna uzywana przy rysowaniu)

//funckcje interfejsu Assigner
    public int[][] getAssign(){
      return przyp;
    }



public UkladaczAntydrzewo(int liczbaProc, Digraph g, int[] d){   //d - tablica
czasow zakonczenia
      this.g = g;
      n = g.V();
      this.liczbaProc = liczbaProc;
      przyp = new int[n][2];
      dp = new Zad[n];
      DigraphChecker gch = new DigraphChecker(g);
      int[] tab = gch.topologicSort();     //tablica indexowana wierzchołkami od
0
      this.d = new int[n];
      numerZew = new int[n];
      numerWew = new int[n];
      for(int i = 0; i < n; i++){
          this.d[tab[i]] = d[i];
          numerZew[tab[i]] = i;
          numerWew[i] = tab[i];
      }
      doAlgorytm();
    }

private void doAlgorytm(){
      for(int i = 0; i < n; i++){
          if(bn(i) >= 0)
            dp[i] = new Zad(i, max(d[i], -1 + dp[bn(i)].intValue()));
          else
            dp[i] = new Zad(i, d[i]);
      }
      Arrays.sort(dp);
      int ulozono = 0;    //ilu zadaniom juz przydzielono procek
      int time = 0;         //ktory kwant czasu rozpatrujemy
      boolean[] mozliwy = new boolean[n]; //czy mozna uzyc wierzcholka i (wynika
z grafu zaleznosci)
      boolean[] ulozony = new boolean[n]; //juz   przydzielono   wierzcholkowi   i
```

```
czas procka
      Arrays.fill(ulozony, false);
      while(ulozono < n){
          //przegladanie grafu, szukanie wierzcholkow ktore juz mozna ulozyc
          Arrays.fill(mozliwy, true);
          for(int i = 0 ; i < n; i++){
            Iterator<Integer> it = g.iterator(numerZew[i]);
            if(it.hasNext()  &&  !ulozony[i]){       //jesli  nieulozone  zadanie
poprzedza inne
                 int v = numerWew[it.next().intValue()];    //jest to antydrzewo
wiec iterator moze zawierac
                mozliwy[v] = false;           //      conjawyzej 1 wierzcholek
            }
          }
          int procNum = 0;    //nastepny wolny procek
          boolean znaleziono = true;
          int wsk = 0;  //wskaznik na tablicy dp
          //szukanie nastepnego zadania o najnizszym dp
          while(procNum < liczbaProc && znaleziono){
            znaleziono = false;
            while(wsk < n){
                if(mozliwy[dp[wsk].nr] && !ulozony[dp[wsk].nr]){
      //przyporzadkowywanie zadaniu procesora
                  przyp[numerZew[dp[wsk].nr]][0] = procNum;
                  przyp[numerZew[dp[wsk].nr]][1] = time;
                  ulozono++;
                  ulozony[dp[wsk].nr] = true;
                  procNum++;
                  znaleziono = true;
                  break;
                }
                wsk++;
            }
          }
          time++;
      }
      timeMax = time;
    }

    private int bn(int u){
      Iterator<Integer> it = g.iterator(numerZew[u]);
      if(it.hasNext()){
          return numerWew[it.next().intValue()];
      }else{
          return -1;
      }
    }
```

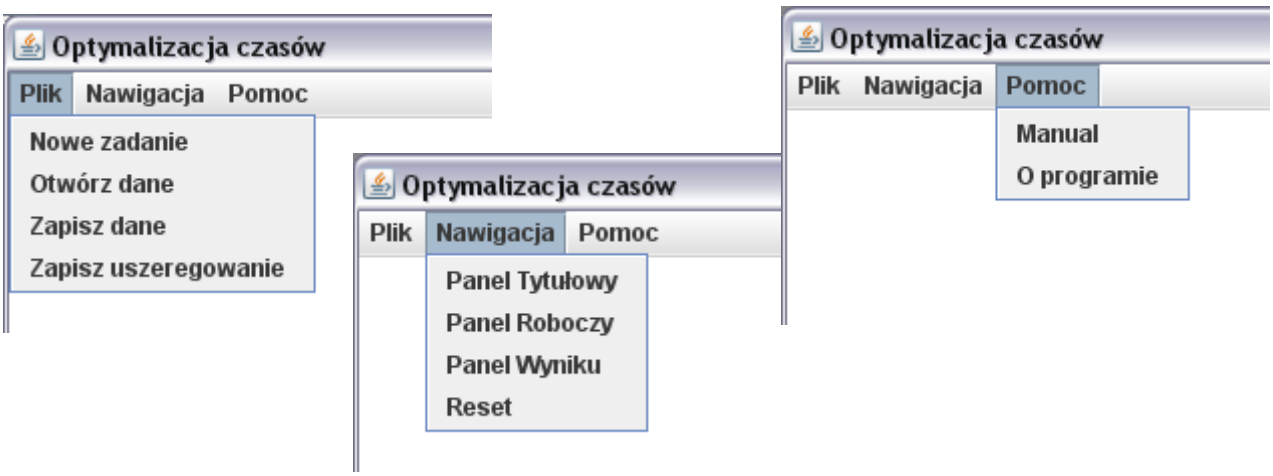
```
public class UkladaczDwaProcesory implements Assigner{
    private Digraph g;      //graf zaleznosci
    private int n;          //liczba zadan
    private int liczbaProc; //liczba procesorow
    private Zad[] dp;       //zmodyfikowane czasy zakonczenia indexowane
numerami wewnetrznymi
    public int[][] przyp;   //przyporzadkowanie tablica postaci [n][2], [i][0]
-procek, [i][1] -czas gdzie i-numer zewnetrzny
                      //procesory i czasy numerujemy od zera
```

```
    private int[] d;        //czasy zakonczenia (preferowane) indexowane
numerami wewnetrznymi wierzcholow
    private int[] numerZew; //numery zewnetrzne wierzcholkow, przed odwolaniem
sie do grafu g nalezy uzyc tej tablicy
    private int[] numerWew; //odwrotnosc permutacji numerZew :)
    private int timeMax;    //najwyzszy punkt na osi czasu w jakim jest
wykonywane zadanie (zmienna uzywana przy rysowaniu)

    //funckcje interfejsu Assigner
    public int[][] getAssign(){
      return przyp;
    }

    public int getProcCount(){
      return liczbaProc;
    }

    public int getMaxTime(){
      return timeMax;
    }

    public int getPreferredTime(int zadanie){      //zwraca preferowany czas
zakonczenia zadanie - numer zewnetrzny
      return d[numerWew[zadanie]];
    }

    public UkladaczDwaProcesory(int liczbaProc, Digraph g, int[] d){  //d -
tablica czasow zakonczenia
      this.g = g;
      n = g.V();
      this.liczbaProc = liczbaProc;
      przyp = new int[n][2];
      dp = new Zad[n];
      DigraphChecker gch = new DigraphChecker(g);
      this.d = new int[n];
      numerZew = new int[n];
      numerWew = new int[n];
      for(int i = 0; i < n; i++){
          this.d[i] = d[i];   //permutacja identycznosciowa
          numerZew[i] = i;
          numerWew[i] = i;
      }
      doAlgorytm();
    }
```

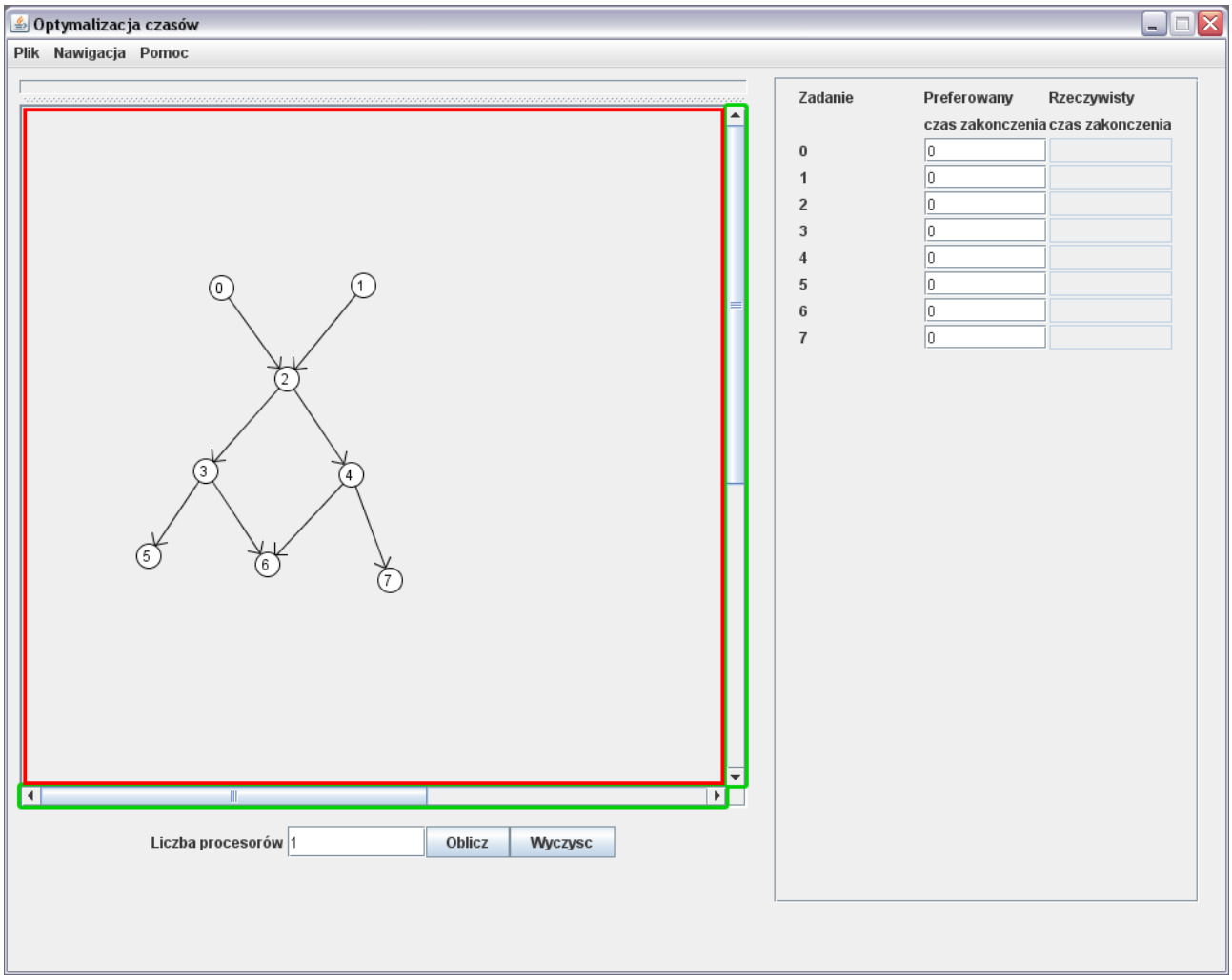
Instrukcja dla użytkownika

Główne okno programu widoczne po uruchomieniu:

Programem sterujemy dzięki menu.
Mamy dostępne 3 rozwijane główne mena programu:



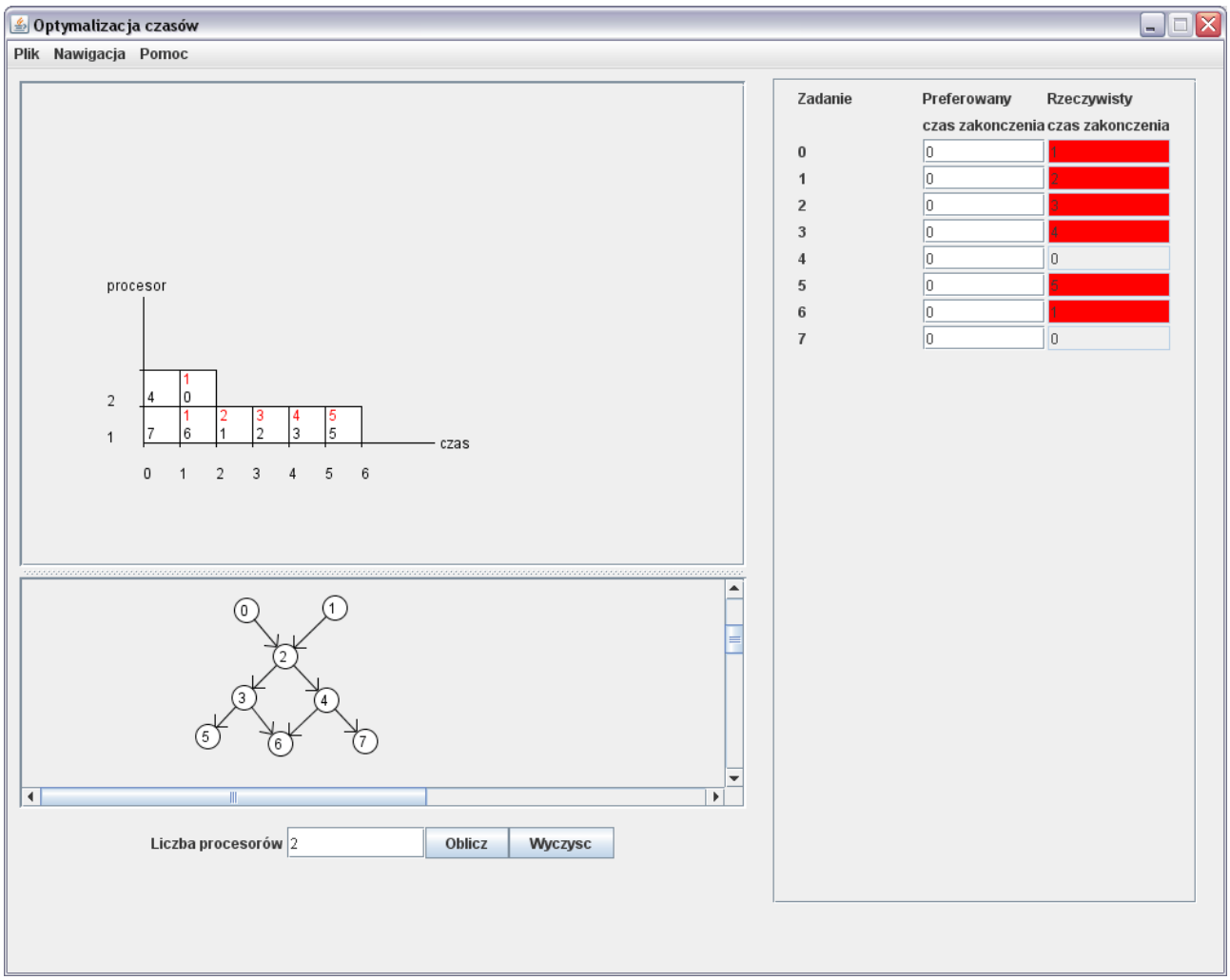
Aby wprowadzić dane do programu należy z menu *Nawigacja* wybrać opcje *Panel Roboczy*.
Pojawi się okno w którym będziemy mogli wstawiać poszczególne wierzchołki grafu oraz łączyć je z innymi wierzchołkami.



Aby wstawić nowy wierzchołek należy pojedynczym kliknięciem myszki kliknąć w obszar zaznaczony na kolor czerwony na zrzucie ekranowym powyżej. Suwakami (zaznaczonymi na zielono) możemy manipulować dostępną powierzchnią jaką możemy wykorzystać do wstawienia drzewa procesów.
Z prawej strony od omawianych komponentów programu znajduje się lista wszystkich dostępnych zadań, które zostały umieszczone w panelu roboczym. Możemy (ale nie musimy) przyporządkować konkretnym zadaniom termin zakończenia zadania.
Na dole pod panelem roboczym jest pole w którym możemy określić na ilu procesorach ma zostać wykonany algorytm szeregowania zadań.

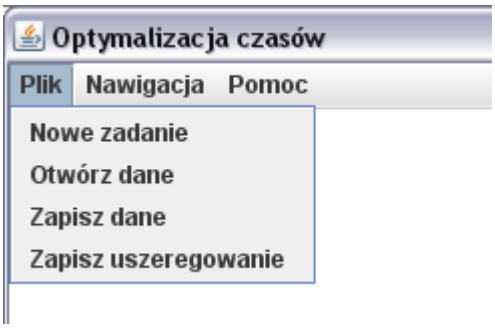
Aby przesunąć wierzchołek w inne miejsce wystarczy kliknąć i przytrzymać lewy przycisk myszy, a następnie przeciągnąć go w dogodne miejsce.
Usuwanie wierzchołka (i krawędzi z nim związanych) następuje po dwukrotnym kliknięciu na niego lewym przyciskiem myszy.

Po rozrysowaniu wszystkich wierzchołków, krawędzi z nimi związanych, liczby procesorów (ich ilość należy zatwierdzić klawiszem ENTER) i preferowanych czasów zakończenia zadań należy kliknąć na przycisk „OBLICZ”.



Zostanie wtedy obliczony optymalne uszeregowanie zadań, które dodaliśmy do obliczenia.
Na czerwono (zarówno na wykresie procesor/czas, jak również w prawej stronie okna) zostały oznaczone procesy, które nie zostały wykonane w preferowanym terminie zakończenia zadania.

W czasie pracy z menu PLIK możemy zapisać swoje efekty pracy, które potem również z menu PLIK możemy odczytać.
Nie stracimy tym sposobem już wprowadzonych wartości do programu.



Literatura

[1] J. Błażewicz, W.Cellary, i inni: *Badania operacyjne*,
PWN 1990.
[2] Cay S.Horstmann, Gary Cornell *JAVA 2 – Podstawy*,
Wydawnictwo Helion 2003
[3] Bruce Eckel: Thinking in Java, 4ed.,
Wydawnictwo Helion 2004
[4] www.java.sun.com - Programowanie w Java.
[5] www.us.edu.pl - Uniwersytet Śląski w Katowicach

Niniejszy program wykonaliśmy samodzielnie i wyrażamy zgodę na korzystanie, oraz modyfikowanie programu „Szeregowanie zadań o jednostkowych czasach wykonania tworzących graf na dowolnej liczbie identycznych procesorów w celu minimalizacji maksymalnego opóźnienia czasu wykonania” w celach edukacyjnych oraz naukowych na potrzeby Uniwersytetu Śląskiego.