МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КУРСОВИЙ ПРОЕКТ

з дисципліни «Системне програмне забезпечення»

Тема: «Моделювання паралельних обчислювальних систем»

Студента 4 курсу групи ІО-21
напряму підготовки 6.050102 Комп'ютерна інженерія
спеціальності 7.8.05010201 Комп'ютерні системи та мережі
Журо Георгія Олександровича
Керівник професор, доктор технічних наук
Симоненко Валерій Павлович

Національна оцінка _____

Кількість балів:_____ Оцінка ECTS_____

_____ _____
(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь,прізвище та ініціали)

Київ – 2015

№рядку	Формат	Позначення	Найменування	Кіл. аркушів	№ екз.	Примітка
1			Загальна документація			
2						
3			Розробка			
4						
5	А4	*ІАЛЦ.462637.001 ВП*	Відомість проекта	1	-	
6					-	
7	А4	*ІАЛЦ.462637.002 ТЗ*	Технічне завдання	3		
8						
9	А4	*ІАЛЦ.467637.003 ПЗ*	Пояснювальна записка	11		
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						

Зм.	Лист	№ докум.	Підпис	Дата	*ІАЛЦ.462637.001 ВП*		
Розроб.		Журо			Відомість проекта	Літера	Лист 1 / Листів 1
Перев.		Симоненко					
Н.контр.							
Затв.							

ТЕХНІЧНЕ ЗАВДАННЯ

ЗМІСТ

1. Найменування та область застосування

В даній курсовій роботі буде розглянуто один з алгоритмів планування для обислювальної системи, що має топологію «матриця процесорів». Розроблений програмний продукт виконує занурення заданого користувачем графа на матрицю процесорів разміром m на m. Сферою застосування програмного продукта є моделювання, вивчення роботи складної паралельної обчислювальної системи типу «матриця процесорів», в якій кожний процесор дозволяє одночасно опрацьовувати і передавати дані, а також разрабка нових алгоритмів занурення графа завдання на різні матриці процессорів з метою визначення оптимального розміру матриці.

2. Підстави для розробки

Підставою для розробки є задання на курсовий проект, а також закріплення отриманих знаннь.

3. Мета і застосування

Метою разробки даного програмного продукту є закріплення вмінь та навичок у програмувані на мовах високого рівня, а також закріплення знань про паралельні обчислювальні системи, алгоритмах планування, отриманих при вивчені курсу «Системне програмне забезпечення».

4. Вимоги до програми

4.1. Вимоги до програмної моделі

Дана програмна модель дозволяє занурювати графи на топологію «матриця процесорів». Одна з основних вимог є занурення на матрицю процесорів оптимального разміру.

Програма повина занурювати граф завдання на задану топологію обчислювальної системи. В програмі повино бути передбачено ручний режим занурення. Також програма повина виконувати занурення всіх вершин на один процесор і занурення по певному власному алгоритму.

4.2. Вимоги до змісту і параметрів технічних засобів

Дана програма повинна бути розроблена на одній з мов високого рівня. Оформити результати роботи у вигляді технічної документації на проект, що включає технічне завдання, пояснювальну записку, необхідні відомості відповідно до госту.

4.3. Вимоги до надійності

Програма повина виконувати моделювання для матриці процесорів будь-якого разміру і при будь-якій кількості вершин графа завдання.

Надійність програмного продукту повинна бути перевірена в результаті тестування на комп'ьютерах різної продуктивності, на різних версіях операційної системи Windows і з різними вхідними параметрами.

4.4. Вимоги до програмної сумістності

Програма повина працювати в середовищах сумісних з ОС Windows.

5. Вимоги до програмної документації

Курсова работа повинна включати наступні документи:

1. Технічне завдання
2. Пояснювальна записка
3. Додаток (лістинг програми)

6. Стадії та етапи розробки

1. Узгодження технічного завдання
2. Розробка схеми алгоритмів і структури даних
3. Розробка програмного забезпечення
4. Тестування програмного забезпечення
5. Оформлення документів
6. Здача проекту

ПОЯСНЮВАЛЬНА ЗАПИСКА

ЗМІСТ

					ІАЛЦ.462637.003 ПЗ		
Зм.	Лист	№ докум.	Підпис	Дата			
Розроб.		Журо			Пояснювальна записка	Літера	Лист 1 / Листів 11
Перев.		Симоненко					
						НТУУ “КПІ” ФІОТ	
Н.контр.							
Затв.							

ВСТУП

У зв'язку з розвитком науки і техніки людству в даний час доводиться оперувати великими обсягами інформації, а також вирішувати складні завдання або безліч завдань. Звичайні персональні комп'ютери не справляються з дуже складними завданнями, або вирішують їх дуже довго, в той час як нас цікавить мінімальний час обробки. Для вирішення цих проблем і були створені мультипроцесорні і мультикомп'ютерні паралельні обчислювальні системи, які складаються з високотехнологічних, високопродуктивних елементів. Але ж наявні ресурси необхідно використовувати з максимальною ефективністю, а ефективність обробки розпаралелених завдань багато в чому залежить від організації обчислень, тобто від того, як і яким чином побудовано управління обчисленнями і вирішені завдання розподілу робіт і ресурсів системи. Планування розподілу завдань на ресурси паралельної обчислювальної системи допомагає вирішити проблему ефективного використання апаратних засобів. Однак планування в багатопроцесорних комплексах і мережах є однією з найбільш важко вирішуємих завдань.

У загальному вигляді обчислювальну систему і обчислювальний вузол (як об'єкт планування) можна розглядати з різним ступенем деталізації як сукупність ресурсів. Крім цього, об'єктами планування обчислень або "завдань" на ресурси можуть бути також і програми, процедури, паралельні ділянки програм, окремі блоки команд, команди і окремі макро- або мікрооперації. Таким чином, об'єктом системи планування ПОС може бути один або безліч ресурсів ОС, або безліч завдань, що поступають на вхід ОС, або безліч зв'язків завдання-ресурс. Об'єкт планування може змінюватися в залежності від цілей завдання планування: або це завантаженість устаткування, або ефективність обслуговування, що надходять на вхід ОС завдань, або і те й інше.

У цій роботі була розроблена програма занурення графа завдань на топологію «матриця процесорів».

1. Система планування для паралельних ОС

Завдання розподілу робочого навантаження вузлів розподіленої ОС в часі може бути сформульована як вибір переміщуваного процесу, моменту переміщення процесу і місця призначення переміщуваного процесу, що забезпечують підвищення загальної ефективності розподіленої системи. Для оцінки ефективності системи найчастіше використовуються наступні показники: пропускна здатність розподіленої системи, час знаходження завдання в системі (час відповіді), довжина черги і час очікування в черзі до ресурсу системи.

Класична трирівнева модель планування обчислювального процесу прийнятна в ОС з невеликим числом процесорів і складається з 3 типів планувальників:

- Планувальник високого рівня (ПВ);

ПВ призначений для попереднього планування вхідного потоку заявок, які надходять в систему і претендують на захоплення ресурсів обчислювальної системи. Його іноді називають планувальником доступу, так як він визначає, яким завданням можна надати доступ до системи. З усіх цих завдань ПВ створює чергу за деякими критеріями: відносні пріоритети, терміни запуску та завершення, час виконання, інтенсивність введення / виведення даних, потреба пам'яті.

- Проміжний планувальник (ПП);

ПП визначає, яким завданням буде дозволено конкурувати за захоплення ресурсів ОС В результаті роботи ПП формується черга, в якій знаходяться роботи (завдання), які ОС прийняла для обробки і виділила основні ресурси, крім часу процесора. Крім цього, планувальник проміжного рівня, реалізуючи функції, пов'язані з ефективністю роботи ОС, може тимчасово припиняти і активізувати (або відновлювати) процеси для досягнення безперебійної роботи ОС. Таким чином планувальник проміжного рівня діє як інтерфейс між доступом завдань в систему і розподілом ресурсів за цими завданнями.

- Планувальник низького рівня (ПН).

ПН виконує такі функції: визначає якому процесу, готовому до виконання, буде призначений процесор, який стає доступним, тобто ПН фактично призначає процесор цьому процесу; забезпечує прийом незавершених — перерваних завдань; виконує розподіл ресурсів без конфліктів; забезпечує завантаження завдань на відповідні ресурси за розробленим розкладом для виконання.

Тільки після цього починається процес виконання завдань.

2. Основні методи вирішення задач оптимізації

Існує три основних метода оптимізації:

Генетичний алгоритм. Суть алгоритму полягає в наступному: беремо базове рішення і змінюючи параметри знаходимо інше рішення. При цьому дивимося отримане рішення краще базового чи ні. Недоліком такого методу оптимізації є те, що оптимальний варіант отримуємо за велику кількість кроків. З цієї причини даний метод практично не використовується.

Метод оціночних функцій. Ціль методу — розбиття на зони. Ми оцінюємо чи потрапило рішення в якусь зону. Виводимо перше завдання на процесор і повинні визначити яку задачу потрібно занурити на той же процесор. Причому кожному рішенню присвоюємо вагу. Чим більше вага тим більше ступінь претендування.

Метод покрокового конструювання. Цей метод ще називають «Метод спрямованого пошуку». За алгоритмом спочатку потрібно піти у область прийнятних значень, тобто потрібно спробувати виключити свідомо невірні варіанти. Потім описуємо оптимальний варіант, а після нього прийнятний.

3. Алгоритм планування

Алгоритм занурює будь-який граф на топологію "матриця процесорів" і полягає в наступному:

1. Визначаємо початкову вершину і занурюємо її на процесор (1).

2. Для кожного вільного процесора формуємо чергу доступних заявок.

3. З цієї черги на процесор занурюємо ту заявку, яка не повторюється в чергах до інших процесорів, або ту, в якій час пересилки більше (якщо вони на одному ярусі), або ту, номер ярусу якої менше.

4. Позначаємо процесор як зайнятий, заявку закріплюємо за ним, видаляємо цю заявку з черг до інших процесорів.

5. Виконуємо п. 3-4 до тих пір, поки в чергах до процесорів не буде заявок, або до тих пір, коли процесори, до яких є черга заявок будуть зайняті.

6. Виконуємо крок планування, зменшуючи час виконання заявки в процесорі і збільшуючи загальний лічильник часу.

7. Перевіряємо, чи є вільні процесори.

8. Повторюємо п. 2-7 доти, доки не завершиться планування.

4. Приклад роботи програми

Запустимо програму і створимо граф, показаний на рисунку 1.

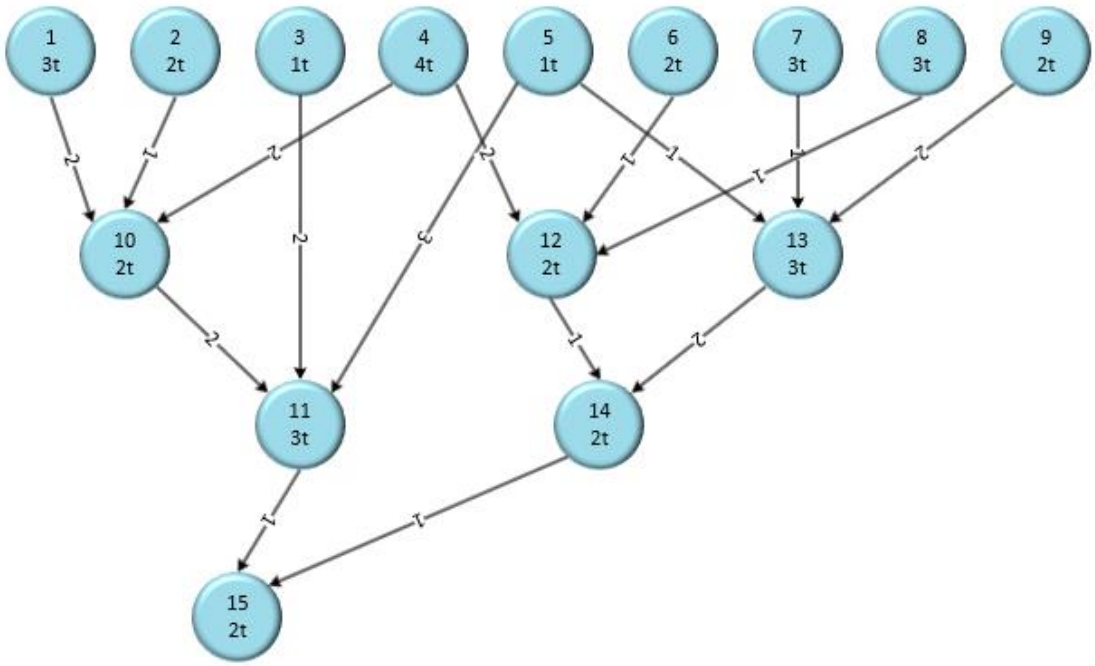

Рисунок 1. Створений граф

В таблиці 1 показані номера вершин і час їх виконання, а в таблиці 2 відображена матриця зв'язків, яка відповідає заданому графу.

Таблиця 1. Вершини графа і час їх виконання

№ вершини	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Час виконання	3	2	1	4	1	2	3	3	2	2	3	2	3	2	2

Таблиця 2. Матриця зв'язків заданого графа

№ вершини	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1										2					
2										1					
3											2				
4										2		2			
5											3		1		
6												1			
7													1		
8												1			
9													2		
10											2				
11															1
12														1	
13														2	
14															1

Розглянемо занурення заданого графа на матрицю процесорів розміром 3х3.

Завдання стає доступним процесору в тому випадку, якщо всі попередні завдання, від яких залежить дане, виконані, і час пересилки даних від всіх процесорів, що виконували попередні завдання минув.

Час пересилки даних буде визначатися ціною з'єднання. Якщо ціна пересилки 2, то буде витрачено по 2 такти на кожну пересилку даних з процесора на процесор. Час виконання завдання визначається його вагою. Якщо вага завдання 7, то на його обробку піде 7 тактів.

Процес занурення буде відбуватися таким чином, задачі першого ярусу будуть розподілені між такими процесорами: 1 задача на 1 процесор, 2 на 2, 3 на 2 і т.д. Відповідно задачі другого ярусу будуть оброблятися на

одному з процесорів де оброблялася одна з попередніх задач гілки графу, це дасть змогу рівномірно розділити задачі між усіма процесорами та досягнути мінімальних втрат при пересилці даних.

Час виконання даного графу на матриці процесорів 3x3 дорівнює 80 тактів. Такий результат пояснюється тим, що ціна деяких з'єднань досить велика, та на останніх етапах обчислення, 4 та 5 яруси, велику кількість тактів було втрачено на пересилці даних через один процесор.

У результаті занурення графа на матрицю процессорів 3x3, отримаємо (рисунок 2):

T	P1	P2	P3	P4	P5	P6	P7	P8	P9
1	1	4	3	2	6	8	7	5	9
2	1	4\R	S	2	6\R	8	7	S	9
3	1	4\R	S	S	R	8	7	R\S	S
4	S	4\R			R	S	S	R\S	S
5	S	R\S			R\S			13	
6		10\R\S			R\S			13	
7		10\R			12\S			13	
8		11			12\R			S	
9		11			R			S	
10		11			14				
11		S			14\R				
12					15				
13					15				
14									
15									
16									

Рисунок 2. Результат занурення графа

5. Опис можливостей програми

При запуску програми відкривається вікно, що показане на рисунку 3.

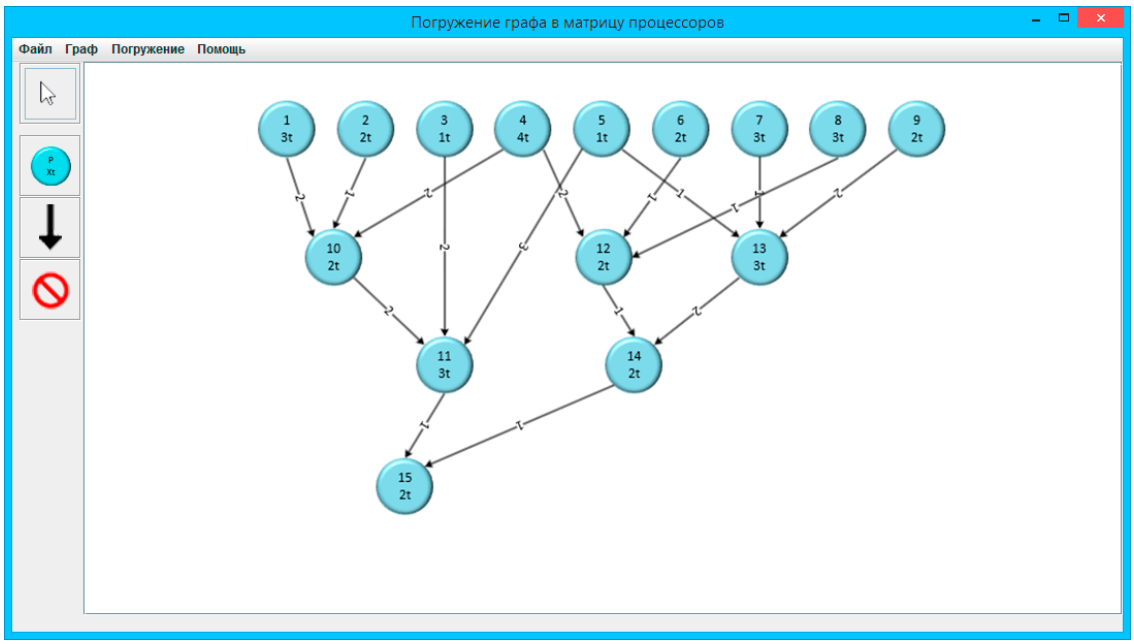

Рисунок 3. Зовнішній вигляд програми

В панелі інструментів є чотири кнопки:

1. Курсор – режим пересування вершин, їх виділення та зміни параметрів вершин і ребер графа. Щоб змінити параметри вершини або ребра, необхідно два рази клікнути на об'єкті, тоді відкриється діалогове вікно, що дозволяє змінити параметри конкретно зазначеної вершини або ребра (рисунок 4).

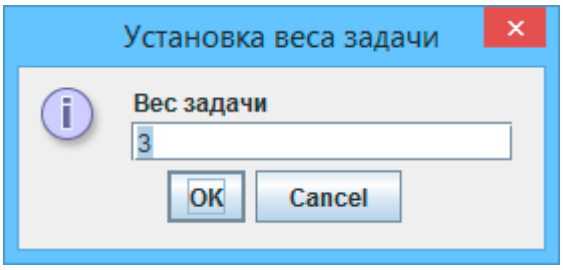

Рисунок 4. Вікно для зміни параметрів вершин або ребер графа

2. Вершина (кружок з номером 1). У цьому режимі можна додавати вершини, вказуючи курсором місце, куди хочемо додати вершину.

3. Лінія з стрілкою - додає зв'язок між вершинами. Спочатку необхідно клікнути мишкою на вершині, з якої виходить зв'язок, а потім на тій, в яку цей зв'язок входить і на графі з'явиться стрілка.

4. Видалити. Виділене ребро або вершину можна видалити з меню, або натисканням клавіші Delete. Спочатку необхідно клікнути на вершині, що

призведе до її видалення.

Перехід в режим пересування вершин і вимірювання їх параметрів здійснюється натисканням клавіші "E"

Перехід в режим додавання вершин також можна здійснити з меню або натиснувши клавішу "A"

Перехід в режим додавання зв'язків здійснюється шляхом натискання клавіші "L" або з меню.

Після створення графа і задання параметрів вершин і ребер можна переходити в режим занурення (рисунок 5).

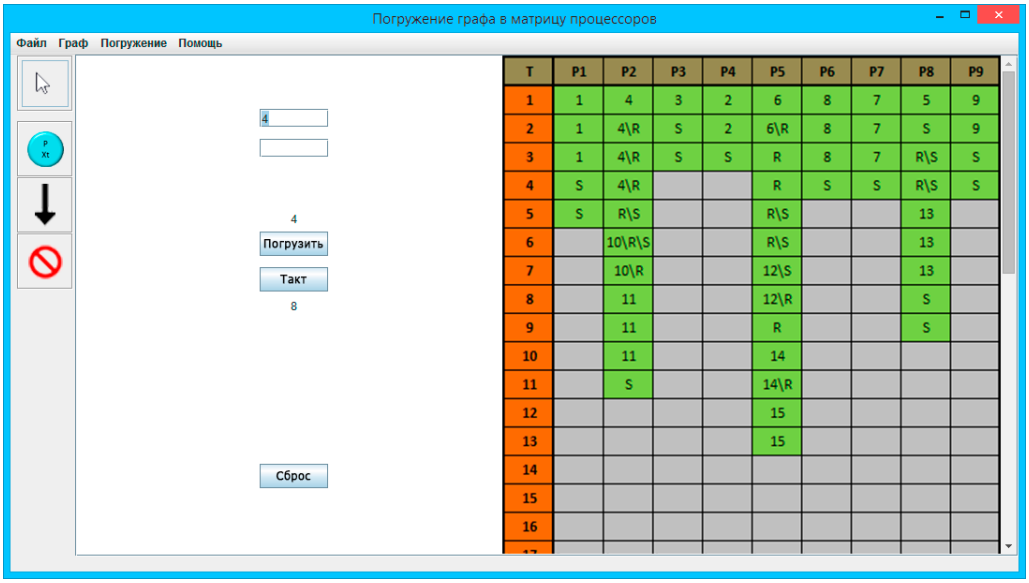

Рисунок 5. Режим занурення графа

В ручному режимі:

Якщо черга до процесора не пуста, то над ним лінія зафарбовується зеленим кольором. Натиснувши на процесорі ми побачимо чергу доступних заявок. Обираємо потрібний і натискаємо кнопку “Погрузить”, а далі кнопку “Такт”. Якщо на який-небудь процесор можно буде занурити деякі заявки, то над ним буде з'являтися прапорець.

Після закінчення занурення програма видає повідомлення пре те, що занурення завершено успішно.

В автоматичному режимі:

Програма занурить граф по вищевказаному алгоритму при натиску на кнопку “Погрузить”.

ВИСНОВОК

При виконанні курсової роботи були вивчені різні методи розв'язання задач оптимізації, розглянуто основні етапи планування, а також розроблено алгоритм планування для обчислювальних систем з топологією «матриця процесорів».

Для перевірки алгоритму була створена моделююча програма.

Виконання даної курсової роботи в рамках навчального плану істотно поліпшило навички в програмуванні та закріпило знання з дисципліни «Системне програмне забезпечення».

СПИСОК ЛІТЕРАТУРИ

1. Є. Таненбаум. «Сучасні операційні системи». 2-ге видання. СПб: Пітер, 2004. – 1040с.
2. В. П. Сімоненко. «Організація обчислювальних процесів».
3. З. В. Князькова, В. П. Сімоненко «Метод направленого пошуку при статичному плануванні задач в розподілених обчислювальних системах». 2002. №1-2. – 247-252с.
4. М. Гері, Д. Джонсон «Обчислювальні машини і важко вирішуваємі задачі». – М: Мир. 1982. – 416с.
5. Конспект лекцій з курсу «Системне програмне забезпечення».

ДОДАТОК

Лістинг програми

Основний модуль

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs,
  ImgList, ComCtrls, ToolWin, ExtCtrls, Menus,
  GraphClass, StdCtrls, Grids;

type
  TTask = record
    time: integer;
    x1: integer;
    y1: integer;
    x2: integer;
    y2: integer;
    exec: boolean; {флаг про те, що задача виконана}
    inp1: integer; {ряд процесора}
    inp2: integer; {стовпчик процесора}
    timeend: integer; {час завершення}
  end;

  tinproc = record
    number: integer;
    time: integer;
  end;

  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    N9: TMenuItem;
    N10: TMenuItem;
    Help1: TMenuItem;
    N11: TMenuItem;
    N12: TMenuItem;
    About1: TMenuItem;
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    ScrollBox1: TScrollBox;
    PaintBox1: TPaintBox;
    ToolBar1: TToolBar;
    StatusBar1: TStatusBar;
    ToolButton1: TToolButton;
    ImageList1: TImageList;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
    N13: TMenuItem;
    ScrollBox2: TScrollBox;
    PaintBox2: TPaintBox;
    Edit1: TEdit;
    UpDown1: TUpDown;
    Edit2: TEdit;
    UpDown2: TUpDown;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    Button1: TButton;
    StringGrid1: TStringGrid;
    Button2: TButton;
    StaticText3: TStaticText;
    Button3: TButton;
    procedure PaintBox1MouseDown(Sender: TObject;
Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure ToolButton1Click(Sender: TObject);
    procedure ToolButton2Click(Sender: TObject);
    procedure ToolButton3Click(Sender: TObject);
    procedure PaintBox1MouseUp(Sender: TObject;
Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure PaintBox1MouseMove(Sender: TObject;
Shift: TShiftState; X,
      Y: Integer);
    procedure PaintBox1Paint(Sender: TObject);
    procedure PaintBox1DblClick(Sender: TObject);
    procedure N8Click(Sender: TObject);
    procedure N7Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure N9Click(Sender: TObject);
    procedure N13Click(Sender: TObject);
    procedure PaintBox2Paint(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure UpDown2Changing(Sender: TObject; var
AllowChange: Boolean);
    procedure PaintBox2MouseDown(Sender: TObject;
Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure Button1Click(Sender: TObject);
    procedure StringGrid1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

Procedure FillMatrix;
Procedure FillTask;

var
  Form1: TForm1;

  Vertex: array[1..100]of TVertex;
  Line: array[1..200]of TLine;
  Matrix,copymatrix,connections:
array[1..100,1..100]of integer;
  Task: array[1..100]of TTask;
  M: integer; {розмірність системи процесорів}
  access: array[1..100] of integer;
  proc: array[1..5,1..5]of Tinproc;
  Count: integer;  {вершин}
  CountLine: integer; {ліній}
  taktcount: integer; {тактів}
  PrevSelectVertex,PrevSelectLine: integer;
  xpos,ypos: integer;
  FAddVertex, FAddLine, FMove, FMouseDown: boolean;
  Flinestart,Fpause: boolean;

implementation

uses Unit2;

{$R *.DFM}

Procedure Reset;
begin
  with form1 do
  begin
   count:=0;
   countline:=0;
   Toolbutton1.Down:=false;
   Toolbutton2.Down:=false;
   Toolbutton3.Down:=false;
   FMove:=false;
   Faddline:=false;
   Faddvertex:=false;
   FMouseDown:=false;
   flinestart:= true;
```

```
       fillmatrix;
       taktcount:=1;
      end;
    end;

    Procedure FillTask;
      var i: integer;
    begin
       for i:=1 to count do
       begin
          task[i].time:=vertex[i].GetTime;
       end;
    end;

    Procedure ResetTask;
      var i:  integer;
    Begin
      for i:=1 to count do
      begin
        task[i].time:=vertex[i].GetTime;
        task[i].x1:=0;
        task[i].y1:=0;
        task[i].exec:=false;
        task[i].inp1:=0;
        task[i].inp2:=0;
        task[i].timeend:=0;
      end;
    End;

    Procedure FillMatrix;
     var i,j: integer;
    begin
      for i:=1 to count do
        for j:=1 to count do
           Matrix[i,j]:=0;
      for i:=1 to countline do
      begin

Matrix[line[i].GetBegVertex,line[i].GetEndVertex]:=line[
i].GetTime;
      end;
      copymatrix:=matrix;
      connections:=matrix;
    end;

    Function FillAccess(p1,p2: integer): integer;
      var i,j,k,cnt,tmp,put,maxput,prevnum,prnum:
integer;
    begin
      cnt:=0;
      for i:=1 to count do
      begin
        tmp:=0;
        for j:=1 to count do
           tmp:=tmp+copymatrix[j,i];

        prnum:=0;
        maxput:=0;

        if tmp=0 then
          for k:=1 to count do
           if (matrix[k,i]<>0) then
           begin
             prevnum:=k;
             put:=abs(task[prevnum].inp1-
p1)+abs(task[prevnum].inp2-p2);

             if task[prevnum].timeend<>0 then

put:=put*matrix[prevnum,i]+task[prevnum].timeend
             else put:=put*matrix[prevnum,i]+100;

             if (maxput<=put) then
             begin
               maxput:=put;
               prnum:=prevnum;
             end;
           end;

        put:=0;
        prevnum:=prnum;
        if prevnum<>0 then
        begin
          put:=abs(task[prevnum].inp1-
p1)+abs(task[prevnum].inp2-p2);
          put:=put*matrix[prevnum,i];
          if (put+task[prevnum].timeend)<=taktcount then
             put:=0;
        end;


        if
(tmp=0)and(proc[p1,p2].number=0)and(task[i].exec=false)
        and(task[i].inp1=0)and(put=0) then
        begin
           cnt:=cnt+1;
           access[cnt]:=i;
        end;
      end;
      fillaccess:=cnt;
      access[cnt+1]:=0;
      form1.StringGrid1.ColCount:=cnt;
      for i:=1 to cnt+1 do
        form1.StringGrid1.Cells[i-
1,0]:=inttostr(access[i]);
    end;

    Procedure Takt;
      var i,j,k: integer;
          fend: boolean;
    begin
      taktcount:=taktcount+1;
      for i:=1 to M do
        for j:=1 to M do
        begin
           if Proc[i,j].time<>0 then
              Proc[i,j].time:=Proc[i,j].time-1;

           if Proc[i,j].time=0 then
           begin
              if proc[i,j].number<>0 then
              begin
                 task[proc[i,j].number].exec:=true;

task[proc[i,j].number].timeend:=taktcount;
                 for k:=1 to count do
                    copymatrix[proc[i,j].number,k]:=0;
                 proc[i,j].number:=0;
              end;
           end;

      end;
      fend:=true;
      for i:=1 to count do
        if task[i].exec=false then
           fend:=false;
      if fend then
        application.messagebox(Pchar('Граф погружен
успешно!'),'Поздравляю!',0);
      form1.PaintBox2.Refresh;
    end;

    Procedure AddVertex(x,y: integer; canvas: TCanvas);
    begin
      Count:=Count+1;
      Vertex[count]:=TVertex.Create;
      Vertex[count].SetCoords(x,y);
      Vertex[count].SetNum(Count);
      Vertex[count].Draw(canvas);
    end;

    Procedure DelVertex(Num: integer; canvas: TCanvas);
      var i: integer;
    begin
      for i:=num to count-1 do
      begin
        vertex[i]:=vertex[i+1];
        Vertex[i].SetNum(i);
      end;
      count:=count-1;
```

```
     form1.PaintBox1.Refresh;
   end;

   Function SelectVertex(x,y: integer): integer;
    var i,d: integer;
   begin
      SelectVertex:=0;
      for i:=1 to count do
      begin
         d:=round(sqrt(sqr(x-vertex[i].GetX)+sqr(y-
vertex[i].GetY)));
         if d<20 then
            SelectVertex:=i;
      end;
   end;

   Procedure AddLine(begx,begy: integer; canvas:
tcanvas);
   begin
     Countline:=Countline+1;
     line[countline]:=Tline.Create;
     line[countline].SetBegXY(begx,begy);
     line[countline].SetNumber(Countline);
   end;

   Procedure DelLine(Num: integer; canvas: TCanvas);
     var i: integer;
   begin
     for i:=num to countline-1 do
     begin
       line[i]:=line[i+1];
       line[i].SetNumber(i);
     end;
     countline:=countline-1;
     form1.PaintBox1.Refresh;
   end;

   Function SelectLine(x,y: integer): integer;
    var i,tmp,tmp1: integer;
   begin
      SelectLine:=0;
      for i:=1 to countline do
      begin
         tmp:=((line[i].GetEndY-
line[i].GetBegY)*(line[i].GetEndX-x));
         tmp1:=((line[i].GetEndX-
line[i].GetBegX)*(line[i].GetEndY-y));
         if (abs(tmp-
tmp1)<400)and(((x>=line[i].GetBegX)and(x<=line[i].GetEnd
X))or

((x<=line[i].GetBegX)and(x>=line[i].GetEndX))) then
            SelectLine:=i;
      end;
   end;

   Procedure DrawAll(canvas: TCanvas);
    var i: integer;
   begin
      for i:=1 to countline do
         Line[i].Draw(canvas);
      for i:=1 to count do
         Vertex[i].Draw(canvas);
   end;

   procedure TForm1.PaintBox1MouseDown(Sender: TObject;
Button: TMouseButton;
     Shift: TShiftState; X, Y: Integer);
     var num,numln: integer;
   begin
     FMouseDown:=true;
     with form1.PaintBox1 do
     begin

      if FMove then
      begin
         xpos:=x;
         ypos:=y;
         num:=SelectVertex(x,y);
         numln:=SelectLine(x,y);
         if (num<>0)then
         begin
           if prevselectvertex<>0 then
           begin
              vertex[prevselectvertex].SetSelect(false);
           end;
           vertex[num].SetSelect(true);
           vertex[num].Draw(canvas);
           prevselectvertex:=num;
         end
         else
           if prevselectvertex<>0 then
           begin
             vertex[prevselectvertex].SetSelect(false);
             prevselectvertex:=0;
           end;

         if (numln<>0)and(num=0)then
         begin
           if prevselectline<>0 then
           begin
              line[prevselectline].SetSelect(false);
           end;
           line[numln].SetSelect(true);
           line[numln].Draw(canvas);
           prevselectline:=numln;
         end
         else
           if prevselectline<>0 then
           begin
             line[prevselectline].SetSelect(false);
             prevselectline:=0;
           end;

      end;

      if (Faddvertex)and(x>20)and(x<paintbox1.Width-20)
        and(y>20)then
      begin
        if y+20>paintbox1.Height then
        begin
          paintbox1.Height:=y+100;
          DrawAll(canvas);
        end;
        AddVertex(x,y,canvas);
      end;

      if Faddline then
      begin
         if (flinestart)and(SelectVertex(x,y)<>0) then
         begin

AddLine(vertex[SelectVertex(x,y)].GetX,vertex[SelectVert
ex(x,y)].GetY,canvas);

line[countline].SetBegVertex(SelectVertex(x,y));
           flinestart:=false;
         end
         else
         if
(flinestart=false)and(SelectVertex(x,y)<>0)and(SelectVer
tex(x,y)<>line[countline].GetBegVertex) then
         begin

line[countline].SetEndXY(vertex[SelectVertex(x,y)].GetX,
vertex[SelectVertex(x,y)].GetY);

line[countline].SetEndVertex(SelectVertex(x,y));
           flinestart:=true;
         end;
      end;

      paintbox1.Refresh;
     end;

   end;

   procedure TForm1.ToolButton1Click(Sender: TObject);
   begin
      Toolbutton1.Down:=true;
```

```
      Toolbutton2.Down:=false;
      Toolbutton3.Down:=false;
      FMove:=true;
      Faddline:=false;
      Faddvertex:=false;
   end;

   procedure TForm1.ToolButton2Click(Sender: TObject);
   begin
      Toolbutton1.Down:=false;
      Toolbutton2.Down:=true;
      Toolbutton3.Down:=false;
      FMove:=false;
      Faddline:=false;
      Faddvertex:=true;

   end;

   procedure TForm1.ToolButton3Click(Sender: TObject);
   begin
      Toolbutton1.Down:=false;
      Toolbutton2.Down:=false;
      Toolbutton3.Down:=true;
      FMove:=false;
      Faddline:=true;
      Faddvertex:=false;

   end;

   procedure TForm1.PaintBox1MouseUp(Sender: TObject;
Button: TMouseButton;
     Shift: TShiftState; X, Y: Integer);
   begin
     FMouseDown:=false;
   end;

   procedure TForm1.PaintBox1MouseMove(Sender: TObject;
Shift: TShiftState; X,
     Y: Integer);
     var i: integer;
   begin
     If
(FMove)and(FMouseDown)and(prevselectvertex<>0)then
     begin
        vertex[prevselectvertex].setcoords(x,y);
        for  i:=1 to countline do
        begin
           if line[i].GetBegVertex=prevselectvertex then
              line[i].SetBegXY(x,y);
           if line[i].GetEndVertex=prevselectvertex then
              line[i].SetEndXY(x,y);
        end;
        paintbox1.Refresh;
        vertex[prevselectvertex].Draw(canvas);
     end;
   end;

   procedure TForm1.PaintBox1Paint(Sender: TObject);
   begin
     DrawAll(form1.paintbox1.canvas);
   end;

   procedure TForm1.PaintBox1DblClick(Sender: TObject);
   begin
     if (prevselectvertex<>0)and(fmove) then
       form2.show;
     if (prevselectline<>0)and(fmove) then
       form2.show;
   end;

   procedure TForm1.N8Click(Sender: TObject);
     var i,j: integer;
   begin

     if prevselectvertex<>0 then
     begin
       DelVertex(prevselectvertex,paintbox1.Canvas);
       for i:=1 to countline do
         for j:=1 to countline do
           if
(line[j].GetBegVertex=prevselectvertex)or(line[j].GetEnd
Vertex=prevselectvertex)then
             DelLine(j,paintbox1.Canvas);
     for i:=1 to countline do
     begin
       if line[i].GetBegVertex>prevselectvertex then
          line[i].SetBegVertex(line[i].getbegvertex-1);
       if line[i].GetEndVertex>prevselectvertex then
          line[i].SetEndVertex(line[i].getendvertex-1);
     end;

     end;

     if prevselectline<>0 then
       DelLine(prevselectline,paintbox1.Canvas);

     fillmatrix;
   end;

   procedure TForm1.N7Click(Sender: TObject);
   begin
     toolbutton2.Click;
   end;

   procedure TForm1.FormShow(Sender: TObject);
   begin
     reset;
   end;

   procedure TForm1.FormResize(Sender: TObject);
   begin
     scrollbox1.Width:=pagecontrol1.Width-20;
     scrollbox1.Height:=pagecontrol1.Height-20;
     paintbox1.Width:=pagecontrol1.Width-20;
     scrollbox2.Width:=pagecontrol1.Width-20;
     scrollbox2.Height:=pagecontrol1.Height-40;
     paintbox1.Width:=pagecontrol1.Width-20;
   end;

   procedure TForm1.N9Click(Sender: TObject);
   begin
     toolbutton3.Click;
   end;

   procedure TForm1.N13Click(Sender: TObject);
   begin
     toolbutton1.Click;
   end;

   Procedure Shkala(canvas: Tcanvas);
     var N,i: integer;
   begin
     N:=40;
     for i:=1 to N do
     begin
       canvas.MoveTo(0,i*10+20);
       canvas.LineTo(10,i*10+20);
       canvas.TextOut(12,i*10+15,inttostr(i));
     end;
     form1.PaintBox2.Height:=N*10+30;
   end;

   procedure TForm1.PaintBox2Paint(Sender: TObject);
    var i: integer;
   begin
     with form1.PaintBox2 do
     begin
       shkala(canvas);
       paintproc(strtoint(edit1.text),canvas);
       for i:=1 to count do
        if task[i].x1<>0 then
          painttask(i,canvas);
       paintconnections(canvas);
     end;
   end;

   Procedure PaintProc(num: integer; canvas: Tcanvas);
    var i,j,tmp,dost: integer;
   begin
```

```
     tmp:=40;
     for i:=1 to num do
       for j:=1 to num do
       begin
          dost:=fillaccess(i,j);
          if dost<>0 then
            canvas.Pen.Color:=cllime;
          canvas.MoveTo(tmp,8);
          canvas.LineTo(tmp+20,8);
          canvas.TextOut(tmp,10,inttostr(i)+' ,
'+inttostr(j));
          canvas.Pen.Color:=clblack;
          tmp:=tmp+40;
       end;
     M:=num;
     form1.paintbox2.Width:=tmp+40;
   end;

   Procedure SetTaskCoords(p1,p2,number,start: integer);
   begin
      Task[number].x1:=(p1-1)*M*40+(p2)*40;
      Task[number].y1:=start*10+20;
      Task[number].x2:=(p1-1)*M*40+(p2)*40+20;
      task[number].y2:=start*10+20+task[number].time*10;
   end;

   procedure TForm1.Button1Click(Sender: TObject);
    var i,p1,p2: integer;
   begin
     if
(edit2.Text<>'')and(statictext1.Caption<>'0')and(statict
ext1.Caption<>'0')then
     begin
        p1:=strtoint(statictext1.caption);
        p2:=strtoint(statictext2.caption);

SetTaskCoords(p1,p2,strtoint(edit2.text),taktcount);

PaintTask(strtoint(edit2.text),paintbox2.Canvas);
        proc[p1,p2].number:=strtoint(edit2.text);

proc[p1,p2].time:=task[strtoint(edit2.text)].time;
        task[strtoint(edit2.text)].inp1:=p1;
        task[strtoint(edit2.text)].inp2:=p2;
        fpause:=false;
        for i:=1 to count do
          matrix[i,strtoint(edit2.text)]:=0;
        edit2.Text:='';
        fillaccess(p1,p2);
     end;
     form1.ScrollBox2.Refresh;
   end;

   Procedure PaintConnections(canvas: Tcanvas);
     var i,j,k,l: integer;
         yes: boolean;
   begin
     for i:=1 to count do
       for j:=1 to count do
       begin
         yes:=false;
         for k:=1 to M do
           for l:=1 to M do
             if proc[k,l].number=j then
               yes:=true;
         if
(connections[i,j]<>0)and(task[i].exec=true)and((task[j].
exec=true)or(yes=true))

and((task[i].inp1<>task[j].inp1)or(task[i].inp2<>task[j]
.inp2)) then
         begin
           canvas.MoveTo(task[i].x2,task[i].y2);
           canvas.LineTo(task[j].x1,task[j].y1);
         end;
       end;
   end;

   procedure TForm1.Edit1Change(Sender: TObject);
   begin
      form1.PaintBox2.Refresh;
   end;

   procedure TForm1.UpDown2Changing(Sender: TObject;
     var AllowChange: Boolean);
   begin
      if access[updown2.Position]<>0 then
        edit2.Text:=inttostr(access[updown2.Position])
      else
        edit2.Text:='';
   end;

   Procedure PaintTask(number: integer; canvas:
Tcanvas);
   begin

canvas.Rectangle(Task[number].x1,Task[number].y1,Task[nu
mber].x2,Task[number].y2);

canvas.TextOut(Task[number].x1+5,Task[number].y1+10,intt
ostr(number));
   end;

   procedure TForm1.StringGrid1Click(Sender: TObject);
   begin
     edit2.text:=stringgrid1.Cells[stringgrid1.col,0];
   end;

   procedure TForm1.PaintBox2MouseDown(Sender: TObject;
Button: TMouseButton;
     Shift: TShiftState; X, Y: Integer);
     var P1,P2: integer;
   begin
     GetP1P2(x,y,p1,p2);
     statictext1.Caption:=inttostr(p1);
     statictext2.Caption:=inttostr(p2);
     fillaccess(p1,p2);
   end;

   Procedure GetP1P2(x,y: integer; var p1,p2: integer);
     var tmp,i,j: integer;
   begin
     p1:=0;
     p2:=0;
     tmp:=0;
     repeat
        tmp:=tmp+1;
        x:=x-40;
     until (x<=40);
     for i:=1 to M do
       for j:=1 to M do
       begin
          tmp:=tmp-1;
          if tmp=0 then
          begin
            p1:=i;
            p2:=j;
          end;
       end;
   end;

   procedure TForm1.Button2Click(Sender: TObject);
   begin
     takt;

fillaccess(strtoint(statictext1.caption),strtoint(static
text2.caption));
     statictext3.Caption:=inttostr(taktcount);
   end;

   procedure TForm1.Button3Click(Sender: TObject);
   begin
     taktcount:=1;
     fillmatrix;
     resettask;
     form1.ScrollBox2.Refresh;
   end;

   end.
```

Клас вершин графа та його ребер

```
unit GraphClass;

interface

Uses Graphics, Sysutils;

Type
TVertex = class
Private
  x: integer;
  y: integer;
  Number: integer;
  Times: integer;
  Selected: boolean;
Public
  Constructor Create;
  Function GetX: integer;
  Function GetY: integer;
  Function GetNum: integer;
  Function GetTime: integer;
  Procedure SetCoords(ax,ay: integer);
  Procedure SetNum(Num: integer);
  Procedure SetTime(Time: integer);
  Procedure Draw(canvas: TCanvas);
  Procedure Move(dx,dy: integer);
  Procedure SetSelect(sel: boolean);
End;

TLine = class
private
  number: integer;
  begvertex: integer;
  endvertex: integer;
  begx: integer;
  begy: integer;
  endx: integer;
  endy: integer;
  Times: integer;
  selected: boolean;
public
  Constructor Create;
  Procedure SetNumber(num: integer);
  Procedure SetBegXY(x,y: integer);
  Procedure SetEndXY(x,y: integer);
  Procedure SetBegVertex(num: integer);
  Procedure SetEndVertex(num: integer);
  Procedure SetTime(Time: integer);
  Procedure Draw(canvas: Tcanvas);
  Procedure SetSelect(select: boolean);
  Function GetBegVertex: integer;
  Function GetEndVertex: integer;
  Function GetTime: integer;
  Function GetBegX: integer;
  Function GetBegY: integer;
  Function GetEndX: integer;
  Function GetEndY: integer;
End;

implementation

{ TVertex }

constructor TVertex.Create;
begin
   x:=20;
   y:=20;
   Number:=0;
   Times:=0;
   Selected:=false;
end;

function TVertex.GetX: integer;
begin
   GetX:=x;
end;

function TVertex.GetY: integer;
begin
   GetY:=y;
end;

function TVertex.GetNum: integer;
begin
   GetNum:=Number;
end;

function TVertex.GetTime: integer;
begin
   GetTime:=Times;
end;

procedure TVertex.SetNum(Num: integer);
begin
   Number:=Num;
end;

procedure TVertex.SetSelect(sel: boolean);
begin
    selected:=sel;
end;

procedure TVertex.SetTime(Time: integer);
begin
   Times:=Time;
end;

procedure TVertex.Move(dx, dy: integer);
begin
   x:=x+dx;
   y:=y+dy;
end;

procedure TVertex.SetCoords(ax, ay: integer);
begin
  x:=ax;
  y:=ay;
end;

procedure TLine.Draw(canvas: Tcanvas);
var a1,b1,c1,x1,y1,x2,y2,k,b: real;
begin

  if (endx<>0)and(endy<>0)then
  begin

    if begy<>endy then
    begin
      k:=(begx-endx)/(begy-endy);
      b:=(begy-k*begx);
      a1:=1+sqr(k);
      b1:=(begx+begy*k-b*k);
      c1:=(sqr(begx)+sqr(begy-b)-25);
      x1:=abs((-b1+sqrt(sqr(b1)-a1*c1))/a1);
      x2:=abs((-b1-sqrt(sqr(b1)-a1*c1))/a1);
      y1:=k*x1+b;
      y2:=k*x2+b;
    end
    else
    begin
      y1:=begy-5;
      y2:=begy+5;
      x1:=begx;
      x2:=begx;
    end;

   if selected then
     canvas.Pen.Color:=cllime
   else
     canvas.Pen.Color:=clblack;
   canvas.MoveTo(round(x1),round(y1));
   canvas.LineTo(endx,endy);
   canvas.MoveTo(round(x2),round(y2));
   canvas.LineTo(endx,endy);
   canvas.TextOut(begx+round((endx-
begx)/2),begy+round((endy-begy)/2),inttostr(times));
  end;
end;
```

```
   procedure TVertex.Draw(canvas: TCanvas);
   begin
     if selected then
     begin
       canvas.Pen.Color:=clblue;
       canvas.Brush.Color:=clred;
     end
     else
     begin
       canvas.Pen.Color:=clblack;
       canvas.Brush.Color:=clnone;
     end;

     canvas.Ellipse(x-20,y-20,x+20,y+20);
     canvas.TextOut(x-3*(length(inttostr(Number))),y-
15,inttostr(Number));
     canvas.TextOut(x-
3*(length(inttostr(Times))),y+5,inttostr(Times));
   end;

   { TLine }

   constructor TLine.Create;
   begin
      begvertex:=0;
      endvertex:=0;
      begx:=0;
      begy:=0;
      endx:=0;
      endy:=0;
      number:=0;
      times:=0;
      selected:=false;
   end;

   function TLine.GetBegVertex: integer;
   begin
      GetBegVertex:=begvertex;
   end;

   function TLine.GetEndVertex: integer;
   begin
      GetEndVertex:=endvertex;
   end;

   function TLine.GetBegX: integer;
   begin
      GetBegX:=begx;
   end;

   function TLine.GetBegY: integer;
   begin
      GetBegY:=begy;
   end;

   function TLine.GetEndX: integer;
   begin
      GetEndX:=endx;
   end;

   function TLine.GetEndY: integer;
   begin
      GetEndY:=endy;
   end;

   function TLine.GetTime: integer;
   begin
      GetTime:=times;
   end;

   procedure TLine.SetBegVertex(num: integer);
   begin
      begvertex:=num;
   end;

   procedure TLine.SetEndVertex(num: integer);
   begin
      endvertex:=num;
   end;

   procedure TLine.SetBegXY(x, y: integer);
   begin
      begx:=x;
      begy:=y;
   end;

   procedure TLine.SetEndXY(x, y: integer);
   begin
      endx:=x;
      endy:=y;
   end;

   procedure TLine.SetNumber(num: integer);
   begin
      number:=num;
   end;

   procedure TLine.SetTime(Time: integer);
   begin
      times:=time;
   end;

   procedure TLine.SetSelect(select: boolean);
   begin
     selected:=select;
   end;

   end.
```