ПРИЛОЖЕНИЕ 1. Описание программы

П1.1. Общие сведения о программном комплексе

При выполнении данной работы была разработана программа – «MAX TSP», связанная с решением задачи коммивояжера на максимум. Программа служит для поиска оптимальных и приближенных к оптимальным гамильтоновых циклов с помощью различных алгоритмов в симметрических и для анализа и сравнения самих алгоритмов на случайных графах.

Программа написана на языке C++ в среде разработки Borland Turbo C++. Язык C++ был выбран, потому что он является объектно-ориентированным языком высокого уровня. Выбор среды программирования объясняется тем, что Borland Turbo C++ позволяет легко разрабатывать как сам код программы, так и удобный пользовательский интерфейс Программа функционирует под управлением операционной системы Microsoft Windows. Она является приложением на основе Windows форм.

Для выполнения программ необходимо присутствие в системе следующих библиотек:

- bcbsmp60.bpl;
- borlndmm.dll;
- cc3260mt.dll;
- rtl60.bpl;
- vcl60.bpl.

Указанные библиотеки могут находиться в папке с исполняемым файлом программы, либо быть установленными в системе (это происходит автоматически при инсталляции среды разработки).

П1.2. Логическая структура программы

Для решения задачи коммивояжера программа использует следующие алгоритмы:

- алгоритм координатного подъема;
- градиентный алгоритм с форой;
- алгоритм Сердюкова;
- улучшенный алгоритм Сердюкова;
- метод ветвей и границ.

Помимо алгоритмов нахождения гамильтонового цикла в программах реализованы следующие алгоритмы:

- нахождение гамильтонового пути максимального веса полным перебором;
- нахождение паросочетания максимального веса;
- нахождение 2-фактора максимального веса.

Описание этих алгоритмов можно найти в главе 3 пояснительной записки.

В состав программы входят следующие модули:

- TGraph.cpp – модуль определения класса TGraph;
- TGraphMarks.cpp – модуль определения класса TGraphMarks;
- TMarkedGraph.cpp – модуль определения класса TMarkedGraph;
- TEdge.cpp – модуль определения класса TEdge;
- MatchingAlgs.cpp – модуль, содержащий функции для нахождения различных подграфов максимального веса в графе;
- BaBAlg.cpp – модуль реализации метода ветвей и границ;
- Alg13_1.cpp – модуль реализации алгоритма координатного подъема;
- Alg13_2.cpp – модуль реализации градиентного алгоритма с форой;
- SerdyukovAlg.cpp – модуль реализации алгоритма Сердюкова;
- SerdyukovAlgMod.cpp – модуль реализации улучшенного алгоритма Сердюкова;
- AlgAnalysis.cpp – модуль формы программы для анализа алгоритмов;
- MAX_TSP.cpp – модуль основной формы программы;

Схема алгоритма программы для формы «MAX TSP» приведена на рис. 16.

Рис. 16. Схема алгоритма программы для формы «MAX TSP»

Схема алгоритма программы для формы «Algorithms» приведена на рис. 17.

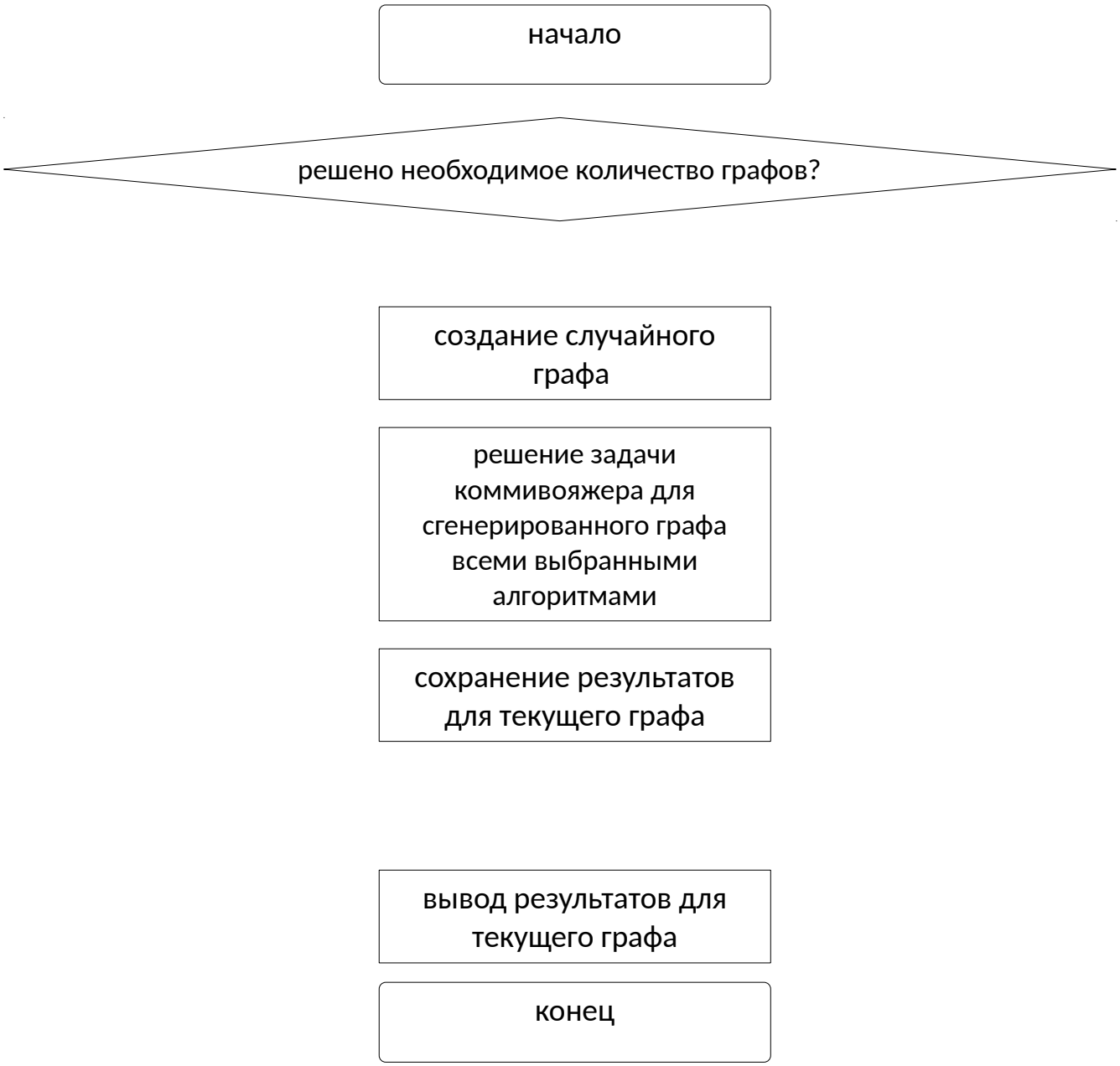

Рис. 17. Схема алгоритма программы для формы «Algorithms»

П1.3. Вызов и загрузка

Загрузка программы «MAX TSP» осуществляется стандартным для операционной системы Microsoft Windows способом, например, двойным щелчком по пиктограмме программы.

П1.4. Входные и выходные данные

Входными данными для программы «MAX TSP» в форме, где выполняются алгоритмы, являются графы, и они могут задаваться как в самом окне программы, так и в загружаемых файлах. Выходными данными являются либо сохраняемые пользователем графы, либо выводимые на экран результаты работы алгоритмов, данные результаты также можно сохранить в файл.

Во входных и выходных файлах содержится информация о размерности графа и его весовой функции. Файлы являются текстовыми. Весовая функция задается матрицей весов. В этом случае в файл записывается матрица, в которой на пересечении *i*-ой строки и *j*-го столбца стоит вес ребра, соединяющего вершины *i* и *j*. Такие файлы имеют расширение *.graph и могут как загружаться, так и сохраняться программой. Пример файла такого формата приведен ниже:

```
5 // Размерность графа
  0.00 82.00 54.00 70.00 84.00 // Матрица весов
82.00   0.00 68.00 24.00   7.00 // ...
54.00 68.00   0.00   6.00   9.00 // ...
70.00 24.00   6.00   0.00 50.00 // ...
84.00   7.00   9.00 50.00   0.00 // ...
```

Выходные данные, выводимые на экран, представляют собой название алгоритма, вес максимального гамильтонового цикла, время работы алгоритма. Эти данный могут быть сохранены в текстовый файл. Входными данными для программы «MAX TSP» в форме анализа алгоритмов «Algorithms» являются установки, выбираемые в ее главном окне. Установки следующие – анализируемые алгоритмы, вид графика. Выходными данными являются построенные графики.

П1.5. Системные требования

Для обеспечения функционирования программы рекомендуются следующие технические средства:

- процессор Windows 2000/XP;
- тактовая частота процессора не ниже 800МГц, рекомендуемая 2,4ГГц;
- объем оперативной памяти не меньше 256 Мб;
- мышь.

ПРИЛОЖЕНИЕ 2. Руководство пользователя

П2.1. Назначение программы

При выполнении дипломной работы была разработана программа «MAX TSP». Она предназначена для решения задачи коммивояжера на максимум. В программе осуществляется поиск оптимальных и приближенных к оптимальным гамильтоновых циклов с помощью различных алгоритмов в симметрических графах и анализ и сравнение самих алгоритмов на случайных графах.

П2.2. Условия выполнения программы

Время выполнения работы алгоритмов будет различаться на вычислительных машинах разных конфигураций. Для обеспечения функционирования программы рекомендуются следующие технические средства:

- процессор Windows 2000/XP;
- тактовая частота процессора не ниже 800МГц, рекомендуемая 2,4ГГц;
- объем оперативной памяти не меньше 256 Мб;
- мышь.

П2.3. Структура и установка программы

Программный продукт состоит из одного файла Project1.exe. Для установки программы необходимо его скопировать в рабочую папку.

П2.4. Выполнение программы

П2.4.1. Запуск программы

Запуск программы «MAX TSP» осуществляется стандартным для операционной системы Microsoft Windows способом, например, двойным щелчком по пиктограмме программы.

П2.4.2. Выполнение программы

При запуске программы появляется окно Windows формы «MAX TSP». Оно поделено на две части: слева задаются параметры и выводится результат, справа представлена матрица весов графа (рис. 18). Создать граф можно либо введение новой размерности, либо кнопкой «Создать симметрический граф».

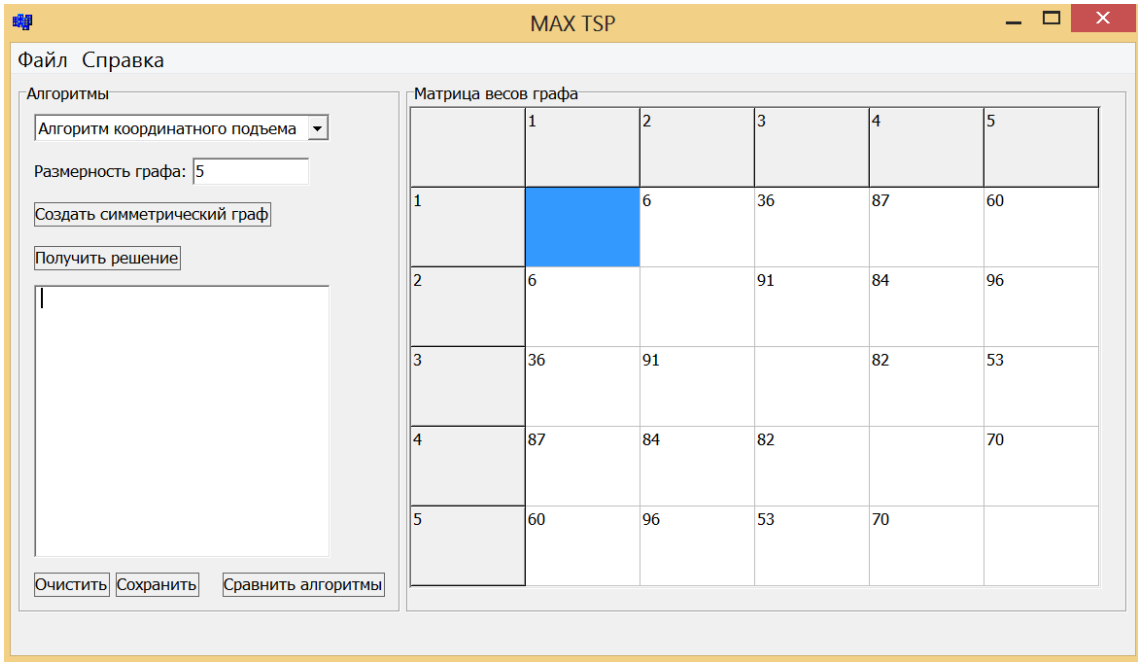

Рис. 18. Окно программы

Результат алгоритма выводится слева в пустом поле, результаты из поля можно убрать кнопкой «Очистить», либо сохранить в файл кнопкой «Сохранить» (рис. 19).

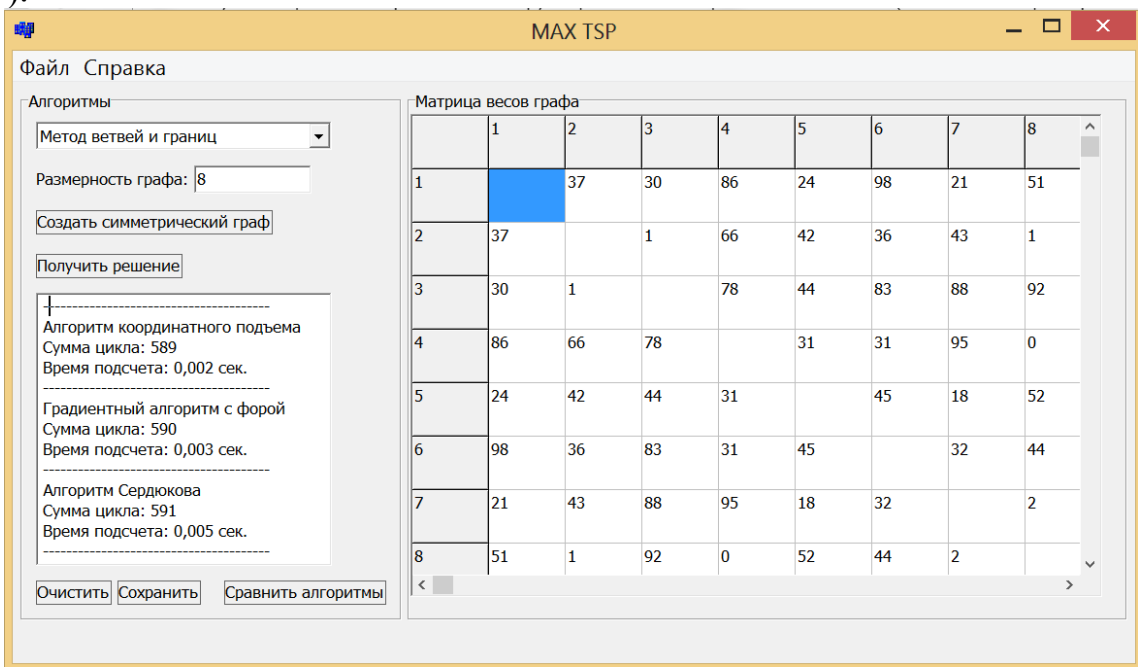

Рис. 19. Вывод результатов

Программа реализует приближенные алгоритмы: алгоритм координатного подъема, градиентный алгоритм с форой, алгоритм Сердюкова, улучшенный алгоритм Сердюкова, и точный метод ветвей и границ. Алгоритм, с помощью которого требуется найти решение задачи коммивояжера на максимум, можно

выбрать в выпадающем списке (рис. 20). Для получения самого решения требуется нажать кнопку «Получить решение».

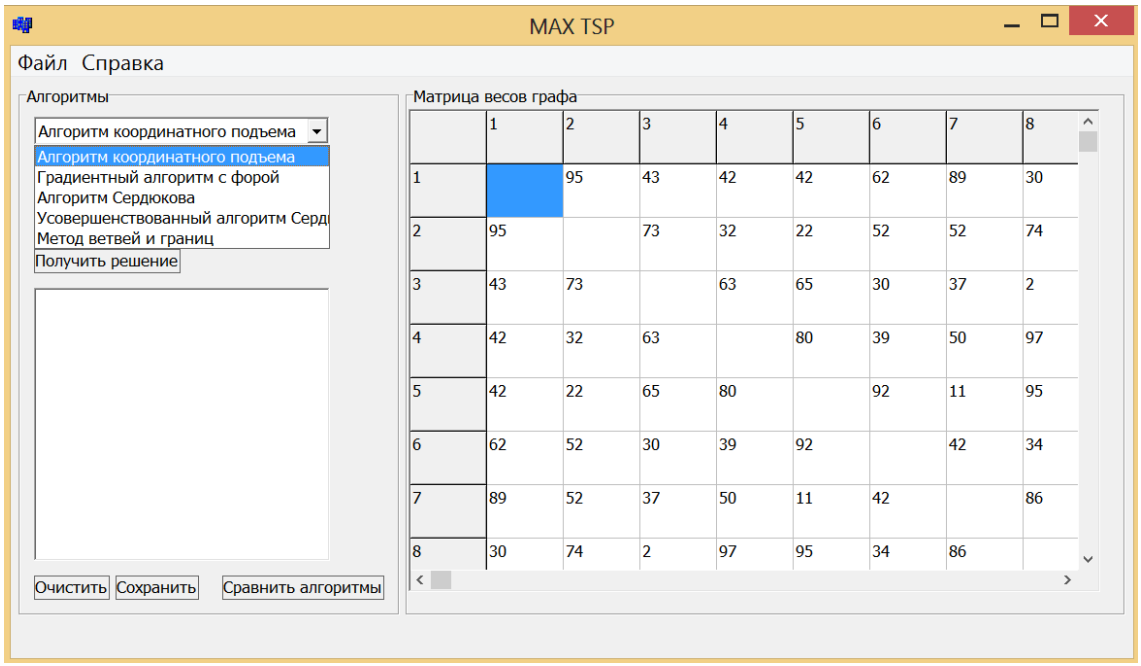

Рис. 20. Выбор алгоритма

Граф можно сохранять и загружать (рис. 21).

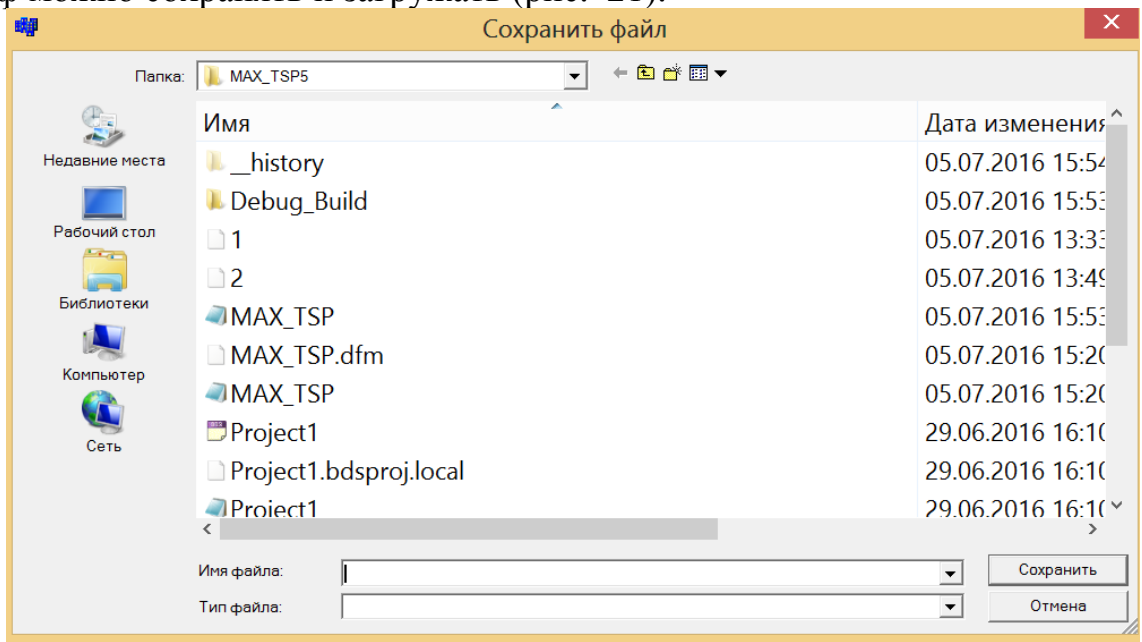

Рис. 21. Сохранение графа

В меню «Справка» можно посмотреть информацию о программе (рис. 22) и помощь (рис. 23).

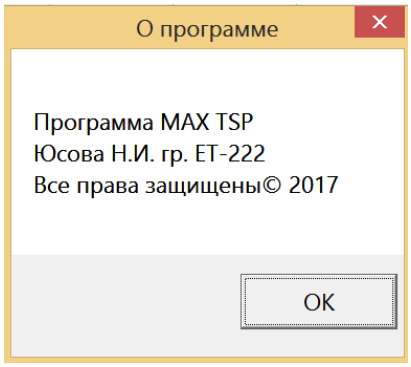

Рис. 22. Окно «О программе»

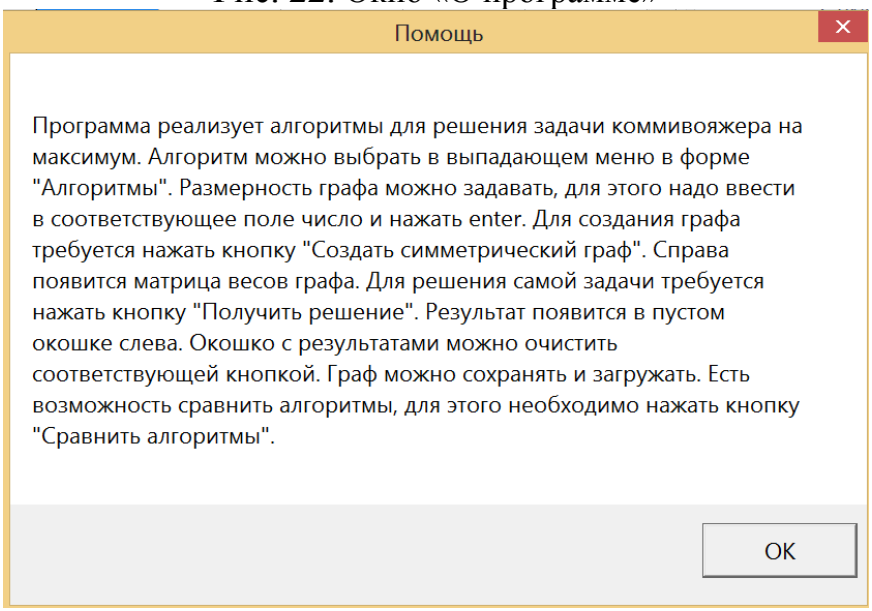

Рис. 23. Окно программы «Помощь»

Для сравнения алгоритмов требуется нажать кнопку «Сравнить алгоритмы». Появится вторая форма программы. Здесь требуется выбрать алгоритмы, которые необходимо сравнить, и график, который нужно построить. Затем следует нажать кнопку «Пуск». Справа появится график, внизу обозначения (рис. 24). Все настройки можно сбросить кнопкой «Очистить».

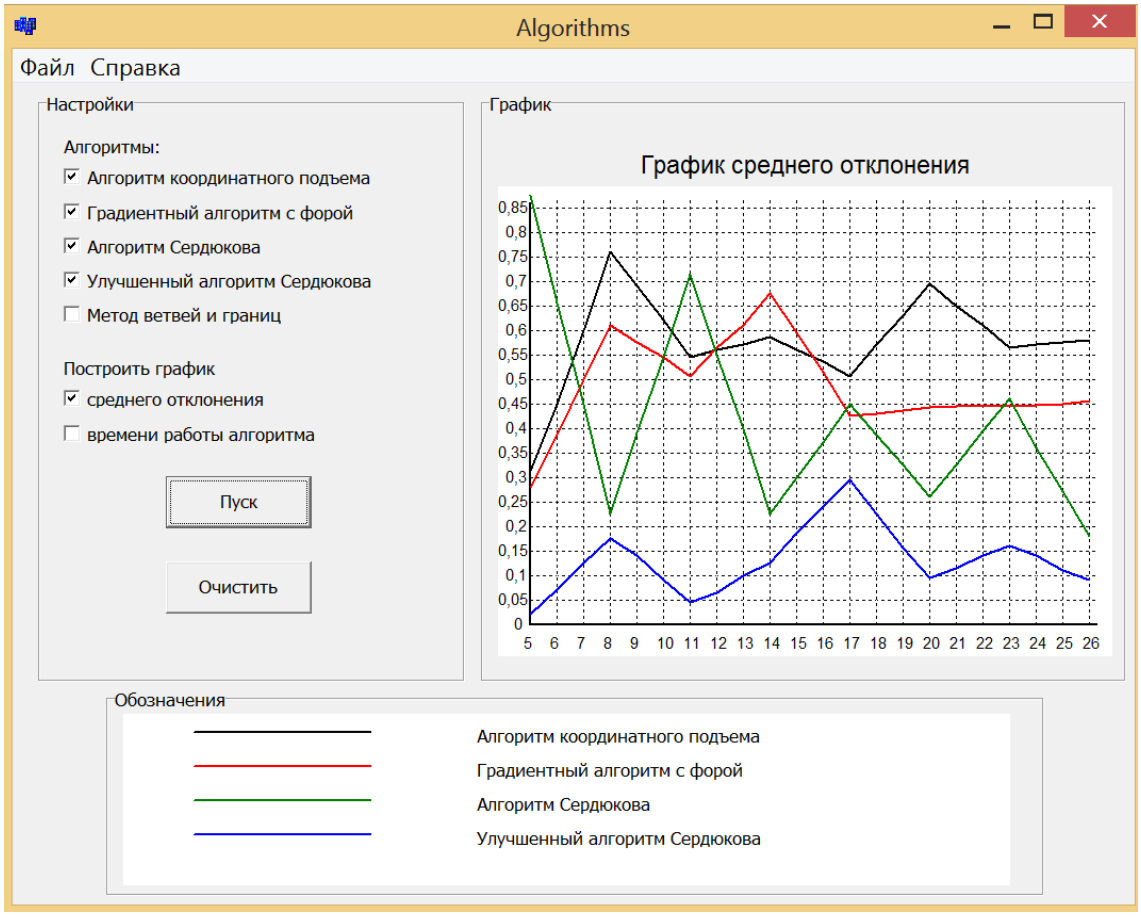

Рис. 24. График точности двух алгоритмов

В данной форме также можно вызвать помощь (рис. 25).

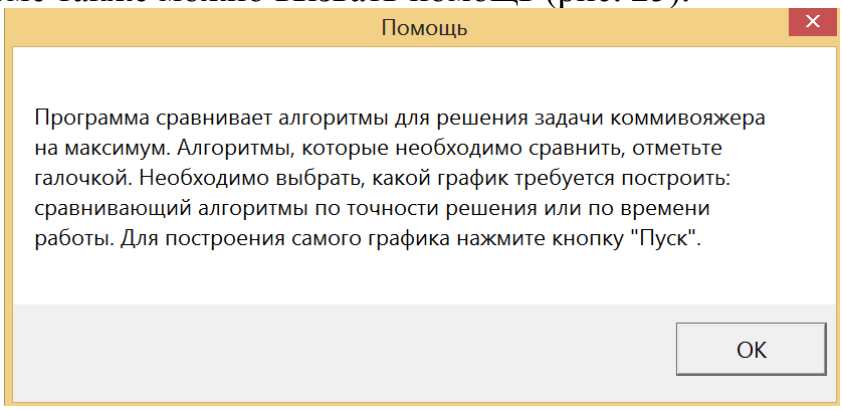

Рис. 25. Помощь пользователю

П2.5. Сообщения пользователю

Если загружаемый граф не симметричен или имеет петли, то пользователь получит соответствующее сообщение (рис. 26).

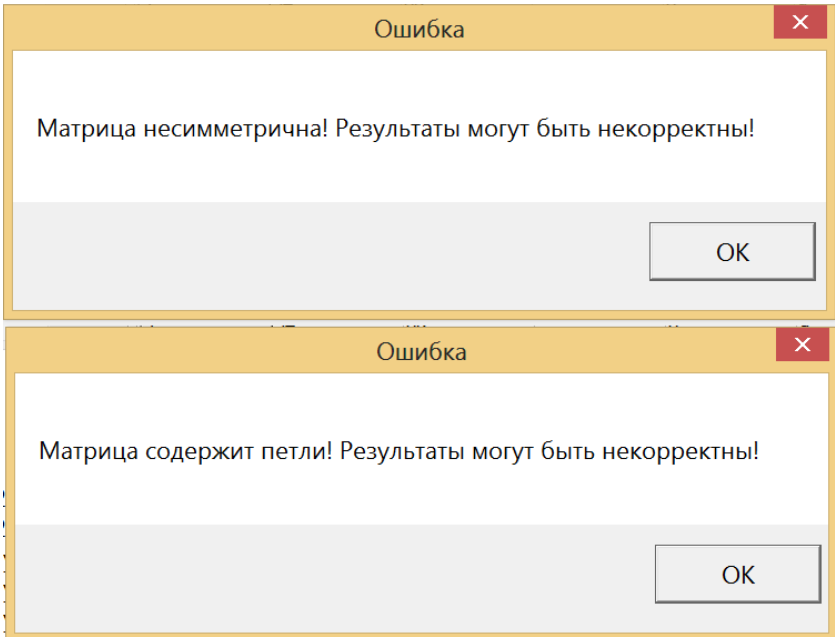

Рис. 26. Сообщение об ошибке при загрузке графа

Если при сравнительном анализе алгоритмов не выбран алгоритм или график, или же отмечены два графика, то пользователь увидит следующие сообщения (рис. 27).

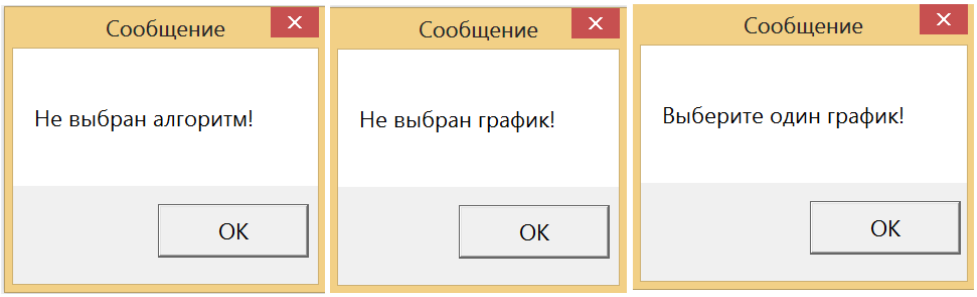

Рис. 27. Сообщения пользователю при анализе алгоритмов

ПРИЛОЖЕНИЕ 3. Текст программы

П3.1. Файл TGraph.h

```
#ifndef TGraphH
#define TGraphH
#include "TEdge.h"
class TGraph
{
protected:
   int       Dim;         // размерность графа
   double   **Weights;     // веса ребер
   bool       Metric;
   void CheckMetric(TEdge E=__EmptyEdge);
public:
   TGraph(void);
   TGraph(int, double**);
   TGraph(const TGraph&);
   ~TGraph();
   int size() {return Dim;}
   TGraph& operator =(TGraph& G);
   bool GetMetric() {return Metric;}
   double GetWeight(int,int) const;
   void SetWeight(int,int,double);
   double GetWeight(TEdge) const;
   void SetWeight(TEdge, double);
};
#endif
```

П3.2. Файл "TGraph.cpp".

```
#pragma hdrstop
#include "TGraph.h"
#include "TEdge.h"
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
// конструктор графа
TGraph::TGraph(void)
{
   Dim=0;
   Weights=NULL;
   Metric=false;
}
// конструктор графа
TGraph::TGraph(int Dim_, double** Weights_)
{
   Dim=Dim_;
   Weights=new double* [Dim];
   for (int i=0;i<Dim;i++)
     Weights[i]=new double [Dim];
   for (int i=0;i<Dim;i++)
     for (int j=0;j<Dim;j++)
          Weights[i][j]=Weights_[i][j];
   CheckMetric();
}
// конструктор копий графа
TGraph::TGraph(const TGraph &G)
{
   Dim=G.Dim;
```

```
   Weights=new double* [Dim];

   for (int i=0;i<Dim;i++)
     Weights[i]=new double [Dim];
    for (int i=0;i<Dim;i++)
     for (int j=0;j<Dim;j++)
          Weights[i][j]=G.Weights[i][j];
   Metric=G.Metric;
}
// деструктор графа
TGraph::~TGraph()
{
   for (int i=0;i<Dim;i++)
     delete[] Weights[i];
   delete[] Weights;
}
// возвращает вес ребра по инцидентным ему вершинам
double TGraph::GetWeight(int From, int To) const
{
   if ((From>=Dim)||(To>=Dim))
      return -1;
   return Weights[From][To];
}
// устанавливает вес ребра по инцидентным ему вершинам
void TGraph::SetWeight(int From, int To, double Value)
{
   if ((From>=Dim)||(To>=Dim))
      return;
   Weights[From][To]=Value;
   Weights[To][From]=Value;
   TEdge E(From, To);
   CheckMetric(E);
}
// возвращает вес ребра
double TGraph::GetWeight(TEdge E) const
{
   int From=E.From;
   int To=E.To;
   if ((From<0)||(To<0))
     return -1;
   if (((int)From>=Dim)||((int)To>=Dim))
     return -1;
   return Weights[From][To];
}
// устанавливает вес ребра
void TGraph::SetWeight(TEdge E, double Value)
{
   int From=E.From;
   int To=E.To;
   if ((From<0)||(To<0))
      return;
   if (((int)From>=Dim)||((int)To>=Dim))
      return;
   Weights[From][To]=Value;
   Weights[To][From]=Value;
   CheckMetric(E);
}
// оператор присваивания
TGraph& TGraph::operator =(TGraph& G)
{
   if (this==&G)
```

```
    return *this;

  if (Dim!=G.Dim)
  {
    if (Dim!=0)
    {
         for (int i=0;i<Dim;i++)
              delete[] Weights[i];
         delete[] Weights;
    }
    Dim=G.Dim;
    Weights=new double* [Dim];
    for (int i=0;i<Dim;i++)
         Weights[i]=new double [Dim];
  }
  for (int i=0;i<Dim;i++)
    for (int j=0;j<Dim;j++)
         Weights[i][j]=G.Weights[i][j];
  return *this;
}
void TGraph::CheckMetric(TEdge E)
{
  int i,j,k;
  if ((Metric==true)&&(E!=__EmptyEdge))
  {
    i=E.From;
    j=E.To;
    for (k=0;k<Dim;k++)
      {
         if (k==i) continue;
         if (k==j) continue;
        if (pow(Weights[i][j],2)+pow(Weights[i][k],2)<pow(Weights[j][k],2))
        {
             Metric=false;
           return;
        }
        if (pow(Weights[j][i],2)+pow(Weights[j][k],2)<pow(Weights[i][k],2))
        {
             Metric=false;
           return;
        }
        if (pow(Weights[k][i],2)+pow(Weights[k][j],2)<pow(Weights[i][j], 2))
        {
             Metric=false;
           return;
        }
      }
  }
  else
  {
    Metric=true;
    for (i=0;i<Dim;i++)
         for (j=i+1;j<Dim;j++)
              for (k=j+1;k<Dim;k++)
           {
                if (Weights[i][j]+Weights[i][k]<Weights[j][k])
               {
                     Metric=false;
                 return;
               }
                if (Weights[j][i]+Weights[j][k]<Weights[i][k])
```

```
                {
                        Metric=false;
                   return;
                }
                  if (Weights[k][i]+Weights[k][j]<Weights[i][j])
                {
                        Metric=false;
                   return;
                }
             }
   }
}
#pragma package(smart_init)
```

П3.3. Файл " TGraphMarks.h".

```
#ifndef TGraphMarksH
#define TGraphMarksH
#include "TEdge.h"
class TGraphMarks
{
protected:
   int   Dim;                 // размерность графа
   bool **Marks;               // отмеченые ребра - частичный тур
   int   MarksCount;          // текущая длина часичного тура
   int  *VertexMarkedDegrees; // число отмеченных инцидентных ребер
public:
   TGraphMarks(void);
   TGraphMarks(int);
   TGraphMarks(const TGraphMarks&);
   ~TGraphMarks();
   TGraphMarks& TGraphMarks::operator =(TGraphMarks& M);
   int GetMarksCount() {return MarksCount;}
   bool GetMarked(int, int) const;
   void SetMark(int, int, bool);
   void SetMarked(int, int);
   void SetUnmarked(int, int);
   bool GetMarked(TEdge) const;
   void SetMark(TEdge, bool);
   void SetMarked(TEdge);
   void SetUnmarked(TEdge);
   int MarkedDegree(int) const;
   void ClearMarks(void);
   void CopyMarks(const TGraphMarks &);
};
#endif
```

П3.4. Файл " TGraphMarks.cpp".

```
#pragma hdrstop
#include "TGraphMarks.h"
#include <stdio.h>
#include <stdlib.h>
// конструктор класса
TGraphMarks::TGraphMarks(void)
{
   Dim=0;
   MarksCount=0;
   Marks=NULL;
   VertexMarkedDegrees=NULL;
}
```

```
// конструктор класса

TGraphMarks::TGraphMarks(int Dim_)
{
   Dim=Dim_;
   MarksCount=0;
   Marks=new bool* [Dim];
   VertexMarkedDegrees=new int [Dim];
   for (int i=0;i<Dim;i++)
     Marks[i]=new bool [Dim];
   for (int i=0;i<Dim;i++)
   {
     VertexMarkedDegrees[i]=0;
     for (int j=0;j<Dim;j++)
          Marks[i][j]=false;
   }
}
//---------------------------------------------------------------------------
// конструктор копий класса
TGraphMarks::TGraphMarks(const TGraphMarks &M)
{
   Dim=M.Dim;
   MarksCount=M.MarksCount;
   Marks=new bool* [Dim];
   VertexMarkedDegrees=new int [Dim];
   for (int i=0;i<Dim;i++)
     Marks[i]=new bool [Dim];
   for (int i=0;i<Dim;i++)
   {
     VertexMarkedDegrees[i]=M.VertexMarkedDegrees[i];
     for (int j=0;j<Dim;j++)
          Marks[i][j]=M.Marks[i][j];
   }
}
// деструктор графа
TGraphMarks::~TGraphMarks()
{
   for (int i=0;i<Dim;i++)
     delete[] Marks[i];
   delete[] Marks;
   delete[] VertexMarkedDegrees;
}
// возвращает наличие или отсутствие пометки по вершинам
bool TGraphMarks::GetMarked(int From, int To) const
{
   if ((From>=Dim)||(To>=Dim))
      return false;
   return Marks[From][To];
}
// утанавливает наличие или отсутствие пометки по вершинам
void TGraphMarks::SetMark(int From, int To, bool Mark)
{
   if (From==To)
     return;
   if ((From>=Dim)||(To>=Dim))
     return;
   if (Marks[From][To]==Mark)
     return;
   Marks[From][To]=Mark;
   Marks[To][From]=Mark;
   if  (Mark==false)
```

```
   {
     VertexMarkedDegrees[From]--;
      VertexMarkedDegrees[To]--;
      MarksCount--;
   }
   if  (Mark==true)
   {
      VertexMarkedDegrees[From]++;
      VertexMarkedDegrees[To]++;
      MarksCount++;
   }
}
//наличие пометки по вершинам
void TGraphMarks::SetMarked(int From, int To)
{
   if (From==To)
     return;
   if ((From>=Dim)||(To>=Dim))
     return;
   if (Marks[From][To]==false)
   {
     Marks[From][To]=true;
     Marks[To][From]=true;
     VertexMarkedDegrees[From]++;
     VertexMarkedDegrees[To]++;
     MarksCount++;
   }
}
//отсутствие пометки по вершинам
void TGraphMarks::SetUnmarked(int From, int To)
{
   if (From==To)
      return;
   if ((From>=Dim)||(To>=Dim))
      return;
   if (Marks[From][To]==true)
   {
     Marks[From][To]=false;
     Marks[To][From]=false;
      VertexMarkedDegrees[From]--;
      VertexMarkedDegrees[To]--;
      MarksCount--;
   }
}
// возвращает наличие или отсутствие пометки по ребру
bool TGraphMarks::GetMarked(TEdge E) const
{
   int From=E.From;
   int To=E.To;
   if ((From>=Dim)||(To>=Dim))
      return false;
   return Marks[From][To];
}
// утанавливает наличие или отсутствие пометки по ребру
void TGraphMarks::SetMark(TEdge E, bool Mark)
{
   int From=E.From;
   int To=E.To;
   if (From==To)
      return;
```

```
   if ((From>=Dim)||(To>=Dim))
      return;
   if (Marks[From][To]==Mark)
      return;
   Marks[From][To]=Mark;
   Marks[To][From]=Mark;
   if  (Mark==false)
   {
      VertexMarkedDegrees[From]--;
      VertexMarkedDegrees[To]--;
      MarksCount--;
   }
   if  (Mark==true)
   {
      VertexMarkedDegrees[From]++;
      VertexMarkedDegrees[To]++;
      MarksCount++;
   }
}
//наличие пометки по ребру
void TGraphMarks::SetMarked(TEdge E)
{
   int From=E.From;
   int To=E.To;
   if (From==To)
     return;
   if ((From>=Dim)||(To>=Dim))
     return;
   if (Marks[From][To]==false)
   {
     Marks[From][To]=true;
     Marks[To][From]=true;
      VertexMarkedDegrees[From]++;
      VertexMarkedDegrees[To]++;
      MarksCount++;
   }
}
//отсутствие пометки по ребру
void TGraphMarks::SetUnmarked(TEdge E)
{
   int From=E.From;
   int To=E.To;
   if (From==To)
     return;
   if ((From>=Dim)||(To>=Dim))
     return;
   if (Marks[From][To]==true)
   {
     Marks[From][To]=false;
     Marks[To][From]=false;
      VertexMarkedDegrees[From]--;
      VertexMarkedDegrees[To]--;
      MarksCount--;
   }
}
//---------------------------------------------------------------------------
//возвращает число отмеченных инцидентных ребер по указан. вершине
int TGraphMarks::MarkedDegree(int Vertex) const
{
   if (Vertex>=Dim)
```

```
     return -1;
   return VertexMarkedDegrees[Vertex];
}

//---------------------------------------------------------------------------
// оператор присваивания
TGraphMarks& TGraphMarks::operator =(TGraphMarks& M)
{
   if (this == &M)
      return *this;

   if (Dim != M.Dim)
   {
      if (Dim != 0)
      {
          for (int i = 0; i < Dim;  i++)
             delete[] Marks[i];
         delete[] Marks;
         delete[] VertexMarkedDegrees;
      }
      Dim = M.Dim;
      Marks   = new bool* [Dim];
     VertexMarkedDegrees = new int [Dim];
     for (int i = 0; i < Dim; i++)
      {
         Marks[i]   = new bool [Dim];
      }
   }
   MarksCount = M.MarksCount;
   for (int i = 0; i < Dim; i++)
   {
      VertexMarkedDegrees[i] = M.VertexMarkedDegrees[i];
     for (int j = 0; j < Dim; j++)
         Marks[i][j]     = M.Marks[i][j];
   }
   return *this;
}
//все ребра не отмечены
void TGraphMarks::ClearMarks(void)
{
   for (int i=0;i<Dim;i++)
   {
     VertexMarkedDegrees[i]=0;
     for (int j=0;j<Dim;j++)
          Marks[i][j]=false;
   }
   MarksCount=0;
}
void TGraphMarks::CopyMarks(const TGraphMarks &M)
{
   if (Dim==M.Dim)
   {
     for (int i=0;i<Dim;i++)
     {
          VertexMarkedDegrees[i]=M.VertexMarkedDegrees[i];
          for (int j=0;j<Dim;j++)
               Marks[i][j]=M.Marks[i][j];
     }
     MarksCount=M.MarksCount;
   }
```

```
}
```



```
#pragma package(smart_init)
```

П3.5. Файл " TMarkedGraph.h".

```
#ifndef TMarkedGraphH
#define TMarkedGraphH
#include "TGraph.h"
#include "TGraphMarks.h"
class TMarkedGraph : public TGraph,
                     public TGraphMarks
{
private:
   void  TMarkedGraph::UnmarkAllPathsThroughVertex(int v);
   int  ReccRouteExisting(int,int,int);
   TPath ReccGetPath(int, int, int);
protected:
   int Dim;
public:
   TMarkedGraph(void);
   TMarkedGraph(int,double**);
   TMarkedGraph(const TMarkedGraph&);
   ~TMarkedGraph();
   TMarkedGraph& operator =(TMarkedGraph&);
   TMarkedGraph& operator *=(double);
   void CreateRandomSymmetricGraph(int,int);
   void ReverseWeights(void);
   double GetMarkedWeight();
   bool CheckEdgeForTourInclusion(int, int);
   bool CheckEdgeForTourInclusion(TEdge);
   TEdge GreedyEdgeFinding();
   int  RouteExisting(int,int);
   TPath GetPath(int, int);
   TPath GetCycle(int);
   TGraphCycles GetGraphCycles();
   double MarkMaxMatching(bool Min = false);
   double MarkMax2Matching(bool Min = false);
   double MarkMax2Factor(bool Min = false);
   double MarkMaxHCycleBABAlg(bool Min = false);
   double MarkMaxHCycleAlg13_1(bool Min = false);
   double MarkMaxHCycleAlg13_2(bool Min = false);
   double MarkMaxHCycleSerdyukovAlg(bool Min = false);
   double MarkMaxHCycleSerdyukovAlgMod(double Eps = 0.2, bool Min = false);
};
#endif
```

П3.6. Файл " TMarkedGraph.cpp".

```
#pragma hdrstop
#include "TMarkedGraph.h"
#include <windows.h>
#include <Math.hpp>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "TEdge.h"
//nclude "MatchingAlgs.h"
#include "Alg13_1.h"                // алгоритм 13.1
#include "Alg13_2.h"                // алгоритм 13.2
#include "SerdyukovAlg.h"           // алгоритм Сердюкова
```

```
#include "SerdyukovAlgMod.h"      // усовершенствованный алгоритм Сердюкова
#include BaBAlg.h"                 // метод ветвей и границ
//---------------------------------------------------------------------------
// конструктор
TMarkedGraph::TMarkedGraph(void):
                              TGraph(), TGraphMarks()
{
   Dim=0;
}
//---------------------------------------------------------------------------
// конструктор
TMarkedGraph::TMarkedGraph(int Dim_, double** Weights_):
                              TGraph(Dim_, Weights_), TGraphMarks(Dim_)
{
   Dim=Dim_;
}
//---------------------------------------------------------------------------
// конструктор копий
TMarkedGraph::TMarkedGraph(const TMarkedGraph &G):
                             TGraph(G), TGraphMarks(G)
{
   Dim=G.Dim;
}

//---------------------------------------------------------------------------
// деструктор
TMarkedGraph::~TMarkedGraph()
{
}
double TMarkedGraph::GetMarkedWeight()
{
   double Summ=0;
   for (int i=0;i<Dim;i++)
     for (int j=i+1;j<Dim;j++)
          if (Marks[i][j]==true)
               Summ+=Weights[i][j];
   return Summ;
}

//---------------------------------------------------------------------------
// оператор присваивания
TMarkedGraph& TMarkedGraph::operator =(TMarkedGraph& G)
{
   if (this==&G)
      return *this;
   if (Dim!=G.Dim)
   {
     if (Dim!=0)
       {
          for (int i=0;i<Dim;i++)
            delete[] Weights[i];
         delete[] Weights;

          for (int i=0;i<Dim;i++)
            delete[] Marks[i];
         delete[] Marks;
         delete[] VertexMarkedDegrees;
       }
     Dim=G.Dim;
     Weights=new double* [Dim];
```

```
     for (int i=0;i<Dim;i++)
          Weights[i]=new double [Dim];
     Marks=new bool* [Dim];
     VertexMarkedDegrees=new int [Dim];
     for (int i=0;i<Dim;i++)
     {
          Marks[i]=new bool [Dim];
       }
   }
   MarksCount = G.MarksCount;
   for (int i=0;i<Dim;i++)
   {
     VertexMarkedDegrees[i]=G.VertexMarkedDegrees[i];
     for (int j=0; j<Dim; j++)
       {
          Marks[i][j]= G.Marks[i][j];
          Weights[i][j]= G.Weights[i][j];
       }
   }
   return *this;
}
//---------------------------------------------------------------------------
void TMarkedGraph::CreateRandomSymmetricGraph(int N, int MaxWeightDigit)
{
   int i,j;
   if (Dim!=N)
   {
     if (Dim!=0)
       {
          for (int i=0;i<Dim;i++)
            delete[] Weights[i];
          delete[] Weights;

          for (int i=0;i<Dim;i++)
            delete[] Marks[i];
          delete[] Marks;
          delete[] VertexMarkedDegrees;
       }
     Dim=N;
     Weights=new double* [Dim];
     for (int i=0;i<Dim;i++)
          Weights[i] = new double [Dim];

     Marks=new bool* [Dim];
     VertexMarkedDegrees=new int [Dim];
     for (int i=0;i<Dim;i++)
       {
          Marks[i]=new bool [Dim];
       }
   }
   for (i=0;i<Dim;i++)
   {
      VertexMarkedDegrees[i] = 0;
     for (j=i;j<Dim;j++)
       {
          if (i==j)
          {
               Weights[i][j]=0;
            continue;
          }
```

```
            Marks[i][j]=false;

            Marks[j][i]=false;
            //for (int k=0;k<MaxWeightDigit;k++) {
                 //int Stepen=Stepen*10;
            //}
            Weights[i][j]=RandomRange(0,pow(10,MaxWeightDigit)-1);
            Weights[j][i]=Weights[i][j];
         }
      }
      MarksCount=0;
      TGraph::Dim=N;
      TGraphMarks::Dim=N;
      CheckMetric();
   }
   //---------------------------------------------------------------------------
   // можно ли присоеденить это ребро к текущему частичному туру
   bool TMarkedGraph::CheckEdgeForTourInclusion(int From,int To)
   {
      if (VertexMarkedDegrees[From]>1) return false;
      if (VertexMarkedDegrees[To]>1) return false;

      if (VertexMarkedDegrees[From]<1) return true;
      if (VertexMarkedDegrees[To]<1) return true;
      if (To==From) return false;
      if (Marks[To][From]==true) return false;
      if ((MarksCount!=Dim - 1)&&(RouteExisting(To,From)!=-1))
         return false;
      return true;
   }

   bool TMarkedGraph::CheckEdgeForTourInclusion(TEdge E)
   {
      return CheckEdgeForTourInclusion(E.From,E.To);
   }
   //---------------------------------------------------------------------------
   // ищет ребро максимального веса, которое можно присоеденить
   // к текущему частичному туру
   TEdge TMarkedGraph::GreedyEdgeFinding()
   {
      TEdge ResEdge;
      long MaxWeight=-1;

      for (int i=0;i<Dim;i++)
      {
        if (VertexMarkedDegrees[i]>1) continue;
        for (int j=i+1;j<Dim;j++)
          {
              if (VertexMarkedDegrees[j]>1) continue;
              if (Marks[i][j]==true)        continue;
                                                                    if
(((int)GetWeight(i,j)>MaxWeight)&&(CheckEdgeForTourInclusion(i,j)==true))
              {
                   ResEdge.From=i;
                   ResEdge.To=j;
                   MaxWeight=GetWeight(i,j);
              }
          }
      }
      return ResEdge;
   }
```

```
//---------------------------------------------------------------------------
// существует ли маршрут по отмеченным ребрам
// возвращает количество ребер в пути
int TMarkedGraph::ReccRouteExisting(int Start, int End, int LastVertex=-1)
{
   int Length;

   if (Start == End)
      return 0;

   for (int i = 0; i < Dim;  i++)
   {
      if (i == Start)      continue;
      if (i == LastVertex) continue;
      if (Marks[Start][i] == true)
      {
         Length = ReccRouteExisting(i, End, Start);
         if (Length >= 0)
            return Length + 1;
      }
   }
   return -1;
}
//---------------------------------------------------------------------------
int TMarkedGraph::RouteExisting(int Start, int End)
{
   return ReccRouteExisting(Start,End,-1);
}
TPath TMarkedGraph::ReccGetPath(int Start, int End, int LastVertex = -1)
{
   TPath Path;
   if (Start == End)
   {
      Path.clear();
      Path.push_front(Start);
      return Path;
   }
   for (int i = 0; i < Dim;  i++)
   {
      if (i == Start)      continue;
      if (i == LastVertex) continue;
      if (Marks[Start][i] == true)
      {
         Path = ReccGetPath(i, End, Start);
         if (Path.size() > 0)
         {
            Path.push_front(Start);
            return Path;
         }
      }
   }
   Path.clear();
   return Path;
}

TPath TMarkedGraph::GetPath(int Start, int End)
{
   return ReccGetPath(Start, End, -1);
}
```

```
TPath TMarkedGraph::GetCycle(int Vertex)
{
   TPath Path;
   int VertexNeighbour;

   if (VertexMarkedDegrees[Vertex] < 2)
   {
      Path.clear();
      return Path;
   }

   if (VertexMarkedDegrees[Vertex] > 2)
   {
      Path.clear();
      return Path;
   }

   // найдем соседа
   for (int i = 0; i < Dim;  i++)
   {
      if (i == Vertex)       continue;
      if (Marks[Vertex][i] == true)
      {
         VertexNeighbour = i;
         break;

      }
   }

   SetUnmarked(Vertex, VertexNeighbour);
   Path = GetPath(Vertex, VertexNeighbour);
   SetMarked(Vertex, VertexNeighbour);
   if (Path.size() > 0)
   {
      Path.push_back(Vertex);
      return Path;
   }
   else
   {
      Path.clear();
      return Path;
   }
}
//---------------------------------------------------------------------------
void TMarkedGraph::UnmarkAllPathsThroughVertex(int v)
{
   int i;
   for (i = 0; i < Dim; i++)
   {
      if (GetMarked(v, i) == true)
      {
         SetUnmarked(v, i);
         UnmarkAllPathsThroughVertex(i);
      }
   }
}
//---------------------------------------------------------------------------
TMarkedGraph& TMarkedGraph::operator *=(double n)
{
   int i, j;
```

```
   TMarkedGraph Temp(*this);

   for (i=0; i < Dim; i++)
      for (j = 0; j < Dim; j++)
         Weights[i][j] *= n;
   return *this;
}
TGraphCycles TMarkedGraph::GetGraphCycles()
{
   TGraphCycles GraphCycles;
   TPath        Path;

   for (int i = 0; i < Dim;  i++)
      if (VertexMarkedDegrees[i] > 2)
      {
         GraphCycles.clear();
         return GraphCycles;
      }

   TMarkedGraph GraphCopy(*this);
   for (int i = 0; i < Dim;  i++)
   {
      if (GraphCopy.VertexMarkedDegrees[i] != 0)
      {
         Path = GraphCopy.GetCycle(i);
         int PathSize = Path.size();
         if (PathSize == 0)
         {
            GraphCopy.UnmarkAllPathsThroughVertex(i);
            continue;
         }
         for (int j = 0; j < PathSize - 1;  j++)
            GraphCopy.SetUnmarked(Path.at(j), Path.at(j+1));
         if (PathSize > 0)
            GraphCycles.push_back(Path);
      }
   }
   return GraphCycles;
}
//---------------------------------------------------------------------------
// изменяет веса для решения задачи на минимум
void TMarkedGraph::ReverseWeights(void)
{
   int i, j;
   double MaxWeight;

   for (i = 0; i < Dim; i++)
      for (j = i+1; j < Dim; j++)
         if (Weights[i][j] > MaxWeight)
            MaxWeight = Weights[i][j];
   for (i = 0; i < Dim; i++)
      for (j = 0; j < Dim; j++)
      {
         if (i == j)
            Weights[i][j] = 0;
         else
            Weights[i][j] = MaxWeight + 1 - Weights[i][j];
      }
}
//---------------------------------------------------------------------------
// находит максимальное паросочетание
```

```
double TMarkedGraph::MarkMaxMatching(bool Min)
{
   if (Min == true)
   {
      ReverseWeights();
      *this = FindMaxMatching(*this);
      ReverseWeights();
   }
   else
      *this = FindMaxMatching(*this);

   return GetMarkedWeight();
}

//---------------------------------------------------------------------------
// находит максимальное 2-паросочетание
double TMarkedGraph::MarkMax2Matching(bool Min)
{
   if (Min == true)
   {
      ReverseWeights();
      *this = FindMaxTwoMatching(*this);
      ReverseWeights();
   }
   else
      *this = FindMaxTwoMatching(*this);

   return GetMarkedWeight();
}

//---------------------------------------------------------------------------
// находит максимальный 2-фактор
double TMarkedGraph::MarkMax2Factor(bool Min)
{
   if (Min == true)
   {
      ReverseWeights();
      *this = FindMaxTwoFactor(*this);
      ReverseWeights();
   }
   else
      *this = FindMaxTwoFactor(*this);

   return GetMarkedWeight();
}
//---------------------------------------------------------------------------
// находит гамильтонов цикл с помощью
// эпсилон-приближенного алгоритма 13.1
//алгоритм координатного подъема
double TMarkedGraph::MarkMaxHCycleAlg13_1(bool Min)
{
   if (Min == true)
   {
      ReverseWeights();
     *this = Alg13_1(*this);
      ReverseWeights();
   }
   else
     *this = Alg13_1(*this);
```

```
   return GetMarkedWeight();
}
// находит гамильтонов цикл с помощью
// эпсилон-приближенного алгоритма 13.2
//градиентный алгоритм с форой
double TMarkedGraph::MarkMaxHCycleAlg13_2(bool Min)
{
   if (Min == true)
   {
      ReverseWeights();
      *this = Alg13_2(*this);
      ReverseWeights();
   }
   else
      *this = Alg13_2(*this);

   return GetMarkedWeight();
}
//---------------------------------------------------------------------------
// находит максимальный гамильтонов цикл методом ветвей и границ
double TMarkedGraph::MarkMaxHCycleBABAlg(bool Min)
{
   if (Min == true)
   {
      ReverseWeights();
      *this = BranchAndBoundAlg(*this);
      ReverseWeights();
   }
   else
      *this = BranchAndBoundAlg(*this);

   return GetMarkedWeight();
}
// находит гамильтонов цикл с помощью
// эпсилон-приближенного алгоритма Сердюкова
double TMarkedGraph::MarkMaxHCycleSerdyukovAlg(bool Min)
{
   if (Min == true)
   {
     ReverseWeights();
     *this = SerdyukovAlg(*this);
      ReverseWeights();
   }
   else
      *this = SerdyukovAlg(*this);

   return GetMarkedWeight();
}
// находит гамильтонов цикл с помощью
// эпсилон-приближенного усовершенствованного алгоритма Сердюкова
double TMarkedGraph::MarkMaxHCycleHR1Alg(double Eps, bool Min)
{
   if (Min == true)
   {
      ReverseWeights();
     *this = SerdyukovAlgMod(*this, Eps);
      ReverseWeights();
   }
   else
      *this = HR1Alg(*this, Eps);
```

```
   return GetMarkedWeight();
}



#pragma package(smart_init)
```

П3.7. Файл "TEdge.h".

```
#ifndef TEdgeH
#define TEdgeH
// структура ребро
class TEdge
{
public:
   int    From;
   int    To;
   TEdge(void);
   TEdge(int From_, int To_);
   TEdge(const TEdge&);
   bool   operator ==(TEdge E);
   bool   operator !=(TEdge E);
   TEdge& TEdge::operator =(const TEdge&);

   bool Adjacent(TEdge E);
};
const TEdge __EmptyEdge(-1, -1);
#endif
```

П3.8. Файл "TEdge.cpp".

```
#pragma hdrstop
#include "TEdge.h"
//ребро графа
TEdge::TEdge(void)
{
   From=-1;
   To=-1;
}
TEdge::TEdge(int From_, int To_)
{
   From=From_;
   To=To_;
}
TEdge::TEdge(const TEdge& E)
{
   From=E.From;
   To=E.To;
}
bool TEdge::operator ==(TEdge E)
{
   return ((From==E.From && To==E.To)||(From==E.To && To==E.From));
}
bool TEdge::operator !=(TEdge E)
{
   return !((*this)==E);
}
TEdge& TEdge::operator =(const TEdge& E)
{
   if (this == &E)
```

```
     return *this;
   From=E.From;
   To=E.To;
   return *this;
}
bool TEdge::Adjacent(TEdge E)
{
   if (*this == E)
      return false;
   if ((From==E.From)||
      (From==E.To) ||
      (To==E.From)||
       (To==E.To)    )
      return true;
   return false;
}
#pragma package(smart_init)
```

П3.9. Файл " MatchingAlgs.h".

```
#ifndef MatchingAlgsH
#define MatchingAlgsH
TMarkedGraph& FindMaxMatching(TMarkedGraph& MarkedGraph);
TMarkedGraph& FindMaxTwoMatching(TMarkedGraph& MarkedGraph);
TMarkedGraph& FindMaxTwoFactor(TMarkedGraph& MarkedGraph);
#endif
```

П3.10. Файл " MatchingAlgs.cpp".

```
#include <vcl.h>
#include <windows.h>
#include <stdio.h>
#pragma hdrstop
#include "TMarkedGraph.h"
#include "MatchingAlgs.h"
#include <stdio.h>
#pragma hdrstop
void SETBOUNDS();
void AddEdge(Graph, int, int, wtype);
void Initialize(int);
void FreeUp();
int *Weighted_Match (int gptr, int type, int maximize)
{
   int g, j, w, outcome;
   SetUp(gptr,type);
   Initialize(maximize);
   for(;;)
   {
         DELTA = 0;
         for (v = 1; v <= U; ++v)
         if (MATE[v] == DUMMYEDGE)
               POINTER (DUMMYVERTEX, v, DUMMYEDGE);
      for(;;)
      {
         i = 1;
         for (j = 2; j <= U; ++j)
               if (EDGE_D[i] > EDGE_D[j])
                  i = j;
         DELTA = EDGE_D[i];
         if (DELTA == LAST_D)
               goto done;
```

```
         v = BLOSSOM[i];
         if (LINK[v] >= 0)       // если v - внешняя вершина
         {                        // либо сжимаем цветок, либо делаем увеличение
               PAIR (&outcome);
               if (outcome == 1)
                  break;
               }
         else                    // иначе
         {
               w = BMATE (v);       // находим соседа
               if (LINK[w] < 0)     // если
            {
                  POINTER (v, w, OPPEDGE(EDGE[i]));
            }
               else                 // иначе
                UNPAIR (v, w);
         }
       }
      LAST_D -=DELTA;
         SETBOUNDS();
         g = OPPEDGE(e);
         REMATCH (BEND(e), g);
         REMATCH (BEND(g), e);
   }
done:
   SETBOUNDS();
   UNPAIR_ALL();
   for (i = 1; i <= U; ++i)
   {
      MATE[i] = END[MATE[i]];
         if (MATE[i] == DUMMYVERTEX)
         MATE[i] = 0;
   }
   FreeUp();
   return(MATE);
}
void Initialize(int maximize)
{
   int i, allocsize;
   wtype max_wt= -MAXWT, min_wt=MAXWT;

   DUMMYVERTEX = U+1;
   DUMMYEDGE = U+2*V+1;
   END[DUMMYEDGE] = DUMMYVERTEX;

   for (i=U+2; i<=U+2*V; i+=2)
   {
      if (WEIGHT[i] > max_wt)
         max_wt = WEIGHT[i];
      if (WEIGHT[i] < min_wt)
         min_wt = WEIGHT[i];
   }
   if (!maximize)
   {
      if (U%2!=0)
       {
          exit(0);
       }
      max_wt += 2;
      for (i=U+1; i<=U+2*V; i++)
```

```
         WEIGHT[i] = max_wt-WEIGHT[i];
      max_wt = max_wt-min_wt;
   }
   LAST_D = max_wt/2;
   allocsize   = (U+2);
   MATE        = new int[allocsize];
   LINK        = new int[allocsize];
   BLOSSOM     = new int[allocsize];
   NEXT        = new int[allocsize];
   LAST        = new int[allocsize];
   S           = new wtype[allocsize];
   EDGE_D      = new wtype[allocsize];
   EDGE        = new int[allocsize];
   allocsize   = (U+2*V+2);
   NEXTPAIR    = new int[allocsize];
   for (i = 1; i <= U+1; ++i)
   {
      MATE[i] = DUMMYEDGE;
      EDGE[i] = DUMMYEDGE;
      NEXT[i] = 0;
      LINK[i] = -DUMMYEDGE;
      BLOSSOM[i] = i;
         LAST[i] = i;
         S[i] = LAST_D;
         EDGE_D[i] = LAST_D;
   }
}

void FreeUp()
{
   delete[] LINK;
   delete[] BLOSSOM;
   delete[] NEXT;
   delete[] LAST;
   delete[] S;
   delete[] EDGE_D;
   delete[] EDGE;
   delete[] NEXTPAIR;
   delete[] A;
   delete[] END;
   delete[] WEIGHT;
}

TMarkedGraph& FindMaxMatching(TMarkedGraph &MarkedGraph)
{
   int *Mate;
   Graph graph;
   int edges, degree, vlabel, elabel, adj_node;
   int i, j;
   int xcoord, ycoord;
   int N = MarkedGraph.size();
   graph = NewGraph(N);
   for (i = 1; i <= N; ++i)
   {
         NLabel(graph,i) = 0;
         Xcoord(graph,i) = 0;
         Ycoord(graph,i) = 0;
         for (j = i+1; j <= N; ++j)
      {
          if (MarkedGraph.GetWeight(i-1, j-1) < 0) continue;
```

```
                AddEdge (graph,i,j, (wtype) MarkedGraph.GetWeight(i-1, j-1));
      }
   }
   Mate = Weighted_Match((int)graph,1,1);
   for (i = 1; i <= N; ++i)
   {
         for (j = i+1; j <= N; ++j)
      {
         if (MarkedGraph.GetWeight(i-1, j-1) < 0) continue;
         RemoveEdge(graph, FindEdge(graph,i,j));
      }
   }
   free(graph);
   for (int i = 1; i <= N; i++)
      if (Mate[i] != 0)
         MarkedGraph.SetMarked(i-1, Mate[i]-1);
   delete[] Mate;
   return MarkedGraph;
}

TMarkedGraph& FindMaxTwoMatching(TMarkedGraph &MarkedGraph)
{
   int *Mate;
   Graph graph;
   int edges, degree, vlabel, elabel, adj_node;
   int i, j;
   int xcoord, ycoord;
   int N = MarkedGraph.size();
   graph = NewGraph(2*N);
   for (i = 1; i <= N; ++i)
   {
         NLabel(graph,i) = 0;
         Xcoord(graph,i) = 0;
         Ycoord(graph,i) = 0;
         NLabel(graph,i+N) = 0;
         Xcoord(graph,i+N) = 0;
         Ycoord(graph,i+N) = 0;
         for (j = i+1; j <= N; ++j)
      {
         if (MarkedGraph.GetWeight(i-1, j-1) < 0) continue;
               AddEdge (graph,i,j, (wtype) MarkedGraph.GetWeight(i-1, j-1));
         AddEdge (graph,i+N,j, (wtype) MarkedGraph.GetWeight(i-1, j-1));
         AddEdge (graph,i,j+N, (wtype) MarkedGraph.GetWeight(i-1, j-1));
         AddEdge (graph,i+N,j+N, (wtype) MarkedGraph.GetWeight(i-1, j-1));
      }
   }
   Mate = Weighted_Match((int)graph,1,1);
   for (i = 1; i <= N; ++i)
   {
         for (j = i+1; j <= N; ++j)
      {
         if (MarkedGraph.GetWeight(i-1, j-1) < 0) continue;
         RemoveEdge(graph, FindEdge(graph,i,j));
         RemoveEdge(graph, FindEdge(graph,i+N,j));
         RemoveEdge(graph, FindEdge(graph,i,j+N));
         RemoveEdge(graph, FindEdge(graph,i+N,j+N));
      }
   }
   free(graph);
   for (int Count = 1; Count <= 2*N; Count++)
```

```
      MarkedGraph.SetMarked((Count-1)%N, (Mate[Count]-1)%N);
   delete[] Mate;
   return MarkedGraph;
}

TMarkedGraph& FindMaxTwoFactor(TMarkedGraph &MarkedGraph)
{
   int *Mate, *MateF;
   Graph graph;
   int edges, degree, vlabel, elabel, adj_node;
   int i, j;
   int xcoord, ycoord;
   int N = MarkedGraph.size();
   graph = NewGraph(N*(N-1) + 2*N);
   int Count = 2*N + 1;
   for (i = 1; i <= N; ++i)
   {
         NLabel(graph,i) = 0;
         Xcoord(graph,i) = 0;
         Ycoord(graph,i) = 0;
         NLabel(graph,i+N) = 0;
         Xcoord(graph,i+N) = 0;
         Ycoord(graph,i+N) = 0;
         for (j = i+1; j <= N; ++j)
      {
         if (MarkedGraph.GetWeight(i-1, j-1) < 0) continue;
               NLabel(graph,Count) = 0;
               Xcoord(graph,Count) = 0;
               Ycoord(graph,Count) = 0;
            NLabel(graph,Count+1) = 0;
         Xcoord(graph,Count+1) = 0;
               Ycoord(graph,Count+1) = 0;
         AddEdge (graph,i,Count, (wtype) MarkedGraph.GetWeight(i-1, j-1));
         AddEdge (graph,i+N,Count, (wtype) MarkedGraph.GetWeight(i-1, j-1));
         AddEdge (graph,Count,Count+1, (wtype) MarkedGraph.GetWeight(i-1,
j-1)+1);
         AddEdge (graph,j,Count+1, (wtype) MarkedGraph.GetWeight(i-1, j-1));
         AddEdge (graph,j+N,Count+1, (wtype) MarkedGraph.GetWeight(i-1, j-1));
         Count = Count + 2;
      }
   }

   Mate = Weighted_Match((int)graph,1,1);

   Count = 2*N + 1;
   for (i = 1; i <= N; ++i)
   {
         for (j = i+1; j <= N; ++j)
      {
         if (MarkedGraph.GetWeight(i-1, j-1) < 0) continue;
         RemoveEdge(graph, FindEdge(graph,i,Count));
         RemoveEdge(graph, FindEdge(graph,i+N,Count));
         RemoveEdge(graph, FindEdge(graph,Count,Count+1));
         RemoveEdge(graph, FindEdge(graph,j,Count+1));
         RemoveEdge(graph, FindEdge(graph,j+N,Count+1));
         Count = Count + 2;
      }
   }
   free(graph);
   for (int Count = 2*N+1; Count <= N*(N-1) + 2*N; Count+=2)
```

```
      if ( ((int)Mate[Count]   <= 2*N)&&(Mate[Count]   > 0)&&
           ((int)Mate[Count+1] <= 2*N)&&(Mate[Count+1] > 0) )
         MarkedGraph.SetMarked((Mate[Count]-1)%N, (Mate[Count+1]-1)%N);

   delete[] Mate;
   return MarkedGraph;
}


#pragma package(smart_init)
```

П3.11. Файл " BaBAlg.h".

```
#pragma hdrstop
#include "Alg13_1.h"
#include "TMarkedGraph.h"
#include "TEdge.h"
#include <stdlib.h>
TMarkedGraph& Alg13_1(TMarkedGraph& Graph)
{
   TEdge E;
   Graph.ClearMarks();
   while (Graph.GetMarksCount() != Graph.size())
   {
      E = Graph.GreedyEdgeFinding();
      Graph.SetMarked(E.From, E.To);
   }
   return Graph;
}
#pragma package(smart_init)
```

П3.12. Файл " BaBAlg.cpp".

```
#pragma hdrstop

#include <stdio.h>
#include <math.hpp>
#include <system.hpp>
#include <classes.hpp>
#include "BaBAlg.h"
#include "TMarkedGraph.h"
#include "TEdge.h"
TNode::TNode(int Dim_, double **Weights_)
   : GraphDim(Dim_)
{
   Dim=Dim_;
   Weights=Weights_;
   Active=true;
   LLink=NULL;
   RLink=NULL;
   Parent=NULL;
   Bound= 0;
   HC=new TEdge[Dim];
   r=new double[Dim];
   s=new double[Dim];
   VVer=new int[Dim];
   VHor=new int[Dim];
   for(int i=0;i<Dim;i++)
   {
      VVer[i]=i;
      VHor[i]=i;
```

```
      HC[i]=__EmptyEdge;
   }
}

TNode::TNode(TNode *Parent, double **Weights_)
   : GraphDim(Parent->GraphDim)
{
   Dim=Parent->Dim;
   Weights=Weights_;
   Active=true;
   LLink=NULL;
   RLink=NULL;
   Parent=Parent;
   Bound=Parent->Bound;
   HC=new TEdge[GraphDim];
   r=new double[Dim];
   s=new double[Dim];
   VVer=new int[Dim];
   VHor=new int[Dim];
   for(int i=0;i <Dim;i++)
   {
      VVer[i]=Parent->VVer[i];
      VHor[i]=Parent->VHor[i];
      HC[i]=Parent->HC[i];
   }
   for(int i=Parent->Dim;i<GraphDim;i++)
      HC[i]=Parent->HC[i];
}

TNode::~TNode()
{
   int i;
   for(i = 0; i < Dim; i++)
      delete[] Weights[i];
   delete[] Weights;
   delete[] VVer;
   delete[] VHor;
   delete[] r;
   delete[] s;
   delete[] HC;
}

void DeleteNode(TNode* Node)
{
   if (Node->LLink != NULL)
   {
      DeleteNode(Node->LLink);
      Node->LLink = NULL;
   }
   if (Node->RLink != NULL)
   {
      DeleteNode(Node->RLink);
      Node->RLink = NULL;
   }
   delete Node;
}


void Reduce(TNode* Node)
{
```

```
int i, j, MaxInd, MaxInd2;
double Max, SndMax, Max2, SndMax2;
MaxInd  = 0;
MaxInd2 = 0;
for(i = 0; i < Node->Dim; i++)
{
   Max = -MaxDouble;
   for(j = 0; j < Node->Dim; j++)
      if (Node->Weights[i][j] > Max)
         Max = Node->Weights[i][j];
   for(j = 0; j < Node->Dim; j++)
   {
      if (Node->Weights[i][j] == -MaxDouble) continue;
      Node->Weights[i][j] -= Max;
   }
   if (Max == -MaxDouble)
      Node->Bound == Max;
   else
      Node->Bound += Max;
}

for(i = 0; i < Node->Dim; i++)
{
   Max = -MaxDouble;
   for(j = 0; j < Node->Dim; j++)
      if (Node->Weights[j][i] > Max)
         Max = Node->Weights[j][i];
   for(j = 0; j < Node->Dim; j++)
   {
      if (Node->Weights[j][i] == -MaxDouble) continue;
      Node->Weights[j][i] -= Max;
   }
   if (Max == -MaxDouble)
      Node->Bound == Max;
   else
      Node->Bound += Max;
}

for(i = 0; i < Node->Dim; i++)
{
   Max  = -MaxDouble;
   Max2 = -MaxDouble;
   for(j = 0; j < Node->Dim; j++)
   {
      if (Node->Weights[i][j] > Max)
      {
         Max = Node->Weights[i][j];
         MaxInd = j;
      }
      if (Node->Weights[j][i] > Max2)
      {
         Max2 = Node->Weights[j][i];
         MaxInd2 = j;
      }
   }
   SndMax  = -MaxDouble;
   SndMax2 = -MaxDouble;
   for(j = 0; j < Node->Dim; j++)
   {
      if (j != MaxInd)
```

```
            if (Node->Weights[i][j] > SndMax)
               SndMax = Node->Weights[i][j];
         if (j != MaxInd2)
            if (Node->Weights[j][i] > SndMax2)
               SndMax2 = Node->Weights[j][i];
      }
      Node->r[i] = SndMax;
      Node->s[i] = SndMax2;
   }
}

void ExcludePossibleCycles(TNode* Node, TMarkedGraph& TestGraph)
{
   TestGraph.ClearMarks();
   int i, j;
   int HCSize = Node->GraphDim - Node->Dim;
   if (HCSize < 2)
      return;
   for(i = 0; i < HCSize; i++)
      TestGraph.SetMarked(Node->HC[i]);
   TEdge E;
   for(i = 0; i < Node->Dim; i++)
      for(j = 0; j < Node->Dim; j++)
         if (Node->Weights[i][j] != -MaxDouble)
         {
            E.From = Node->VVer[i];
            E.To   = Node->VHor[j];
            if (TestGraph.CheckEdgeForTourInclusion(E) == false)
               Node->Weights[i][j] = -MaxDouble;
         }
}

TEdge ChooseEdge(TNode* Node)
{
   double Min = MaxDouble, at_i, at_j;
   int i, j;
   TEdge E;
   double Value;
   if (Node->Dim == 1)
   {
      E.From = 0;
      E.To   = 0;
      return E;
   }
   for(i = 0; i < Node->Dim; i++)
      for(j = 0; j < Node->Dim; j++)
      {
         if (Node->Weights[i][j] != 0)
            continue;
         at_i = Node->r[i];
         at_j = Node->s[j];
         if ((at_i == -MaxDouble)||(at_j == -MaxDouble))
            Value = -MaxDouble;
         else
            Value = Node->r[i] + Node->s[j];
         if (Value < Min)
         {
            Min = Value;
            E.From = i;
            E.To   = j;
```

```
         }
      }
   if (Min == MaxDouble)
      return __EmptyEdge;
   return E;
}

void DeleteRowCol(TNode *Node, int Row, int Col)
{
   double **Weights;
   Node->Dim = Node->Dim-1;
   Weights = new double*[Node->Dim];
   for(int i = 0; i < Node->Dim; i++)
      Weights[i] = new double[Node->Dim];
   int i, j, i2, j2;
   i2 = 0;
   for(i = 0; i < Node->Dim+1; i++)
   {
      if (i == Row) continue;
      j2 = 0;
      for(j = 0; j < Node->Dim+1; j++)
      {
         if (j == Col) continue;
         Weights[i2][j2] = Node->Weights[i][j];
         j2++;
      }
      i2++;
   }
   for(int i = 0; i < Node->Dim+1; i++)
      delete[] Node->Weights[i];
   delete[] Node->Weights;
   Node->Weights = Weights;
   int *VVer_, *VHor_;
   VVer_     = new int[Node->Dim];
   VHor_     = new int[Node->Dim];
   int VerInd, HorInd;
   VerInd = 0;
   HorInd = 0;

   for(i = 0; i < Node->Dim+1; i++)
   {
      if (i != Row)
      {
         VVer_[VerInd] = Node->VVer[i];
         VerInd++;
      }
      if (i != Col)
      {
         VHor_[HorInd] = Node->VHor[i];
         HorInd++;
      }
   }

   delete[] Node->VVer;
   delete[] Node->VHor;
   Node->VVer = VVer_;
   Node->VHor = VHor_;
}
```

```
void MakeSons(TNode *Node, TEdge E)
{
   double **WeightsL, **WeightsR;
   int i, j;
   WeightsL = new double*[Node->Dim];
   for(i = 0; i < Node->Dim; i++)
      WeightsL[i] = new double[Node->Dim];
   WeightsR = new double*[Node->Dim];
   for(i = 0; i < Node->Dim; i++)
      WeightsR[i] = new double[Node->Dim];
   for(i = 0; i < Node->Dim; i++)
      for(j = 0; j < Node->Dim; j++)
      {
         WeightsL[i][j] = Node->Weights[i][j];
         WeightsR[i][j] = Node->Weights[i][j];
      }
   Node->LLink = new TNode(Node, WeightsL);
   Node->RLink = new TNode(Node, WeightsR);
   Node->LLink->Bound += Node->Weights[E.From][E.To];
   Node->LLink->Note = Node->Note + (AnsiString)"L";
   Node->RLink->Note = Node->Note + (AnsiString)"R";
   TEdge RealE;
   RealE.From = Node->VVer[E.From];
   RealE.To   = Node->VHor[E.To];
   Node->LLink->HC[Node->GraphDim - Node->Dim] = RealE;
   DeleteRowCol(Node->LLink, E.From, E.To);
   for(i = 0; i < Node->LLink->Dim; i++)
   {
      if ((int)Node->LLink->VVer[i] != RealE.To)
         continue;
      for(j = 0; j < Node->LLink->Dim; j++)
      {
         if ((int)Node->LLink->VHor[j] != RealE.From)
            continue;
         Node->LLink->Weights[i][j] = -MaxDouble;
         break;
      }
      break;
   }

   Node->RLink->Weights[E.From][E.To] = -MaxDouble;
   Node->Active = false;
}

TNode* ChooseActiveNode(TList *Active)
{
   int i, ind;
   TNode *Node;
   double Max = -MinDouble;
   for (i = 0; i < (int)Active->Count; i++)
   {
      Node = (TNode *)Active->Items[i];
      if (Node->Bound > Max)
      {
         Max = Node->Bound;
         ind = i;
      }
   }
   Node = (TNode *)Active->Items[ind];
   Active->Delete(ind);
```

```
   return Node;
}

void ThinActiveNodes(TList *Active, double CurrBound)
{
   TNode *Node;
   int i = 0;
   while (i < (int)Active->Count)
   {
      Node = (TNode*) Active->Items[i];
      if (Node->Bound <= CurrBound)
      {
         Active->Delete(i);
         delete Node;
      }
      else
         i++;
   }
}
TMarkedGraph& BranchAndBoundAlg(TMarkedGraph& Graph)
{
   TNode* CurrMaxNode = NULL;
   double **Weights, CurrBound = 0;
   TList *Active;
   TEdge CurrE;
   TEdge RealE;
   int i,j, Dim;
   Dim = Graph.size();
   Weights = new double*[Dim];
   for(int i = 0; i < Dim; i++)
      Weights[i] = new double[Dim];
   for(i = 0; i < Dim; i++)
   {
      for(j = i; j < Dim; j++)
      {
         if (i == j)
         {
            Weights[i][j] = -MaxDouble;
            continue;
         }
         Weights[i][j] = Graph.GetWeight(i, j);
         Weights[j][i] = Weights[i][j];
      }
   }
   TNode *Root, *CurrNode;
   Root = new TNode(Dim, Weights);
   Root->Note = (AnsiString)"+";
   Reduce(Root);
   Active = new TList;
   Active->Add(Root);
   int ActiveNodesCount;
   while(Active->Count > 0)
   {
      CurrNode = ChooseActiveNode(Active);
      CurrE    = ChooseEdge(CurrNode);
      if (CurrE == __EmptyEdge)
         continue;

      RealE.From = CurrNode->VVer[CurrE.From];
      RealE.To   = CurrNode->VHor[CurrE.To];
```

```
      MakeSons(CurrNode, CurrE);

      if (CurrNode->RLink != NULL)
      {
         ExcludePossibleCycles(CurrNode->RLink, Graph);
         Reduce(CurrNode->RLink);
         if (CurrNode->RLink->Bound > CurrBound)
            Active->Insert(0, CurrNode->RLink);
      }
      if (CurrNode->LLink != NULL)
      {
         ExcludePossibleCycles(CurrNode->LLink, Graph);
         Reduce(CurrNode->LLink);
         if (CurrNode->LLink->Dim == 0)
         {
            if (CurrNode->LLink->Bound > CurrBound)
            {
               CurrMaxNode = CurrNode->LLink;
               CurrBound   = CurrNode->LLink->Bound;
               ThinActiveNodes(Active, CurrBound);
            }
         }
         else
            if (CurrNode->LLink->Bound > CurrBound)
               Active->Insert(0, CurrNode->LLink);
      }
      delete CurrNode;
   }
   Graph.ClearMarks();
   Dim = CurrMaxNode->GraphDim - CurrMaxNode->Dim;
   for(i = 0; i < Dim; i++)
   {
      CurrE = CurrMaxNode->HC[i];
      Graph.SetMarked(CurrE);
   }
   delete Active;
   return Graph;
}

TMarkedGraph& BranchAndBoundExtndAlg(TMarkedGraph& Graph)
{
   double **Weights, CurrBound = 0;
   TList *Active;
   TEdge CurrE;
   TEdge RealE;
   int i,j, Dim;
   Dim = Graph.size();
   Weights = new double*[Dim];
   for(int i = 0; i < Dim; i++)
      Weights[i] = new double[Dim];
   for(i = 0; i < Dim; i++)
   {
      for(j = i; j < Dim; j++)
      {
         if (i == j)
         {
            Weights[i][j] = -MaxDouble;
            continue;
         }
         Weights[i][j] = Graph.GetWeight(i, j);
         Weights[j][i] = Weights[i][j];
```

```
   }
}
TNode *Root, *CurrNode;
TEdge *MaxHC;
MaxHC = new TEdge[Dim];
double SaturatedSumm = 0, Summ = 0;
TMarkedGraph SaturatedGraph(Graph);
SaturatedGraph.ClearMarks();
Summ = Graph.MarkMaxHCycleHR1Alg();
if (Summ > SaturatedSumm)
{
   SaturatedGraph.CopyMarks(Graph);
   SaturatedSumm = Summ;
}
Summ = Graph.MarkMaxHCycleSerdyukovAlg();
if (Summ > SaturatedSumm)
{
   SaturatedGraph.CopyMarks(Graph);
   SaturatedSumm = Summ;
}
Summ = Graph.MarkMaxHCycleAlg14_1();
if (Summ > SaturatedSumm)
{
   SaturatedGraph.CopyMarks(Graph);
   SaturatedSumm = Summ;
}
Summ = Graph.MarkMaxHCycleAlg13_2();
if (Summ > SaturatedSumm)
{
   SaturatedGraph.CopyMarks(Graph);
   SaturatedSumm = Summ;
}
Summ = Graph.MarkMaxHCycleAlg13_1();
if (Summ > SaturatedSumm)
{
   SaturatedGraph.CopyMarks(Graph);
   SaturatedSumm = Summ;
}
TPath HCycle;
deque<TEdge> HC;
HCycle = SaturatedGraph.GetCycle(1);
CurrBound = SaturatedGraph.GetMarkedWeight();
if (Summ > 0)
   for(i = 0; i < Dim; i++)
   {
      CurrE.From = HCycle.at(i);
      CurrE.To   = HCycle.at(i+1);
      HC.push_back(CurrE);
      MaxHC[i] = CurrE;
   }
Root = new TNode(Dim, Weights);
Root->Note = (AnsiString)"+";
Reduce(Root);
Active = new TList;
Active->Add(Root);
CurrNode = Root;
int ActiveNodesCount;
while(Active->Count > 0)
{
   if (HC.empty() == false)
```

```
{
   if (CurrNode->LLink == NULL)
      CurrNode = ChooseActiveNode(Active);
   else
   {
      CurrNode = CurrNode->LLink;
      for (i = 0; i < (int)Active->Count; i++)
      {
         if ((TNode *)Active->Items[i] == CurrNode)
         {
            Active->Delete(i);
            break;
         }
      }
   }
   RealE    = HC.front();

   for(i = 0; i < CurrNode->Dim; i++)
   {
      if ((int)CurrNode->VVer[i] != RealE.From)
         continue;
      for(j = 0; j < CurrNode->Dim; j++)
      {
         if ((int)CurrNode->VHor[j] != RealE.To)
            continue;
         CurrE.From = i;
         CurrE.To   = j;
         break;
      }
      break;
   }
   HC.pop_front();
}
else
{
   CurrNode = ChooseActiveNode(Active);
   CurrE    = ChooseEdge(CurrNode);
}
if (CurrE == __EmptyEdge)
   continue;
RealE.From = CurrNode->VVer[CurrE.From];
RealE.To   = CurrNode->VHor[CurrE.To];
// создаем сыновей
MakeSons(CurrNode, CurrE);
if (CurrNode->RLink != NULL)
{
   ExcludePossibleCycles(CurrNode->RLink, Graph);
   Reduce(CurrNode->RLink);
   if (CurrNode->RLink->Bound > CurrBound)
      Active->Insert(0, CurrNode->RLink);
}
if (CurrNode->LLink != NULL)
{
   ExcludePossibleCycles(CurrNode->LLink, Graph);
   Reduce(CurrNode->LLink);
   if (CurrNode->LLink->Dim == 0)
   {
      if (CurrNode->LLink->Bound > CurrBound)
      {
         CurrBound   = CurrNode->LLink->Bound;
```

```
                for (i = 0; i < CurrNode->LLink->GraphDim; i++)
                   MaxHC[i] = CurrNode->LLink->HC[i];
                ThinActiveNodes(Active, CurrBound);
             }
          }
          else
          {
             if (CurrNode->LLink->Bound > CurrBound)
             {
                Active->Insert(0, CurrNode->LLink);
             }
          }
       }
       delete CurrNode;
    }
    Graph.ClearMarks();
    for(i = 0; i < Dim; i++)
    {
       CurrE = MaxHC[i];
       Graph.SetMarked(CurrE);
    }
    delete Active;
    return Graph;
}
#pragma package(smart_init)
```

П3.13. Файл " Alg13_1.h".

```
#ifndef Alg13_1H
#define Alg13_1H
#include "TMarkedGraph.h"
TMarkedGraph& Alg13_1(TMarkedGraph&);
#endif
```

П3.14. Файл " Alg13_1.cpp".

```
#pragma hdrstop
#include "Alg13_1.h"
#include "TMarkedGraph.h"
#include "TEdge.h"
#include <stdlib.h>
TMarkedGraph& Alg13_1(TMarkedGraph& Graph)
{
   TEdge E;
   Graph.ClearMarks();
   while (Graph.GetMarksCount() != Graph.size())
   {
      E = Graph.GreedyEdgeFinding();
      Graph.SetMarked(E.From, E.To);
   }
   return Graph;
}
//---------------------------------------------------------------------------
#pragma package(smart_init)
```

П3.15. Файл " Alg13_2.h".

```
#ifndef Alg13_2H
#define Alg13_2H
#include "TMarkedGraph.h"
```

```
TMarkedGraph& Alg13_2(TMarkedGraph&);
```



```
//---------------------------------------------------------------------------
#endif
```

П3.16. Файл " Alg13_2.cpp".

```
#pragma hdrstop
#include "Alg13_2.h"
#include "TMarkedGraph.h"
#include "TEdge.h"
#include <stdlib.h>
#pragma package(smart_init)
TMarkedGraph& Alg13_2(TMarkedGraph& Graph)
{
   TEdge E;
   Graph.ClearMarks();
   Graph.MarkMaxMatching();
   while (Graph.GetMarksCount()!=Graph.size())
   {
         E=Graph.GreedyEdgeFinding();
         Graph.SetMarked(E.From,E.To);
   }
   return Graph;
}
#pragma package(smart_init)
```

П3.17. Файл " SerdyukovAlg.h".

```
#ifndef SerdyukovAlgH
#define SerdyukovAlgH
#include "TMarkedGraph.h"
TMarkedGraph& SerdyukovAlg(TMarkedGraph&);
#endif
```

П3.18. Файл " SerdyukovAlg.cpp".

```
#include <stdlib.h>
#pragma hdrstop
#include <Math.hpp>
#include "SerdyukovAlg.h"
#include "TMarkedGraph.h"
#include "TEdge.h"
TMarkedGraph& SerdyukovAlg(TMarkedGraph& Graph)
{
   int CycleLength;
   int V1, V2, V3, Seq;
   TEdge E;
   if (Graph.size() % 2 == 0)  // четный случай
   {
      TMarkedGraph GraphC(Graph);
      TMarkedGraph GraphM(Graph);
      TGraphCycles GraphCycles;
      GraphC.ClearMarks();
      GraphM.ClearMarks();
      GraphC.MarkMax2Factor();
      GraphM.MarkMaxMatching();
      GraphCycles = GraphC.GetGraphCycles();
      randomize();
      int CycleCount = GraphCycles.size();
      for (int i = 0; i < CycleCount; i++)
      {
```

```
      CycleLength = GraphCycles.at(i).size() - 1;
      Seq = random(CycleLength);
      V1 = GraphCycles.at(i).at((Seq + CycleLength)%CycleLength);
      V2 = GraphCycles.at(i).at((Seq + CycleLength - 1)%CycleLength);
      V3 = GraphCycles.at(i).at((Seq + CycleLength + 1)%CycleLength);
      if ((GraphM.GetMarked(V1, V2) == false)&&
 (GraphM.RouteExisting(V1, V2) == -1))
      {
         GraphM.SetMarked(V1, V2);
         GraphC.SetUnmarked(V1, V2);
      }
      else if ((GraphM.GetMarked(V1, V3) == false)&&
(GraphM.RouteExisting(V1, V3) == -1))
      {
         GraphM.SetMarked(V1, V3);
         GraphC.SetUnmarked(V1, V3);
      }
   }
   if (GraphM.GetMarkedWeight() > GraphC.GetMarkedWeight())
      Graph.CopyMarks(GraphM);
   else
      Graph.CopyMarks(GraphC);
   while (Graph.GetMarksCount() != Graph.size())
   {
      E = Graph.GreedyEdgeFinding();
      Graph.SetMarked(E.From, E.To);
   }
}
else                        // нечетный случай
{
   TMarkedGraph GraphC(Graph);
   TMarkedGraph GraphM(Graph);
   TGraphCycles GraphCycles;
   GraphC.ClearMarks();
   GraphM.ClearMarks();
   GraphC.MarkMax2Factor();
   GraphM.MarkMaxMatching();
   int V0;
   // найдем непокрытую паросочетанием вершину
   for (int i = 0; i < GraphM.size(); i++)
      if (GraphM.MarkedDegree(i) == 0)
      {
         V0 = i;
         break;
      }
   int V1 = 1;
   double EdgeWeight;
   double MaxEdgeWeight = -MaxDouble;
   // опередлим ребро (V0, V1) максимального веса
   for (int i = 0; i < GraphC.size(); i++)
   {
      if (i == V0) continue;
      if (GraphC.GetMarked(V0, i) == true) continue;
      EdgeWeight = GraphC.GetWeight(V0, i);
      if (GraphC.GetWeight(V0, i) > MaxEdgeWeight)
      {
         V1 = i;
         MaxEdgeWeight = EdgeWeight;
      }
   }
```

```
TEdge E0(V0, V1);

// найдем ребра E1 и E2
TEdge E1 = __EmptyEdge;
TEdge E2 = __EmptyEdge;
for (int i = 0; i < GraphC.size(); i++)
{
   if (GraphC.GetMarked(V0, i) == true)
   {
      if (E1 == __EmptyEdge)
      {
         E1.From = V0;
         E1.To   = i;
      }
      else
      {
         E2.From = V0;
         E2.To   = i;
         break;
      }
   }
}
TPath C1 = GraphC.GetCycle(V0);
// отметим в графе Graph цикл C0
Graph.ClearMarks();
for (int i = 0; i < (int)C1.size()-1; i++)
   Graph.SetMarked(C1.at(i), C1.at(i+1));
if (Graph.MarkedDegree(V1) == 0)
{
   TPath C2 = GraphC.GetCycle(V1);
    for (int i = 0; i < (int)C2.size()-1; i++)
      Graph.SetMarked(C2.at(i), C2.at(i+1));
}
// найдем ребро E3
TEdge E3 = __EmptyEdge;
for (int i = 0; i < Graph.size(); i++)
{
   if (GraphM.GetMarked(V1, i) == true)
      continue;
   if (Graph. GetMarked(V1, i) == true)
   {
      E3.From = V1;
      E3.To   = i;
      break;
   }
}
// найдем ребро E
TEdge E = __EmptyEdge;
E = E1;
GraphC.SetMarked(E0);
GraphC.SetUnmarked(E);
GraphC.SetUnmarked(E3);
if (GraphC.GetCycle(V0).size() > 0)
{
   GraphC.SetMarked(E);
   E = E2;
   GraphC.SetUnmarked(E);
}
GraphM.SetMarked(E);
GraphM.SetMarked(E3);
GraphCycles = GraphC.GetGraphCycles();
```

```
      randomize();

      int CycleCount = GraphCycles.size();
      for (int i = 0; i < CycleCount; i++)
      {
         CycleLength = GraphCycles.at(i).size() - 1;
         Seq = random(CycleLength);
         V1 = GraphCycles.at(i).at((Seq + CycleLength)%CycleLength);
         V2 = GraphCycles.at(i).at((Seq + CycleLength - 1)%CycleLength);
         V3 = GraphCycles.at(i).at((Seq + CycleLength + 1)%CycleLength);
         if ((GraphM.GetMarked(V1, V2) == false)&&
    (GraphM.RouteExisting(V1, V2) == -1))
         {
            GraphM.SetMarked(V1, V2);
            GraphC.SetUnmarked(V1, V2);
         }
         else if ((GraphM.GetMarked(V1, V3) == false)&&
                  (GraphM.RouteExisting(V1, V3) == -1))
         {
            GraphM.SetMarked(V1, V3);
            GraphC.SetUnmarked(V1, V3);
         }
      }
      if (GraphM.GetMarkedWeight() > GraphC.GetMarkedWeight())
         Graph.CopyMarks(GraphM);
      else
         Graph.CopyMarks(GraphC);

      while (Graph.GetMarksCount() != Graph.size())
      {
         E = Graph.GreedyEdgeFinding();
         Graph.SetMarked(E.From, E.To);
      }
   }
   return Graph;
}
#pragma package(smart_init)
```

П3.19. Файл " SerdyukovAlgMod.h".

```
#ifndef HR1AlgH
#define HR1AlgH
#include "../Common/TMarkedGraph.h"
TMarkedGraph& HR1Alg(TMarkedGraph&, double);
#endif
```

П3.20. Файл " SerdyukovAlgMod.cpp".

```
#pragma hdrstop
#include "HR1Alg.h"
#include "../Common/TMarkedGraph.h"
#include "../Common/Types.h"
#include "../Common/TEdge.h"
#include <stdlib.h>
#include <Math.hpp>
#include <deque>
using namespace std;
void A1(TMarkedGraph& Graph, double Eps)
{
   deque<int> Nodes;
   TGraphCycles GraphCycles;
   int CycleCount, CycleLength;
```

```
   int i, j, k;

   TEdge E, Emin;
   GraphCycles = Graph.GetGraphCycles();
   CycleCount = GraphCycles.size();
   for (i = 0; i < CycleCount; i++)
   {
      CycleLength = GraphCycles.at(i).size() - 1;
      if ((double)CycleLength < (double)Power(Eps, -1))
      {
         int Dim = CycleLength;
         double **Weights;
         Nodes.clear();
         for (j = 0; j < CycleLength; j++)
         {
            Nodes.push_back(GraphCycles.at(i).at(j));
         }
         Weights = new double* [Dim];
         for (k = 0; k < Dim; k++)
            Weights[k]   = new double [Dim];
         for (k = 0; k < Dim; k++)
            for (j = k; j < Dim; j++)
            {
               Weights[k][j] = Graph.GetWeight(Nodes.at(k), Nodes.at(j));
               Weights[j][k] = Weights[k][j];
            }
         TMarkedGraph SubGraph(Dim, Weights);
         for (k = 0; k < Dim; k++)
            delete[] Weights[k];
         delete[] Weights;
         SubGraph.MarkMaxHamiltonianPathCE();
         for (int i = 0; i < SubGraph.size(); i++)
            for (int j = i; j < SubGraph.size(); j++)
            {
               Graph.SetUnmarked(Nodes.at(i), Nodes.at(j));
               if (SubGraph.GetMarked(i, j) == true)
                  Graph.SetMarked(Nodes.at(i), Nodes.at(j));
            }
      }
      else  // удалить самое легкое ребро цикла
      {
         Emin.From = GraphCycles.at(i).at(0);
         Emin.To   = GraphCycles.at(i).at(1);
         for (int j = 0; j < CycleLength; j++)
         {
            E.From = GraphCycles.at(i).at((j + CycleLength)%CycleLength);
            E.To   = GraphCycles.at(i).at((j + CycleLength + 1)%CycleLength);
            if (Graph.GetWeight(E) < Graph.GetWeight(Emin))
               Emin = E;
         }
         Graph.SetUnmarked(Emin);
      }
   }
   // дополним жадным алгоритмом до полного ГЦ
   while (Graph.GetMarksCount() != Graph.size())
   {
      E = Graph.GreedyEdgeFinding();
      Graph.SetMarked(E.From, E.To);
   }
}
void A2(TMarkedGraph& Graph1, TMarkedGraph& Graph2)
```

```
{
   deque<TEdge> M1, M2, M;
   TMarkedGraph GraphM(Graph1);
   TGraphCycles GraphCycles;
   int CycleCount, CycleLength, iCycleLength, jCycleLength;
   int i, j, x, y, size;
   TEdge E, Emin;
   GraphCycles = Graph1.GetGraphCycles();
   CycleCount = GraphCycles.size();
   size = Graph1.size();
   for (i = 0; i < size; i++)
      for (j = i; j < size; j++)
      {
         if (i == j) GraphM.SetWeight(i, j, 0);
         GraphM.SetWeight(i, j, -1);
      }
   for (i = 0; i < CycleCount; i++)
      for (j = i+1; j < CycleCount; j++)
      {
         iCycleLength = GraphCycles.at(i).size() - 1;
         jCycleLength = GraphCycles.at(j).size() - 1;
         for (x = 0; x < iCycleLength; x++)
            for (y = 0; y < jCycleLength; y++)
            {
               E.From = GraphCycles.at(i).at(x);
               E.To   = GraphCycles.at(j).at(y);
               GraphM.SetWeight(E, Graph1.GetWeight(E));
            }
      }
   GraphM.ClearMarks();
   GraphM.MarkMaxMatching();
   Graph1.ClearMarks();
   Graph1.MarkMaxMatching();
   {
   TMarkedGraph GraphW1(Graph1);
   TMarkedGraph GraphW2(Graph1);
   for (i = 0; i < CycleCount; i++)
   {
      CycleLength = GraphCycles.at(i).size() - 1;
      bool AddToFirst = true;
      bool E2AssignedToM2 = false;
      bool E1SkippedToM1  = false;
      bool E2SkippedToM2  = false;
      M1.clear();
      M2.clear();
      for (j = 0; j < CycleLength; j++)
      {
         E.From = GraphCycles.at(i).at(j);
         E.To   = GraphCycles.at(i).at(j+1);
         if (AddToFirst == true)
         {
            if (GraphW1.CheckEdgeForTourInclusion(E) == true)
            {
               if (j == CycleLength - 1)
               {
                  if (E2AssignedToM2 == true)
                  {
                     GraphW1.SetUnmarked(M1.at(0));
                     M1.pop_front();
                  }
```

```
                else if (E2SkippedToM2 == true)
                {
                   GraphW2.SetMarked(M1.at(0));
                   M2.push_back(M1.at(0));
                   GraphW1.SetUnmarked(M1.at(0));
                   M1.pop_front();
                }
             }
             GraphW1.SetMarked(E);
             AddToFirst = false;
             M1.push_back(E);
          }
          else
          {
             if (j == 0) E1SkippedToM1 = true;
          }
       }
       else
       {
          if (GraphW2.CheckEdgeForTourInclusion(E) == true)
          {
             if (j == 1) E2AssignedToM2 = true;
             GraphW2.SetMarked(E);
             AddToFirst = true;
             M2.push_back(E);
          }
          else
          {
             if (j == 1) E2SkippedToM2 = true;
             if (j == CycleLength - 1)
                if (E1SkippedToM1 == true)
                {
                   E.From = GraphCycles.at(i).at(0);
                   E.From = GraphCycles.at(i).at(1);
                   M2.push_back(E);
                   GraphW2.SetMarked(E);
                }
          }
       }
    }
    randomize();
    if (random(2) == 1)
    {
       for (j = 0; j < M1.size(); j++)
       {
          Graph1.SetMarked(M1.at(j));
          Graph2.SetUnmarked(M1.at(j));
       }
    }
    else
    {
       for (j = 0; j < M2.size(); j++)
       {
          Graph1.SetMarked(M2.at(j));
          Graph2.SetUnmarked(M2.at(j));
       }
    }
}
}
// дополним жадным алгоритмом до полного ГЦ
```

```
   while (Graph1.GetMarksCount() != Graph1.size())
   {
      E = Graph1.GreedyEdgeFinding();
      Graph1.SetMarked(E.From, E.To);
   }  // получили тур T2
   GraphCycles = Graph2.GetGraphCycles();
   CycleCount = GraphCycles.size();
   // теперь получим тур T2
   for (i = 0; i < GraphM.size(); i++)
   {
      if (Graph2.MarkedDegree(i) != 1)
         continue;
      for (j = i+1; j < GraphM.size(); j++)
      {
         if (Graph2.MarkedDegree(j) != 1)
            continue;
         if (GraphM.GetMarked(i, j) == true)
            Graph2.SetMarked(i, j);
      }
   }
   int Seq;
   GraphCycles = Graph2.GetGraphCycles();
   CycleCount = GraphCycles.size();
   for (i = 0; i < CycleCount; i++)
   {
      CycleLength = GraphCycles.at(i).size() - 1;
      M.clear();
      for (j = 0; j < CycleLength; j++)
      {
         E.From = GraphCycles.at(i).at(j);
         E.To   = GraphCycles.at(i).at(j+1);
         if (GraphM.GetMarked(E) == true)
            M.push_back(E);
      }
      Seq = random(M.size());
      Graph2.SetUnmarked(M.at(Seq));
   }
   // дополним жадным алгоритмом до полного ГЦ
   while (Graph2.GetMarksCount() != Graph2.size())
   {
      E = Graph2.GreedyEdgeFinding();
      Graph2.SetMarked(E.From, E.To);
   }  // получили тур T3
}
TMarkedGraph& HR1Alg(TMarkedGraph& Graph, double Eps)
{
   TEdge E;
   Graph.ClearMarks();
   Graph.MarkMax2Factor();
   TMarkedGraph Graph1(Graph);
   TMarkedGraph Graph2(Graph);
   TMarkedGraph Graph3(Graph);
   A1(Graph1, Eps);
   A2(Graph2, Graph3);
   int Graph1MarkedWeight = Graph1.GetMarkedWeight();
   int Graph2MarkedWeight = Graph2.GetMarkedWeight();
   int Graph3MarkedWeight = Graph3.GetMarkedWeight();
   if      ((Graph1MarkedWeight > Graph2MarkedWeight)&&
            (Graph1MarkedWeight > Graph3MarkedWeight))
      Graph.CopyMarks(Graph1);
```

```
   else if ((Graph2MarkedWeight > Graph1MarkedWeight)&&
            (Graph2MarkedWeight > Graph3MarkedWeight))
      Graph.CopyMarks(Graph2);
   else
      Graph.CopyMarks(Graph3);
   return Graph;
}
#pragma package(smart_init)
```

П3.21. Файл "AlgAnalysis.h".

```
#ifndef AlgAnalysisH
#define AlgAnalysisH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include <ExtCtrls.hpp>
#include <ComCtrls.hpp>
#include <Chart.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
#include <Series.hpp>
#include <Dialogs.hpp>
#include "TEdge.h"
#include "Types.h"
#include "TMarkedGraph.h"
#pragma hdrstop
using namespace std;
struct TGraphSol
{
   int     GraphSize;
   double   Time;
   double   Summ;
   double   Variation;
   int AlgUsed;
};
typedef vector<TGraphSol>        TAllAlgGraphSols;
typedef vector<TAllAlgGraphSols> TOneDimGraphSols;
typedef vector<TOneDimGraphSols> TAllDimGraphSols;
//---------------------------------------------------------------------------
class TForm2 : public TForm
{
__published:  // IDE-managed Components
   TMainMenu *MainMenu1;
   TMenuItem *N1;
   TMenuItem *N2;
   TMenuItem *N3;
   TMenuItem *N4;
   TMenuItem *N5;
   TMenuItem *N6;
   TGroupBox *GroupBox1;
   TGroupBox *GroupBox2;
   TGroupBox *GroupBox3;
   TCheckBox *CheckBox1;
   TCheckBox *CheckBox2;
   TCheckBox *CheckBox3;
   TCheckBox *CheckBox4;
```

```
   TCheckBox *CheckBox5;
   TCheckBox *CheckBox6;
   TCheckBox *CheckBox7;
   TStaticText *StaticText1;
   TStaticText *StaticText2;
   TButton *Button1;
   TPaintBox *PaintBox1;
   TPaintBox *PaintBox2;
   TButton *Button2;
   TPaintBox *PaintBox3;
   void __fastcall N6Click(TObject *Sender);
   void __fastcall Button1Click(TObject *Sender);
   void __fastcall N3Click(TObject *Sender);
   void __fastcall N4Click(TObject *Sender);
   void __fastcall FormCreate(TObject *Sender);
   void __fastcall Button2Click(TObject *Sender);
   void __fastcall EnableTimer();
   void __fastcall DisableTimer();
   void __fastcall DisplayProgress();
   void __fastcall DisplayResults();
   void __fastcall AnalysisThreadOnTerminate(TObject *Sender);
   void __fastcall StartAnalysisButtonClick(TObject *Sender);
   void __fastcall YesButtonClick(TObject *Sender);
   void __fastcall TimerTimer(TObject *Sender);
private: // User declarations
public:       // User declarations
   __fastcall TForm2(TComponent* Owner);
    int From;
   int To;
   int Step;
   int GraphCount;
   bool Metric;
   vector <int> AlgsNumbers;
   TSPThread *TSP_Thread;
   TThreadPriority Priority;
   AnsiString Filepath;
   void Execute();
   InvertAglsSelection();
   double Eps;
   void DeleteAglsSelection()
   //для анализа результатов
   TAllDimGraphSols AllDimGraphSols;
   TOneDimGraphSols OneDimGraphSols;
   TAllAlgGraphSols AllAlgGraphSols;
   TGraphSol        GraphSol;
   int Progress;
   bool flag;
};
//---------------------------------------------------------------------------
extern PACKAGE TForm2 *Form2;
typedef Set <char, '0', '9'> Numeral;
extern void CheckNumericKeyPressing(char&);
extern TMarkedGraph CreateSymmetricGraph(int, int);
extern Numeral NumericSet;
//---------------------------------------------------------------------------
#endif
```

П3.22. Файл "AlgAnalysis.cpp".

```
#include <vcl.h>
```

```
#pragma hdrstop

#include "AlgAnalysis.h"
#include "TEdge.h"
#include "TMarkedGraph.h"
#include "MAX_TSP.h"
#include <vector>
#include <system.hpp>
#include <FileCtrl.hpp>
#include <fstream>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
AnsiString Title;
AnalysisThread *Analysis_Thread;    // поток для анализа алгоритмов
using namespace std;
// вектора с номерами алгоритмов
vector<int> SymmetricAlgs;
vector<int> PreciseAlgs;
vector<int> AccessorialAlgs;
Set <char, '0', '9'> NumericSet;
bool AnalysisStarted;
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
   : TForm(Owner)
{
   // помещаем номера симметрических алгоритмов
   SymmetricAlgs.push_back(0);
   SymmetricAlgs.push_back(1);
   SymmetricAlgs.push_back(2);
   SymmetricAlgs.push_back(3);
   SymmetricAlgs.push_back(4);
   SymmetricAlgs.push_back(5);
   SymmetricAlgs.push_back(6);
   // помещаем номера точных алгоритмов
   PreciseAlgs.push_back(7);
   PreciseAlgs.push_back(8);
   PreciseAlgs.push_back(9);
   // помещаем номера вспомогательных алгоритмов
   AccessorialAlgs.push_back(10);
   AccessorialAlgs.push_back(11);
   AccessorialAlgs.push_back(12);
   StartAnalysisButton->Enabled = true;
   AUSureText->Visible          = false;
   YesButton->Visible           = false;
   NoButton->Visible            = false;
   // считываем установки
   ifstream infile("AlgAnalysis.cfg");
   if (!infile)
   {
         DimFromEdit->Text    = 3;
         DimToEdit->Text      = 50;
         DimStepEdit->Text    = 1;
         GraphCountEdit->Text = 5;
         MaxTimeEdit->Text    = 60;
         EHR1Edit->Text       = 2;
   }
   else
   {
```

```
            char c;

            int DimFromEditC;
            int DimToEditC;
            int DimStepEditC;
            int GraphCountEditC;
            int MaxTimeEditC;
            int EHR1EditC;
            infile   >>   DimFromEditC   >>   DimToEditC   >>   DimStepEditC   >>
GraphCountEditC
                 >> MaxTimeEditC >> EHR1EditC;
            DimFromEdit->Text    = DimFromEditC;
            DimToEdit->Text      = DimToEditC;
            DimStepEdit->Text    = DimStepEditC;
            GraphCountEdit->Text = GraphCountEditC;
            MaxTimeEdit->Text    = MaxTimeEditC;
            EHR1Edit->Text       = EHR1EditC;
            FilepathMemo->Clear();
            AnsiString Path;
            while ((c = infile.get()) != '\n')
            {
                  Path += (AnsiString) c;
            }
            FilepathMemo->Lines->Strings[0] = Path.Trim();
            int i;
            c = infile.get();
            while (!infile.eof())
            {
                  infile >> i;
                  AlgorithmsListBox->Selected[i] = true;
            }
      }
      SaveGraphsCheckBox->Checked = false;
      NumericSet << '0' << '1' << '2' << '3' << '4' << '5' << '6' << '7'
   << '8' << '9';
      AnalysisStarted = false;
      Timer->Enabled  = false;
      ProgressBar->Visible = false;
   }
   //---------------------------------------------------------------------------
   void __fastcall TForm2::N6Click(TObject *Sender)
   {
      Form2->Close();
   }
   void AddAglsSelection(vector<int> Algs)
   {
      for (int i = 0; i < Algs.size(); i++)
         Form2->AlgorithmsListBox->Selected[Algs.at(i)] = true;
   }
   //---------------------------------------------------------------------------

   void DeleteAglsSelection(vector<int> Algs)
   {
      for (int i = 0; i < Algs.size(); i++)
         Form2->AlgorithmsListBox->Selected[Algs.at(i)] = false;
   }
   //---------------------------------------------------------------------------

   void InvertAglsSelection(vector<int> Algs)
   {
      for (int i = 0; i < Algs.size(); i++)
```

```
      Form2->AlgorithmsListBox->Selected[Algs.at(i)] =
         !Form2->AlgorithmsListBox->Selected[Algs.at(i)];
}
//---------------------------------------------------------------------------

void __fastcall TForm2::StartAnalysisButtonClick(TObject *Sender)
{
   StartAnalysisButton->Enabled = false;
   AUSureText->Visible          = true;
   YesButton->Visible           = true;
   NoButton->Visible            = true;
}
//---------------------------------------------------------------------------

void __fastcall TForm2::YesButtonClick(TObject *Sender)
{
   int i;
   if (AlgorithmsListBox->SelCount > 0)
   {
      // создаем поток анализа алгоритмов
      Analysis_Thread = new AnalysisThread(true);
      // заполняем свойства потока
      Analysis_Thread->From = DimFromEdit->Text.ToInt();
      Analysis_Thread->To   = DimToEdit  ->Text.ToInt();
      Analysis_Thread->Step = DimStepEdit->Text.ToInt();
      Analysis_Thread->GraphCount = GraphCountEdit->Text.ToInt();
      Analysis_Thread->Eps  = (double)EHR1Edit->Text.ToInt()/
(Power(10, Ceil(Log10(EHR1Edit->Text.ToInt()))));
      Analysis_Thread->Filepath  = FilepathMemo->Lines->Text;
      if      (PriorityComboBox->ItemIndex == 0)
         Analysis_Thread->Priority = tpIdle;
      else if (PriorityComboBox->ItemIndex == 1)
         Analysis_Thread->Priority = tpLowest;
      else if (PriorityComboBox->ItemIndex == 2)
         Analysis_Thread->Priority = tpLower;
      else if (PriorityComboBox->ItemIndex == 3)
         Analysis_Thread->Priority = tpNormal;
      else if (PriorityComboBox->ItemIndex == 4)
         Analysis_Thread->Priority = tpHigher;
      else if (PriorityComboBox->ItemIndex == 5)
         Analysis_Thread->Priority = tpHighest;
      else if (PriorityComboBox->ItemIndex == 6)
         Analysis_Thread->Priority = tpTimeCritical;
      Analysis_Thread->Metric = false;
      for (i = 0; i < MetricAlgs.size(); i++)
         if (AlgorithmsListBox->Selected[MetricAlgs.at(i)] == true)
         {
            Analysis_Thread->Metric = true;
            break;
         }
      for (int i = 0; i < AlgorithmsListBox->Items->Count; i++)
         if (AlgorithmsListBox->Selected[i] == true)
            Analysis_Thread->AlgsNumbers.push_back(i);
      // создаем поток анализа алгоритмов
      if (MaxTimeEdit->Text == "")
         Timer->Interval = MaxLongint;
      else
         Timer->Interval = 1000 * MaxTimeEdit->Text.ToInt();
      AnalysisStarted = true;
      KeyPreview = true;
```

```
        ProgressBar->Visible  = true;

         ProgressBar->Position = 0;
         AUSureText->Visible   = false;
         YesButton->Visible    = false;
         NoButton->Visible     = false;
         // запускаем поток анализа алгоритмов
         Analysis_Thread->Resume();
      }
   }


   //---------------------------------------------------------------------------
   void __fastcall TForm2::N3Click(TObject *Sender)
   {
      MessageBox(Handle, ((AnsiString)"Программа Algorithms\n"+
      "Юсова Н.И. гр. ЕТ-222\n"+
      "Все права защищены© 2017").c_str(), "О программе", MB_OK);
   }
   //---------------------------------------------------------------------------
   void __fastcall TForm2::N4Click(TObject *Sender)
   {
      MessageBox(Handle,
      ((AnsiString)"Программа сравнивает алгоритмы для решения задачи коммивояжера
на максимум. "+
      "Алгоритмы, которые необходимо сравнить, отметьте галочкой. "+
      "Необходимо выбрать, какой график требуется построить: сравнивающий "+
      "алгоритмы по точности решения или по времени работы. "+
      "Для построения самого графика нажмите кнопку \"Пуск\".").c_str(), "Помощь",
MB_OK);
   }
   //---------------------------------------------------------------------------
   void __fastcall TForm2::FormCreate(TObject *Sender)
   {
      if (AnalysisStarted == true)
      {
         Analysis_Thread->Terminate();
         EnableItems();
         AnalysisStarted = false;
         KeyPreview = false;
       StartAnalysisButton->Enabled = true;
         ProgressBar->Visible = false;
      }
   }
   //---------------------------------------------------------------------------
    void __fastcall TForm2::TimerTimer(TObject *Sender)
   {
      // прекрацаем выполнение потока
      TerminateThread((void *) Analysis_Thread->TSP_Thread->Handle, -1);
      HMODULE DllInstance;
      DllInstance = GetModuleHandle("MatchingAlgs//MatchingAlgs.dll");
      FreeLibrary(DllInstance);
      DllInstance = GetModuleHandle("PreciseAlgs//PreciseAlgs.dll");
      FreeLibrary(DllInstance);
      DllInstance = GetModuleHandle("ApproxAlgs//ApproxAlgs.dll");
      FreeLibrary(DllInstance);
   }
   void __fastcall TForm2::EnableTimer()
   {
      Form2->Timer->Enabled = true;
   }
```

```
//---------------------------------------------------------------------------
void __fastcall TForm2::DisableTimer()
{
   Form2->Timer->Enabled = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DisplayProgress()
{
   Form2->ProgressBar->Position = Progress;
}
//---------------------------------------------------------------------------
void __fastcall TForm2::DisplayResults()
{
   int i, k, j, Number;
   int Count;
   int AlgsNumber = AlgsNumbers.size();
   TGraphSol TmpGraphSol;
   AnsiString Title;
   int Pos;
   Form2->TimeChart        ->Visible = true;
   Form2->PrecisionChart   ->Visible = true;
   Form2->MaxVariationChart->Visible = true;
   Form2->MinVariationChart->Visible = true;
   Form2->AvgVariationChart->Visible = true;
   for (int k = 0; k < Form2->TimeChart->SeriesCount(); k++)
   {
      Form2->TimeChart        ->Series[k]->Active = false;
      Form2->TimeChart        ->Series[k]->Clear();
      Form2->PrecisionChart   ->Series[k]->Active = false;
      Form2->PrecisionChart   ->Series[k]->Clear();
      Form2->MaxVariationChart->Series[k]->Active = false;
      Form2->MaxVariationChart->Series[k]->Clear();
      Form2->MinVariationChart->Series[k]->Active = false;
      Form2->MinVariationChart->Series[k]->Clear();
      Form2->AvgVariationChart->Series[k]->Active = false;
      Form2->AvgVariationChart->Series[k]->Clear();
   }
   for (k = 0; k < AlgsNumber; k++)
   {
      Number = AlgsNumbers.at(k);
      Title = Form2->AlgorithmsListBox->Items->Strings[AlgsNumbers.at(k)];
      Pos = Title.Pos("Алгоритм ");
      if (Pos > 0)
         Title.Delete(Pos, 9);
      Form2->TimeChart        ->Series[Number]->Title = Title;
      Form2->PrecisionChart   ->Series[Number]->Title = Title;
      Form2->MaxVariationChart->Series[Number]->Title = Title;
      Form2->MinVariationChart->Series[Number]->Title = Title;
      Form2->AvgVariationChart->Series[Number]->Title = Title;
      Form2->TimeChart        ->Series[Number]->Active = true;
      Form2->PrecisionChart   ->Series[Number]->Active = true;
      Form2->MaxVariationChart->Series[Number]->Active = true;
      Form2->MinVariationChart->Series[Number]->Active = true;
      Form2->AvgVariationChart->Series[Number]->Active = true;
   }
   double MaxVariation;
   double MinVariation;
   double AvgVariation;
   double Variation;
   // цикл по размерностям графов
```

```
   for (i = 0; i < AllDimGraphSols.size(); i++)
   {
      for (k = 0; k < AlgsNumber; k++)
      {
         Number = AlgsNumbers.at(k);
         Count = 0;
         TmpGraphSol.Time = 0;
         TmpGraphSol.Summ = 0;
         TmpGraphSol.GraphSize = 0;
         MaxVariation = -Math::MaxDouble;
         MinVariation =  Math::MaxDouble;
         AvgVariation =  0;
         for (j = 0; j < AllDimGraphSols.at(i).size(); j++)
         {
            if (AllDimGraphSols.at(i).at(j).at(k).Summ == -1) continue;
            TmpGraphSol.Time += AllDimGraphSols.at(i).at(j).at(k).Time;
            TmpGraphSol.Summ += AllDimGraphSols.at(i).at(j).at(k).Summ;
            TmpGraphSol.GraphSize =
AllDimGraphSols.at(i).at(j).at(k).GraphSize;
            Variation = AllDimGraphSols.at(i).at(j).at(k).Variation;
            Variation = 100 * Variation / (Variation + TmpGraphSol.Summ);
            if (Variation > MaxVariation)
               MaxVariation = Variation;
            if (Variation < MinVariation)
               MinVariation = Variation;
            AvgVariation += Variation;
            Count++;
         }
         if (Count == 0) continue;
         TmpGraphSol.Time /= Count;
         TmpGraphSol.Summ /= Count;
         AvgVariation     /= Count;
         Form2->TimeChart        ->
Series[Number]->AddXY(TmpGraphSol.GraphSize, TmpGraphSol.Time);
         Form2->PrecisionChart   ->
Series[Number]->AddXY(TmpGraphSol.GraphSize, TmpGraphSol.Summ);
         Form2->MaxVariationChart->
Series[Number]->AddXY(TmpGraphSol.GraphSize, MaxVariation);
         Form2->MinVariationChart->
Series[Number]->AddXY(TmpGraphSol.GraphSize, MinVariation);
         Form2->AvgVariationChart->
Series[Number]->AddXY(TmpGraphSol.GraphSize, AvgVariation);
      }
   }

   int HeightChange = 80;
   int NewMargin    = 30;
   if (AlgsNumber > 2)
   {
      HeightChange = 50;
      NewMargin    = 40;
   }
   if (AlgsNumber > 4)
   {
      HeightChange = 40;
      NewMargin    = 70;
   }
   if (AlgsNumber > 6)
   {
      HeightChange = 20;
```

```
      NewMargin    = 80;
   }
   if (AlgsNumber > 8)
   {
      HeightChange = 0;
      NewMargin    = 100;
   }
   Form2->PrecisionChart   ->Height -= HeightChange;
   Form2->AvgVariationChart->Height -= HeightChange;
   Form2->MaxVariationChart->Height -= HeightChange;
   Form2->MinVariationChart->Height -= HeightChange;
   Form2->TimeChart        ->Height -= HeightChange;
   Form2->PrecisionChart   ->Legend->VertMargin = NewMargin;
   Form2->AvgVariationChart->Legend->VertMargin = NewMargin;
   Form2->MaxVariationChart->Legend->VertMargin = NewMargin;
   Form2->MinVariationChart->Legend->VertMargin = NewMargin;
   Form2->TimeChart        ->Legend->VertMargin = NewMargin;
   Form2->PrecisionChart   ->Color = 0xFFFFFF;
   Form2->AvgVariationChart->Color = 0xFFFFFF;
   Form2->MaxVariationChart->Color = 0xFFFFFF;
   Form2->MinVariationChart->Color = 0xFFFFFF;
   Form2->TimeChart        ->Color = 0xFFFFFF;
   Form2->PrecisionChart   ->SaveToMetafile( "Precision.wmf" );
   Form2->AvgVariationChart->SaveToMetafile( "AvgVariation.wmf" );
   Form2->MaxVariationChart->SaveToMetafile( "MaxVariation.wmf" );
   Form2->MinVariationChart->SaveToMetafile( "MinVariation.wmf" );
   Form2->TimeChart        ->SaveToMetafile( "TimeChart.wmf" );
   Form2->PrecisionChart   ->Color = 0xEFEEEB;
   Form2->AvgVariationChart->Color = 0xEFEEEB;
   Form2->MaxVariationChart->Color = 0xEFEEEB;
   Form2->MinVariationChart->Color = 0xEFEEEB;
   Form2->TimeChart        ->Color = 0xEFEEEB;
   Form2->PrecisionChart   ->Legend->VertMargin = 100;
   Form2->AvgVariationChart->Legend->VertMargin = 100;
   Form2->MaxVariationChart->Legend->VertMargin = 100;
   Form2->MinVariationChart->Legend->VertMargin = 100;
   Form2->TimeChart        ->Legend->VertMargin = 100;
   Form2->PrecisionChart   ->Height += HeightChange;
   Form2->AvgVariationChart->Height += HeightChange;
   Form2->MaxVariationChart->Height += HeightChange;
   Form2->MinVariationChart->Height += HeightChange;
   Form2->TimeChart        ->Height += HeightChange;
}

//---------------------------------------------------------------------------
void __fastcall TForm2::Execute()
{
   FILE *file;
   AnsiString Filename;
   int i, j, k, N;
   TMarkedGraph Graph;
   int AlgsNumber = AlgsNumbers.size();
   int Result;
   double MaxCycle;
   TDateTime  BeginTime, EndTime;
   int Vsego = ((To - From) / Step + 1) * GraphCount * AlgsNumber;
   LARGE_INTEGER CounterBegin, CounterEnd;
   LARGE_INTEGER Frequency;
   QueryPerformanceFrequency(&Frequency);
   bool SaveResults = true;
```

```
   if (Filepath == "")
      SaveResults = false;
   Randomize();
   AllDimGraphSols.clear();
   for (i = From; i <= To; i += Step)
   {
      OneDimGraphSols.clear();
      for (j = 0; j < GraphCount; j++)
      {
         if (Metric == true)
            Graph.CreateRandomMetricGraph(i, 4, 0);
         else
            Graph.CreateRandomSymmetricGraph(i, 4);
         if (SaveResults == true)
         {
            Filename = Filepath + "\\" + (AnsiString)i + "_" +
(AnsiString)j + ".graph";
            file = fopen(Filename.c_str(),"w");
            N = Graph.size();
            fprintf(file, "%d\n", N);
            for (int i_ = 0; i_ < N; i_++)
            {
               for (int j_ = 0; j_ < N; j_++)
                  fprintf(file, "%.2f ", Graph.GetWeight(i_, j_));
               fprintf(file, "\n");
            }
            fprintf(file, "\n");
            fprintf(file, "//-----------------------------------\n");
            fprintf(file, "//\n");
         }
         MaxCycle = -Math::MaxDouble;
         AllAlgGraphSols.clear();
         for (k = 0; k < AlgsNumber; k++)
         {
            if (Terminated == true)
               return;
            Graph.ClearMarks();
            // создаем поток
            TSP_Thread = new TSPThread(true);
            TSP_Thread->Graph = &Graph;
            TSP_Thread->AlgNumber = AlgsNumbers.at(k);
            TSP_Thread->Eps = Eps;
            TSP_Thread->Priority = Priority;
            // включаем таймер
            Synchronize(EnableTimer);
            QueryPerformanceCounter(&CounterBegin);
            // запускаем алгоритм
            TSP_Thread->Resume();
            Result = TSP_Thread->WaitFor();
            QueryPerformanceCounter(&CounterEnd);
            delete TSP_Thread;
            if (Result == -1)
            {
               GraphSol.Time = 0;
               GraphSol.Summ = -1;
               GraphSol.AlgUsed = AlgsNumbers.at(k);
               GraphSol.GraphSize = i;
            }
            else
            {
```

```
               GraphSol.Time = (double)1000*(CounterEnd.QuadPart -
CounterBegin.QuadPart) / Frequency.QuadPart;
               GraphSol.Summ = Graph.GetMarkedWeight();
               GraphSol.AlgUsed = AlgsNumbers.at(k);
               GraphSol.GraphSize = i;
            }
            if (GraphSol.Summ > MaxCycle)
               MaxCycle = GraphSol.Summ;
            AllAlgGraphSols.push_back(GraphSol);
            Progress = (double) 100 * (((i - From) / Step) *
GraphCount * AlgsNumber + j * AlgsNumber + k)
                       / Vsego;
            Synchronize(DisplayProgress);
            Synchronize(DisableTimer);
         }
         // подсчитаем отклонения
         for (k = 0; k < AllAlgGraphSols.size(); k++)
            AllAlgGraphSols.at(k).Variation =
MaxCycle - AllAlgGraphSols.at(k).Summ;
         if (SaveResults == true)
         {
            for (k = 0; k < AllAlgGraphSols.size(); k++)
            {
               fprintf(file, "// Алгоритм %02d\n",
AllAlgGraphSols.at(k).AlgUsed);
               fprintf(file, "//          Время %.2f мс\n",
AllAlgGraphSols.at(k).Time);
               fprintf(file, "//          Сумма %.2f\n",
AllAlgGraphSols.at(k).Summ);
               fprintf(file, "//          Макс- %.2f\n",
AllAlgGraphSols.at(k).Variation);
               fprintf(file, "//          Макс%c %.2f %\n",
'%', 100*AllAlgGraphSols.at(k).Summ/MaxCycle);
            }

            fprintf(file, "//\n");
            fprintf(file, "//-----------------------------------");
            fclose(file);
         }
         OneDimGraphSols.push_back(AllAlgGraphSols);
      }
   AllDimGraphSols.push_back(OneDimGraphSols);
   }
   Synchronize(DisplayResults);
}
//---------------------------------------------------------------------------
void __fastcall TForm2::AnalysisThreadOnTerminate(TObject *Sender)
{
   EnableItems();
   AnalysisStarted = false;
   Form2->KeyPreview = false;
   Form2->StartAnalysisButton->Enabled = true;
   Form2->ProgressBar->Visible = false;
}
```

П3.23. Файл " MAX_TSP.h".

```
#ifndef MAX_TSPH
#define MAX_TSPH
//---------------------------------------------------------------------------
```

```
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Dialogs.hpp>
#include <Menus.hpp>
#include <Grids.hpp>
#include <Mask.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:  // IDE-managed Components
   TMainMenu *MainMenu1;
   TOpenDialog *OpenDialog;
   TSaveDialog *SaveDialog;
   TGroupBox *AlgorithmsGroupBox;
   TGroupBox *GraphGroupBox;
   TMenuItem *N1;
   TMenuItem *N2;
   TMenuItem *N3;
   TMenuItem *N4;
   TMenuItem *N5;
   TMenuItem *N6;
   TMenuItem *N7;
   TMemo *OutputMemo;
   TComboBox *AlgorithmsBox;
   TLabel *Label1;
   TEdit *Edit1;
   TMaskEdit *HiddenEdit1;
   TStaticText *CreateSymmetricGraphButton;
   TStringGrid *GraphWeightsGrid;
   TStaticText *FindSubgraphButton;
   TStaticText *ClearMemoButton;
   TStaticText *Hint;
   TCheckBox *CyclicOrderCheckBox;
   TMemo *CityNamesMemo;
   TMaskEdit *HiddenEdit;
   TStaticText *Alg;
   TStaticText *StaticText1;
   void __fastcall AlgorithmsBoxChange(TObject *Sender);
   void __fastcall N4Click(TObject *Sender);
   void __fastcall Edit1Exit(TObject *Sender);
   void __fastcall Edit1KeyPress(TObject *Sender, char &Key);
   void __fastcall N2Click(TObject *Sender);
   void __fastcall N3Click(TObject *Sender);
   void __fastcall CreateSymmetricGraphButtonClick(TObject *Sender);
   void __fastcall ClearMemoButtonClick(TObject *Sender);
   void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
   void __fastcall FindSubgraphButtonClick(TObject *Sender);
   void __fastcall N6Click(TObject *Sender);
   void __fastcall N7Click(TObject *Sender);
   void __fastcall GraphWeightsGridDblClick(TObject *Sender);
   void __fastcall HiddenEdit1Exit(TObject *Sender);
   void __fastcall HiddenEdit1KeyPress(TObject *Sender, char &Key);
   void __fastcall AlgClick(TObject *Sender);
   void __fastcall StaticText1Click(TObject *Sender);
private: // User declarations
public:       // User declarations
   __fastcall TForm1(TComponent* Owner);
```

```
    typedef Set <char, '0', '9'> Numeral;

    Numeral NumericSet;
    const int MaxWeightDigit;
    const int MaxWeightFractionalDigit;
    const int _MIN_MATRIX_CELL_SIZE_;
    int N;
    double **G;
    void ShowInGrid(double**, int);
    void FillWithRandomSymmetricValues();
    void CheckNumericKeyPressing(char&);
    void CreateFormTable();
    int Alg13_1(double**,int);
    int SerdyukovAlg(double**,int);
    int MethodVG(double **G, int N);
    AnsiString FillWithSpacesLeft(double Num, int Digits);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
extern TStringList Cities;
//---------------------------------------------------------------------------
#endif
```

П3.24. Файл " MAX_TSP.cpp".

```
#include <vcl.h>
#pragma hdrstop
#include <stdio.h>
#include <stdlib.h>
#include <DateUtils.hpp>
#include <algorithm>
#include <math.h>
#include "TEdge.h"
#include "TMarkedGraph.h"
#include "AlgAnalysis.h"
#include "MAX_TSP.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
const int MaxWeightDigit = 4;
const int MaxWeightFractionalDigit = 2;
const int _MIN_MATRIX_CELL_SIZE_ = 40;
int N=5;
double **G;
bool GeoGraph = false;
AnsiString HintStr;
int HintLeft;
int HintTop;
double S;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
   : TForm(Owner)
{
   N=5;
   NumericSet << '0' << '1' << '2' << '3' << '4' << '5' << '6' << '7' << '8'
<< '9';
   AlgorithmsBox->ItemIndex = 0;
   CyclicOrderCheckBox->Enabled=true;
   CyclicOrderCheckBox->Checked=true;
```

```
      HiddenEdit->Width=MaxWeightDigit*8+2*8+MaxWeightFractionalDigit*8-
(MaxWeightDigit+MaxWeightFractionalDigit);
```



```
      HiddenEdit->EditMask = "";
      for (int i=1; i<=MaxWeightDigit;i++)
            HiddenEdit->EditMask=HiddenEdit->EditMask+(AnsiString) "9";
      HiddenEdit->EditMask=HiddenEdit->EditMask+(AnsiString) "\,";
      for (int i=1; i<=MaxWeightFractionalDigit;i++)
            HiddenEdit->EditMask=HiddenEdit->EditMask+(AnsiString) "9";
      HiddenEdit->EditMask=HiddenEdit->EditMask+(AnsiString) ";1";
      Edit1->Text=N;
      CreateFormTable();
       FillWithRandomSymmetricValues();
   }
   //---------------------------------------------------------------------------
   void __fastcall TForm1::AlgorithmsBoxChange(TObject *Sender)
   {
      if ((AlgorithmsBox->ItemIndex == 12)||
           (AlgorithmsBox->ItemIndex == 13))
      {
         CyclicOrderCheckBox->Enabled = false;
         CyclicOrderCheckBox->Checked = false;
      }
      else
      {
         CyclicOrderCheckBox->Enabled = true;
         CyclicOrderCheckBox->Checked = true;
      }

   }
   //---------------------------------------------------------------------------
   //выход из программы
   void __fastcall TForm1::N4Click(TObject *Sender)
   {
      Application->Terminate();
   }
   //---------------------------------------------------------------------------

   void __fastcall TForm1::Edit1Exit(TObject *Sender)
   {
      int i;
      if ((int)N==Edit1->Text.ToInt()) return;
      for (i=0;i<N; i++)
            delete[] G[i];
      delete[] G;
      N=Edit1->Text.ToInt();
      CreateFormTable();
      FillWithRandomSymmetricValues();

   }
   //---------------------------------------------------------------------------
   //изменение размерности графа
   void __fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
   {
      // Enter
      if (Key==13)
      {
            Form1->ActiveControl=GraphWeightsGrid;
            Key=0;
            return;
      }
```

```
      CheckNumericKeyPressing(Key);
   }
   //---------------------------------------------------------------------------
   //сохранение графа
   void __fastcall TForm1::N2Click(TObject *Sender)
   {
      int i, j;
      FILE *file;
      SaveDialog->Title="Сохранить файл";
      if (SaveDialog->Execute())
      {
            file=fopen(SaveDialog->FileName.c_str(),"w");
            fprintf(file, "%d\n", N);
            for (i=0;i<N;i++)
            {
                  for (j=0;j<N;j++)
                  fprintf(file, "%.2f ", G[i][j]);
                  fprintf(file, "\n");
            }
            fclose(file);
      }
   }
   //---------------------------------------------------------------------------
   //загрузка графа
   void __fastcall TForm1::N3Click(TObject *Sender)
   {
      int i, j;
      FILE *file;
      bool ErrFlag1 = true;
      bool ErrFlag2 = true;
      OpenDialog->Title = "Открыть файл";
      if (OpenDialog->Execute())
      {
            if (FileExists(OpenDialog->FileName))
            {
                  file = fopen(OpenDialog->FileName.c_str(),"r+");
                  for (i=0;i<N;i++)
                        delete[] G[i];
                  delete[] G;
                  fscanf(file, "%d", &N);
                  Edit1->Text = N;
                  CreateFormTable();
                  GeoGraph = false;
                  CityNamesMemo->Lines->Clear();
                  for (i=0;i<N;i++)
                  {
                        CityNamesMemo->Lines->Add((AnsiString)(i+1));
                        for (j=0;j<N;j++)
                        {
                              fscanf(file, "%lf ", G[i] + j);
                              G[i][j]=RoundTo(G[i][j], -MaxWeightFractionalDigit);
                              G[i][j]=G[i][j];
                              if (i!=j)
                                    GraphWeightsGrid->Cells[j+1]
[i+1]=FillWithSpacesLeft(G[i][j], MaxWeightDigit);
                              if ((ErrFlag1==true)&&(j<i)&&(G[j][i]!=G[i][j]))
                              {
                                    Application->MessageBox("Матрица несимметрична!
Результаты могут быть некорректны!", "Ошибка", MB_OK);
```

```
                                    ErrFlag1==false;
                              }
                              if ((ErrFlag2==true)&&(j==i)&&(G[j][i]!=0))
                              {
                                    Application->MessageBox("Матрица       содержит
петли! Результаты могут быть некорректны!", "Ошибка", MB_OK);
                                    ErrFlag1==false;
                              }
                        }
                  }
                  fclose(file);
            }
            else Application->MessageBox("Не найден файл", "Ошибка", MB_OK);
      }

   }
   //---------------------------------------------------------------------------
   //создание случайного графа
   void __fastcall TForm1::CreateSymmetricGraphButtonClick(TObject *Sender)
   {
      GeoGraph = false;
      FillWithRandomSymmetricValues();
   }
   //---------------------------------------------------------------------------
   void __fastcall TForm1::ClearMemoButtonClick(TObject *Sender)
   {
      OutputMemo->Lines->Clear();
   }
   //---------------------------------------------------------------------------
   void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
   {
      int i;
      for (i=0;i<N;i++)
      {
            delete[] G[i];
      }
      delete[] G;
   }
   //---------------------------------------------------------------------------
   //решение по выбранному алгоритму
   void __fastcall TForm1::FindSubgraphButtonClick(TObject *Sender)
   {
      int i,j;
      double Summ;
      bool ForMin=false;
      AnsiString iCity,jCity;
      TDateTime  BeginTime, EndTime;
      TMarkedGraph Graph(N,G);
      BeginTime=Now();
      OutputMemo->Lines->Add("----------------------------------------");
        // алгоритм 13.1 (алг координатного подъема)
       if (AlgorithmsBox->ItemIndex == 0)
       {
        OutputMemo->Lines->Add("Алгоритм координатного подъема");
        Summ = Graph.MarkMaxHCycleAlg13_1(ForMin);
          if (Summ == -1)
             return;
       }
      // алгоритм 13.2 (градиентный алг с форой)
       else if (AlgorithmsBox->ItemIndex == 1)
```

```
      {
           OutputMemo->Lines->Add("Градиентный алгоритм с форой");
           Summ = Graph.MarkMaxHCycleAlg13_2(ForMin);
        if (Summ == -1)
           return;
      }
      // алгоритм Сердюкова
      else if (AlgorithmsBox->ItemIndex == 2)
      {
        OutputMemo->Lines->Add("Алгоритм Сердюкова");
        Summ = Graph.MarkMaxHCycleSerdyukovAlg(ForMin);
          if (Summ == -1)
             return;
      }
      // улучшенный алгоритм Сердюкова
      else if (AlgorithmsBox->ItemIndex == 3)
      {
        OutputMemo->Lines->Add("Улучшенный алгоритм Сердюкова");
        Summ = Graph.MarkMaxHCycleSerdyukovAlgMod(0.2, ForMin);
          if (Summ == -1)
             return;
      }
      // метод ветвей и границ
      else if (AlgorithmsBox->ItemIndex == 4)
      {
        OutputMemo->Lines->Add("Метод ветвей и границ");
        Summ = Graph.MarkMaxHCycleBABAlg(ForMin);
        if (Summ == -1)
           return;

      }
      EndTime = Now();
      OutputMemo->Lines->Add("Сумма цикла: " + (AnsiString)Summ);
      OutputMemo->Lines->Add("Время подсчета: " + (AnsiString)
      ((double) MilliSecondsBetween(BeginTime,EndTime) / 1000) + " сек.");



   }
   //---------------------------------------------------------------------------
   //о программе
   void __fastcall TForm1::N6Click(TObject *Sender)
   {
      MessageBox(Handle, ((AnsiString)"Программа MAX TSP\n"+
      "Юсова Н.И. гр. ЕТ-222\n"+
      "Все права защищены© 2017").c_str(), "О программе", MB_OK);
   }
   //---------------------------------------------------------------------------
   //помощь
   void __fastcall TForm1::N7Click(TObject *Sender)
   {
      MessageBox(Handle,
      ((AnsiString)"Программа реализует алгоритмы для решения задачи коммивояжера
на максимум. "+
      "Алгоритм можно выбрать в выпадающем меню в форме \"Алгоритмы\". "+
      "Размерность графа можно задавать, для этого надо ввести в соответствующее
поле число и нажать enter. "+
      "Для создания графа требуется нажать кнопку \"Создать симметрический граф\".
"+
```

```
      "Справа появится матрица весов графа. "+
      "Для решения самой задачи требуется нажать кнопку \"Получить решение\". "+
      "Результат появится в пустом окошке слева. "+
      "Окошко с результатами можно очистить соответствующей кнопкой. "+
      "Граф можно сохранять и загружать. "+
      "Есть возможность сравнить алгоритмы, для этого необходимо нажать "+
      "кнопку \"Сравнить алгоритмы\".").c_str(), "Помощь", MB_OK);
   }
   //---------------------------------------------------------------------------
   //создание таблицы
   void TForm1::CreateFormTable()
   {
      int i, j;
      G=new double*[N];
      for (j=0;j<N;j++)
            G[j]=new double[N];
      Form1->GraphWeightsGrid->ColCount = N + 1;
      Form1->GraphWeightsGrid->RowCount = N + 1;
      Form1->GraphWeightsGrid->DefaultColWidth=Floor((Form1->GraphWeightsGrid-
>Width-10)/Form1->GraphWeightsGrid->ColCount);
      Form1->GraphWeightsGrid->DefaultRowHeight=Floor((Form1->GraphWeightsGrid-
>Height-10)/Form1->GraphWeightsGrid->RowCount);
      if    ((Form1->GraphWeightsGrid->DefaultColWidth<_MIN_MATRIX_CELL_SIZE_)    ||
(Form1->GraphWeightsGrid->DefaultRowHeight<_MIN_MATRIX_CELL_SIZE_))
      {
            Form1->GraphWeightsGrid->DefaultColWidth=_MIN_MATRIX_CELL_SIZE_;
            Form1->GraphWeightsGrid->DefaultRowHeight=_MIN_MATRIX_CELL_SIZE_;
      }
   }
   //---------------------------------------------------------------------------
   //создание случайного графа
   void TForm1::FillWithRandomSymmetricValues()
   {
      int i,j;
      randomize();
      Form1->CityNamesMemo->Lines->Clear();
      for (i=0;i<N;i++)
      {
        Form1->CityNamesMemo->Lines->Add((AnsiString)(i+1));
        for (j=i;j<N;j++)
        {
             if (i==j)
             {
                  G[i][j]=0;
                  continue;
             }
             G[i][j]=random(100);
             G[j][i]=G[i][j];
        }
      }
      ShowInGrid(G,N);
      TMarkedGraph Graphpro(N,G);
      S = Graphpro.MarkMaxHamiltonianCycleCE(false);


   }
   //---------------------------------------------------------------------------
   //запись графа в таблицу
   void TForm1::ShowInGrid(double **G, int N)
   {
```

```
      int i,j;
      for(i=1;i<N+1;i++)
      {
            if (Form1->CityNamesMemo->Lines->Count < (int) i)
            {
                  Form1->GraphWeightsGrid->Cols[0]->Strings[i]=i;
                  Form1->GraphWeightsGrid->Rows[0]->Strings[i]=i;
            }
            else
            {
                  Form1->GraphWeightsGrid->Cols[0]->Strings[i]=Form1-
>CityNamesMemo->Lines->Strings[i-1];
                  Form1->GraphWeightsGrid->Rows[0]->Strings[i]=Form1-
>CityNamesMemo->Lines->Strings[i-1];
            }
      }
      for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
            {
                  if (i == j)
                  {
                        Form1->GraphWeightsGrid->Cols[i+1]-
>Strings[j+1]=(AnsiString) "";
                        continue;
                  }
            Form1->GraphWeightsGrid->Cols[i+1]-
>Strings[j+1]=FillWithSpacesLeft(G[i][j], MaxWeightDigit);
                                                Form1->GraphWeightsGrid->Cols[j+1]-
>Strings[i+1]=FillWithSpacesLeft(G[i][j], MaxWeightDigit);
            }
   }
   //---------------------------------------------------------------------------
   AnsiString TForm1::FillWithSpacesLeft(double Num, int Digits)
   {
      AnsiString Result = "";
      for (int i=1;i<Digits;i++)
      {
            int Stepen=1;
            for (int j=1;j<=i;j++)
            {
                  Stepen=Stepen*10;
            }
            if (Num<Stepen)
                  Result=Result+" ";
      }
      Result=Result+((AnsiString)Num).ToDouble();
      return Result;
   }
   //---------------------------------------------------------------------------
   void TForm1::CheckNumericKeyPressing(char &Key)
   {
      // Backspace
      if (Key == 8)
            return;
      if (NumericSet.Contains(Key)==false)
      {
            Key=0;
            return;
      }
   }
```

```
//---------------------------------------------------------------------------
int TForm1::SerdyukovAlg(double **G, int N)
{
   int Summa=0;
   for (int i=0;i<N;i++)
   {
         double max=0;
         for (int j=i+1;j<N;j++)
         {
               if (G[i][j]>max) {
                     max=G[i][j];
               }
         }
         Summa=Summa+max;
   }
   return Summa;
}
//---------------------------------------------------------------------------
int TForm1::Alg13_1(double **G, int N)
{
   int Summa=0;
   for (int i=0;i<N;i++)
   {
         double max=0;
         for (int j=i+1;j<N;j++)
         {
               if (G[j][i]>max) {
                     max=G[j][i];
               }
         }
         Summa=Summa+max;
   }
   return Summa;
}
//---------------------------------------------------------------------------
int TForm1::MethodVG(double **G, int N)
{
   double **GG;
   GG=new double*[N];
   for (int j=0;j<N;j++)
         GG[j]=new double[N];
   double Sum=0;
   for (int i=0;i<N;i++)
   {
         for (int j=0;j<N;j++)
         {
               GG[i][j]=G[i][j];
         }
   }
   for (int i=0;i<N;i++)
   {
         GG[i][i]=-1;
   }
   double *di,*dj;
   di=new double[N];
   dj=new double[N];
   for (int i=0;i<N;i++)
   {
         double Maxi=0;
         for (int j=0;j<N;j++)
```

```
        {
              if (GG[i][j]>Maxi && GG[i][j]!=-1)
              {
                    Maxi=GG[i][j];
              }
        }
        di[i]=Maxi;
   }
   for (int i=0;i<N;i++)
   {
        for (int j=0;j<N;j++)
        {
              if (GG[i][j]!=-1)
              {
                    GG[i][j]=di[i]-GG[i][j];
              }
        }
   }
    for (int i=0;i<N;i++)
   {
        double Minj=100000;
        for (int j=0;j<N;j++)
        {
              if (GG[j][i]<Minj && GG[j][i]!=-1)
              {
                    Minj=GG[j][i];
              }
        }
        dj[i]=Minj;
   }
   for (int i=0;i<N;i++)
   {
        for (int j=0;j<N;j++)
        {
              if (GG[i][j]!=-1)
              {
                    GG[i][j]=GG[i][j]-dj[j];
              }
        }
   }
   for (int i=0;i<N;i++)
   {
        for (int j=0;j<N;j++)
        {
              if (GG[i][j]==0) {
                    Sum=Sum+G[i][j];
              }
        }
   }
   return Sum;

}
//---------------------------------------------------------------------------

void __fastcall TForm1::GraphWeightsGridDblClick(TObject *Sender)
{

   if (GraphWeightsGrid->Col==GraphWeightsGrid->Row)
        return;
   if ((GraphWeightsGrid->Col>0)&&(GraphWeightsGrid->Row>0))
```

```
      {
            HiddenEdit->Text=GraphWeightsGrid->Cells[GraphWeightsGrid->Col]
[GraphWeightsGrid->Row];
            HiddenEdit->Left=GraphWeightsGrid->Left+(GraphWeightsGrid->Col-
GraphWeightsGrid->LeftCol+1.5)*GraphWeightsGrid->DefaultColWidth;
            HiddenEdit->Top=          GraphWeightsGrid->Top+(GraphWeightsGrid->Row-
GraphWeightsGrid->TopRow+1.5)*GraphWeightsGrid->DefaultRowHeight;
            HiddenEdit->Visible=true;
            ActiveControl=HiddenEdit;
            HiddenEdit->SetFocus();
      }
   }
   //---------------------------------------------------------------------------

   void __fastcall TForm1::HiddenEdit1Exit(TObject *Sender)
   {

      AnsiString Text = HiddenEdit->Text;
      char c;
      for (int i = 1; i <= HiddenEdit->Text.Length(); i++)
      {
            c = Text[i];
            if (c == ' ') Text[i] = '0';
      }
      HiddenEdit->Text = Text;
      if     ((GraphWeightsGrid->Row     ==     GraphWeightsGrid->Col)&&(HiddenEdit-
>Text.ToDouble() != 0))
      {
            Application->MessageBox("Матрица не должна содержать петли!","Ошибка",
MB_OK);
            HiddenEdit->Text = "00";
            ActiveControl = HiddenEdit;
            HiddenEdit->SetFocus();
            return;
      }
      GraphWeightsGrid->Cols[GraphWeightsGrid->Row]->Strings[GraphWeightsGrid-
>Col]=FillWithSpacesLeft(HiddenEdit->Text.ToDouble(), MaxWeightDigit);
      GraphWeightsGrid->Cols[GraphWeightsGrid->Col]->Strings[GraphWeightsGrid-
>Row]=FillWithSpacesLeft(HiddenEdit->Text.ToDouble(), MaxWeightDigit);
      G[GraphWeightsGrid->Row-1][GraphWeightsGrid->Col-1]=HiddenEdit-
>Text.ToDouble();
      G[GraphWeightsGrid->Col-1][GraphWeightsGrid->Row-1]=HiddenEdit-
>Text.ToDouble();
      HiddenEdit->Visible=false;

   }
   //---------------------------------------------------------------------------
   void __fastcall TForm1::HiddenEdit1KeyPress(TObject *Sender, char &Key)
   {
      // Enter
      if (Key == 13)
      {
            Form1->ActiveControl=GraphWeightsGrid;
            Key = 0;
            return;
      }
   }
   //---------------------------------------------------------------------------
   void __fastcall TForm1::AlgClick(TObject *Sender)
```

```
{
    Form2->Show();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::StaticText1Click(TObject *Sender)
{
   OutputMemo->Lines->SaveToFile("save.txt");
}
```