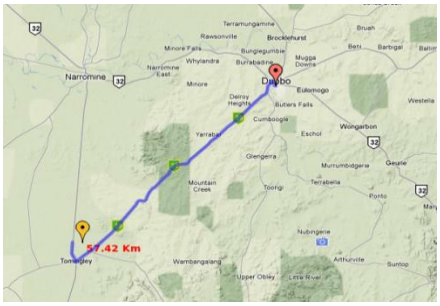
TRƯỜNG CAO ĐẲNG KỸ THUẬT LÝ TỰ TRỌNG

KHOA CÔNG NGHỆ THÔNG TIN

MÔN : LÝ THUYẾT ĐỒ THỊ

ĐỀ TÀI:

TÌM ĐƯỜNG ĐI BẰNG THUẬT TOÁN BFS



Giáo viên hướng dẫn : Vũ Thị Thái Linh

Sinh viên thực hiên :

- **Dương Bá Vận (NT)**
- **Nguyễn Hoài Nam**
- **Lê Trọng Thành**

Lớp : 11CD-TM1

Mục lục

1. Cơ sở lý thuyết và giải thuật

1.1. Thuật toán BFS (Breadth-first-search)

1.1.1. Mở đầu

Trong lý thuyết đồ thị, tìm kiếm theo chiều rộng (BFS) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) thăm một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa thăm vào danh sách có thể thăm trong tương lai. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt thăm các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt thăm các đỉnh kề với nó mà chưa được thăm trước đó và lặp lại. Thuật toán này thực ra là sự cải tiến về thứ tự duyệt đỉnh trên đồ thị của tìm kiếm theo chiều sâu bằng cách thay vì dùng một stack thì ta lại dùng một hàng đợi queue để kết nạp đỉnh được thăm. Như vậy, đỉnh được thăm càng sớm sẽ càng sớm trở thành duyệt xong (cơ chế First In First Out - Vào trước ra trước).

1.1.2. Thuật toán

Thuật toán sử dụng một cấu trúc dữ liệu hàng đợi để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm:

1. Chèn đỉnh gốc vào hàng đợi
2. Lấy ra đỉnh v đầu tiên trong hàng đợi và thăm nó
3. Chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được thăm trước đó vào hàng đợi.
4. Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được thăm – dừng việc tìm kiếm.
5. Nếu hàng đợi không rỗng thì quay về bước 2.

Mã giả:

```
procedure BFS(Graph, source):
    tạo hàng đợi Q
    chèn source vào hàng đợi Q
    đánh dấu source
    while Q không rỗng:
        lấy một đỉnh từ Q gán vào v
        for mỗi cạnh e trên v in Graph:
            gắn w cho đỉnh đầu kia của cạnh e
            if w chưa được thăm:
                đánh dấu w
                chèn w vào
```

1.1.3. Các đặc tính của thuật toán

- Không gian lưu trữ:
  - Chi phí lưu trữ bằng ma trận kề luôn luôn cố định là $O(n^2)$.
  - Chi phí lưu trữ bằng danh sách kề bằng O(|V|+|E|) với |V| là tổng số đỉnh và |E| là tổng số cạnh.
- Độ phức tạp thuật toán: Thời gian thực hiện thuật toán cũng được thể hiện O(|V| + |E|) với |V| là tổng số đỉnh và |E| là tổng số cạnh. Vì mỗi đỉnh và mỗi cạnh đều được duyệt.

1.2. Tìm đường đi từ hai điểm bất kỳ bằng giải thuật BFS

1.2.1. Ý tưởng thuật toán

Cho đồ thị G=(V,E). Với hai đỉnh s và t là hai đỉnh nào đó của đồ thị. Hãy tìm đường đi từ s đến t.

Do thủ tục BFS(s) sẽ thăm lần lượt các đỉnh liên thông với u nên sau khi thực hiện xong thủ tục thì có hai khả năng :

-Nếu visited[t]=True thì có nghĩa: tồn tại một đường đi từ đỉnh s tới đỉnh t.

-Ngược lại, thì không có đường đi nối giữa s và t.

Vấn đề còn lại của bài toán là: *Nếu tồn tại đường đi nối đỉnh s và đỉnh t thì làm cách nào để viết được hành trình (gồm thứ tự các đỉnh) từ s đến t.*

Để giải quyết vấn đề trên ta dùng một mảng prev với prev[w] = v, w là các đỉnh kề với đỉnh v được lấy ra từ hàng đợi.

1.2.2. Mã giải

```
procedure FindPath(s):
      for each vertex v:
            do visited[v]:=false;
                prev[v] = -1;
      Q = empty queue;
      visited [s]:=true;
      enqueue(Q,s);
      while Q is not empty:
            do v:= dequeue(Q);
                  for each w adjacent to v:
                        do if visited [w] = false
                              then visited [w] := true;
                              prev[w] := v;
                              enqueue(Q,w);
```

Việc viết lại đường đi từ mảng prev có thể thực hiện như sau:

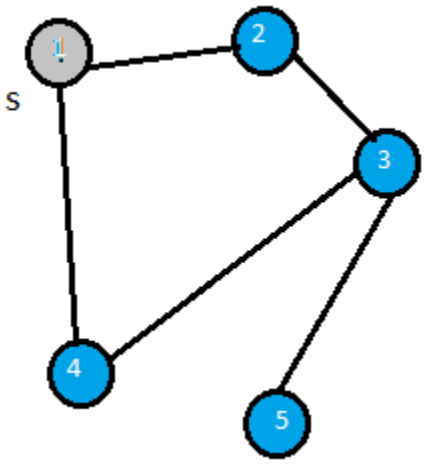
+ Dùng mảng phụ để lưu

```
procedure get_Path():
      CurrentV = t
      while prev[CurrentV]!=-1):
             do tmp[index++] = CurrentV;
                 CurrentV =  prev[CurrentV];
      Reverse(tmp)
```

+ Dùng đệ qui để viết đường đi

```
procedure print_path(CurrentV):
  if prev[CurrentV] != -1:
      print_path(prev[CurrentV]);
                 print prev[CurrentV];
```

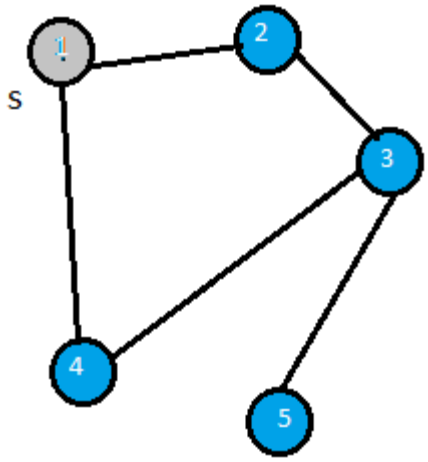
1.2.3. Ví dụ



Danh sách kề	
1	2 , 4
2	1 , 3
3	2, 4 , 5
4	1 , 3
5	3

	Visited	Prev
1	F	-1
2	F	-1
3	F	-1
4	F	-1
5	F	-1

+ Khởi tạo:
  - Q = {}
  - Bảng visited tất cả bằng false
  - Bảng prev tất cả bằng -1

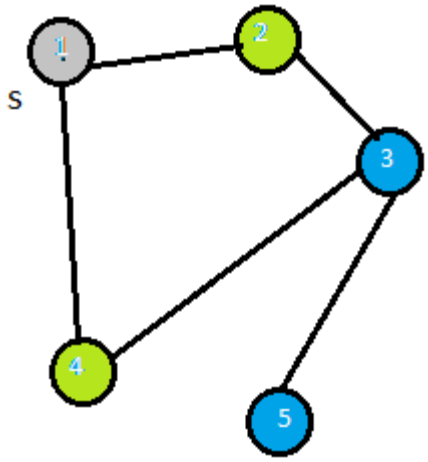


	Visited	Prev
1	T	-1
2	F	-1
3	F	-1
4	F	-1
5	F	-1

- Chèn s = 1 vào Q:

  Q = {1}

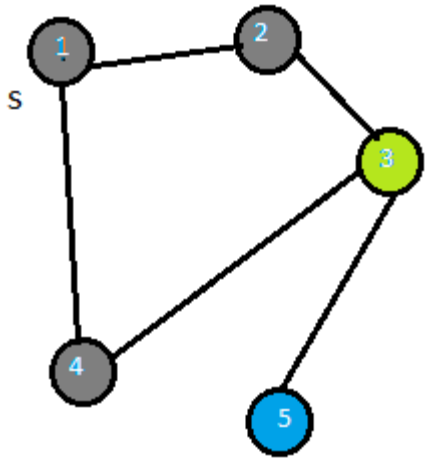
- Bật visited[1] = true



	Visited	Prev
1	T	-1
2	T	1
3	F	-1
4	T	1
5	F	-1

Q = {1} → {2,4}

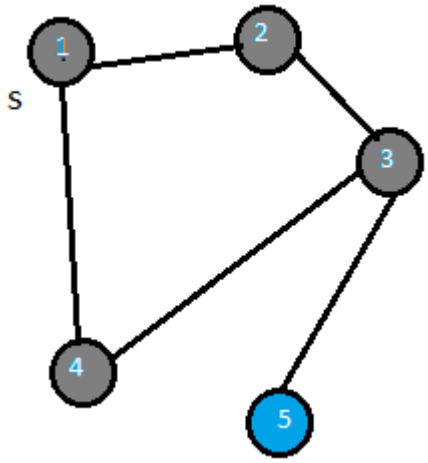
Lấy v = 1 ra khỏi Q. Tiến hành thăm các đỉnh kề với v. Đánh dấu các đỉnh đã thăm đồng thời lưu các đỉnh đứng sau v và chèn vào Q.



	Visited	Prev
1	T	-1
2	T	1
3	T	2
4	T	1
5	F	-1

Q = {2,4} → {4,3}

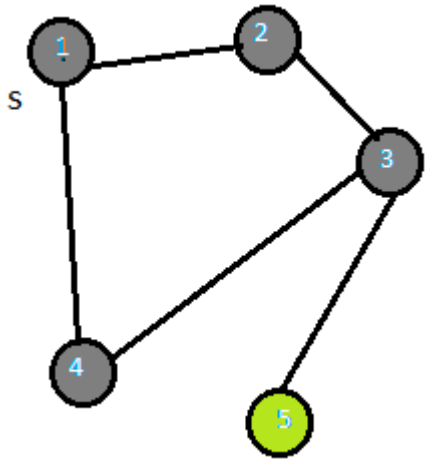
Lấy v = 2 ra khỏi Q. Tiến hành thăm các đỉnh kề với v. Đánh dấu các đỉnh đã thăm đồng thời lưu các đỉnh đứng sau v và chèn vào Q.



	Visited	Prev
1	T	-1
2	T	1
3	T	2
4	T	1
5	F	-1

Q = {4,3} → {3}

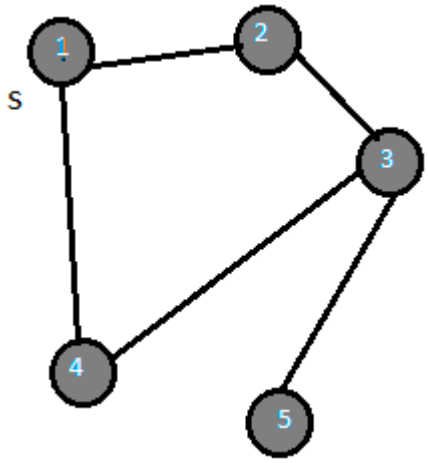
Lấy v = 4 ra khỏi Q. Tiến hành thăm các đỉnh kề với v. Đánh dấu các đỉnh đã thăm đồng thời lưu các đỉnh đứng sau v và chèn vào Q.



	Visited	Prev
1	T	-1
2	T	1
3	T	2
4	T	1
5	T	3

Q = {3} → {5}

Lấy v = 3 ra khỏi Q. Tiến hành thăm các đỉnh kề với v. Đánh dấu các đỉnh đã thăm đồng thời lưu các đỉnh đứng sau v và chèn vào Q.



	Visited	Prev
1	T	-1
2	T	1
3	T	2
4	T	1
5	T	3

Q = {5} → {}

Lấy v = 3 ra khỏi Q. Tiến hành thăm các đỉnh kề với v. Đánh dấu các đỉnh đã thăm đồng thời lưu các đỉnh đứng sau v và chèn vào Q. Q rỗng kết thúc thuật toán.

Bây giờ từ bảng prev ta có thể tìm đường đi từ 1 đến một đỉnh bất kì trong đồ thị.

Ví dụ : Path(1,2) => 1 → 2

Path(1,5) => 1 → 2 → 3 → 5

Để tìm đường đi từ s đến t ta chỉ cần kiểm tra điểm t đã được thăm hay chưa. Nếu được thăm rồi thì kết thúc thuật toán mà không cần phải duyệt cho đến khi Q rỗng.

1.2.4. Cài đặt

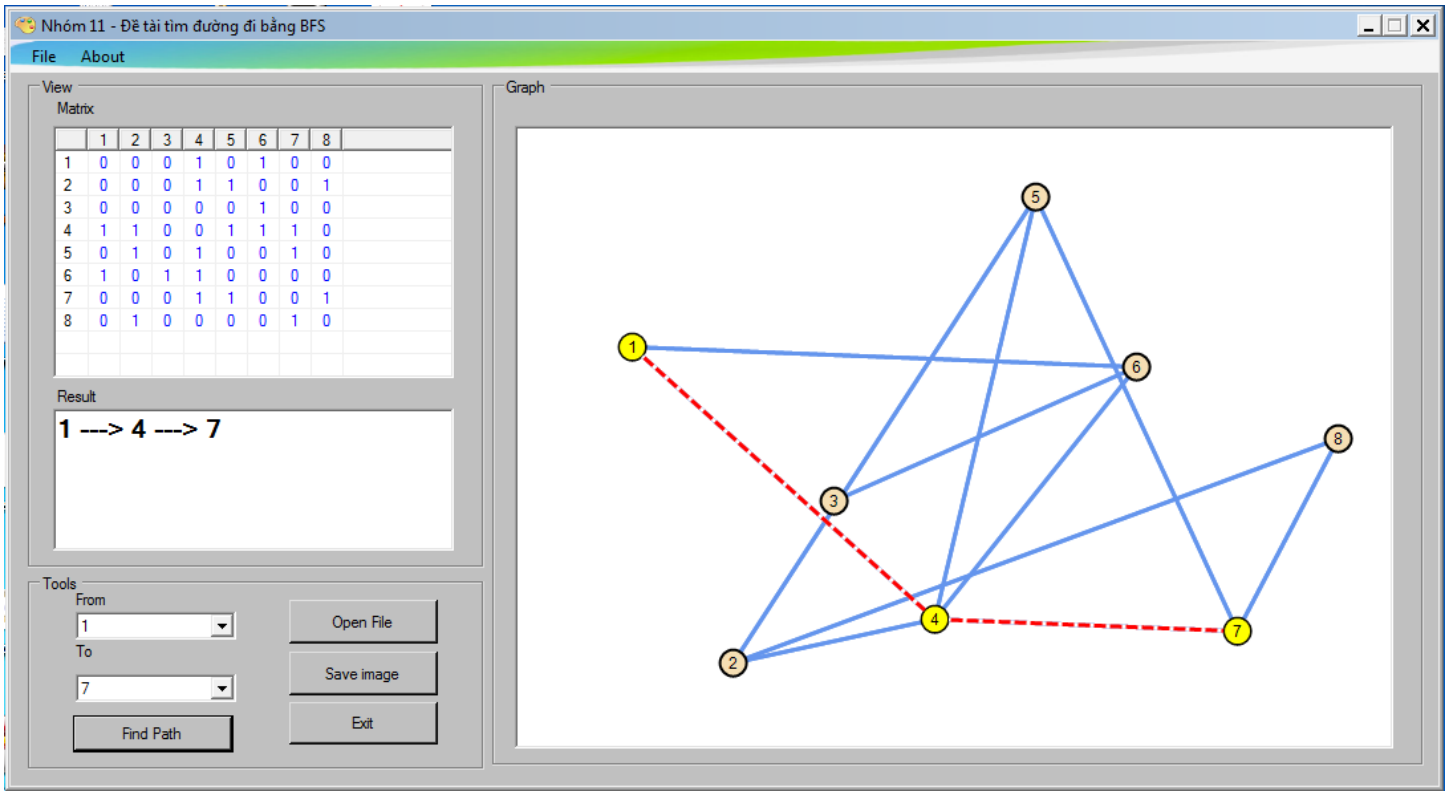
```
class BFS
{
    private Queue<int> VertexQueue = new Queue<int>(); // hang doi chua cac dinh
    private List<List<int>> _adjacent; // danh sach canh ke
    private List<int> _reportPath = new List<int>(); // danh sach dinh duong di
    public BFS(List<List<int>> adjacent)
    {
        this._adjacent = adjacent;

    }
    //find path
    public List<int> findPathbyBfs(int tips, int start, int end)
    {
        if (this._adjacent[start] == null || this._adjacent[end] == null)
            return null;
        bool[] visited = new bool[tips]; // danh dau cac dinh da tham
        int[] previous = new int[tips + 1]; // luu dinh truoc
        // khoi tao mang visited va previous
        for (int index = 0; index < tips; ++index)
        {
            visited[index] = false;
            previous[index] = -1;
        }
        this.VertexQueue.Enqueue(start);
        visited[start] = true;
        while (this.VertexQueue.Count != 0)
        {
            int v = this.VertexQueue.Dequeue();

            List<int> row = new List<int>(this._adjacent[v]);
            if (row != null)
            {
                foreach (int col in row)
                {
                    if (!visited[col])
                    {
                        this.VertexQueue.Enqueue(col);
                        previous[col] = v;
                        visited[col] = true;
                        // neu diem ket thu duoc tham thi ket thuc thuat toan
                        if (visited[end]) break;
                    }
                }
            }
            // diem cuoi duoc tham thi ket thuc thuat toan
            if (visited[end]) break;
        }
        // neu diem ket thuc khong duoc tham thi tra ve null
        if (!visited[end]) return null;
              // truy vet duong di
        int current = end;
        this._reportPath.Add(end);
        while (previous[current] != -1)
        {
            this._reportPath.Add(previous[current]);
            current = previous[current];
        }
        this._reportPath.Reverse();
        return this._reportPath;
    }
```

```
} // end find path
```

2. Mô tả thuật toán trên c#



2.1. Class frmMain

2.1.1. Variable

```
List<List<int>> Adjacent;
int Vertices; // so dind
GraphicsTools g; // khai bao 1 graphicsTools
List<Point> lstPointVertices; // danh sach toa do cac dinh
string FileName;
public const int PicturePadding = 50; // picture padding
```

2.1.2. Method

- `public frmMain()`
- `private void aboutToolStripMenuItem_Click(object sender, EventArgs e)`
- `private void btnExit_Click(object sender, EventArgs e)`
- `private void btnFindPath_Click(object sender, EventArgs e)`
- `private void btnOpenFile_Click(object sender, EventArgs e)`
- `private void openFileToolStripMenuItem_Click(object sender, EventArgs e)`
- `private void saveImageToolStripMenuItem_Click(object sender, EventArgs e)`
- `private void frmMain_Load(object sender, EventArgs e)`
- `private void exitToolStripMenuItem_Click(object sender, EventArgs e)`

2.1.3. Toàn văn class frmMain

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Nhom_11_DuongBaVan_11CDTM1
{
    public partial class frmMain : Form
    {
        List<List<int>> Adjacent;
        int Vertices; // so dinh

        GraphicsTools g; // khai bao 1 graphicsTools
        List<Point> lstPointVertices; // danh sach toa do cac dinh
        string FileName;
        public const int PicturePadding = 50; // picture padding

        public frmMain()
        {
            InitializeComponent();
        }

        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {

            About frmAbout = new About();
            frmAbout.StartPosition = FormStartPosition.CenterScreen;
            frmAbout.Show(this);
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
```

```
        private void btnFindPath_Click(object sender, EventArgs e)
        {
            BFS bfs = new BFS(this.Adjacent);

            int start = this.cboFrom.SelectedIndex;
            int end = this.cboTo.SelectedIndex;

            // reset picGraphics va txtResult
            this.picGraphics.Image = this.g.Reset(this.Adjacent,
this.lstPointVertices);
            this.txtResult.Clear();

            if (start == end)
            {
                MessageBox.Show("Vertices are duplicate. Please choose again!",
"Error vertices Selected");
                return;
            }
            List<int> res = bfs.findPathbyBfs(this.Vertices, start, end);

            if (res == null)
            {
        string text = "Can't find any path from {0} to {1}.";
        MessageBox.Show(string.Format(text, start + 1, end + 1),"Find Path");
        return;
    }
    else
    {

        int index;
        this.txtResult.Text = "";

        // reset bit map

        // xuat ket qua ra text box
        for (index = 0; index < res.Count - 1; ++index)
            this.txtResult.Text += (1 + res[index]).ToString() + " ---> ";
        this.txtResult.Text += (1 + res[index]).ToString();

        // ve duong di len bitmap
        this.picGraphics.Image = this.g.DrawPath(res, this.lstPointVertices);
    }
}
```

```
        private void btnOpenFile_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "Text Document File(*.txt)|*.txt|All File(*.*)|*.*";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    // enable button findpath va button save image
                    this.btnFindPath.Enabled = true;
                    this.btnSaveImage.Enabled = true;

                    // reset control values
                    this.cboFrom.Items.Clear();
                    this.cboTo.Items.Clear();
                    this.lvMatrix.Items.Clear();
                    this.lvMatrix.Columns.Clear();
                    this.txtResult.Clear();
                    this.picGraphics.Image = null;
                    //
                    Matrix m = new Matrix();
                    //
                    this.FileName = ofd.FileName;
                    this.Adjacent = m.LoadFile(this.FileName, this.lvMatrix,
this.cboFrom, this.cboTo);
                    this.cboFrom.SelectedIndex = 0;
                    this.cboTo.SelectedIndex = 1;

                    this.Vertices = m.Vertices;

                    this.g = new GraphicsTools(this.picGraphics.Width,
this.picGraphics.Height);// khoi tao graphics tool



                    // lay danh sach toa do cac dinh
                    this.lstPointVertices = new Vector2D(this.Vertices,
this.picGraphics.Width - frmMain.PicturePadding,
      this.picGraphics.Height - frmMain.PicturePadding).getRandomPoint();

                    // tao bitmap do thi voi danh sach ke va danh sach toa cac dinh

                    this.picGraphics.Image = this.g.CreateGraphics(this.Adjacent,
this.lstPointVertices);

                }
                catch (Exception EX)
                {
                    MessageBox.Show(EX.Message, "Error");

                }
            }
            ofd.Dispose();
        }
```

```
        private void btnSaveImage_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveImgDialog = new SaveFileDialog();
            saveImgDialog.DefaultExt = "png";
            saveImgDialog.Filter = "Bitmap Image (*.png)|*.png|All File (*.*)|*.*";
            saveImgDialog.AddExtension = true;
            saveImgDialog.RestoreDirectory = true;
            saveImgDialog.Title = "Save graph to image";
            string initFileName =
System.IO.Path.GetFileNameWithoutExtension(this.FileName);
            saveImgDialog.FileName = initFileName;
            try
            {
                if (saveImgDialog.ShowDialog() == DialogResult.OK)
                {
                    string imgPath = saveImgDialog.FileName;

                    if (imgPath != null)
                    {
                        if (this.picGraphics.Image != null)
                        {
                            this.picGraphics.Image.Save(imgPath,
System.Drawing.Imaging.ImageFormat.Png);
                        }
                    }
                }
            }
            catch (Exception)
            {

                throw;
            }
            saveImgDialog.Dispose();
        }

        private void openFileToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.btnOpenFile_Click(sender, e);
        }


        private void saveImageToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.btnSaveImage_Click(sender, e);
        }

        private void frmMain_Load(object sender, EventArgs e)
        {
            // disable button findpath va button save image
            this.btnFindPath.Enabled = false;
            this.btnSaveImage.Enabled = false;
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

2.2. Class BFS

2.2.1. Variable

```
private Queue<int> VertexQueue = new Queue<int>(); // hang doi chua cac
dinh
private List<List<int>> _adjacent; // danh sach  ke
private List<int> _reportPath = new List<int>(); //luu vet duong di
```

2.2.2. Method

- `public BFS(List<List<int>> adjacent)`
- `public List<int> findPathbyBfs(int tips, int start, int end)`

2.2.3. Toàn văn class BFS

```
class BFS
{
    private Queue<int> VertexQueue = new Queue<int>(); // hang doi chua cac dinh
    private List<List<int>> _adjacent; // danh sach  ke
    private List<int> _reportPath = new List<int>(); // danh sach dinh duong di

    public BFS(List<List<int>> adjacent)
    {
        this._adjacent = adjacent;

    }
    //find path
    public List<int> findPathbyBfs(int tips, int start, int end)
    {
        if (this._adjacent[start] == null || this._adjacent[end] == null)
            return null;
        bool[] visited = new bool[tips]; // danh dau cac dinh da tham
        int[] previous = new int[tips + 1]; // luu dinh truoc
        // khoi tao mang visited va previous
        for (int index = 0; index < tips; ++index)
        {
            visited[index] = false;
            previous[index] = -1;
        }
        this.VertexQueue.Enqueue(start);
        visited[start] = true;
        while (this.VertexQueue.Count != 0)
        {
            int v = this.VertexQueue.Dequeue();
            List<int> row = new List<int>(this._adjacent[v]);
            if (row != null)
            {
                foreach (int col in row)
                {
                    if (!visited[col])
                    {
                        this.VertexQueue.Enqueue(col);
                        previous[col] = v;
                        visited[col] = true;
                        // neu diem ket thu duoc tham thi ket thuc thuat toan
                        if (visited[end]) break;
                    }
                }
            }
            // diem cuoi duoc tham thi ket thuc thuat toan
           if (visited[end]) break;

        }
        // neu diem ket thuc khong duoc tham thi tra ve null
        if (!visited[end]) return null;
        // truy vet duong di
        int current = end;
        this._reportPath.Add(end);
        while (previous[current] != -1)
        {
            this._reportPath.Add(previous[current]);
            current = previous[current];
        }
        this._reportPath.Reverse();
        return this._reportPath;
    }

} // end find path
```

```
}
```

2.3. Class GraphicsTools

2.3.1. Variable

```
private  Size sizeCircle = new Size(20,20); // size node
private Bitmap bmpGraphics;
private const int _RADIUS = 10; // ban kinh node
private const int _WIDTH = 20; // chieu rong node
private const int _HEIGHT = 20; // chieu cao node
```

2.3.2. Method

```
public GraphicsTools(int Width, int Height)

public int RADIUS

public int HEIGHT

public int WIDTH

private void DrawLine(Graphics g, Pen pLine, Point ptStart, Point ptEnd)

private void DrawNode(Graphics g, Brush bFillNode, Pen pEllipse,
Rectangle rect, Point pt, string name)

public Bitmap CreateGraphics(List<List<int>> adjacent, List<Point>
lstPointVertex)

public Bitmap DrawPath(List<int> ReportPath, List<Point>
lstPointVertices)

 public Bitmap Reset(List<List<int>> adjacent, List<Point>
lstPointVertices)
```

2.3.3. Toàn văn class GraphicsTools

```
class GraphicsTools
    {
        private  Size sizeCircle = new Size(20,20); // size node
        private Bitmap bmpGraphics;
        private const int _RADIUS = 10; // ban kinh node
        private const int _WIDTH = 20; // chieu rong node
        private const int _HEIGHT = 20; // chieu cao node



        public GraphicsTools(int Width, int Height)
        {
            this.bmpGraphics = new Bitmap(Width, Height);
        }

        public int RADIUS
        {
            get { return _RADIUS; }
        }
        public int HEIGHT
        {
            get { return _HEIGHT; }
        }

        public int WIDTH
        {
            get { return _WIDTH; }
        }

        // draw line
        private void DrawLine(Graphics g, Pen pLine, Point ptStart, Point ptEnd)
        {
            g.DrawLine(pLine, ptStart, ptEnd);
        }
        // draw node
        private void DrawNode(Graphics g, Brush bFillNode, Pen pEllipse, Rectangle
rect, Point pt, string name)
        {
            // ve hinh tron
            g.FillEllipse(bFillNode, rect);
            //ve duong tron
            g.DrawEllipse(pEllipse, rect);
            //font cho ten cua node
            Font stringFont = new Font("Arial", 9);

            // lay gia tri width va height cua ten node
            SizeF stringSize = g.MeasureString(name, stringFont);

            // ve ten o chinh giua node
            g.DrawString(name, stringFont, Brushes.Black,
                        pt.X + (this.WIDTH  -  stringSize.Width) /2,
                        pt.Y + (this.HEIGHT -  stringSize.Height)/2);
        }// #End
```

```
// tao 1 bitmap do thi
public Bitmap CreateGraphics(List<List<int>> adjacent, List<Point> lstPointVertex)
{
    Graphics g = Graphics.FromImage(this.bmpGraphics);
    g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

    g.Clear(Color.White);
    Pen pLine = new Pen(Color.FromArgb(100, 149, 237    ), 3);
    Brush cbrush = new SolidBrush(Color.Wheat);
    Pen pCircle = new Pen(Color.Black,2);

    // lam min bitmap

    Point ptStart, ptEnd;

    // ve nhung duong thang noi cac dinh lai
    for (int index = 0; index < adjacent.Count; ++index)
    {

        List<int> row = new List<int>(adjacent[index]);
        // diem dau cua duong thang
        ptStart = new Point(lstPointVertex[index].X,
                            lstPointVertex[index].Y);

        foreach (int col in row)
        {
            // diem cuoi cua duong thang
            ptEnd = new Point(lstPointVertex[col].X,
                              lstPointVertex[col].Y);
            // tien hang ve duong thang noi cac noi ke lai voi nhau
            this.DrawLine(g, pLine, ptStart, ptEnd);
        }
    }

    // ve node
    for (int index = 0; index < adjacent.Count; ++index)
    {
        // toa do ve duong tron
        Point pt = new Point(lstPointVertex[index].X - this.RADIUS,
                             lstPointVertex[index].Y - this.RADIUS);
        // tao rect cho node
        Rectangle rect = new Rectangle(pt,this.sizeCircle);
        this.DrawNode(g, cbrush, pCircle, rect, pt, (index + 1).ToString());
    }
    g.Dispose();
    pLine.Dispose();
    cbrush.Dispose();
    pCircle.Dispose();
    return this.bmpGraphics;
}// #end create graphics
```

```
        // draw path
         public Bitmap DrawPath(List<int> ReportPath, List<Point> lstPointVertices)
         {
             Graphics g = Graphics.FromImage(this.bmpGraphics);

             g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

             Pen pLine = new Pen(Color.Red, 3); // pen for line
             Pen pClear = new Pen(Color.White, 3); // pen clear
             pLine.DashStyle = DashStyle.Dash; // dash type for pen line
             Brush cbrush = new SolidBrush(Color.Yellow); // fill ecllipse

             Pen pCircle = new Pen(Color.Black); // pen for circle

             // hightlight duong di
             for (int index = 0 ; index < ReportPath.Count-1; ++index)
             {
                 this.DrawLine(g, pClear, lstPointVertices[ReportPath[index]],
lstPointVertices[ReportPath[index + 1]]); // clear line
                 this.DrawLine(g, pLine,  lstPointVertices[ReportPath[index]],
lstPointVertices[ReportPath[index + 1]]);
             }

             // hightlight node nam tren duong di
             for (int index = 0; index < ReportPath.Count; ++index)
             {
                 Point ptCircle = new Point
                           (lstPointVertices[ReportPath[index]].X - this.RADIUS,
                           lstPointVertices[ReportPath[index]].Y - this.RADIUS);
                 Rectangle rect = new Rectangle(ptCircle, this.sizeCircle);

                  this.DrawNode(g, cbrush, pCircle, rect, ptCircle,
               (ReportPath[index]+1).ToString());

             }

             g.Dispose();
             pLine.Dispose();
             cbrush.Dispose();
             pClear.Dispose();
             pCircle.Dispose();

             return this.bmpGraphics;
         }// # End draw path

         public Bitmap Reset(List<List<int>> adjacent, List<Point> lstPointVertices)
         {
             return this.CreateGraphics(adjacent, lstPointVertices);
         }
     } // #End class
```

2.4. Class Matrix

2.4.1. Variable

```
private int _vertices; // dinh
      private List<List<int>> _adjacent; // danh sach ke
```

2.4.2. Method

```
public List<List<int>> Adjacent

public int Vertices

private ColumnHeader AddColunmListView(string text)

public List<List<int>> LoadFile(string fileName, ListView lv, ComboBox
cboFrom, ComboBox cboTo)
```

2.4.3 Toàn văn class matrix

```
class Matrix
 {
     private int _vertices; // dinh
     private List<List<int>> _adjacent; // danh sach ke

     public List<List<int>> Adjacent
     {
         get { return _adjacent; }
         set { _adjacent = value; }
     }

     public int Vertices
     {
         get { return _vertices; }
         set { _vertices = value; }
     }

     // them columnheader cho list view
     private ColumnHeader AddColunmListView(string text)
     {
         ColumnHeader headerCol = new ColumnHeader();
         headerCol.Text = text;
         headerCol.Width = 24;
         headerCol.TextAlign = HorizontalAlignment.Center;
         headerCol.AutoResize(ColumnHeaderAutoResizeStyle.HeaderSize);
         return headerCol;
     }
```

```
        public List<List<int>> LoadFile(string fileName, ListView lv, ComboBox
cboFrom, ComboBox cboTo)
        {
            StreamReader fin = new StreamReader(fileName);

            // danh sach can ke
            this._adjacent = new List<List<int>>();
            this.Vertices = Convert.ToInt32(fin.ReadLine());
            string line = "";
            List<int> row;

            lv.Columns.Add(AddColunmListView(""));

            for (int iX = 0; iX < this.Vertices; ++iX)
            {
                line = fin.ReadLine();
                string []words = line.Split(' ');

                cboFrom.Items.Add((iX+1).ToString()); // them item vao
comboBox From
                cboTo.Items.Add((iX + 1).ToString()); // them item vao
comboBox To

                lv.Columns.Add(AddColunmListView((iX+1).ToString()));

                // them dong vao list view
                ListViewItem lvi = new ListViewItem((iX+1).ToString());
                lvi.UseItemStyleForSubItems = false;

                row = new List<int>();
                for (int iY = 0; iY < this.Vertices; ++iY)
                {
                    // list view sub item
                    ListViewItem.ListViewSubItem col =
lvi.SubItems.Add(words[iY]);
                    col.ForeColor = Color.Blue; // mau cua list view subitem
                    if (words[iY] != "0")
                        row.Add(iY);
                }
                this._adjacent.Add(row);
                lv.Items.Add(lvi);
            }
            fin.Close();
            return this._adjacent;
        }
    }
```

2.5. Class Vector2D

2.5.1. Variable

```
private int _vertices;
private int _width;
private int _height;
```

2.5.2. Method

```
public Vector2D(int vertices, int width, int height)

public List<Point> getRandomPoint()
```

2.5.3. Toàn văn class Vector2D

```
class Vector2D
{
    private int _vertices;
    private int _width;
    private int _height;
    public Vector2D(int vertices, int width, int height)
    {
        this._vertices = vertices;
        this._width = width;
        this._height = height;
    }

    public List<Point> getRandomPoint()
    {
        List<Point> lstPoint = new List<Point>();
        Random ran = new Random();
        int distance = (int)(this._width / this._vertices);
        int x = 10, y = 0;
        for (int index = 1; index <= this._vertices; ++index)
        {
            x += distance;
            y = ran.Next(50, this._height);
            lstPoint.Add(new Point(x, y));
        }

            return lstPoint;
    }
}
```

3. Kết luận

- Nếu có đường đi từ s đến t, thì đường đi tìm được do thuật toán tìm kiếm theo chiều rộng cho chúng ta một hành trình cực tiểu về số cạnh.

- Độ phức tạp tính toán của thuật toán BFS là O(V+E). Từ đó cũng suy ra thời gian tính toán của BFS tỷ lệ tuyến tính với kích thước của danh sách kề của đồ thị G.

Tham khảo

[1]. *GDI+ Custom Controls With Visual CSharp 2005 (2006).pdf*

[2]. *Giáo trình lý thuyết đồ thị (Trường cao đẳng Lý Tự Trọng)*

[3]. *http://msdn.microsoft.com/en-us/vstudio/hh341490.aspx*

[4].*http://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_r%E1%BB%99ng*

Task

Thành viên	Mô tả công việc	Kết quả thực hiện
Lê Trọng Thành	1.2. Tìm đường đi từ hai điểm bất kỳ bằng giải thuật BFS 1.2.1. Ý tưởng 1.2.3. Ví dụ	Đã hoàn thành
Nguyễn Hoài Nam	1.1. Thuật toán BFS 1.2.3. Các đặc tính của thuật toán	Đã hoàn thành
Dương Bá Vận	1.1.2. Thuật toán 1.2.2. Mã giả 1.2.4. Cài đặt 2. Mô tả chương trình trên c# 3. Kết luận	Đã hoàn thành
Nguyễn Kim Phương	Tìm kiếm tài liệu, viết báo cáo. 1.1.1. Mở đầu	Đã hoàn thành