

# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

## LAB 6 – MỘT SỐ BÀI TOÁN TRÊN ĐỒ THỊ (4 TIẾT)

### I. Mục tiêu

Sau khi thực hành, sinh viên cần:

- Nắm vững định nghĩa đồ thị, các khái niệm liên quan đến đồ thị.
- Cài đặt được kiểu dữ liệu đồ thị và các thao tác, phép toán trên đồ thị được biểu diễn bởi ma trận kề.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

### II. Yêu cầu

- Sinh viên phải hoàn thành cả **3 bài** thuộc mục V. Mỗi bài tạo một project, xóa các thư mục debug của project này. Sau đó chép các project vào thư mục: Lab6\_CTK39\_HoTen\_MSSV\_Nhom#. Nén thư mục, đặt tên tập tin nén theo dạng sau: Lab6\_CTK39\_HoTen\_MSSV\_Nhom#.rar.

Ví dụ: Lab6\_CTK39\_NguyenVanA\_141111\_Nhom4.rar.

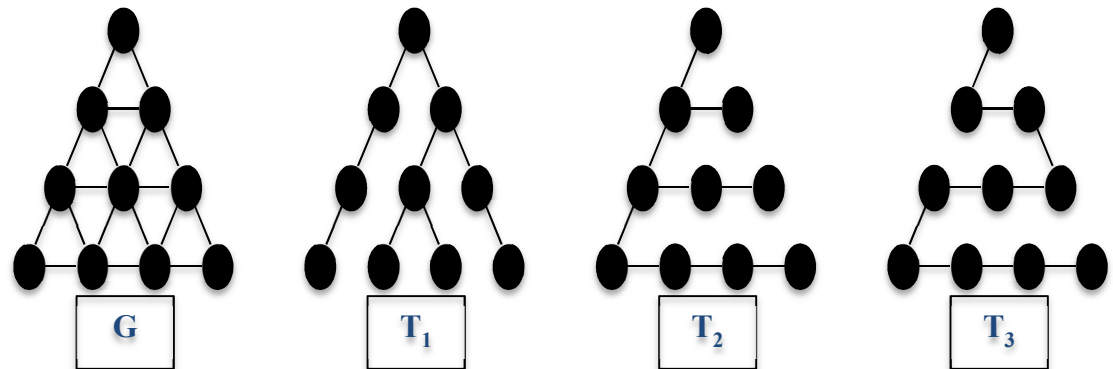
- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

### III. Ôn tập lý thuyết

#### 1. Bài toán tìm cây bao trùm tối thiểu

Phát biểu bài toán:

- Cho đồ thị vô hướng  $G=(V,E)$
- Tìm đồ thị  $T=(V,F)$  trong đó  $F$  là tập con của  $E$  sao cho:
  - (1)  $T$  liên thông
  - (2)  $T$  không có chu trình
  - (3) Tổng độ dài các cạnh trong  $T$  là nhỏ nhất
- Nếu  $T$  chỉ thỏa (1), (2) thì  $T$  gọi là cây bao trùm
- Nếu thỏa (1), (2), (3) thì  $T$  gọi là cây bao trùm tối thiểu.
- Một số tài liệu sử dụng thuật ngữ: Cây khung



Các thuật toán tìm cây bao trùm tối thiểu

Thuật toán Prim	Thuật toán Kruskal
<ul style="list-style-type: none"><li>▪ Input: <math>G = (V, E)</math></li><li>▪ Output: <math>T = (V, F)</math> nhỏ nhất</li></ul>	<ul style="list-style-type: none"><li>▪ Input: <math>G = (V, E)</math></li><li>▪ Output: <math>T = (V, F)</math> nhỏ nhất</li></ul>

<ul style="list-style-type: none"> <li>▪ U: Tập các đỉnh chưa được chọn (xét)</li> <li>▪ F: Tập các cạnh được chọn</li> <li>▪ Khởi tạo <math>U = \emptyset, F = \emptyset</math></li> <li>▪ Chọn một đỉnh v bất kỳ làm gốc của cây bao trùm</li> <li>▪ Đưa v vào U</li> <li>▪ Trong khi U khác V <ul style="list-style-type: none"> <li>▪ Chọn cạnh (u, v) nhỏ nhất sao cho <math>u \in U, v \in V - U</math></li> <li>▪ Thêm v vào U</li> <li>▪ Thêm (u, v) vào F</li> </ul> </li> <li>▪ Kết thúc: <math>T=(V, F)</math> là cây bao trùm tối thiểu</li> </ul>	<ul style="list-style-type: none"> <li>▪ Khởi tạo cây T không có cạnh nào (<math>F = \emptyset</math>), chỉ gồm n đỉnh</li> <li>▪ Sắp xếp các cạnh của G tăng dần theo trọng số</li> <li>▪ Lần lượt xét từng cạnh (u,v) từ trọng số nhỏ nhất đến lớn nhất</li> <li>▪ Thêm cạnh (u, v) vào F nếu không tạo thành chu trình</li> <li>▪ Lặp lại bước trên cho đến khi đủ n-1 cạnh hoặc mọi cạnh còn lại đều tạo thành chu trình.</li> <li>▪ Kết thúc: <math>T=(V, F)</math> là cây bao trùm tối thiểu</li> </ul>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 2. Bài toán tìm đường đi ngắn nhất

### ❖ Phát biểu bài toán

- Cho đồ thị  $G = (V, E)$
- Mỗi cạnh được gán một giá trị không âm, gọi là trọng số (hoặc giá, chi phí) của cạnh
- Cho trước một đỉnh  $s_0$ , gọi là đỉnh nguồn
- Vấn đề: Tìm đường đi ngắn nhất từ đỉnh  $s_0$  đến các đỉnh còn lại sao cho tổng chi phí trên đường đi là nhỏ nhất.

### ❖ Input

- $s_0$ : Đỉnh nguồn
- G: Đồ thị được biểu diễn bởi ma trận kề hoặc danh sách kề

### ❖ Output

- L: Mảng biểu diễn độ dài đường đi ngắn nhất từ  $s_0$  đến các đỉnh #
- T: Mảng lưu vết đường đi từ  $s_0$  đến các đỉnh còn lại

Thuật toán Dijkstra	Thuật toán Floyd
<ul style="list-style-type: none"> <li>❖ Tìm đường đi từ 1 đỉnh tới các đỉnh còn lại trong đồ thị</li> <li>❖ Ký hiệu: <ul style="list-style-type: none"> <li>▪ Giả sử G có n đỉnh, trọng số được lưu trong ma trận C</li> <li>▪ <math>C[i,j]</math> là chi phí của cung (i, j). <math>C[i,j] = \infty</math> nếu i không kề j</li> <li>▪ <math>L[w]</math>: lưu độ dài đường đi từ <math>s_0</math> đến đỉnh w</li> <li>▪ <math>T[w]</math>: lưu đỉnh trước w trên đường đi</li> <li>▪ <math>M[w]</math>: cho biết đỉnh w đã được xét hay chưa</li> <li>▪ L, T và M sẽ được cập nhật lại sau mỗi bước của thuật toán</li> <li>▪ S: là tập các đỉnh đã được chọn</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>❖ Tìm đường đi ngắn nhất giữa mọi cặp đỉnh</li> <li>❖ Ký hiệu: <ul style="list-style-type: none"> <li>▪ <math>L[i, j]</math>: Độ dài đường đi ngắn nhất từ đỉnh i đến đỉnh j</li> <li>▪ Ban đầu, L được khởi tạo giống với C: <math>L[i,j] = C[i,j]</math></li> <li>▪ <math>T[i, j]</math>: Lưu lại đường đi từ i đến j</li> <li>▪ <math>T[i, j] = k</math>: k là đỉnh trước đỉnh j trong đường đi từ i <math>\rightarrow</math> j</li> </ul> </li> <li>❖ Ý tưởng:</li> </ul>

<ul style="list-style-type: none"> <li>▪ U: là tập các đỉnh chưa được chọn</li> </ul> <p>❖ Ý tưởng</p> <ul style="list-style-type: none"> <li>▪ Khởi đầu: <math>S = \{s_0\}</math>, <math>L[w] = C[s_0, w]</math>, <math>T[w] = w</math></li> <li>▪ Tại mỗi bước, ta chọn một đỉnh w trong U mà khoảng cách <math>L[w]</math> từ đỉnh nguồn tới w là nhỏ nhất.</li> <li>▪ Loại w khỏi U và thêm w vào tập S.</li> <li>▪ Với mỗi đỉnh u thuộc U, tính lại độ dài đường đi ngắn nhất từ <math>s_0</math> đến u.</li> </ul> $L(u) = \min(L(u), L(w) + C[w, u])$ <ul style="list-style-type: none"> <li>▪ Kết thúc: S = tập các đỉnh của G</li> <li>▪ Để tìm đường đi từ <math>s_0</math> đến một đỉnh w, ta sử dụng mảng T để lần ngược các đỉnh trước w trên đường đi.</li> </ul>	<ul style="list-style-type: none"> <li>▪ Tại lần lặp thứ k sẽ xác định khoảng cách ngắn nhất giữa hai đỉnh i và j theo công thức:  <math display="block">L_k[i, j] = \min( L_{k-1}[i, j], L_{k-1}[i, k] + L_{k-1}[k, j] )</math></li> <li>▪ Để tìm đường đi từ v đến một đỉnh w, ta sử dụng mảng T để lần ngược các đỉnh trước k trên đường đi.</li> </ul>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## IV. Hướng dẫn thực hành

### 1. Tạo dự án

Sinh viên có thể chọn cài đặt kiểu dữ liệu đồ thị theo ngôn ngữ C# hoặc C++. Các phần sau minh họa cách cài đặt, hướng dẫn bằng mã giả trên cả ngôn ngữ C# (bên trái) và C++ (bên phải).

*Sử dụng dự án đã tạo trong bài Lab3*

## V. Bài tập thực hành

### Bài 1. Cây bao trùm tối thiểu

#### a. Thuật toán Prim

Tạo một lớp mới, đặt tên là Prim.cs. Bổ sung đoạn mã sau:

```
class Prim
{
    private Graph g;
    private Path[] tree;

    public Prim(Graph graph)
    {
        g = graph;
        tree = new Path[g.NumVertices];
    }

    public void FindMST()
    {
```

```
// Tìm cây bao trùm nhỏ nhất theo thuật toán Prim
// Kết quả được lưu trong mảng tree. Chỉ số mảng i
// là đỉnh nguồn, tree[i].Parent là đỉnh cuối và
// tree[i].Length là trọng số của cạnh được chọn.
void Prim(Graph g, Path tree[MAX])
{
    // Khởi tạo cây ban đầu có tất cả các cạnh
    // xuất phát từ đỉnh đầu tiên (đỉnh 0)
    for (int i=1; i< SỐ đỉnh của g; i++)
    {
        tree[i].Length = g.Cost[0][i];
        tree[i].Parent = 0;
    }

    CostType min;    // Lưu cạnh có trọng số bé nhất
```

```

        this.Run();
        this.PrintMST();
    }
    private void Run()
    {
        g.ResetFlags();
        for (int i = 0; i < g.NumVertices; i++)
            tree[i] = new Path(g.Cost[0, i], 0);

        int min;    // Lưu cạnh có trọng số bé nhất
        int minVertex;    // Lưu đỉnh cuối của cạnh đó

        for (int i = 1; i < g.NumVertices; i++)
        {
            min = Graph.INF;    // Giả sử min = vô cùng
            minVertex = 1;

            for (int j = 2; j < g.NumVertices; j++)
            {
                if (g.Vertices[j].Visited == false &&
                    tree[j].Length < min)
                {
                    min = tree[j].Length;
                    minVertex = j;
                }
            }
            g.Vertices[minVertex].Visited = true;
            for (int j = 0; j < g.NumVertices; j++)
            {
                if (g.Vertices[j].Visited == false &&
                    g.Cost[minVertex, j] < tree[j].Length)
                {
                    tree[j].Length = g.Cost[minVertex, j];
                    tree[j].Parent = minVertex;
                }
            }
        }
    }

    private void PrintMST()
    {
        Console.WriteLine("Cay bao trum gom cac canh sau:");
        int sum = 0;
        for (int i = 1; i < g.NumVertices; i++)
        {
            sum += tree[i].Length;
        }
    }

```

```

    int minVertex;    // Lưu đỉnh cuối của cạnh đó

    // Tìm n-1 cạnh cho cây bao trùm
    for (int i=1; i< Số đỉnh của g; i++)
    {
        min = INF;    // Giả sử min = vô cùng
        minVertex = 1;

        // Duyệt qua các cạnh để tìm cạnh min
        for (int j=2; j< Số đỉnh của g; j++)
        {
            if (Đỉnh j chưa được xét &&
                tree[j].Length < min)
            {
                min = tree[j].Length;
                minVertex = j;
            }
        }
        // Đánh dấu đỉnh cuối của cạnh min là đã xét
        g.Vertices[minVertex].Visited = YES;

        // Cập nhật lại trọng số của các cạnh
        // với đỉnh nguồn bây giờ là minVertex
        for (int j=0; j< Số đỉnh của g; j++)
        {
            if (Đỉnh j chưa được xét && Chi phí từ
                đỉnh minVertex đến j < tree[j].Length)
            {
                tree[j].Length = g.Cost[minVertex][j];
                tree[j].Parent = minVertex;
            }
        }
    }

    // Xuất danh sách cạnh được chọn làm cây bao trùm
    // theo thuật toán Prim
    void PrintPrimMST(Graph g, Path tree[MAX])
    {
        cout << endl << "Cay bao trum gom cac canh sau:";
        CostType sum = 0;    // Lưu tổng trọng số

        // Duyệt qua các đỉnh để tìm cạnh nối với đỉnh đó
        // và nằm trong cây bao trùm tối thiểu
        for (int i=1; i<g.NumVertices; i++)
        {
            Cập nhật tổng trọng số;    // sum
            Xuất nhãn của đỉnh đầu, nhãn của đỉnh cuối và

```

```

        Console.WriteLine("{0} --> {1} : {2}",
g.Vertices[i].Label,
g.Vertices[tree[i].Parent].Label, tree[i].Length);
    }
    Console.WriteLine("Cay bao trum ngan nhat co chieu
        dai : {0}", sum);
    }
}

```

## b. Thuật toán Kruskal

Tạo một lớp mới, đặt tên là Kruskal.cs. Bổ sung đoạn mã sau:

```

class Kruskal
{
    private Graph g;
    private Edge[] tree;

    public Kruskal(Graph graph)
    {
        g = graph;
        tree = new Edge[g.NumEdges];
    }

    public void FindMST()
    {
        this.Run();
        this.PrintMST();
    }

    private void AdjMatrix2EdgeList()
    {
        int count = 0;
        for (int i = 0; i < g.NumVertices; i++)
            for (int j = 0; j < i; j++)
                if (g.IsConnected(i, j))
                {
                    tree[count] = new Edge();
                    tree[count].Source = i;
                    tree[count].Target = j;
                    tree[count].Weight = g.Cost[i, j];
                    tree[count].Marked = false;
                    count++;
                }
    }

    private void QSortEdges(int d, int c)
    {

```

```

        chiều dài (trọng số) của cạnh nối 2 đỉnh đó;
    }
    cout << endl << "Cay bao trum ngan nhat co chieu dai
        : " << sum;
}

```

```

// Duyệt ma trận kề và lấy ra danh sách các cạnh của
// đồ thị. Mỗi cạnh lưu đỉnh đầu, cuối và trọng số.
int AdjMatrix2EdgeList(Graph g, Edge edgeList[UPPER])
{
    int count = 0; // Lưu số cạnh
    for (int i=0; i<g.NumVertices; i++) // Duyệt nửa dưới
        for (int j=0; j<i; j++) // của ma trận kề
            if (Có cạnh nối 2 đỉnh I và J)
            {
                Tạo cạnh v;
                Gán đỉnh nguồn của v = i;
                Gán đỉnh đích của v = j;
                Gán trọng số của v = chi phí đi từ
                đỉnh I đến đỉnh J;
                Đánh dấu cạnh v chưa được xét;
                Đưa cạnh v vào danh sách edgeList;
                Tăng số cạnh (count) lên 1;
            }
    return count;
}

// Sắp xếp danh sách cạnh tăng dần theo trọng số của cạnh
void QSortEdges(Edge edgeList[MAX], int d, int c)
{
    int i = d, j = c; // d = đầu, c = cuối

    // Giá trị ở giữa mảng
    CostType mid = edgeList[(d + c) / 2].Weight;

    // tiến hành tách mảng thành 2 phần
    while (i <= j)
    {
        // Tìm các ptu đúng sai vị trí trong mảng
        while (edgeList[i].Weight < mid) i++;
        while (edgeList[j].Weight > mid) j--;
    }
}

```

```

int i = d, j = c; // d = đầu, c = cuối
int mid = tree[(d + c) / 2].Weight;

while (i <= j)
{
    while (tree[i].Weight < mid) i++;
    while (tree[j].Weight > mid) j--;

    if (i <= j)
    {
        Edge temp = tree[i];
        tree[i] = tree[j];
        tree[j] = temp;
        i++;
        j--;
    }
}

if (i < c) QSortEdges(i, c);
if (d < j) QSortEdges(d, j);
}

// Tìm nút gốc của cây chứa đỉnh x
private int Find(int[] leader, int x)
{
    while (x != leader[x])
        x = leader[x];
    return x;
}

// Hợp nhất 2 cây bằng cách nối thêm cạnh e
private bool Union(int[] leader, Edge e)
{
    int x = Find(leader, e.Source);
    int y = Find(leader, e.Target);

    if (x == y)
        return false;
    else if (x < y)
        leader[y] = x;
    else
        leader[x] = y;
    return true;
}

private void Run()
{
    AdjMatrix2EdgeList();

```

```

// Nếu có 2 ptu sai vị trí -> hoán vị chúng
if (i <= j)
{
    Edge temp = edgeList[i];
    edgeList[i] = edgeList[j];
    edgeList[j] = temp;
    i++;
    j--;
}

// Sắp xếp mảng con bên phải mid
if (i < c) QSortEdges(edgeList, i, c);

// Sắp xếp mảng con bên trái mid
if (d < j) QSortEdges(edgeList, d, j);
}

// Tìm nút gốc của cây chứa đỉnh x
int Find(int leader[MAX], int x)
{
    // chừng nào chưa tìm thấy gốc thì
    while (x != leader[x])
        x = leader[x];    // Chuyển đến nút cha
    return x;
}

// Hợp nhất 2 cây bằng cách nối thêm cạnh e
bool Union(int leader[MAX], Edge e)
{
    int x = Tìm nút gốc của cây chứa đỉnh e.Source;
    int y = Tìm nút gốc của cây chứa đỉnh e.Target;

    // Nếu trùng gốc => không thêm cạnh
    if (x == y)
        return false;
    else if (x < y)    // Nhập chung cây y vào cây x
        leader[y] = x;    // hay cây chứa y có gốc là x
    else
        leader[x] = y;    // Nhập chung cây x vào y
    return true;
}

// Thuật toán Kruskal tìm cây bao trùm tối thiểu.
// Kết quả được lưu trong mảng tree. Chỉ số mảng i
// là đỉnh nguồn, tree[i].Parent là đỉnh cuối và
// tree[i].Length là trọng số của cạnh được chọn.
void Kruskal(Graph g, Edge tree[UPPER])

```

```

QSortEdges(0, g.NumEdges - 1);

// Khởi tạo đỉnh gốc của các cây con
int[] leader = new int[g.NumVertices];
for (int i = 0; i < g.NumVertices; i++)
    leader[i] = i;

// Duyệt các cạnh tăng dần theo trọng số
int count = 0;
for (int i = 0; i < g.NumEdges; i++)
{
    if (Union(leader, tree[i]))
    {
        tree[i].Marked = true;
        count++;
        if (count == g.NumVertices - 1)
            break;
    }
}

private void PrintMST()
{
    Console.WriteLine("Cay bao trum gom cac canh sau:");
    int sum = 0;

    for (int i = 0; i < g.NumEdges; i++)
        if (tree[i].Marked)
        {
            Console.WriteLine("{0} --> {1} : {2}",
                g.Vertices[tree[i].Source].Label,
                g.Vertices[tree[i].Target].Label,
                tree[i].Weight);
            sum += tree[i].Weight;
        }
    Console.WriteLine("Tong chieu dai cay bao trum la {0}", sum);
}
}

```

```

{
    // Tạo ra danh sách các cạnh từ MT kề
    int ne = AdjMatrix2EdgeList(g, tree);

    // Sắp xếp các cạnh tăng dần theo trọng số
    QSortEdges(tree, 0, ne-1);
    // Khởi tạo đỉnh gốc của các cây con
    int leader[MAX];
    for (int i=0; i<g.NumVertices+1; i++)
        leader[i] = i;

    // Duyệt các cạnh tăng dần theo trọng số
    int count = 0;
    for (int i=0; i<ne; i++)
    {
        // Nếu có thể ghép nó vào cây bao trùm
        if (Union(leader, tree[i]))
        {
            tree[i].Marked = YES;    // Đánh dấu chọn
            count++;                // Tăng biến đếm

            // Nếu đã chọn đủ n-1 cạnh cho cây
            if (count == g.NumVertices - 1)
                break;              // thì dừng
        }
    }

    // Xuất danh sách các cạnh được chọn để tạo cây bao
    // trùm nhỏ nhất theo thuật toán Kruskal
    void PrintKruskalMST(Graph g, Edge tree[UPPER])
    {
        cout << endl << "Cay bao trum gom cac canh sau:";
        CostType sum = 0;    // Lưu tổng chiều dài cây

        // Duyệt qua các cạnh
        for (int i=0; i<g.NumEdges; i++)
        {
            if (Cạnh I được chọn)
            {
                Xuất nhãn của đỉnh nguồn, nhãn của đỉnh
                đích và trọng số của cạnh nối 2 đỉnh đó;
                Cập nhật tổng chiều dài cây bao trùm;
            }
        }
        cout << endl << "Tong chieu dai cay bao trum la "
            << sum;
    }
}

```

## Bài 2. Tìm đường đi ngắn nhất

### a. Tìm đường đi ngắn nhất từ 1 đỉnh tới các đỉnh còn lại (thuật toán Dijkstra)

Tạo lớp mới, đặt tên lớp là Dijkstra.cs. Bổ sung đoạn mã sau:

```
class Dijkstra
{
    private Graph g;
    private Path[] route;

    public Dijkstra(Graph graph)
    {
        g = graph;
        route = new Path[g.NumVertices];
    }

    public void FindPath(string sourceLabel)
    {
        int source = g.FindVertex(sourceLabel);
        if (source == Graph.NOTFOUND)
            throw new IndexOutOfRangeException();

        Run(source);
        for (int i = 0; i < g.NumVertices; i++)
        {
            if (i == source) continue;
            if (route[i].Length == Graph.INF)
            {
                Console.WriteLine("\r\nKhong co duong di tu {0} den {1}", sourceLabel, g.Vertices[i].Label);
            }
            else
            {
                Console.WriteLine("\r\nDuong di ngan nhat tu {0} den {1} la : ", sourceLabel, g.Vertices[i].Label);
                PrintPath(i);
                Console.WriteLine(" : Do dai = {0}", route[i].Length);
            }
        }
    }

    private void Run(int source)
    {
        g.ResetFlags();
        for (int i = 0; i < g.NumVertices; i++)
            route[i] = new Path(g.Cost[source, i], source);

        g.Vertices[source].Visited = true;
        route[source].Length = 0;

        int counter = 0, curr = source, minVertex, minw;
        while (counter < g.NumVertices - 1)
        {
            minw = Graph.INF;
```

```
// Thuật toán Dijkstra tìm đường đi ngắn nhất từ đỉnh nguồn
// source đến tất cả các đỉnh còn lại. Độ dài đường đi được
// lưu trong mảng labels, đường đi lưu trong mảng path.
void Dijkstra(Graph g, int source, Path road[MAX])
{
    CostType min; // Lưu độ dài đường đi ngắn nhất
    int counter, // Đếm số đỉnh đã xét
        minVertex, // Lưu đỉnh có nhãn nhỏ nhất
        curr; // Đỉnh hiện tại đang xét

    // Khởi gán giá trị cho các đoạn đường đi
    for (int i=0; i<Số đỉnh của g; i++)
    {
        road[i].Length = g.Cost[source][i];
        road[i].Parent = source;
    }

    Đánh dấu đỉnh source đã được xét;
    Gán nhãn cho đỉnh source = 0;
    counter = 1; // Có 1 đỉnh đã xét
    curr = source; // Bắt đầu từ nguồn

    // Trong khi chưa xét hết các đỉnh
    while (counter < Số đỉnh của g - 1)
    {
        min = INF; // Giả sử min = vô cùng
        minVertex = curr; // Đỉnh min = đỉnh hiện tại

        // Duyệt qua từng đỉnh để kiểm tra
        for (int i=0; i< Số đỉnh của g; i++)
        {
            if (Đỉnh i chưa được xét)
            {
                // Gán lại nhãn cho các đỉnh
                if (road[i].Length >
                    road[curr].Length + g.Cost[curr][i])
                {
                    road[i].Length =
                        road[curr].Length + g.Cost[curr][i];
                    road[i].Parent = curr;
                }
                // Tìm đỉnh có nhãn nhỏ nhất
                if (min > road[i].Length)
```



```

minVertex = curr;

for (int i = 0; i < g.NumVertices; i++)
{
    if (g.Vertices[i].Visited == false)
    {
        if (route[i].Length >
            route[curr].Length + g.Cost[curr, i])
        {
            route[i].Length =
                route[curr].Length + g.Cost[curr, i];
            route[i].Parent = curr;
        }
        if (minw > route[i].Length)
        {
            minw = route[i].Length;
            minVertex = i;
        }
    }
}
curr = minVertex;
g.Vertices[curr].Visited = true;
counter++;
}

private void PrintPath(int target) {
    if (target != route[target].Parent)
        PrintPath(route[target].Parent);
    Console.WriteLine(" --> {0}", g.Vertices[target].Label);
}
}

```

## b. Tìm đường đi ngắn nhất giữa mọi cặp đỉnh (thuật toán Floyd)

Tạo lớp mới, đặt tên lớp là Floyd.cs. Bổ sung đoạn mã sau:

```

class Floyd
{
    private Graph g;
    private Path[, ] route;

    public Floyd(Graph graph)
    {
        g = graph;
        route = new Path[g.NumVertices, g.NumVertices];
        this.Run();
    }

    public void FindPath(string start, string end)
    {
        int source = g.FindVertex(start);

```

```

        {
            min = road[i].Length;
            minVertex = i;
        }
    }

    curr = minVertex; // Xét đỉnh có nhãn nhỏ nhất
    Đánh dấu đỉnh curr là đã xét;
    Tăng số đỉnh đã xét lên 1;
}

// In ra đường đi từ đỉnh nguồn tới đỉnh đích target
void PrintPath(Graph g, Path road[MAX], int target)
{
    // Nếu chưa gặp đỉnh nguồn
    if (road[target].Parent != target)
        // thì tìm đường từ nguồn tới đỉnh trước target
        PrintPath(g, road, road[target].Parent);

    // Sau đó in ra nhãn của đỉnh trên đường đi
    cout << " --> " << g.Vertices[target].Label;
}

```

```

// Tìm đường đi ngắn nhất giữa mọi cặp đỉnh theo thuật toán Floyd
void Floyd(Graph g, Path route[MAX][MAX])
{
    int i, j, k;

    // Khởi tạo chiều dài đường đi giữa các cặp đỉnh
    for (i=0; i< Số đỉnh của g; i++)
        for (j=0; j< Số đỉnh của g; j++)
        {
            route[i][j].Length = g.Cost[i][j];
            route[i][j].Parent = i;
        }

    // Tính toán lại đường đi giữa các cặp đỉnh
    for (k=0; k< Số đỉnh của g; k++)

```

```

int target = g.FindVertex(end);

if (route[source, target].Length == Graph.INF)
    Console.WriteLine("Khong co duong di tu {0}
                        den {1}", start, end);
else {
    Console.WriteLine("\r\nDuong di ngan nhat
                      tu {0} den {1} la : ", start, end);
    PrintPath(source, target);
    Console.WriteLine(" : Do dai = {0}",
                      route[source, target].Length);
}
}

private void Run() {
    int i, j, k;
    for (i = 0; i < g.NumVertices; i++)
        for (j = 0; j < g.NumVertices; j++)
            route[i, j] = new Path(g.Cost[i, j], i);

    for (k = 0; k < g.NumVertices; k++)
        for (i = 0; i < g.NumVertices; i++)
            for (j = 0; j < g.NumVertices; j++)
                if (route[i, j].Length >
                    route[i, k].Length + route[k, j].Length)
                {
                    route[i, j].Length = route[i,
                        k].Length + route[k, j].Length;
                    route[i, j].Parent = route[k, j].Parent;
                }
}

private void PrintPath(int source, int target) {
    if (route[source, target].Parent != target) {
        int prev = route[source, target].Parent;
        PrintPath(source, prev);
    }
    Console.Write("-->{0}", g.Vertices[target].Label);
}
}

```

```

for (i=0; i< Số đỉnh của g; i++)
    for (j=0; j< Số đỉnh của g; j++)
        // Nếu đường đi từ i->j qua đỉnh k
        // ngắn hơn đường đi trực tiếp i->j
        if (route[i][j].Length >
            route[i][k].Length + route[k][j].Length)
        {
            // thì cập nhật lại độ dài đường đi giữa 2 đỉnh i & j
            route[i][j].Length=route[i][k].Length+route[k][j].Length;
            // Cập nhật đỉnh trung gian
            route[i][j].Parent = route[k][j].Parent;
        }
}

// Xuất đường đi ngắn nhất từ đỉnh source đến đỉnh target
// Ma trận road lưu độ dài đường đi ngắn nhất và đường đi
void FloydPath(Graph g, Path route[MAX][MAX],
                int source, int target)
{
    if (route[source][target].Parent != target)
        FloydPath(g, route, source,
                    route[source][target].Parent);

    // Sau đó in ra nhãn của đỉnh trên đường đi
    cout << " --> " << g.Vertices[target].Label;
}

```

### Bài 3. Kiểm tra chương trình và xem kết quả

```

class Program
{
    static void Main(string[] args)
    {
        Graph g = new Graph();
    }
}

```

```

#include <iostream>
#include <fstream>
#include <conio.h>

using namespace std;
#include "common.h"

```

```

g.AddVertex("A"); // 0
g.AddVertex("B"); // 1
g.AddVertex("C"); // 2
g.AddVertex("D"); // 3
g.AddVertex("E"); // 4
g.AddVertex("F"); // 5
g.AddVertex("G"); // 6
g.AddVertex("H"); // 7

g.AddEdge(0, 1, 4); // AB 4
g.AddEdge(0, 4, 6); // AE 6
g.AddEdge(0, 5, 2); // AF 2
g.AddEdge(1, 2, 1); // BC 1
g.AddEdge(1, 4, 1); // BE 1
g.AddEdge(1, 7, 5); // BH 5
g.AddEdge(2, 3, 2); // CD 2
g.AddEdge(3, 7, 1); // DH 1
g.AddEdge(4, 5, 2); // EF 2
g.AddEdge(4, 6, 3); // EG 3

Console.WriteLine("So dinh : ", g.NumVertices);
Console.WriteLine("So canh : ", g.NumVertices);

IVisitor visitor = new NameVisitor();

Console.WriteLine("Ket qua duyet theo chieu sau");
g.ResetFlags();
g.DFSLoop(0, visitor);
//g.DFSRecursion(0, visitor);

Console.WriteLine();
Console.WriteLine("Ket qua duyet theo chieu rong");
g.ResetFlags();
g.BFS(0, visitor);

Dijkstra dijkstra = new Dijkstra(g);
dijkstra.FindPath("A");

Floyd floyd = new Floyd(g);
floyd.FindPath("A", "F");

Warshall warshall = new Warshall(g);
warshall.FindPath();

Prim prim = new Prim(g);
prim.FindMST();

```

```

#include "stack.h"
#include "queue.h"
#include "graph.h"

void main()
{
    Graph g = InitGraph(false);

    // =====
    // ... Đoạn này xem trong bài Lab 3 ...
    // =====

    cout << endl << endl << "=====";
    cout << endl << "Tim duong di ngan nhat theo thuat
        toan Dijkstra" << endl;

    ResetFlags(g);

    Path road[MAX];
    Dijkstra(g, 0, road);

    for (int i=1; i<g.NumVertices; i++) {
        if (road[i].Length == INF)
            cout << endl << "Khong co duong di tu dinh "
                << g.Vertices[0].Label << " den dinh "
                << g.Vertices[i].Label;
        else
        {
            cout << endl << "Duong di ngan nhat tu dinh "
                << g.Vertices[0].Label << " den dinh "
                << g.Vertices[i].Label << " la " << endl;
            PrintPath(g, road, i);
            cout<<" : do dai = " << road[i].Length << endl;
        }
    }

    cout << endl << "=====";
    cout << endl << "Tim duong di ngan nhat theo thuat
        toan Floyd" << endl;

    ResetFlags(g);

    Path route[MAX][MAX];
    Floyd(g, route);

    for (int i=1; i<g.NumVertices; i++) {
        if (route[0][i].Length == INF)
            cout << endl << "Khong co duong di tu dinh "
                << g.Vertices[0].Label << " den dinh "

```

```

Kruskal kruskal = new Kruskal(g);
kruskal.FindMST();

Console.ReadLine();
}
}

```

```

        << g.Vertices[i].Label;
    else
    {
        cout << endl << "Duong di ngan nhat tu dinh "
            << g.Vertices[0].Label << " den dinh "
            << g.Vertices[i].Label << " la " << endl;
        FloydPath(g, route, 0, i);
        cout<<" : do dai = "<<route[0][i].Length<<endl;
    }
}

cout << endl << "===== ";
cout << endl << "Tim bao dong chuyen tiep" << endl;
ResetFlags(g);
Warshall(g, route);

for (int i=0; i<g.NumVertices; i++) {
    for (int j=0; j<g.NumVertices; j++) {
        if (i == j) continue;
        cout << endl << g.Vertices[i].Label <<
            " --> " << g.Vertices[j].Label;
        if (route[i][j].Length == 0)
            cout << " : Khong co duong di";
        else
            cout << " : Co duong di";
    }
}

cout << endl << endl << "===== ";
cout << endl << "Tim cay bao trum ngan nhat theo
    thuat toan Prim" << endl;

ResetFlags(g);
Prim(g, road);
PrintPrimMST(g, road);

cout << endl << endl << "===== ";
cout << endl << "Tim cay bao trum ngan nhat theo
    thuat toan Kruskal" << endl;

ResetFlags(g);
Edge mst[UPPER];
Kruskal(g, mst);
PrintKruskalMST(g, mst);

_getch();
}

```

## VI. Bài tập

### Bài 1. Mạng truyền thông

Một công ty lập kế hoạch xây dựng một mạng truyền thông nối năm trung tâm máy tính với nhau. Bất kỳ hai trung tâm nào cũng có thể được nối kết với nhau bằng đường điện thoại. Cần phải kết nối như thế nào để đảm bảo giữa hai trung tâm máy tính bất kỳ luôn có đường truyền thông sao cho tổng số tiền thuê bao của toàn mạng là tối thiểu? Phí thuê bao phải trả hàng tháng đối với các đường truyền thông được cho trong bảng sau:

Từ trung tâm	Đến trung tâm	Phí thuê bao
San Francisco	New York	\$2000
San Francisco	Chicago	\$1200
San Francisco	Denver	\$900
San Francisco	Atlanta	\$2200
Chicago	Denver	\$1300

Từ trung tâm	Đến trung tâm	Phí thuê bao
Chicago	New York	\$1000
Chicago	Atlanta	\$700
New York	Denver	\$1600
New York	Atlanta	\$800
Denver	Atlanta	\$1400

### Bài 2. Ông Ngâu, Bà Ngâu.

Hắn các bạn đã biết ngày "ông Ngâu bà Ngâu" hàng năm, đó là một ngày đầy mưa và nước mắt. Tuy nhiên, một ngày trước đó, nhà Trời cho phép 2 "ông bà" được đoàn tụ. Trong vũ trụ vùng thiên hà nơi ông Ngâu bà Ngâu ngự trị có  $N$  hành tinh đánh số từ 1 đến  $N$ , ông ở hành tinh Adam (có số hiệu là  $S$ ) và bà ở hành tinh Eva (có số hiệu là  $T$ ). Họ cần tìm đến gặp nhau.

$N$  hành tinh được nối với nhau bởi một hệ thống cầu vòng. Hai hành tinh bất kỳ có thể không có hoặc có duy nhất một cầu vòng (hai chiều) nối giữa chúng. Họ luôn đi tới mục tiêu theo con đường ngắn nhất. Họ đi với tốc độ không đổi và nhanh hơn tốc độ ánh sáng. Điểm gặp mặt của họ chỉ có thể là tại một hành tinh thứ 3 nào đó.

Yêu cầu: Hãy tìm một hành tinh sao cho ông Ngâu và bà Ngâu cùng đến đó một lúc và thời gian đến là sớm nhất. Biết rằng, hai người có thể cùng đi qua một hành tinh nếu như họ đến hành tinh đó vào những thời điểm khác nhau.

Dữ liệu được cho trong file ongbangau.inp có cấu trúc như sau:

Dòng đầu là 4 số  $N M S T$  ( $N \leq 100$ ,  $1 \leq S \neq T \leq N$ ),  $M$  là số cầu vòng.  $M$  dòng tiếp, mỗi dòng gồm hai số  $I J L$  thể hiện có cầu vòng nối giữa hai hành tinh  $I, J$  và cầu vòng đó có độ dài là  $L$  ( $1 \leq I \neq J \leq N$ ,  $0 < L \leq 200$ ).

