Chương 2:

Đồ thị

Nội dung chính

1. Các định nghĩa
2. Biểu diễn đồ thị
3. Các phép duyệt đồ thị
4. Một số bài toán trên đồ thị

2

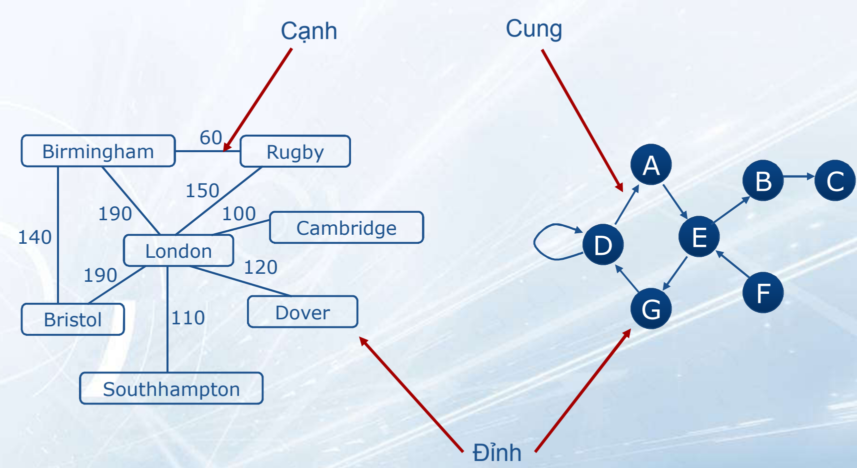
Các định nghĩa

- Một đồ thị G được định nghĩa là

  G = (V, E) V: vertex (vertices) E: edge
- Trong đó:
  - V(G) là tập hữu hạn các đỉnh và khác rỗng
  - E(G) là tập các cạnh(cung) = Tập các cặp đỉnh (u, v) mà u, v ∈ V.
- Các đỉnh còn gọi là các nút (node) hay điểm (point)
- Mỗi cạnh nối giữa hai đỉnh v, w có thể ký hiệu là cặp (v,w). Hai đỉnh có thể trùng nhau.
- Nếu cặp (v,w) có thứ tự thì gọi là cạnh có thứ tự hay cạnh có hướng, hay là cung. Ngược lại, ta nói là cạnh không có thứ tự hay cạnh vô hướng, hay văn tắt là cạnh.

3

Ví dụ về đồ thị

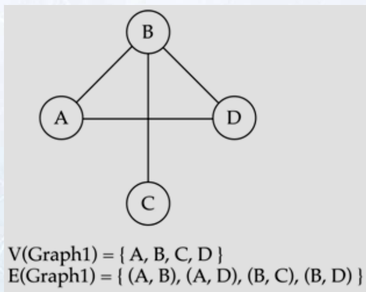


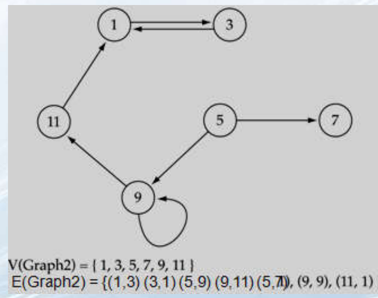
4

Các định nghĩa

❖ Có hai loại đồ thị:

- Đồ thị có hướng là đồ thị mà các cạnh của nó là có hướng. Nghĩa là các cặp đỉnh (v,w) có phân biệt thứ tự.





- Đồ thị vô hướng là đồ thị mà các cặp đỉnh tương ứng với các cạnh của nó không phân biệt thứ tự.

-> Ta nói cạnh khi xét đồ thị vô hướng, cung khi xét đồ thị có hướng.

5

Các định nghĩa

❖ Đỉnh kề: Hai đỉnh được gọi là kề nhau nếu chúng được nối với nhau bởi một cạnh (cung).

❖ Đường đi là một dãy tuần tự các đỉnh v_1, v_2, …, v_n sao cho (v_i, v_{i+1}) là một cạnh (cung) trên đồ thị.

- Đỉnh v_1 được gọi là đỉnh đầu, đỉnh v_n được gọi là đỉnh cuối.
- Độ dài đường đi là số cạnh (cung) trên đường đi.

❖ Đỉnh X gọi là có thể đi đến được từ đỉnh Y nếu tồn tại một đường đi từ đỉnh Y đến đỉnh X.

❖ Đường đi đơn là đường đi mà mọi đỉnh trên đó đều khác nhau, ngoại trừ đỉnh đầu và cuối có thể trùng nhau.

❖ Chu trình là đường đi có đỉnh đầu trùng với đỉnh cuối.

6

Các định nghĩa

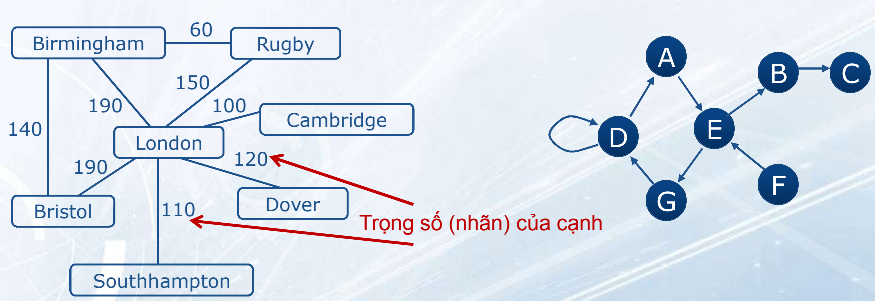
- Chu trình đơn là một đường đi đơn có đỉnh đầu và đỉnh cuối trùng nhau.
- Cấp của đồ thị là số đỉnh của đồ thị.
- Kích thước của đồ thị là số cạnh (cung) của đồ thị
- Đồ thị rỗng là đồ thị có kích thước bằng 0
- Trong nhiều ứng dụng, ta thường kết hợp các giá trị (value) hoặc nhãn (label) cho các cạnh (cung). Khi đó, ta gọi là đồ thị có trọng số (vô hướng, có hướng).

Nhãn có thể có kiểu tùy ý và cạnh (cung) dùng để biểu diễn chi phí, khoảng cách,…

7

Ví dụ

- Birmingham, London, A, D là nhãn của các đỉnh
- Bậc của đỉnh có nhãn London là 6; bậc của đỉnh có nhãn A là 2.



- London,Bristol,Birmingham,London,Dover là một đường đi
- London, Bristol, Birmingham, London là một chu trình
- D, A, E, B, C là một đường đi
- D, A, E, G, D là một chu trình

8

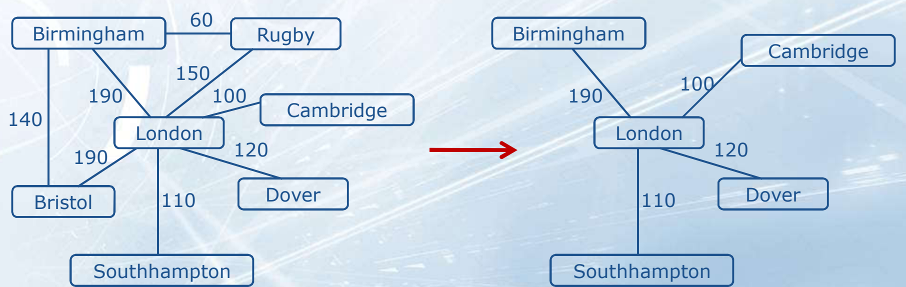
Các định nghĩa

- Đồ thị con của một đồ thị G=(V, E) là đồ thị G'=(V', E') trong đó

$$V' \subseteq V$$

$$E' = \{ (u,v) \in E \mid u \in V' \text{ và } v \in V' \}$$

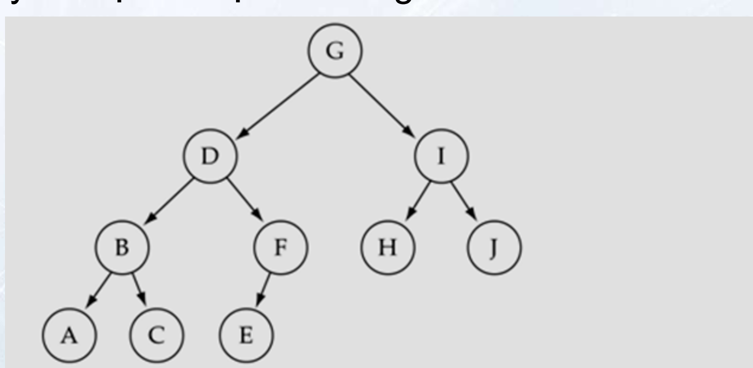
- Nếu G' là đồ thị con của G, ta viết G' ≤ G



9

Các định nghĩa

- Cây là một dạng đặc biệt của đồ thị
- Cây là một đồ thị có hướng



V(Graph3) = { A, B, C, D, E, F, G, H, I, J }
E(Graph3) = { (G, D), (G, I), (D, B), (D, F) (I, H), (I, J), (B, A), (B, C), (F, E) }

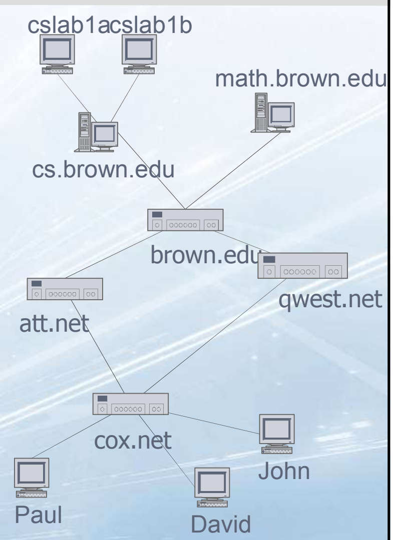
10

Một số ví dụ về đồ thị

❖ Bản đồ định tuyến máy bay
- Đỉnh: cảng hàng không
- Cạnh: Tuyến đường bay

❖ Mạng máy tính
- Đỉnh: Các thiết bị mạng máy tính
- Cạnh: Liên kết (đường) truyền thông

❖ Lược đồ quan hệ - thực thể
- Đỉnh: Các thực thể
- Cạnh: Quan hệ giữa các thực thể



Một số ứng dụng của đồ thị

❖ Xác định đường đi ngắn nhất giữa hai thành phố
❖ Xác định mạng giao thông chi phí thấp nhất
❖ Thiết kế mạch tối ưu cho một chip máy tính
❖ Xây dựng trình biên dịch
❖ Tạo bộ thu gom rác trong các chương trình
❖ Biểu diễn nhật ký gia tộc, gia phả
❖ Sơ đồ Pert (Program Evaluation and Review Technique)
❖ . . .

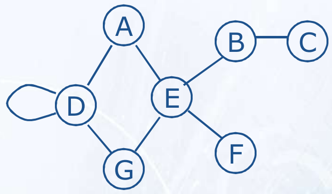
Nội dung chính

1. Các định nghĩa
2. **Biểu diễn đồ thị**
3. Các phép duyệt đồ thị
4. Một số bài toán trên đồ thị

Các phương pháp biểu diễn đồ thị

- Ma trận kề
- Danh sách kề

Ma trận kề



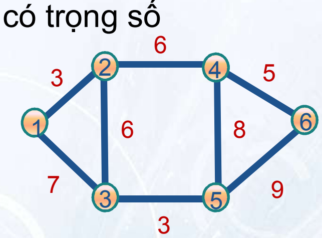
	A	B	C	D	E	F	G
A				●	●		
B			●		●		
C		●					
D	●			●			●
E	●	●				●	●
F					●		
G				●	●		

- Ma trận kề $A_G=(a_{ij})$ của đồ thị G là ma trận vuông cấp nxn trong đó:
  - n là số đỉnh của đồ thị
  - a_{ij} = w nếu (v_i, v_j) là một cạnh(cung) của G
    - w có thể mang giá trị 1 hay true (đồ thị không có trọng số)
    - w cũng có thể là nhãn của cạnh (đồ thị có trọng số). Khi đó, A_G gọi là ma trận trọng số
- Ma trận kề của đồ thị vô hướng là ma trận đối xứng qua đường chéo chính.
- Chỉ sử dụng ma trận kề cho các đồ thị có "cấp" nhỏ.

15

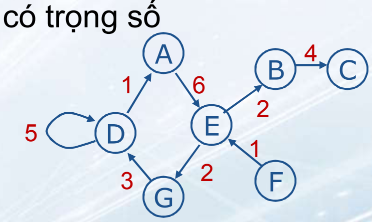
Biểu Diễn Đồ Thị Bằng Ma Trận Kề

- Trường hợp đồ thị vô hướng có trọng số



	1	2	3	4	5	6
1	0	3	7	0	0	0
2	3	0	6	6	0	0
3	7	6	0	0	3	0
4	0	6	0	0	8	5
5	0	0	3	8	0	9
6	0	0	0	5	9	0

- Trường hợp đồ thị có hướng có trọng số



	A	B	C	D	E	F	G
A	0	0	0	0	6	0	0
B	0	0	4	0	0	0	0
C	0	0	0	0	0	0	0
D	1	0	0	5	0	0	0
E	0	2	0	0	0	0	2
F	0	0	0	0	1	0	0
G	0	0	0	3	0	0	0

16

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Định nghĩa hằng số
#define     UPPER       100    // Số ptu tối đa
#define     ZERO        0      // Giá trị 0
#define     MAX         30     // Số đỉnh tối đa
#define     INF         1000   // Vô cùng
#define     YES         1      // Đã xét
#define     NO          0      // Chưa xét
#define     NULLDATA    -1     // Giá giá trị rỗng

// Định nghĩa các kiểu dữ liệu
typedef     char        LabelType;
Typedef     int         CostType;
Typedef     CostType    MaTrix[MAX][MAX];
```

17

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Định nghĩa cấu trúc của một đỉnh
struct Vertex
{
  LabelType       Label;      // Nhãn của đỉnh
  int             Visited;    // Trạng thái
};

// Định nghĩa cấu trúc một cạnh
struct Edge
{
  int         Source;     // Đỉnh đầu
  int         Target;     // Đỉnh cuối
  CostType    Weight;     // Trọng số
  int         Marked;     // Trạng thái
};
```

18

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Định nghĩa cấu trúc một đoạn đường đi
struct Path
{
    CostType  Length;      // Độ dài đđi
    int       Parent;      // Đỉnh trước
};
// Định nghĩa kiểu dữ liệu đồ thị
struct      Graph
{
    bool      Directed;          // ĐT có hướng?
    int       NumVertices;       // Số đỉnh
    int       NumEdges;          // Số cạnh
    Vertex    Vertices[MAX];     // DS đỉnh
    MaTrix    Cost;              // MTrận kề
};
```

19

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Tạo một đỉnh có nhãn label
Vertex CreateVertex(LabelType lab)
{
    Vertex v;                    // Khai báo 1 đỉnh
    v.Label = lab;               // Gán nhãn cho đỉnh
    v.Visited = NO;              // Cho biết đỉnh chưa xét
    return v;
}

// Hiển thị thông tin đỉnh thứ pos trong đồ thị
void DisplayVertex(Graph g, int pos)
{
    cout << g.Vertices[pos].Label << "\t";
}

// Thiết lập lại trạng thái của các đỉnh là chưa xét
void ResetFlags(Graph &g)
{
    for (int i=0; i<g.NumVertices; i++)
        g.Vertices[i].Visited = NO;
}
```

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Thêm một đỉnh có nhãn lab vào đồ thị g
void AddVertex(Graph &g, LabelType lab)
{
        Vertex v; // Khai báo 1 đỉnh
        v = CreateVertex(lab)//Gọi hàm tạo một đỉnh v với nhãn lab
        g.Vertices[g.NumVertices]=v;// Đưa đỉnh v vào đồ thị g
        g.NumVertices++;// Tăng số đỉnh của g lên 1
}
// Thêm một cạnh có trọng số là weight bắt đầu từ đỉnh //start và
kết thúc tại đỉnh end, directed: loại đồ thị
void AddEdge(Graph &g, int start, int end, CostType weight, bool
directed)
{
        if (Hai đỉnh start, end không kề nhau)
                Tăng số cạnh lên 1 ;
                Gán chi phí đi từ start đến end = weight;
        if (Nếu đồ thị là vô hướng)
                Gán chi phí đi từ end đến start = weight;
}
```

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Thêm một cạnh có trọng số là weight bắt đầu
// từ đỉnh start và kết thúc tại đỉnh end
void AddEdge(Graph &g, int start, int end, CostType weight)
{
        AddEdge(g, start, end, weight, g.Directed);
}

// Thêm một cạnh dùng cho đồ thị không có trọng số.
void AddEdge(Graph &g, int start, int end)
{
        AddEdge(g, start, end, 1);
}
```

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Kiểm tra hai đỉnh start và end có cạnh nối?
int IsConnected(Graph g, int start, int end)
{
   if ((g.Cost[start][end] == 0) ||(g.Cost[start][end] == INF))
       return 0;
   else
       return 1;
}

// Tìm đỉnh đầu tiên kề với curr mà chưa xét
int FindFirstAdjacencyVertex(Graph g, int curr)
{
   for (int i=0; i<g.NumVertices; i++)
   {
       // Kiểm tra đỉnh đã xét chưa và kề với curr ko?
       if ((g.Vertices[i].Visited==NO) && IsConnected(g,curr,i))
              return i;             // Thỏa đk -> tìm thấy
   }
   return NULLDATA;                 // Không tìm thấy
}
```

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Khởi tạo một đồ thị. Directed=true: Đồ thị có hướng
Graph InitGraph(bool directed)
{
   Graph g;                          // Khai báo 1 biến Graph
   g.NumEdges = 0;                   // khởi tạo số cạnh = 0
   g.NumVertices = 0;                // Khởi tạo số đỉnh = 0
   g.Directed = directed;            // Đồ thị có hướng hay ko?

   // Khởi tạo ma trận kề (ma trận trọng số, chi phí)
   for (int i=0; i<MAX; i++)
       for (int j=0; j<MAX; j++)
              if (i == j)
                     g.Cost[i][j] = 0;
              else
                     g.Cost[i][j] = INF;
   return g;
}
```

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
// Đọc dữ liệu từ file để tạo đồ thị
void OpenGraph(Graph &g, char* fileName)
{
        // Khai báo biến và mở file để đọc
        ifstream is(fileName);
        // kiểm tra đã mở được file chưa?
        if (is.is_open())
        {
                int n = 0, m = 0;
                bool d = false;
                LabelType lab;
                is >> n; // Đọc số đỉnh của đồ thị
                is >> m; // Đọc số cạnh
                is >> d; // Đọc loại đồ thị
```

25

Cài đặt đồ thị biểu diễn bởi ma trận kề

```
...
                g = InitGraph(d); // Khởi tạo đồ thị
                g.NumEdges = m; // Gán số cạnh
                // Khởi tạo nhãn của các đỉnh
                for (int i=0; i<n; i++)
                {
                        is >> lab; // Đọc nhãn
                        AddVertex(g, lab); // Thêm đỉnh
                }
                // Đọc ma trận kề từ file
                for (int i=0; i<n; i++)
                        for (int j=0; j<n; j++)
                        {
                                // Và lưu vào đồ thị
                                is >> g.Cost[i][j];
                        }

                is.close(); // Đóng file
        }
}
```

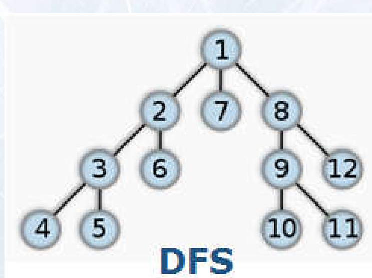
26

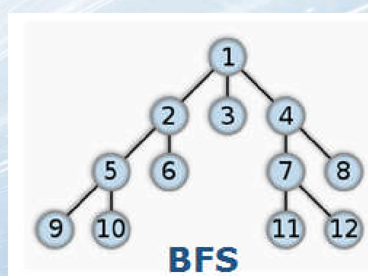
Nội dung chính

1. Các định nghĩa
2. Biểu diễn đồ thị
3. **Các phép duyệt đồ thị**
4. Một số bài toán trên đồ thị

Duyệt đồ thị

- Duyệt đồ thị là một thủ tục có hệ thống để khám phá đồ thị bằng cách kiểm tra tất cả các đỉnh và các cạnh của nó.
- Có hai cách duyệt đồ thị phổ biến
  - Duyệt đồ thị theo chiều sâu (Depth First Search - DFS)
  - Duyệt đồ thị theo chiều rộng (Breadth First Search - BFS)





Duyệt đồ thị theo chiều sâu

❖ Các bước thực hiện:

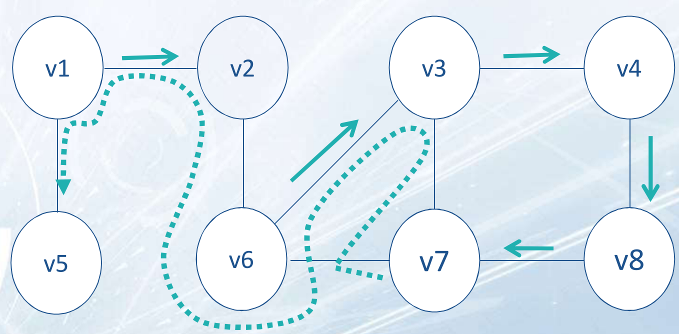
- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt
- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt
- B3. Xử lý đỉnh v
- B4. Với mỗi đỉnh w chưa được duyệt và kề với v, ta thực hiện đệ quy quá trình trên cho w.

❖ Một số ứng dụng

- Xử lý các đỉnh và các cạnh của đồ thị
- Xác định đồ thị có liên thông hay không
- Đếm số thành phần liên thông của đồ thị
- Tìm cây bao trùm
- Tìm đường đi giữa hai đỉnh

29

Thực hiện duyệt đồ thị theo chiều sâu trên đồ thị G dưới đây:

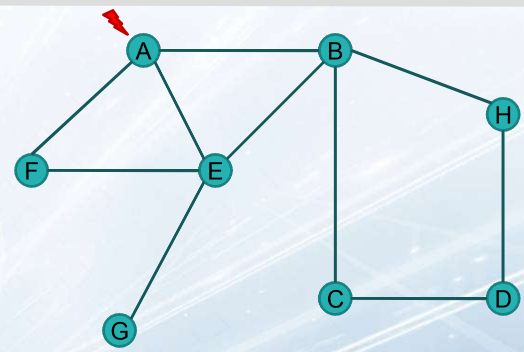


Bảng duyệt

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
v1	v2	v6	v3	v4	v8	v7	v5

Ví dụ 2 DFS

A Đỉnh chưa duyệt
A Đỉnh đã duyệt
A Đỉnh kề
Cạnh đang xét



Stack

Kết quả: A B C D H E F G

Duyệt đồ thị theo chiều sâu

```
// Đệ quy
void DFS(Vertex v)
{
   Process(v);
   v.Visited = true;
   foreach (w kề với v)
       if (!w.Visited)
            DFS(w);
}
void Traverse()
{
   foreach (w in V)
       w.Visited = false;
   foreach (w in V)
       if (!w.Visited)
            DFS(w);
}
```

```
void DFS(Vertex v)   // dạng lặp
{
   Stack s = CreateStack();
   Process(v);
   v.Visited = true;
   Push(s, v);
   while (s khác rỗng)
   {
       Vertex x = Pop(s);
       foreach (w kề với x)
         if (!w.Visited)
         {
              Process(w);
              w.Visited = true;
              Push(s, x);//Giu
   //lai dia chi quay lui
              Push(s, w);
              break;
         }
   }
}
```

Duyệt đồ thị theo chiều rộng

❖ Các bước thực hiện

- B1. Khởi gán tất cả các đỉnh đều chưa được duyệt
- B2. Xuất phát từ một đỉnh **v** bất kỳ của đồ thị, đánh dấu đỉnh v đã được duyệt. Đưa v vào hàng đợi.
- B3. Lấy x ra khỏi hàng đợi. Xử lý đỉnh x.
- B4. Với mỗi đỉnh w chưa được duyệt và kề với x, ta đánh dấu w đã được duyệt. Đưa w vào hàng đợi.
- B5. Quay lại B3.

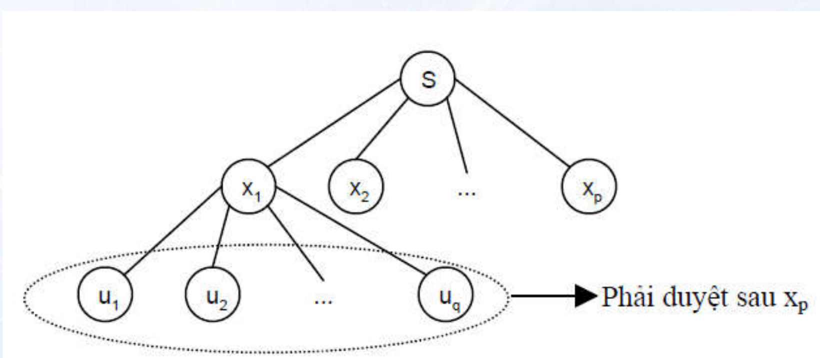
❖ Một số ứng dụng

- Xử lý các đỉnh và các cạnh của đồ thị
- Xác định đồ thị có liên thông hay không
- Đếm số thành phần liên thông của đồ thị
- Tìm cây bao trùm
- Tìm đường đi có số cạnh nhỏ nhất giữa hai đỉnh

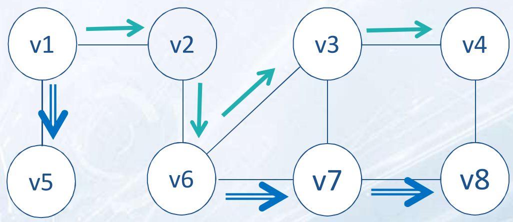
33

3. Duyệt đồ thị

3.3 Duyệt đồ thị theo chiều rộng



Thực hiện duyệt đồ thị theo chiều rộng trên đồ thị G dưới đây:

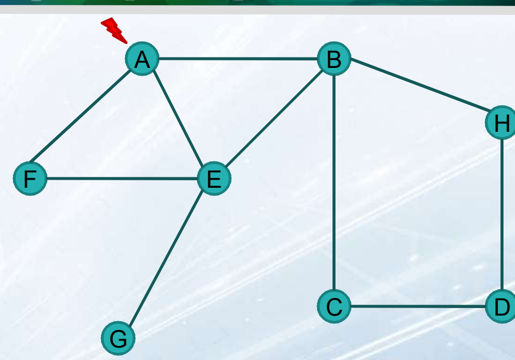


Bảng duyệt

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
v1	v2	v5	v6	v3	v7	v4	v8

Ví dụ 2 (BFS)

- (A) Đỉnh chưa duyệt
- (A) Đỉnh đã duyệt
- (A) Đỉnh kề
- Cạnh đang xét



Kết quả: A B E F C H G D

B E F C H G D

Queue

Duyệt đồ thị theo chiều rộng

```
// Lặp

void Traverse()
{
   foreach (w in V)
        w.Visited = false;

   foreach (w in V)
        if (!w.Visited)
              BFS(w);
}
```

```
void BFS(Vertex v)
{
   Queue q = CreateQueue();
   v.Visited = true;
   EnQueue(q, v);//q.push()
   while (q khác rỗng)
   {
        Vertex x = DeQueue(q);
        //q.pop()
        Process(x);
        foreach (w kề với x)
          if (!w.Visited)
          {
               w.Visited=true;
               EnQueue(s, w);
               //q.push()
          }
   }
}
```

37

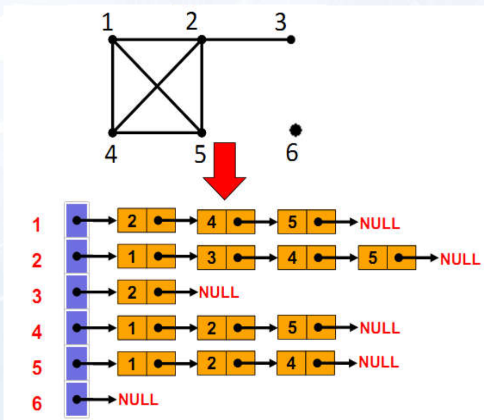
Bài tập về nhà

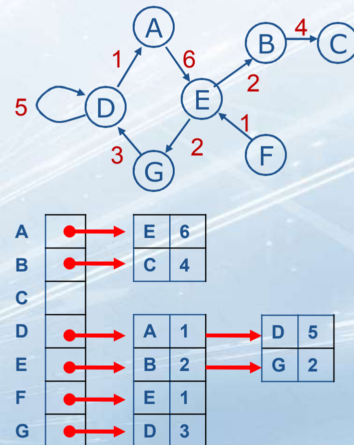
- Với mỗi cách biểu diễn đồ thị, hãy
  - Cài đặt hàm duyệt đồ thị theo chiều rộng
  - Cài đặt hàm duyệt đồ thị theo chiều sâu
    - Phương pháp lặp
    - Phương pháp đệ quy

38

Biểu Diễn Đồ Thị Bằng Danh sách kề

❖ Danh sách liền kề là một cách biểu diễn đồ thị bằng cách liệt kê tất cả các đỉnh nối với mỗi đỉnh của đồ thị





Cài đặt đồ thị biểu diễn bởi danh sách kề

```
// Định nghĩa hằng số
#define     TAB         '\t'   // Khoảng TAB
#define     EOL         '\n'   // Xuống dòng
#define     UPPER       100    // Số ptu tối đa
#define     ZERO        0      // Giá trị 0
#define     MAX         30     // Số đỉnh tối đa
#define     INF         1000   // Vô cùng
#define     YES         1      // Đã xét
#define     NO          0      // Chưa xét
#define     NULLDATA    -1     // Giả giá trị rỗng

// Định nghĩa các kiểu dữ liệu
typedef     char        LabelType;
Typedef     int         CostType;
```

Cài đặt đồ thị biểu diễn bởi danh sách kề

```
// Định nghĩa cấu trúc một cạnh
struct Edge
{
   int        Marked;      // Trạng thái
   int        Target;      // Đỉnh cuối
   CostType   Weight;      // Trọng số
   Edge*      Next;        // Cạnh tiếp
};
// Định nghĩa cấu trúc của một đỉnh
struct VertexPtr
{
   LabelType        Label;       // Nhãn của đỉnh
   int              Visited;     // Trạng thái
   Edge*            EdgeList;    // DS cạnh kề
};
```

41

Cài đặt đồ thị biểu diễn bởi danh sách kề

```
struct Path                             // Một đoạn đường đi
{
   CostType   Length;                   // Độ dài đđi
   int        Parent;                   // Đỉnh trước
};
typedef       Path*         PathPtr;
typedef       Edge*         EdgePtr;
typedef       Vertex*       VertexPtr;
// Định nghĩa kiểu dữ liệu đồ thị
struct GraphADT
{
   bool       Directed;                 // ĐT có hướng?
   int        NumVertices;              // Số đỉnh
   int        NumEdges;                 // Số cạnh
   VertexPtr  Vertices[MAX];            // DS kề
};
typedef       GraphADT*     Graph;
```

42

Cài đặt đồ thị biểu diễn bởi danh sách kề

```
// Tạo một cạnh có đỉnh cuối là target và trọng số là w.
EdgePtr       CreateEdge(int target, CostType w) {
   EdgePtr e = new Edge;               // Tạo biến động
   e->Target = target;                 // Lưu đỉnh đích
   e->Weight = w;                      // Lưu trọng số
   e->Marked = NO;                     // Đđấu chưa xét
   e->Next = NULL;
   return e;
}
// Tạo một đỉnh có nhãn label
VertexPtr CreateVertex(LabelType lab) {
   VertexPtr v = new Vertex;           // Khởi tạo một đỉnh
   v->Label = lab;                     // Gán nhãn cho đỉnh
   v->Visited = NO;                    // Cho biết đỉnh chưa xét
   v->EdgeList = NULL;
   return v;
}
```

Cài đặt đồ thị biểu diễn bởi danh sách kề

```
// Khởi tạo một đồ thị. directed = true: Đồ thị có hướng
Graph InitGraph(bool directed) {
   Graph g = new GraphADT;             // Khai báo 1 biến đồ thị
   g->NumEdges = 0;                    // khởi tạo số cạnh = 0
   g->NumVertices = 0;                 // Khởi tạo số đỉnh = 0
   g->Directed = directed;             // Đồ thị có hướng hay ko?
   // Khởi tạo danh sách đỉnh
   for (int i=0; i<MAX; i++) {
        g->Vertices[i] = NULL;
   }
   return g;
}
// Thiết lập lại trạng thái của các đỉnh
void ResetFlags(Graph &g) {
   for (int i=0; i<g->NumVertices; i++)
        g->Vertices[i]->Visited = NO;
}
```

Cài đặt đồ thị biểu diễn bởi danh sách kề

```
// Tìm cạnh nối 2 đỉnh start và end
EdgePtr FindEdge(Graph g, int start, int end) {
   // Bắt đầu tìm từ cạnh kề đầu tiên với start
   EdgePtr e = g->Vertices[start]->EdgeList;
   while (e != NULL)        // Duyệt qua các cạnh kề
   {
        // Nếu có một cạnh với đỉnh cuối là end
        if (e->Target == end) break;      // thì dừng tìm. Nếu ko
        e = e->Next;                      // thì sang cạnh tiếp
   }
   return e;
}
// Kiểm tra hai đỉnh start và end có cạnh nối với nhau
int IsConnected(Graph g, int start, int end) {
   EdgePtr e = FindEdge(g, start, end);
   return (e != NULL);
}
```

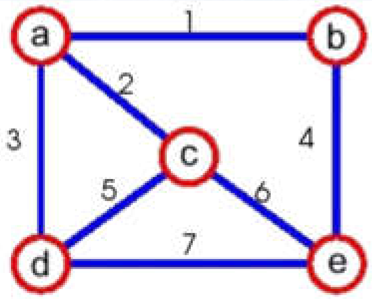
Nhận xét

- Ma trận kề
  - Ưu điểm
    - Có thể kiểm tra trực tiếp 2 đỉnh kề nhau hay không.
    - Trực quan, dễ cài đặt
  - Nhược điểm
    - Phải mất thời gian duyệt toàn bộ mảng để xác định tất cả các cạnh.
    - Tốn không gian lưu trữ
- Danh sách kề
  - Ưu điểm:
    - Tiết kiệm không gian lưu trữ trong trường hợp số đỉnh lớn, số cạnh ít (đồ thị thưa).
    - Dễ dàng tìm mọi đỉnh kề với một đỉnh cho trước
  - Nhược điểm:
    - Khó xác định (u, v) có phải là một cạnh hay không

Biểu diễn đồ thị bằng Ma trận liên thuộc (Incidence_matrix)

1. Nếu G là đồ thị vô hướng không có khuyên, ma trận liên thuộc (hay liên kết đỉnh cạnh) của đồ thị G, ký hiệu A(G), là ma trận *n*m* (n: số đỉnh, m: số cạnh) được định nghĩa là $A = (A_{ij})$ với quy ước:

- $A_{ij} = 1$ nếu đỉnh i kề với cạnh j;
- $A_{ij} = 0$ nếu ngược lại.



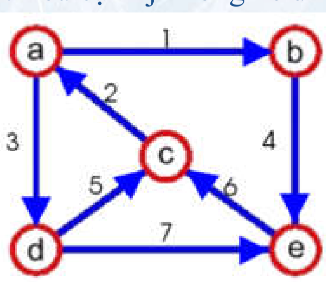
	1	2	3	4	5	6	7
a	1	1	1	0	0	0	0
b	1	0	0	1	0	0	0
c	0	1	0	0	1	1	0
d	0	0	1	0	1	0	1
e	0	0	0	1	0	1	1

47

Ma trận liên thuộc (Incidence_matrix)

2. Nếu G là đồ thị có hướng không có khuyên, ma trận liên thuộc (hay liên kết đỉnh cạnh) của đồ thị G, ký hiệu A(G), là ma trận *n*m* (n: số đỉnh, m: số cạnh) được định nghĩa là $A = (A_{ij})$ với quy ước:

- $A_{ij} = 1$ nếu cạnh j hướng ra khỏi đỉnh I;
- $A_{ij} = -1$ nếu cạnh j hướng vào đỉnh I;
- $A_{ij} = 0$ nếu cạnh j không kề đỉnh i.



	1	2	3	4	5	6	7
a	1	-1	1	0	0	0	0
b	-1	0	0	1	0	0	0
c	0	1	0	0	-1	-1	0
d	0	0	-1	0	1	0	1
e	0	0	0	-1	0	1	-1

48

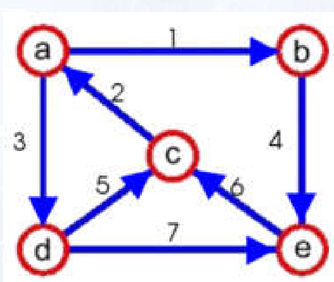
Biểu diễn đồ thị bằng danh sách cạnh (adjacency list)

- Trong trường hợp đồ thị thưa (đồ thị có số cạnh m thoả mãn bất dẳng thức: m<6n) người ta thường dùng cách biểu diễn đồ thị dưới dạng danh sách cạnh.
- Trong cách biểu diễn đồ thị bởi danh sách cạnh (cung) chúng ta sẽ lưu trữ danh sách tất cả các cạnh (cung) của đồ thị vô hướng (có hướng). Một cạnh (cung) e=(x,y) của đồ thị sẽ tương ứng với hai biến Dau[e], Cuoi[e]. như vậy, để lưu trữ đồ thị ta cần sử dụng 2m đơn vị bộ nhớ. Nhược điểm của cách biểu diễn này là để xác định những đỉnh nào của đồ thị là kề với một đỉnh cho trước chúng ta phải làm cỡ m phép so sánh (khi duyệt qua danh sách tất cả các cạnh của đồ thị).
- *Chú ý*: Trong trường hợp đồ thị có trọng số ta cần thêm m đơn vị bộ nhớ để lưu trữ trọng số của các cạnh.

49

Biểu diễn đồ thị bằng danh sách cạnh (adjacency list)

- Một cạnh (cung) e=(x,y) của đồ thị sẽ tương ứng với hai biến Dau[e], Cuoi[e]. Như vậy, để lưu trữ đồ thị ta cần sử dụng 2m đơn vị bộ nhớ



Đầu	Cuối
a	b
a	d
b	e
c	a
d	c
d	e
e	c

50

Biểu diễn đồ thị

- Bài tập về nhà
- Bài 1: Với mỗi cách biểu diễn đồ thị, hãy cài đặt các hàm sau
  - Thêm một đỉnh có nhãn label vào đồ thị
  - Thêm một cạnh nối 2 đỉnh start, end trong 2 trường hợp
    - Có trọng số
    - Không có trọng số
  - Tìm cạnh nối đỉnh curr với đỉnh kề (chưa xét) đầu tiên
  - Tìm đỉnh chưa được xét đầu tiên kề với đỉnh curr
  - Lưu đồ thị vào file
  - Tạo đồ thị với dữ liệu lấy từ file

51

Một số cách biểu diễn khác

- Bài 2
  - Tìm hiểu cách biểu diễn ma trận bằng:
    - Ma trận liên thuộc
    - Danh sách cạnh
  - Cài đặt kiểu dữ liệu đồ thị với 2 cách biểu diễn trên.

52

Nội dung chính

1. Các định nghĩa
2. Biểu diễn đồ thị
3. Các phép duyệt đồ thị
4. **Một số bài toán trên đồ thị**

53

Một số bài toán trên đồ thị

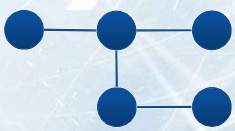
- Tìm đường đi ngắn nhất từ một đỉnh của đồ thị
  - Thuật toán Dijkstra
- Tìm cây bao trùm tối thiểu (MST = Minimum Cost Spanning Tree)
  - Thuật toán Kruskal
  - Thuật toán Prim

54

Bài toán tìm cây bao trùm nhỏ nhất

- Đồ thị liên thông là đồ thị G=(V, E) trong đó luôn có ít nhất một đường đi giữa hai cặp đỉnh bất kỳ.
- Nếu đồ thị không liên thông thì nó sẽ là hợp của hai hay nhiều đồ thị con liên thông.
- Các đồ thị con từng đôi một rời nhau gọi là các thành phần liên thông.

Đồ thị liên thông



Đồ thị không liên thông

55

Bài toán tìm cây bao trùm nhỏ nhất

- 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 (cây khung)
  - Nếu thỏa (1), (2), (3) thì T gọi là cây bao trùm tối thiểu.

G T_1 56 T_2 T_3

Bài toán tìm cây bao trùm nhỏ nhất

❖ Thuật toán Kruskal

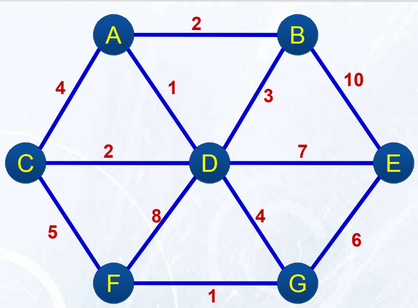
- Input: G = (V, E)
- Output: T = (V, F) nhỏ nhất

❖ Ý tưởng

- Khởi tạo cây T không có cạnh nào (F = Ø), chỉ gồm n đỉnh
- Sắp xếp các cạnh của G tăng dần theo trọng số
- Lần lượt xét từng cạnh (u,v) từ trọng số nhỏ nhất đến lớn nhất
- Thêm cạnh (u, v) vào F nếu không tạo thành chu trình
- Lặp lại bước trên cho đến khi đủ n-1 cạnh hoặc mọi cạnh còn lại đều tạo thành chu trình (đồ thị G là không liên thông).
- Kết thúc: T=(V, F) là cây bao trùm tối thiểu

57

Minh họa thuật toán Kruskal

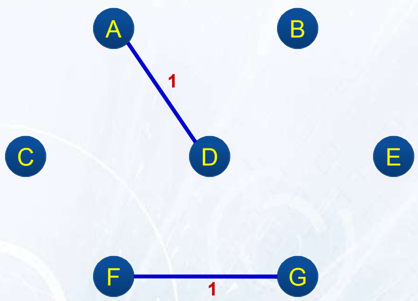


Đỉnh	A	B	C	D	E	F	G
Gốc	A	B	C	A	E	F	G

Nguồn	Đích	Trọng số
A	B	2
A	G	4
A	B	2
B	D	3
B	B	10
A	D	2
D	G	4
D	E	7
D	G	8
D	G	7
D	G	8
B	G	10

58

Minh họa thuật toán Kruskal

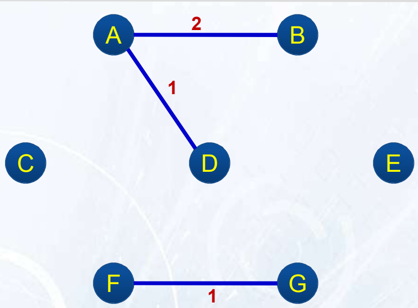


Đỉnh	A	B	C	D	E	F	G
Gốc	A	B	C	A	E	F	G

Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

59

Minh họa thuật toán Kruskal

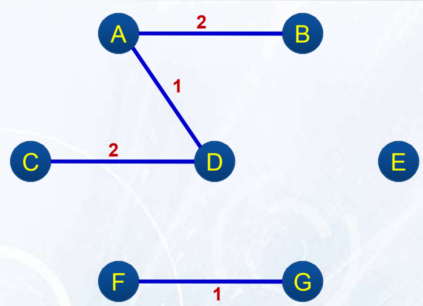


Đỉnh	A	B	C	D	E	F	G
Gốc	A	B	C	A	E	F	F

Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

60

Minh họa thuật toán Kruskal

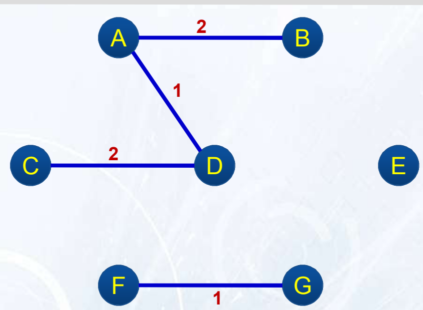


Đỉnh	A	B	C	D	E	F	G
Gốc	A	A	A	A	E	F	F

Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

61

Minh họa thuật toán Kruskal

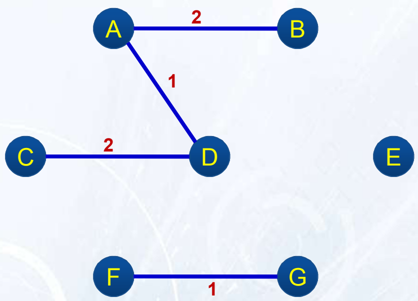


Đỉnh	A	B	C	D	E	F	G
Gốc	A	A	A	A	E	F	F

Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

62

Minh họa thuật toán Kruskal

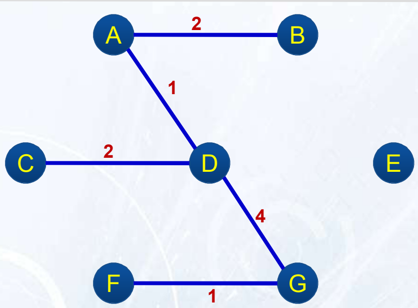


Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

Đỉnh	A	B	C	D	E	F	G
Gốc	A	A	A	A	E	F	F

63

Minh họa thuật toán Kruskal

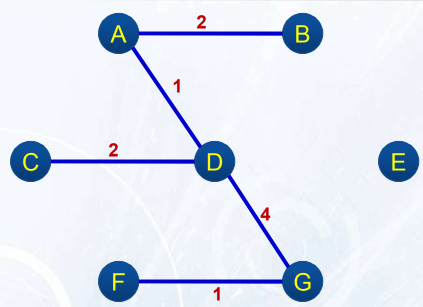


Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

Đỉnh	A	B	C	D	E	F	G
Gốc	A	A	A	A	E	~~F~~ A	F

64

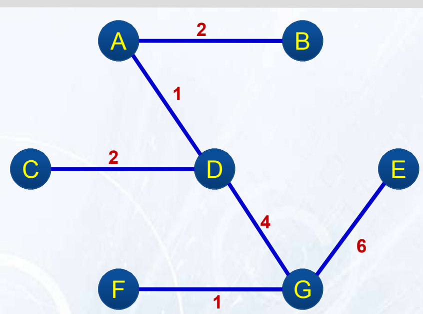
Minh họa thuật toán Kruskal



Đỉnh	A	B	C	D	E	F	G
Gốc	A	A	A	A	E	A	F

Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

Minh họa thuật toán Kruskal



Đỉnh	A	B	C	D	E	F	G
Gốc	A	A	A	A	A	A	A

Nguồn	Đích	Trọng số
A	D	1
F	G	1
A	B	2
C	D	2
B	D	3
A	C	4
D	G	4
C	F	5
E	G	6
D	E	7
D	F	8
B	E	10

Hai vấn đề của thuật toán Kruskal

❖ Thứ nhất, làm thế nào để xét được các cạnh từ cạnh có trọng số nhỏ tới cạnh có trọng số lớn?

Nên sử dụng các thuật toán sắp xếp hiệu quả để đạt được tốc độ nhanh trong trường hợp số cạnh lớn. Trong trường hợp tổng quát, thuật toán HeapSort là hiệu quả nhất bởi nó cho phép chọn lần lượt các cạnh từ cạnh trọng nhỏ nhất tới cạnh trọng số lớn nhất ra khỏi Heap và có thể xử lý (bỏ qua hay thêm vào) luôn.

67

Hai vấn đề của thuật toán Kruskal

❖ Thứ hai, làm thế nào kiểm tra xem việc thêm một cạnh có tạo thành chu trình đơn trong T hay không?

Muốn thêm một cạnh (u, v) vào T mà không tạo thành chu trình đơn thì (u, v) phải nối hai cây khác nhau của rừng T, bởi nếu u, v thuộc cùng một cây thì sẽ tạo thành chu trình đơn trong cây đó. Ban đầu, ta khởi tạo rừng T gồm n cây, mỗi cây chỉ gồm đúng một đỉnh, sau đó, mỗi khi xét đến cạnh nối hai cây khác nhau của rừng T thì ta kết nạp cạnh đó vào T, đồng thời hợp nhất hai cây đó lại thành một cây.

Vậy để kiểm tra một cạnh (u, v) có nối hai cây khác nhau của rừng T hay không? ta có thể kiểm tra Leader(u) có khác Leader(v) hay không, bởi mỗi cây chỉ có duy nhất một gốc.

68

Mô hình của thuật toán Kruskal

❖ Mô hình thuật toán Kruskal:

for ∀k∈ V do Lab[k] := -1;
for ∀(u, v) ∈ E (theo thứ tự từ cạnh trọng số nhỏ tới cạnh trọng số lớn)
{
    r1 := Leader(u); r2 := Leader(v);
    if r1 ≠ r2 then {(u, v) nối hai cây khác nhau}
    {
        <Kết nạp (u, v) vào cây, nếu đã đủ n - 1 cạnh thì thuật toán dừng>
        Union(r1, r2); {Hợp nhất hai cây lại thành một cây}
    };
};

Thuật toán Kruskal

MST-Kruskal(G,l){

1. Xếp E theo l tăng (trọng số tăng dần); n = # V; T = Ø;
2. Đặt n tplt, mỗi tplt chứa 1 phần tử của V;
3. **do**{
4. (u,v) cạnh độ dài min chưa xét đến;
5. **if** *Leader*(u)<> *Leader* (v) {
6. T = T U (u,v); *union*(*set*(u),*set*(v));
7. }
8. } **while**(#T = n-1)
9. **return** T;
10. }

Thuật toán Kruskal

```
int Find(int leader[MAX], int x)
{
   while (x != leader[x])
       x = leader[x];
   return x;
}
bool Union(int leader[MAX], Edge e)
{
   int x = Find(leader, e.Source);
   int y = Find(leader, e.Target);
   if (x == y)       return false;
   else if (x < y)          // Nhập chung cây
       leader[y] = x;       // y vào cây x
   else
       leader[x] = y;       // Nhập cây x vào y
   return true;
}
```

71

Thuật toán Kruskal

```
void Kruskal(Graph g, Edge tree[UPPER]) {
   tree = GetEdgeList(g);
   QSortEdges(tree, 0, ne-1);

   int leader[MAX];
   for (int i=0; i<g.NumVertices; i++)
       leader[i] = i;

   int count = 0;
   foreach (Cạnh e trong tree)     {
       if (Union(leader, e)) {
            e.Marked = YES;
            count++;
            if (count == g.NumVertices - 1)   break;
       }
   }
}
```

72

Prim vs. Kruskal

	Kruskal	Prim
Bộ nhớ	n + m	n^2
Thời gian	O(m * (logm + logn))	$O(n^2)$
Áp dụng	Phù hợp với đồ thị thưa (đồ thị có số cạnh nhỏ hơn rất nhiều so với số đỉnh)	Thuật toán tốt nhất cho đồ thị đầy đủ

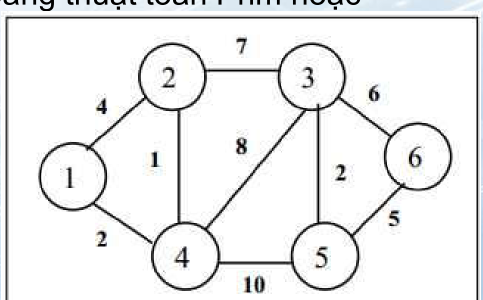
73

Kiểm tra

I. Nếu thỏa điều kiện nào thì đồ thị G được gọi là cây bao trùm tối thiểu?

II. Cho đồ thị G như hình

1. Hãy biểu diễn đồ thị G bằng ma trận kề.
2. Xác định đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại.
3. Tìm cây bao trùm tối thiểu bằng thuật toán Prim hoặc Kruskal từ đỉnh 1.



74

Đường đi ngắn nhất từ đỉnh s = 1 đến các đỉnh còn lại:

Bước 0:

S = {1};

d(1,1) = L(1) = 0;

L(s) = ∞; ∀s≠1

Bước 1:

- Tính nhãn tạm thời L(v), v∉ S:

* Các đỉnh ∉ S và kề với 1 là 2,4:

L(2) = Min{ L(2), L(1)+m(1,2)} = Min{∞, 4} = 4.

L(4) = Min{ L(3), L(1)+m(1,4)} = 2.

* Các đỉnh ∉ S và không kề với 1 là 4,6:

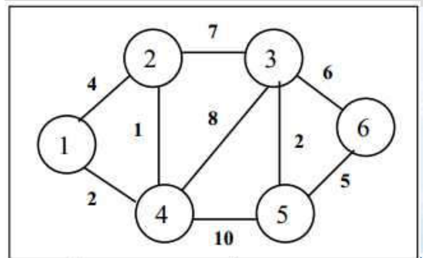
L(3) = L(5) = L(6) = ∞.

- Tìm s_1 ∉ S và kề với 1 sao cho:L(s_1) = Min{L(v): ∀v ∉ S };

L(4) = Min{L(v): ∀v ∉ S} = L(1)+m(1,4) = 2.

Đường đi từ 1 đến 4 xác định bởi: 1 → 4 là ngắn nhất trong tất cả các đường đi từ 1 đến các đỉnh khác và d(1,4) = L(4) = 2.

- S = S ∪ {4}; // S = {1,4}



	1	2	3	4	5	6
1	0	4	VC	2	VC	VC
2	4	0	7	1	VC	VC
3	VC	7	0	8	2	6
4	2	1	8	0	10	VC
5	VC	VC	2	10	0	5
6	VC	VC	6	VC	5	0

Ma trận kề

	1	2	3	4	5	6
1	0	4	VC	2	VC	VC
2	4	0	7	1	VC	VC
3	VC	7	0	8	2	6
4	2	1	8	0	10	VC
5	VC	VC	2	10	0	5
6	VC	VC	6	VC	5	0

Bước Lặp	Đường đi ngắn nhất là đường đi từ đỉnh 1	đến đỉnh	Chiều dài của đường đi ngắn nhất từ đỉnh s (=1) đến các đỉnh khác: tsnn[]					
			1	2	3	4	5	6
Bước 1	1→ 4	4	-	4	∞	2	∞	∞
Bước 2	1→ 4→ 2	2	-	3	10	-	12	∞
Bước 3	1→ 4→ 3	3	-	-	10	-	12	∞
Bước 4	1→ 4→5	5	-	-	-	-	12	16
Bước 5	1→ 4→ 3→ 6	6	-	-	-	-	-	16

Ma trận kề

	1	2	3	4	5	6
1	0	4	VC	2	VC	VC
2	4	0	7	1	VC	VC
3	VC	7	0	8	2	6
4	2	1	8	0	10	VC
5	VC	VC	2	10	0	5
6	VC	VC	6	VC	5	0

Khởi tạo

	2	3	4	5	6
lowcost	4	VC	2	VC	VC
closest	1	1	1	1	1
mark	0	0	0	0	0

Bước 1
Min = 2
K = 4

	2	3	4	5	6
lowcost	1	8	2	10	VC
closest	4	4	1	4	1
mark	0	0	1	0	0

Bước 2
Min = 1
K = 2

	2	3	4	5	6
lowcost	1	7	2	10	VC
closest	4	2	1	4	1
mark	1	0	1	0	0

Bước 3
Min = 7
K = 3

	2	3	4	5	6
lowcost	1	7	2	2	6
closest	4	2	1	3	3
mark	1	1	1	0	0

Bước 4
Min = 2
K = 5

	2	3	4	5	6
lowcost	1	7	2	2	5
closest	4	2	1	3	5
mark	1	1	1	1	0

Bước 5
Min = 5
K = 6

	2	3	4	5	6
lowcost	1	7	2	2	5
closest	4	2	1	3	5
mark	1	1	1	1	1

77

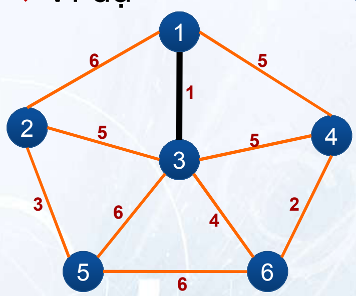
Bài toán tìm cây bao trùm tối thiểu

- ❖ Thuật toán Prim
  - Input: G = (V, E)
  - Output: T = (V, F) nhỏ nhất
- ❖ Mô tả
  - U: Tập các đỉnh được chọn (xét)
  - F: Tập các cạnh được chọn
  - Khởi tạo U = Ø, F = Ø
  - Chọn một đỉnh u bất kỳ làm gốc của cây bao trùm
  - Đưa u vào U
  - Trong khi U khác V
    - Chọn cạnh (u, v) nhỏ nhất sao cho u ϵ U, v ϵ V – U
    - Thêm v vào U
    - Thêm (u, v) vào F
  - Kết thúc: T=(V, F) là cây bao trùm tối thiểu

78

Minh họa thuật toán Prim

❖ Ví dụ



Ma trận kề

	1	2	3	4	5	6
1	0	6	1	5	VC	VC
2	6	0	5	VC	3	VC
3	1	5	0	5	6	4
4	5	VC	5	0	VC	2
5	VC	3	6	VC	0	6
6	VC	VC	4	2	6	0

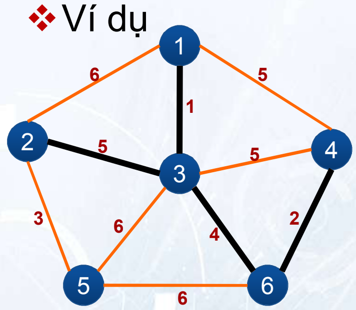
Khởi tạo

	2	3	4	5	6
lowcost	6	1	5	VC	VC
closest	1	1	1	1	1
mark	0	0	0	0	0

79

Minh họa thuật toán Prim

❖ Ví dụ



	2	3	4	5	6
lowcost	6	1	5	VC	VC
closest	1	1	1	1	1
mark	0	0	0	0	0

Bước 1
Min = 1
K = 3

	2	3	4	5	6
lowcost	5	1	5	6	4
closest	3	1	1	3	3
mark	0	1	0	0	0

Bước 2
Min = 4
K = 6

	2	3	4	5	6
lowcost	5	1	2	6	4
closest	3	1	6	3	3
mark	0	1	0	0	1

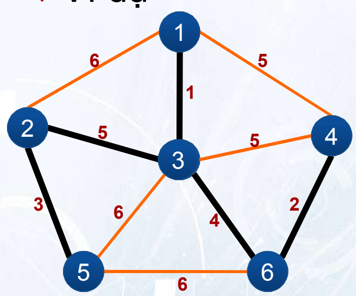
Bước 3
Min = 2
K = 4

	2	3	4	5	6
lowcost	5	1	2	6	4
closest	3	1	6	3	3
mark	0	1	1	0	1

80

Minh họa thuật toán Prim

❖ Ví dụ



Bước 4
Min = 5
K = 2

	2	3	4	5	6
lowcost	5	1	2	3	4
closest	3	1	6	2	3
mark	1	1	1	0	1

Bước 5

	2	3	4	5	6
lowcost	5	1	2	3	4
closest	3	1	6	2	3
mark	1	1	1	1	1

	2	3	4	5	6
lowcost	5	1	2	6	4
closest	3	1	6	3	3
mark	0	1	1	0	1

2), (6, 3)

+ 4 + 2 + 5 + 3 = 15

81

Thuật toán Prim

❖ Ký hiệu

- C[n,n]: Ma trận kề biểu diễn đồ thị G
- Closest[v]: là đỉnh thuộc U và gần với v nhất, v ϵ V – U
- Lowcost[v]: lưu trọng số của cạnh (v, closest[v])
- Mark[v]: Đánh dấu đỉnh v đã được xét hay chưa

❖ Ý tưởng

- Tại mỗi bước, duyệt mảng lowcost để tìm đỉnh closest[v] ϵ U sao cho lowcost[v] = (v, closest[v]) là nhỏ nhất.
- Chọn và lưu lại (hoặc xuất) cạnh (v, closest[v])
- Cập nhật các mảng closest và lowcost.
- Thêm v vào U bằng cách gán Mark[v] = true;

82

Cài đặt thuật toán Prim

```
#define      VC      ......           // Vô cùng = 1000
#define      DX      ......           // Đã xét = 1
…
void Prim(MaTran C)          // Bắt đầu từ đỉnh 0 (đỉnh đầu tiên)
{                            // làm nút gốc của cây bao trùm
   double     lowcost[MAX];
   int closest[MAX],mark[MAX];
   int i, j, k, min;

   for (i=2; i<=n; i++)               // Khởi tạo các mảng
   {
       lowcost[i] = C[1, i ];      // Chi phí từ đỉnh 1 -> i
       closest[i] = 1;       // Đỉnh gần nhất với i là đỉnh 0
       mark[i] = 0;          // Đỉnh i chưa được xét
   }
   …
```


Cài đặt thuật toán Prim

```
  for (i=2; i<=n; i++)  { // Tìm đỉnh k sao cho cạnh (k,
     min = lowcost[2];    // closest[k]) có trọng số nhỏ nhất
     k = 2;
     for (j=3; j<=n; j++){
        if (mark[ j ] == 0 && lowcost[ j ] < min)   {
                  min = lowcost[ j ];
                  k = j;
        }
     }
     mark[k] = DX;
     for (j=2; j<=n; j++)// Khởi động lại closest và lowcost
           if (mark[ j ] == 0 && C[k, j] < lowcost[ j ]) {
                  lowcost[ j ] = C[k, j];
                  closet[ j ] = k;
           }
  }
}
```

(Tìm đường đi ngắn nhất trong đồ thị có trọng số)

❖ Phát biểu bài toán

Cho G = (V,E) là đơn đồ thị liên thông (vô hướng hoặc có hướng) có trọng số, V = {1,.., n} là tập các đỉnh, E là tập các cạnh (cung).

Cho $s_0 \in V$. Tìm đường đi ngắn nhất đi từ s_0 đến các đỉnh còn lại.

Giải bài toán trên bằng thuật toán Dijkstra .

Ý tưởng

- ❖ Thuật toán Dijkstra cho phép tìm đường đi ngắn nhất từ một đỉnh s_0 đến các đỉnh còn lại của đồ thị với chiều dài (trọng số) tương ứng.
- ❖ Phương pháp của thuật toán là xác định tuần tự đỉnh có chiều dài đến s_0 theo thứ tự tăng dần.
- ❖ Thuật toán được xây dựng trên cơ sở gán cho mỗi đỉnh các nhãn tạm thời. Nhãn tạm thời của các đỉnh cho biết cận trên của chiều dài đường đi ngắn nhất từ s_0 đến đỉnh đó. Nhãn của các đỉnh sẽ biến đổi trong các bước lặp, mà ở mỗi bước lặp sẽ có một nhãn tạm thời trở thành chính thức. Nếu nhãn của một đỉnh nào đó trở thành chính thức thì đó cũng chính là chiều dài ngắn nhất của đường đi từ s_0 đến đỉnh đó.

Mô tả thuật toán:

Ký hiệu:

- L(v): để chỉ nhãn của đỉnh v, tức là cận trên của chiều dài đường đi ngắn nhất từ s_0 đến v.
- $d(s_0,v)$: chiều dài đường đi ngắn nhất từ s_0 đến v.
- $m(s_0,v)$: là trọng số của cung (cạnh) (s_0,v).

Thuật toán Dijkstra tìm chiều dài đường đi ngắn nhất từ đỉnh s_0 đến n-1 đỉnh còn lại được mô tả như sau:

- Input: G, s_0
- Output: $d(s_0,v)$, $\forall v \neq s_0$;

Mô tả:

- **Bước 0 (Khởi động):**
  $S = \{s_0\}$; // s_0 có nhãn chính thức
  $d(s_0,s_0) = L(s_0) = 0$;
  $L(v) = \infty$, $\forall v \neq s_0$; //Nhãn tạm thời
- **Bước 1:**
  - Tính lại nhãn tạm thời L(v), $v \notin S$:
    Nếu v kề với s_0 thì
    $L(v) = Min\{L(v), L(s_0) + m(s_0,v)\}$;
  - Tìm $s_1 \notin S$ và kề với s_0 sao cho:
    $L(s_1) = Min\{L(v): \forall v \notin S, \}$;
    (Khi đó: $d(s_0,s_1) = L(s_1)$)
  - $S = S \cup \{s_1\}$; // $S = \{s_0, s_1\}$; s1 có nhãn chính thức

- **Bước 2:**

  - Tính lại nhãn tạm thời $L(v)$, $v \notin S$:

    Nếu v kề với s_1 thì

    $L(v) = Min\{L(v), L(s_1) + m(s_1,v)\}$;

  - Tìm $s_2 \notin S$ và kề với s_1 hoặc s_0 sao cho:

    $L(s_2) = Min\{L(v): \forall v \notin S\}$;

    (Khi đó: $d(s_0,s_2) = L(s_2)$); //$0 \le d(s_0,s_1) \le d(s_0,s_2)$

    Nếu $L(s_2) = Min\{L(s_2), L(s_1) + m(s_1,s_2)\} = L(s_1) + m(s_1,s_2)$

  thì đường đi từ s đến s_2 đi qua đỉnh s_1 là ngắn nhất, và s_1 là đỉnh đứng kề trước s_2.

  - $S = S \cup \{s_2\}$; // S = {s0, s1, s2 } ;

  . . .

- **Bước i-1:**

  Có: Tính lại nhãn tạm thời $L(v)$, $v \notin S$:

    Nếu v kề với s_{i-2} thì

    $L(v) = Min\{L(v), L(s_{i-2}) + m(s_{i-2},v)\}$;

  Tìm $s_{i-1} \notin S$ và kề với s_{i-2} hoặc $s_0...s_{i-3}$ sao cho:

    $L(s_{i-1}) = Min\{L(v): \forall v \notin S\}$;

    Khi đó: $d(s_0,s_{i-1}) = L(s_{i-1})$); //$0 \le d(s_0,s_1) \le d(s_0,s_2) \le \ldots \le d(s_0,s_{i-1})$

  - $S = S \cup \{s_{i-1}\}$; // $S = \{s_0, \ldots, s_{i-1}\}$; //s_{i-1} có nhãn chính thức

  . . .

- **Bước i:**

  - Tính lại nhãn tạm thời $L(v)$, $v \notin S$:

    Nếu v kề với s_{i-1} thì

    $L(v) = Min\{L(v), L(s_{i-1}) + m(s_{i-1},v)\}$;

  - Tìm $s_i \notin S$ và kề với s_j, $j \in \{0,\ldots,i-1\}$ sao cho:

    $L(s_i) = Min\{L(v): v \notin S\}$; //$d(s,s_i) = L(s_i)$

    //$0 \le d(s_0,s_1) \le d(s_0,s_2) \le . . \le d(s_0,s_i)$

  Nếu $L(s_i) = Min\{L(s_i), L(s_j) + m(s_j,s_i)\} = L(s_j) + m(s_j,s_i)$

  thì đường đi ngắn nhất từ s đến s_i đi qua đỉnh s_j, và s_j là đỉnh đứng kề trước s_i.

  - $S = S \cup \{s_i\}$; // $S = \{s_0,s_1,..,s_i\}$; //s_i có nhãn chính thức

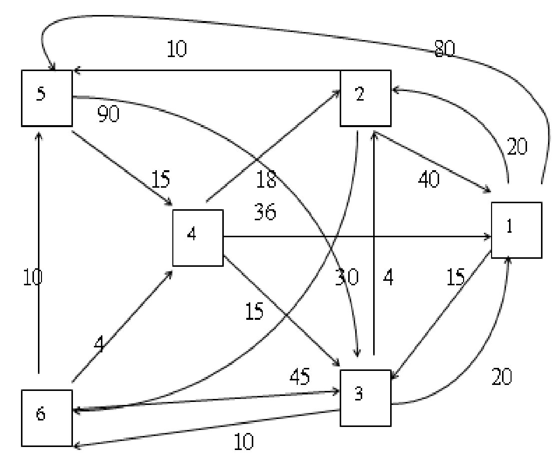
  Thuật toán dừng khi i = n-1;

  Khi thuật toán kết thúc, ta có:

  $0 = d(s_0,s_0) \le d(s_0,s_1) \le d(s_0,s_2) \le . . \le d(s_0,s_{n-1})$

Nếu chỉ tìm đường đi ngắn nhất từ s_0 đến t, thì thuật toán dừng khi có $t \in S$.

Minh hoạ: Xét đồ thị có hướng G



$$\begin{bmatrix} \infty & 20 & 15 & \infty & 80 & \infty \\ 40 & \infty & \infty & \infty & 10 & 30 \\ 20 & 4 & \infty & \infty & \infty & 10 \\ 36 & 18 & 15 & \infty & \infty & \infty \\ \infty & \infty & 90 & 15 & \infty & \infty \\ \infty & \infty & 45 & 4 & 10 & \infty \end{bmatrix}$$

Đường đi ngắn nhất từ đỉnh s = 1 đến các đỉnh còn lại:

Bước 0:

S = {1};

d(1,1) = L(1) = 0;

L(s) = ∞; $\forall s \neq 1$

Bước 1:

- Tính nhãn tạm thời L(v), $v \notin S$:
  - * Các đỉnh $\notin$ S và kề với 1 là 2,3,5:
  - L(2) = Min{ L(2), L(1)+m(1,2)} = Min{∞, 20} = 20.
  - L(3) = Min{ L(3), L(1)+m(1,3)} = 15.
  - L(5) = Min{ L(5), L(1)+m(1,5)} = 80.
  - * Các đỉnh $\notin$ S và không kề với 1 là 4,6:
  - L(4) = L(6) = ∞.

- Tìm $s_1 \notin$ S và kề với 1 sao cho:L(s_1) = Min{L(v): $\forall v \notin$ S };

L(3) = Min{L(v): $\forall v \notin$ S} = 15 = L(1)+m(1,3).

Đường đi từ 1 đến 3 xác định bởi: 1 $\rightarrow$ 3 là ngắn nhất trong tất cả các đường đi từ 1 đến các đỉnh khác và d(1,3) = L(3) = 15.

- S = S $\cup$ {3}; // S = {1,3}

$$\begin{bmatrix} \infty & 20 & 15 & \infty & 80 & \infty \\ 40 & \infty & \infty & \infty & 10 & 30 \\ 20 & 4 & \infty & \infty & \infty & 10 \\ 36 & 18 & 15 & \infty & \infty & \infty \\ \infty & \infty & 90 & 15 & \infty & \infty \\ \infty & \infty & 45 & 4 & 10 & \infty \end{bmatrix}$$

Bước 2:

- Tính nhãn tạm thời L(v), v$\notin$ S (S = {1,3}):
  * Các đỉnh $\notin$ S và kề với 3 là 2, 6:

L(2) = Min{ L(2), L(3)+m(3,2)} = Min{ 20, 15+4} = 19.

L(6) = Min{ L(6), L(3)+m(3,6)} = Min{ ∞, 15+10} = 25.

L(4) = ∞.

L(5) = 80 //Đã tính ở bước 1.

Nhãn tạm thời sau bước 1					
L(1)	(L2)	L(3)	L(4)	L(5)	L(6)
0	20	15	∞	80	∞

Nhãn tạm thời sau bước 2					
L(1)	(L2)	L(3)	L(4)	L(5)	L(6)
0	19	15	∞	80	25

- Tìm $s_2 \notin$ S và kề với 1 hoặc 3 sao cho:

L(s_2) = Min{L(v): $\forall$v $\notin$ S };

L(2) = Min{L(v): $\forall$v $\notin$ S} = Min{ L(2), L(3)+m(3,2)}

= L(3)+m(3,2) = 15 + 4 = 19.

Đường đi từ 1 đến 2 xác định bởi: 1 $\rightarrow$ 3$\rightarrow$ 2 là ngắn nhất trong tất cả các đường đi từ 1 đến các đỉnh j $\neq$ 3 và:

d(1,2) = L(2) = 19.

$$\begin{bmatrix} \infty & 20 & 15 & \infty & 80 & \infty \\ 40 & \infty & \infty & \infty & 10 & 30 \\ 20 & 4 & \infty & \infty & \infty & 10 \\ 36 & 18 & 15 & \infty & \infty & \infty \\ \infty & \infty & 90 & 15 & \infty & \infty \\ \infty & \infty & 45 & 4 & 10 & \infty \end{bmatrix}$$

- S = S $\cup$ {2}; // S = {1,3,2}

. . . tương tự, ta có kết quả tính toán sau đây:

$$\begin{bmatrix} \infty & 20 & 15 & \infty & 80 & \infty \\ 40 & \infty & \infty & \infty & 10 & 30 \\ 20 & 4 & \infty & \infty & \infty & 10 \\ 36 & 18 & 15 & \infty & \infty & \infty \\ \infty & \infty & 90 & 15 & \infty & \infty \\ \infty & \infty & 45 & 4 & 10 & \infty \end{bmatrix}$$

Bước Lặp	Đường đi ngắn nhất là đường đi từ đỉnh 1	đến đỉnh	Chiều dài của đường đi ngắn nhất từ đỉnh s (=1) đến các đỉnh khác: tsnn[]					
			1	2	3	4	5	6
Bước 1	1$\rightarrow$ 3	3	-	20	15	∞	80	∞
Bước 2	1$\rightarrow$3$\rightarrow$ 2	2	-	19	-			25
Bước 3	1$\rightarrow$3$\rightarrow$ 6	6	-	-	-		29	25
Bước 4	1$\rightarrow$3$\rightarrow$ 6$\rightarrow$4	4	-	-	-	29		-
Bước 5	1$\rightarrow$3$\rightarrow$ 2$\rightarrow$5	5	-	-	-	-	29	-

5.3.4 Cài đặt:

Ta biểu diễn đơn đồ thị có hướng G bằng ma trận các trọng số của cạnh: a= $(a_{uv})_{nxn}$;

trong đó:
$$a_{uv} = \begin{cases} \text{trọng số của } (u,v); (u,v) \in E; \\ \infty; (u,v) \notin E; \end{cases}$$

- Dùng mảng 1 chiều để lưu trữ các nhãn tạm thời của các đỉnh, ký hiệu là L[ ].
- Dùng mảng 1 chiều Daxet[ ] chứa các giá trị logic để đánh dấu các đỉnh đã được đưa vào tập S (gồm các đỉnh có nhãn chính thức):

Mỗi bước, nếu xác định được đỉnh k để đưa vào tập S thì ta gán Daxet[k] = 1; và khi đó L[k] là nhãn chính thức của k
(chỉ chiều dài của đường đi nhỏ nhất của đường từ s đến k) .

- Khởi động dữ liệu:

Khởi động Daxet[] là rỗng: Daxet[i] = 0, $\forall$i.

Khởi động cận trên chiều dài của đường đi ngắn nhất từ s đến đỉnh khác (đánh nhãn tạm thời) bằng ∞ .

L[i] = ∞; i $\neq$ s.

Khởi động trọng số nhỏ nhất đường đi từ s đến s bằng 0.

L[s] = 0; //d(s,s) = 0

- Giả sử tại mỗi bước:

(Với Dht (đỉnh hiện tại) là đỉnh vừa đưa được vào S, Daxet[Dht] = 1), các đỉnh i chưa được xét sẽ được đánh nhãn lại như sau:

Nếu (L[Dht] + m(Dht,i)) < L[i]) thì:

L[i] = L[Dht] + m(Dht,i);

Và trong trường hợp này, đường đi ngắn nhất từ s đến i sẽ đi qua đỉnh Dht (đó là đỉnh kề trước i)

- Để lưu trữ các đỉnh trên đường đi ngắn nhất từ s đến t, với t $\in$ S, ta dùng mảng một chiều Ddnn[ ], với tính chất Ddnn[i] là đỉnh trước đỉnh i.

Thuật toán được cài đặt bằng hàm sau:

Input a[n][n], s

Output - Xuất ra màn hình đường đi ngắn nhất từ s đến các đỉnh còn lại

- Chiều dài tương ứng

```
void dijkstra( int s)
{
        int Ddnn[max]; // Chứa đường đi ngắn nhất từ s đến đỉnh t tại mỗi bước
        int i,k,Dht,Min;
        int Daxet[max]; //Đánh dấu các đỉnh đã đưa vào S
        int L[max];
        for ( i = 1; i <= n; i++)
        {
                Daxet[i] = 0;
                L[i] = VC;
        }
        //Dua dinh s vao tap dinh S da xet
        Daxet[s] = 1;
        L[s] = 0;
        Dht = s;
        int h = 1; //đếm mỗi bước: cho đủ n-1 bước
```

```
while (h<= n-1) {
                Min = VC;
                for ( i = 1; i <= n; i++)
                        if(!Daxet[i])
                        {
                                if ( L[dht] + a[dht][i] < L[i] ) //Tính lại nhãn
                                {
                                        L[i] = L[dht] + a[dht][i] ;
                                        Ddnn[i] = dht;
                                }
                                if(L[i] < Min)          // Chọn đỉnh k
                                {
                                        Min = L[i];
                                        k = i;
                                }
                        }
                xuatdd(s,k,Ddnn);
                cout<<"\nTrong so: "<<L[k];
                dht = k;// Khởi động lại Dht
                Daxet[dht] = 1;         //Đưa nút k vào tập nút đã xét
                h++;
        }
}
```

THUẬT TOÁN FLOYD

TÌM ĐƯỜNG ĐI NGẮN NHẤT
GIỮA CÁC CẶP ĐỈNH

Bài toán:

Cho G = (V,E) là một đơn đồ thị có hướng có trọng số. V = {1,..,n} là tập các đỉnh. E là tập các cung. Tìm đường đi ngắn nhất giữa các cặp đỉnh của đồ thị.

❖ *Ý tưởng:*

Thuật toán Floyd được thiết kế theo phương pháp quy hoạch động. Nguyên lý tối ưu được vận dụng cho bài toán này là:

"Nếu k là đỉnh nằm trên đường đi ngắn nhất từ i đến j thì đoạn đường từ i đến k và từ k đến j cũng phải ngắn nhất".

❖ *Thiết kế:*

Đồ thị được biểu diễn bởi ma trận kề các trọng số của cung:

$a = (a_{ij})_{nxn}$

Ta ký hiệu :

$$\forall i, j \in \{1, \ldots, n\} : a_{ij} = \begin{cases} \text{Troïngsoá}(i, j); (i, j) \in E \\ 0; i = j \\ \infty; (i, j) \notin E \end{cases}$$

❖ Ma trận trọng số đường đi ngắn nhất giữa các cặp đỉnh: d = (d_{ij})

d_{ij}: Trọng số của đường đi ngắn nhất từ i đến j.

❖Ma trận xác định các đỉnh trung gian của đường đi ngắn nhất từ i đến j: p = (p_{ij}

- pij: đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian pij hay không?
- p_{ij} = 0; đường đi ngắn nhất từ i đến j không có đi qua đỉnh trung gian pij.
- $p_{ij} \neq$ 0; đường đi ngắn nhất từ i đến j đi qua đỉnh trung gian pij.

❖ Ở bước k:

- Ký hiệu ma trận d là d^k cho biết chiều dài nhỏ nhất của đường đi từ i đến j
- Ký hiệu ma trận p là p^k cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian thuộc tập đỉnh {1,..,k}.

Input a
Output d,p;

❖ Mô tả:

Bước 0:

- Khởi động d: $d = a$; $(= d^0)$
- Khởi động p: $p_{ij} = 0$;

Bước 1:

- Kiểm tra mỗi cặp đỉnh i, j: Có/không một đường đi từ i đến j đi qua đỉnh trung gian 1, mà có trọng số nhỏ hơn bước 0? Trọng số của đường đi đó là:

$$d^1_{ij} = \mathrm{Min}\{ d^0_{ij} , d^0_{i1} + d^0_{1j} \}$$

- Nếu $d^1_{ij} = d^0_{i1} + d^0_{1j}$ thì $p^1_{ij} = 1$, tức là đường đi tương ứng đi qua đỉnh 1.

Bước 2:

- Kiểm tra mỗi cặp đỉnh i, j: Có/không một đường đi từ i đến j đi qua đỉnh trung gian 2, mà có trọng số nhỏ hơn bước 1? Trọng số của đường đi đó là:

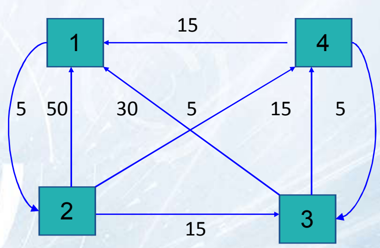
$$d^2_{ij} = \mathrm{Min}\{ d^1_{ij} , d^1_{i2} + d^1_{2j}\}$$

- Nếu $d^2_{ij} = d^1_{i2} + d^1_{2j}$ thì $p^2_{ij} = 2$: tức là đường đi tương ứng đi qua đỉnh 2.
- ...

Cứ tiếp tục như vậy, thuật toán kết thúc sau bước n, ma trận d xác định trọng số đường đi ngắn nhất giữa 2 đỉnh bất kỳ i, j. Ma trận p cho biết đường đi ngắn nhất từ i đến j có đi qua đỉnh trung gian p_{ij}.

❖ Minh hoạ:

Tìm đường đi ngắn nhất giữa các cặp đỉnh của đồ thị:



$$\begin{bmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{bmatrix}$$

❖ **Hoạt động của thuật toán như sau :**

b^1		1	2	3	4
d^1	1	0	5	∞	∞
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^1	1	0	0	0	0
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

$$\begin{bmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{bmatrix}$$

$d^1_{ij} = \text{Min}\{ d^0_{ij} , d^0_{i1} + d^0_{1j} \}$

Nếu $d^1_{ij} = d^0_{i1} + d^0_{1j}$ thì $p^1_{ij} = 1$, tức là đường đi tương ứng đi qua đỉnh 1.

b^2		1	2	3	4
d^2	1	0	5	20	10
	2	50	0	15	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^2	1	0	0	2	2
	2	0	0	0	0
	3	0	1	0	0
	4	0	1	0	0

$d^2_{ij} = \text{Min}\{ d^1_{ij} , d^1_{i2} + d^1_{2j}\}$

Nếu $d^2_{ij} = d^1_{i2} + d^1_{2j}$ thì $p^2_{ij} = 2$: tức là đường đi tương ứng đi qua đỉnh 2.

0	5	20	10
50	0	15	5
30	35	0	15
15	20	5	0

$d^3_{ij} = \text{Min}\{ d^2_{ij} , d^2_{i3} + d^2_{3j}\}$

Nếu $d^3_{ij} = d^2_{i3} + d^2_{3j}$ thì $p^3_{ij} = 3$: tức là đường đi tương ứng đi qua đỉnh 3.

b^3		1	2	3	4
d^3	1	0	5	20	10
	2	45	0	15	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^3	1	0	0	2	2
	2	3	0	0	0
	3	0	1	0	0
	4	0	1	0	0

b^4		1	2	3	4
d^4 = d	1	0	5	15	10
	2	20	0	10	5
	3	30	35	0	15
	4	15	20	5	0

		1	2	3	4
p^4 = p	1	0	0	4	2
	2	4	0	4	0
	3	0	1	0	0
	4	0	1	0	0

$d^4_{ij} = \text{Min}\{ d^3_{ij} , d^3_{i4} + d^3_{4j}\}$

Nếu $d^4_{ij} = d^3_{i4} + d^3_{4j}$ thì $p^4_{ij} = 4$: tức là đường đi tương ứng đi qua đỉnh 4.

- Căn cứ vào ma trận d, ta chỉ ra khoảng cách đường đi ngắn nhất từ i đến j, và dựa vào p có thể xác định các đỉnh nằm trên đường đi ngắn nhất này.

- Chẳng hạn, với i = 1, j = 3.
  - Theo d, d_{13} = 15. Nên đường đi ngắn nhất từ 1 đến 3 có khoảng cách là 15.
  - Theo p, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3 đi qua đỉnh trung gian p_{13} = 4, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 4 đi qua đỉnh trung gian p_{14} = 2, đường đi ngắn nhất từ đỉnh 1 đến đỉnh 2 không đi qua đỉnh trung gian nào (p_{12} = 0).
  - Vậy đường đi ngắn nhất từ đỉnh 1 đến đỉnh 3 là:

$$1 \rightarrow 2 \rightarrow 4 \rightarrow 3.$$

- ***Cài đặt:***

```
void floyd()
{
    int i, j, k;
    // Khoi dong ma tran d va p
    for (i = 1; i<= n; i++)
        for (j = 1; j<= n; j++)
        {
            d[i][j] = a[i][j];
            p[i][j] = 0;
        }
    for (k = 1; k <= n; k++) // Tính ma trận d và p ở bước lặp k
      for (i = 1; i <= n; i++)
        if (d[i][k] > 0 && d[i][k] < vc )
          for (j = 1; j<= n; j++)
            if (d[k][j] > 0 && d[k][j] < vc )
              if (d[i][k] + d[k][j] < d[i][j] )
                {
                    d[i][j] = d[i][k] + d[k][j];
                    p[i][j] = k;
                }
}
```

Hàm xuất đường đi ngắn nhất từ x đến y cài đặt như sau:

```
void xuatdd(int x, int y)
{
    int r;
    if (p[x][y] == 0)
    {
        cout<<y<<" -> ";
        return;
    }
    else
    {
        r = p[x][y];
        xuatdd(x,r);
        xuatdd(r,y);
    }
}
```