CHƯƠNG 5: ĐỒ THỊ

Nguyễn Văn Linh
Khoa CNTT-TT
Đại học Cần Thơ
nvlinh@ctu.edu.vn

NỘI DUNG

- Các khái niệm
- Biểu diễn đồ thị
- Các phép duyệt trên đồ thị
- Các bài toán trên đồ thị

CÁC KHÁI NIỆM (1)

- Một đồ thị G bao gồm một tập hợp V các đỉnh và một tập hợp E các cạnh, ký hiệu G=(V,E).
- Các đỉnh còn được gọi là nút (node) hay điểm (point). Các cạnh nối giữa hai đỉnh, hai đỉnh này có thể trùng nhau.
- Hai đỉnh có cạnh nối nhau gọi là hai đỉnh kề (adjacency). Một cạnh nối giữa hai đỉnh v, w có thể coi như là một cặp điểm (v,w).
- Nếu cặp này có thứ tự thì ta có cạnh có thứ tự, ngược lại thì cạnh không có thứ tự.
- Nếu các cạnh trong đồ thị G có thứ tự thì G gọi là đồ thị có hướng (directed graph).
- Nếu các cạnh trong đồ thị G không có thứ tự thì đồ thị G là đồ thị vô hướng (undirected graph).

CÁC KHÁI NIỆM (2)

- Thông thường trong một đồ thị, các đỉnh biểu diễn cho các đối tượng còn các cạnh biểu diễn mối quan hệ giữa các đối tượng đó.
- Một đường đi (path) trên đồ thị là một dãy tuần tự các đỉnh v1, v2,..., vn sao cho (vi,vi+1) là một cạnh trên đồ thị (i=1,...,n-1).
- Đỉnh v1 còn gọi là đỉnh đầu, vn gọi là đỉnh cuối. Độ dài của đường đi này bằng số đỉnh trừ 1.
- Trường hợp đặc biệt dãy chỉ có một đỉnh v thì ta coi đó là đường đi từ v đến chính nó có độ dài bằng không.

CÁC KHÁI NIỆM (3)

- Đường đi gọi là đơn nếu mọi đỉnh trên đường đi đều đôi một khác nhau, ngoại trừ đỉnh đầu và đỉnh cuối có thể trùng nhau.
- Một đường đi có đỉnh đầu và đỉnh cuối trùng nhau gọi là một chu trình (cycle).
- Đồ thị có trọng số: các cạnh hoặc các đỉnh có giá trị

CANTHO UNIVERSITY

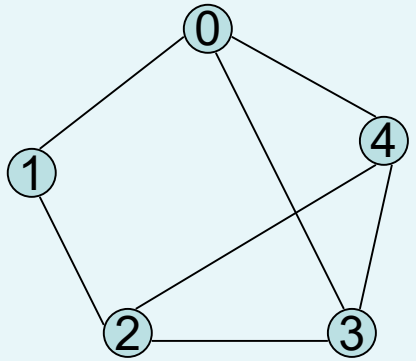
BIỂU DIỄN ĐỒ THỊ

- Biểu diễn đồ thị bằng ma trận kề
- Biểu diễn đồ thị bằng danh sách các đỉnh kề

BIỂU DIỄN BẰNG MA TRẬN KỀ (1)

- Dùng mảng A logic 2 chiều có nxn phần tử để biểu diễn cho đồ thị G có n đỉnh.
- Giả sử các đỉnh của đồ thị được đánh số từ 0 đến n-1.
- Nếu có cạnh (i,j) thì A[i,j] = 1, ngược lại thì A[i,j] = 0.

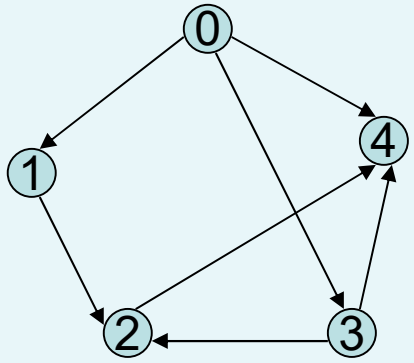
BIỂU DIỄN BẰNG MA TRẬN KỀ (2)



A	0	1	2	3	4
0		1	0	1	1
1	1		1	0	0
2	0	1		1	1
3	1	0	1		1
4	1	0	1	1	

Ma trận kề của đồ thị vô hướng là ma trận đối xứng

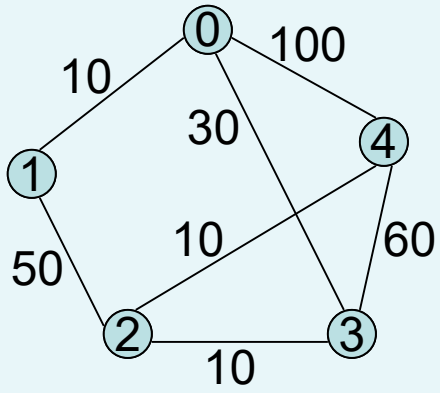
BIỂU DIỄN BẰNG MA TRẬN KỀ (3)



A	0	1	2	3	4
0		1	0	1	1
1	0		1	0	0
2	0	0		0	1
3	0	0	1		1
4	0	0	0	0	

Ma trận kề của đồ thị có hướng là ma trận không đối xứng

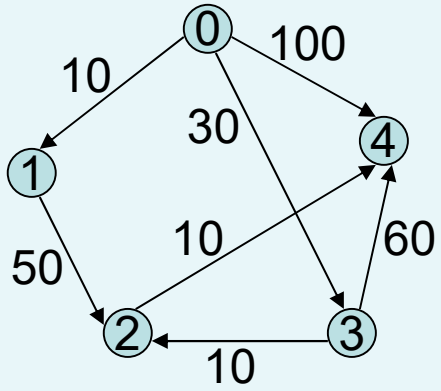
MA TRẬN TRỌNG SỐ (1)



A	0	1	2	3	4
0		10	∞	30	100
1	10		50	∞	∞
2	∞	50		10	10
3	30	∞	10		60
4	100	∞	10	60	

Ma trận trọng số của đồ thị vô hướng là ma trận đối xứng

MA TRẬN TRỌNG SỐ (2)



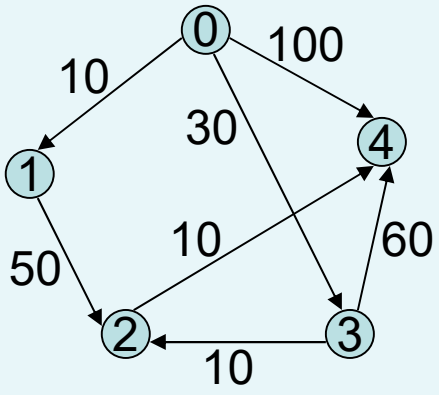
A	0	1	2	3	4
0		10	∞	30	100
1	∞		50	∞	∞
2	∞	∞		∞	10
3	∞	∞	10		60
4	∞	∞	∞	∞	

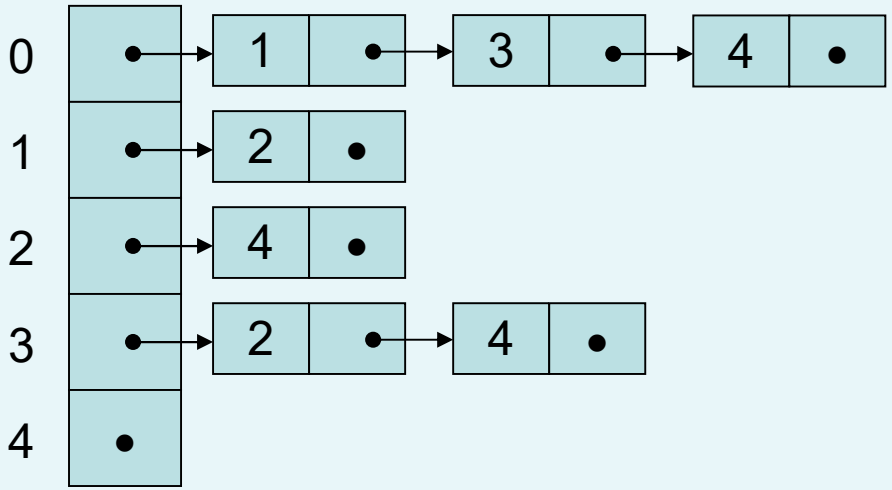
Ma trận trọng số của đồ thị có hướng là ma trận không đối xứng

BIỂU DIỄN BẰNG DANH SÁCH CÁC ĐỈNH KỀ (1)

- Lưu trữ các đỉnh kề với một đỉnh i trong một danh sách liên kết theo một thứ tự nào đó.
- Như vậy ta cần một mảng HEAD một chiều có n phần tử để biểu diễn cho đồ thị có n đỉnh.
- HEAD[i] là con trỏ trỏ tới danh sách các đỉnh kề với đỉnh i.

BIỂU DIỄN BẰNG DANH SÁCH CÁC ĐỈNH KỀ (2)





CÁC PHÉP DUYỆT TRÊN ĐỒ THỊ

- Duyệt theo chiều sâu (depth-first search)
- Duyệt theo chiều rộng (breadth-first search)

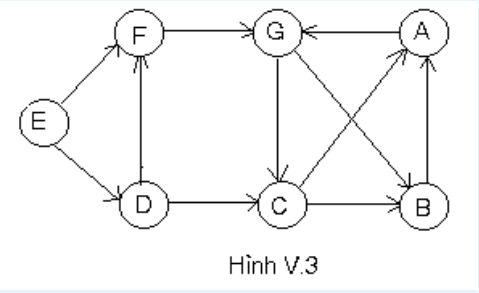
DUYỆT THEO CHIỀU SÂU (1)

- Giả sử ta có đồ thị G=(V,E) với các đỉnh ban đầu được đánh dấu là chưa duyệt (unvisited).
- Từ một đỉnh v nào đó ta bắt đầu duyệt như sau: đánh dấu v đã duyệt, với mỗi đỉnh w chưa duyệt kề với v, ta thực hiện đệ qui quá trình trên cho w.
- Dùng một mảng mark có n phần tử để đánh dấu các đỉnh của đồ thị là đã duyệt hay chưa.

DUYỆT THEO CHIỀU SÂU (2)

```
void dfs(Vertex v) {
    mark[v]=visited;
    for (mỗi đỉnh w là đỉnh kề với v)
        if (mark[w] == unvisited)
            dfs(w);
}
for (v=0; v<n ; v++) mark[v]=unvisited;
for (v=0; v<n ; v++) //duyệt theo chiều sâu từ đỉnh đánh số 0
    if (mark[v] == unvisited)
        dfs(v); //duyệt theo chiều sâu đỉnh v
```

DUYỆT THEO CHIỀU SÂU (3)



Hình V.3

- Khởi đầu tại A, đánh dấu A, duyệt G,
- Đánh dấu G, duyệt B,
- Đánh dấu B, #
- Duyệt C, đánh dấu C, #
- Duyệt D, đánh dấu D, duyệt F .
- Đánh dấu F, #
- Duyệt E, đánh dấu E, #
- Kết quả: A, G, B, C, D, F, E

DUYỆT THEO CHIỀU RỘNG (1)

- Giả sử ta có đồ thị G với các đỉnh ban đầu được đánh dấu là chưa duyệt (unvisited).
- Từ một đỉnh v nào đó ta bắt đầu duyệt như sau: đánh dấu v đã được duyệt, kế đến là duyệt tất cả các đỉnh kề với v.
- Khi ta duyệt một đỉnh v rồi đến đỉnh w thì các đỉnh kề của v được duyệt trước các đỉnh kề của w, vì vậy ta dùng một hàng để lưu trữ các nút theo thứ tự được duyệt để có thể duyệt các đỉnh kề với chúng.
- Ta cũng dùng mảng một chiều mark để đánh dấu một nút là đã duyệt hay chưa.

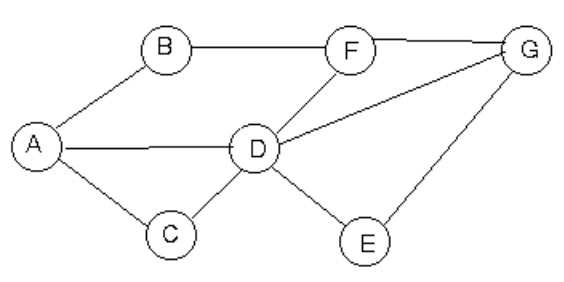
DUYỆT THEO CHIỀU RỘNG (2)

```
for (v=0; v<n; v++)
      mark[v]=unvisited;
//duyệt theo chiều rộng từ đỉnh đánh số 0
for (v=0;  v<n; v++)
   if (mark[v] == unvisited)
        bfs(v);
```

DUYỆT THEO CHIỀU RỘNG (3)

```
void bfs(Vertex v) {
   Queue_of_Vertex Q;
   Vertex x, y;
    mark[v]=visited;
    En_Queue(v,Q);
    while (!Empy_Queue(Q)) {
          x=Front(Q);
          De_Queue(Q);
          for (mỗi đỉnh y kề với x) do
                if (mark[y] == unvisited) {
                      mark[y]=visited; //duyệt y
                      En_Queue(y,Q); }
    }// while
}
```

DUYỆT THEO CHIỀU RỘNG (4)



Hình V.4

- Khởi đầu từ A, đánh dấu A, đưa A vào hàng.
- Lấy A ra, đánh dấu B, đưa B vào hàng
- Đánh dấu C, đưa C vào hàng
- Đánh dấu D, đưa D vào hàng
- Lấy B ra, đánh dấu F, đưa F vào hàng
- Lấy C ra
- Lấy D ra, đánh dấu E, đưa E vào hàng
- Đánh dấu G, đưa G vào hàng
- Lấy F, E và G ra khỏi hàng.
- Kết quả duyệt: A, B, C, D, F, E, G

CÁC BÀI TOÁN TRÊN ĐỒ THỊ

- Tìm đường đi ngắn nhất từ một đỉnh của đồ thị (the single source shorted path problem)
- Tìm cây bao trùm tối thiểu (minimum-cost spanning tree)
- Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh
- Tìm bao đóng chuyển tiếp (transitive closure)

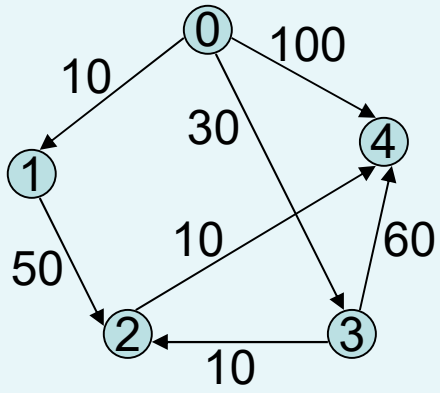
TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (1)

- Cho đồ thị với các cạnh có trọng số G =(V, E) (có hướng hoặc vô hướng).
- Trọng số của một cạnh có thể xem là khoảng cách giữa 2 đỉnh
- Cho trước một đỉnh v, gọi là đỉnh nguồn.
- Tìm đường đi ngắn nhất từ v đến các đỉnh còn lại của G.
- Chú ý rằng nếu đồ thị có hướng thì đường đi này là đường đi có hướng.

TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (2)

- Xác định một tập hợp S chứa các đỉnh mà khoảng cách ngắn nhất từ đỉnh nguồn v đến nó đã biết.
- Khởi đầu S={v}, sau đó tại mỗi bước ta sẽ thêm vào S các đỉnh mà khoảng cách từ v đến nó là ngắn nhất.
- Với giả thiết mỗi cạnh có một khoảng cách không âm thì ta luôn luôn tìm được một đường đi ngắn nhất như vậy mà chỉ đi qua các đỉnh đã tồn tại trong S.

TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (3)



- Giả sử đỉnh nguồn là 0
- Khởi đầu S = {0}
- S = {0, 1}
- S = {0, 1, 3}
- S = {0, 1, 3, 2}
- S = {0, 1, 3, 2, 4}

ĐĐNN từ 0 đến 2 là 0, 3, 2 với tổng độ dài 40

ĐĐNN từ 0 đến 4 là 0, 3, 2, 4 với tổng độ dài 50

TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (4)

- Ta dùng mảng hai chiều C để lưu độ dài các cạnh, tức là C[i,j] là độ dài của cạnh (i,j), nếu không có cạnh (i,j) thì C[i,j]=∞.
- Ta dùng mảng 1 chiều D có n phần tử để lưu độ dài của đường đi ngắn nhất từ v đến mỗi đỉnh của đồ thị.
- Khởi đầu khoảng cách này chính là độ dài cạnh (v,i), tức là D[i]:=C[v,i].
- Tại mỗi bước của giải thuật thì D[i] sẽ được cập nhật lại để lưu độ dài đường đi ngắn nhất từ đỉnh v tới đỉnh i, đường đi này chỉ đi qua các đỉnh đã có trong S.
- Để cài đặt giải thuật dễ dàng, ta giả sử các đỉnh của đồ thị được đánh số từ 0 đến n-1, tức là V={0,..,n-1} và 0 là đỉnh nguồn

TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (5)

```
void Dijkstra () {
    S=[0]; //S chỉ chứa một đỉnh nguồn
    for (i=1; i<n; i++)
        D[i]=C[0,i]; //khởi đầu các giá trị cho D
    for (i=1; i<n; i++) {
        Lấy đỉnh w trong V-S sao cho D[w] nhỏ nhất;
        Thêm w vào S;
        for (mỗi đỉnh u thuộc V-S) do
            D[u] = min(D[u], D[w] + C[w,u]);
    }
}
```

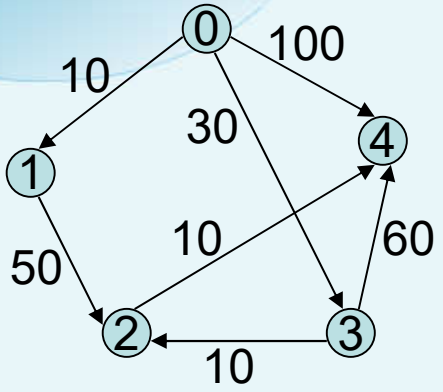
TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (6)

- Để lưu lại các đỉnh trên đường đi ngắn nhất, ta dùng một mảng P.
- Mảng này sẽ lưu P[u]=w với u là đỉnh "trước" đỉnh w trong đường đi.
- Lúc khởi đầu P[u]=0.

TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (7)

```
void Dijkstra () {
    S=[0]; //S chỉ chứa một đỉnh nguồn
    for (i=1; i<n; i++) {
        D[i]=C[0,i]; //khởi đầu các giá trị cho D
        P[i] = 0; }
    for (i=1; i<n; i++) {
        Lấy đỉnh w trong V-S sao cho D[w] nhỏ nhất;
        Thêm w vào S;
        for (mỗi đỉnh u thuộc V-S) do
            if  (D[w] + C[w,u] < D[u]) {
                D[u]=D[w] + C[w,u];
                P[u]=w;    }
    }
}
```

TÌM ĐƯỜNG ĐI NGẮN NHẤT TỪ MỘT ĐỈNH CỦA ĐỒ THỊ (8)




```
for (i=1; i<n; i++) {
   Lấy đỉnh w trong V-S sao cho D[w] nhỏ nhất, Thêm w vào S;
   for (mỗi đỉnh u thuộc V-S) do
      if  (D[w] + C[w,u] < D[u]) {
            D[u]=D[w] + C[w,u];
            P[u]=w; }
}
```

Lần lặp	S	W	D[1]	D[2]	D[3]	D[4]
Khởi đầu	{0}	-	**10**	∞	30	100
i=1	{0,1}	1	10	60	**30**	100
i=2	{0,1,3}	3	10	**40**	30	90
i=3	{0,1,3,2}	2	10	40	30	**50**
i=4	{0,1,3,2,4}	4	10	40	30	50

P	0	1	2	3	4
		0	0	0	0
i=1		0	1	0	0
i=2		0	3	0	3
i=3		0	3	0	2
i=4		0	3	0	2

Từ mảng P, suy ngược lại ta sẽ có đường đi ngắn nhất. Ví dụ trước 4 là 2, trước 2 là 3 và trước 3 là 0. Vậy đường đi ngắn nhất từ 0 đến 4 là 0,3,2,4 với độ dài lưu trong D[4] = 50

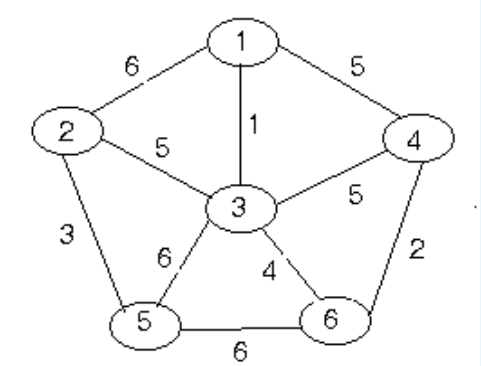
TÌM CÂY BAO TRÙM TỐI THIỂU (1)

- Giả sử ta có một đồ thị vô hướng G=(V,E).
- Đồ thị G gọi là liên thông nếu tồn tại đường đi giữa hai đỉnh bất kỳ.
- Cây bao trùm tối thiểu (hoặc cây phủ tối thiểu) của đồ thị G là một cây T có tập các nút là V và tổng độ dài các cạnh trong T là nhỏ nhất.
- Một ứng dụng thực tế là bài toán thiết lập mạng truyền thông, ở đó các đỉnh là các thành phố còn các cạnh của cây bao trùm là đường nối mạng giữa các thành phố.

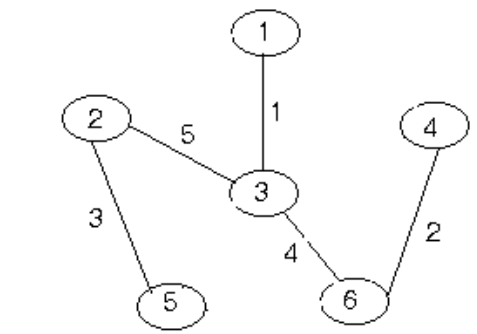
TÌM CÂY BAO TRÙM TỐI THIỂU (2)

- Giả sử G có n đỉnh được đánh số 1..n.
- Giải thuật Prim để giải bài toán này như sau:
- Bắt đầu, cho U = {1} và T=∅
- Sau đó ta lặp lại cho đến khi U=V, tại mỗi bước lặp ta chọn cạnh nhỏ nhất (u,v) sao cho u ∈ U và v ∈ V-U. Nếu thêm (u,v) vào T mà không tạo thành chu trình thì thêm v vào U và (u,v) vào T.
- Khi giải thuật kết thúc thì (U,T) là một cây phủ tối tiểu.

TÌM CÂY BAO TRÙM TỐI THIỂU (3)



Hình V.6



Hình V.7

- Khởi đầu cho U={1} và T=∅
- (1,3) nhỏ nhất nên U={1,3} và T={(1,3)}
- (3,6) nhỏ nhất nên U={1,3,6} và T={(1,3); (3,6)}
- (6,4) nhỏ nhất nên U={1,3,4,6} và T={(1,3); (3,6); (6,4)}
- (3,2) nhỏ nhất nên U={1,2,3,4,6} và T={(1,3); (3,6); (6,4), (3,2)}
- (2,5) nhỏ nhất nên U={1,2,3,4,5,6} và T={(1,3); (3,6); (6,4); (3,2); (2,5)}

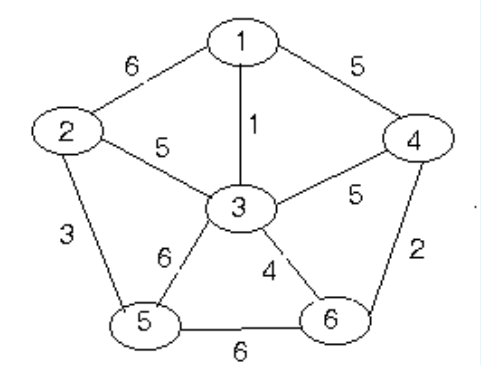
TÌM CÂY BAO TRÙM TỐI THIỂU (4)

```
void Prim(Graph G, Set_of_Edges &T) {
    Set_of_Vertices U = [1] ; //tập hợp các đỉnh, khởi đầu chỉ
    chứa 1
    Vertex u,v; //u,v là các đỉnh
    T= ∅;
    while (U!=V){  //V là tập hợp các đỉnh của G
        Nếu (u,v) là cạnh ngắn nhất sao cho u ∈ U, v ∈ V-U và
        (u,v) không tạo thành chu trình trên T {
            U=U ∪ [v];
            T=T ∪ [(u,v)];
        }
    }
}
```

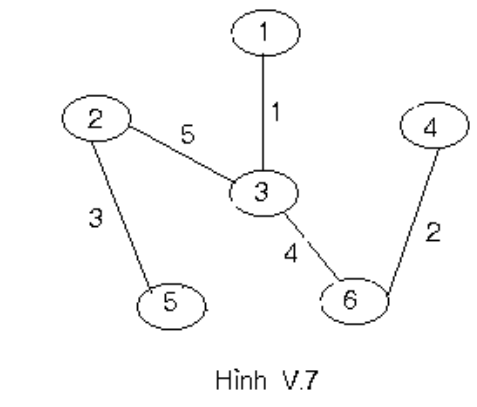
TÌM CÂY BAO TRÙM TỐI THIỂU (5)

- **Giải thuật Kruskal:**
- Khởi đầu cho T= ∅, ta thiết lập đồ thị khởi đầu G'=(V,T).
- Xét các cạnh của G theo thứ tự độ dài tăng dần.
- Với mỗi cạnh được xét ta sẽ đưa nó vào T nếu nó không làm cho G' có chu trình.
- (1,3)=1, (4,6)=2, (2,5)=3, (3,6)=4, (1,4)=(2,3)=(3,4)=5, (1,2)=(3,5)= (5,6)=6.

TÌM CÂY BAO TRÙM TỐI THIỂU (6)



Hình V.6



Hình V.7

- Khởi đầu T= ∅
- Lần lặp 1: T={(1,3)}
- Lần lặp 2: T={(1,3),(4,6)}
- Lần lặp 3: T={(1,3),(4,6),(2,5)}
- Lần lặp 4: T={(1,3),(4,6),(2,5),(3,6)}
- Lần lặp 5: Cạnh (1,4) không được đưa vào T vì nó sẽ tạo ra chu trình 1,3,6,4,1.
- Kế tiếp cạnh (2,3) được xét và được đưa vào T.
- T={(1,3),(4,6),(2,5),(3,6),(2,3)}

CANTHO UNIVERSITY

TÌM ĐƯỜNG ĐI NGẮN NHẤT GIỮA TẤT CẢ CÁC CẶP ĐỈNH (1)

- Cho đồ thị G có n đỉnh được đánh số từ 0 đến n-1.
- Khoảng cách giữa các cặp đỉnh được cho trong mảng C[i,j].
- Nếu hai đỉnh i,j không được nối thì C[i,j]= ∞.
- Giải thuật Floyd xác định đường đi ngắn nhất giữa hai cặp đỉnh bất kỳ bằng cách lặp n lần, ở lần lặp thứ k sẽ xác định khoảng cách ngắn nhất giữa hai đỉnh i,j theo công thức: Ak[i,j]=min(Ak-1[i,j], Ak-1[i,k]+Ak-1[k,j]).
- Ta cũng dùng mảng P để lưu các đỉnh trên đường đi.

TÌM ĐƯỜNG ĐI NGẮN NHẤT GIỮA TẤT CẢ CÁC CẶP ĐỈNH (2)

```
void Floyd() {
  for (i=0; i<n; i++)
    for (j=0 ; j<n; j++){
          A[i,j]=C[i,j];
          P[i,j]=0;
     }
  for (i=0; i<n; i++) A[i,i]=0;
  for (k=0; k<n; k++)
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
          if (A[i,k] + A[k,j] < A[i,j]){
              A[i,j]=A[i,k] + A[k,j];
              P[i,j]=k;
          }
}
```

TÌM BAO ĐÓNG CHUẨN TIẾP (1)

- Cần xác định có hay không có đường đi nối giữa hai đỉnh i,j bất kỳ.
- Giải thuật Floyd có thể đặc biệt hoá để giải bài toán này. Ma trận kề C có C[i,j]=1nếu i,j được nối nhau bởi một cạnh, ngược lại C[i,j]=0.
- Lúc này mảng A[i,j] không cho khoảng cách ngắn nhất giữa i,j mà nó cho biết là có đường đi từ i đến j hay không.
- A gọi là bao đóng chuyển tiếp của đồ thị G có biểu diễn ma trận kề là C.
- Giải thuật Floyd sửa đổi như trên gọi là giải thuật Warshall.

TÌM BAO ĐÓNG CHUẨN TIẾP (2)

```
void Warshall() {
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            A[i,j]=C[i,j];
    for (k=0; k<n; k++)
        for (i=0; i<n; i++)
            for (j=0; j<n; j++)
                if (!A[i,j])
                    A[i,j]=(A[i,k] &&|A[k,j]);
}
```


ĐẠI HỌC CẦN THƠ