CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Các thuật toán trên đồ thị

ThS. Nguyễn Văn Tiến

Nội dung

1. Giới thiệu về đồ thị
2. Biểu diễn đồ thị
3. Các thuật toán tìm kiếm trên đồ thị
4. Đồ thị Euler
5. Xây dựng cây khung đồ thị
6. Bài toán tìm đường đi ngắn nhất

Giới thiệu về đồ thị

Định nghĩa

ĐỒ THỊ G =<V, E>

V là tập hợp hữu hạn được gọi là tập đỉnh (V = {1, 2,.., n})

E là tập các cặp đỉnh trong V được gọi là tập cạnh.

Đồ thị vô hướng: E không tính đến thứ tự các đỉnh.

Đơn đồ thị vô hướng: Giữa hai đỉnh bất kỳ thuộc V có nhiều nhất một cạnh.

Đa đồ thị vô hướng: Tồn tại một cặp đỉnh trong V có nhiều hơn một cạnh nối.

Giả đồ thị vô hướng: Có khuyên, tức là cạnh e = (u, u)

Giới thiệu về đồ thị

Định nghĩa

Đơn đồ thị có hướng:

- E là tập các cặp có thứ tự hai đỉnh thuộc V.
- Có thể gọi là cạnh có hướng, hoặc cung

Đa đồ thị có hướng: Có cạnh lặp (cung lặp) trên một cặp đỉnh.

Một số thuật ngữ trên đồ thị vô hướng

Đỉnh kề: Hai đỉnh u và v của đồ thị vô hướng G =<V, E> được gọi là kề nhau nếu (u,v) là cạnh thuộc đồ thị G.

Bậc của đỉnh: Ta gọi bậc của đỉnh v trong đồ thị vô hướng là số cạnh xuất phát từ nó và ký hiệu là deg(v).

Đường đi từ u đến v: dãy x0, x1, . . ., xn-1, xn, trong đó n là số nguyên dương, x0=u, xn=v, (xi, xi+1) ∈ E.

Chu trình: Đường đi có đỉnh đầu trùng với đỉnh cuối.

Một số thuật ngữ trên đồ thị vô hướng

Tính liên thông: Đồ thị vô hướng được gọi là liên thông nếu luôn tìm được đường đi giữa hai đỉnh bất kỳ của nó.

Thành phần liên thông: Đồ thị vô hướng liên thông thì số thành phần liên thông là 1. Đồ thị vô hướng không liên thông thì số liên thông của đồ thị là số các đồ thị con của nó liên thông.

Đỉnh trụ: Đỉnh $u \in V$ được gọi là đỉnh trụ nếu loại bỏ u cùng với các cạnh nối với u làm tăng thành phần liên thông của đồ thị.

Cạnh cầu: Cạnh $(u,v) \in E$ được gọi là cầu nếu loại bỏ (u,v) làm tăng thành phần liên thông của đồ thị.

Một số thuật ngữ trên đồ thị có hướng

Liên thông mạnh:

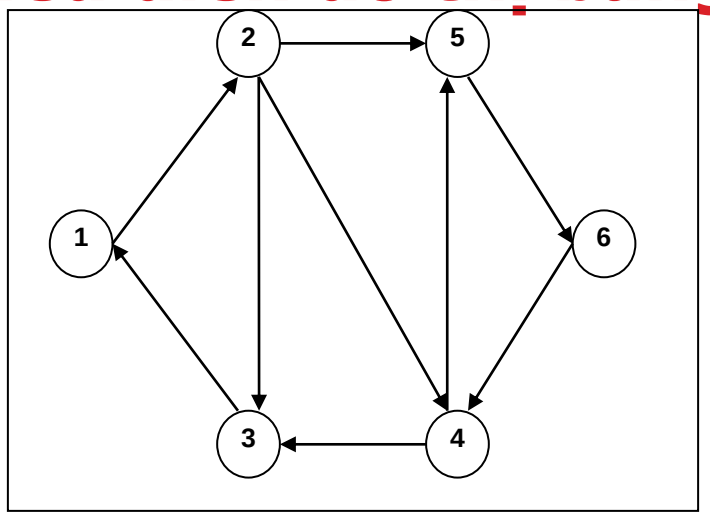
Luôn tìm được đường đi giữa hai đỉnh bất kỳ.

Liên thông yếu:

Đồ thị vô hướng nền là liên thông.

Biểu diễn đồ thị

Biểu diễn đồ thị bằng ma trận kề



$$\begin{matrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{matrix}$$

Ưu điểm của ma trận kề:

Đơn giản

Dễ dàng kiểm tra hai đỉnh kề nhau

Qua một phép so sánh.

Nhược điểm của ma trận kề:

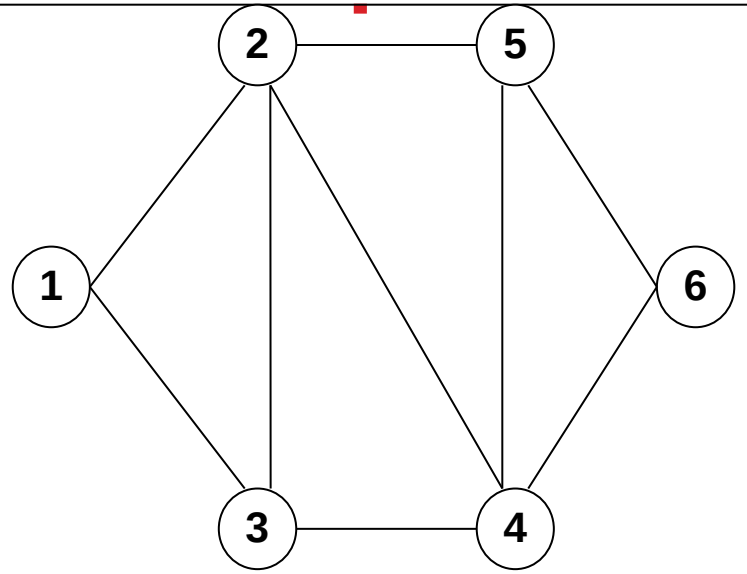
Tốn bộ nhớ

Không thể biểu diễn được với các đồ thị có số đỉnh lớn

Có thể có những phép so sánh không cần thiết khi đồ thị thưa.

Biểu diễn đồ thị

Biểu diễn bằng danh sách cạnh



Đỉnh đầu	Đỉnh cuối
1	2
1	3
2	3
2	4
2	5
3	4
4	5
4	6
5	6

Ưu điểm của danh sách cạnh:

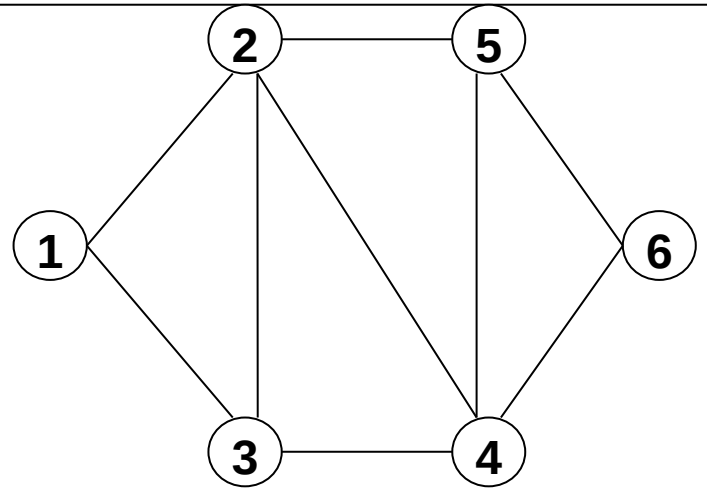
Trong trường hợp đồ thị thưa sẽ tiết kiệm được không gian nhớ;

Thuận lợi cho một số thuật toán chỉ quan tâm đến các cạnh của đồ thị.

Nhược điểm của danh sách cạnh:

Khi cần duyệt các đỉnh kề với đỉnh u phải duyệt tất cả các cạnh của đồ thị.

Biểu diễn bằng danh sách kề



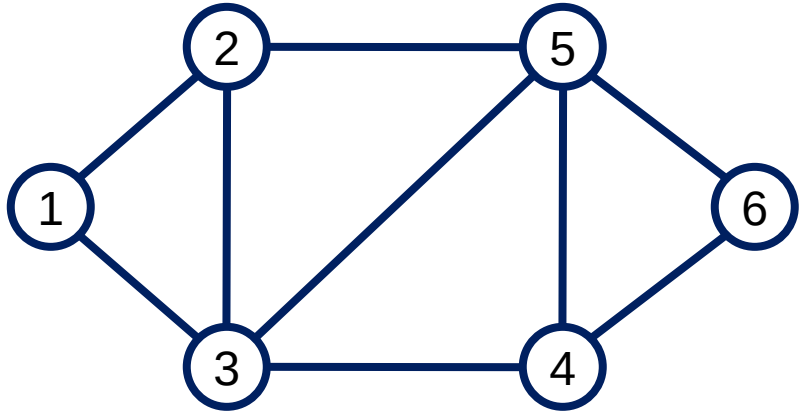
List(1) = {2, 3 }
List(2) = {1, 3, 4, 5 }
List(3) = {1, 2, 4 }
List(4) = {2, 3 , 5, 6}
List(5) = {2, 4, 6 }
List(6) = {4, 5 }

Ưu điểm của danh sách kề:
Dễ dàng duyệt tất cả các đỉnh của một danh sách kề.
Thời gian duyệt đồ thị nhanh hơn.
Nhược điểm của danh sách kề:
Khó cài đặt với các bài toán xử lý cạnh.

Biểu diễn đồ thị

MA TRẬN KỀ

```
6
0  1  1  0  0  0
1  0  1  0  1  0
1  1  0  0  1  0
0  1  1  0  1  1
0  1  1  1  0  1
0  0  0  1  1  0
```



DANH SÁCH KỀ

```
6
2  3
1  3  5
1  2  4  5
3  5  6
2  3  4  6
4  5
```

DANH SÁCH CẠNH

```
6  8
1  2
1  3
2  3
2  5
3  4
3  5
4  5
4  6
5  6
```

Biểu diễn đồ thị

Chuyển danh sách cạnh sang ma trận kề

```
cin>>n>>m;
for(i=0; i<m; i++){
    cin>>u>>v;
    a[u][v] = 1;
    a[v][u] = 1;
}
```

Chuyển danh sách cạnh sang danh sách kề

```
cin >> n >> m;
for(i=1; i<=m; i++){
    cin >> u >> v;
    List[u].push_back(v);
    List[v].push_back(u);
}
```

Chuyển danh sách kề sang ma trận kề

```
for(int i = 1; i <= n; i++){
    string s, token;
    getline(cin, s);
    stringstream ss(s);
    while(ss >> token){
        int k = 0;
        for(int j = 0; j < token.size(); j++) {
            k = k * 10 + token[j] - 48;
        }
        a[i][k] = 1;
    }
}
```

Tìm kiếm theo chiều sâu (DFS)

Bài toán: Cho đồ thị vô hướng hoặc có hướng G =<V,E>.

Hãy duyệt (thăm) các đỉnh của đồ thị bắt đầu tại một đỉnh u tùy ý thuộc V.

Tư tưởng tìm kiếm theo chiều sâu:

Đánh dấu trạng thái các đỉnh

chuaxet[u] = true ứng với trạng thái đỉnh u chưa được duyệt

chuaxet[u] = false ứng với trạng thái đỉnh u đã được duyệt.

Bắt đầu tại đỉnh tùy ý u ∈ V ta thăm luôn đỉnh u.

Bật trạng thái chuaxet[u]=false.

Duyệt trên tập đỉnh Ke(u) = { v ∈ V : (u,v) ∈ E } và duyệt theo chiều sâu tại đỉnh v ∈ Ke(u) đầu tiên chưa được duyệt

Tìm kiếm theo chiều sâu (DFS)

Thuật toán đệ quy DFS(u)

```
void    DFS ( u) {
        <thăm đỉnh u>;
        chuaxet[u] = False;
        for (v=1; v<=n; v++)
                if ( v ∈ Ke(u) and chuaxet[v])
                        DFS(v);
}
```

Thuật toán DFS(u):

Begin

Bước 1 (Khởi tạo):

stack = ∅; //*Khởi tạo stack là* ∅

Push(stack, u); //*Đưa đỉnh u vào ngăn xếp*

<Thăm đỉnh u>; //*Duyệt đỉnh u*

chuaxet[u] = False; //*Xác nhận đỉnh u đã duyệt*

Bước 2 (Lặp) :

while (stack ≠ ∅) do

s = Pop(stack); //*Loại đỉnh ở đầu ngăn xếp*

for each t∈ Ke(s) do //*Lấy mỗi đỉnh t∈Ke(s)*

if (chuaxet[t]) then //*Nếu t đúng là chưa duyệt*

<Thăm đỉnh t>; // *Duyệt đỉnh t*

chuaxet[t] = False; // *Xác nhận đỉnh t đã duyệt*

Push(stack, s);//*Đưa s vào stack*

Push(stack, t); //*Đưa t vào stack*

break; //*Chỉ lấy một đỉnh t*

EndIf;

EndFor;

EndWhile;

Bước 3 (Trả lại kết quả): Return(<Tập đỉnh đã duyệt>);

End.

Thuật toán BFS(u):

Bước 1(Khởi tạo):

Queue = ∅; // *Khởi tạo hàng đợi rỗng*

Push(Queue,u); //*Đưa u vào hàng đợi*

chuaxet[u] = False;//*Ghi nhận đỉnh đã xét*

Bước 2 (Lặp):

while (Queue ≠ ∅) do //*Lặp đến khi hàng đợi rỗng*

s = Pop(Queue); //*Lấy s ra khỏi hàng đợi*

<Thăm đỉnh s>;//*Thăm đỉnh s*

for each t∈Ke(s) do //*Lặp trên danh sách kề của s*

if (chuaxet[t]) *then //Nếu t chưa xét*

Push(Queue, t); //*Đưa t vào hàng đợi*

chuaxet[t] = False;//*Ghi nhận t đã xét*

EndIf ;

EndFor ;

EndWhile ;

Bước 3 (Trả lại kết quả) :

Return(<Tập đỉnh được duyệt>) ;

End.

Độ phức tạp tính toán

Độ phức tạp thuật toán DFS(u), BFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị.

O(n2) trong trường hợp đồ thị biểu diễn dưới dạng ma trận kề, với n là số đỉnh

O(n.m) trong trường hợp đồ thị biểu diễn dưới dạng danh sách cạnh, với n là số đỉnh của đồ thị, m là số cạnh.

O(max(n, m)) trong trường hợp đồ thị biểu diễn dưới dạng danh sách kề, với n là số đỉnh của đồ thị, m là số cạnh của đồ thị.

Ứng dụng của DFS và BFS

Duyệt tất cả các đỉnh của đồ thị.
Tìm đường đi từ s đến t.
Xác định các thành phần liên thông của đồ thị.
Duyệt các đỉnh trụ của đồ thị.
Duyệt các cạnh cầu của đồ thị.
Xây dựng cây khung của đồ thị.
Xác định tính liên thông mạnh.

Duyệt tất cả các đỉnh của đồ thị

Bước 1 (Khởi tạo):

```
for (u=1; u ≤ n; u++) do { //thiết lập tất cả các đỉnh đều chưa duyệt
        chuaxet[u] = True;
Endfor;
```

Bước 2 (Lặp):

```
for ( u =1; u ≤ n; u++)  do { //lặp trên tập đỉnh V
        if (chuaxet[u] ) then
                BFS (u); //DFS(u);
        endif;
endfor;
```

Duyệt các thành phần liên thông

Bước 1 (Khởi tạo):

```
Solt =0; //thiết lập số thành phần liên thông ban đầu là 0
for (u=1; u ≤ n; u++) //thiết lập tất cả các đỉnh đều chưa duyệt
        chuaxet[u] = True;
```

Bước 2 (Lặp):

```
for ( u =1; u ≤ n; u++)  do { //lặp trên tập đỉnh V
        if (chuaxet[u] ) then
                Solt++; //tăng thành phần liên thông
                <ghi nhận các đỉnh cùng một thành phần liên thông>;
                BFS (u); //DFS(u);
        endif;
```

Bước 3 (trả lại kết quả):

```
Return(Solt);
```

Tìm đường đi từ đỉnh s đến đỉnh t

Trace-DFS(s, t):

Bước 1 (Khởi tạo):

stack = ∅; //Khởi tạo stack là ∅

Push(stack, s); //Đưa đỉnh s vào ngăn xếp

chuaxet[s] = False; //Xác nhận đỉnh u đã duyệt

Tìm đường đi từ đỉnh s đến đỉnh t

Bước 2 (Lặp) :

```
while ( stack ≠∅) do
    u = Pop(stack); //Loại đỉnh ở đầu ngăn xếp
        for each v ∈ Ke(u) do //Lấy mỗi đỉnh u ∈ Ke(v)
        if ( chuaxet[v] ) then //Nếu v đúng là chưa duyệt
            chuaxet[v] = False; // Xác nhận đỉnh v đã duyệt
            Push(stack, u);//Đưa u vào stack
            Push(stack, v); //Đưa v vào stack
            truoc[v] = u; //Ghi nhận truoc[v] là u
            break;          //Chỉ lấy một đỉnh t
```

Bước 3 (Trả lại kết quả):

```
Return(truoc[]);//trả lại mảng ghi lại các đỉnh DFS  đã duyệt
```

Tìm đường đi từ đỉnh s đến đỉnh t

Trace-BFS(s, t):

Bước 1(Khởi tạo):

Queue = ∅; //Khỏi tạo hàng đợi là rỗng

Push(Queue,s); //Đưa s vào hàng đợi

chuaxet[s] = False; //Bật trạng thái đỉnh s

Tìm đường đi từ đỉnh s đến đỉnh t

Bước 2 (Lặp):

```
while (Queue ≠∅) do { //lặp đến khi hàng đợi rỗng
    u = Pop(Queue); //lấy u ra khỏi hàng đợi
    for each v ∈ Ke(u) do { //duyệt trên các đỉnh v kề với u
        if ( chuaxet[v] ) { //nếu đỉnh v đúng là chưa được xét
            Push(Queue, v); //đưa v vào hàng đợi
            chuaxet[v]=False;//ghi nhận v đã xét
            truoc[v]=u; //Ghi nhận muốn đi đến v phải qua u
```

Bước 3 (Trả lại kết quả) :

```
Return(truoc[]) ; //trả lại mảng ghi lại các đỉnh BFS  đã duyệt
End.
```

Ghi nhận đường đi từ đỉnh s đến đỉnh t

```
Ghi-Nhan-Duong-Di (s, t) {
        if ( truoc[t] == 0 ) {
                <Không có đường đi từ s đến t>;
        }
        else {
                <Đưa ra đỉnh t>; //Đưa ra trước đỉnh t
                u = truoc[t]; //u là đỉnh trước khi đến được t
                while (u ≠ s ) { //Lặp nếu u chưa phải là s
                        <Đưa ra đỉnh u>; //Đưa ra đỉnh u
                        u = truoc[u]; // Lần ngược lại đỉnh truoc[u].
                }
                <Đưa ra nốt đỉnh s>;
        }
}
```

Đồ thị Euler và nửa Euler

Chu trình đơn trong đồ thị G đi qua mỗi cạnh của đồ thị đúng một lần được gọi là chu trình Euler.

Đường đi đơn trong G đi qua mỗi cạnh của nó đúng một lần được gọi là đường đi Euler.

Đồ thị được gọi là đồ thị Euler nếu nó có chu trình Euler.

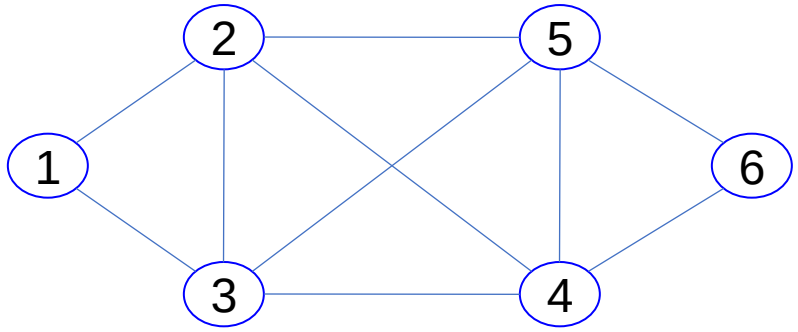
Đồ thị có đường đi Euler được gọi là nửa Euler.

Đồ thị Euler và nửa Euler

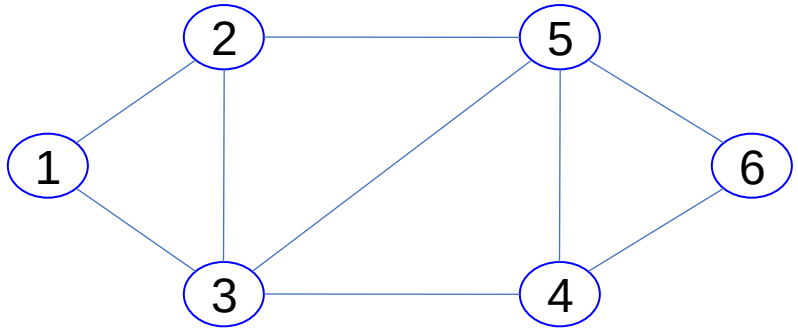
Ví dụ :

Đồ thị G1 là Euler vì G1 có chu trình Euler: 1-2-3-4-5-2-4-6-5-3-1

Đồ thị G2 là nửa Euler vì G1 có đường đi Euler: 2-1-3-2-5-3-4-6-5-4.



Đồ thị G1



Đồ thị G2

Định lý đồ thi Euler

Định lý 1 (Đồ thị vô hướng):

Đồ thị vô hướng liên thông G =<V, E> là Euler khi và chỉ khi tất cả các đỉnh của V đều có bậc chẵn.

Đồ thị vô hướng liên thông G =<V, E> là nửa Euler nhưng không là Euler khi và chỉ khi G có đúng hai đỉnh bậc lẻ. Đường đi Euler xuất phát tại một đỉnh bậc lẻ và kết thúc tại đỉnh bậc lẻ còn lại.

Định lý đồ thi Euler

Định lý 2 (Đồ thị có hướng):

Đồ thị có hướng liên thông yếu G =<V, E> là Euler khi và chỉ khi mọi đỉnh của G đều có bán đỉnh bậc ra bằng bán đỉnh bậc vào.

Đồ thị có hướng liên thông yếu G =<V, E> là nửa Euler nhưng không là Euler khi và chỉ khi tồn tại đúng hai đỉnh u, v sao cho bán đỉnh bậc ra của u trừ bán đỉnh bậc vào của u bằng bán đỉnh bậc vào của v trừ bán đỉnh bậc ra của v và bằng 1.

Kiểm tra đồ thị Euler vô hướng

```
boolean Check-Euler( A[][], n ) {//A là ma trận kề, n là số đỉnh của đồ thị
int  H; //H là tổng theo hàng;
int  dem=0;
for (int u=1; u<=n; u++){ //duyệt u ∈ V.
        H = 0; //thiết lập H=0;
        for (int v=1; v<=n; v++) { // duyệt v ∈ V.
                H = H + A[u][v]; //tổng các phần tử của hàng u.
        }
        if ( H % 2 == 1)
                dem ++; // tăng số đỉnh bậc lẻ lên 1
}
if (dem == 0 ) return(1); //nếu tất cả các đỉnh đều có bậc chẵn.
else if (dem ==2) return (-1); //nếu chỉ có hai đỉnh bậc lẻ.
return (0);
}
```

Kiểm tra đồ thị Euler có hướng (1)

```
boolean Check-Euler( A[][], n ) {//A là ma trận kề, n là số đỉnh của đồ thị
int  H, C; //H là tổng theo hàng; C là tổng theo cột
int  dem1=0, dem2=0, dem3 = 0;
for (int u=1; u<=n; u++){ //duyệt u ∈ V.
        H = 0; C =0; //thiết lập H=0; C=0
        for (int v=1; v<=n; v++) { // duyệt v ∈ V.
                H = H + A[u][v]; //tổng các phần tử của hàng u.
                C = C + A[v][u]; //tổng các phần tử của hàng u.
        }
        if ( H ==C) dem1 ++; //Nếu bán đỉnh bậc ra bằng bán đỉnh bậc vào.
        else if (H-C ==1) { //nếu bán đỉnh bậc ra trừ bán đỉnh bậc vào là 1
                dem2 ++; // tăng số đỉnh bậc lẻ lên 1.
                x = u; //ghi nhận đỉnh u.
        }
```

Kiểm tra đồ thị Euler có hướng (2)

```
        else if ((C-H ==1) { //nếu bán đỉnh bậc vào trừ bán đỉnh bậc ra là 1
                dem3 ++; // tăng số đỉnh bậc lẻ lên 1.
                y = u; //ghi nhận đỉnh v.
        }
}
if (dem1==n ) return(1); //nếu tất cả các đỉnh đều có deg+(u)=deg-(u).
else if (dem2 ==1 && dem3==1) return (-1); //nếu chỉ tồn tại u và v.
return (0);
}
```

Đồ thị Hamilton

Đường đi qua tất cả các đỉnh của đồ thị mỗi đỉnh đúng một lần được gọi là đường đi Hamilton.

Chu trình bắt đầu tại một đỉnh v nào đó qua tất cả các đỉnh còn lại mỗi đỉnh đúng một lần sau đó quay trở lại v được gọi là chu trình Hamilton.

Đồ thị có chu trình Hamilton được gọi là đồ thị Hamilton.

Đồ thị có đường đi Hamilton được gọi là đồ thị nửa Hamilton.

Đồ thị Hamilton

```
Thuật toán Hamilton( int k) {
/* Liệt kê các chu trình Hamilton của đồ thị bằng cách phát triển dãy đỉnh
(X[1], X[2], . . ., X[k-1] ) của đồ thị G = (V, E) */
        for y ∈ Ke(X[k-1]) {
                if (k==n+1) and (y == v0) then
                        Ghinhan(X[1], X[2], . . ., X[n], v0);
                else {
                        X[k]=y; chuaxet[y] = false;
                        Hamilton(k+1);
                        chuaxet[y] = true;
                }
        }
}
```

Cây khung của đồ thị

Cây là đồ thị vô hướng liên thông không có chu trình.
Đồ thị không liên thông được gọi là rừng.
Định lý: Giả sử T = <V, E> là đồ thị vô hướng n đỉnh. Khi đó những khẳng định sau là tương đương
T là một cây;
T không có chu trình và có n-1 cạnh;
T liên thông và có đúng n-1 cạnh;
T liên thông và mỗi cạnh của nó đều là cầu;
Giữa hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn;
T không chứa chu trình nhưng nếu thêm vào nó một cạnh ta thu được đúng một chu trình;

Thuật toán Tree-DFS(u) {

Bước 1 (khởi tạo):

Stack = ∅; sc =0; T = ∅; //Khởi tạo stack, số cạnh, và tập cạnh cây

Push(Stack, u); chuaxet[u] = False;

Bước 2 (Lặp):

While (Stack ≠ ∅)//lặp đến khi stack rỗng

s = Pop(Stack); //Lấy s ra khỏi ngăn xếp

For t ∈ Ke(s)//Duyệt các đỉnh t kề với s

If (chuaxet[t]) { //nếu t đúng là chưa xét

T= T ∪(s,t); sc++; chuaxet[t] = False;

Push(Stack, s); Push(Stack, t); break;

}

Bước 3 (Trả lại kết quả):

if (sc<n-1) <Đồ thị không liên thông>;

else return (T);

}

Thuật toán Tree-BFS(u) {

Bước 1 (khởi tạo):

Queue = ∅; sc =0; T = ∅; //Khởi tạo queue, số cạnh, và tập cạnh cây

Push(Queue, u); chuaxet[u] = False;

Bước 2 (Lặp):

While (Queue ≠ ∅)//lặp đến khi hàng đợi rỗng

s = Pop(Queue); //Lấy s ra khỏi hàng đợi

For t ∈ Ke(s) do //Duyệt các đỉnh t kề với s

If (chuaxet[t]) { //nếu t đúng là chưa xét

T= T ∪ (s,t); sc++; //Thêm cạnh (s,t) vào cây

Push(Queue, t); chuaxet[t] = False;

}

Bước 3 (Trả lại kết quả):

if (sc<n-1) <Đồ thị không liên thông>;

else return (T);

}

Xây dựng cây khung của đồ thị

Bài toán tìm cây khung nhỏ nhất

Cho G=<V, E> là đồ thị vô hướng liên thông với tập đỉnh V = {1, 2, . . ., n } và tập cạnh E gồm m cạnh. Mỗi cạnh e của đồ thị được gán với một số không âm c(e) được gọi là độ dài cạnh.

Giả sử H=<V, T> là một cây khung của đồ thị G. Ta gọi độ dài c(H) của cây khung H được tính bằng tổng độ dài các cạnh.

Bài toán được đặt ra là, trong số các cây khung của đồ thị hãy tìm cây khung có độ dài nhỏ nhất của đồ thị.

Xây dựng cây khung của đồ thị

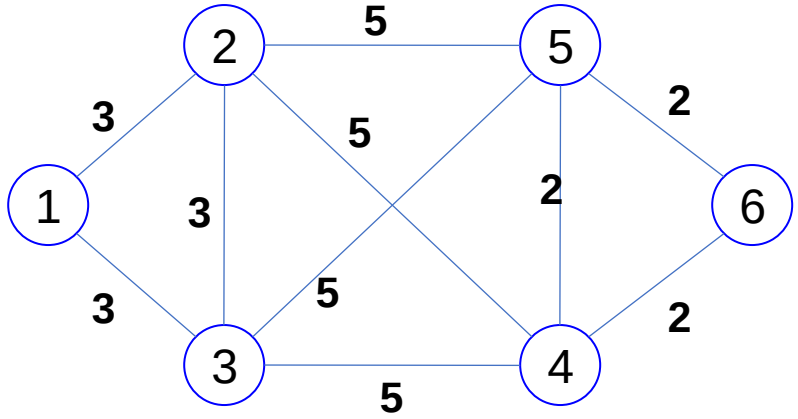
Bài toán tìm cây khung nhỏ nhất

Input:

```
6  10
1  2  3
1  3  3
2  3  3
2  4  5
2  5  5
3  4  5
3  5  5
4  5  2
4  6  2
5  6  2
```

Output:

```
6  5  15
4  5  2
4  6  2
1  2  3
1  3  3
2  4  5
```



Thuật toán Kruskal:

Begin

Bước 1 (Khởi tạo):

T = ∅; *//Khởi tạo tập cạnh cây khung là ∅*

d(H) = 0; *//Khởi tạo độ dài nhỏ nhất cây khung là 0*

Bước 2 (Sắp xếp):

<Sắp xếp các cạnh của đồ thị theo thứ tự giảm dần của trọng số>;

Bước 3 (Lặp):

while (|T<n-1| && E≠∅) do { *// Lặp nếu E≠∅ và |T|<n-1*

e = <Cạnh có độ dài nhỏ nhất>;

E = E \ {e}; *//Loại cạnh e ra khỏi đồ thị*

if (T∪{e} không tạo nên chu trình) then {

T = ∪{e}; *// Kết nạp e vào tập cạnh cây khung*

d(H) = d(H) + d(e); *//Độ dài của tập cạnh cây khung*

endif;

endwhile;

Bước 4 (Trả lại kết quả):

if (|T| < n-1) then <Đồ thị không liên thông>;

else

Return(T, d(H));

end.

Thuật toán PRIM (s):

Begin:

Bước 1 (Khởi tạo):

$V_H = \{s\}$; //Tập đỉnh cây khung thiết lập ban đầu là s

V = V\{s}; //Tập đỉnh V được bớt đi s

T = ∅; //Tập cạnh cây khung thiết lập ban đầu là ∅

d(H) = 0; //Độ dài cây khung được thiết lập là 0

Bước 2 (Lặp):

while (V≠ ∅) do {

e = <u, v>: cạnh có độ dài nhỏ nhất thỏa mãn u∈V, v∈V_H;

d(H) = d(H) + d(e); // Thiết lập đồ dài cây khung nhỏ nhất

T = T ∪ {e}; //Kết nạp e vào cây khung

V = V \{u}; // Tập đỉnh V bớt đi đỉnh u

$V_H = V_H \cup \{u\}$; // Tập đỉnh V_H thêm vào đỉnh u

endwhile;

Bước 3 (Trả lại kết quả):

if (|T|<n-1) then <Đồ thị không liên thông>;

else Return(T, d(H));

End.

Phát biểu bài toán

Phát biểu bài toán

Xét đồ thị G=<V, E>; trong đó | V| = n, | E | = m. Với mỗi cạnh (u, v) ∈ E, ta đặt tương ứng với nó một số thực A[u][v] được gọi là trọng số của cạnh. Ta sẽ đặt A[u,v]=∞ nếu (u, v)∉E.

Nhiệm vụ của bài toán là tìm đường đi ngắn nhất từ một đỉnh xuất phát s ∈ V (đỉnh nguồn) đến đỉnh cuối t ∈ V (đỉnh đích).

Độ dài của đường đi d(s,t) được gọi là khoảng cách ngắn nhất từ s đến t (trong trường hợp tổng quát d(s,t) có thể âm).

Nếu như không tồn tại đường đi từ s đến t thì độ dài đường đi d(s,t)= ∞.

Tìm đường đi ngắn nhất

Trường hợp 1:

Nếu s cố định và t thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất từ s đến tất cả các đỉnh còn lại trên đồ thị.

Đối với đồ thị có trọng số không âm => Thuật toán **Dijkstra**.

Đối với đồ thị có trọng số âm nhưng không tồn tại chu trình âm => Thuật toán **Bellman-Ford**.

Trong trường hợp đồ thị có chu trình âm, bài toán không có lời giải.

Trường hợp 2:

Nếu s thay đổi và t cũng thay đổi, khi đó bài toán được phát biểu dưới dạng tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh của đồ thị.
Bài toán luôn có lời giải trên đồ thị không có chu trình âm.

Đối với đồ thị có trọng số không âm, bài toán được giải quyết bằng cách thực hiện lặp lại n lần thuật toán **Dijkstra**.

Đối với đồ thị không có chu trình âm, bài toán có thể giải quyết bằng thuật toán **Floyd**.

Thuật toán Dijkstra (s): //s∈ V là một đỉnh bất kỳ của G = <V,E>
Begin

Bước 1 (Khởi tạo):

d[s]=0; *//Gán nhãn của đỉnh s là 0*

T = V\{s}; *// T là tập đỉnh có nhãn tạm thời*

for each v∈ V do { *//Sử dụng s gán nhãn cho các đỉnh còn lại*

d[v] = A[s,v];

truoc[v]=s;

endfor;

Bước 2 (Lặp):

while (T≠ ∅) do {

Tìm đỉnh u∈T sao cho d[u] = min { d[z] | z∈T};

T= T\{u}; *//cố định nhãn đỉnh u*

for each v∈ T do { //Sử dụng u, gán nhãn lại cho các đỉnh

if (d[v] > d[u] + A[u, v]) then {

d[v] = d[u] + A[u, v]; //Gán lại nhãn cho đỉnh v;

truoc[v] = u;

endif;

endfor;

endwhlie;

Bước 3 (Trả lại kết quả):

Return (d[s], truoc[s]);

End.

Thuật toán Bellman-Ford (s): *//s ∈ V là đỉnh bất kỳ của đồ thị*

Begin:

Bước 1 (Khởi tạo):

```
for v∈V do { //Sử dụng s gán nhãn cho các đỉnh v∈V
        D[v] = A[s][v];
        Truoc[v] = s;
}
```

Bước 2 (Lặp) :

```
D[s] = 0; K=1;
while (K<=N-2 ) { //N-2 vòng lặp
        for v∈V\{s} do { //Lấy mỗi đỉnh v∈V\s
                for u∈V do { //Gán nhãn cho v
                        if (D[v] > D[u] + A[u][v] ) {
                                D[v]= D[u] + A[u][v];
                                Truoc[v] = u;
                        endif;
                endfor;
        endfor;
endwlie;
```

Bước 3 (Trả lại kết quả):

```
Return( D[v], Truoc[v]: v∈U);
```

End.

Thuật toán Floy:

Begin:

Bước 1 (Khởi tạo):

```
for (i=1; i≤ n; i++) {
    for (j =1; j≤ n; j++) {
        d[i,j] = a[i, j];
        p[i,j] = i;
    }
}
```

Bước 2 (lặp) :

```
for (k=1; k≤ n; k++) {
    for (i=1; i≤ n; i++){
        for (j =1; j≤ n; j++) {
            if (d[i,j] > d[i, k] + d[k, j]) {
                d[i, j] = d[i, k] + d[k, j];
                p[i,j] = p[k, j];
            }
        }
    }
}
```

Bước 3 (Trả lại kết quả):

```
Return (p([i,j], d[i,j]: i, j∈V);
```

THANK YOU FOR YOUR ATTENTION!