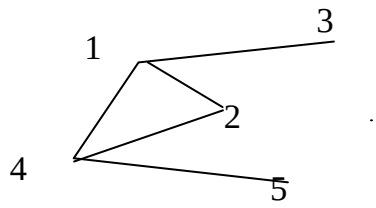
Chương 3

CÁC THUẬT TOÁN DUYỆT VÀ TÌM KIẾM TRÊN ĐỒ THỊ

1. Tìm theo chiều sâu (Depth First Search)

*** Ý tưởng**

- Từ đỉnh v_1 nào đó chưa thăm, thăm v_1. Tìm đỉnh v_2 chưa thăm kề với v_1, thăm v_2. Tìm đỉnh v_3 chưa thăm kề với v_2, thăm v_3… Lặp lại việc thăm cho tới khi tất cả các đỉnh đều được thăm.
- Nếu tại một đỉnh v_i nào đó, không còn đỉnh nào kề với v_i là chưa thăm thì quay trở lại đỉnh v_{i-1} tìm đỉnh kề chưa thăm nếu có của v_{i-1} để thăm.



Nếu bắt đầu từ đỉnh 4, thì thứ tự thăm có thể là:
4,1,2,3,5.

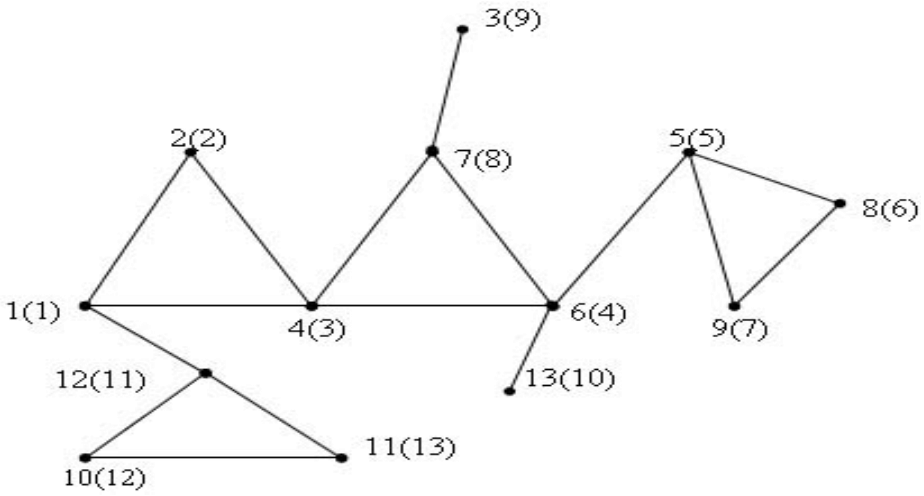
*** Cài đặt**

```
const int Max=20;
int a[Max][Max],n, tham[Max];
void DFS1(int i){
        cout<<i<<" ";
        tham[i]=1;
        for (int j=1;j<=n;j++)
                if (tham[j]==0&&a[i][j]==1) DFS1(j);
}
void DFS(){
        for (int i=1;i<=n;i++)  tham[i]=0;
        for (i=1;i<=n;i++)
                if (tham[i]==0)  DFS1(i);
}
```

*** Nhận xét**

- Mỗi đỉnh sẽ được thăm đúng một lần.
- Thứ tự thăm phụ thuộc vào đỉnh xuất phát và phụ thuộc vào việc chọn đỉnh kề để thăm. Để kết quả thăm là duy nhất ta qui ước việc chọn đỉnh kề để thăm là theo thứ tự từ đỉnh có chỉ số nhỏ đến đỉnh có chỉ số lớn.
- DFS1(v) thăm tất cả các đỉnh thuộc cùng một thành phần liên thông chứa đỉnh v. Số lần DFS gọi DFS1 chính là số thành phần liên thông của đồ thị.

*** Ví dụ**



Thứ tự thăm theo DFS là: 1, 2, 4, 6, 5, 8, 9, 7, 3, 13, 12, 10, 11.

2. Tìm theo chiều rộng (Breadth First Search)

*** Ý tưởng**

- Từ đỉnh v_1 nào đó chưa thăm, thăm v_1 (v_1 gọi là đỉnh ở mức 0). Tìm tất cả các đỉnh chưa thăm kề với v_1 (gọi là những đỉnh ở mức 1) và lần lượt thăm tất cả những đỉnh ở mức 1. Sau đó lại lần lượt thăm tất cả các đỉnh chưa thăm kề với những đỉnh ở mức 1 (gọi là những đỉnh ở mức 2),...
- Lặp lại việc thăm cho đến khi tất cả các đỉnh được thăm thì ngừng.

*** Mã giả**

B1: Từ đỉnh i nào đó chưa thăm, cất i vào hàng đợi, đánh dấu i thăm rồi.
B2: Lấy từ hàng đợi một đỉnh i, thăm i, rồi cất các đỉnh j chưa thăm, j kề i vào hàng đợi, đánh dấu j thăm rồi.
B3: Lặp lại B2 cho tới khi hàng đợi rỗng.

*** Cài đặt BFS**

```
void BFS1(int i){
        int queue[max],d,c;
        d=c=0; queue[c++]=i; tham[i]=1;
        while( d!=c) {
                i=queue[d++]; cout<<i;
                for( int j=1;j<=n;j++)
                        if (tham[j]==0&&a[i][j]==1){
                                queue[c++]=j;
                                tham[j]=1;
                        }
        }
}
void BFS(){
        for (int i=1;i<=n;i++)  tham[i]=0;
        for (i=1;i<=n;i++)
                if (tham[i]==0)  BFS1(i);
}
```

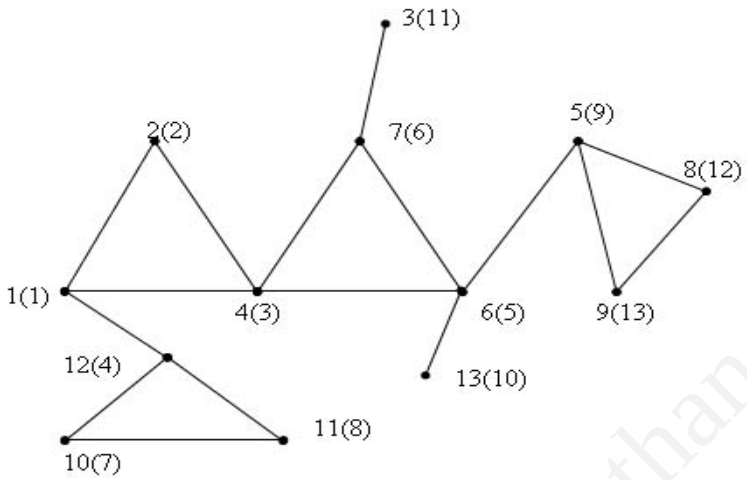
*** Nhận xét:**

- Mỗi đỉnh sẽ được thăm đúng một lần.
- Thứ tự thăm phụ thuộc vào đỉnh xuất phát và phụ thuộc vào việc chọn đỉnh kề để thăm. Để kết quả thăm là duy nhất ta qui ước việc chọn đỉnh kề để thăm là theo thứ tự từ đỉnh có chỉ số nhỏ đến đỉnh có chỉ số lớn.
- BFS1(v) thăm tất cả các đỉnh thuộc cùng một thành phần liên thông chứa đỉnh v. Số lần BFS gọi BFS1 chính là số thành phần liên thông của đồ thị.
- So sánh BFS1() với DFS1()

DFS1()	BFS1()
Từ một đỉnh chỉ chọn một đỉnh kề chưa thăm để thăm	Từ một đỉnh thăm tất cả các đỉnh kề chưa thăm

- DFS() và BFS() hoàn toàn giống nhau.

*** Ví dụ:**



1	2	3	4	5	6	7	8	9	10	11	12	13
1												
	2	4	12									
		4	12									
			12	6	7							
				6	7	10	11					
					7	10	11	5	13			
						10	11	5	13	3		
							11	5	13	3		
								5	13	3		
									13	3	8	9
										3	8	9
											8	9
												9

Thứ tự thăm theo BFS là: 1, 2, 4, 12, 6, 7, 10, 11, 5, 13, 3, 8, 9.

3. Tìm đường đi từ s đến t

*** Ý tưởng**

Gọi hàm DFS1(s) hoặc BFS1(s) để thăm tất cả các đỉnh thuộc cùng một thành phần liên thông với s. Nếu sau khi thực hiện xong mà thăm[t] vẫn bằng 0 thì không có đường đi từ s đến t, ngược lại thì có đường đi từ s đến t. Để ghi nhận đường đi, ta dùng thêm biến Trước[j] để ghi nhận đỉnh ở ngay trước đỉnh j trên đường đi từ s đến v.

*** Cài đặt**

```
int tham[Max]={0}, truoc[Max];
//duyệt một thành phần liên thông dùng DFS1() hoặc BFS1()
void DFS1(int i){
        tham[i]=1;
        for (int j=1;j<=n;j++)
                if (tham[j]==0 && a[i][j]==1){
                        truoc[j]=i;
                        DFS1(j);
                }
}
void BFS1(int i){
        int queue[max],d,c;
        d=c=0; queue[c++]=i; tham[i]=1;
```

```
        while( d!=c) {
                i=queue[d++]; cout<<i;
                for( int j=1;j<=n;j++)
                        if (tham[j]==0&&a[i][j]==1){
                                truoc[j]=i;
                                queue[c++]=j;
                                tham[j]=1;
                        }
        }
}
void main(){
        int s,t,v;
        cin>>s>>t;
        BFS1(s); //DFS1(s);
        if (tham[t]==0) cout<<“khong co duong di”;
        else{
                cout<<t ; v=t ;
                do{
                        v=truoc[v];
                        cout<< “← “<<v ;
                }while(v !=s);
        }
}
```

*** Chú ý**

Đường đi tìm theo BFS là đường đi ngắn nhất theo số cạnh từ s đến t nhưng DFS thì không.

4. Tìm các thành phần liên thông

*** Ý tưởng**

- Do số thành phần liên thông bằng số lần DFS() gọi DFS1() hoặc BFS() gọi BFS1(), nên ta dùng biến stplt để đếm số thành phần liên thông, mỗi lần DFS() gọi DFS1() hoặc BFS() gọi BFS1() ta tăng biến stplt lên 1.
- Khi thăm i thay vì gán thăm[i]=1 ta gán thăm[i]=stplt (số hiệu thành phần liên thông chứa i).

*** Cài đặt**

```
const int Max=20;
int a[Max][Max],n, tham[Max];
int stplt=0;

void DFS1(int i){
        cout<<i<<" ";
        tham[i]=stplt;
        for (int j=1;j<=n;j++)
                if (tham[j]==0&&a[i][j]==1)  DFS1(j);
}
void DFS(){
        for (int i=1;i<=n;i++)  tham[i]=0;
        for (i=1;i<=n;i++)
                if (tham[i]==0)  {stplt++; DFS1(i);}
}
void BFS1(int i){
        int queue[max],d,c;
        d=c=0; queue[c++]=i; tham[i]=1;
```

```
        while( d!=c) {
                i=queue[d++]; cout<<i;
                for( int j=1;j<=n;j++)
                        if (tham[j]==0&&a[i][j]==1){
                                queue[c++]=j;
                                tham[j]=1;
                        }
        }
}
void BFS(){
        for (int i=1;i<=n;i++)  tham[i]=0;
        for (i=1;i<=n;i++)
                if (tham[i]==0)  {stplt++; DFS1(i);}
}
void main(){
        DFS(); //BFS();
        if (stplt==1)     cout<<"\nDo thi lien thong";
        else{
                for (i=1;i<=stplt;i++){
                        cout<<"\nTplt thu ”<<i <<”:”;
                        for  (j=1;j<=n;j++)
                                if (tham[j]==i)  cout<<j;
                }
        }
}
```

***Bài tập chương 3**

Bài 1: Cài đặt thuật toán duyệt theo chiều sâu. Đồ thị được lưu trong file dạng danh sách cạnh.

Bài 2: Cài đặt thuật toán duyệt theo chiều rộng. Đồ thị được lưu trong file dạng danh sách cạnh.

Bài 3: Cài đặt thuật toán kiểm tra đồ thị VH hoặc CH có liên thông hay không? Đồ thị được lưu trong file dạng danh sách cạnh. Sử dụng hai cách: DFS hoặc BFS.

Bài 4: Cài đặt thuật toán tìm các thành phần liên thông trên đồ thị VH, cho biết mỗi TPLT gồm những đỉnh nào. Đồ thị được lưu trong file dạng danh sách cạnh. Sử dụng hai cách: DFS hoặc BFS.

Bài 5: Cài đặt thuật toán tìm đường đi giữa 2 đỉnh cho trước. Đồ thị được lưu trong file dạng danh sách cạnh. Sử dụng hai cách: DFS hoặc BFS.

Bài 6:Cài đặt thuật toán tối ưu đồ thị gồm hai bước:

B1- bổ sung thêm số cạnh ít nhất để đồ thị liên thông.

B2- loại bỏ số cạnh nhiều nhất sao cho đồ thị vẫn liên thông.

Đồ thị được lưu trong file dạng danh sách cạnh. Sử dụng hai cách: DFS hoặc BFS.

Bài 7: Cài đặt thuật toán tìm tất cả các chu trình của đồ thị. Đồ thị được lưu trong file dạng danh sách cạnh. Sử dụng hai cách: DFS hoặc BFS.