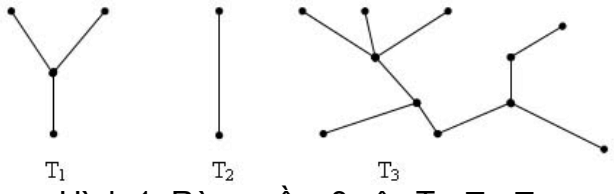
CHƯƠNG 5

CÂY VÀ CÂY KHUNG CỦA ĐỒ THỊ

1. Cây và các tính chất cơ bản của cây

- Cây là đồ thị vô hướng, liên thông, không có chu trình.
- Rừng là đồ thị vô hướng, không có chu trình. Như vậy, rừng là đồ thị mà mỗi thành phần liên thông của nó là một cây.

Ví dụ:



Hình 1. Rừng gồm 3 cây T_1, T_2, T_3.

1.1 Định lý 1:

Giả sử G=(V,E) là đồ thị vô hướng n đỉnh. Khi đó các mệnh đề sau đây là tương đương:
(1) T là cây;
(2) T không chứa chu trình và có n-1 cạnh;
(3) T liên thông và có n-1 cạnh;
(4) T liên thông và mỗi cạnh của nó là cầu;
(5) Hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi đơn;
(6) T không chứa chu trình nhưng thêm vào một cạnh ta thu được đúng một chu trình.

Chứng minh:
Ta sẽ chứng minh định lý theo sơ đồ sau:
(1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (4) $\Rightarrow$ (5) $\Rightarrow$ (6) $\Rightarrow$ (1)

(1) $\Rightarrow$ (2): T là cây nên T không chứa chu trình. Ta sẽ chứng minh "cây có n đỉnh sẽ có n-1 cạnh". Rõ ràng khẳng định đúng với n=1. Giả sử n>1. Trước hết nhận xét rằng trong mọi cây T có n đỉnh đều tìm được ít nhất một đỉnh là đỉnh treo (đỉnh có bậc là 1). Thực vậy, gọi $v_1, v_2, \ldots, v_k$ là đường đi dài nhất (theo số cạnh) trong T. Khi đó rõ ràng v_1 và v_k là các đỉnh treo, vì từ v_1 (hoặc v_k) không có cạnh nối với bất cứ đỉnh nào trong số các đỉnh $v_2, v_3, \ldots, v_k$ (do đồ thị không chứa chu trình), cũng như với bất cứ đỉnh nào khác của đồ thị (do đường đi đang xét dài nhất). Loại bỏ v_1 và cạnh (v_1,v_2) khỏi T ta thu được cây T_1 với n-1 đỉnh, mà theo giả thiết qui nạp có n-2 cạnh. Vậy cây T có n-2+1 = n-1 cạnh.

(2) $\Rightarrow$ (3) Giả sử T không liên thông. Khi đó T có k ($k \geq 2$) phần liên thông $T_1, T_2, \ldots, T_k$. Do T không chứa chu trình nên mỗi T_i (i=1,2,. . .,k) cũng không chứa chu trình, vì thế mỗi T_i là cây. Do đó nếu gọi $n(T_i)$ và $e(T_i)$ là số đỉnh và cạnh của T_i.
Ta có:

$$e(T_i) = n(T_i) - 1,\ i = 1, 2, \ldots, k,$$

suy ra

$$n-1 = e(T) = e(T_1) + \ldots + e(T_k) = n(T_1) + \ldots + n(T_k) - k = n(T) - k < n-1 \text{ (do } k \geq 2)$$

Mâu thuẫn chứng tỏ là T liên thông.

(3) $\Rightarrow$ (4) Việc loại bỏ một cạnh bất kỳ khỏi T dẫn đến đồ thị với n đỉnh và n-2 cạnh rõ ràng là đồ thị không liên thông. Vậy mọi cạnh trong T đều là cầu.

(4) ⇒ (5) Do T là liên thông nên hai đỉnh bất kỳ của nó được nối với nhau bởi một đường đi đơn. Nếu có cặp đỉnh nào của T có hai đường đi đơn khác nhau nối chúng, thì từ đó suy ra đồ thị chứa chu trình, và vì thế các cạnh trên chu trình này không phải là cầu (mâu thuẫn giả thiết)

(5) ⇒ (6) T không chứa chu trình, bởi vì nếu có chu trình thì suy ra tìm được cặp đỉnh của T được nối với nhau bởi hai đường đi đơn. Bây giờ, nếu thêm vào T một cạnh e nối hai đỉnh u và v nào đó của T. Khi đó cạnh này cùng với đường đi đơn nối u với v sẽ tạo thành chu trình trong T. Chu trình thu được này là duy nhất, vì nếu thu được nhiều hơn một chu trình thì suy ra trong T trước đó phải có sẵn chu trình.

(6) ⇒ (1) Giả sử T không liên thông. Khi đó gồm ít ra là 2 thành phần liên thông. Vì vậy, nếu thêm vào T một cạnh nối hai đỉnh thuộc hai thành phần liên thông khác nhau ta không thu được thêm một chu trình nào cả. Điều đó mâu thuẫn với giả thiết (6).

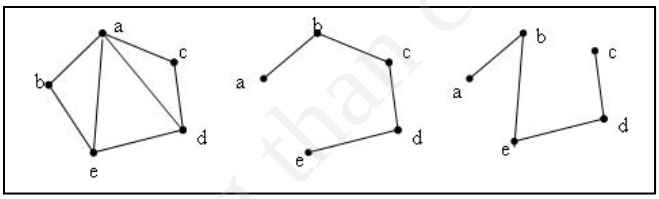
Định lý được chứng minh.

2. Cây khung của đồ thị

G=(V,E) là đồ thị vô hướng liên thông. Cây khung của đồ thị G là cây T=(V,F) với $F \subset E$.
Như vậy Cây khung là

- Cây (liên thông, không chu trình)
- Có cùng số đỉnh với đồ thị nhưng số cạnh có thể ít hơn

Hình 2. Đồ thị và các cây khung của nó



2.1 Định lý 2 (Cayley): (chấp nhận)

Số cây khung của đồ thị đầy đủ K_n là n^{n-2}.

Ví dụ: n=3=> có 3 cây khung, n=4 có 16 cây khung

Nhận xét: số lượng cây khung của đồ thị có thể rất lớn.

2.2 Các thuật toán tìm cây khung

2.2.1 Sử dụng thuật toán duyệt theo chiều sâu

```
void DFS1(v)
{
        tham[v]=1; //ghi nhận là đã thăm v để về sau không thăm nữa.
        For  (u∈ Ke(v)) // xét tất cả các đỉnh u kề với v
                If  (!tham[u])
                {
                        T=T∪ (v,u); //them canh (v,u) vao tap T
                        DFS1(u); //neu u chua thăm, thăm u
```

```
            }
}
```

2.2.2 Sử dụng thuật toán duyệt theo chiều rộng

```
void BFS1(v)
{
        queue=ϕ ; //khởi tạo hàng đợi là rỗng
        push(queue,v); //cất v vào queue
        tham[v]=1; //ghi nhận là đã thăm v để về sau không thăm nữa.
        while  (queue ≠ ϕ ) // xét tất cả các đỉnh u kề với v
        {
                v=pop(queue); //lay v tu queue
                for  (u∈ Ke(v)) // xét tất cả các đỉnh u kề với v
                        If  (!tham[u])
                        {
                                push(queue,u); tham[u]=1;
                                T=T∪ (v,u); //them canh (v,u) vao tap T
                        }
        }
}
```

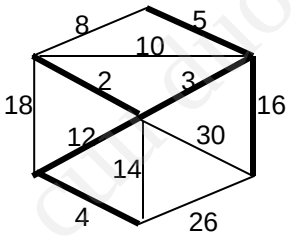
3. Tìm cây khung nhỏ nhất

Cho G=(V,E) là đồ thị vô hướng liên thông, mỗi cạnh e có trọng số c(e)>=0. Giả sử H=(V,T) là cây khung của đồ thị G. Ta gọi độ dài c(H) của cây khung H là tổng trọng số các cạnh của nó:

$$c(H) = \sum_{e \in T} c(e)$$

Bài toán đặt ra là tìm cây khung với độ dài nhỏ nhất.

Ví dụ: Cây khung nhỏ nhất được chỉ ra bởi các cạnh tô đậm



Ví dụ:

* Bài toán xây dựng hệ thống đường sắt:

Xây dựng một hệ thống đường sắt nối n thành phố sao cho giữa hai thành phố bất kỳ luôn có đường đi và chi phí xây dựng nhỏ nhất.

Bài toán đặt ra chính là bài toán tìm cây khung nhỏ nhất trên đồ thị đầy đủ n đỉnh, mỗi đỉnh tương ứng với một thành phố, với độ dài trên các các cạnh chính là chi phí xây dựng đường ray nối hai thành phố tương ứng.

* Bài toán nối mạng máy tính:

Trong một cơ quan có n phòng ban, mỗi phòng có một máy tính. Cần nối n máy tính thành một mạng LAN sao cho chi phí ít nhất. Biết rằng chi phí nối máy i với máy j là c[i,j]

*** Nhận xét:**

Nếu xét tất cả các cây khung và chọn ra cây khung nhỏ nhất thì trong trường hợp đồ thị đầy đủ, sẽ đòi hỏi thời gian cỡ n^{n-2} , do đó không thể thực hiện được khi số đỉnh nhiều. Rất may là đối với bài toán cây khung nhỏ nhất chúng ta đã có những thuật toán rất hiệu quả để giải chúng.

3.1 Thuật toán Kruskal

***Ý tưởng**

- Sắp xếp danh sách cạnh theo trọng số tăng dần. (Để sắp xếp nhanh nên sử dụng thuật toán Heap Sort hoặc Quick Sort)
- Lần lượt chọn n-1 cạnh trong danh sách cạnh từ trái sang phải sao cho cạnh được chọn không tạo thành chu trình với những cạnh đã chọn trước đó.
- Thuật toán kiểm tra chu trình
  - ✓ Ban đầu xem có n cây con, mỗi cây chỉ có một đỉnh và là gốc cây.
  - ✓ Mỗi cây có một đỉnh gọi là gốc, đỉnh gốc có trước bằng -1
  - ✓ Xét cạnh (i,j), giả sử i thuộc cây gốc u, j thuộc cây gốc v. Nếu u khác v thì cạnh (i,j) chọn được và ta gán truoc[u]=v để ghép cây gốc u và cây gốc v thành một cây có gốc là v.

***Cài đặt**

```
int dsc[max][3];//danh sach canh
int n,  m;//so dinh va so canh
int truoc[max]={-1};

int Goc(int i){
        while (truoc[i] ! = -1)  i=truoc[i];
        return i;
}
void Kruskal(){
        SapXepDSC();
        int sc=0, cd=0;
        int ck[max][3];
        for (int k=0; k<m;k++){
                int i=dsc[k][0]; int j=dsc[k][1]; int l=dsc[k][2];
                int u=Goc(i), v=Goc(j);
                if (u!=v){
                        ck[sc][0]=i;ck[sc][1]=j; ck[sc][2]=l;
                        sc++; cd+=l; truoc[u]=v;
                        if(sc==n-1) break;
                }
        }
        if(sc<n-1) cout<<"khong co cay khung";
        else{
                for (int k=0; k<n-1;k++) cout<<"("<<ck[k][0]<<","<< ck[k][1]<<","<< ck[k][2]<<") ";
                cout<<endl<<"Do dai = "<<cd<<endl;
        }
}
```

3.2 Thuật toán Prim

Prim tốt hơn Kruskal vì không cần sắp xếp và không cần phát hiện chu trình

***Ý tưởng**

Cây khung ban đầu chỉ có một đỉnh bất kỳ gọi là s và chưa có cạnh nào, ta lần lượt bổ sung đỉnh và cạnh cho đến khi đủ n-1 cạnh thì ngừng.

Mỗi đỉnh v gán hai giá trị:

d[v]: là khoảng cách ngắn nhất từ v đến cây khung.

near[v]: là đỉnh của cây khung gần v nhất.

ban đầu gán: ∀ v∈V; d[v] = trọng số cạnh (s,v), hoặc ∞ nếu không có cạnh (s,v) và near[v]=s;
đánh dấu s chọn rồi
lần lượt lặp (n-1) lần, mỗi lần chọn một đỉnh, một cạnh để bổ sung vào cây khung.

đỉnh được chọn là đỉnh u chưa chọn và có d[u] nhỏ nhất , cạnh được chọn là (u,near[u])

đánh dấu u chọn rồi

cập nhật giá trị cho các đỉnh v thỏa điều kiện: v chưa chọn, v kế u, d[v]>a[u][v]. Cập nhật bằng cách gán d[v]=a[u][v] và near[v]=u;

***Cài đặt**

```
int a[max][max], n;
void Prim(){
	int tham[max]={0},d[max],Near[max],ck[max];
	int sc=0,cd=0;
	int u,v,s=1,vc=30000;
	for(v=1;v<=n;v++){
		tham[v]=0;d[v]=a[s][v];Near[v]=s;
	}
	tham[s]=1;
	for(int i=1;i<=n-1;i++){
		int temp=vc;
		for ( v=1;v<=n;v++)
			if(tham[v]==0&&d[v]<temp){
				temp=d[v]; u=v;
			}
		If(temp==vc) break;
		ck[sc][0]=u;ck[sc][1]=Near[u]; ck[sc][2]= a[u][Near[u]];
		cd+=a[u][Near[u]]; tham[u]=1; sc++;
		for (v=1;v<=n;v++)
			if (tham[v]==0&&d[v]>a[u][v]){
				d[v]=a[u][v];Near[v]=u;
			}
	}
	if(sc<n-1) cout<<”khong co cay khung”;
	else{
		for (int k=0; k<n-1;k++) cout<<"("<<ck[k][0]<<","<< ck[k][1]<<","<< ck[k][2]<<") ";
		cout<<endl<<"Do dai = "<<cd<<endl;
	}
}
```

Ví dụ : Tìm cây khung nhỏ nhất cho đồ thị xét trong hình 4 theo thuật toán Prim.
Ma trận trọng số của đồ thị có dạng

C =		1	2	3	4	5	6
	1	0	33	17	∞	∞	∞
	2	33	0	18	20	∞	∞
	3	17	18	0	16	4	∞
	4	∞	20	16	0	9	8
	5	∞	∞	4	9	0	14
	6	∞	∞	∞	8	14	0

Bảng dưới đây ghi nhãn của các đỉnh trong các bước lặp của thuật toán, đỉnh đánh dấu * là đỉnh được chọn để bổ sung vào cây khung (khi đó nhãn của nó không còn bị biến đổi trong các bước lặp tiếp theo, vì vậy ta đánh dấu * để ghi nhận điều đó):

Bước lặp	Đỉnh 1	Đỉnh 2	Đỉnh 3	Đỉnh 4	Đỉnh 5	Đỉnh 6	V_H	T
Khởi tạo	[0,1]	[33,1]	[17,1]*	[∞ ,1]	[∞ ,1]	[∞ ,1]	1	ϕ
1	-	[18,3]	-	[16,3]	[4,3]*	[∞ ,1]	1,3	(3,1)
2	-	[18,3]	-	[9,5]	-	[14,5]	1,3,5	(3,1), (5,3)
3	-	[18,3]	-	-	-	[8,4]	1,3,5,4	(3,1), (5,3), (4,5)
4	-	[18,3]*	-	-	-	-	1,3,5,4,6	(3,1),(5,3),(4,5),(6,4)
5	-	-	-	-	-	-	1,3,5,4,6,2	(3,1), (5,3), (4,5), (6,4), (2,3)

* Bài tập chương 5

Bài 1: cài đặt thuật toán Kruskal
Bài 2: cài đặt thuật toán Prim