CHƯƠNG 5. CÂY VÀ CÂY KHUNG ĐỒ THỊ

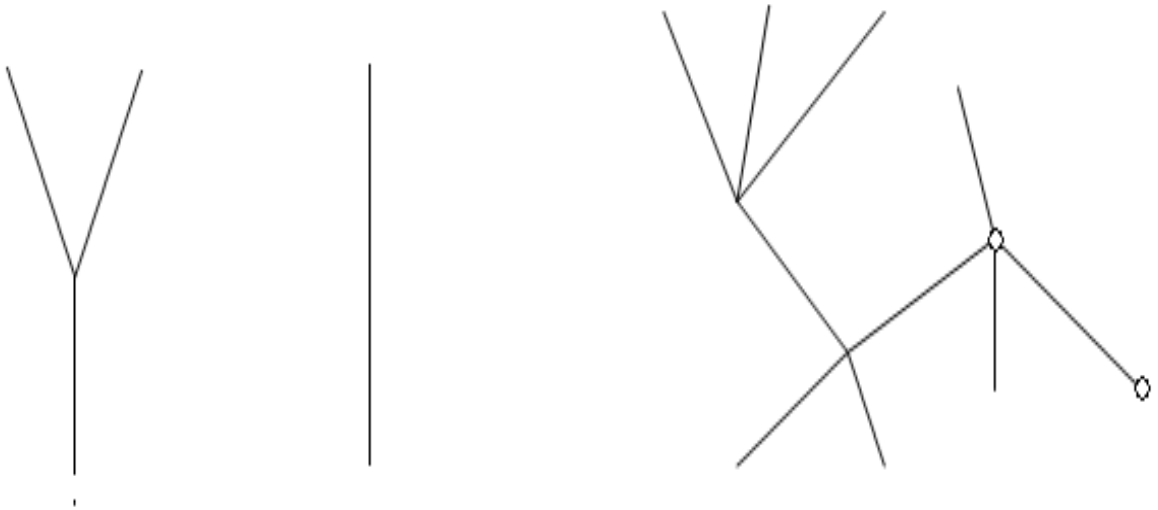
NỘI DUNG:

- Định nghĩa và tính chất
- Xây dựng cây khung của đồ thị
- Bài toán tìm cây khung nhỏ nhất

5.1 Định nghĩa và các tính chất cơ bản

1. Cây tự do

- Cây tự do (không gốc) T là đồ thị vô hướng liên thông và không có chu trình đơn.
- Đồ thị F là một rừng $\Leftrightarrow$ mỗi thành phần liên thông của F là một cây.
- Đồ thị F là một rừng $\Leftrightarrow$ F không có chu trình đơn.



Cây tự do (không gốc)

Định lý 1:

T= (V, E) là một đồ thị vô hướng có n đỉnh.

Các mệnh đề sau 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 T là cầu ;

(5) Hai đỉnh bất kỳ của T có duy nhất một đường đi nối đến nhau ;

(6) T không chứa chu trình nhưng nếu thêm 1 cạnh thì thu được đúng 1 chu trình.

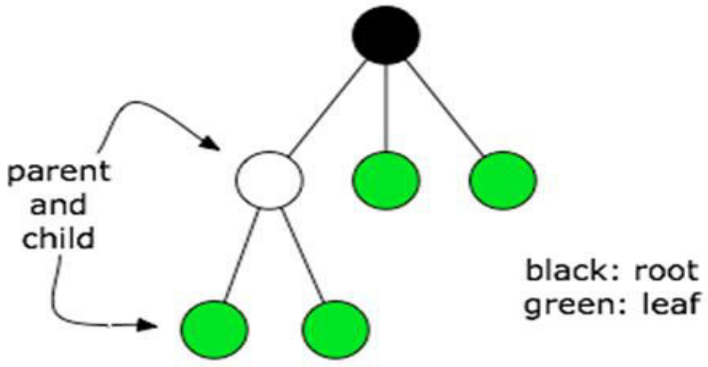
2. Cây có gốc

Định nghĩa 1: Cây là tập hợp hữu hạn các nút thỏa mãn:

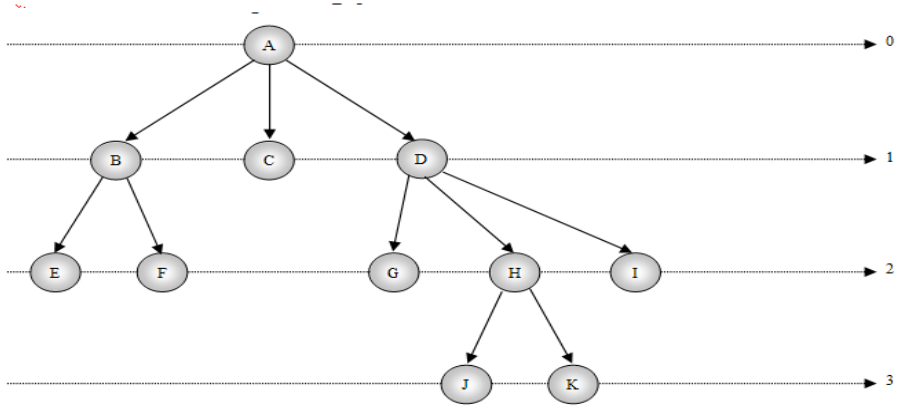
- Có một nút gọi là gốc
- Có quan hệ phân cấp "cha-con" giữa các nút.

Định nghĩa 2 (đệ qui): - Nếu T chỉ gồm 1 nút $\Rightarrow$ T là một cây với gốc là chính nút đó

- Nếu $T_1, \ldots, T_n$ ($n \geq 1$) là các cây có gốc tương ứng $r_1, \ldots, r_n \Rightarrow$ T là cây với gốc r được tạo thành bằng cách cho r thành nút cha của các nút $r_1, \ldots, r_n$.



Cây có gốc tam phân



Các mức của cây

Các ví dụ về cây:

- Mục lục của một cuốn sách
- Cấu trúc thư mục trên đĩa máy tính.
- Dùng cây để biểu diễn biểu thức số học.

Một số khái niệm:

- **Nút gốc**: là nút của cây T không có cha.
- **Nút cha**: Nút r là cha của các nút r_1, ..., r_n $\Rightarrow$ Nút gốc là nút không có nút cha.
- **Nút con**: Các nút r_1, …, r_n gọi là nút con của r
- **Bậc của một nút**: là số các nút con của nút đó.
- **Bậc của một cây T**: là bậc lớn nhất của các nút $\in$ T. T có bậc m $\Rightarrow$ T gọi là cây m-phân.

Nút lá: là nút có bậc = 0 $\Rightarrow$ nút lá không có nút con.

Nút nhánh (nút trong hay nút trung gian): là nút vừa có con vừa có cha.

T là cây m-phân đầy đủ $\Leftrightarrow$ mỗi nút nhánh có đúng m con.

- **Cây con: Mỗi cây có gốc tại nút a ∈ T là một cây con của T.**
- **Đường đi:** Dãy các đỉnh r_1, …, r_k, trong đó r_i là cha của r_{i+1} gọi là đường đi từ r_1 đến r_k. Độ dài của đường đi là số nút trên đường đi -1 (Đường đi trên có độ dài k-1).
- **Mức của nút:** là độ dài đường đi từ gốc đến nút (độ cao của nút) ⇒ Nút gốc có mức 0.
- **Chiều cao của cây: l**à mức lớn nhất của các nút ∈ T.
- **Cây được sắp thứ tự:** là cây mà mỗi cây con được sắp theo một thứ tự nào đó.
- **Cây gán nhãn: l**à cây mà mỗi đỉnh được gắn với một giá trị (nhãn).

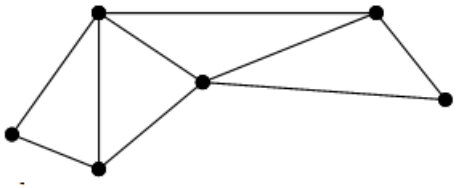
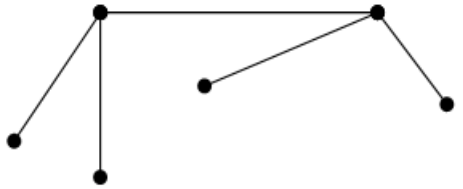
Cây nhị phân:

Cây nhị phân là cây mà mỗi nút có không quá 2 con. Phân biệt cây con bên trái và cây con bên phải.

3. Định nghĩa cây khung

Định nghĩa 1. Cho G là một đơn đồ thị.

Một cây T gọi là *cây khung* của G $\Leftrightarrow$ T là đồ thị con của G và chứa tất cả các đỉnh của G.

 Đồ thị G	 Một cây khung T của G

Định lý 2. Một đơn đồ thị là liên thông nếu và chỉ nếu nó có cây khung.

Chứng minh. Giả sử đồ thị G có cây khung T chứa tất cả các đỉnh của G và có đường đi trong T giữa hai đỉnh bất kỳ $\Rightarrow$ có đường đi trong G giữa hai đỉnh bất kỳ của nó $\Rightarrow$ G là liên thông.

Giả sử G là liên thông. Nếu G không phải là cây thì G có chu trình đơn. Xóa đi một cạnh trong các chu trình đơn này. Đồ thị nhận được chứa một số cạnh ít hơn nhưng vẫn còn chứa tất cả các đỉnh của G và liên thông. Nếu đồ thị con này không phải là cây thì nó chứa chu trình đơn. Cũng giống như trên, xóa đi một cạnh của chu trình đơn. Lặp lại quá trình này cho đến khi không còn chu trình đơn. Điều này có thể vì chỉ có hữu hạn cạnh trong đồ thị. Đồ thị con cuối cùng sau khi xóa đi các cạnh của các chu trình là cây khung.

5.2 Bài toán tìm cây khung

1) Đặt bài toán:

Input: Đồ thị G gồm n đỉnh cho bởi danh sách kề; Đỉnh u;

Output: Cây khung T của G bắt đầu từ đỉnh u;

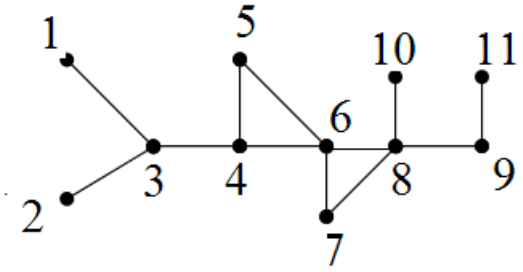
2. Xây dựng cây khung bằng thuật toán DFS

Thuật toán TreeDfs(u):

- Tạo cây T bằng thuật toán DFS bắt đầu từ đỉnh u;
- Nếu số đỉnh của T bằng n thì xuất kết quả T;
- Nếu số đỉnh của T nhỏ hơn n thì xuất thông báo: Không có cây khung.;

Độ phức tạp tính toán: Giải thuật tìm cây khung có độ phức tạp O(n).

Ví dụ: Tìm cây khung của đồ thị G cho dưới đây:



- Tìm kiếm theo chiều sâu bắt đầu từ 1:

T = {(1, 3), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9), (9, 11), (8, 10)}.

- Tìm kiếm theo chiều rộng bắt đầu từ 1:

T = {(1, 3), (2, 3), (3,4), (4, 5), (4, 6), (6, 7), (6, 8), (8, 9), (8, 10), (9, 11)}.

Ghi chú: - Cây khung T tìm kiếm theo chiều rộng bắt đầu từ 1 gồm các đường đi ngắn nhất xuất phát từ 1 đến các đỉnh khác.

- Đồ thị đầy đủ K_n có n^{n-2} cây khung khác nhau.

3. Xây dựng cây khung bằng thuật toán BFS

Thuật toán TreeBfs(u):

- Tạo cây T bằng thuật toán BFS bắt đầu từ đỉnh 1.
- Nếu số đỉnh của T bằng n thì xuất kết quả T.
- Nếu số đỉnh của T nhỏ hơn n thì xuất thông báo: Không có cây khung.

Độ phức tạp tính toán: Giải thuật tìm cây khung có độ phức tạp O(n).

5.3 Bài toán tìm cây khung nhỏ nhất

1. Cây khung nhỏ nhất

Cho đồ thị vô hướng có trọng số G = (V, E).

Gọi T là một câu khung của G. Trọng số WT của T là tổng trọng số các cạnh thuộc cây.

Cây khung T là cây khung *nhỏ nhất* $\Leftrightarrow$ WT có giá trị nhỏ nhất.

2. Điều kiện:

G có cây khung nhỏ nhất $\Leftrightarrow$ G liên thông;

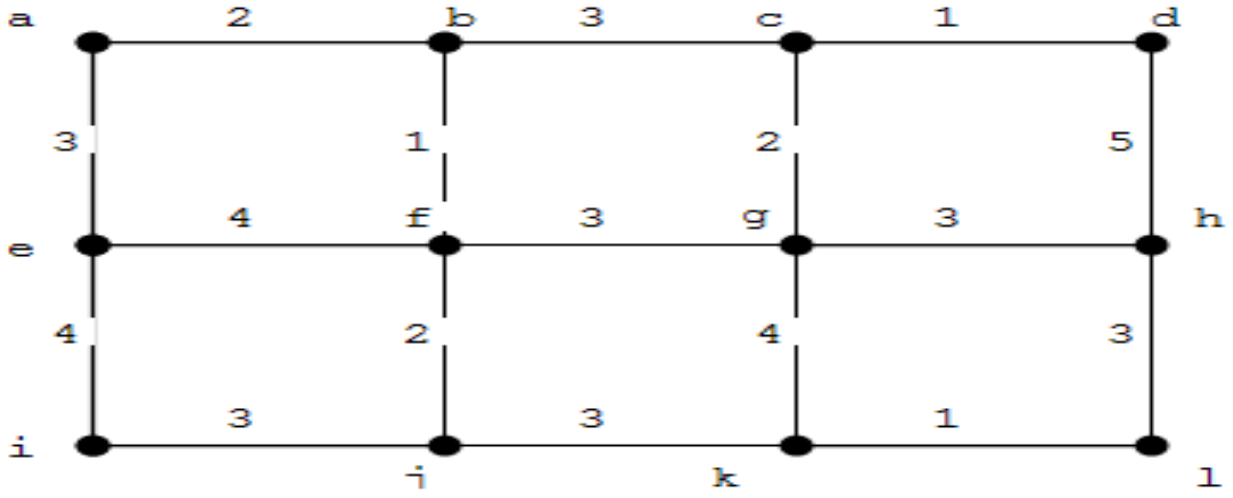
3. Thuật toán tìm cây khung nhỏ nhất:

Thuật toán 1. Thuật toán Prim

Input: Đồ thị G = (V, E) gồm n đỉnh cho bởi ma trận trọng số a[i][j];
Đỉnh s ∈ G;
Output: Cây khung nhỏ nhất T và WT;

Khởi tạo: T= ∅; VT = {s}; WT= 0;
while (V\VT ≠∅) {
<Tìm e = (u, v) có trọng số nhỏ nhất, u ∈ VT, v ∈ V\VT>;
if (Tìm được e) {T = T ∪ e;
WT= WT + trọng số của e;
VT = VT ∪ {v};
}
else return (G không có cây khung); }
return (T và WT);

Ví dụ 1. Dùng thuật toán Prim, hãy tìm cây khung nhỏ nhất của đồ thị G bắt đầu từ s = b;



Giải.

Khởi tạo T = ∅; VT = {b}; WT = 0;

Cây khung nhỏ nhất được xây dựng bằng thuật toán Prim được thể hiện ở bảng sau:

Bước chọn	Cạnh được chọn	Trọng số	WT
1	b,f	1	1
2	a,b	2	3
3	f,j	2	5
4	a,e	3	8
5	f,g	3	11
6	c,g	2	13
7	c,d	1	14
8	g,h	3	17
9	h,l	3	20
10	i,j	3	23
11	k,l	1	24

Kết luận: T = { (b,f), (a,b), (f,j), (a,e), (f,g), (c,g), (c,d), (g,h), (i,j). (h,l), (k,l) }

WT = 24

Cài đặt:

```
int n, a[100][100], s;
int vt[100], d[100], t[100];
void Prim()
{for (int i= 1; i<= n, i++)
        {vt[i]= 0; d[i]= a[s][i]; t[i]= s};
    vt[s]= 1; d[s]= 0; t[s]= 0;
    int wt= 0;
int m = 1;
while (m < n) { int v = 0;
int min = 30000;
for (i = 1; i<= n; i++)
  if (vt[i]==0 && d[i] < min) {min= d[i]; v= i;}
```

```
if (v==0)
      {cout << "Khong co cay khung"; return;}
   vt[v]= 1; wt= wt + a[v][t[v]];
for (i= 1; i<= n; i++)
  if (vt[i]==0 && d[i]) > a[v][i])
   {d[i]= a[v][i]; t[i]= v; }
}
  cout << wt << endl;
for (i= 1; i<= n; i++)
  if (t[i] !=0) cout << i << " " << t[i] << endl;
return;
}
```

Chứng minh tính đúng đắn của thuật toán Prim:

Xét G là đồ thị liên thông có trọng số.

Giả sử các cạnh lần lượt được chọn theo thuật toán Prim là $e_1, e_2,..., e_{n-1}$.

Gọi S là cây với các cạnh $e_1, e_2,..., e_{n-1;}$

S_k là cây với các cạnh $e_1, e_2,..., e_k$;

T là cây khung nhỏ nhất của G có chứa các cạnh $e_1, e_2,..., e_k$.

Giả sử k là số nguyên lớn nhất thỏa mãn điều kiện trên.

Định lý được chứng minh nếu ta chỉ ra rằng S = T.

Giả sử $S \neq T$, với $k < n-1 \Rightarrow$T chứa $e_1, e_2,..., e_k$ nhưng không chứa e_{k+1}.

Xét đồ thị $T' = T \cup e_{k+1}$. T' liên thông và có n cạnh $\Rightarrow$ T' có một chu trình đơn.

$\Rightarrow$ Chu trình chứa e_{k+1} vì trong T không có chu trình.

$\Rightarrow$ Chu trình phải có một cạnh không thuộc S_{k+1} vì S_{k+1} là cây.

Xuất phát từ điểm đầu mút của cạnh e_{k+1} và cũng là điểm đầu mút của một trong các cạnh e_1, e_2,..., e_k , đi dọc theo chu trình cho tới khi gặp một cạnh không thuộc S_{k+1}.

$\Rightarrow$ Tìm được cạnh e không thuộc S_{k+1} và có một đầu mút liên thuộc với một trong các cạnh e_1, e_2,..., e_k .

Xóa e khỏi T và thêm vào e_{k+1} nhận được cây T'' có n-1 cạnh.

T'' chứa các cạnh e_1, e_2,..., e_k , e_{k+1}.

e_{k+1} được chọn bằng thuật toán Prim ở bước k+1 và e cũng có thể được chọn tại bước này

$\Rightarrow$ trọng số của $e_{k+1} \leq$ trọng số của e.

$\Rightarrow$ T'' cũng là cây khung nhỏ nhất.

Điều này mâu thuẫn với cách chọn k là số nguyên lớn nhất sao cho cây khung nhỏ nhất chứa e_1, e_2,..., e_k .

Vì thế k = n-1 và S = T, tức là thuật toán Prim tạo ra cây khung nhỏ nhất.

Thuật toán 2. Thuật toán Kruskal.

Input: Đồ thị G với n đỉnh và m cạnh cho bởi danh sách cạnh;
Output: Cây khung nhỏ nhất T và WT;

Khởi tạo: Sắp xếp các cạnh theo thứ tự giảm của trọng số;

T= ∅; WT= 0; k = 0;
for (i=1; i<= m; i++){
if (T ∪ {e_i}không chứa chu trình) {
T = T ∪ {e_i}; WT= WT + trọng số của e_i; k++;
if (k = n-1) return (T và WT) ;
}
}
Return (G không có cây khung);

Ví dụ 2. Dùng thuật toán Kruskal, hãy tìm cây khung nhỏ nhất của đồ thị trong ví dụ 1.

Giải.

Khởi tạo:

Sắp xếp các cạnh của G theo thứ tự giảm;

T = ∅; VT = {b}; WT = 0;

Cây khung nhỏ nhất và cách chọn các cạnh trong mỗi bước của thuật toán Kruskal được thể hiện trong bảng sau:

Bước chọn	Cạnh được chọn	Trọng số	WT
1	b, f	1	1
2	c, d	1	2
3	k,l	1	3
4	a,b	2	5
5	c,g	2	7
6	f,j	2	9
7	a,e	3	12
8	b,c	3	15
9	g,h	3	18
10	i,j	3	21
11	j,k	3	24

Kết luận: T = { (b,f), (c,d), (k,l), (a,b), (c,g), (f,j), (a,e), (b,c), (g,h), (i,j), (j,k) }

WT = 24

Nhận xét:

Sự khác nhau giữa thuật toán Prim và thuật toán Kruskal:

- Trong thuật toán Prim lựa chọn cạnh theo tiêu chí:

  (a) các cạnh có trọng số tối thiểu,

  (b) liên thuộc với các đỉnh đã thuộc cây và đỉnh không thuộc cây.

- Trong thuật toán Kruskal cần xếp các cạnh theo thứ tự giảm để việc chọn một cạnh được xác định.