

CHƯƠNG 3 CÂY VÀ CÂY KHUNG

Cây là khái niệm quan trọng trong lý thuyết đồ thị, xuất hiện từ năm 1857, khi nhà toán học Anh là Arthur Cayley dùng cây để xác định những dạng khác nhau của hợp chất hóa học. Cây được ứng dụng rất nhiều trong cấu trúc dữ liệu và giải thuật như giải thuật tìm kiếm, giải thuật sắp xếp và nhiều bài toán khác như cây quyết định, nén dữ liệu...

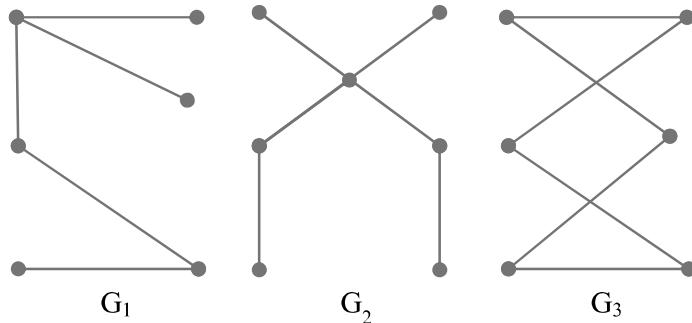
3.1. ĐỊNH NGHĨA CÂY

3.1.1. Cây

Cây là một đồ thị vô hướng, liên thông và không có chu trình.

Do cây không có chu trình, nên cây không thể có cạnh bội và khuyên. Vì vậy mọi cây là đơn đồ thị.

Ví dụ:



Hình 3.1. Minh họa cây

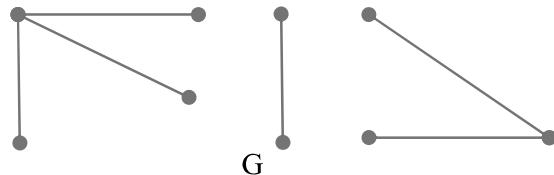
Trong các đồ thị hình 3.1: G_1 và G_2 là cây, còn G_3 có chu trình nên không là cây.

3.1.2. Rừng

Rừng là đồ thị vô hướng không có chu trình.

Hay nói cách khác, rừng là một đồ thị có nhiều thành phần liên thông mà mỗi thành phần liên thông của nó là một cây.

Ví dụ:



Hình 3.2. Minh họa rừng

Trong đồ thị hình 3.2: G là một rừng và có 3 thành phần liên thông.

3.1.3. Định lý 1 (*sự tồn tại của đỉnh treo*)

Một cây T gồm n đỉnh với $n \geq 2$ chứa ít nhất 2 đỉnh treo.

Chứng minh:

Lấy một cạnh (a, b) tùy ý của cây T. Trong tập hợp các đường đi sơ cấp chứa cạnh (a, b) , ta lấy đường đi từ u đến v dài nhất. Vì T là một cây nên $u \neq v$.

Mặt khác, u và v phải là hai đỉnh treo. Vì nếu một đỉnh, chẳng hạn đỉnh u , không phải là đỉnh treo thì u phải là đầu mút của một cạnh (u, x) , với x là đỉnh không thuộc đường đi từ u đến v . Do đó, đường đi sơ cấp từ x đến v chứa cạnh (a, b) , dài hơn đường đi từ u đến v , điều này trái với tính chất đường đi từ u đến v đã chọn.

3.1.4. Định lý 2 (*điều kiện cần và đủ của cây*)

Giả sử $T = (V, E)$ 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 đều là cầu.
- 5) Hai đỉnh bất kỳ của T được nối với nhau bởi đúng một đường đi sơ cấp.
- 6) T không chứa chu trình nhưng khi thêm vào một cạnh ta thu được đúng một chu trình.

Chứng minh: Ta sẽ chứng minh định lý theo sơ đồ sau:

$$1) \Rightarrow 2) \Rightarrow 3) \Rightarrow 4) \Rightarrow 5) \Rightarrow 6) \Rightarrow 1)$$

Mệnh đề 1 \Rightarrow mệnh đề 2: Nếu T là cây n đỉnh thì T không có chu trình và có $n-1$ cạnh.

Chứng minh bằng phương pháp quy nạp:

- Với $n = 1$ thì đồ thị có $n - 1 = 1 - 1 = 0$ cạnh (Đúng)
- Giả sử khẳng định đúng với mọi cây có $k \geq 1$ đỉnh. Ta sẽ chỉ ra \forall cây T có $k+1 \geq 1$ đỉnh sẽ có số cạnh là k . Chọn đường đi dài nhất trong T là $P = (v_1, v_2, \dots, v_m)$. Rõ ràng v_1 là đỉnh treo:
 - v_1 không thể kè với các đỉnh v_3, \dots, v_m vì T không có chu trình.
 - v_1 không thể được nối với các đỉnh khác vì P là dài nhất

Xét $T' = T \setminus \{v_1, (v_1, v_2)\}$ (không thể bỏ các đỉnh trung gian), ta được T' có k đỉnh. Theo giả thiết quy nạp T' có $k-1$ cạnh. Do đó T có k cạnh. (Điều phải chứng minh.)

Mệnh đề 2 \Rightarrow mệnh đề 3: Nếu T không chứa chu trình và có $n-1$ cạnh thì T liên thông.

Chứng minh bằng phương pháp phản chứng:

Giả sử T không liên thông, khi đó T được phân rã thành $k > 1$ thành phần liên thông T_1, T_2, \dots, T_k . Theo giả thiết, vì T không chứa chu trình nên các cây cũng vậy, suy ra T_i là cây.

Gọi $v(T)$ và $e(T)$ tương ứng là số đỉnh và cạnh của T . Theo phần trước, mệnh đề 1 \Rightarrow mệnh đề 2, ta có $e(T_i) = v(T_i) - 1$. Suy ra:

$$\begin{aligned} \sum e(T_i) &= \sum (v(T_i) - 1) = \sum v(T_i) - k \\ \Leftrightarrow e(T) &= v(T) - k \\ \Leftrightarrow n - 1 &= n - k \quad \text{Vô lý với } k > 1. \quad (\text{Điều phải chứng minh.}) \end{aligned}$$

Mệnh đề 3 \Rightarrow mệnh đề 4: Nếu T liên thông và có $n-1$ cạnh thì mỗi cạnh của T là cầu.

Suy luận tương tự như chứng minh mệnh đề 1 \Rightarrow mệnh đề 2.

Chọn đường đi dài nhất $P = (v_1, v_2, \dots, v_m)$.

Nếu từ đồ thị T ta bỏ đi một cạnh nào đó trên đường đi P , thì rõ ràng không còn đường nào khác để đi từ v_1 đến v_m , vì nếu có thì T có chu trình. Vì vậy các cạnh của T đều là cầu.

Mệnh đề 4 \Rightarrow mệnh đề 5: Nếu T liên thông và mỗi cạnh của T là cầu thì hai đỉnh bất kỳ của T được nối với nhau đúng bởi 1 đường đơn.

T liên thông nên mọi 2 đỉnh của T tồn tại đường nối giữa chúng. Đường nối này là duy nhất vì trái lại T sẽ có chu trình và các cạnh trên chu trình đó sẽ không thể là cầu. (Điều phải chứng minh.)

Mệnh đề 5 \Rightarrow mệnh đề 6: Nếu hai đỉnh bất kỳ của T được nối với nhau đúng bởi 1 đường đơn thì T không chứa chu trình nhưng nếu thêm vào nó một cạnh ta thu được đúng 1 chu trình.

T không chứa chu trình vì nếu T có chu trình thì sẽ có cặp đỉnh được nối với nhau bởi 2 đường đơn. Thêm vào cạnh (u, v) ta sẽ nhận được chu trình gồm đường đơn nối u với v và cạnh (u, v) mới. Do đường đi đơn này là duy nhất nên chu trình nhận được cũng là duy nhất. (Điều phải chứng minh.)

Mệnh đề 6 \Rightarrow mệnh đề 1: T không chứa chu trình nhưng nếu thêm vào nó một cạnh ta thu được đúng 1 chu trình thì T là cây (liên thông và không có chu trình).

Chứng minh bằng phản chứng:

Giả sử T không liên thông, khi đó T gồm ít nhất 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 này mâu thuẫn với giả thiết.

Định lý được chứng minh.

3.2. CÂY KHUNG (CÂY BAO TRÙM, CÂY PHỦ, CÂY TỐI ĐẠI)

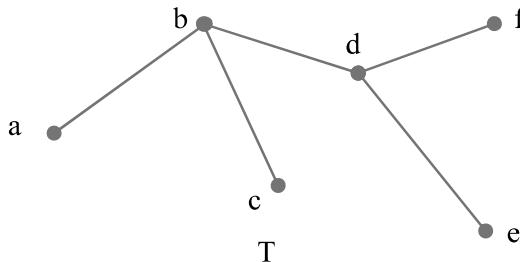
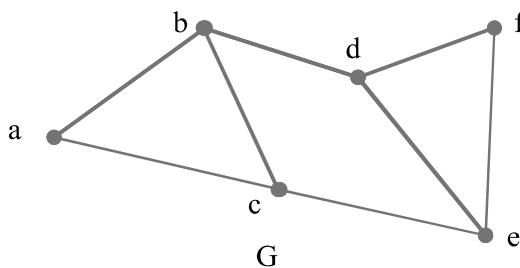
Trong đồ thị liên thông G , nếu ta loại bỏ cạnh nằm trên chu trình nào đó thì ta sẽ được đồ thị vẫn là liên thông. Nếu cứ loại bỏ các cạnh ở các chu trình khác cho đến khi nào đồ thị không còn chu trình, mà vẫn liên thông, thì ta thu được một cây nối các đỉnh của G . Cây này gọi là cây khung hay cây bao trùm của đồ thị G .

3.2.1. Định nghĩa

Cho $G = (V, E)$ là một đồ thị liên thông và $T = (V, E_T)$ là một đồ thị bộ phận của G . Nếu T là cây thì T được gọi là cây khung của G .

Cây khung còn có các tên gọi khác là cây bao trùm, cây phủ, cây tối đại.

Ví dụ:



Hình 3.3. Minh họa cây khung

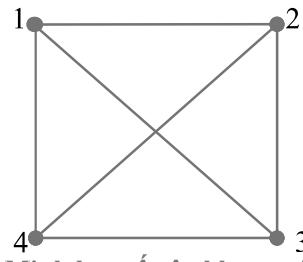
Trong hình 3.3: đồ thị T là cây khung của đồ thị G .

3.2.2. Định lý

Mọi đồ thị liên thông đều có chứa ít nhất một cây khung.

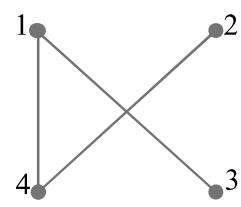
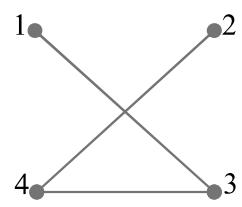
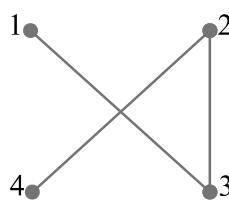
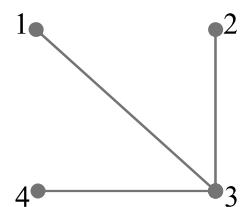
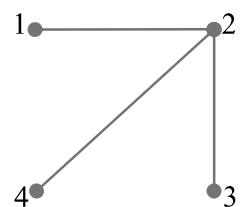
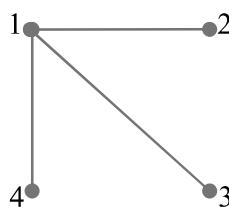
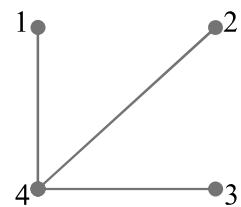
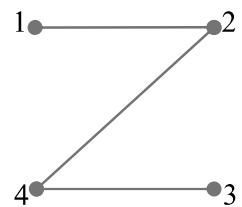
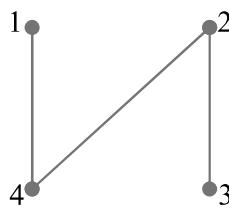
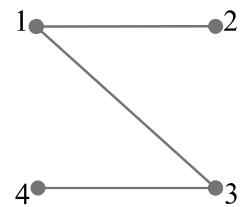
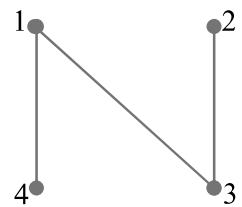
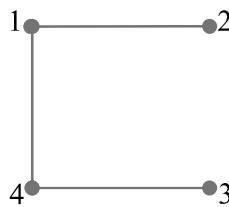
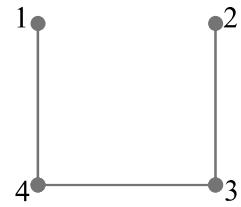
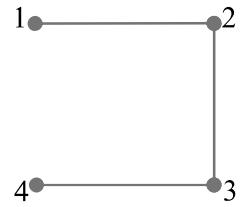
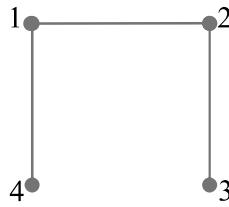
Định lý Cayley: Số cây khung của đồ thị K_n là n^{n-2} .

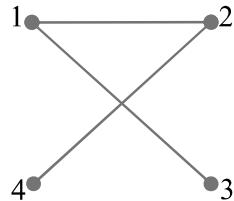
Ví dụ: Xét đồ thị K_4



Hình 3.4. Minh họa số cây khung của đồ thị K_n

Đồ thị K_4 có $4^2 = 16$ cây khung, là các cây bên dưới:





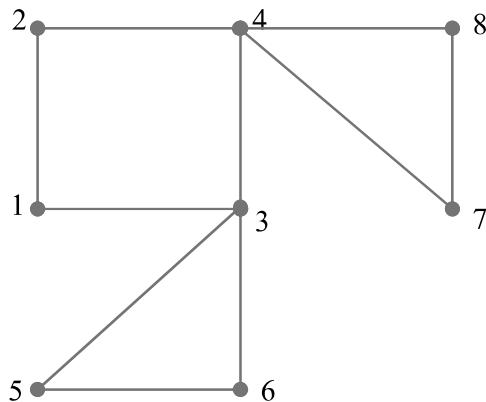
Hình 3.5. Các cây khung của đồ thị K_4

3.2.3. Thuật toán tìm cây khung

Cây khung của đồ thị có thể được xác định bằng các thuật toán duyệt theo chiều sâu hoặc duyệt theo chiều rộng. Trong phần này chúng ta sẽ xét hai ví dụ tìm cây khung của đồ thị theo chiều sâu, theo chiều rộng và minh họa cài đặt thuật toán tương ứng. Các phần cài đặt sử dụng tham số đầu vào là đồ thị G được lưu dưới dạng danh sách kè là mảng $Ke[]$, và một mảng để đánh dấu các đỉnh đã được xét hay chưa là mảng $ChuaXet[]$.

Tìm cây khung theo chiều sâu

Ví dụ: Tìm cây khung của đồ thị sau



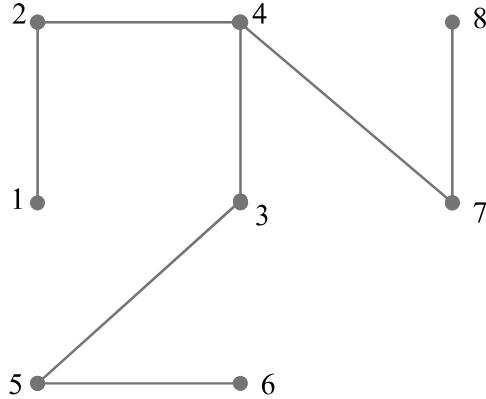
Hình 3.6. Minh họa tìm cây khung theo chiều sâu

Đỉnh v	$K\hat{e}(v)$
1	2, 3
2	4, 1
3	1, 4, 5, 6
4	2, 3, 7, 8
5	3, 6
6	3, 5

7	4, 8
8	4, 7

Thứ tự duyệt các đỉnh theo chiều sâu $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$

Cây khung của đồ thị gồm các cạnh $\{(1, 2), (2, 4), (4, 3), (3, 5), (5, 6), (4, 7), (7, 8)\}$



Hình 3.7. Cây khung tìm được theo chiều sâu

❸ Thuật toán tìm cây khung theo chiều sâu:

// Khai báo các biến toàn cục ChuaXet, Ke, T

void Tree_DFS(vertex v)

{

ChuaXet[v] = 0;

for ($u \in Ke(v)$)

if (ChuaXet[u])

{

$T = T \cup (v, u);$

Tree_DFS(u);

}

}

main(){

// Nhập đồ thị, tạo biến Ke

for ($v \in V$)

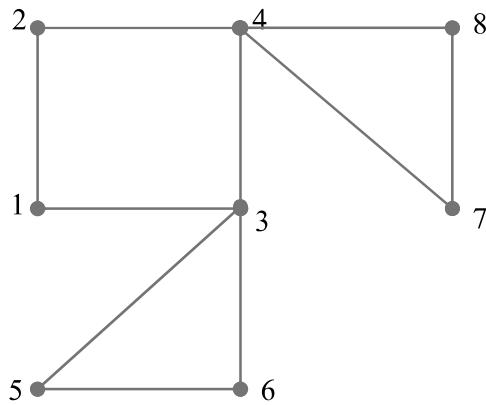
```

ChuaXet[v] = 1;      // khởi tạo cờ cho đỉnh
T = Ø;                // T là tập cạnh cây khung
Tree_DFS(root);       // root là đỉnh nào đó của đồ thị
}

```

Tìm cây khung theo chiều rộng

Ví dụ: Tìm cây khung của đồ thị sau

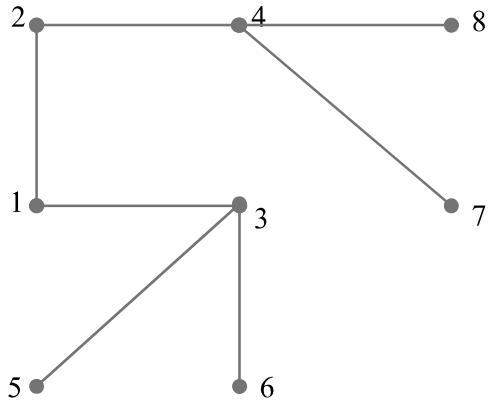


Hình 3.8. Minh họa tìm cây khung theo chiều rộng

Đỉnh v	Kè(v)
1	2, 3
2	4, 1
3	1, 4, 5, 6
4	2, 3, 7, 8
5	3, 6
6	3, 5
7	4, 8
8	4, 7

Thứ tự duyệt các đỉnh theo chiều rộng $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$

Cây khung của đồ thị gồm các cạnh $\{(1, 2), (1, 3), (2, 4), (3, 5), (3, 6), (4, 7), (4, 8)\}$



Hình 3.9. Cây khung tìm được theo chiều rộng

➊ Thuật toán tìm cây khung theo chiều rộng:

// Khai báo các biến toàn cục ChuaXet, Ke, QUEUE

```
void Tree_BFS(vertex v)
```

```
{
```

```
QUEUEUE =  $\emptyset$ ;
```

```
QUEUEUE  $\leftarrow$  v;
```

```
ChuaXet[v] = 0;
```

```
while (QUEUEUE !=  $\emptyset$ )
```

```
{
```

```
v  $\leftarrow$  QUEUEUE;
```

```
for ( $u \in Ke(v)$ )
```

```
if ( ChuaXet[u] )
```

```
{
```

```
QUEUEUE  $\leftarrow$  u;
```

```
ChuaXet[u] = 0;
```

```
T = T  $\cup$  (v,u);
```

```
}
```

```
}
```

```

main()
{
    // Nhập đồ thị, tạo biến Ke

    for ( $v \in V$ )
        ChuaXet[v] = 1;      // khởi tạo cờ cho đỉnh

     $T = \emptyset$ ;          // T là tập cạnh cây khung

    Tree_DFS(root);        // root là đỉnh nào đó của đồ thị

}

```

3.2.4. Bài toán tìm cây khung nhỏ nhất

Bài toán tìm cây khung nhỏ nhất của đồ thị là một trong số những bài toán tối ưu trên đồ thị được ứng dụng trong nhiều lĩnh vực khác nhau của đời sống. Bài toán được phát biểu như sau.

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

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

Bài toán đặt ra là trong số tất cả các cây khung của đồ thị G , hãy tìm cây khung có độ dài nhỏ nhất. Cây khung như vậy được gọi là cây khung nhỏ nhất của đồ thị và bài toán đặt ra được gọi là bài toán tìm cây khung nhỏ nhất.

Bài toán tìm cây khung nhỏ nhất đã có những thuật toán rất hiệu quả để giải chúng. Ta sẽ xét hai trong số những thuật toán như vậy trong phần tiếp theo: thuật toán Prim và thuật toán Kruskal.

3.3. THUẬT TOÁN PRIM

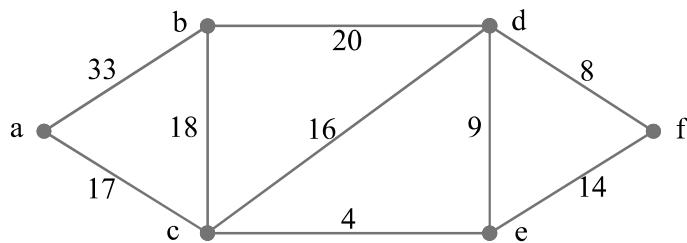
Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất. Trong phương pháp này bắt đầu từ một đỉnh tùy ý x của đồ thị, đầu tiên ta nối x với đỉnh lân cận gần nó nhất, chẳng hạn là đỉnh y . Nghĩa là trong số các cạnh kề của đỉnh x , cạnh (x, y) có độ dài nhỏ nhất. Tiếp theo trong số các cạnh kề với hai đỉnh x hoặc y ta tìm cạnh có độ dài nhỏ nhất, cạnh này dẫn đến đỉnh thứ ba z , và ta thu được cây bộ phận gồm 3 đỉnh và 2 cạnh. Quá trình này sẽ tiếp tục cho đến khi ta thu được cây gồm n đỉnh và $n-1$ cạnh sẽ chính là cây khung nhỏ nhất cần tìm.

Ví dụ: Tìm cây khung nhỏ nhất của đồ thị có ma trận trọng số sau theo thuật toán Prim:

$$C = \begin{matrix} & \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \\ f \end{matrix} & \left(\begin{matrix} 0 & 33 & 17 & \infty & \infty & \infty \\ 33 & 0 & 18 & 20 & \infty & \infty \\ 17 & 18 & 0 & 16 & 4 & \infty \\ \infty & 20 & 16 & 0 & 9 & 8 \\ \infty & \infty & 4 & 9 & 0 & 14 \\ \infty & \infty & \infty & 8 & 14 & 0 \end{matrix} \right) \end{matrix}$$

Giải:

Đồ thị tương ứng với ma trận trọng số đã cho có thể được vẽ lại như hình 3.10:



Hình 3.10. Minh họa thuật toán Prim

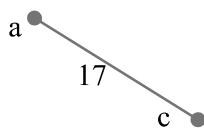
Để tìm cây khung nhỏ nhất theo thuật toán Prim, ta chọn một đỉnh bất kỳ để bắt đầu. Giả sử đỉnh được chọn là a, ta có:

Tập đỉnh $V = \{ a \}$, tập cạnh $U = \emptyset$.

Thực hiện lặp đến khi đủ n đỉnh:

- Lần lặp đầu tiên, chọn đỉnh c (c là đỉnh lân cận gần với a nhất):

$$V = \{ a, c \}, U = \{ (c, a) \}$$



Hình 3.11. Minh họa thuật toán Prim (lần lặp 1)

- Lần lặp thứ hai, chọn đỉnh e:

$$V = \{ a, c, e \},$$

$$U = \{ (c, a), (e, c) \}$$

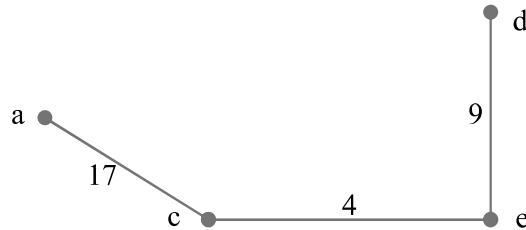


Hình 3.12. Minh họa thuật toán Prim (lần lặp 2)

- Lần lặp thứ ba, chọn đỉnh d:

$$V = \{ a, c, e, d \},$$

$$U = \{ (c, a), (e, c), (d, e) \}$$

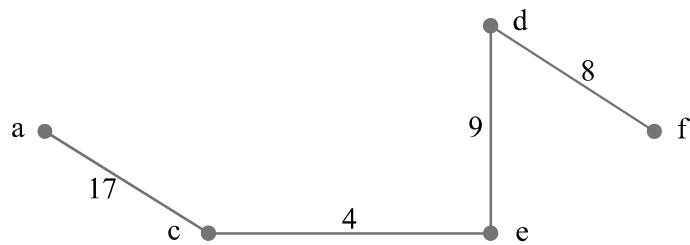


Hình 3.13. Minh họa thuật toán Prim (lần lặp 3)

- Lần lặp thứ tư, chọn đỉnh f:

$$V = \{ a, c, e, d, f \},$$

$$U = \{ (c, a), (e, c), (d, e), (f, d) \}$$



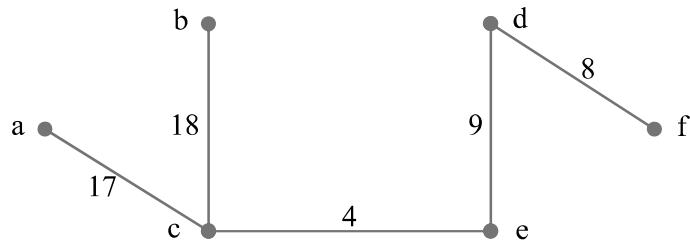
Hình 3.14. Minh họa thuật toán Prim (lần lặp 4)

- Lần lặp thứ năm, chọn đỉnh b:

$$V = \{ a, c, e, d, f, b \},$$

$$U = \{ (c, a), (e, c), (d, e), (f, d), (b, c) \}$$

Ở lần lặp này, ta thu được cây gồm n đỉnh (và $n - 1$ cạnh) nên dừng.



Hình 3.15. Cây khung tìm được theo thuật toán Prim

Tổng trọng số của cây khung là $w(T) = 56$.

❶ Các bước thực hiện thuật toán Prim được mô tả như sau:

Cho đồ thị liên thông $G = (X, E)$ gồm n đỉnh, tìm cây khung nhỏ nhất $T = (V, U)$ của G .

Bước 1. Chọn tùy ý đỉnh $v \in X$ và khởi tạo $V = \{v\}$; $U = \emptyset$.

Bước 2. Chọn đỉnh w sao cho cạnh (w, v) có trọng lượng nhỏ nhất trong các cạnh mà $w \in X \setminus V$ và $v \in V$:

$$V = V \cup \{w\}, U = U \cup \{e\}.$$

Bước 3. Nếu U đủ n đỉnh (hoặc $n-1$ cạnh) thì dừng, ngược lại lặp bước 2.

❸ Thuật toán có thể viết như sau:

void Prim()

{

vertex u, v; // các đỉnh u, v

T = Ø;

U = [1]; // tập các đỉnh

while (U != V)

{

// (u, v) là cạnh ngắn nhất

// sao cho $u \in U$ và $v \in V - U$;

U = U ∪ [v];

T = T ∪ [(u, v)];

}

}

3.4. THUẬT TOÁN KRUSKAL

Thuật toán sẽ xây dựng tập cạnh U của cây khung nhỏ nhất $T = (V, U)$ theo từng bước. Trước hết sắp xếp các cạnh của đồ thị G theo thứ tự không giảm của độ dài. Bắt đầu từ tập cạnh U rỗng, ở mỗi bước ta sẽ lần lượt duyệt trong danh sách các cạnh đã sắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bổ sung nó vào tập cạnh U không tạo thành chu trình. Thuật toán kết thúc khi thu được tập cạnh U gồm $n-1$ cạnh.

Ví dụ:

Tìm cây khung nhỏ nhất của đồ thị hình 3.10 theo thuật toán Kruskal.

Giải:

Bước khởi tạo:

- Đặt $U = \emptyset$.

- Sắp xếp các cạnh của đồ thị theo thứ tự không giảm của độ dài ta có dãy:

Cạnh	Độ dài
(c, e)	4
(d, f)	8
(d, e)	9
(e, f)	14
(c, d)	16
(a, c)	17
(b, c)	18
(b, d)	20
(a, b)	23

Thực hiện lặp đến khi đủ $n-1$ cạnh:

- Ở lần lặp đầu tiên ta bổ sung vào tập U cạnh có trọng số nhỏ nhất (c, e):

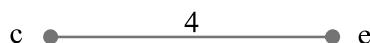
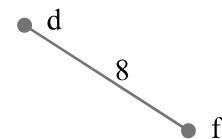
$$U = \{ (c, e) \}$$



Hình 3.16. Minh họa thuật toán Kruskal (lần lặp 1)

- Lần lặp thứ hai ta bổ sung vào tập U cạnh (d, f):

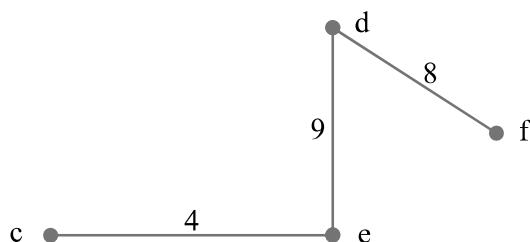
$$U = \{ (c, e), (d, f) \}$$



Hình 3.17. Minh họa thuật toán Kruskal (lần lặp 2)

- Lần lặp thứ ba ta bổ sung vào tập U cạnh (d, e):

$$U = \{ (c, e), (d, f), (d, e) \}$$

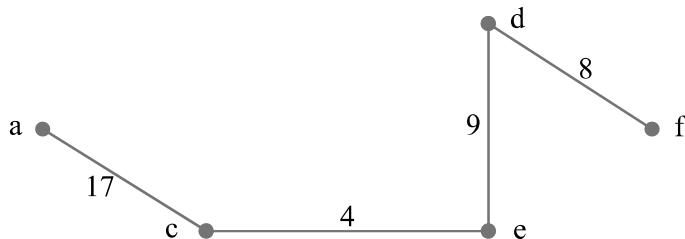


Hình 3.18. Minh họa thuật toán Kruskal (lần lặp 3)

- Lần lặp thứ tư: nếu ta thêm cạnh (e, f) vào U thì nó sẽ tạo thành chu trình với 2 cạnh (d, e), (d, f) đã có trong U. Tình huống tương tự cũng xảy ra đối với cạnh (c, d) là cạnh tiếp theo của dãy.

Vì vậy ở lần lặp này ta bỏ sung cạnh (a, c) vào tập U.

$$U = \{ (c, e), (d, f), (d, e), (a, c) \}$$

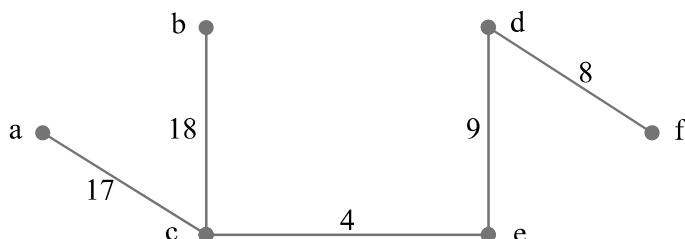


Hình 3.19. Minh họa thuật toán Kruskal (lần lặp 4)

- Lần lặp kế tiếp ta bỏ sung cạnh (b, c) vào tập U. Lúc này ta thu được tập U gồm 5 cạnh:

$$U = \{ (c, e), (d, f), (d, e), (a, c), (b, c) \}$$

chính là tập cạnh của cây khung nhỏ nhất cần tìm.



Hình 3.20. Cây khung tìm được theo thuật toán Kruskal

Tổng trọng số của cây khung tìm được là $w(T) = 56$.

⦿ Các bước thực hiện thuật toán Kruskal được mô tả như sau:

Cho đồ thị liên thông $G = (X, E)$ gồm n đỉnh, tìm cây khung nhỏ nhất $T = (V, U)$ của G .

Bước 1. Sắp xếp các cạnh trong G tăng dần theo trọng số, khởi tạo $U = \emptyset$

Bước 2. Lần lượt lấy từng cạnh e thuộc danh sách đã sắp xếp. Nếu $U + \{e\}$ không chứa chu trình thì thêm e vào U .

Bước 3. Nếu U đủ $n-1$ cạnh thì dừng, ngược lại lặp bước 2.

⦿ Thuật toán có thể viết như sau:

```
void Kruskal()
```

```
{
```

```

 $T = \emptyset;$ 
while ( $|T| < (n-1)$   $\&\&$  ( $E \neq \emptyset$ ))
{
     $E = E \setminus \{e\};$ 
    if ( $T \cup \{e\}$  không chứa chu trình)
    {
         $T = T \cup \{e\};$ 
    }
}
if ( $|T| < n-1$ )
    // Đồ thị không liên thông;
}

```

Nhận xét

Độ phức tạp của thuật toán Prim được đánh giá như sau: nếu $T(k)$ có k đỉnh thì có $n-k$ đỉnh không thuộc $T(k)$, do đó ta phải chọn chiều dài nhỏ nhất của nhiều nhất là $k(n-k)$ cạnh. Do $k(n-k) < (n-1)^2$, nên độ phức tạp của bước chọn e_{k+1} là $O(n^2)$. Vì phải chọn $n-1$ cạnh, nên độ phức tạp của thuật toán Prim là $O(n^3)$.

Độ phức tạp của thuật toán Kruskal được đánh giá như sau: trước tiên, ta sắp xếp các cạnh của G theo thứ tự có chiều dài tăng dần; việc sắp xếp này có độ phức tạp $O(p^2)$, với p là số cạnh của G . Người ta chứng minh được rằng việc chọn e_{i+1} không tạo nên chu trình với i cạnh đã chọn trước đó có độ phức tạp là $O(n^2)$. Do $p \leq n(n-1)/2$, thuật toán Kruskal có độ phức tạp là $O(p^2)$.

Đối với những đồ thị dày (có số cạnh $m \approx n(n-1)/2$) thì thuật toán Prim hiệu quả hơn thuật toán Kruskal.

Một số ứng dụng của bài toán tìm cây khung nhỏ nhất

Bài toán xây dựng hệ thống đường sắt: Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất cứ một thành phố nào đến bất kỳ một trong số các thành phố còn lại. Mặt khác, trên quan điểm kinh tế đòi hỏi là chi phí về xây dựng hệ thống đường phải là nhỏ nhất. Rõ ràng là đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng, với phương án xây dựng tối ưu phải là cây. Vì vậy, bài toán đặt ra dẫn về 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 cạnh chính là chi phí xây dựng hệ thống đường sắt nối hai thành phố.

Bài toán nối mạng máy tính: Giả sử cần nối mạng một hệ thống gồm n máy tính đánh số từ 1 đến n . Biết chi phí nối máy i với máy j là $m(i, j)$, thông thường chi

phi phí này phụ thuộc vào độ dài cáp nối cần sử dụng. Hãy tìm cách nối mạng sao cho tổng chi phí là nhỏ nhất. Bài toán này cũng dẫn về bài toán tìm cây khung nhỏ nhất.

3.5. CÂY CÓ GỐC

3.5.1. Định nghĩa

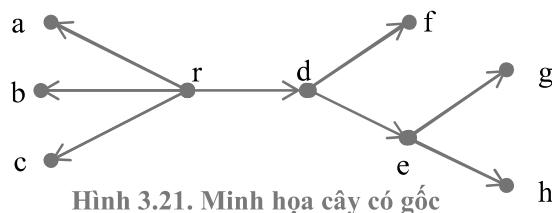
Định nghĩa 1

Cây có hướng là đồ thị có hướng mà đồ thị vô hướng nền của nó là một cây.

Cây có gốc là một cây có hướng, trong đó có một đỉnh đặc biệt gọi là gốc, từ gốc có đường đi đến mọi đỉnh khác của cây.

Trong cây có gốc thì gốc r có bậc vào bằng 0, còn tất cả các đỉnh khác đều có bậc vào bằng 1.

Ví dụ:



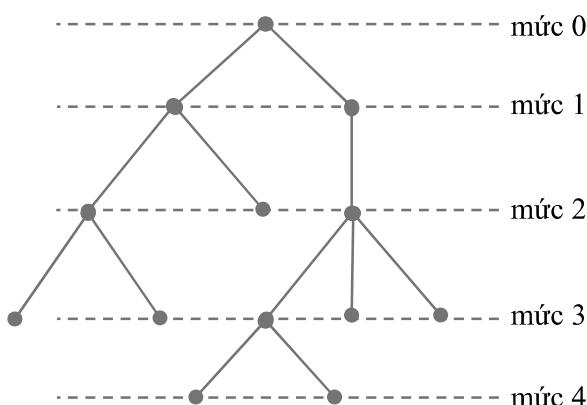
Hình 3.21. Minh họa cây có gốc

Một cây có gốc thường được vẽ với gốc r ở trên cùng và cây phát triển từ trên xuống:

- Gốc r gọi là đỉnh mức 0.
- Các đỉnh kề với r được xếp ở phía dưới gốc và gọi là đỉnh mức 1.
- Đỉnh ngay dưới đỉnh mức 1 là đỉnh mức 2, ...

Tổng quát, trong một cây có gốc thì v là đỉnh mức k khi và chỉ khi đường đi từ r đến v có độ dài bằng k . Độ cao của cây là mức cao nhất của các đỉnh.

Ta thường vẽ cây như hình bên dưới để làm rõ mức của các đỉnh. Ngoài ra, trong cây có gốc, mọi cung đều có hướng từ trên xuống, vì vậy vẽ mũi tên chỉ hướng đi là không cần thiết.



Hình 3.22. Mức của đỉnh

Định nghĩa 2

Cho cây T có gốc r

- Nếu uv là một cung của T thì u được gọi là cha của v, còn v gọi là con của u.
- Đỉnh không có con gọi là lá (hay đỉnh ngoài). Đỉnh không phải là lá gọi là đỉnh trong.
- Hai đỉnh có cùng cha gọi là anh em.
- Nếu có đường đi $v_1v_2\dots v_k$ thì v_1, v_2, \dots, v_{k-1} gọi là tổ tiên của v_k . Còn v_k gọi là hậu duệ của v_1, v_2, \dots, v_{k-1} .
- Cây con tại đỉnh v là cây có gốc là v và tất cả các đỉnh khác là hậu duệ của v trong cây T đã cho.

Định nghĩa 3

Cho T là cây có gốc

- T được gọi là cây k-phân nếu mỗi đỉnh của T có nhiều nhất là k con.
- Cây 2-phân được gọi là cây nhị phân.
- Cây k-phân đủ là cây mà mọi đỉnh trong có đúng k con.
- Cây k-phân với độ cao h được gọi là cân đối nếu các đỉnh ngoài đều ở mức h hoặc h – 1.

3.5.2. Cây nhị phân

Định nghĩa

Cây nhị phân là cây có gốc với mỗi đỉnh có nhiều nhất 2 con.

Trong một cây nhị phân, mỗi con được chỉ rõ là con bên trái hay con bên phải của cha.

Duyệt cây nhị phân

Trong nhiều trường hợp, ta cần phải duyệt một cách có hệ thống mọi đỉnh của một cây nhị phân, mỗi đỉnh chỉ qua một lần. Có nhiều thuật toán duyệt cây nhị phân, các thuật toán đó khác nhau chủ yếu ở thứ tự thăm các đỉnh. Phần này sẽ giới thiệu ba thuật toán để quy là duyệt tiền thứ tự, hậu thứ tự và trung thứ tự.

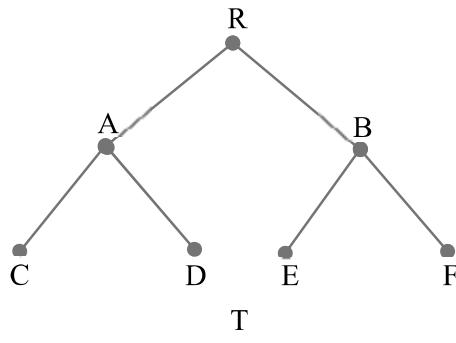
⦿ Phép duyệt tiền thứ tự (preorder):

Bước 1. Duyệt gốc r.

Bước 2. Duyệt cây con bên trái của r theo tiền thứ tự.

Bước 3. Duyệt cây con bên phải của r theo tiền thứ tự.

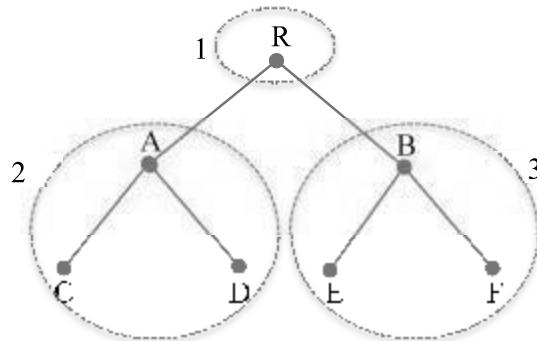
Ví dụ: Duyệt cây T trong hình 3.23 theo phép duyệt tiền thứ tự.



Hình 3.23. Minh họa phép duyệt cây

Giải:

Trước tiên duyệt nút gốc R. Tiếp theo, duyệt cây con bên trái của R, rồi duyệt cây con bên phải R.



Hình 3.24. Thứ tự duyệt cây (tiền thứ tự)

Thứ tự duyệt các đỉnh của T theo phép duyệt tiền thứ tự là:

$$R \rightarrow A \rightarrow C \rightarrow D \rightarrow B \rightarrow E \rightarrow F$$

⦿ Phép duyệt hậu thứ tự (postorder):

Bước 1. Duyệt cây con bên trái của r theo hậu thứ tự.

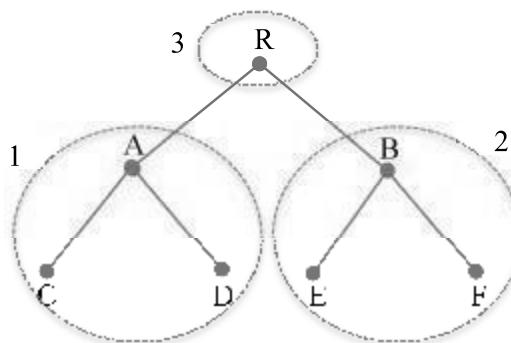
Bước 2. Duyệt cây con bên phải của r theo hậu thứ tự.

Bước 3. Duyệt gốc r.

Ví dụ: Duyệt cây T (Hình 3.23. Minh họa phép duyệt cây) theo phép duyệt hậu thứ tự.

Giải:

Trước tiên, duyệt cây con bên trái của R, rồi duyệt cây con bên phải R, sau đó duyệt đến R.



Hình 3.25. Thứ tự duyệt cây (hậu thứ tự)

Thứ tự duyệt các đỉnh của T theo phép duyệt hậu thứ tự là:

$$C \rightarrow D \rightarrow A \rightarrow E \rightarrow F \rightarrow B \rightarrow R$$

⦿ Phép duyệt trung thứ tự (inorder)

Bước 1. Duyệt cây con bên trái của r theo trung thứ tự.

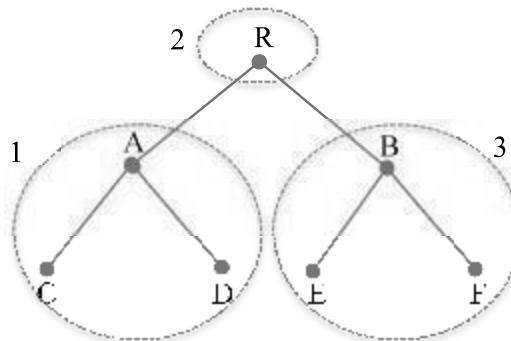
Bước 2. Duyệt gốc r.

Bước 3. Duyệt cây con bên phải của r theo trung thứ tự.

Ví dụ: Duyệt cây T (Hình 3.23. Minh họa phép duyệt cây) theo phép duyệt trung thứ tự.

Giải:

Trước tiên duyệt cây con bên trái của R. Tiếp theo, duyệt nút gốc R. Sau đó duyệt cây con bên phải R.



Hình 3.26. Thứ tự duyệt cây (trung thứ tự)

Thứ tự duyệt các đỉnh của T theo phép duyệt trung thứ tự là:

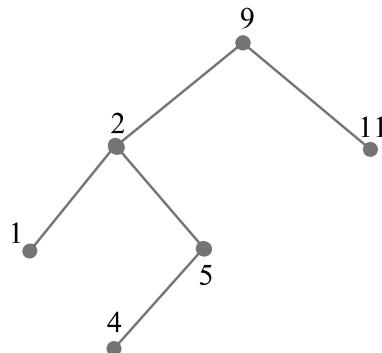
$$C \rightarrow A \rightarrow D \rightarrow R \rightarrow E \rightarrow B \rightarrow F$$

📖 **Cây tìm kiếm nhị phân**

Cây tìm kiếm nhị phân là cây nhị phân mà mỗi nút (đỉnh) đều được gán một khóa (trọng số) sao cho với mỗi nút k:

- Mọi khóa trên cây con trái đều nhỏ hơn khóa của nút k,
- Mọi khóa trên cây con phải đều lớn hơn khóa của nút k.

Ví dụ:



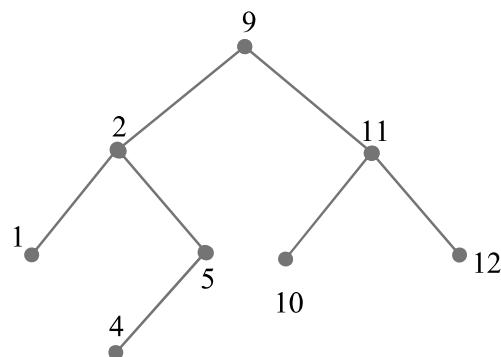
Hình 3.27. Cây tìm kiếm nhị phân

Cây tìm kiếm nhị phân là một cấu trúc rất thuận lợi cho các giải thuật tìm kiếm, sắp xếp.

❑ Cây nhị phân cân bằng

Cây nhị phân cân bằng là cây tìm kiếm nhị phân mà tại mỗi nút, chiều cao của hai cây con sai khác nhau không quá một.

Ví dụ:



Hình 3.28. Cây nhị phân cân bằng

BÀI TẬP CHƯƠNG 3

1. Vẽ cây khung cho các đồ thị sau:

a) K_5

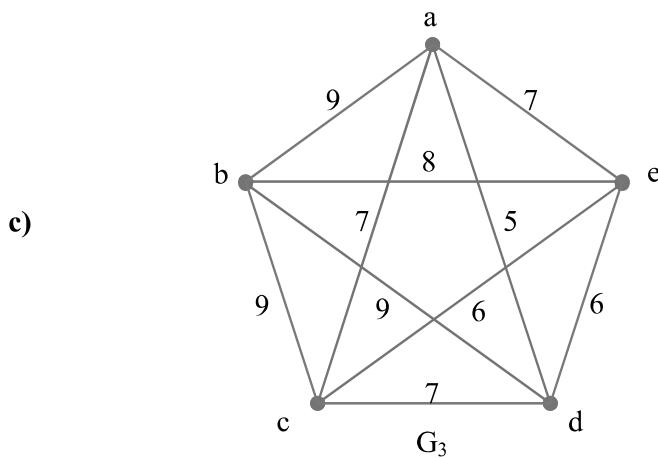
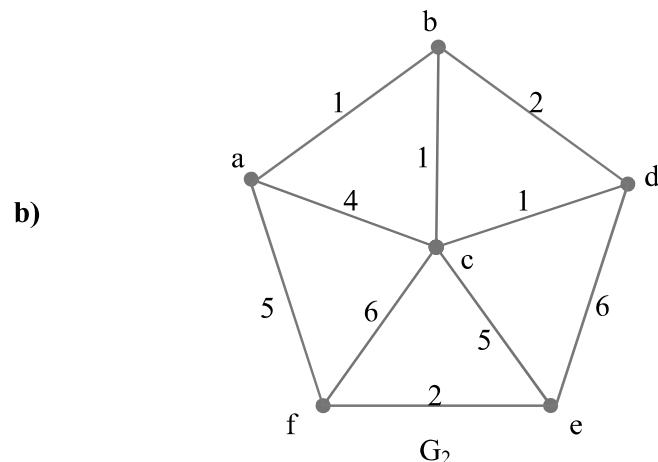
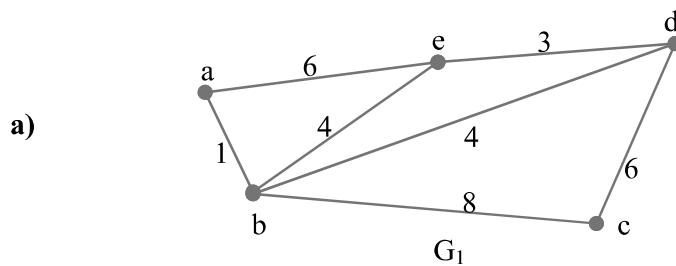
b) $K_{4,4}$

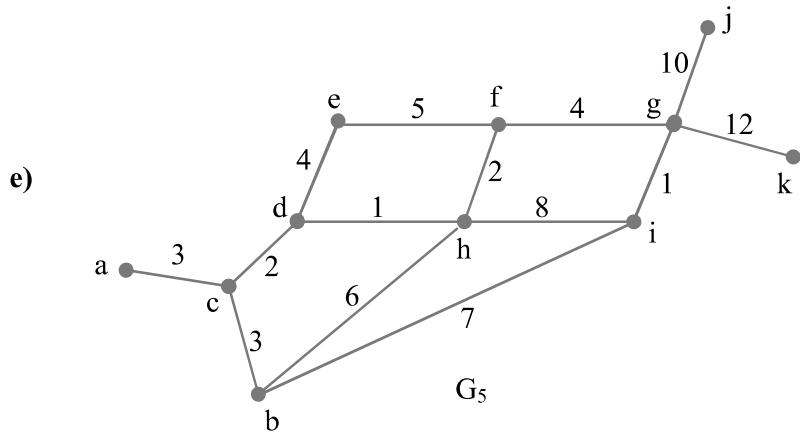
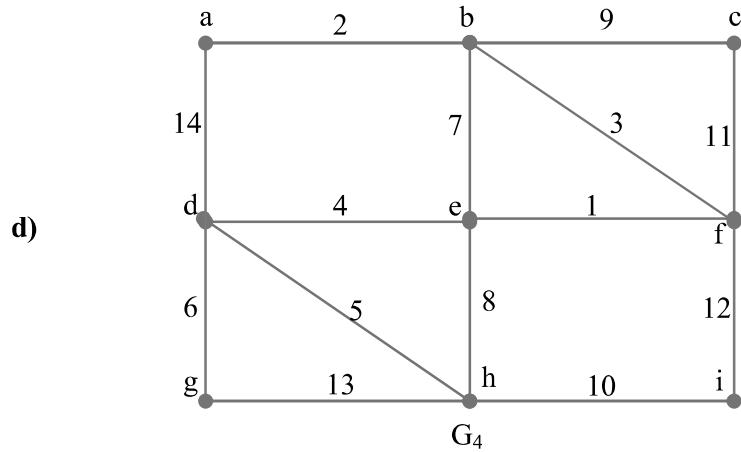
c) Q_3

d) C_5

e) W_5

2. Tìm cây khung của các đồ thị được cho bên dưới bằng phép duyệt DFS và BFS (không kể các trọng số).





3. Tìm cây khung nhỏ nhất của các đồ thị được cho trong bài 2 bằng giải thuật Kruskal và Prim. Cho biết trọng số của cây khung là bao nhiêu?

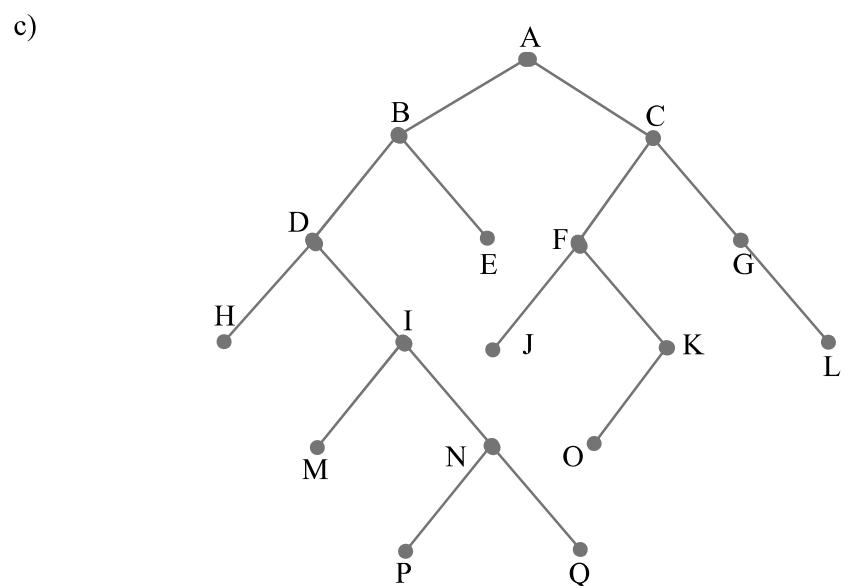
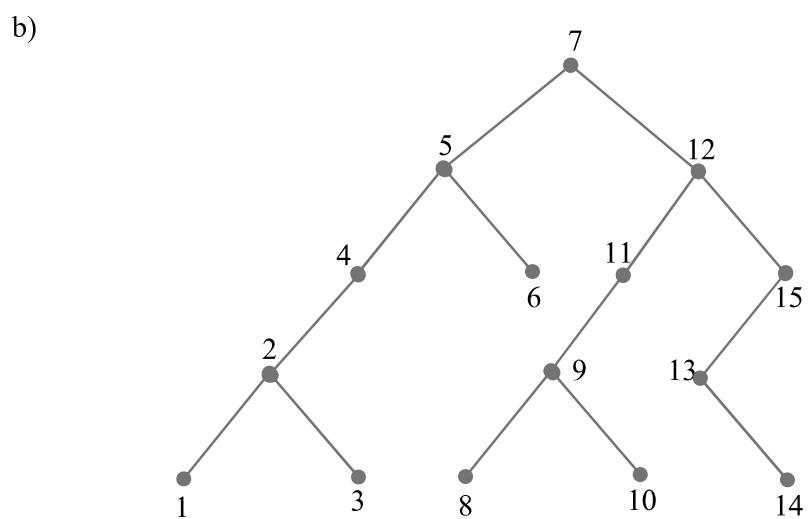
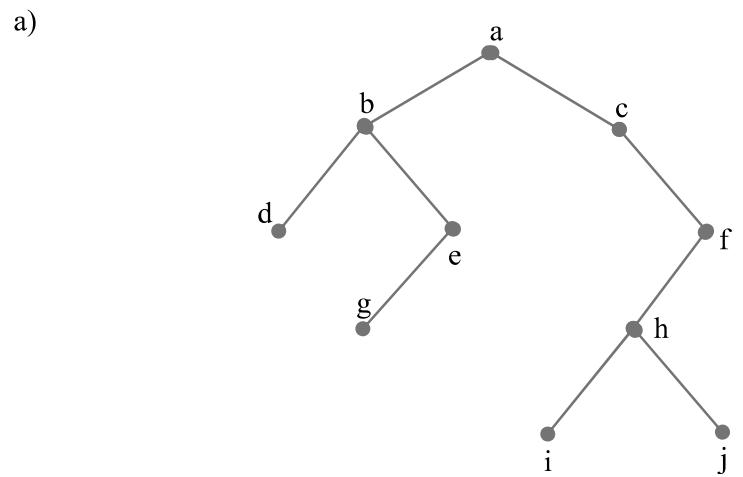
4. Vẽ các cây gồm:

- a) 3 đỉnh.
- b) 4 đỉnh.
- c) 5 đỉnh.

5. Vẽ một cây nhị phân đầy đủ có chiều cao $h = 3$, có 4 đỉnh trong và 5 lá.

6. Duyệt các cây nhị phân sau theo các phép duyệt:

- Tiền thứ tự
- Hậu thứ tự
- Trung thứ tự.



7. Viết chương trình nhập vào danh sách kề của đồ thị, xác định cây khung của đồ thị bằng phép duyệt:

a) DFS.

b) BFS.

8. Viết chương trình nhập vào ma trận trọng số của đồ thị, xác định cây khung nhỏ nhất của đồ thị bằng thuật toán :

a) Prim.

b) Kruskal.