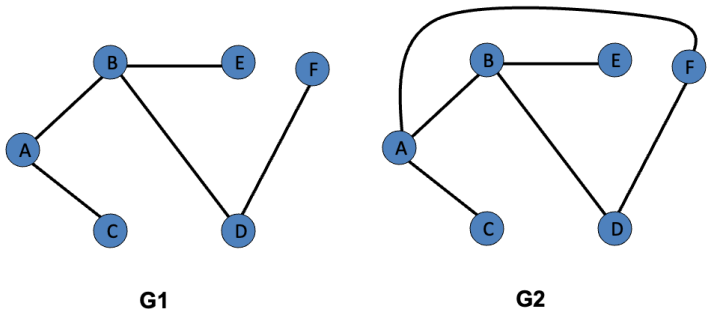
BÀI 5
CÂY KHUNG CỦA ĐỒ THỊ

Mục tiêu

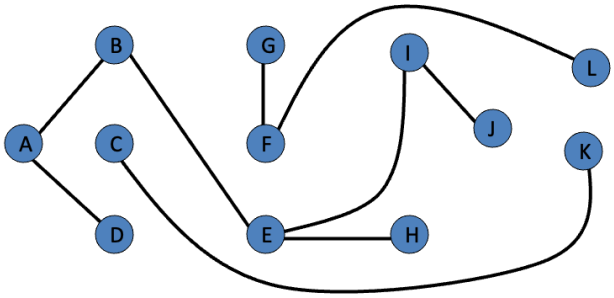
- Hiểu khái niệm cây; cây khung và cây khung nhỏ nhất của đồ thị vô hướng;
- Hiểu ý tưởng và cài đặt được thuật toán **Kruskal** và thuật toán **Prim** để tìm cây khung nhỏ nhất;
- Vận dụng giải các bài toán liên quan.

1. Một số khái niệm



Hình 1. Ví dụ về cây: đồ thị G1 là cây, đồ thị G2 không phải là cây

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



Hình 2. Một rừng bao gồm 3 cây

Rừng (Forest) là một đồ thị mà mỗi thành phần liên thông của nó là một cây.

Định lý: cho đồ thị vô hướng $G = (V, E)$ có n đỉnh. Các mệnh đề sau là tương đương:

(1) G là một cây;

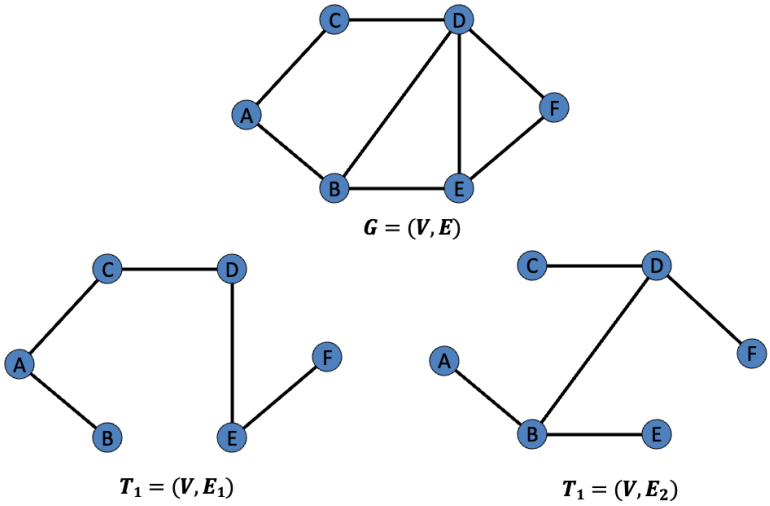
(2) G không có chu trình và có n-1 cạnh;

(3) G liên thông và có n-1 cạnh;

(4) G liên thông và nếu bỏ đi một cạnh bất kỳ thì G mất tính liên thông (mỗi cạnh của G đều là cạnh cầu);

(5) Giữa hai đỉnh bất kỳ của G được nối với nhau bởi đúng một đường đi;

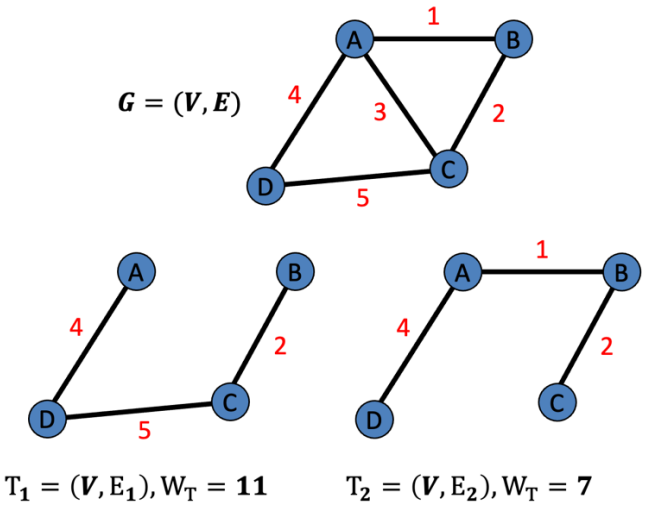
(6) G không có chu trình nhưng khi thêm một cạnh bất kỳ ta thu được đúng một chu trình.



Hình 3. Đồ thị G và các cây khung của đồ thị: T_1 và T_2

Cây khung (Spanning Tree) của một đồ thị vô hướng, liên thông $G = (V, E)$ là cây chứa tất cả các đỉnh của G và mỗi cạnh của cây thuộc đồ thị G. Nói các khác, nếu T là cây khung của đồ thị $G = (V, E)$ thì:

- $T = (V, E')$
- $E' \subseteq E$



Hình 4. Đồ thị G có cây khung nhỏ nhất là T_2

Cây khung nhỏ nhất (Minimum Spanning Tree) của một đồ thị vô hướng, liên thông, có trọng số $G = (V, E)$ là cây khung mà tổng trọng số của các cạnh trong cây là nhỏ nhất.

2. Tìm cây khung của đồ thị bằng thuật toán DFS/BFS

2.1. Tìm cây khung bằng thuật toán DFS

Input: đồ thị G(V,E)

Output: một cây khung T của đồ thị

```
SpanningTree()
    visited[v] ← false (với mọi v ∈ V)
    T ← ∅
    // u là một đỉnh bất kỳ trong đồ thị
    DFS_SpanningTree(u, visited, T)

DFS_SpanningTree(u, visited, T)
    visited[u] ← true
    foreach v in adjList[u]
        if visited[v] = false
            T ← T ∪ (u,v)
            DFS(v)
```

2.2. Tìm cây khung bằng thuật toán BFS

Input: đồ thị G(V,E)

Output: một cây khung T của đồ thị

```
SpanningTree()
    visited[v] ← false (với mọi v ∈ V)
    T ← ∅
    q ← ∅  //queue lưu các đỉnh chờ xét
    // u là một đỉnh bất kỳ trong đồ thị
    q.Enqueue(u)
    visited[u] ← true
    while(q ≠ ∅)
        u ← q.Dequeue()
        foreach v in adjList[u]
        if visited[v] = false
            T ← T ∪ (u,v)
            q.Enqueue(v)
            visited[v] ← true
```

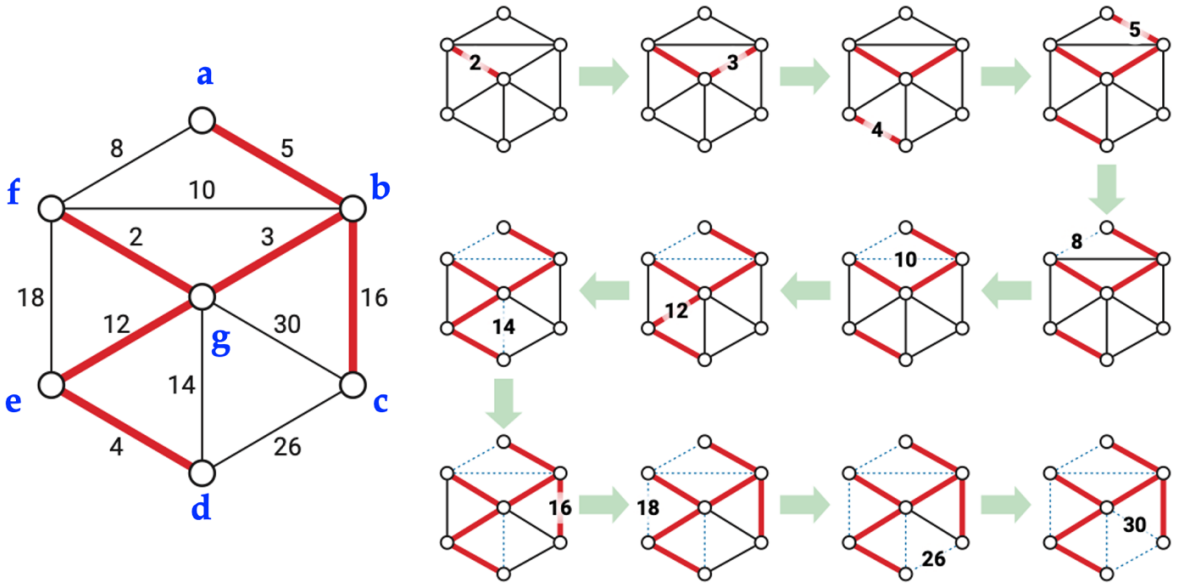
3. Thuật toán Kruskal

Thuật toán **Krushkal** được sử dụng để **tìm cây khung nhỏ nhất của đồ thị vô hướng, có trọng số** bằng cách lần lượt chọn cạnh nhỏ nhất trong toàn bộ đồ thị.

3.1. Ý tưởng của thuật toán Kruskal

- Là một thuật toán **tham lam** (*greedy*);
- Sắp xếp các cạnh theo thứ tự **tăng dần dựa trên trọng số**;
- Lần lượt **chọn đủ (n-1) cạnh** trong danh sách cạnh đã sắp xếp sao cho các cạnh được chọn **không tạo thành chu trình**.

3.2. Minh họa thuật toán Kruskal



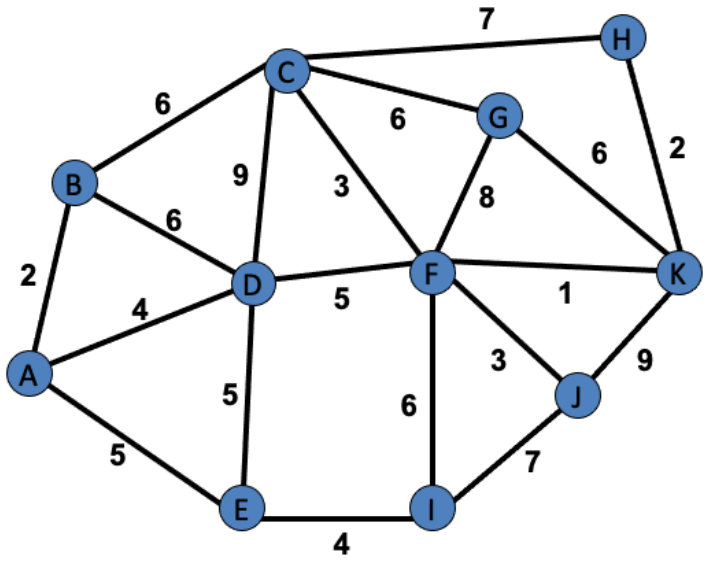
Minh họa quá trình tìm cây khung nhỏ nhất của đồ thị bằng thuật toán **Krushkal**:

Bước 1: Sắp xếp danh sách cạnh tăng dần dựa vào trọng số	**Bước 2:** Lần lượt chọn từ danh sách cạnh đã sắp xếp đủ (n-1) cạnh và không tạo thành chu trình (*6 cạnh*)
(f, g, 2)	Chọn (1)
(b, g, 3)	Chọn (2)
(d, e, 4)	Chọn (3)
(a, b, 5)	Chọn (4)
(a, f, 8)	Không chọn vì tạo thành chu trình với các cạnh (1), (2) và (4)
(b, f, 10)	Không chọn vì tạo thành chu trình với các cạnh (1) và (2)

(e, g, 12)	Chọn (5)
(d, g, 14)	Không chọn vì tạo thành chu trình với các cạnh (3) và (5)
(b, c, 16)	Chọn (6). Đã chọn đủ 6 cạnh nên dừng thuật toán
(e, f, 18)	
(c, d, 26)	
(c, g, 30)	

➔ **Cây khung nhỏ nhất của đồ thị là T = {(f, g, 2), (b, g, 3), (d, e, 4), (a, b, 5), (e, g, 12), (b, c, 16)}, có tổng trọng số = 2 + 3 + 4 + 5 + 12 + 16 = 42.**

Bài tập: Minh họa quá trình tìm cây khung nhỏ nhất của đồ thị bên dưới bằng thuật toán **Krushkal**:



3.3. Cài đặt thuật toán Kruskal

Input: đồ thị G(V,E)

Output: một cây khung *T* là cây khung nhỏ nhất của đồ thị

```
Krushkal()
    Sắp xếp các cạnh E tăng dần dựa trên trọng số
    T ← ∅
    Với mỗi cạnh (u, v) ∈ E
        Nếu (u, v) không tạo chu trình với các cạnh của T(*)
            T ← T ∪ (u,v)
            Nếu đủ (n-1) cạnh thì dừng thuật toán
```

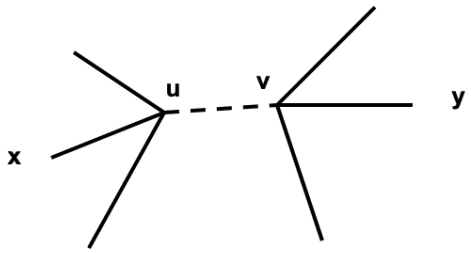
(*) Kỹ thuật kiểm tra cạnh (u,v) có tạo thành chu trình với các cạnh của T hay không?

→ Nếu u và v liên thông với nhau thì thêm cạnh (u,v) sẽ tạo thành chu trình. Để kiểm tra 2 đỉnh u và v có liên thông với nhau hay không có thể sử dụng 1 trong 3 cách sau:

Cách 1: Sử dụng một mảng 2 chiều **int[,] connected**. Trong đó:

- connected[u,v] = 1 nếu u liên thông với v
- connected[u,v] = 0 nếu u KHÔNG liên thông với v

- Ban đầu khởi tạo connected[v,v] = 1 với mọi $v \in V$
- Khi thêm 1 cạnh (u,v) vào cây khung thì cập nhật mảng connected như thế nào?



→ Nếu thêm cạnh (u,v) thì **tất cả các đỉnh x kề với u sẽ liên thông với tất cả các đỉnh y kề với v**! (sử dụng 2 vòng lặp for)

```
for(x : 1 → n)
   if(connected[u,x] == 1) //x liên thông với u
      for(y: 1 → n)
         if(connected[v,y] == 1) //y liên thông với v
            //Cập nhật x liên thông với y
            connected[x,y] = connected[y,x] = 1
```

Cách 2: Sử dụng mảng một chiều để gán nhãn các đỉnh.

- Những đỉnh thuộc cùng một miền liên thông sẽ có nhãn giống nhau;
- Khi thêm một cạnh (u, v):
  - Cập nhật tất cả những đỉnh có nhãn giống v thành nhãn của u (*hợp nhất thành phần liên thông chưa đỉnh v vào thành phần liên thông chứa đỉnh u*);
  - Hoặc ngược lại, cập nhật tất cả những đỉnh có nhãn giống u thành nhãn của v (*hợp nhất thành phần liên thông chưa đỉnh u vào thành phần liên thông chứa đỉnh v*);

Cách 3: Sử dụng **Disjoint Set Union**

(tham khảo https://wiki.vnoi.info/algo/data-structures/disjoint-set-union)

3.4. Độ phức tạp của thuật toán Kruskal: **O(|E|. log|E|)**

4. Thuật toán Prim

Thuật toán **Prim** được sử dụng để **tìm cây khung nhỏ nhất của đồ thị vô hướng, có trọng số** bằng cách lần lượt chọn cạnh nhỏ nhất từ các đỉnh đã có trong cây.

4.1. Ý tưởng của thuật toán Prim

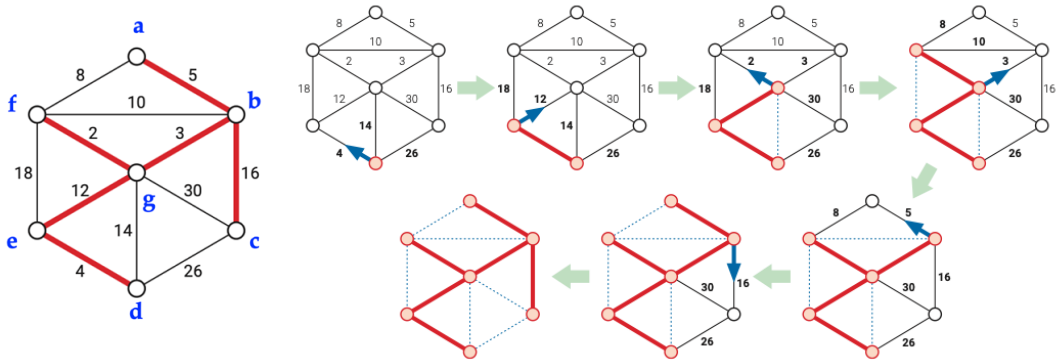
- Là một thuật toán **tham lam** (*greedy*);
- Bắt đầu **từ một đỉnh bất kỳ**;
- Lần lượt **chọn (n-1) đỉnh** còn lại, tương ứng với (n-1) cạnh. Cạnh (u, v) được chọn khi thỏa các điều kiện sau:

  (1) $\boldsymbol{u}$ là đỉnh **thuộc cây** (*đã chọn*);

  (2) $\boldsymbol{v}$ là đỉnh **không thuộc cây** (*chưa chọn*);

  (3) **Độ dài cạnh** $(\boldsymbol{u}, \boldsymbol{v})$ **là nhỏ nhất** so với các cạnh thỏa điều kiện (1) và (2).

4.2. Minh họa thuật toán Prim

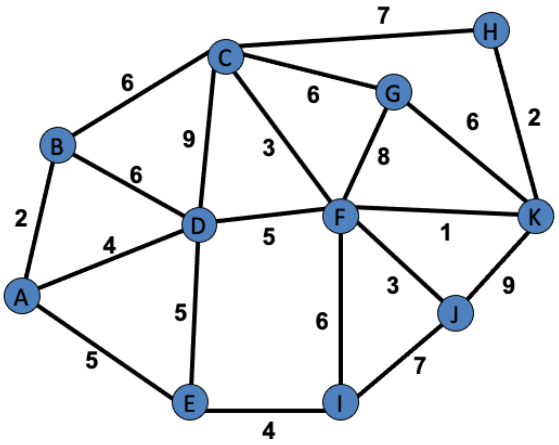


Minh họa quá trình tìm cây khung nhỏ nhất của đồ thị bằng thuật toán **Prim**:

Lần lặp	Các cạnh được xét	Cạnh được chọn	Các đỉnh v ∈T	T
Khởi tạo	-	-	d	∅
1	(d,c,26); (d,e,4); (d,g,14)	(d,e,4)	d, e	(d,e,4)
2	(d,c,26); (d,g,14); (e,f,18); (e,g,12)	(e,g,12)	d, e, g	(d,e,4); (e,g,12)
3	(d,c,26); (e,f,18); (g,b,3); (g,c,30); (g,f,2)	(g,f,2)	d, e, g, f	(d,e,4); (e,g,12); (g,f,2)
4	(d,c,26); (g,b,3); (g,c,30); (f,a,8); (f,b,10)	(g,b,3)	d, e, g, f, b	(d,e,4); (e,g,12); (g,f,2); (g,b,3)
5	(d,c,26); (g,c,30); (f,a,8); (b,a,5); (b,c,16)	(b,a,5)	d, e, g, f, b, a	(d,e,4); (e,g,12); (g,f,2); (g,b,3); (b,a,5)
6	(d,c,26); (g,c,30); (b,c,16);	(b,c,16)	d, e, g, f, b, c	(d,e,4); (e,g,12); (g,f,2); (g,b,3); (b,a,5); (b,c,16)

➔ **Cây khung nhỏ nhất của đồ thị là T = {(d, e, 4), (e, g, 12), (g, f, 2), (g, b, 3), (b, a, 5), (b, c, 16)}, có tổng trọng số = 4 + 12 + 2 + 3 + 5 + 16 = 42.**

Bài tập: Minh họa quá trình tìm cây khung nhỏ nhất của đồ thị bên dưới bằng thuật toán **Prim**:



4.3. Cài đặt thuật toán Prim

Input: đồ thị G(V,E)

Output: một cây khung *T* là cây khung nhỏ nhất của đồ thị

```
Prim()
    T ← ∅
    T_Vertices ← x0   //Danh sách các đỉnh trong cây khung
    Lặp (n-1) lần để chọn (n-1) đỉnh còn lại
        minWeight ← INF    //Trọng số cạnh nhỏ nhất
        min_u ← -1
        min_v ← -1
        Với mỗi đỉnh u ∈ T_Vertices
            Với mỗi đỉnh v kề u và v ∉ T_Vertices
                Nếu w(u,v) < minWeight
                    minVertex ← v
                    min_u ← u
                    min_v ← v
        T ← T ∪ (min_u, min_v)
        T_Vertices ← min_v
```

4.4. Độ phức tạp của thuật toán Prim: tùy thuộc vào dữ liệu lưu đồ thị để tìm cạnh có trọng số nhỏ nhất:

- Ma trận kề: **O(|V|²)**
- Heap nhị phân + Danh sách kề: **O(|E|.log(|V|))**

--- HẾT ---