

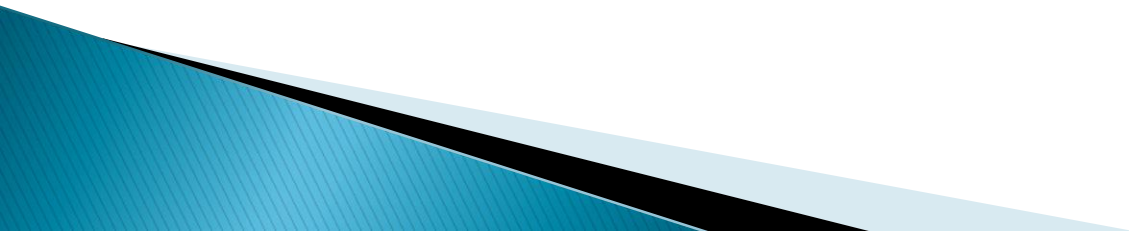
# Parcurgerea în adâncime



# Parcurgerea în adâncime

Se vizitează

- Inițial: vârful de start  $s$  – devine vârful curent



# Parcurgerea în adâncime

Se vizitează

- **Inițial:** vârful de start  $s$  – devine vârful curent
- **La un pas:**
  - se trece la primul vecin nevizitat al vârfului curent, **dacă există**

# Parcurgerea în adâncime


Se vizitează

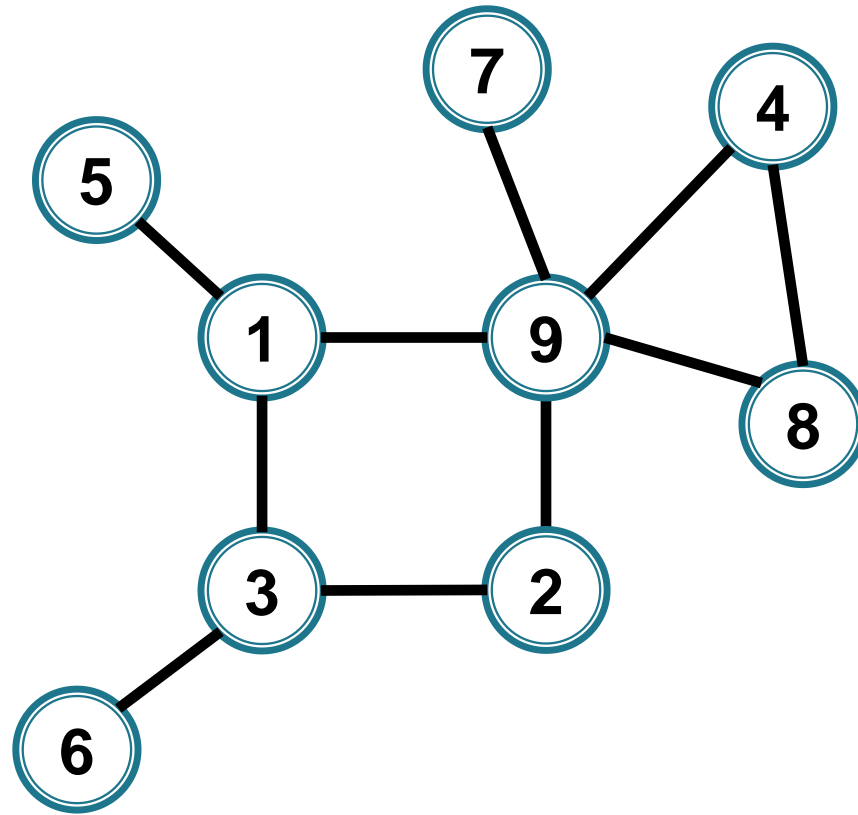
- **Inițial:** vârful de start  $s$  – devine vârf curent
- **La un pas:**
  - se trece la primul vecin nevizitat al vârfului curent, **dacă există**
  - altfel
    - se merge **înapoi** pe drumul de la  $s$  la vârful curent, până se ajunge la un vârf cu vecini nevizitați

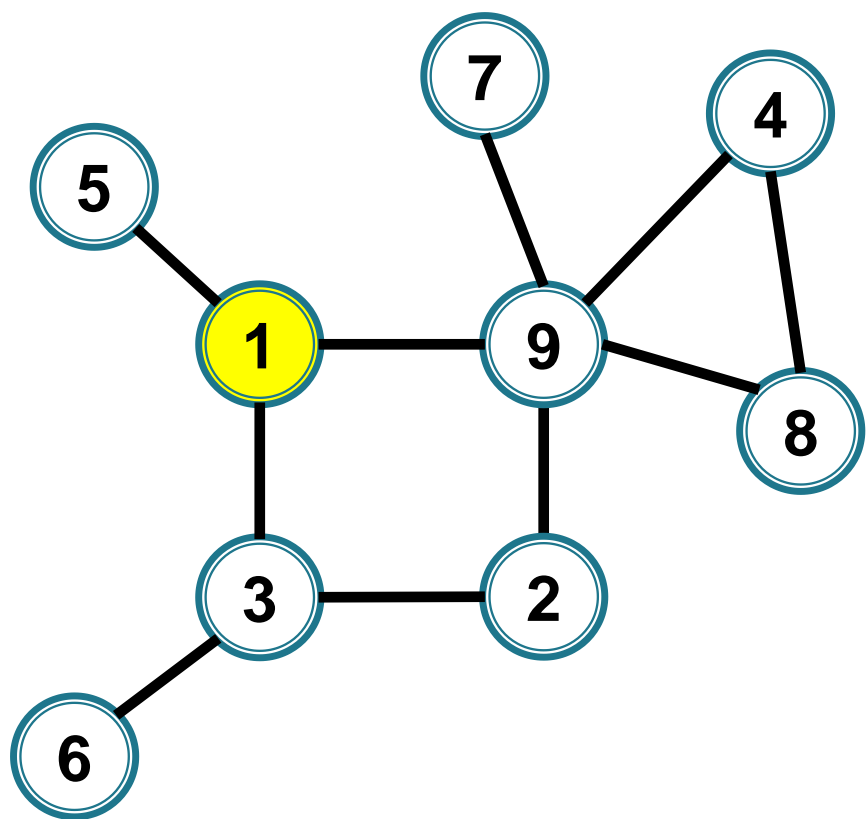
-

# Parcurgerea în adâncime

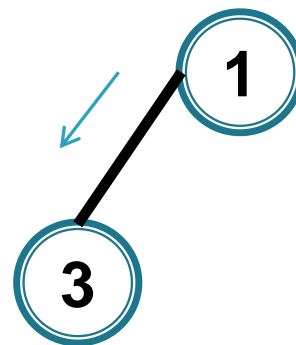
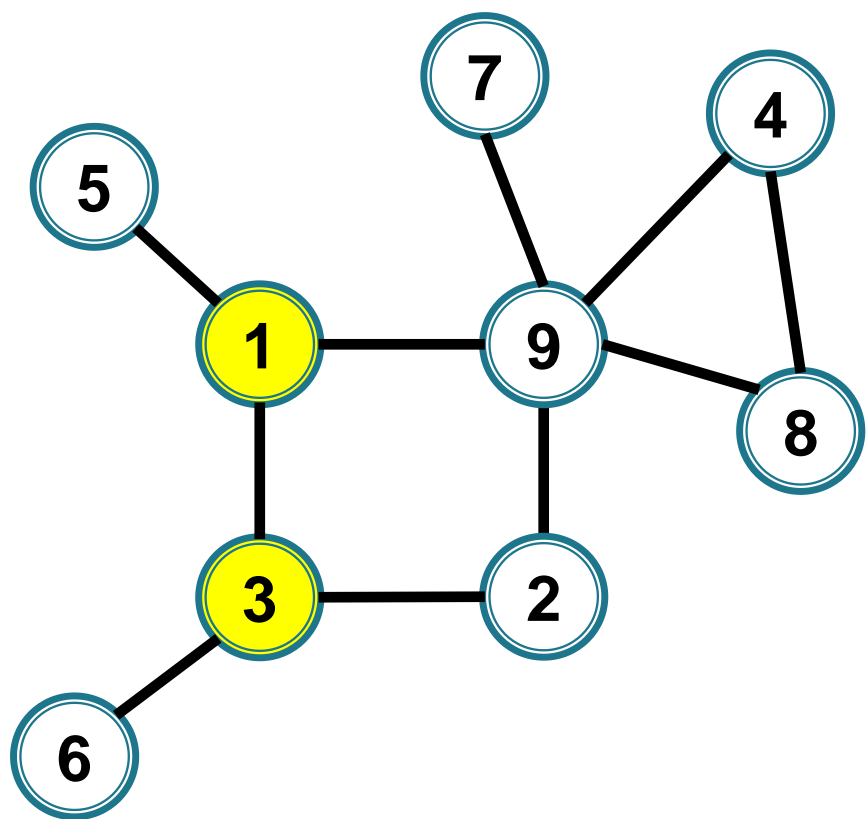
Se vizitează

- Inițial: vârful de start  $s$  – devine vârf curent
  - La un pas:
    - se trece la primul vecin nevizitat al vârfului curent, **dacă există**
    - altfel
      - se merge **înapoi** pe drumul de la  $s$  la vârful curent, până se ajunge la un vârf cu vecini nevizitați
      - se trece la **primul** dintre aceștia și se reia procesul
- 

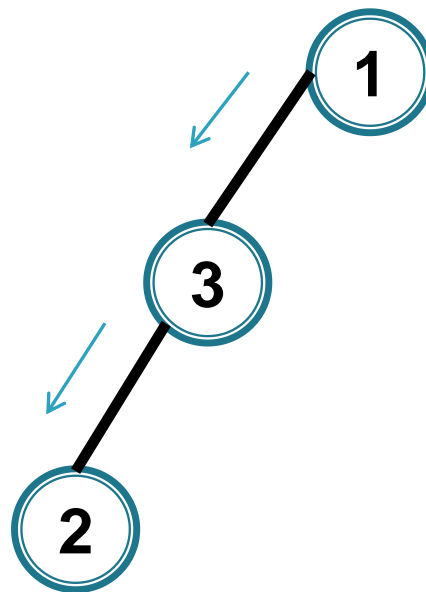
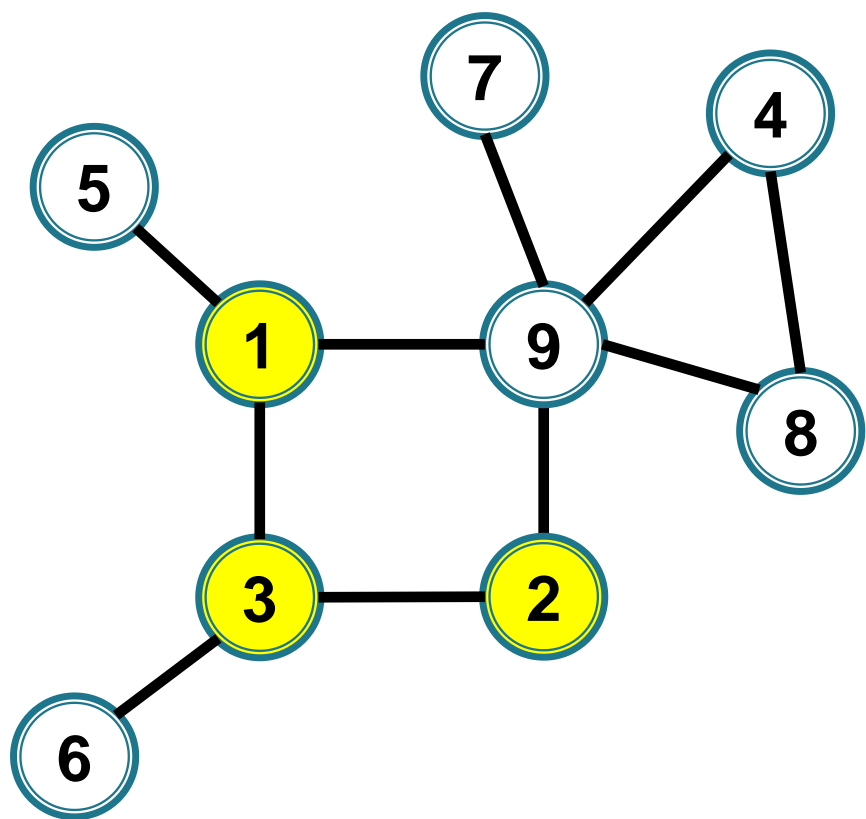


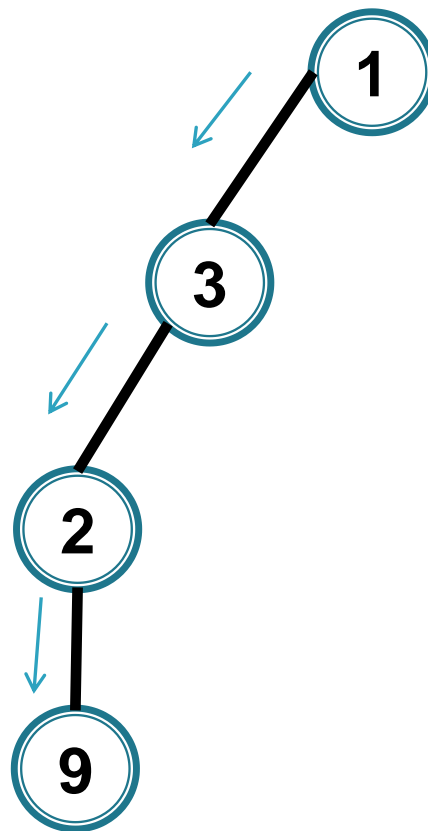
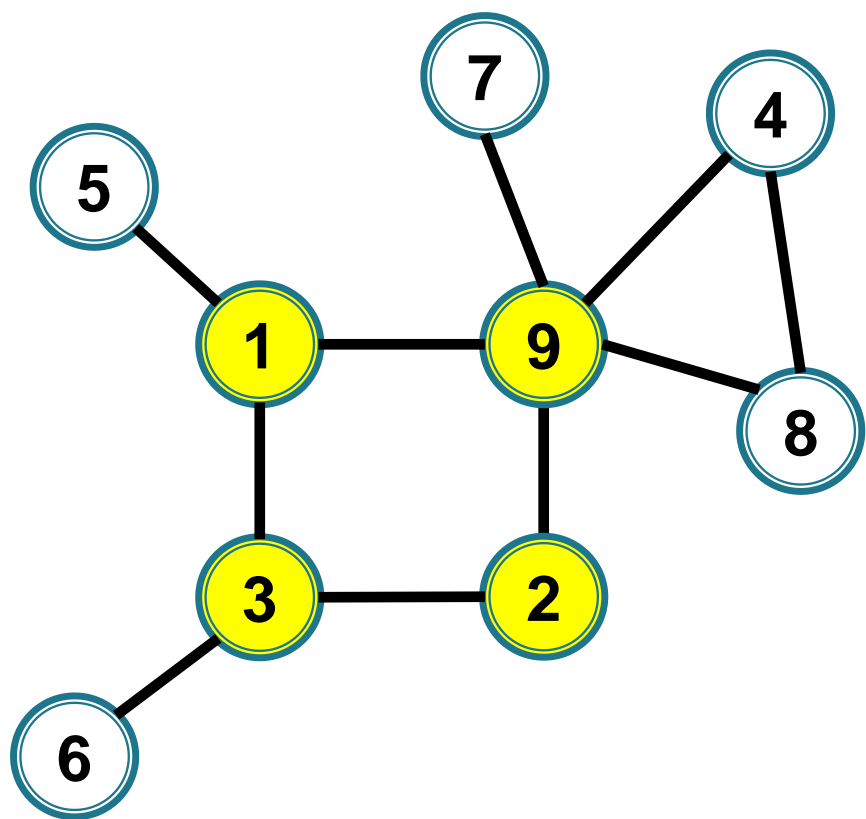


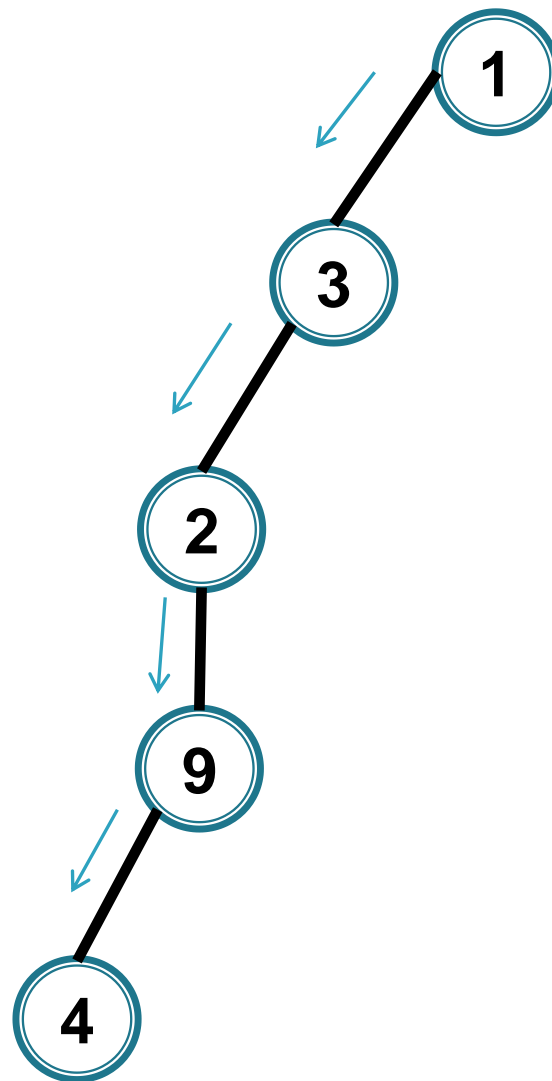
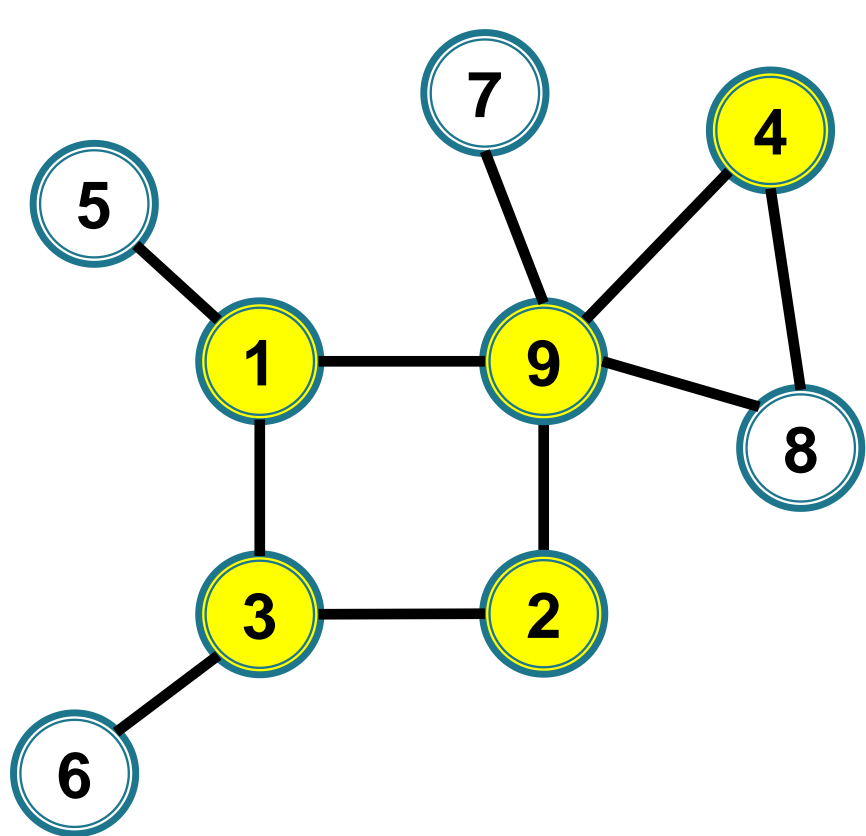
1

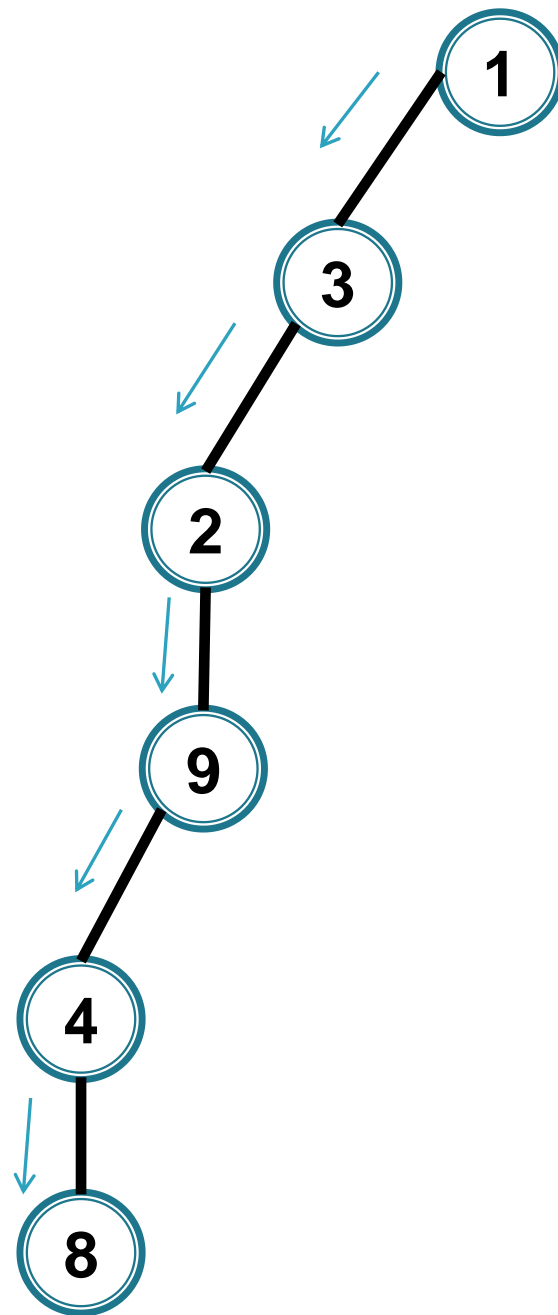
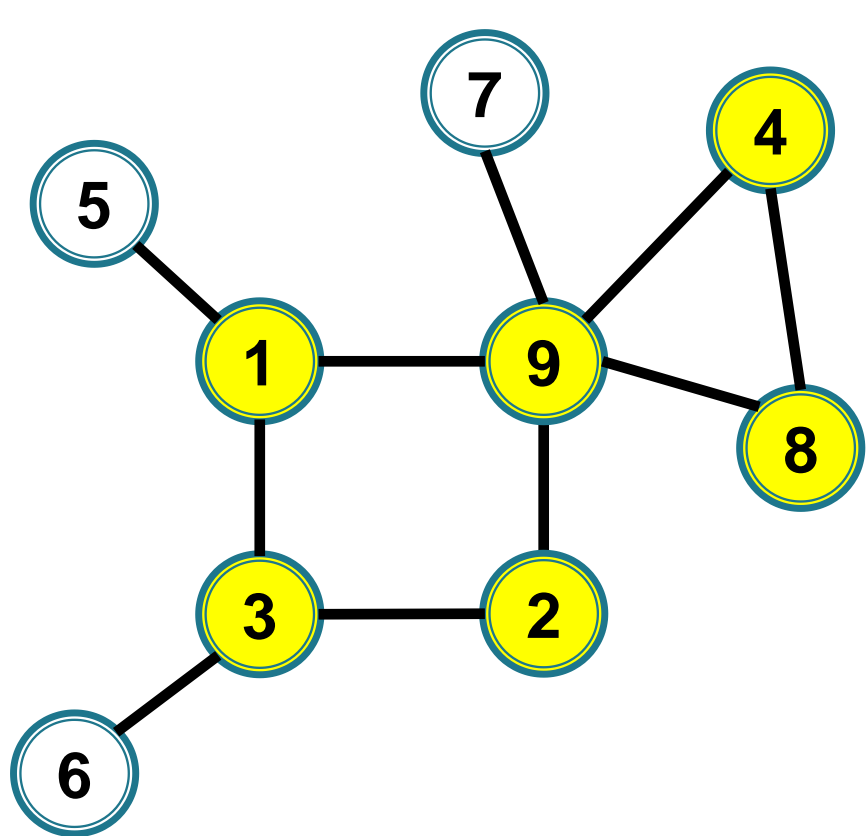


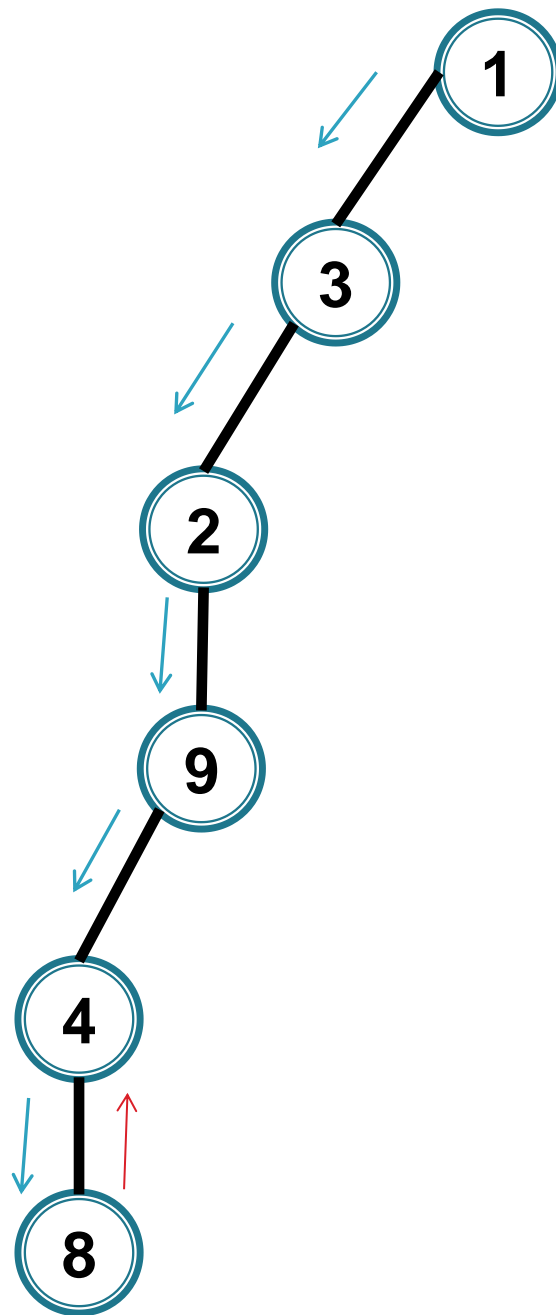
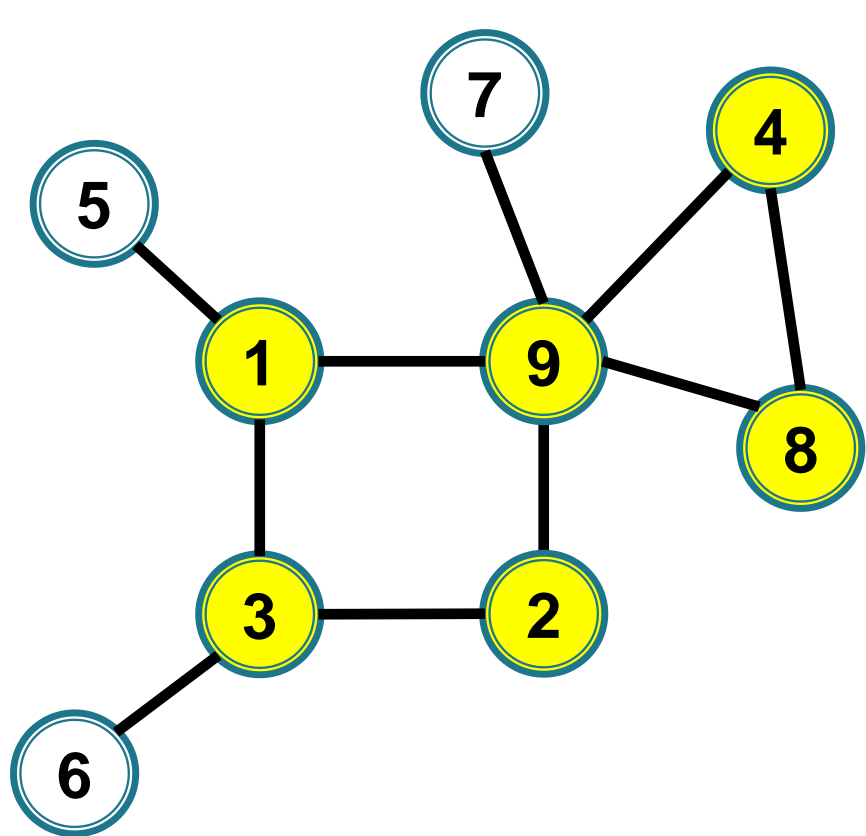


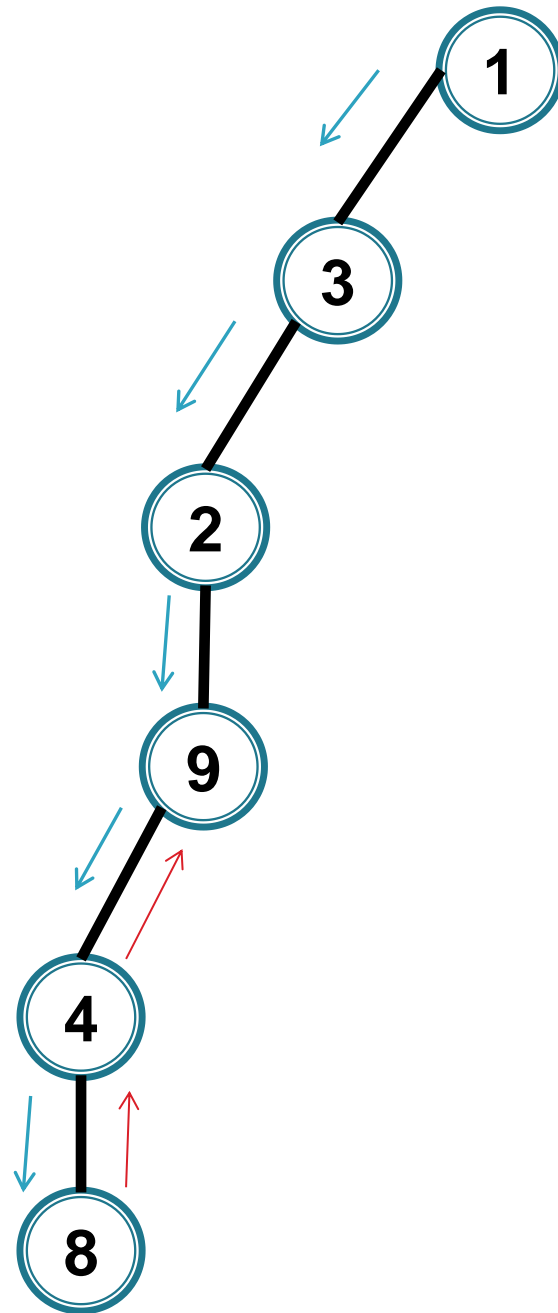
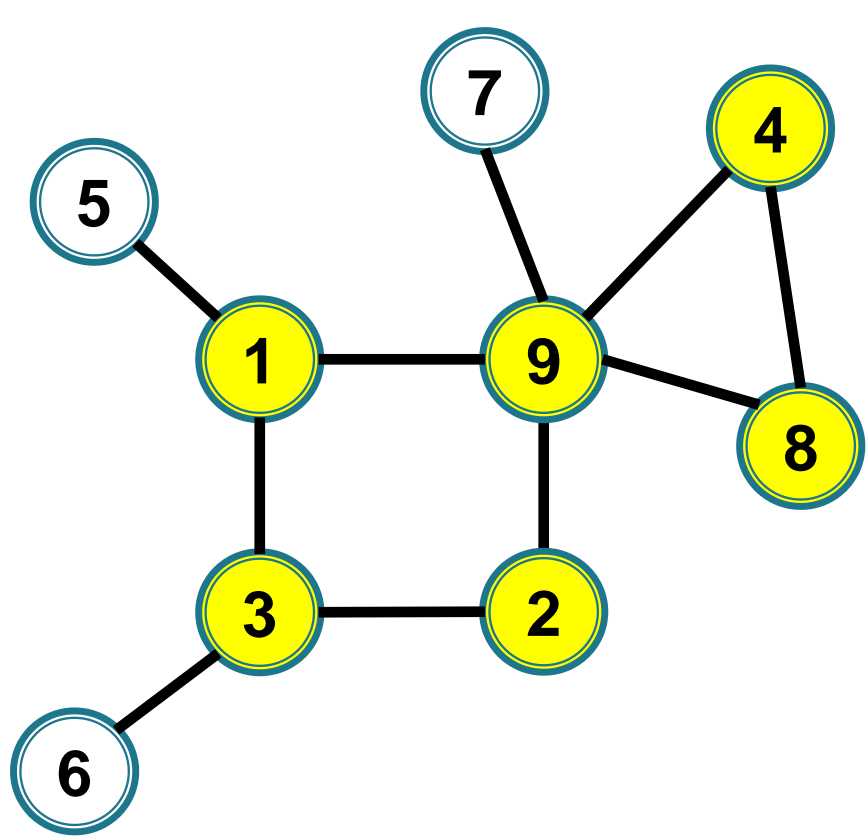


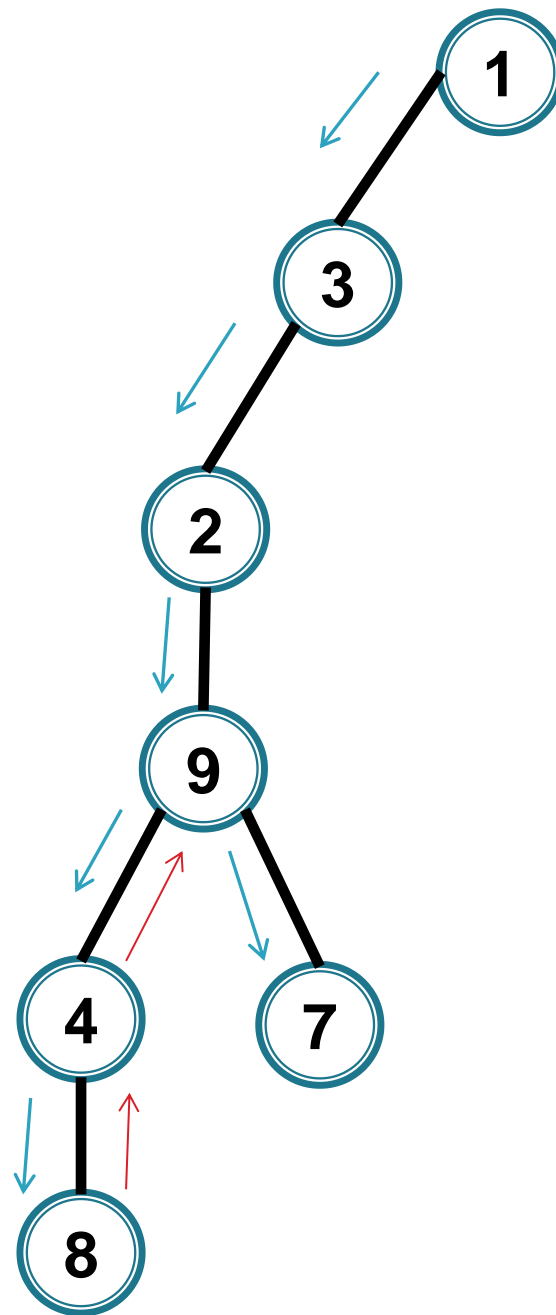
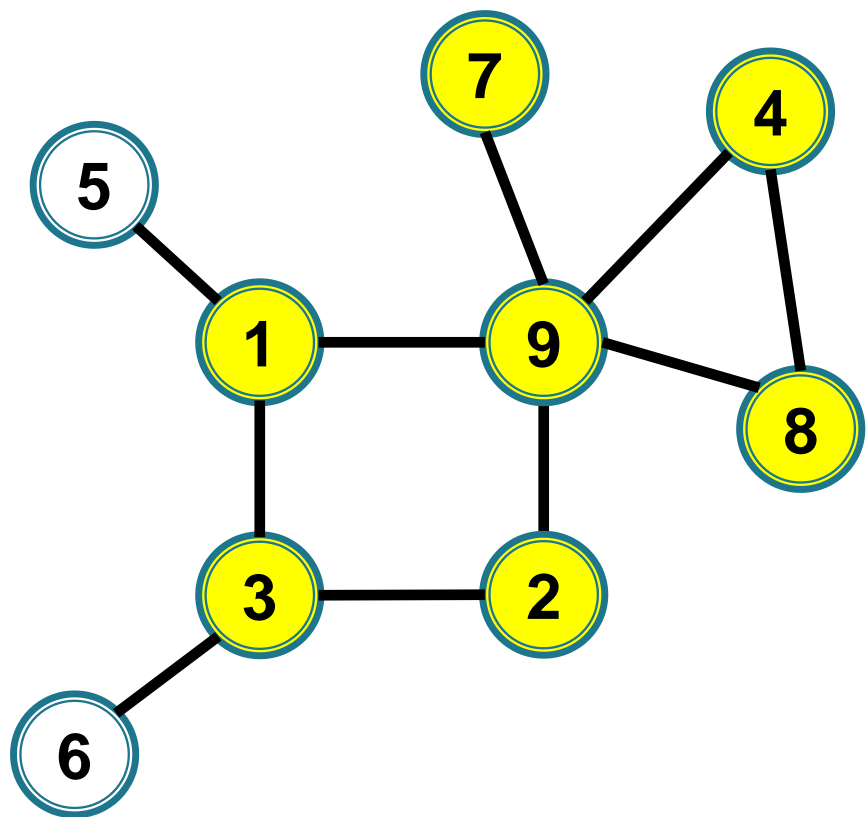


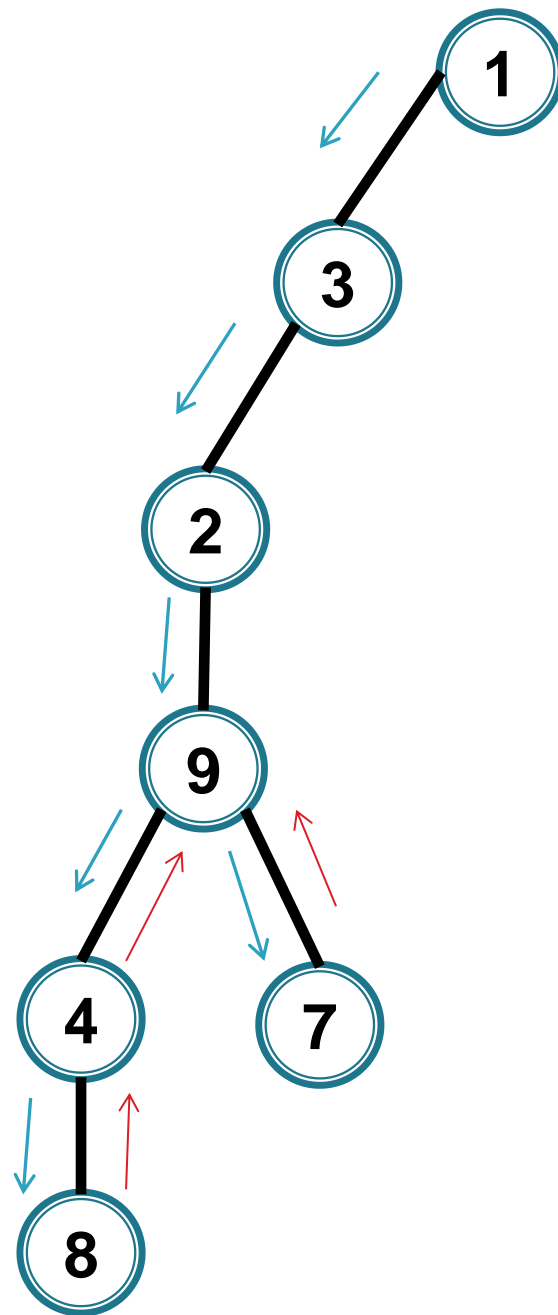
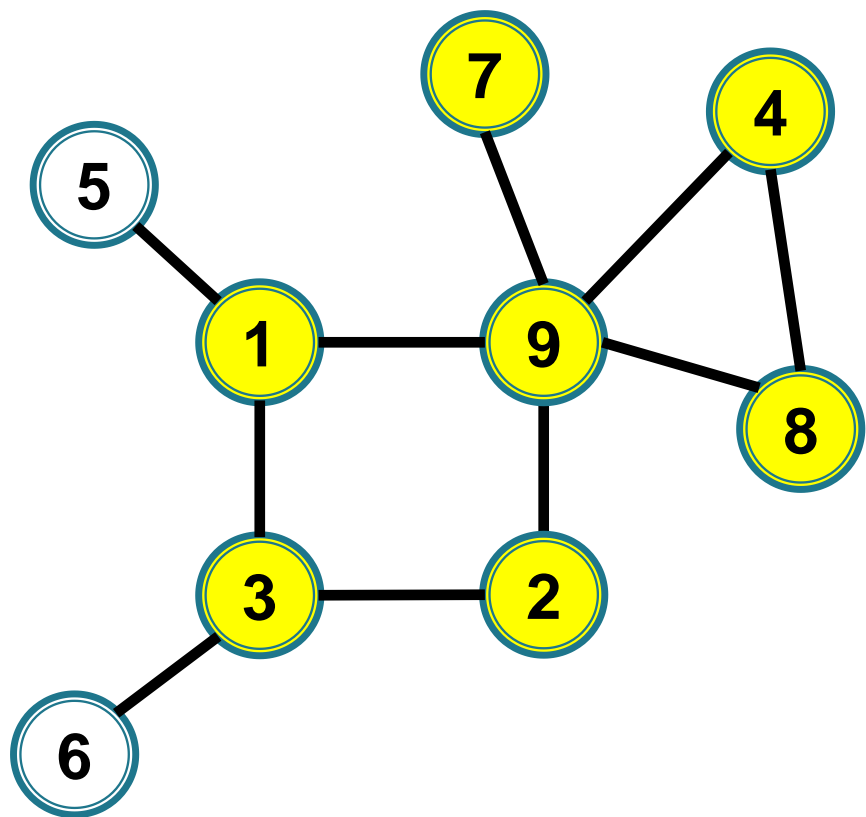




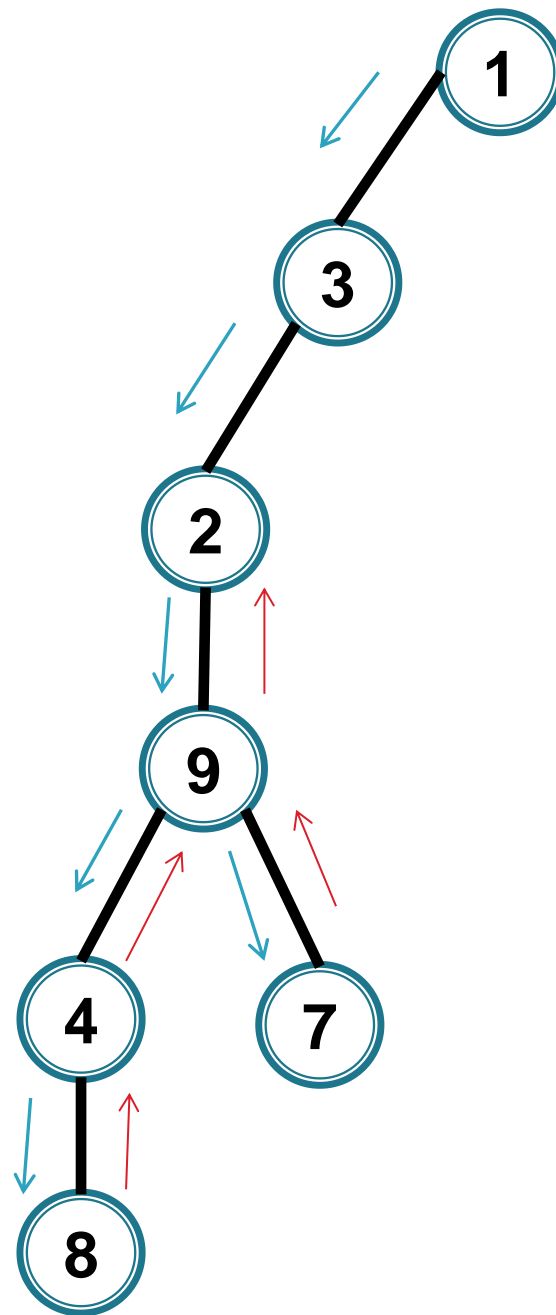
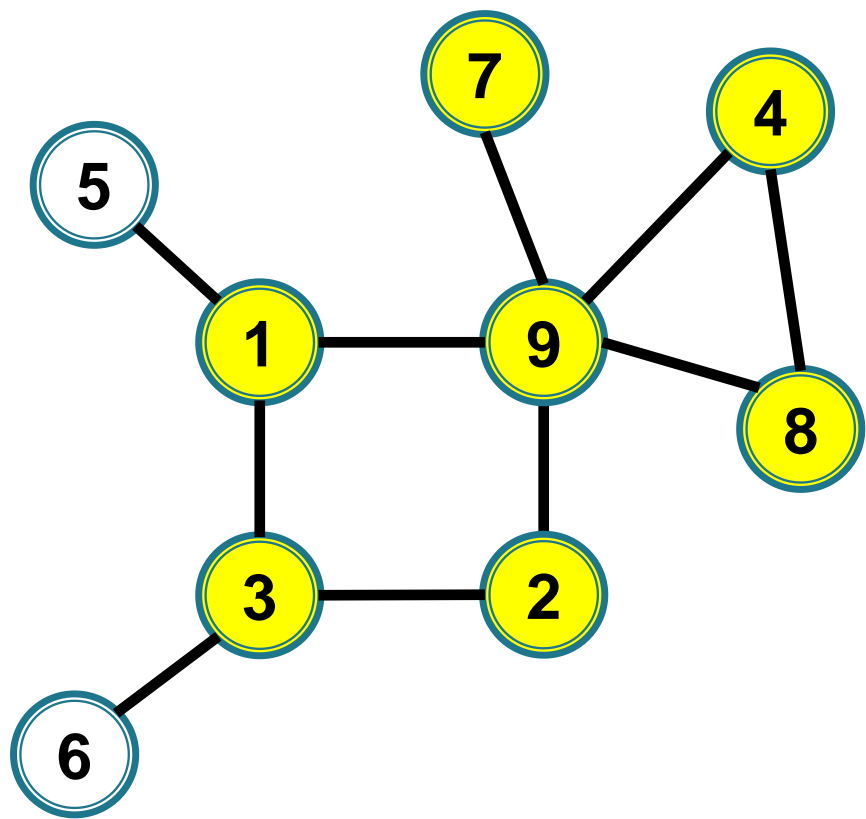


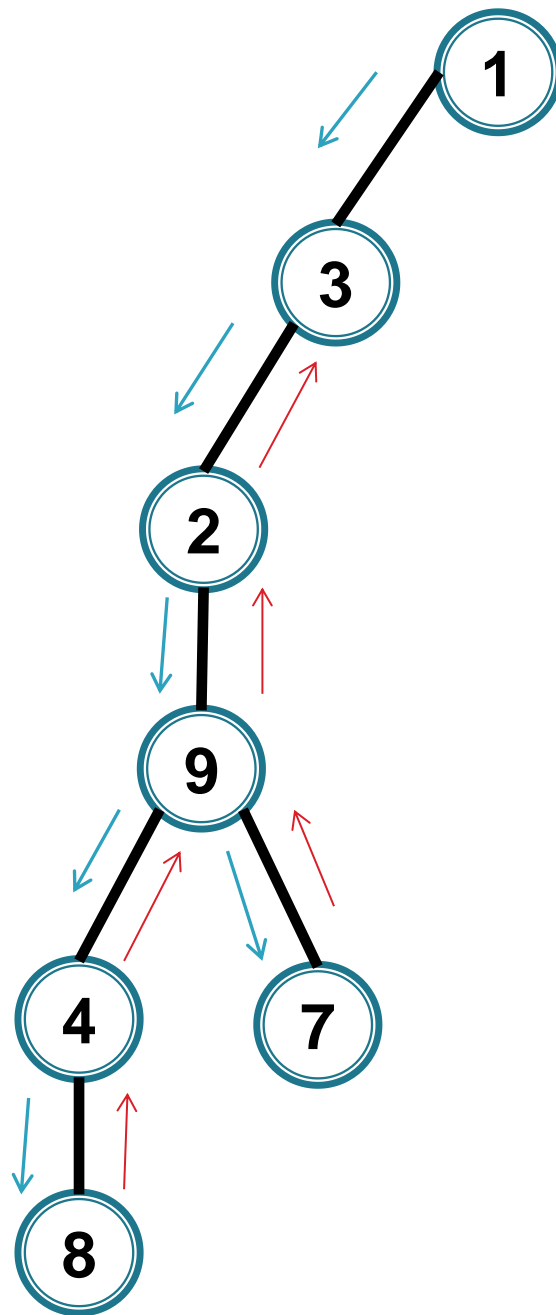
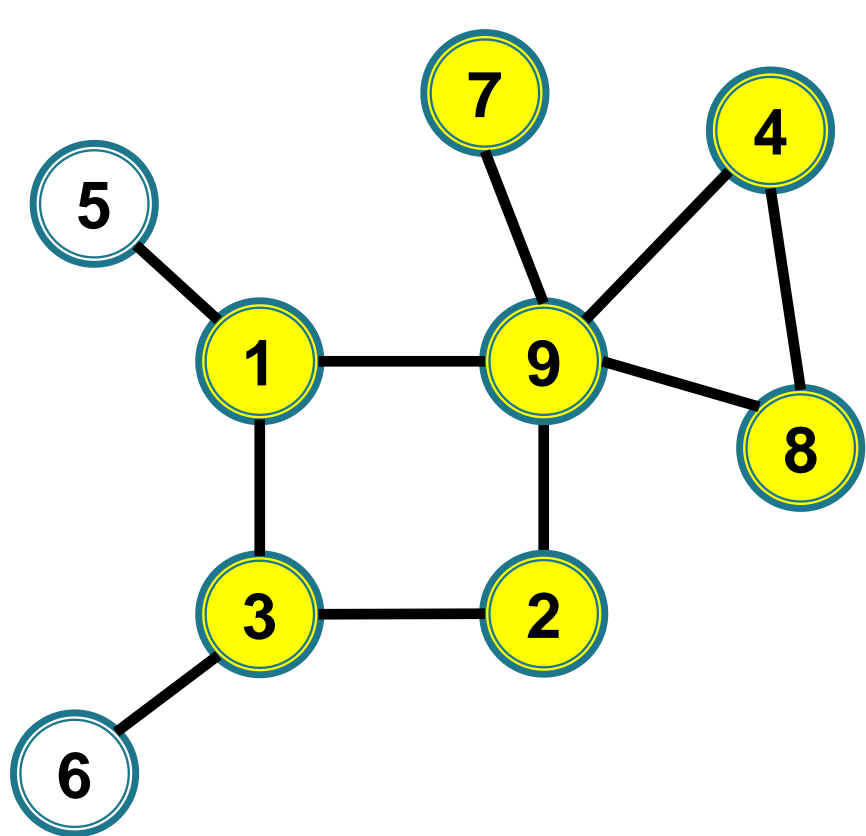


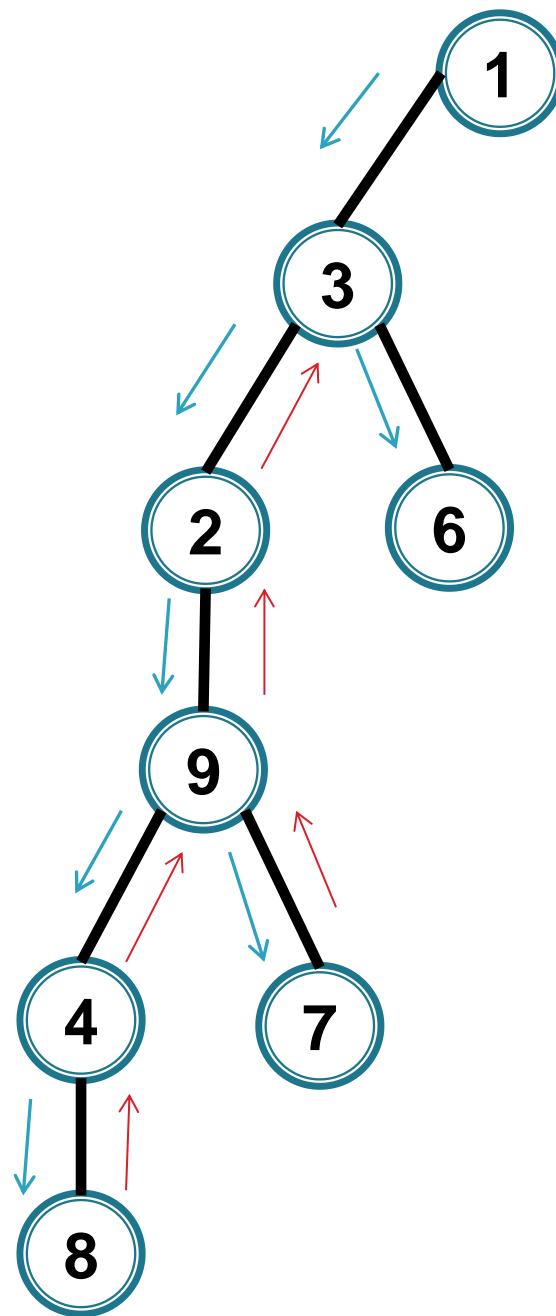
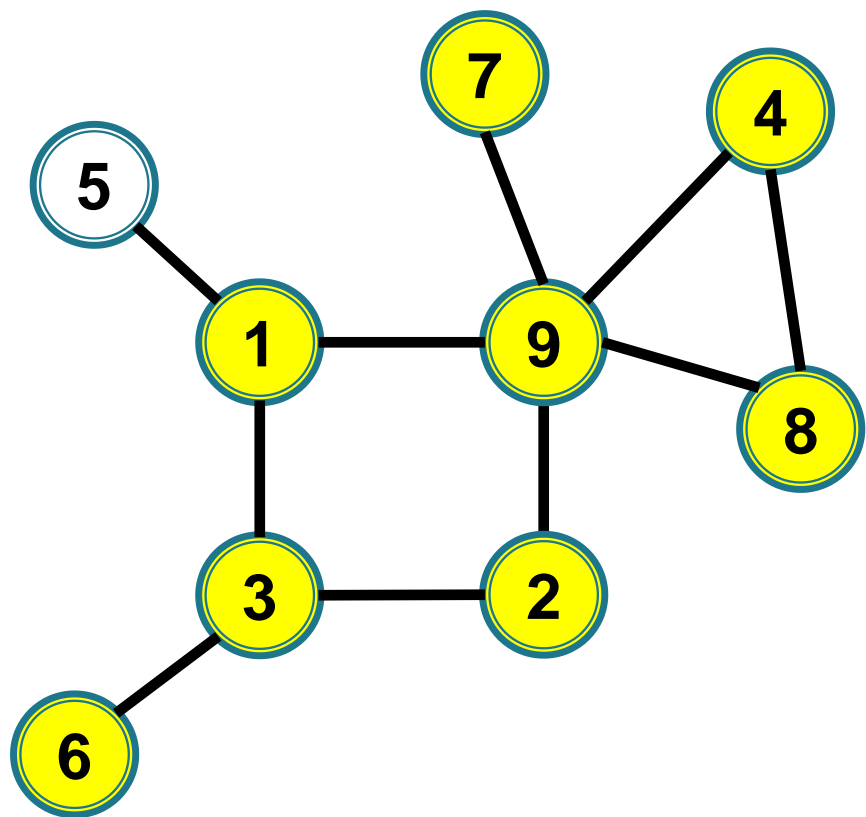


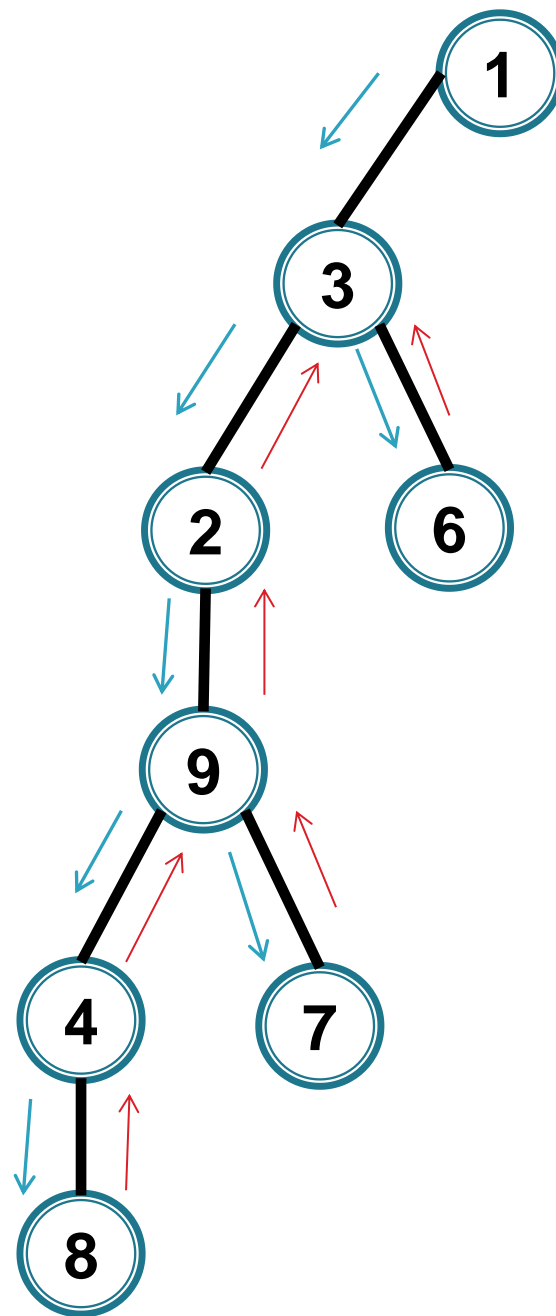
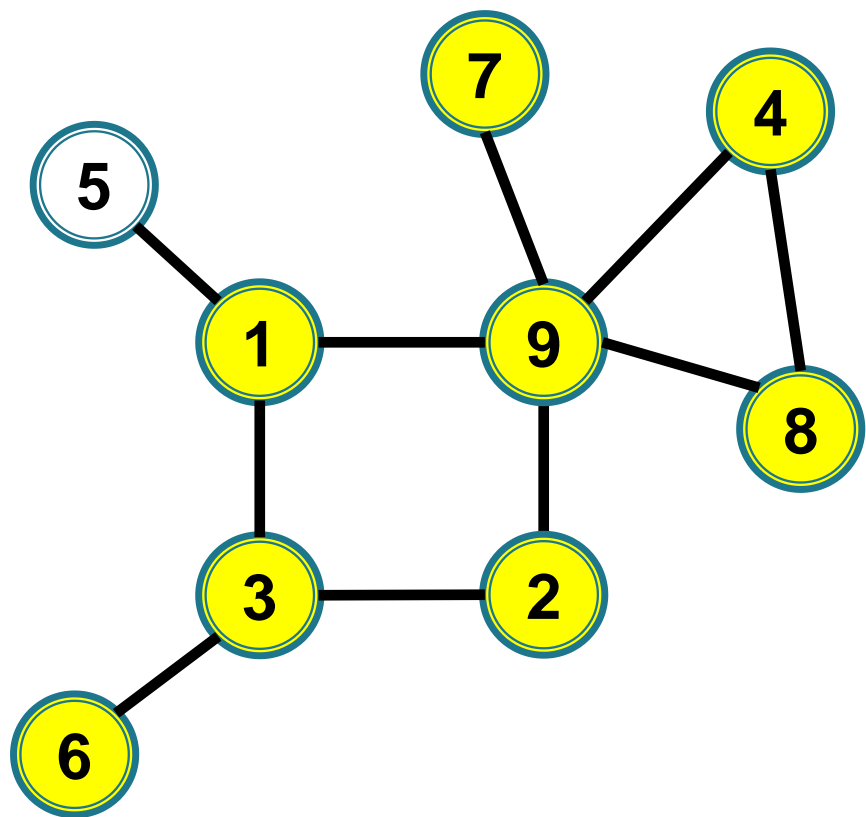


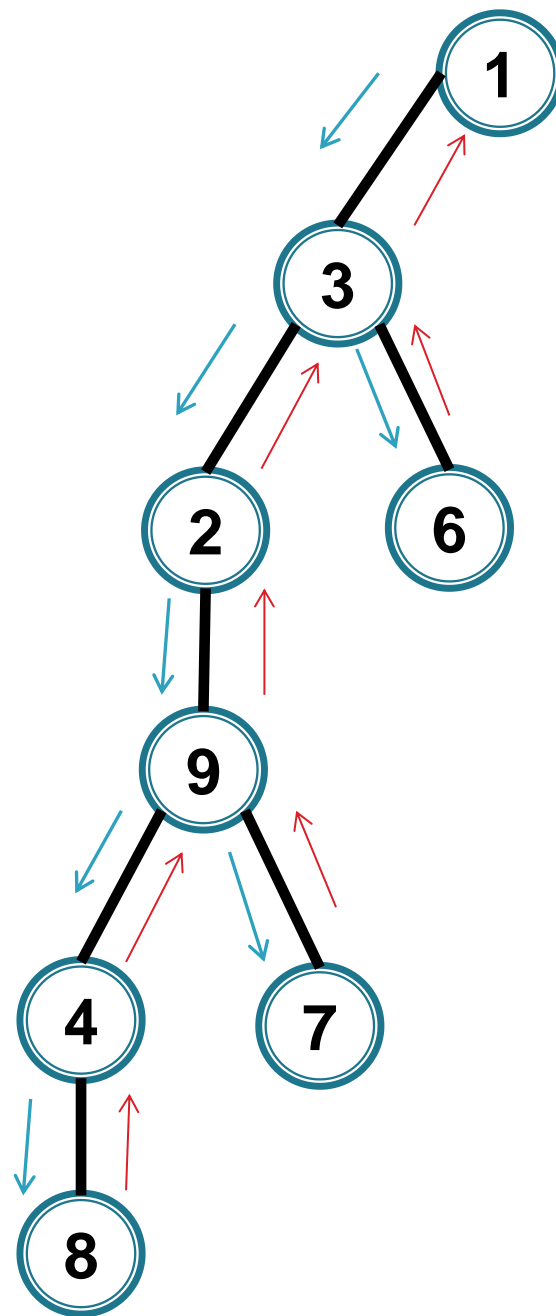
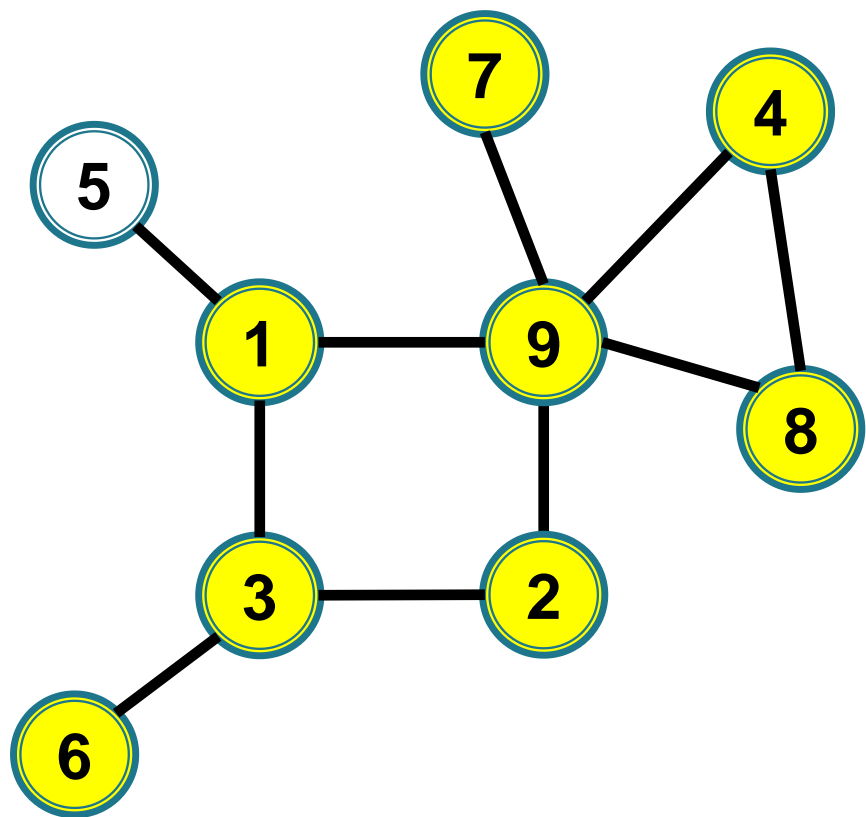


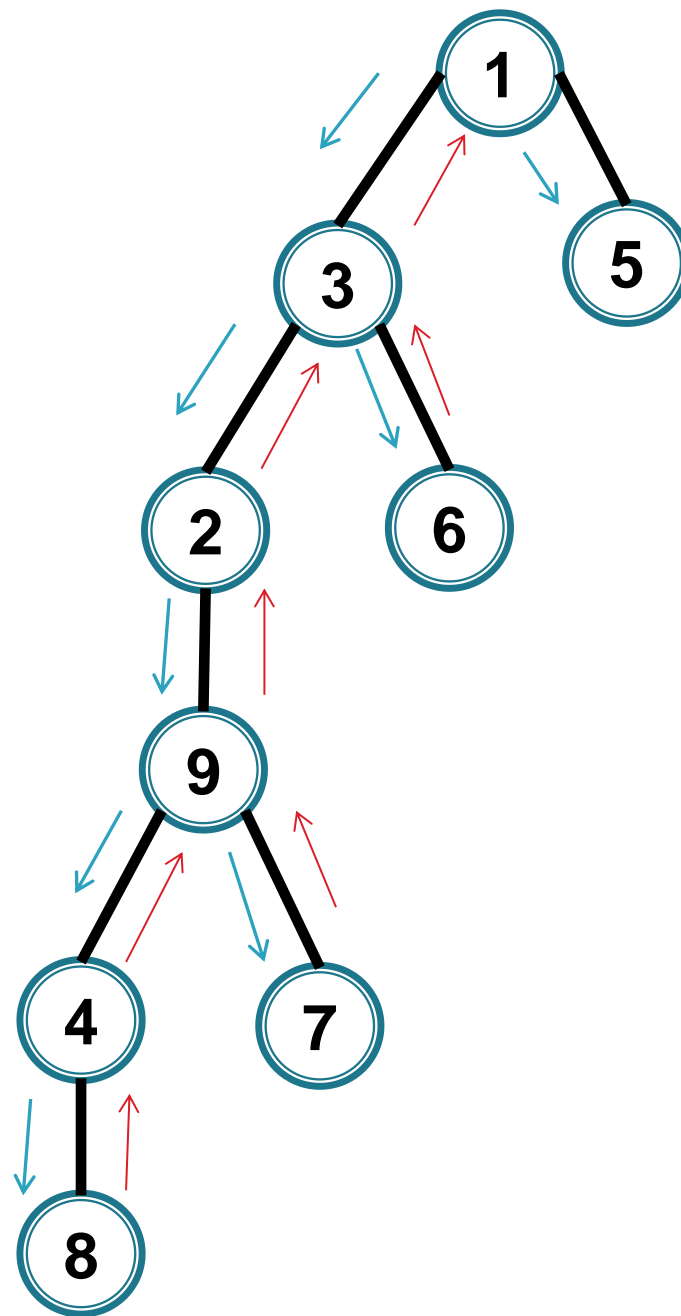
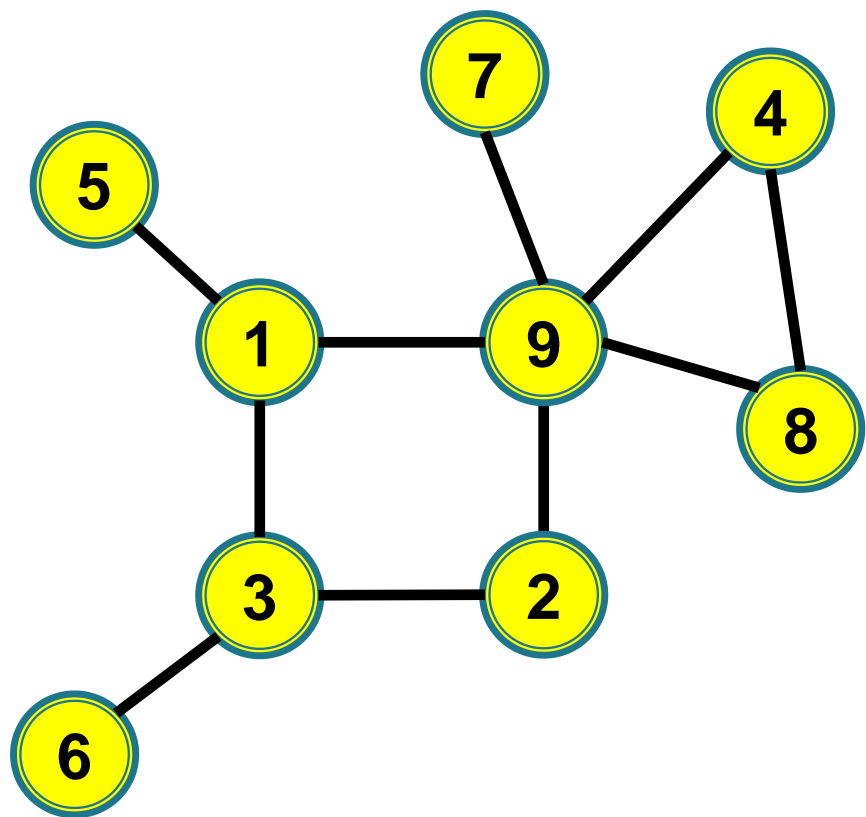


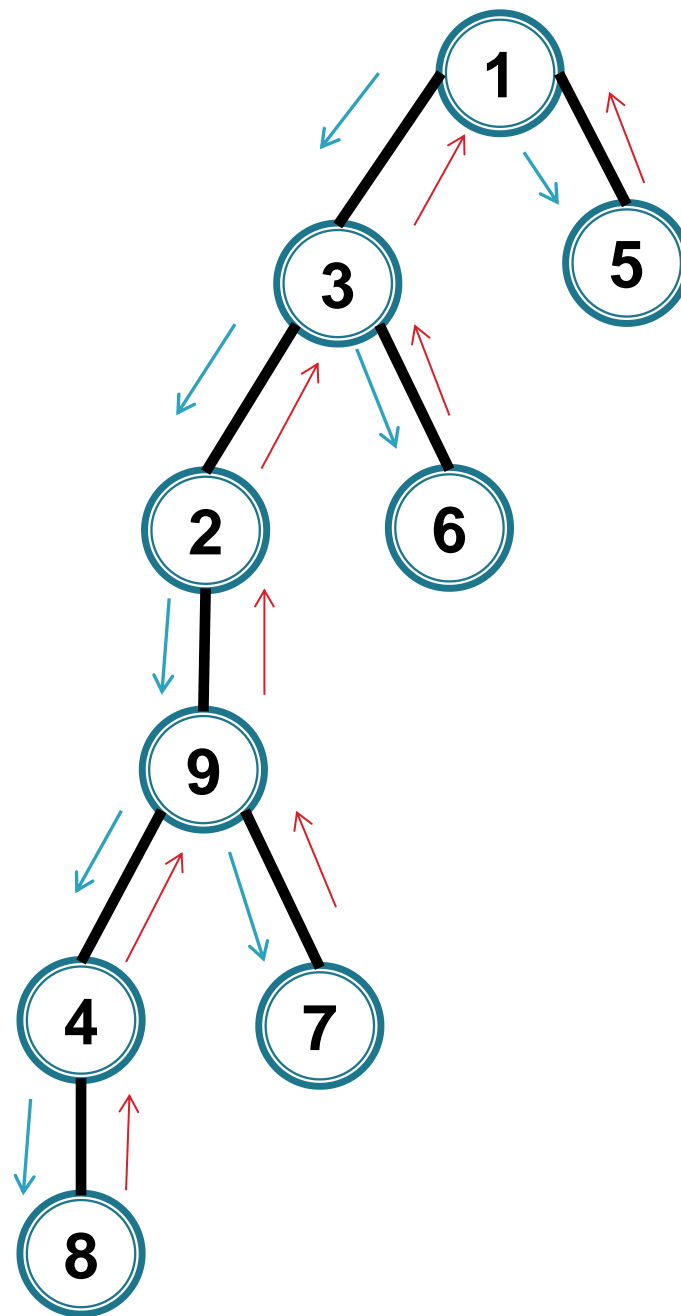
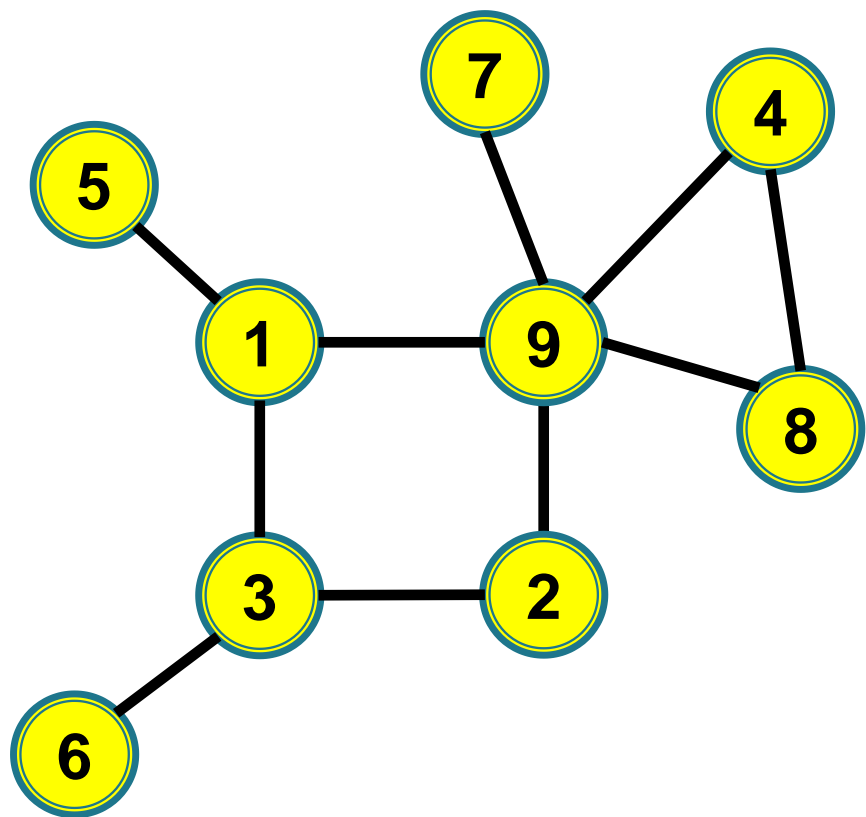


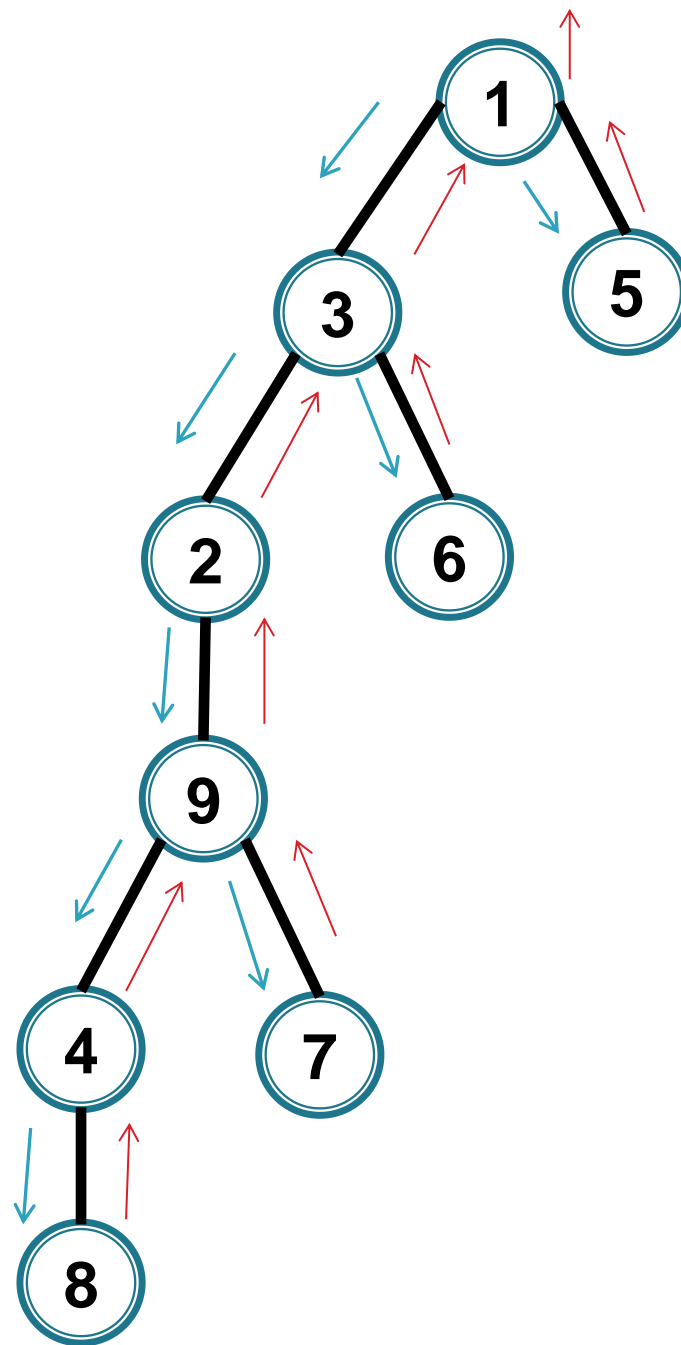
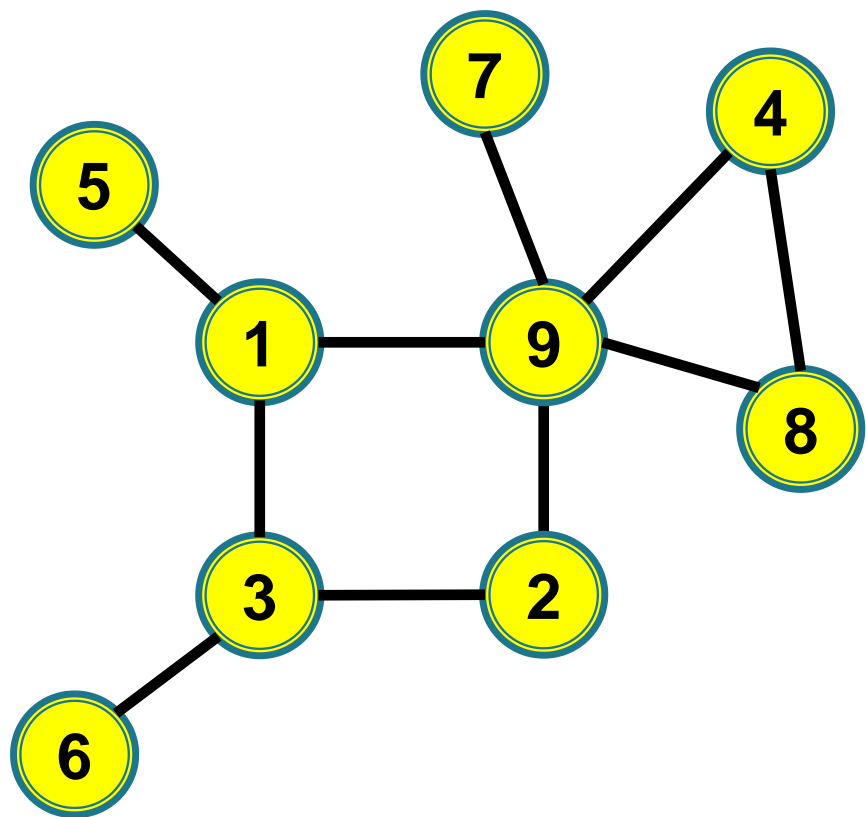














```
void df(int i) {
```

```
void df(int i){  
    viz[i]=1;  
    cout<<i<<" ";
```

```
void df(int i){  
    viz[i]=1;  
    cout<<i<<" ";  
    for(int j=1;j<=n ;j++)  
        if(a[i][j]==1)
```

```
void df(int i){
    viz[i]=1;
    cout<<i<<" ";
    for(int j=1;j<=n ;j++)
        if(a[i][j]==1)
            if(viz[j]==0){
                tata[j]=i;
                df(j);
            }
}
```

```
void df(int i){
    viz[i]=1;
    cout<<i<<" ";
    for(int j=1;j<=n ;j++)
        if(a[i][j]==1)
            if(viz[j]==0){
                tata[j]=i;
                df(j);
            }
}
```

Apel:

df(s)

# Alte aplicații



Dat un graf neorientat, să se verifice dacă graful conține cicluri și, în caz afirmativ, să se afișeze un ciclu al său

# Alte aplicații



Dat un graf neorientat, să se verifice dacă graful conține cicluri și, în caz afirmativ, să se afișeze un ciclu al său



Un ciclu se încheie în parcurgere când vârful curent are un vecin deja vizitat, care nu este tatăl lui

# Alte aplicații



Să se verifice dacă un graf neorientat dat este bipartit



# Graf bipartit

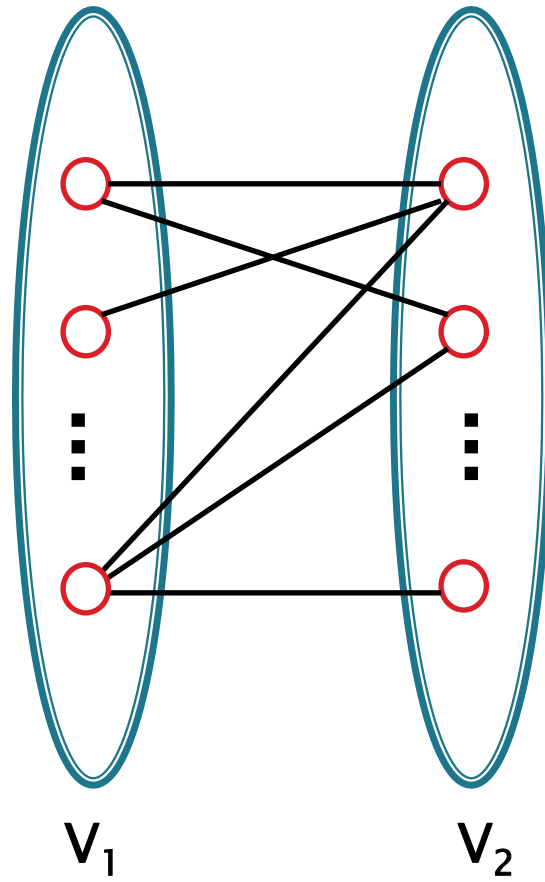
- ▶ Un graf neorientat  $G = (V, E)$  se numește **bipartit**  $\Leftrightarrow$  există o partiție a lui  $V$  în două submulțimi nevide  $V_1, V_2$  (**bipartiție**):

$$V = V_1 \cup V_2$$

$$V_1 \cap V_2 = \emptyset$$

astfel încât orice muchie  $e \in E$  are o extremitate în  $V_1$  și cealaltă în  $V_2$ :

$$|e \cap V_1| = |e \cap V_2| = 1$$



# Graf bipartit

- ▶  $G = (V, E)$  **bipartit**  $\Leftrightarrow$  există o colorare a vârfurilor cu două culori:

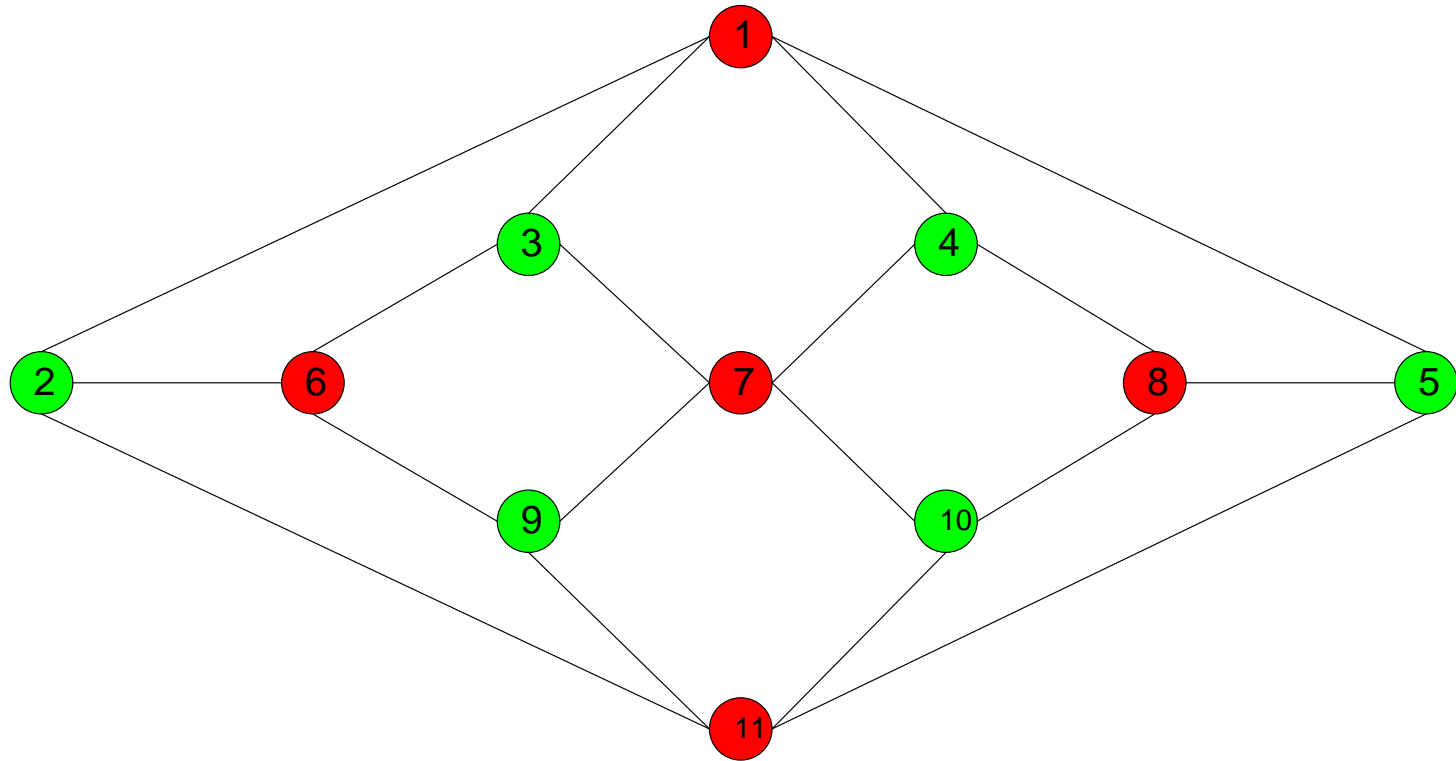
$$c : V \rightarrow \{1, 2\}$$

astfel încât pentru orice muchie  $e=xy \in E$  avem

$$c(x) \neq c(y)$$

(**bicolorare**)

# Graf bipartit



# Graf bipartit

