ProgTeam Week 9

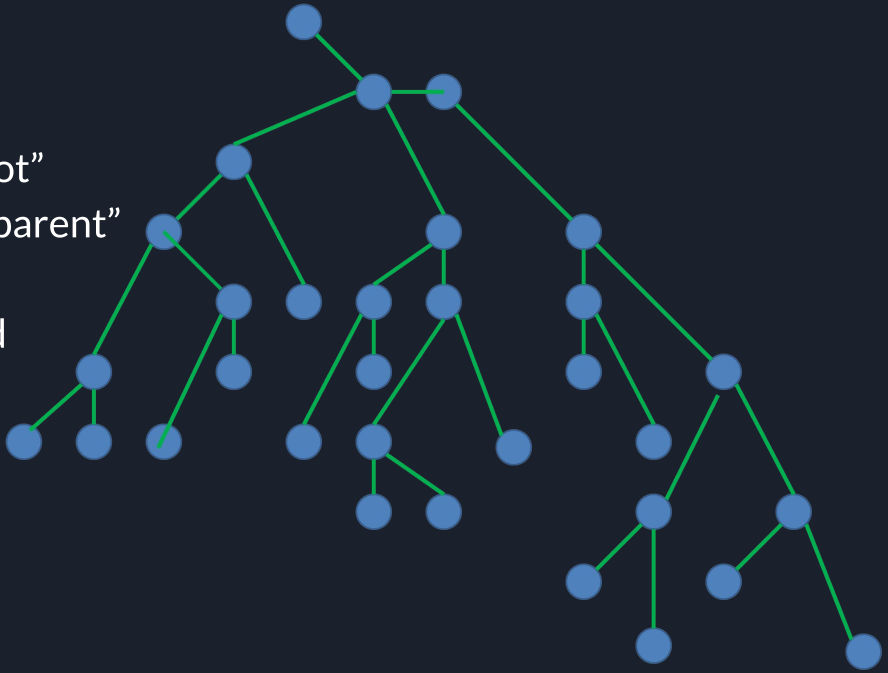
Trees

Trees

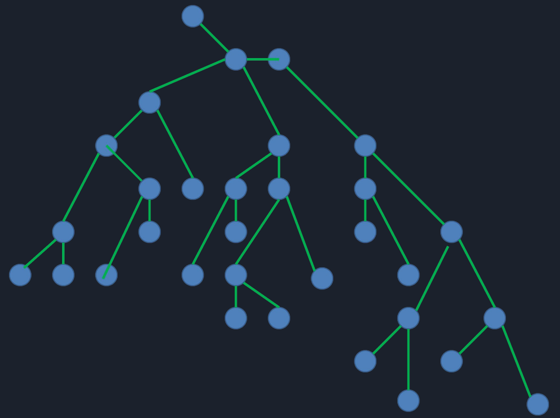
- Special graph with (n - 1) edges
- Connected, but has no cycles

- Can be represented with a single “root”
- All other vertices have exactly one “parent”

- Vertices without “children” are called “leaves”



DFS



- Most common algorithm on a tree is DFS
- Store edges using a series of lists
- Recurse to "'children", but not to the parent

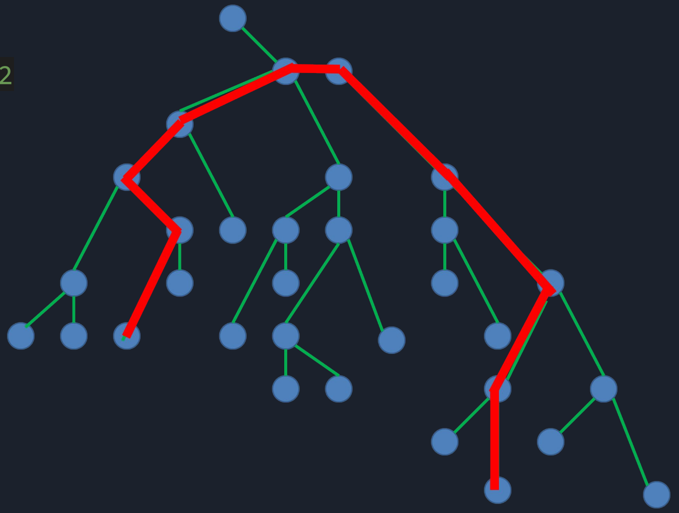
```
// This counts how many vertices are in a tree, by calling dfs(root, -1)
int dfs(int cur, int parent){
    int cnt = 1;
    for(int child : adj[cur]){
        if(child != parent){
            cnt += dfs(child, cur);
        }
    }
    return cnt;
}
```

DFS Examples: Finding the distance to the root

```
// The distance from the root to the node (the "depth")
// is equal to the depth of the parent plus one.
int depth[N];
void dfs(int cur, int parent){
    for(int child : adj[cur]){
        if(child != parent){
            // pass information to the child from the parent
            depth[child] = depth[cur] + 1;
            dfs(child, cur);
        }
    }
}
```

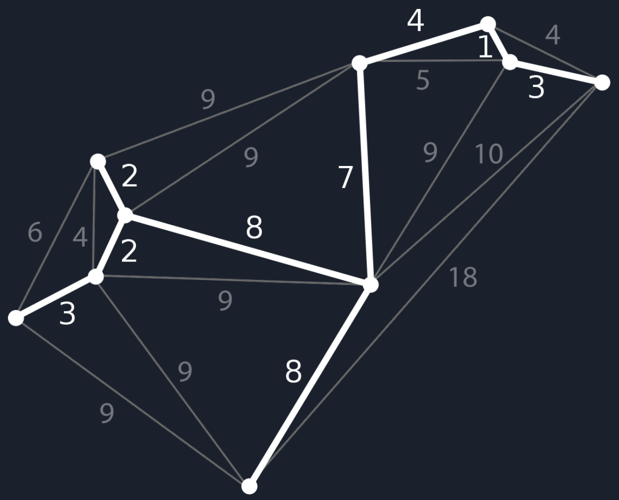
DFS Examples: “Diameter” of a tree

```
// The "diameter" of a tree is the maximum distance
// between any two nodes in the tree.
int diameter = 0;
int dfs(int u, int parent){ //return the longest distance to a leaf
    int max1 = 0, max2 = 0; // longest two paths to a leaf
    for(int v : adj[u]){
        if(u == parent) continue;
        int dist = dfs(v, u) + 1;
        if(dist > max1){
            max2 = max1; // max1 is always larger than max2
            max1 = dist;
        }else if(dist > max2){
            max2 = dist;
        }
    }
    diameter = max(diameter , max1 + max2);
    return max1;
}
```



Minimum Spanning Tree

- Given a graph with more than (N - 1) edges, the minimum spanning tree is the tree that connects all vertices using the minimum weight



Kruskal’s MST Algorithm

- Greedily process edges from smallest to largest weight

```
// Pseudocode for Kruskal's MST algorithm
initialize disjoint-sets // (see last week)
sort edges by weight (low to high)
for edge[u,v,weight] in edges:
    if u and v not in same set:
        add edge to MST
        union sets containing u and v
```

Prim’s MST Algorithm

- Build tree one edge at a time; select next vertex that is closest to current tree

```
initialize priority queue
initialize distance array to {INF}, visited array to {false}
add root to priority queue // any node can be root
while priority queue is not empty:
    next = pop from priority queue
    if visited[next]: continue
    visited[next] = true
    for (candidate, weight) in adjacent[next]:
        if not visited[candidate] and weight < distance[candidate]:
            distance[candidate] = weight
            add candidate to priority queue
```

Prim’s MST Algorithm

- If N is small (<= ~5000) we can ditch the priority queue
    - A little easier to implement

```
initialize distance array to {INF}, visited array to {false}
dist[root] = 0 // any node can be root
for i from 1 to n - 1:
    best_dist = INF, next = -1
    for j from 1 to n:
        if not visited[j] and dist[j] < best_dist:
            best_dist = dist[j]
            next = j
    visited[next] = true
    for (neighbor, weight) in graph[next]:
        if weight < dist[neighbor]:
            dist[neighbor] = weight
```