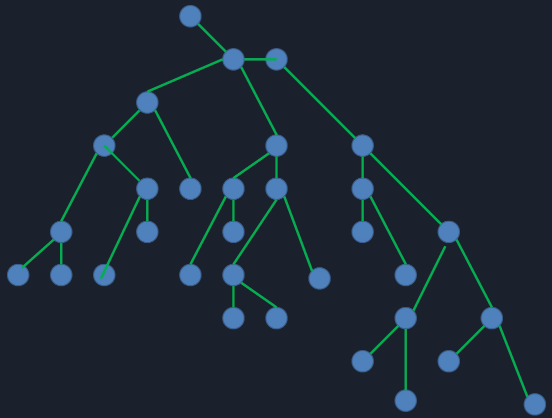
ProgTeam Spring Week 10

Directed Acyclic Graphs

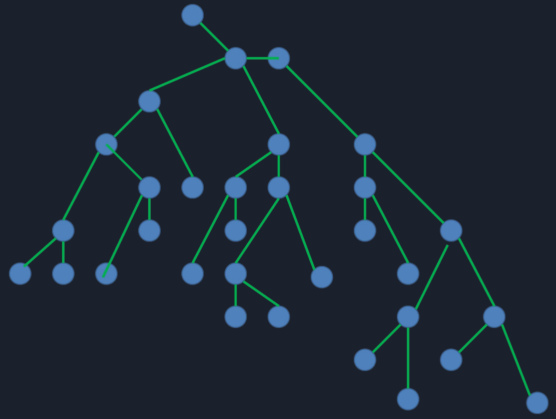
Refresher: Trees

- Trees are special graph with N vertices and exactly (N - 1) vertices
- Exactly one path between any two vertices
- We can use this information to DFS:
  - Recurse to children to gain information
- Example: find the furthest descendant with the opposite color (if all vertices are red and blue)



Refresher: Trees

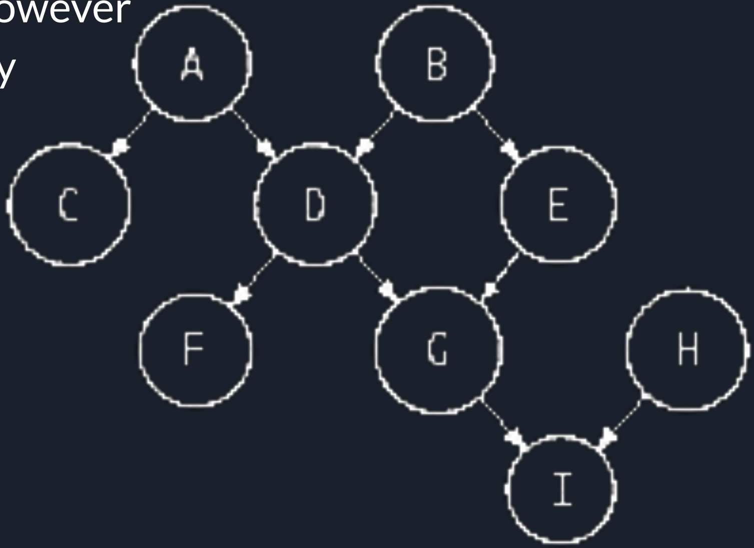
- Example: find the furthest descendant with the color red (we can re-run the algorithm on blue to get the final answer)



```
CalcDistance(i):
      Distance[i] = -INF; // No such vertex
      if i is red:
          Distance[i] = 0
      for j in Children[i]:
          Distance[i] = max(CalcDistance(j) + 1, Distance[i]);
      endfor
      return Distance[i];
```

DAGs: A generalization of trees

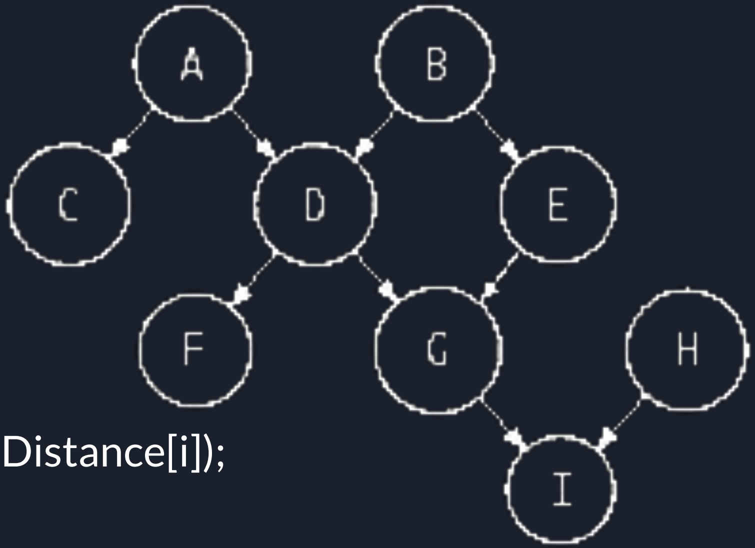
- Very similar structure: all vertices have a “parent”
- Now vertices can have several parents, however
- The general idea of our algorithm can stay the same, but we need to optimize it
    - Otherwise we’d keep visiting the same vertices over and over



DAGs: A generalization of trees

Idea: Similar to DP

```
CalcDistance(i):
    if visited[i]: return Distance[i];
    visited[i] = true;
    Distance[i] = -INF; // No such vertex
    if i is red:
        Distance[i] = 0
    for j in Children[i]:
        Distance[i] = max(CalcDistance(j) + 1, Distance[i]);
    endfor
    return Distance[i];
```



Detecting Cycles in a Graph

```
FindCycle(i):
    if onStack[i]: return true; // If i is an ancestor of itself
    if visited[i]: return false;
    visited[i] = true;
    onStack[i] = true;  // all vertices process while onStack[i] are descendants of i
    for j in Children[i]:
        if FindCycle(j): return true;
    end for
    onStack[i] = false; // signal that vertices are no longer descendants of i
    return false;
```