Lecture 36: Graphs II

CS 62

Spring 2018

Alexandra Papoutsaki & William Devanny

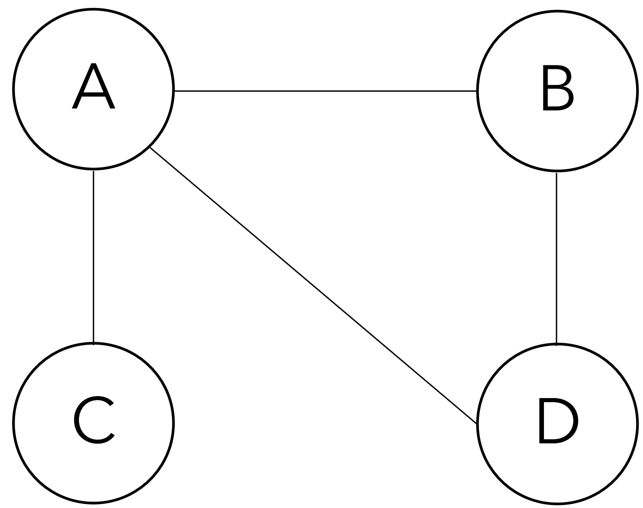
Number of Edges

- If $|V| = n$, then:
- minimum number of edges: 0
  - A graph can have only nodes
- For simple directed graphs, maximum number: $n(n-1)$
- For simple undirected graphs, maximum number: $\frac{n(n-1)}{2}$

- Dense graphs → #edges close to maximum
- Sparse graphs → #edges close to n

Graph Representations

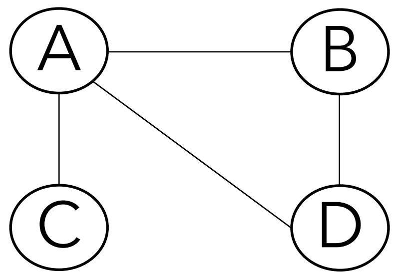
- Adjacency Matrix
- Adjacency List



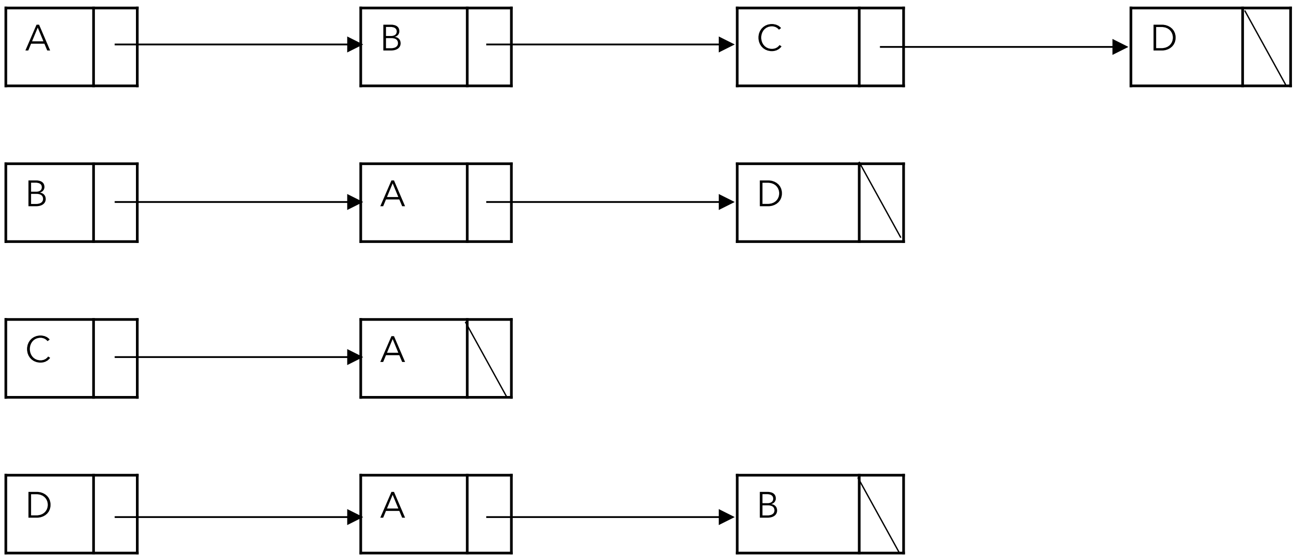
Adjacency Matrix

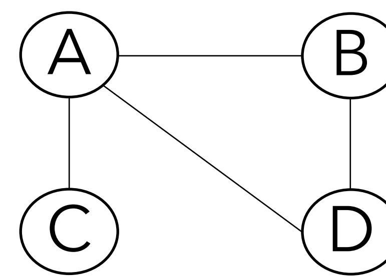
	A	**B**	**C**	**D**
A	0	1	1	1
B	1	0	0	1
C	1	0	0	0
D	1	1	0	0

- Good for dense graphs
- Constant time for lookup for edges.
- Constant time for adding/removing an edge
- Symmetric if undirected.
- Can hold weights.



Adjacency Lists





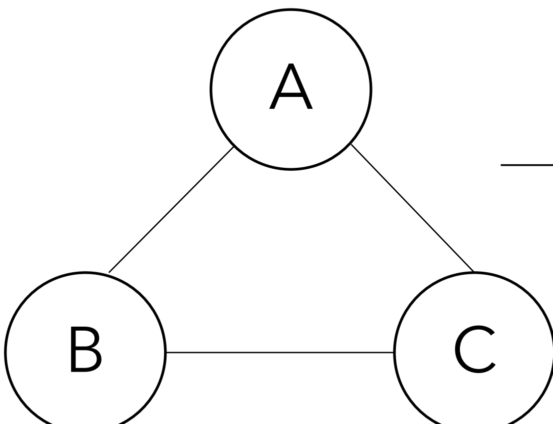
- Good for sparse graphs, saves space.
- Linear time lookup for edges.

Time complexity comparison

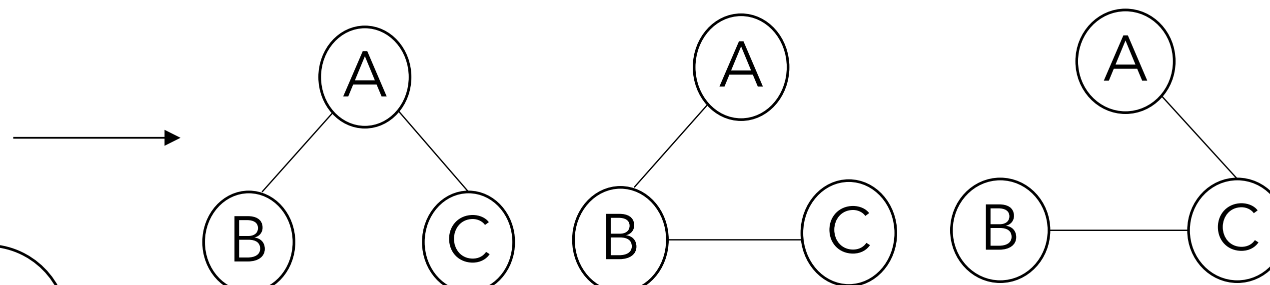
Operation	Adjacency Matrix	Adjacency List
Store graph	$O(\|V\|^2)$	$O(\|V\| + \|E\|)$
Add vertex	$O(\|V\|^2)$	$O(1)$
Add edge	$O(1)$	$O(1)$
Remove vertex	$O(\|V\|^2)$	$O(\|E\|)$
Remove edge	$O(1)$	$O(\|V\|)$
Are two vertices adjacent	$O(1)$	$O(\|V\|)$

Spanning Trees

- *Tree*: connected undirected graph with no cycles
- *Spanning tree of G*: includes every vertex of G and is a subgraph of G (every edge belongs to G)
- Can have properties like minimum-cost
- Can be constructed by search algorithms



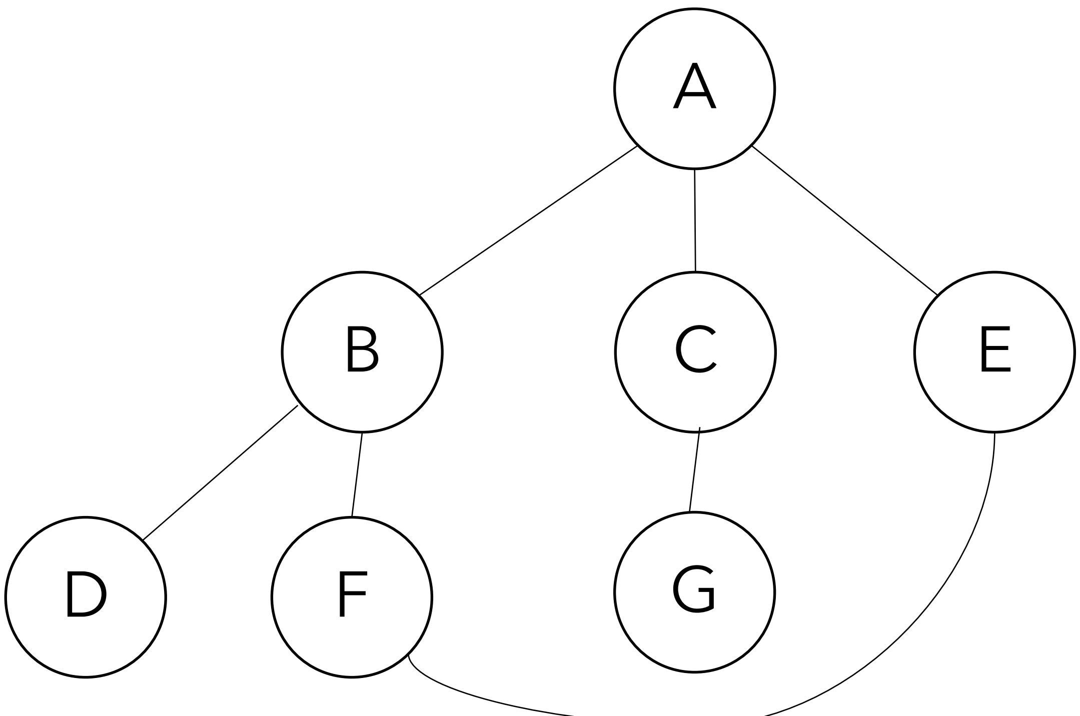
Graph G



3 different spanning trees of graph G

Depth-First Search

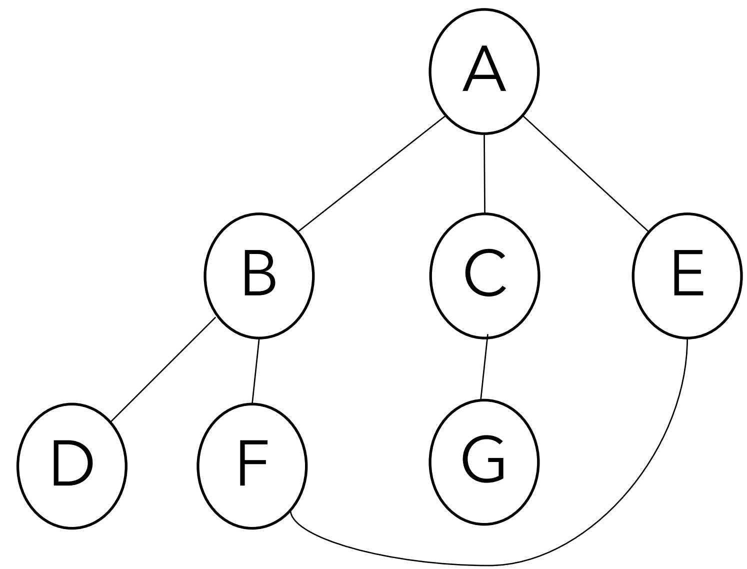
- Explore the graph without revisiting nodes
- Depth-first means go until you hit a dead end, then back up to branch out



Example from wikipedia

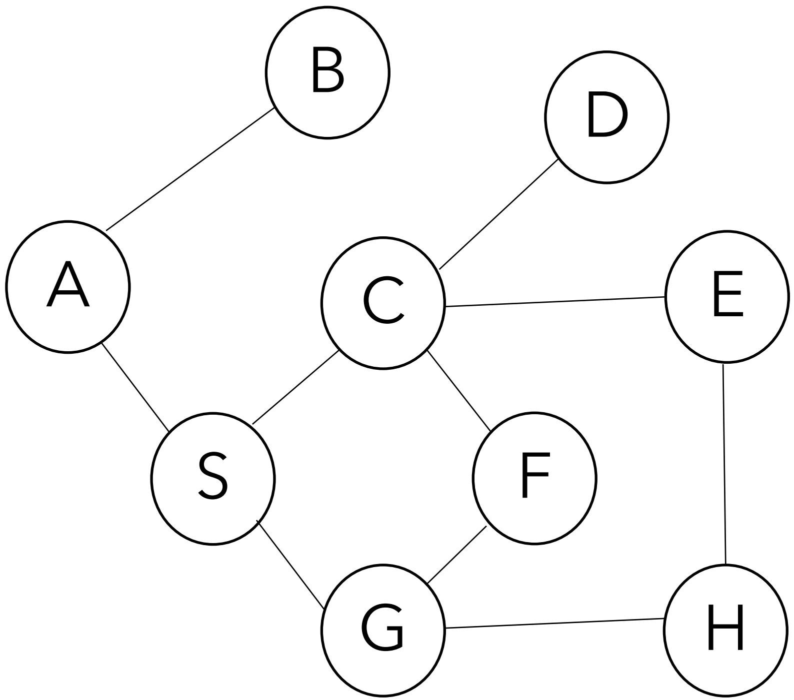
Recursive DFS pseudocode

```
DFS(G,v){
      visited[v] = true;
      for(every edge (v,w)){
            if(!visited[w]){
                  DFS(G,w);
            }
      }
}
```



Order of visit: A B D F E C G

Practice time

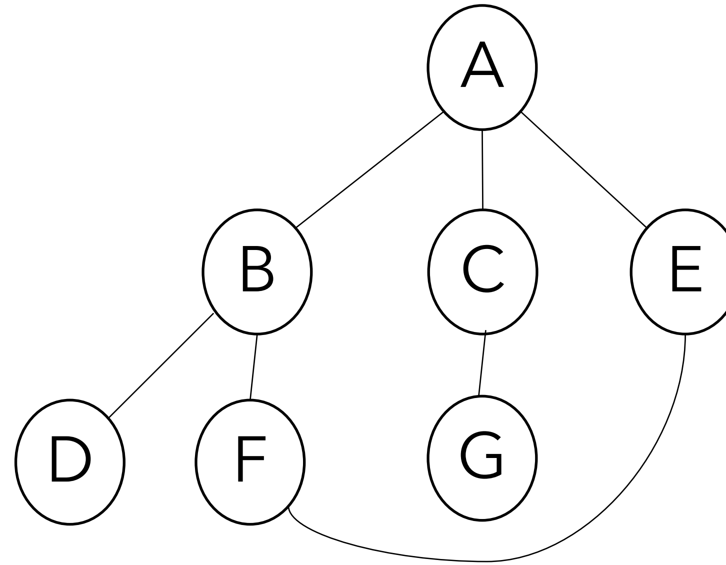


Order of visit: A B S C D E H G F

Example graph from: https://www.youtube.com/watch?v=iaBEKo5sM7w

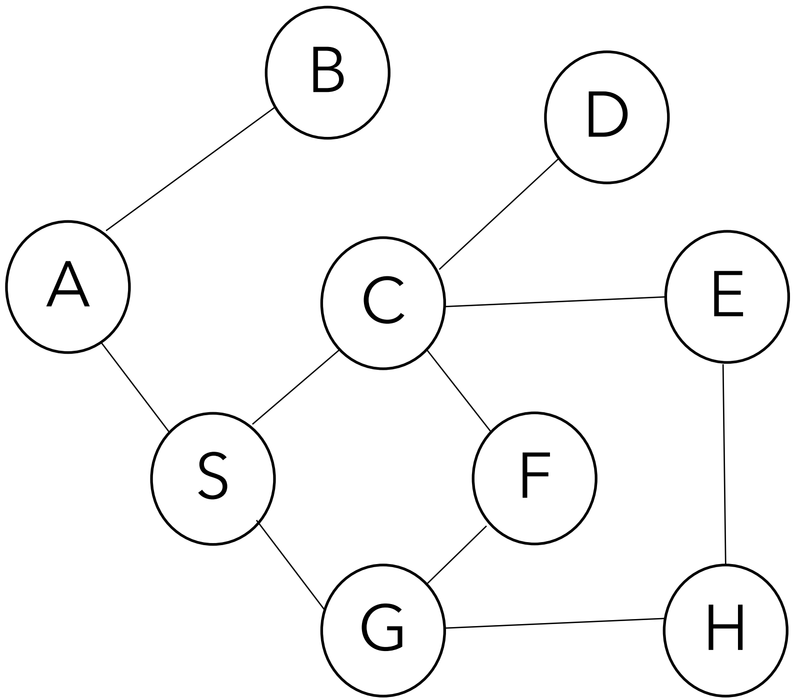
Non-recursive DFS pseudocode

```
for(every vertex v)
   visited[v]=false;
s=new Stack();
s.push(v1);
while(!s.isEmpty())
{
   v = s.pop();
   if (!visited[v])
   {
      visited[v] = true;
      for (every edge (v, w))
         if (!visited[w])
           s.push(w);
   }
}
```



Order of visit: A E F B D C G

Practice time



Order of visit: A S G H E C F D B

Example graph from: https://www.youtube.com/watch?v=iaBEKo5sM7w

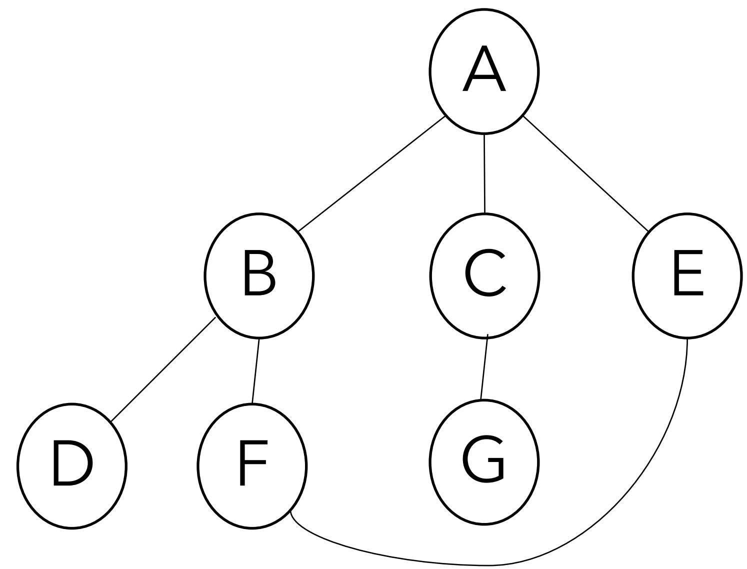
Breadth-First Search

- Replace stack with queue
- Now we explore in order of distance from start

- Algorithm:
  1. Mark start vertex
  2. Add all unmarked neighbors to queue and mark them
  3. Repeat step 2 with next from queue until it's empty

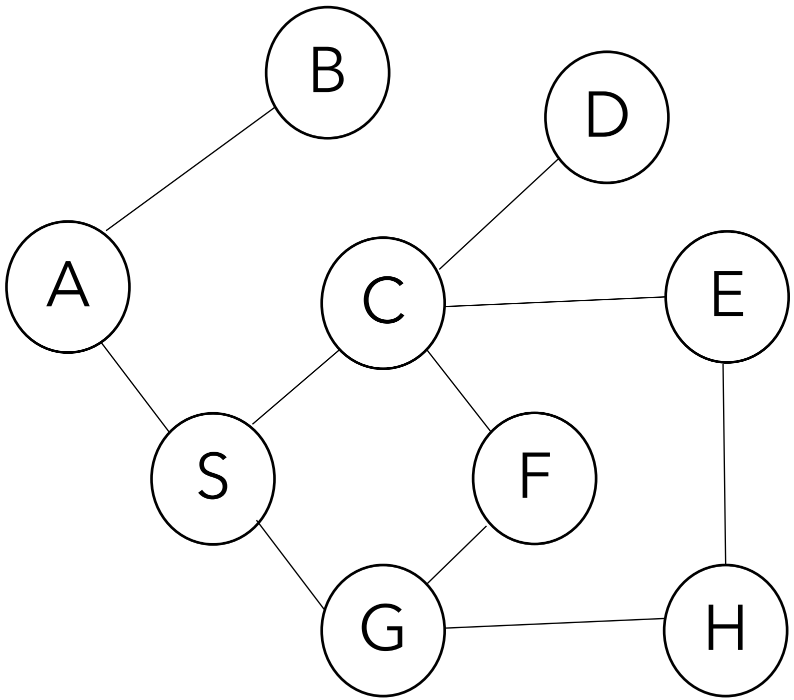
BFS pseudocode

```
for(every vertex v)
   visited[v]=false;
q=new Queue();
q.enqueue(v1);
while(!q.isEmpty())
{
   v = q.dequeue();
   if (!visited[v])
   {
      visited[v] = true;
      for (every edge (v, w))
         if (!visited[w])
           q.enqueue(w);
   }
}
```



Order of visit: A B C E D F G

Practice time



Order of visit: A B S C G D E F H

Example graph from: https://www.youtube.com/watch?v=iaBEKo5sM7w

DFS/BFS traversal

- Can be performed in $O(n+m)$, where $n = |V|, m = |E|$
- Can :
  - Test if G is connected
    - If traversal visited all vertices, then graph is connected
  - Compute a spanning tree of G, if G is connected
  - Find a path between two vertices, if it exits
  - Compute the connected components of G
    (needs to loop over all vertices and run DFS/BFS again)

Connectivity in Digraphs

- **reachable vertices**: when there is a directed path from one to another.
- **strongly connected vertices**: if mutually reachable
- **strongly connected digraph**: directed path from every vertex to every other vertex
- **weakly connected graph**: a digraph that would be connected if all of its directed edges were replaced by undirected edges.

Testing connectivity

- For an undirected graph:
  - Run DFS/BFS from any vertex without restarting and see if all vertic es are marked

- For strong connectivity on a directed graph:
  - 1. Initialize all vertices are not visited
  - 2. Run DFS/BFS from an arbitrary vertex v.
    - If traversal does not visit all vertices return false
  - 3. Reverse all edges
  - 4. Start from same vertex v and perform DFS/BFS. Graph is strongly connected iff all vertices are marked as visited again.