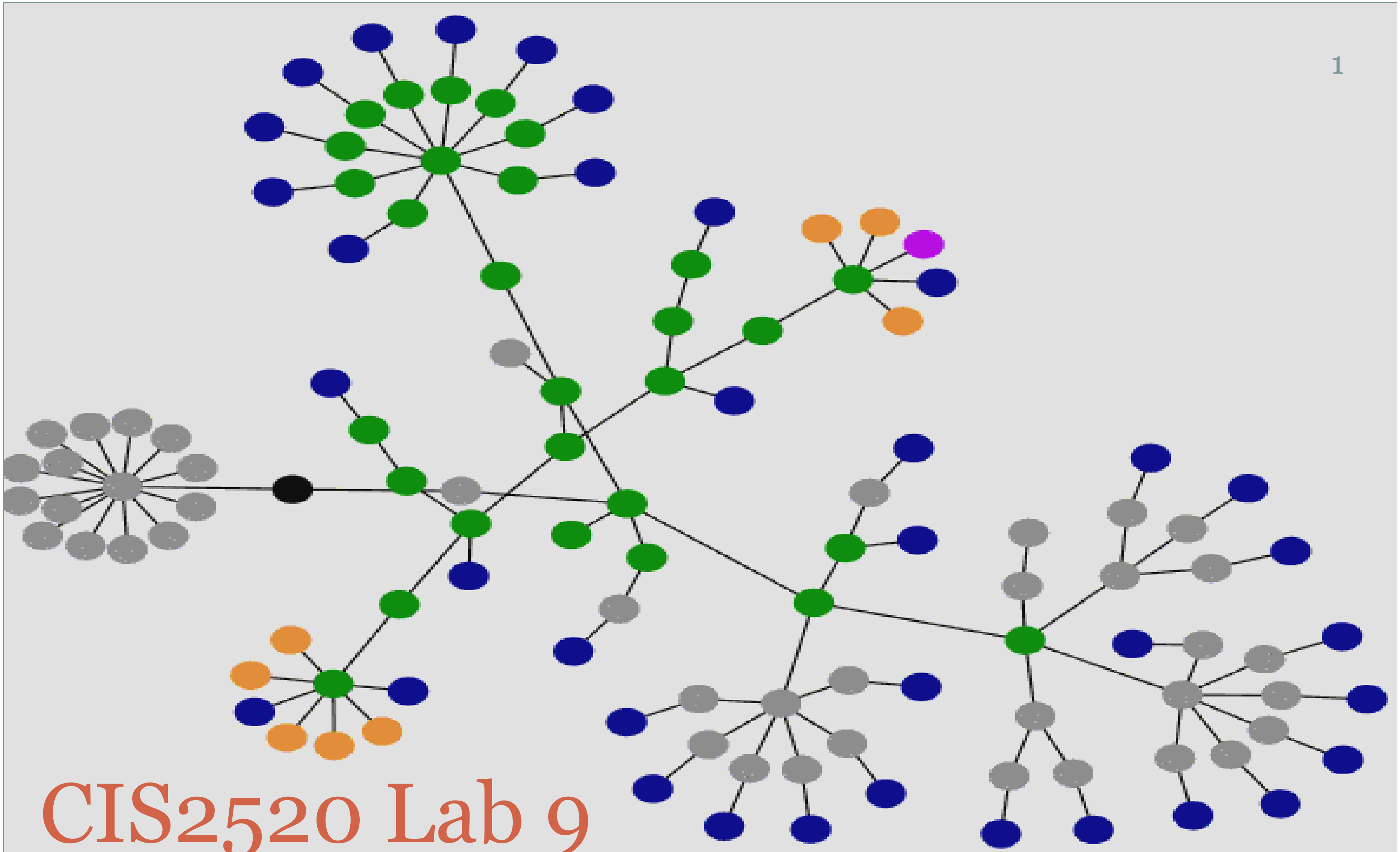


CIS2520 Lab 9

DATA STRUCTURE

Outline

- Graphs data structures
- Breadth First Seach
- Depth First Search

Graphs

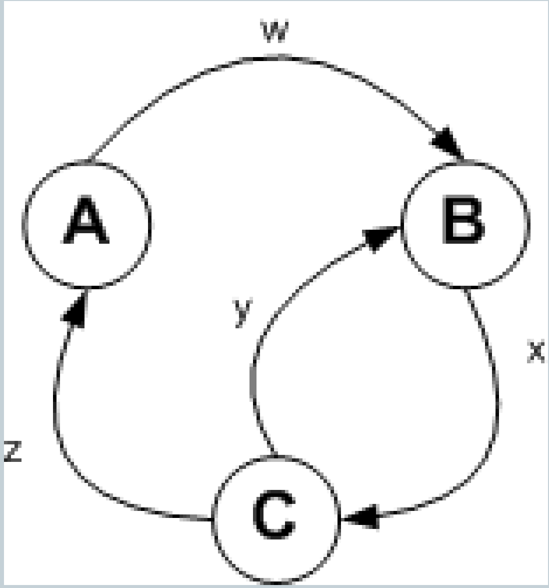
- Datastructures for graphs:
  - Adjacency lists: list of vertices, list of edges
  - Adjacency matrix: matrix of n vertices, edges are represented by values (1 or 0) on a n x n matrix

Adjacency lists

- An edge contains references to a beginning vertex and an ending vertex
  - i.e. pointers to two vertex structures
- A vertex contains references to a list of incident edges
  - i.e. for a directed graph, a vertex would contains pointers to a list of incoming edges and a list of outgoing edges

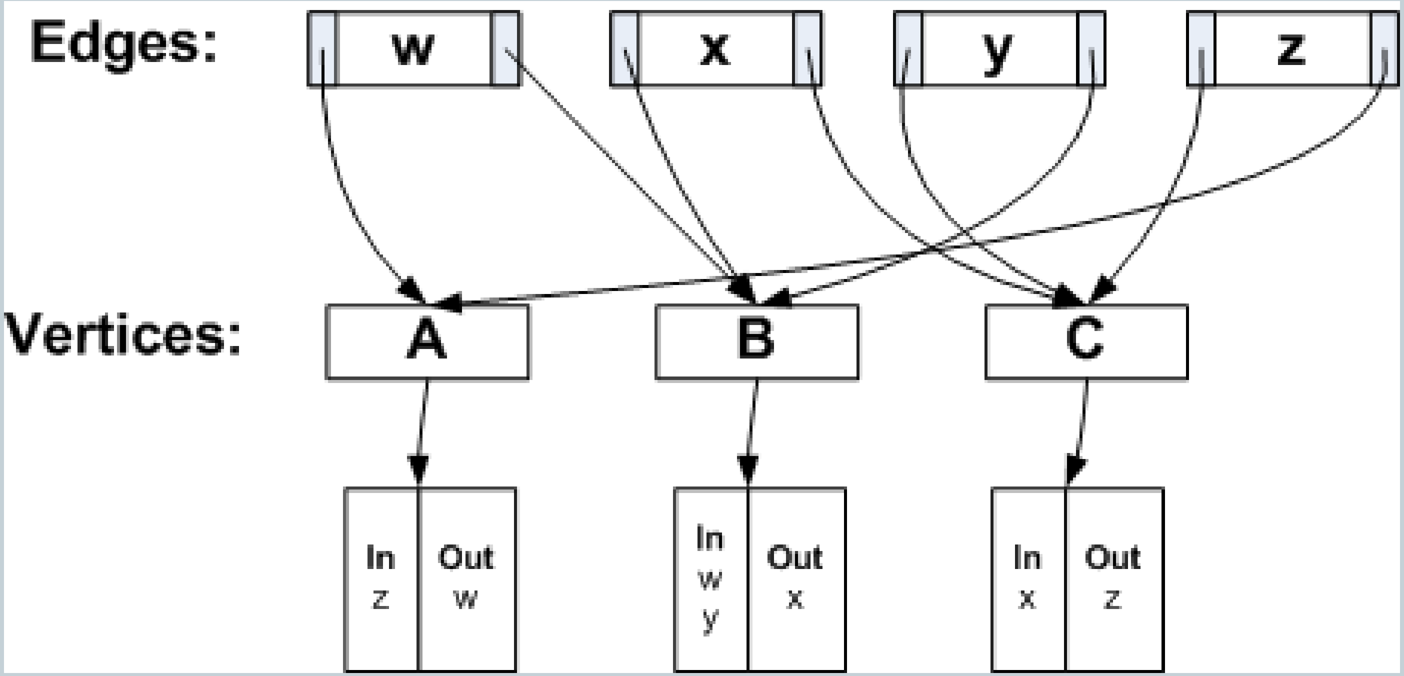
Adjacency lists

- Example: a digraph G with 3 vertices
  - Vertices: A, B, C
  - Edges: w, x, y, z



Adjacency list

- Structural view



Adjacency matrix

- For a graph G with n vertices
  - Need array A of pointers with with n x n size to store the information about edges

Adjacency matrix

		To		
		A	**B**	**C**
	A	null	w	null
From	**B**	null	null	x
	C	z	y	null

Performances

- In average cases (sparse edges): adjacency list has better time complexity, most noticeable:
  - insertVertex(v): O(1) vs O(n^2)
  - removeVertex(v): O(deg(v)) vs O(n^2)

- Adjacency matrix contains many important mathematical properties:
  - Eg. An undirected graph would result in a symmetrical matrix

Depth First Search

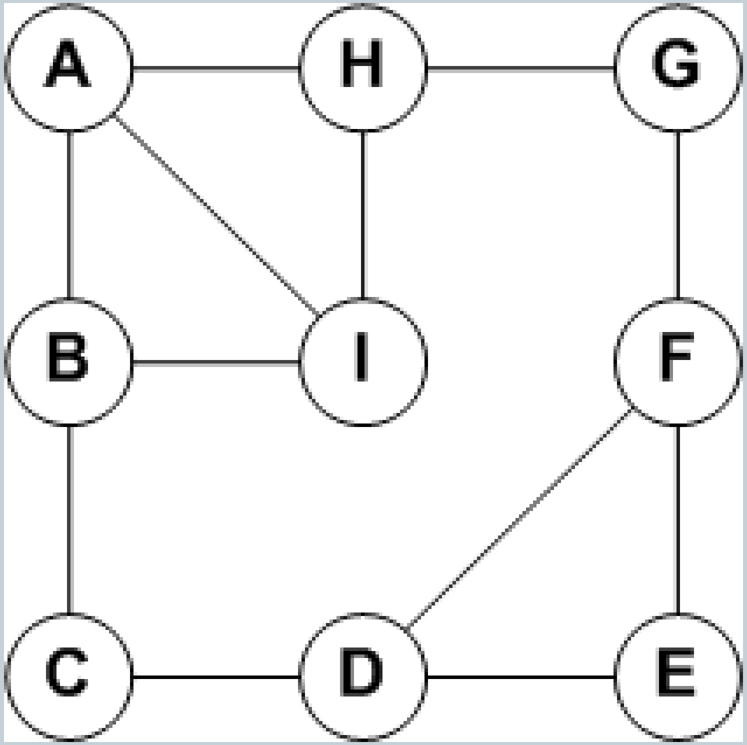
- Search away from the starting position first.

Input: a vertex v in the graph
Output: a labeling of the edges as “discovery” edges and “back edges”

DFS(v):
    for each edge e incident to v **do**:
        if edge e is unexplored **then**
            let w be the other endpoint of e
            if vertex w is unexplored **then**
                label e as a discovery edge
                recursively call DFS(w)
            else
                label e as a back edge
end

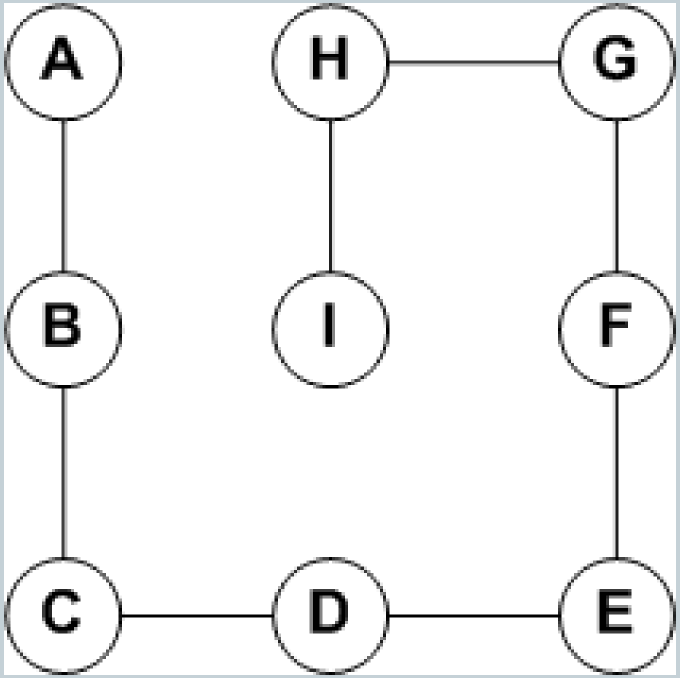
Depth First Search

- Eg. A, B, C, D, E, F, G, H, I



Depth First Search

- Spanning forest created from DFS



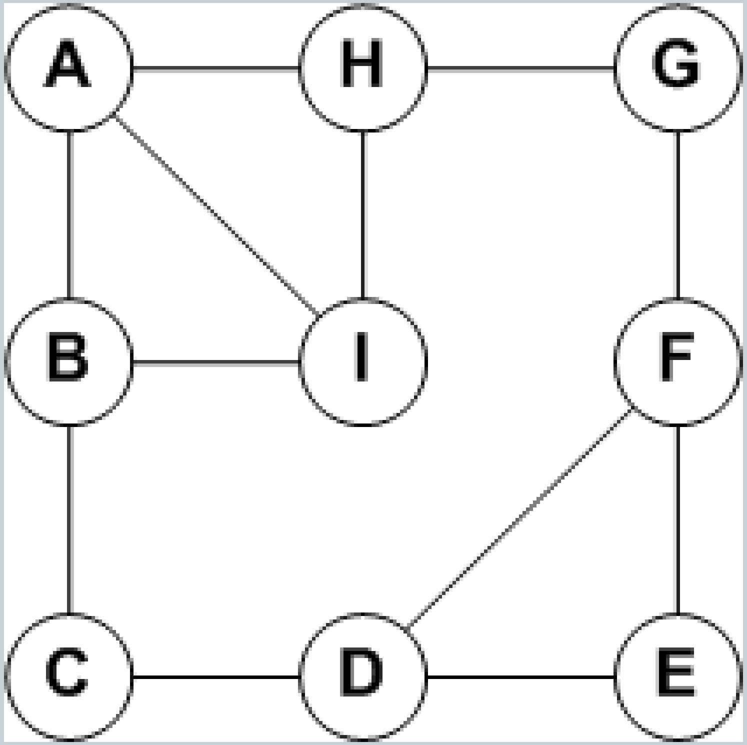
Breadth First Search

- Search around the starting position first.

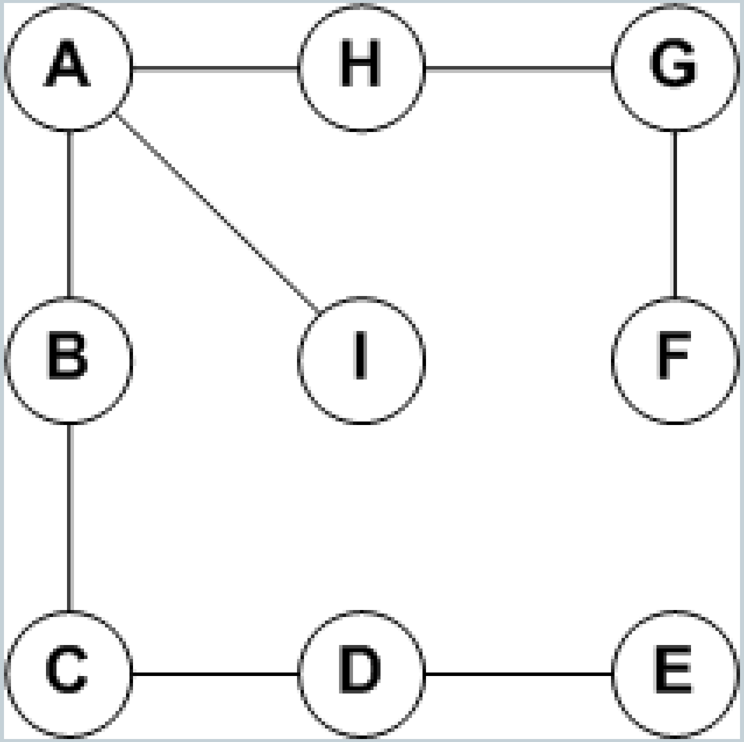
```
Function BFS(s):
    initialize container L0 to contain vertex s
    i = 0
    while Li is not empty do
        create container Li+1 to initially be empty
        for each vertex v in Li do
            for each edge e incident on v do
                if edge e is unexplored then
                    let w be the other endpoint of e
                    if vertex w is unexplored then
                        label e as a discovery edge
                        insert w into Li+1
                    else
                        label e as a cross edge
        i = i +1
End function
```

Breadth First Search

- Eg. A, B, I, H, C, G, D, F, E



Breadth First Search



Dijkstra’s Algorithm for Shortest Path

- Single source
- Calculate the shortest paths to all vertices from a single starting position
- Based on Greedy method
  - Expand based on the one that creates the best solution

Dijkstra’s algorithm

ShortestPath(G,v):
Input: a weighted graph G and a vertex v in G
Output: An array label D[u], for each vertex u of G, such that D[u] is the shortest path from v to u in G

```
initialize D[v] = 0 and D[u] = +inf for each vertex u ≠ v
let a priority queue Q contain all vertices of G using the D labels as keys
while Q is not empty
    u = removeMinElement(Q)
    for each vertex z adjacent to u such that z is in Q do
        if D[u] + w((u,z)) < D[z] then
            D[z] = D[u] + w((u,z))
            change to D[z] the key value of z in Q
return the label D[u] of each vertex u
```

Dijkstra's algorithm