Welcome back!

Last Time

Breadth First
- Graph Traversal

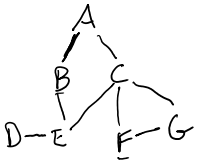
Today

- Breadth First Search
- Depth First
  - Traverse
  search

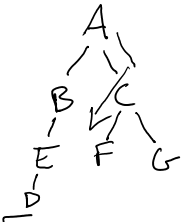
Recap:

what is breadth First?

Given some graph:



BF "look"



Find path from A to F.

Not following breadth first could find path:

A → B → E → C → G → F

Redraw as tree where we only add child if it doesn't appear at an equal or higher level in tree. A → C → F is guaranteed

Goal of BFT: traverse (and print) entire graph in a BF order

Goal of BFS: Find a vertex via shortest path

Differences b/w BFT and BFS:

- BFS takes in Z values
- add "distance" numeric value to vertex struct
- Increment distance of every vertex as each "layer" is traversed
- return vertex if/when found
  - ↳ distance from starting vertex will be encoded in the returned vertex

update the vertex struct:

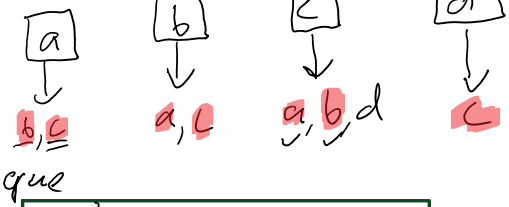
```
struct vertex
{
   string key;
   vector <adjVertex>;
   bool visited;
   int distance;
}
```

BFS Algorithm: unweighted - only

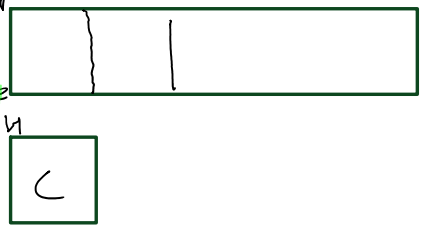
```
vertex BreadthFirstSearch( startValue, searchValue )
{
   vertex = search (startValue);
   vertex.visited = T;
   vertex.distance = 0;
   que.enque (vertex);
   while(!que.isEmpty())
   {
      n= que.deque();
      for x= 0 to n.adj.end
         if (!n.adj[x].v.visited)
            n.adj[x].v.distance =n.distance
                                 +1
         if(n.adj[x].v.key == searchValue)
            return n.adj[x].v
         else
            n.adj [x].v.visited = T;v
            que.enque (n.adj[x].v) v
   }
```

e.g. BFS(a, d)

distance	
a	0
b	1
c	1
d	2



que

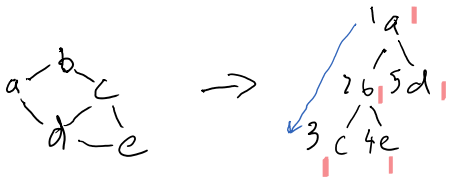


result -> vertex, key = d
.distance = 2

Depth First

Instead of evaluating all the vertices in the adjacent "layer" before moving onto next layer, with DF each branch is pursued until furthest vertex is encountered.

e.g. DFT(a)



Two possible implementations:
- recursive
- non-recursive (iterative)

recursive algorithm: DF traverse

```
DFT(vertex)
    vertex.visited = T;
    for x=0 to vertex.adjacent.end
        if(!vertex.adjacent[x].v.visited)
            print(vertex.adjacent[x].v.key
            DFT(vertex.adjacent[x].v)
```

(1)

```
depthFirstTrav(value)
    vertex = search(value)
    print(vertex.key)
    DFT(vertex)
```