CSCI 136
Data Structures &
Advanced Programming

Shortest Paths in Unweighted Graphs

(BFS)

Finding Shortest Paths (Edge Count)

Recall: Distance from u to v in an undirected graph G is the *number of edges* in (any) *minimum length path* between u and v

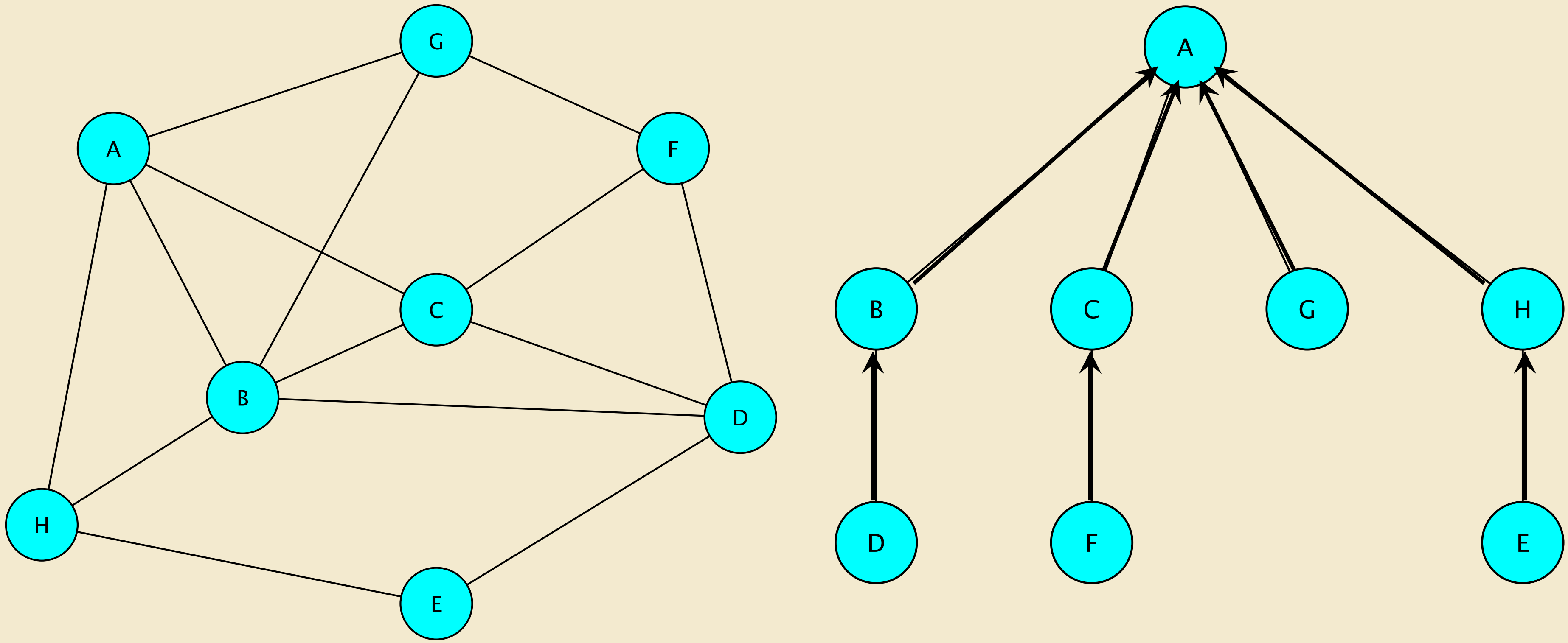
Goal: Find distance between *every pair* of vertices

Assumption: G is connected

Idea: For each vertex v in G, build a BFS tree from v

- This tree will contain, for each u ≠ v, a shortest path from v to u

The BFS Tree of Shortest Paths From A



For each vertex u≠A, store (u, parent(u))

How: Use a Map<V,V> : The *Routing Table* for A

Storing The Path Information

Given a BFS tree some vertex v

- For each vertex u ≠ v, store the pair (u, parent(u))
- From these pairs we can build the path from v to u
  - By starting with u and working backwards
- Store the pairs in a Map<V,V> : the routingTable for v
- Store the Routing Tables in a Map<V, Map<V,V>>!
  - Entries are (v, routingTable(v))
- To find path from v to u
  - Get routing table for v from Routing Tables Map
  - Look up u in the routing table for v
  - Follow parents back from u to v

Finding Shortest Paths (Edge Count)

BuildRoutingTables(G) : Map of routing Tables

    Create an empty Map routingTables of Maps
    for each vertex v in G
        build routing table for v
        add the routing table for v to routingTables

GetShortestPath(RoutingTables, v, u) : List of vertices on path

    routingTable= routing table of v from RoutingTables
    if u isn't in the routing table for v return null   // u and v in different components!
    let path be an empty list
    add u to path
    while(u ≠ v)
        u = routingTable.get(u)   // u becomes u's parent
        add u to beginning of path
    return path

Building a Routing Table

```
Map BuildRoutingTable(G, v)   // Using BFS of G starting at v
Create empty map routingTable to hold BFS tree from v
Create empty queue Q; enqueue v; mark v as visited;
Add (v,v) to routingTable // v will have itself as predecessor
While Q isn't empty
        current ← Q.dequeue()
        for each  unvisited neighbor u  of current
                add (u,current) to routingTable
                add u to Q; mark u as visited
return routingTable;
```

```
Map singleSourcePaths(G, v)
 Create empty map routingTable
Create empty queue Q;
enqueue v
mark v as visited;
Add (v,v) to routingTable
While Q isn't empty
   cur ← Q.dequeue()
   for each unvisited
            neighbor u of cur
      add (u,cur) to routingTable
      add u to Q
      mark u as visited
return routingTable;
```

```
public static <V,E> Map<V,V>
SSSP(Graph<V,E> g, V src) {
  Map<V,V> routingTable =
      new Hashtable<V,V>();
  Queue<V> todo = new QueueList<V>();
  todo.enqueue(src);
  g.visit(src);
  routingTable.put(src,src);
  while (!todo.isEmpty()) {
    V node = todo.dequeue();
    AbstractIterator<V> neighbors =
      (AbstractIterator<V>)
        g.neighbors(node);
    while (neighbors.hasNext()) {
      V next = neighbors.next();
      if (!g.isVisited(next)) {
        routingTable.put(next,node);
        todo.enqueue(next);
        g.visit(next);
  }}}
 return routingTable;}
```

```java
public static <V,E> Map<V,V> SSSP(Graph<V,E> g, V src) {

  Map<V,V> routingTable = new Hashtable<V,V>();
  Queue<V> todo = new QueueList<V>();
  todo.enqueue(src);
  g.visit(src);
  routingTable.put(src,src);

  while (!todo.isEmpty()) {
    V node = todo.dequeue();
    AbstractIterator<V> neighbors = AbstractIterator<V>)
          g.neighbors(node);
    while (neighbors.hasNext()) {
      V next = neighbors.next();
      if (!g.isVisited(next)) {
        routingTable.put(next,node);
        todo.enqueue(next);
        g.visit(next);
  }}}
 return routingTable;}
```

Finding Shortest Paths: Complexity

Using GraphListUndirected implementation

- singleSourcePaths(G,v) visits exactly those vertices and edges reachable from v
  - It does not visit any other vertex or edge
- So, finding i^{th} map $M_i=(V_i,E_i)$ takes time $O(|V_i|+|E_i|)$
- Worst Case: G is connected: $|V_i|= |V|$, $|E_i|=|E|$
- Run time: $O(|V| \cdot (|V| + |E|))=O(|V|^2+|V||E|)$
  - Could be $O(|V|^3)$

Summary & Observations

Using GraphListUndirected implementation

- Can compute shortest path information for all pairs of vertices
  - In $O(|V|^2+|V||E|)$ time and $O(|V|^2)$ space
    - Really $O(|V|^2+|E|)$ space, but $|E|$ is $O(|V|^2)$
- A path can be computed from the tables in time proportional to the length of the path
  - Assuming $O(1)$ lookup times for Maps
- Up next: Shortest paths using edge weights!