INFO 6205
Program Structure and Algorithms

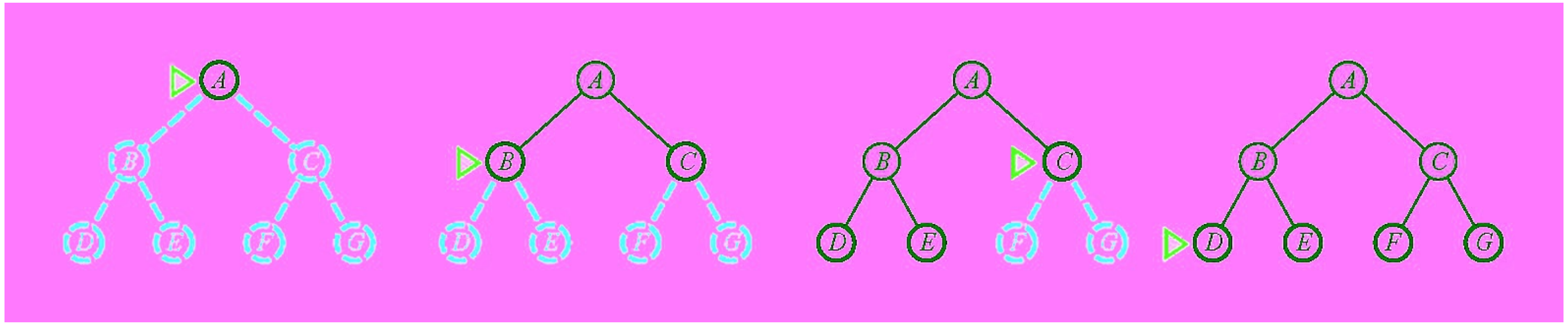
Breadth-First Search

Nik Bear Brown

Topics

- Breadth-First Search

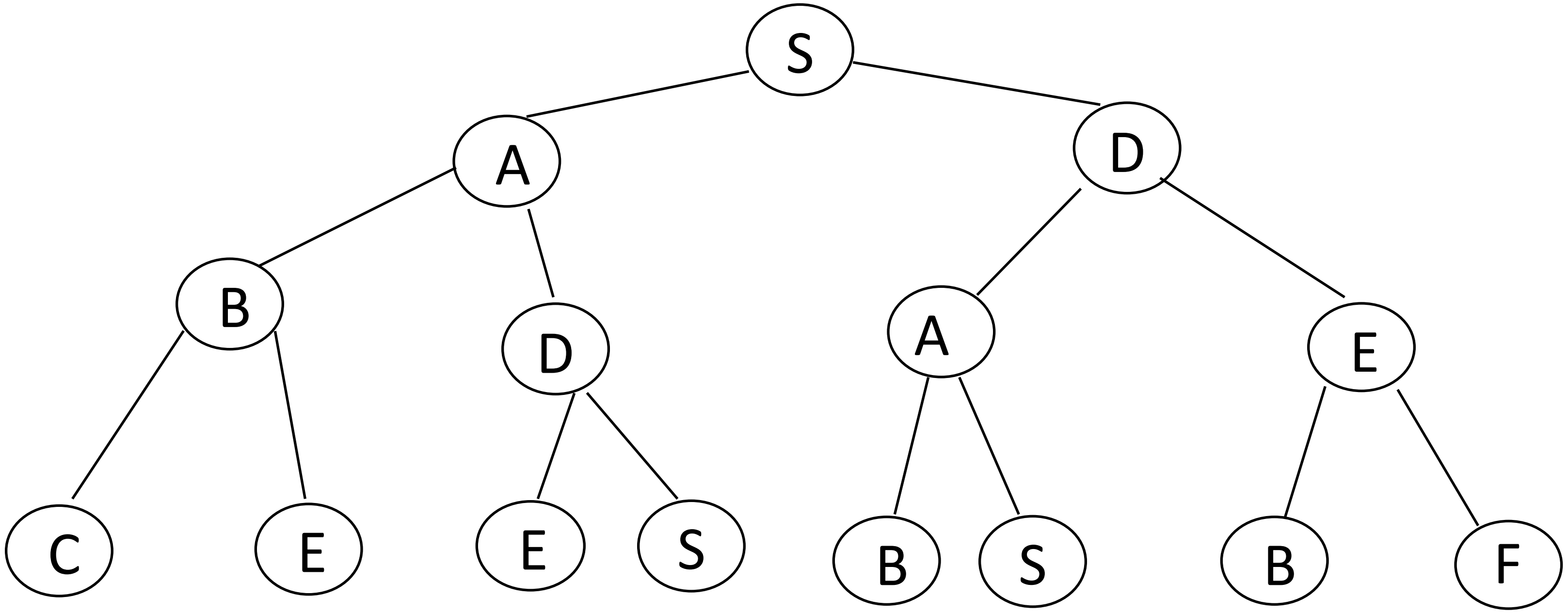
Breadth-First Search



Pseudocode for Breadth-First Search

```
Initialize: Let Q = {S}
While Q is not empty
        pull Q1, the first element in Q
        if Q1 is a goal
                report(success) and quit
        else
                child_nodes = expand(Q1)
                eliminate child_nodes which represent loops
                put remaining child_nodes at the back of Q
        end
Continue
```

Breadth First Search

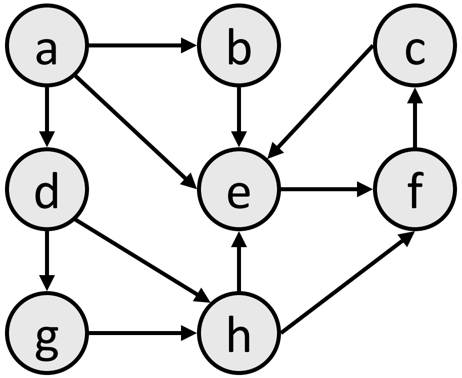


(Use the simple heuristic of not generating a child node if that node is a parent to avoid "obvious" loops: this clearly does not avoid all loops and there are other ways to do this)

Breadth-first search

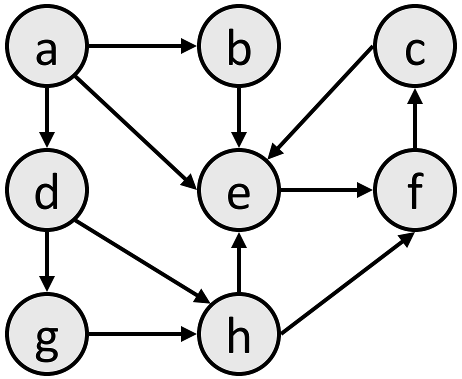
- **breadth-first search** (BFS): Finds a path between two nodes by taking one step down all paths and then immediately backtracking.
  - Often implemented by maintaining a queue of vertices to visit.

- BFS always returns the shortest path (the one with the fewest edges) between the start and the end vertices.
  - to b: {a, b}
  - to c: **{a, e, f, c}**
  - to d: {a, d}
  - to e: **{a, e}**
  - to f: **{a, e, f}**
  - to g: {a, d, g}
  - to h: **{a, d, h}**



BFS pseudocode

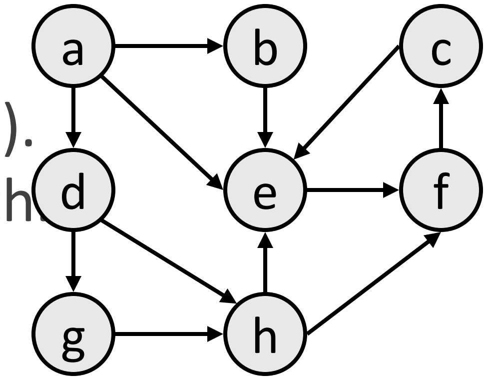
```
function bfs(v1, v2):
    queue := {v1}.
    mark v1 as visited.

    while queue is not empty:
        v := queue.removeFirst().
        if v is v2:
            a path is found!

        for each unvisited neighbor n of v:
            mark n as visited.
            queue.addLast(n).

    // path is not found.
```



- Trace bfs(a, f) in the above graph.

BFS observations

- *optimality*:
  - always finds the shortest path (fewest edges).
  - in unweighted graphs, finds optimal cost path.
  - In weighted graphs, *not* always optimal cost.



- *retrieval*: harder to reconstruct the actual sequence of vertices or edges in the path once you find it
  - conceptually, BFS is exploring many possible paths in parallel, so it's not easy to store a path array/list in progress
  - solution: We can keep track of the path by storing predecessors for each vertex (each vertex can store a reference to a *previous* vertex).

- DFS uses less memory than BFS, easier to reconstruct the path once found; but DFS does not always find shortest path. BFS does.

BFS runtime

- What is the expected runtime of DFS in terms of the number of vertices V and the number of edges E ?

- Answer: O(|V| + |E|)
  - where |V| = number of vertices, |E| = number of edges
  - Must potentially visit every node and/or examine every edge once.

  - why not O(|V| * |E|) ?

- What is the space complexity of each algorithm?
  - (How much memory does each algorithm require?)