Depth-first search

- **depth-first search (DFS):** finds a path between two vertices by exploring each possible path as many steps as possible before backtracking
- Often implemented recursively
DFS example

- All DFS paths from A to others (assumes alphabetical edge order)
  - A
  - A → B
  - A → B → D
  - A → B → F
  - A → B → F → E
  - A → C
  - A → C → G

- What are the paths that DFS did not find?
DFS pseudocode

- Pseudo-code for depth-first search:
  - dfs(v1, v2):
    - dfs(v1, v2, {})
  - dfs(v1, v2, path):
    - path += v1
    - mark v1 as visited.
    - if v1 is v2:
      - path is found.
    - for each unvisited neighbor v_i of v1 where there is an edge from v1 to v_i:
      - if dfs(v_i, v2, path) finds a path, path is found.
  - path -= v1.  path is not found.
VertexInfo class

```java
public class VertexInfo<V> {
    public V v;
    public boolean visited;

    public VertexInfo(V v) {
        this.v = v;
        clear();
    }

    public void clear() {
        visited = false;
    }
}
```
DFS observations

- Guaranteed to find a path if one exists

- Easy to retrieve exactly what the path is (to remember the sequence of edges taken) if we find it

- **Optimality**: Not optimal. DFS is guaranteed to find a path, not necessarily the best/shortest path
  - Example: DFS(A, E) may return A → B → F → E
Another DFS example

- Using DFS, find a path from BOS to LAX.
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 list or queue of vertices to visit.
  - BFS always returns the path with the fewest edges between the start and the goal vertices.
BFS example

- All BFS paths from A to others (assumes alphabetical edge order)
  - A
  - A → B
  - A → C
  - A → E
  - A → B → D
  - A → B → F
  - A → C → G

- What are the paths that BFS did not find?
**BFS pseudocode**

- Pseudo-code for breadth-first search:
  
  ```pseudo```
  ```
  bfs(v1, v2):
      List := {v1}
      mark v1 as visited.
      
      while List not empty:
          v := List.removeFirst()
          if v is v2:
              path is found.
          for each unvisited neighbor v_i of v
              where there is an edge from v to v_i:
                  mark v_i as visited
                  List.addLast(v_i).
      
  path is not found.
  ```
  ```

```
BFS observations

- **Optimality**: 
  - In unweighted graphs, optimal. (fewest edges = best)
  - In weighted graphs, not optimal. (path with fewest edges might not have the lowest weight)

- **Disadvantage**: Harder to reconstruct what the actual path is 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

- **Observation**: Any particular vertex is only part of one partial path at a time
  - We can keep track of the path by storing predecessors for each vertex (references to the previous vertex in that path)
Another BFS example

- Using BFS, find a path from BOS to LAX.
DFS, BFS runtime

- In terms of the number of vertices $|V|$ and the number of edges $|E|$:  
  - What is the expected runtime of DFS?  
  - What is the expected runtime of BFS?

  **Answer:** $O(|V| + |E|)$
  - Each algorithm must potentially visit every node and/or examine every edge once.

- What is the space complexity of each algorithm?
  - $O(|V|)$