Where did David leave that book?
Where did David leave that book?
Where did David leave that book?

Go as far down a path as possible before backtracking – Depth-First Search
Graph Algorithms

• **Search**
  – Depth-first search
  – Breadth-first search

• **Shortest paths**
  – Dijkstra's algorithm

• **Minimum spanning trees**
  – Prim's algorithm
  – Kruskal's algorithm
Reachability

Node $v$ is reachable from node $u$ if there is a path from $u$ to $v$.

Which nodes are reachable from node 1?
Reachability

Node $v$ is reachable from node $u$ if there is a path from $u$ to $v$.

Which nodes are reachable from node 1?
0, 1, 2, 3, 5
Reachability

Node $v$ is reachable from node $u$ if there is a path from $u$ to $v$.

Which nodes are reachable from node 4?
Reachability

Node $v$ is reachable from node $u$ if there is a path from $u$ to $v$.

Which nodes are reachable from node 4?
3, 4, 5, 6
Reachability

How to determine reachability efficiently?

We need an invariant!
Reachability

Node $v$ is reachable from node $u$ without green nodes if there is a path from $u$ to $v$ without green nodes.

Which nodes are reachable from node 1 without green nodes?
Reachability

Node $v$ is reachable from node $u$ without green nodes if there is a path from $u$ to $v$ without green nodes.

Which nodes are reachable from node 1 without green nodes?

1
Reachability

Node $v$ is reachable from node $u$ without green nodes if there is a path from $u$ to $v$ without green nodes.

Which nodes are reachable from node 4 without green nodes?
Reachability

Node $v$ is reachable from node $u$ without green nodes if there is a path from $u$ to $v$ without green nodes.

Which nodes are reachable from node 4 without green nodes? None!

Node 4 is green, so all paths from node 4 contain a green node!
Depth-First Search

- Keep pushing the search forward
- Mark nodes as “visited” (green) as you go
- Backtrack only when you can’t go any further

Which nodes are reachable from node 1?
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• Start at node 1
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• Extend path to some child
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• Extend path to some child
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• No new way to extend path, so backtrack
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• Extend path to a different child
Depth-First Search

- Keep pushing the search forward
- Mark nodes as “visited” (green) as you go
- Backtrack only when you can’t go any further
- Extend path to some child

Which nodes are reachable from node 1?
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• Already visited, so backtrack
Depth-First Search

- Keep pushing the search forward
- Mark nodes as “visited” (green) as you go
- Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

- No new way to extend path, so backtrack
Depth-First Search

- Keep pushing the search forward
- Mark nodes as “visited” (green) as you go
- Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

- No new way to extend path, so backtrack
Depth-First Search

- Keep pushing the search forward
- Mark nodes as “visited” (green) as you go
- Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

- Extend path to a different child
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• Extend path to some child
Depth-First Search

- Keep pushing the search forward
- Mark nodes as “visited” (green) as you go
- Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

- Already visited, so backtrack
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• No new way to extend path, so backtrack
Depth-First Search

• Keep pushing the search forward
• Mark nodes as “visited” (green) as you go
• Backtrack only when you can’t go any further

Which nodes are reachable from node 1?

• Nothing to backtrack, so all done!
Depth-First Search using Recursion

/** Visit all nodes reachable from u without visited nodes */
void dfs(Node u) {
    if (u.hasBeenVisited()) return;
}

Which nodes are reachable from node 4 without green nodes? None!
Depth-First Search using Recursion

/** Visit all nodes reachable from u without visited nodes */
void dfs(Node u) {
    if (u.hasBeenVisited()) return;
}


Depth-First Search using Recursion

/** Visit all nodes reachable from u without visited nodes */
void dfs(Node u) {
    if (u.hasBeenVisited()) return;
    u.visit();
    for (Node v with edge from u to v) dfs(v);
}

1 2 3
0 5 4
6
Depth-First Search using Recursion

/** Visit all nodes reachable from u without visited nodes */
void dfs(Node u) {
    if (u.hasBeenVisited()) return;
    u.visit();
    for (Node v with edge from u to v) dfs(v);
}

33
Depth-First Search using Recursion

/** Visit all nodes reachable from u without visited nodes */
void dfs(Node u) {
    if (u.hasBeenVisited()) return;
    u.visit();
    for (Node v with edge from u to v) dfs(v);
}
class Node {
    final List<Node> targets; // edges go from this to targets
    boolean visited = false; // has this node been visited?

    Node(Node... targets) {
        this.targets = Arrays.asList(targets);
    }

    /*Visit all nodes reachable from this without visited nodes*/
    void dfs() {
        if (visited) return;
        visited = true;
        for (Node v : targets) v.dfs();
    }
}
Depth-First Search using Iteration

/** Visit all nodes reachable from u without visited nodes */
void dfs(Node u) {
    Collection<Node> work= new Stack<Node>();
    work.add(u);
    // inv: all nodes that have to be visited are
    // reachable (without visited nodes) from some node in work
    while (!work.isEmpty()) {
        Node u= work.pop();  // Remove first node and put it in u
        if (!u.hasBeenVisited()) {
            u.visit();
            for (Node v with edge from u to v)
                work.add(v); // Stack adds nodes to front
        }
    }
}
Breadth-First Search

• Mark closest nodes as “visited” (green) first
• Then push search out further

Which nodes are reachable from node 1?
Breadth-First Search

• Mark closest nodes as “visited” (green) first
• Then push search out further

Which nodes are reachable from node 1?

• Visit nodes distance 0 from node 1
Breadth-First Search

- Mark closest nodes as “visited” (green) first
- Then push search out further

Which nodes are reachable from node 1?

- Visit nodes distance 1 from node 1
Breadth-First Search

- Mark closest nodes as “visited” (green) first
- Then push search out further

Which nodes are reachable from node 1?

- Visit nodes distance 2 from node 1
Breadth-First Search

• Mark closest nodes as “visited” (green) first
• Then push search out further

Which nodes are reachable from node 1?

• No nodes at distance 3, so all done!
Depth-First Search using Iteration

/** Visit all nodes reachable from u without visited nodes */
void dfs(Node u) {
    Collection<Node> work= new Stack<Node>();
    work.add(u);
    // inv: all nodes that have to be visited are
    // reachable (without visited nodes) from some node in work
    while (!work.isEmpty()) {
        Node u= work.pop();  // Remove first node and put it in u
        if (!u.hasBeenVisited()) {
            u.visit();
            for (Node v with edge from u to v)
                work.add(v); // Stack adds nodes to front
        }
    }
}
Breadth-First Search using Iteration

/** Visit all nodes reachable from u without visited nodes */
void bfs(Node u) {
    Collection<Node> work= new Queue<Node>();
    work.add(u);
    // inv: all nodes that have to be visited are
    //     reachable (without visited nodes) from some node in work
    while (!work.isEmpty()) {
        Node u= work.pop();  // Remove first node and put it in u
        if (!u.hasBeenVisited()) {
            u.visit();
            for (Node v with edge from u to v)
                work.add(v);  // Queue adds nodes to back
        }
    }
}