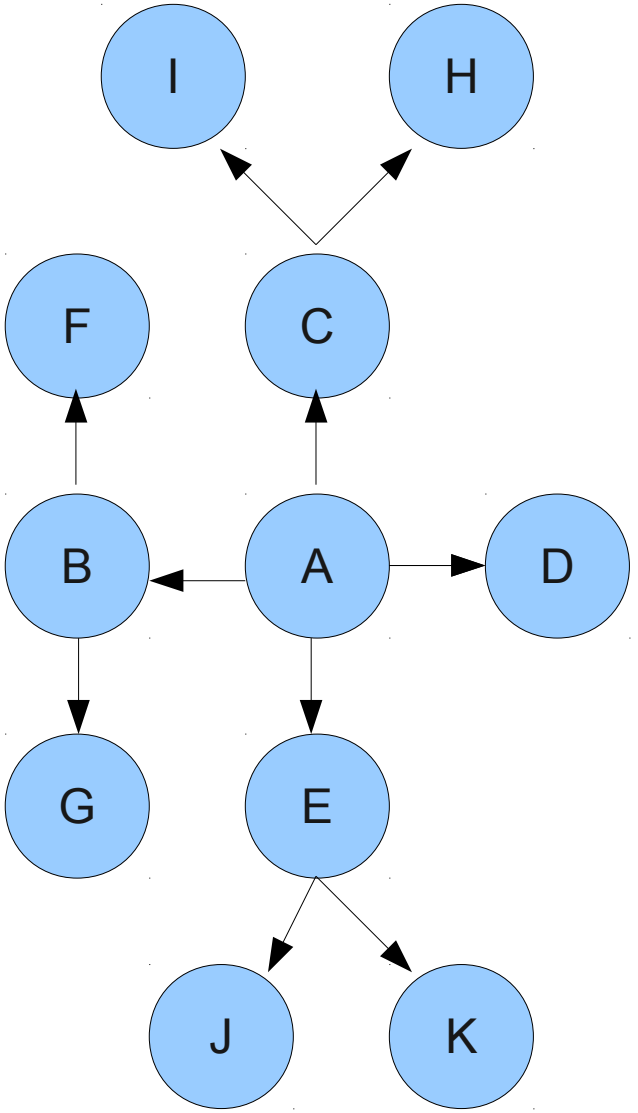
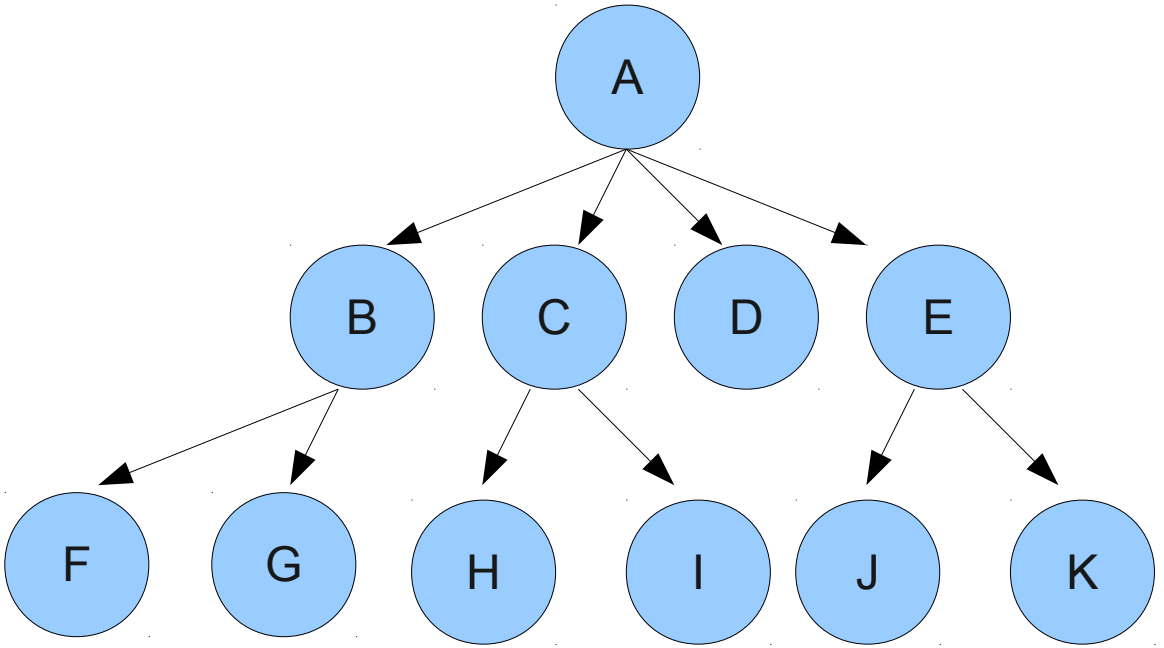
Searching Graphs

Doug Blank
Fall 2010

Searching a Graph



Searching a Graph



Breadth First Search

```
def BFS(graph, goal):
    current = graph.initial_state
    to_explore = expand(current)
    while to_explore:
        if current == goal:
            break
        current = to_explore[0]
        to_explore = to_explore + expand(current)
```

Depth First Search

```
def DFS(graph, goal):
    current = graph.initial_state
    to_explore = expand(current)
    while to_explore:
        if current == goal:
            break
        current = to_explore[0]
        to_explore = expand(current) + to_explore
```

Terminology

- Visited nodes, states, or *vertices*
- *Fringe Vertices*: expanded, but not yet visited
- *Unseen Vertices*: not yet encountered
- *Adjacency Vertices*: two states that have a connecting edge
  - List per state, or global Matrix
- Depth First Search
  - Last-in, first-out (LIFO)
- Breadth First Search
  - First-in, First-out (FIFO)
- DFS and BFS: O(|V| + |E|) - linear in the number of vertices and edges

BFS Graph Traversal

```
def BFS(graph, goal):
    visited = [ ]
    current = graph.initial_state
    to_explore = expand(current)
    while to_explore:
        visited.append(current)
        if current == goal:
            break
        current = to_explore[0]
        if not current in visited:
            to_explore = to_explore + expand(current)
```

BFS Graph Traversal

```
def search(graph, goal):
    visited = [ ]
    current = graph.initial_state
    to_explore = Queue(expand(current))
    while not to_explore.is_empty():
        visited.append(current)
        if current == goal:
            break
        current = to_explore.pop()
        if not current in visited:
            to_explore = to_explore.extend(expand(current))
```

DFS Graph Traversal

```
def search(graph, goal):
    visited = [ ]
    current = graph.initial_state
    to_explore = Stack(expand(current))
    while not to_explore.is_empty():
        visited.append(current)
        if current == goal:
            break
        current = to_explore.pop()
        if not current in visited:
            to_explore = to_explore.extend(expand(current))
```

Generalized Graph Traversal

```
def search(graph, goal, datastruct):
    visited = [ ]
    current = graph.initial_state
    to_explore = datastruct(expand(current))
    while not to_explore.is_empty():
        visited.append(current)
        if current == goal:
            break
        current = to_explore.pop()
        if not current in visited:
            to_explore = to_explore.extend(expand(current))

search(engine, “laboratory”, Stack)
```

Stack

```
class Stack:
    def __init__(self, nodes=None):
        self.nodes = [ ]
        self.extend(nodes)
    def push(self, v):
        self.nodes.append(v)
    def extend(self, nodes):
        for node in nodes:
            self.insert(node)
    def pop(self):
        return self.nodes.pop()
    def is_empty(self): return len(self.nodes) == 0
```

Queue

```
class Queue:
    def __init__(self, nodes=None):
        self.nodes = [ ]
        self.extend(nodes)
    def push(self, v):
        self.nodes.append(v)
    def extend(self, nodes):
        for node in nodes:
            self.insert(node)
    def pop(self):
        return self.nodes.pop(0)
    def is_empty(self): return len(self.nodes) == 0
```

Queue

```
class Queue(Stack):
    def pop(self):
        return self.nodes.pop(0)
```

Generalized Graph Traversal

```
def search(graph, goal, datastruct):
    visited = [ ]
    current = graph.initial_state
    to_explore = datastruct(expand(current))
    while not to_explore.is_empty():
        visited.append(current)
        if current == goal:
            break
        current = to_explore.pop()
        if not current in visited:
            to_explore = to_explore.extend(expand(current))

search(engine, “laboratory”, Stack)
```

Generalized Graph Traversal

```
def search(graph, goal, datastruct):
    current = graph.initial_state
    to_explore = datastruct(expand(current))
    while not to_explore.is_empty():
        current.visited = True
        if current == goal:
            break
        current = to_explore.pop()
        if not current.visited:
            to_explore = to_explore.extend(expand(current))

search(engine, “laboratory”, Stack)
```

Generalized Graph Traversal

```
def search(graph, goal, datastruct):
    current = graph.initial_state
    to_explore = datastruct(expand(current))
    while not to_explore.is_empty():
        current.visited = True
        if current == goal:
            return current
        current = to_explore.pop()
        if not current.visited:
            to_explore = to_explore.extend(expand(current))
    return None

result = search(engine, “laboratory”, Stack)
```

Generalized Graph Traversal

```
all_states = engine.states

def expand(state_name):
    retval = [ ]
    for edge in all_states[state_name].edges:
        retval.append(edge.to_state)
    return retval
```

Python: List Comprehension

```
all_states = engine.states

def expand(state_name):
    return [edge.to_state for edge in
                                all_states[state_name].edges]
```

Reporting a Path

```
all_states = engine.states

def expand(state_name):
    retval = [ ]
    for edge in all_states[state_name].edges:
        all_states[edge.to_state].parent = all_states[state_name]
        retval.append(edge.to_state)
    return retval
```

Reporting a Path

- Before beginning search:
  - Initialize all state.visited to False
  - Initialize all state.parent to None
- To report the path:
  - search(engine, “goal”, Queue) returns the goal state name, or None
  - State.parent will give you the state before that one on path to goal
  - Collect, until State.parent == None
  - Reverse. That is the Path to goal from start