Graph Search

These are slides I usually use for teaching AI, but they offer another perspective on BFS, DFS, and Dijkstra's algorithm. Ignore mentions of a "problem" and simply consider that the search starts at an initial node and that the expand-nodes method returns the potential successor nodes.

Essentially, each of the three graph searches can be formed by using the basic search algorithm (on the next slide) and substituting in different types of queues:

BFS – standard FIFO queue

DFS – LIFO stack

Dijkstra's – Priority queue by path cost

State-space search algorithm

```
function general-search (problem, QUEUEING-FUNCTION)
;; problem describes the start state, operators, goal test, and operator costs
;; queueing-function is a comparator function that ranks two states
;; general-search returns either a goal node or failure
nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
             problem.OPERATORS))
 end
  ;; Note: The goal test is NOT done when nodes are generated
  ;; Note: This algorithm does not detect loops
```

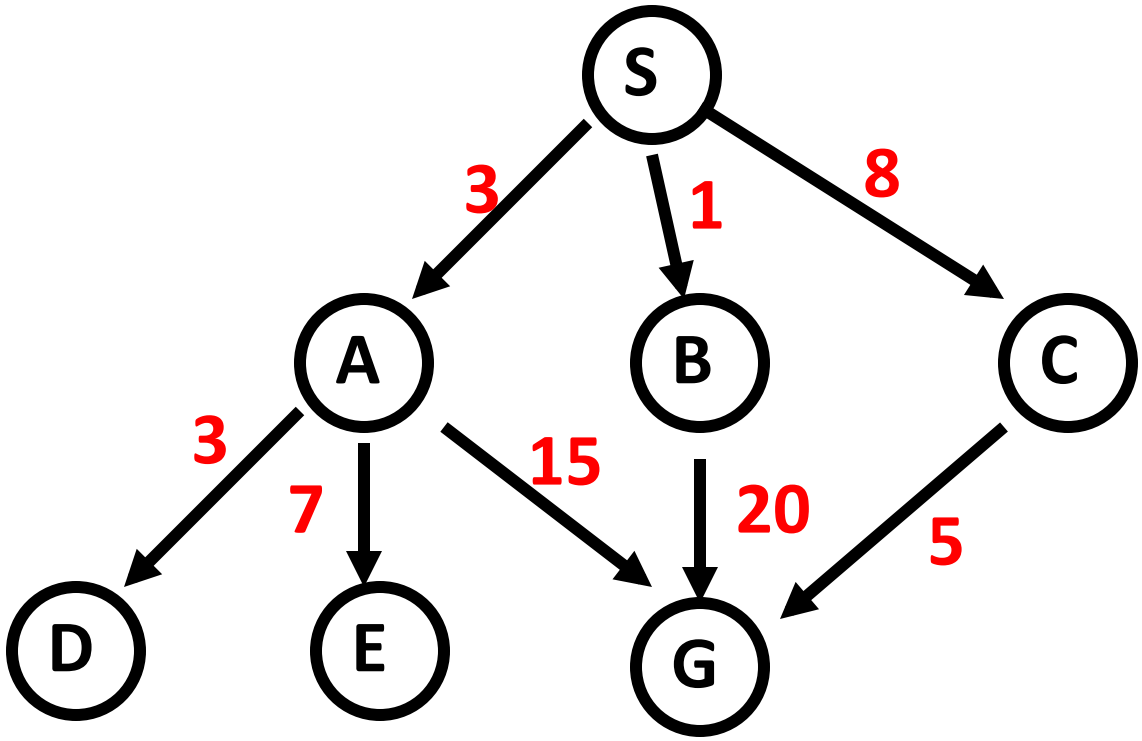
Key procedures to be defined

- EXPAND
  - Generate all successor nodes of a given node
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion

Evaluating Search Strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded
- **Space complexity**
  - How much space is used by the algorithm? Usually measured in terms of the maximum size of the "nodes" list during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?

Example for illustrating uninformed search strategies



Breadth-First

- Enqueue nodes in **FIFO** (first-in, first-out) order.
- **Complete**
- **Optimal** (i.e., admissible) if all operators have the same cost. Otherwise, not optimal but finds solution with shortest path length.
- **Exponential time and space complexity**, $O(b^d)$, where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node
- Will take a **long time to find solutions** with a large number of steps because must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children, has a total of $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes
  - For a complete search tree of depth 12, where every node at depths 0, ..., 11 has 10 children and every node at depth 12 has 0 children, there are $1 + 10 + 100 + 1000 + ... + 10^{12} = (10^{13} - 1)/9 = O(10^{12})$ nodes in the complete search tree. If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

Depth-First (DFS)

- Enqueue nodes in **LIFO** (last-in, first-out) order. That is, use a stack data structure to order nodes.
- **May not terminate** without a "depth bound," i.e., cutting off search below a fixed depth D ("depth-limited search")
- **Not complete** (with or without cycle detection, and with or without a cutoff depth)
- **Exponential time**, $O(b^d)$, but only **linear space**, O(bd)
- Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)
- When search hits a dead-end, can only back up one level at a time even if the "problem" occurs because of a bad operator choice near the top of the tree. Hence, only does "chronological backtracking"

Uniform-Cost (UCS)

- Enqueue nodes by **path cost**. That is, let g(n) = cost of the path from the start node to the current node n. Sort nodes by increasing value of g.
- Called *"Dijkstra's Algorithm"* in the algorithms literature and similar to *"Branch and Bound Algorithm"* in operations research literature
- **Complete (*)**
- **Optimal/Admissible** (*)
- Admissibility depends on the goal test being applied *when a node is removed from the nodes list*, not when its parent node is expanded and the node is first generated
- **Exponential time and space complexity**, $O(b^d)$