Dcs

Year IV
Semester I
2016

Uninformed Search

SESSION V

gechb21@gmail.com

Dcs

Year IV
Semester I
2016

SESSION
V

Contents to be covered

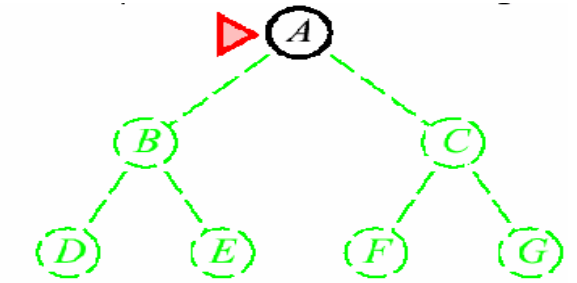
- Uninformed Searching Strategies

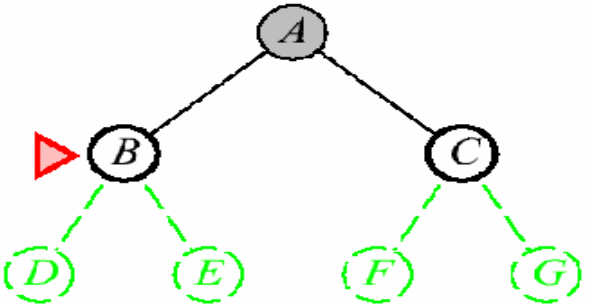
Uninformed Search Methods:

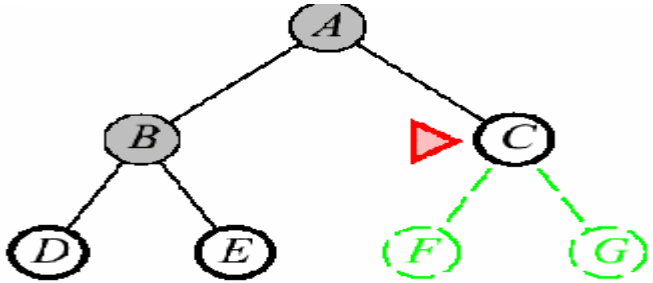
- Breadth first
- Depth first
- Depth limited search
- Iterative deepening
- Uniform cost
- etc.

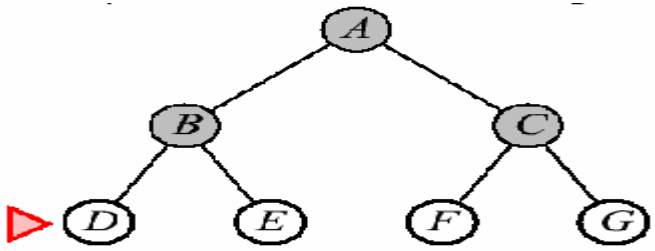
Breadth first search

- Expand shallowest unexpanded node,
  - i.e. expand all nodes on a given level of the search tree before moving to the next level
- **Implementation**: use **queue data structure** to store the list:
  - **Expansion:** put successors at the end of queue
  - Pop nodes from the front of the queue
- Properties:
  - **Takes space:** keeps every node in memory
  - **Optimal and complete:** guarantees to find solution







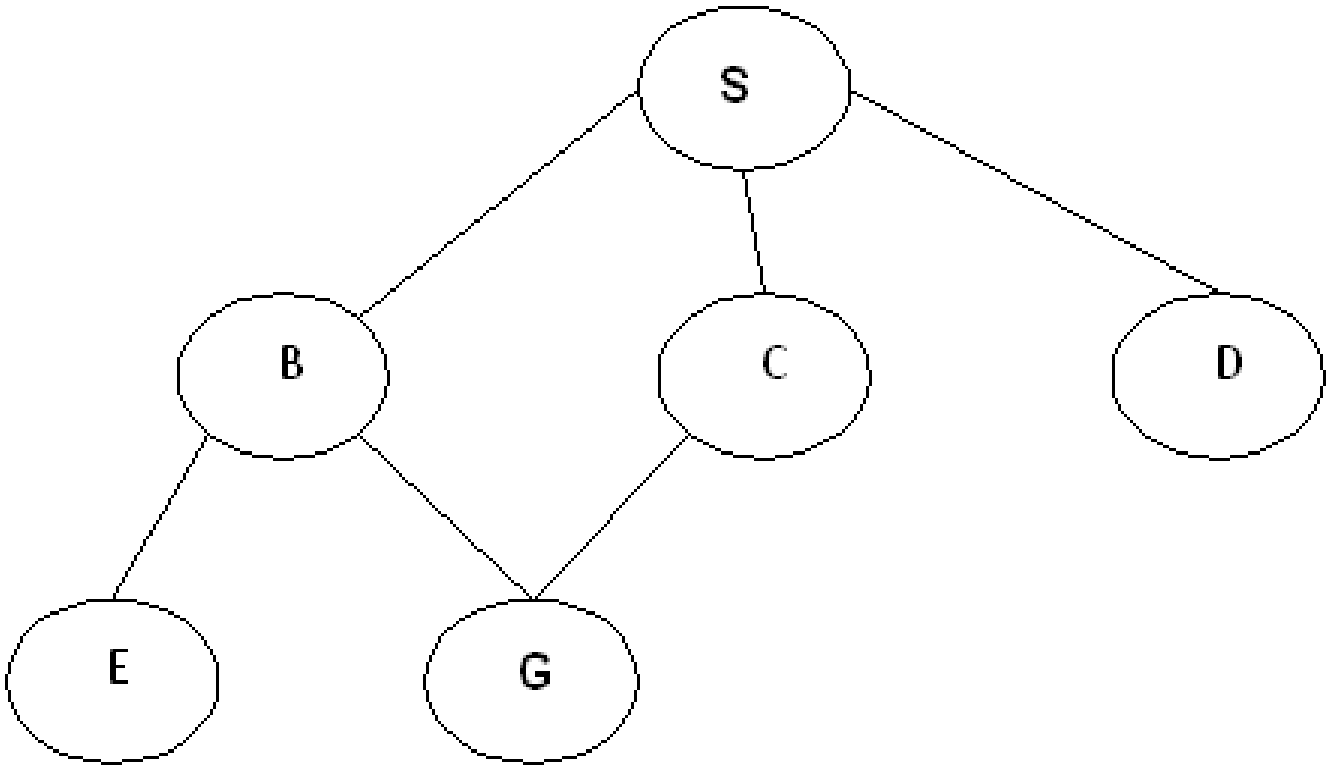


Algorithm for Breadth first search

```
function BFS (problem){
        open = (C_0); //put initial state C_0 in the List
        closed = {}; /maintain list of nodes examined earlier
        while (not (empty (open))) {
            f = remove_first(open);
            if IsGoal (f) then return (f);
            closed = append (closed, f);
          l = not-in-set (Successors (f), closed );
            open = merge ( rest(open), l); //append to the list
         }
        return ('fail')
    }
```

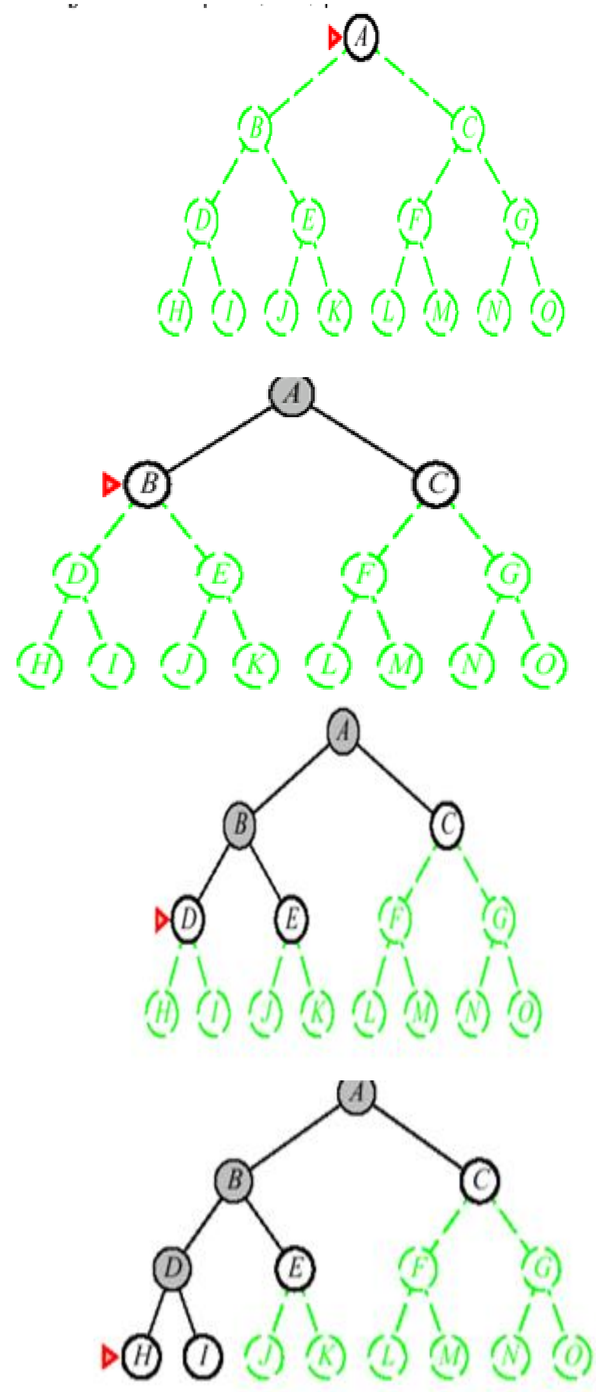
Exercise

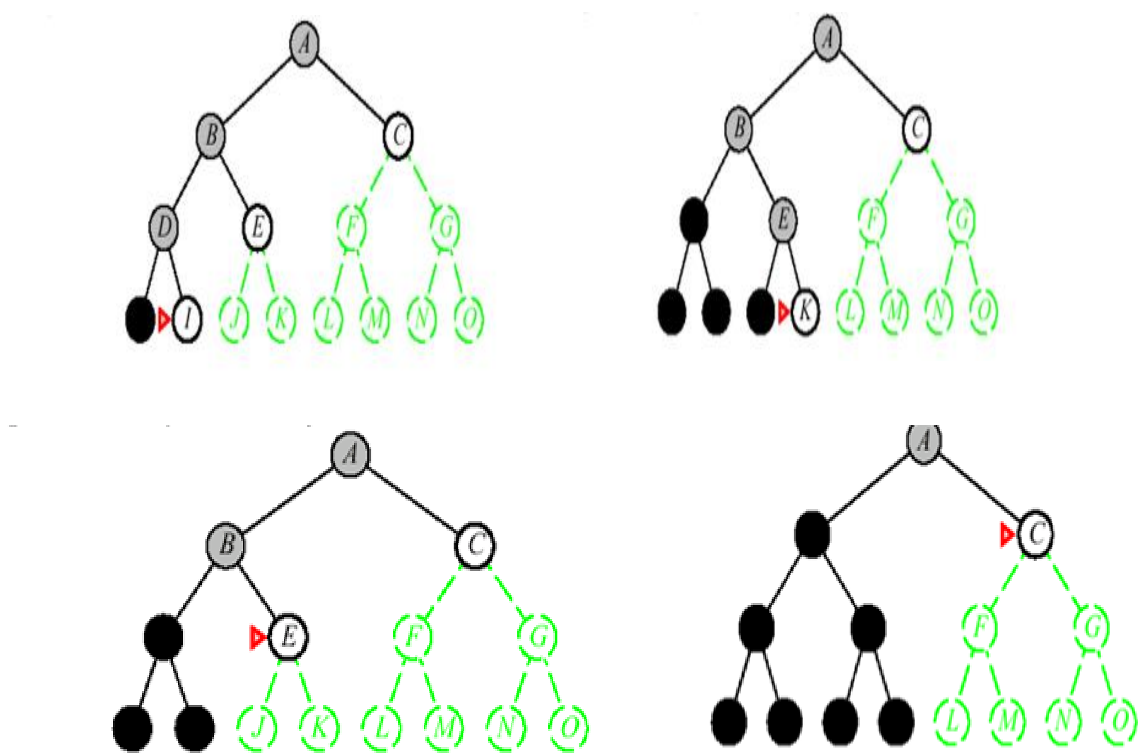
- Apply BFS to find an optimal path from start node to Goal node.



Depth-first search

- Expand one of the node at the deepest level of the tree.
  - Only when the search hits a non-goal dead end does the search go back and expand nodes at shallower levels





Depth-first search…

- **Implementation**: treat the list as **stack**
  - Expansion: push successors at the top of stack
  - Pop nodes from the top of the stack
- Properties
  - Incomplete and not optimal: fails in infinite-depth spaces, spaces with loops.
    - Modify to avoid repeated states along the path
  - Takes less space (Linear): Only needs to remember up to the depth expanded

Algorithm for Depth first search

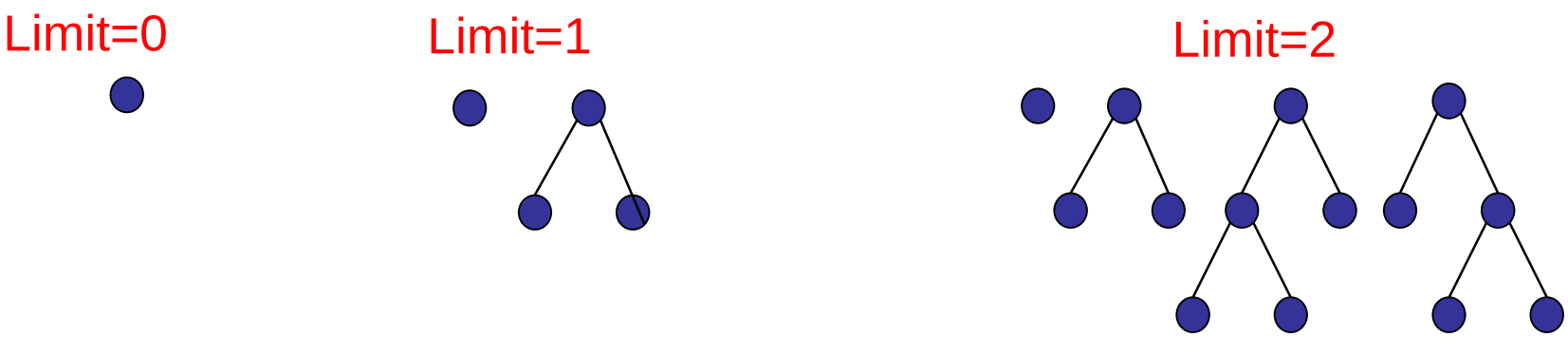
```
function DFS (problem){
        open = (C_0); //put initial state C_0 in the List
        closed = {}; /maintain list of nodes examined earlier
        while (not (empty (open))) {
            f = remove_first(open);
            if IsGoal (f) then return (f);
            closed = append (closed, f);
           l = not-in-set (Successors (f), closed );
            open = merge ( rest(open), l); //prepend to the list
         }
        return ('fail')
    }
```

Iterative Deepening Search (IDS)

- IDS solves the issue of choosing the best depth limit by trying all possible depth limit:
  - Perform depth-first search to a bounded depth *d*, starting at *d = 1* and increasing it by 1 at each iteration.

Example: for route finding problem we can take the diameter of the state space. In our example, at most 9 steps is enough to reach any node

- This search combines the benefits of DFS and BFS
  - DFS is efficient in space, but has no path-length guarantee
  - BFS finds min-step path towards the goal, but requires memory space
  - IDS performs a sequence of DFS searches with increasing depth-cutoff until goal is found

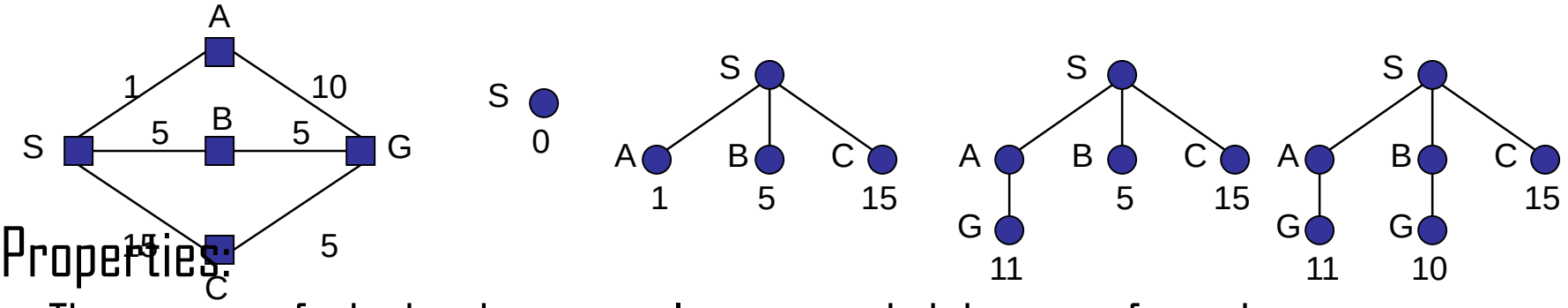


Algorithm for IDS

```
function IDS (problem){
        open = (C_0); //put initial state C_0 in the List
        closed = {}; /maintain list of nodes examined earlier
        while (not reached maxDepth) {
          while (not (empty (open))) {
                f = remove_first(open);
                if (IsGoal (f)) then return (f);
                closed = append (closed, f);
               l = not-in-set (Successors (f), closed);
                if (depth(l) < maxDepth) then
                        open = merge (rest(open), l); //prepend to the list
           }
         }
         return ('fail')
}
```

Uniform cost Search

- The goal of this technique is to find the shortest path to the goal in terms of cost.
  - It modifies the BFS by always expanding least-cost unexpanded node
- Implementation: nodes in list keep track of total path length from start to that node
  - List kept in priority queue ordered by path cost



- Properties:
  - This strategy finds the cheapest solution provided the cost of a path must never decrease as we go along the path

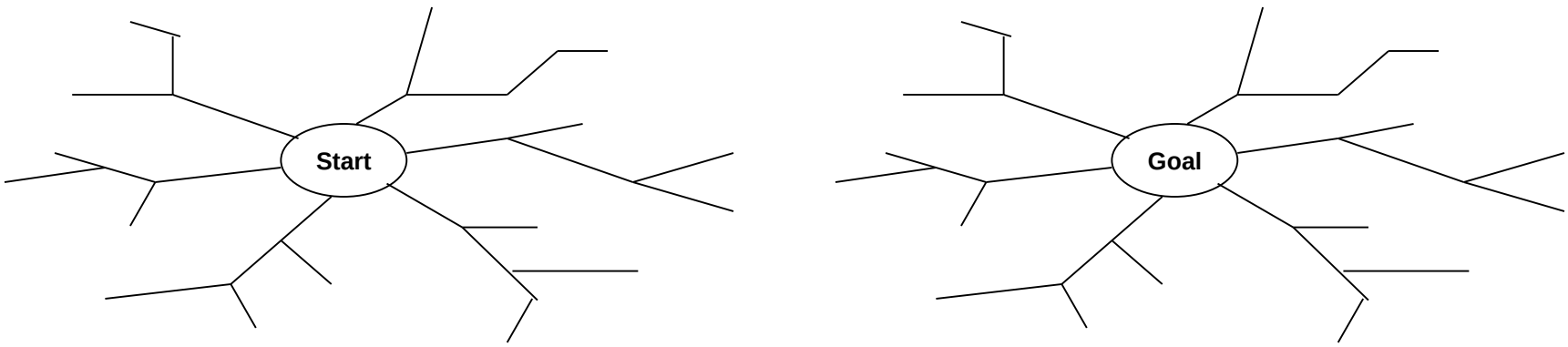
    $g(\text{successor}(n)) \geq g(n)$, for every node n
  - Takes space since it keeps every node in memory

Algorithm for Uniform Cost search

```
function uniform_cost (problem){
        open = (C_0); //put initial state C_0 in the List
       g(s) = 0;
        closed = {}; /maintain list of nodes examined earlier
        while (not (empty (open))) {
            f = remove_first(open);
            if IsGoal (f) then return (f);
            closed = append (closed, f);
          l = not-in-set (Successors (f), closed );
           g(f,l_i) = g(f) + c(f,l_i);
          open = merge(rest(open), l, g(f,l_i)); //keep the open list sorted in
           ascending order by edge cost
            }
        return ('fail')
    }
```

Bidirectional Search

- Simultaneously search both forward from the initial state to the goal and backward from the goal to the initial state, and stop when the two searches meet somewhere in the middle
  - Requires an explicit goal state and invertible operators (or backward chaining).
  - Decide what kind of search is going to take place in each half using BFS, DFS, uniform cost search, etc.



Bidirectional Search

- Advantages:
  - Only need to go to half depth
  - It can enormously reduce time complexity, but is not always applicable

- Difficulties
  - Do you really know solution? Unique?
  - Cannot reverse operators
  - Memory requirements may be important: Record all paths to check they meet
    - Memory intensive
- Note that if a heuristic function is inaccurate, the two searches might miss one another.

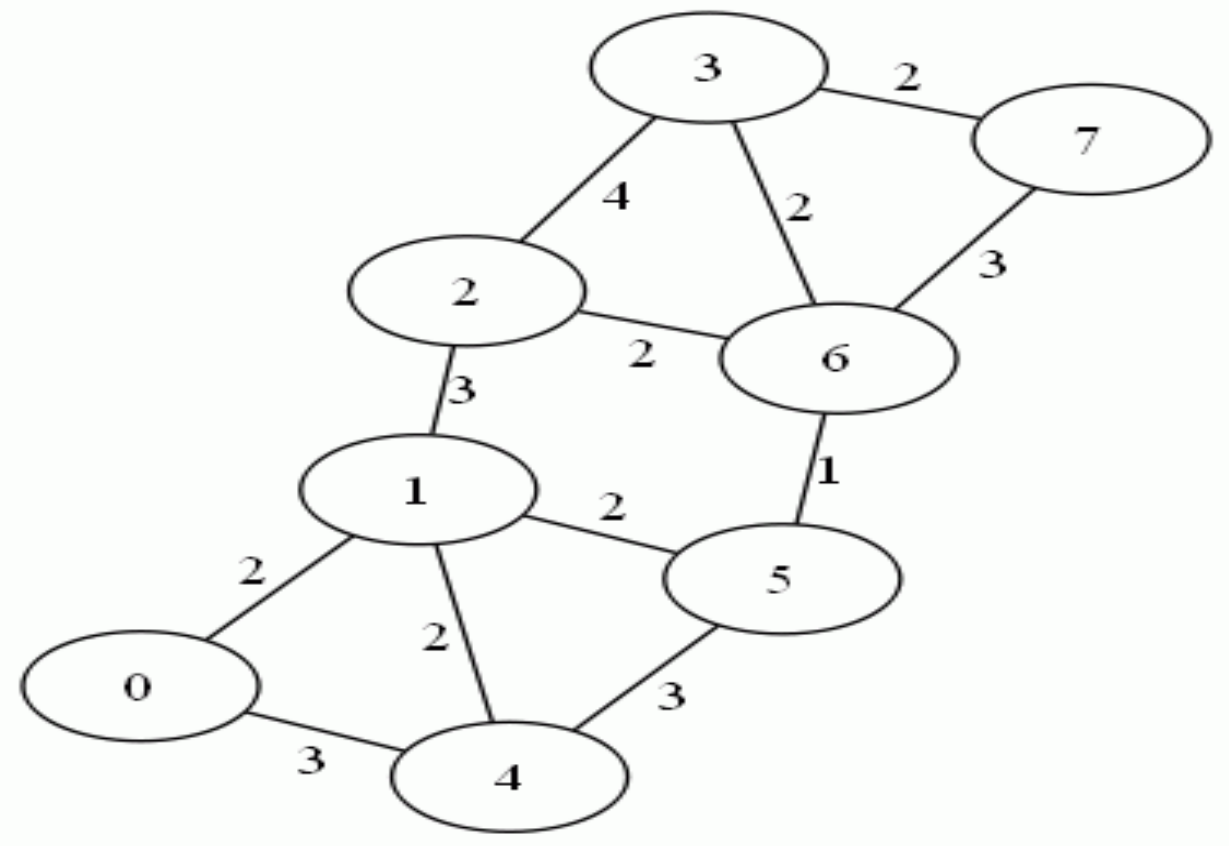
Comparing Uninformed Search

Strategies	Complete	Optimal	Time complexity	Space complexity
Breadth first search	yes	yes	$O(b^d)$	$O(b^d)$
Depth first search	no	no	$O(b^m)$	$O(bm)$
Uniform cost search	yes	yes	$O(b^d)$	$O(b^d)$
Depth limited search	if l >= d	no	$O(b^l)$	$O(bl)$
Iterative deepening search	yes	yes	$O(b^d)$	$O(bd)$
bi-directional search	yes	yes	$O(b^{d/2})$	$O(b^{d/2})$

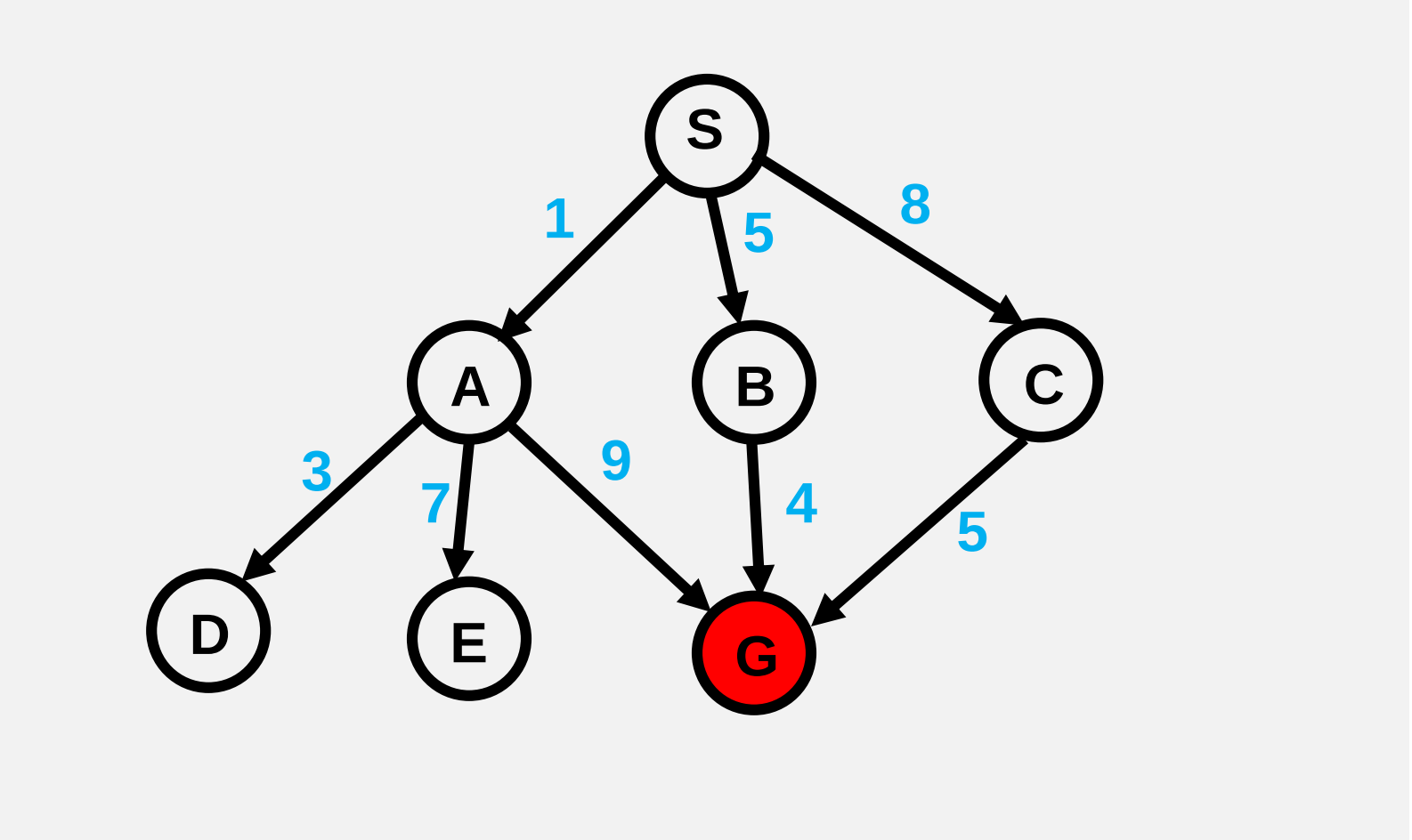
- b is branching factor,
- d is depth of the shallowest solution,
- m is the maximum depth of the search tree,
- l is the depth limit

Exercise: Uninformed Search Strategies

- Assume that node 3 is the initial state and node 5 is the goal state



Exercise: Apply Uninformed Search Strategies to identify optimal path



አመሰግናለሁ !!!