UNINFORMED SEARCH
-BFS
-DFS

INT404

- **State-space search** is the process of searching through a state space for a solution by making explicit a sufficient portion of an implicit state-space graph to include a goal node.
  - Hence, initially V={S}, where S is the start node;
  - when S is expanded, its successors are generated and those nodes are added to V and the associated arcs are added to E.
  - This process continues until a goal node is generated (included in V) and identified (by goal test)
- During search, a node can be in one of the three categories:
  - Not generated yet (has not been made explicit yet)
  - **OPEN**: generated but not expanded
  - **CLOSED**: expanded
  - Search strategies differ mainly on how to select an OPEN node for expansion at each step of search

A General State-Space Search Algorithm

```
open := {S}; closed :={};
repeat
    n := select(open);          /* select one node from open for expansion */
        if n is a goal
            then exit with success;  /* delayed goal testing */
        expand(n)
                /* generate all children of n
                   put these newly generated nodes in open (check duplicates)
                   put n in closed (check duplicates) */
until open = {};
exit with failure
```

Some Issues

- Search process constructs a search tree, where
  - **root** is the initial state S, and
  - **leaf nodes** are nodes
    - not yet been expanded (i.e., they are in OPEN list) or
    - having no successors (i.e., they're "deadends")
- Some important issue that arises
  - The direction in which conduct the search(forward vs. backward reasoning)
  - How to select applicable rules(matching).
  - How to represent each node of search process(the knowledge representation problem)
  - Search tree vs. search graph

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 that the "OPEN" list** becomes during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? For example, is it the one with minimum cost?

Algorithm : Breadth-First Search

1. Create a variable called *NODE-LIST* and set it to the initial state.
2. Until a goal state is found or *NODE-LIST* is empty:

(a) Remove the first element from *NODE-LIST* and call it *E.* If *NODE-LIST* was empty, quit.

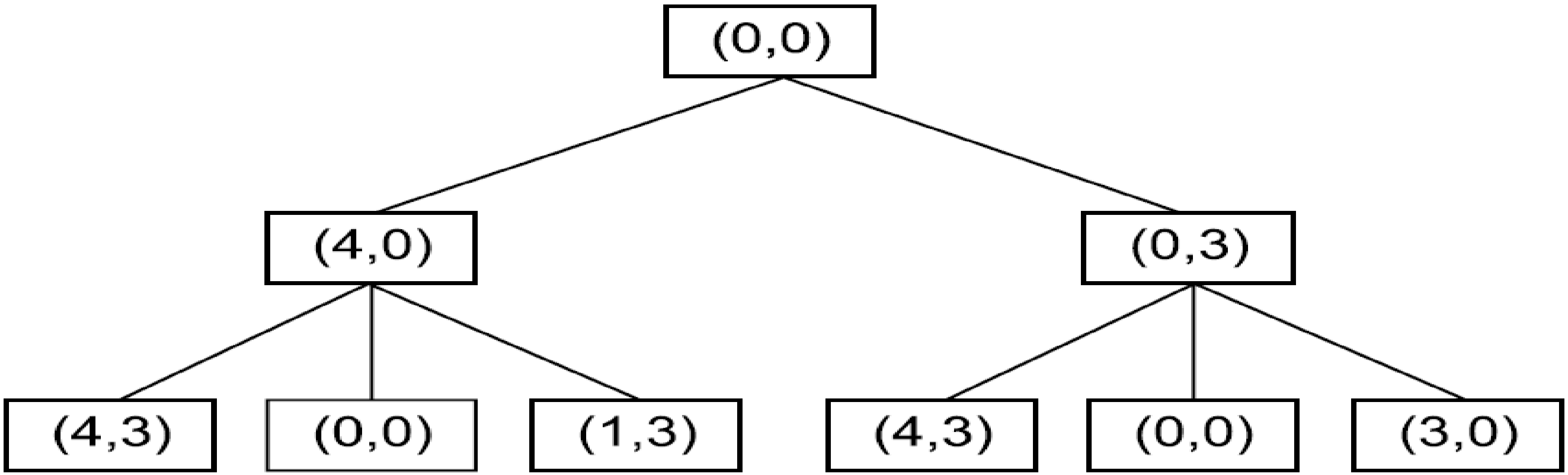
(b) For each way that each rule can match the state described in *E* do:

(i) Apply the rule to generate a new state,

(ii) If the new state is a goal state, quit and return this state.

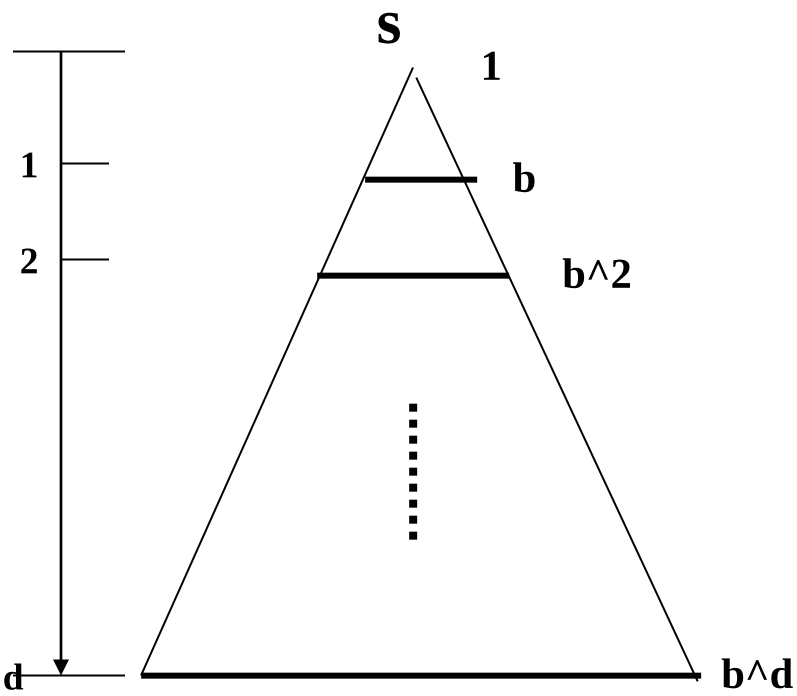
(iii) Otherwise, add the new state to the end of NODE-LIST.

Two Levels of a Breadth-First Search Tree



Breadth-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
- Time complexity (# of nodes generated): O(b^d)
- Space complexity (maximum length of OPEN): O(b^d)



– 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.

- BFS is suitable for problems with shallow solutions

Host

Searcher

Host

Host

Algorithm : Depth-First Search

1. If the initial state is a goal state, quit and return success.

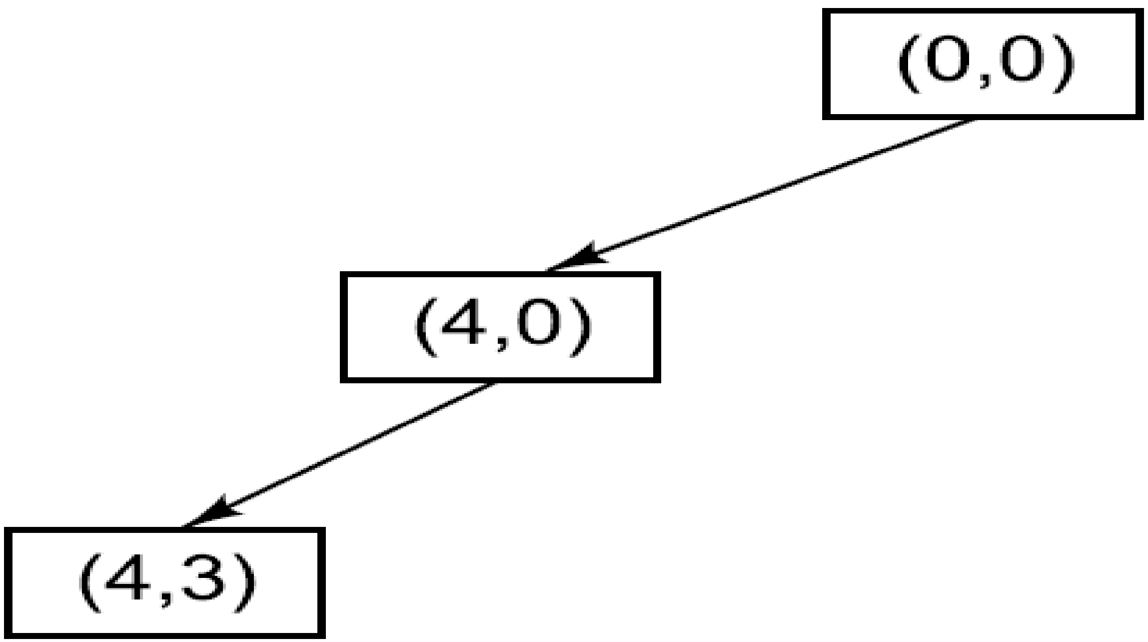
2. Otherwise, do the following until success or failure is signaled:

   (a) Generate a successor, *E,* of the initial state. If there are no more successors, signal failure.

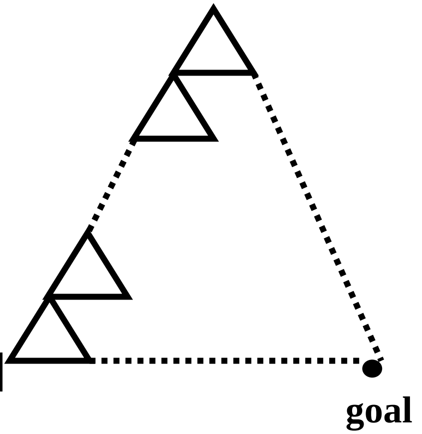
   (b) Call Depth-First Search with *E* as the initial state.

   (c) If success is returned, signal success. Otherwise continue in this loop.

A Depth-First Search Tree



Depth-First (DFS)



- Algorithm outline:
  - Always select from the OPEN the node with the greatest depth for expansion, and put all newly generated nodes into OPEN
  - OPEN is organized as **LIFO** (last-in, first-out) list.
  - Terminate if a node selected for expansion is a goal
- **May not terminate** without a "depth bound," i.e., cutting off search below a fixed depth D (How to determine the depth bound?)
- **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), required
- Can find **deep solutions quickly** if lucky
- When search hits a deadend, 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"

Thank You !!!