HOMEWORK – 1

CS 6343

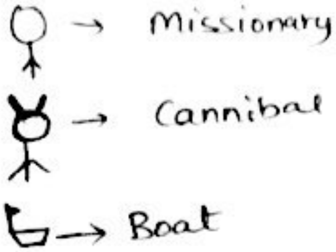
Submission Date: February 08, 2015

Submitted By: Kanika Kapoor
Kxk140230

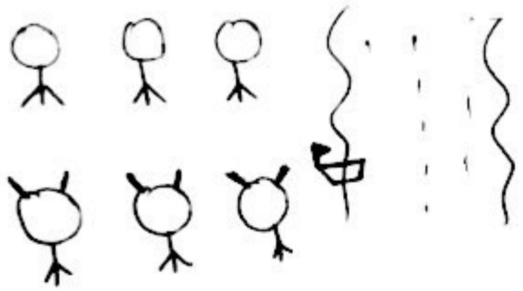
QUESTION – 1
Missionary and Cannibals Problem

Three missionaries and three cannibals are on one side of the river, along with a boat that can hold at most two people. Find a way to get everyone from one side of the river to the other side of the river, without leaving any group of missionaries (on either side of the river or in the boat) outnumbered by the cannibals in that place.

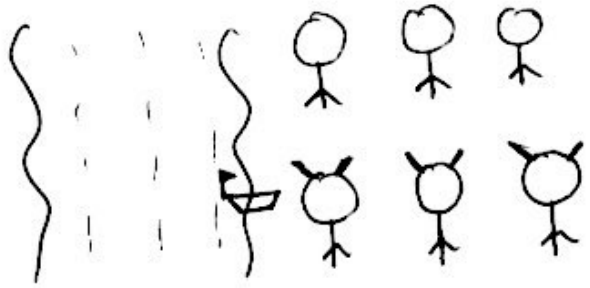
. a)



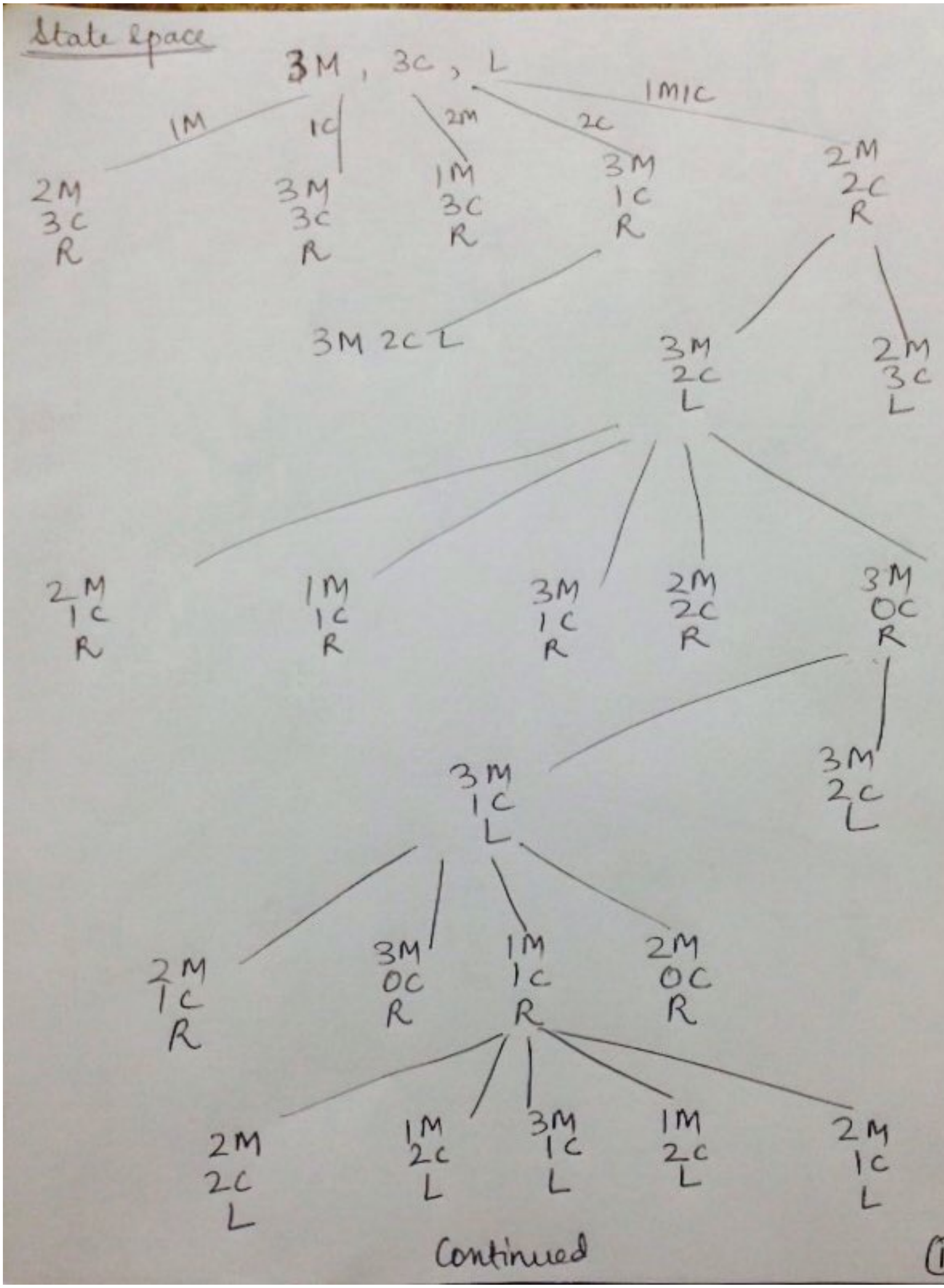
Start/Initial State



Goal/Final State

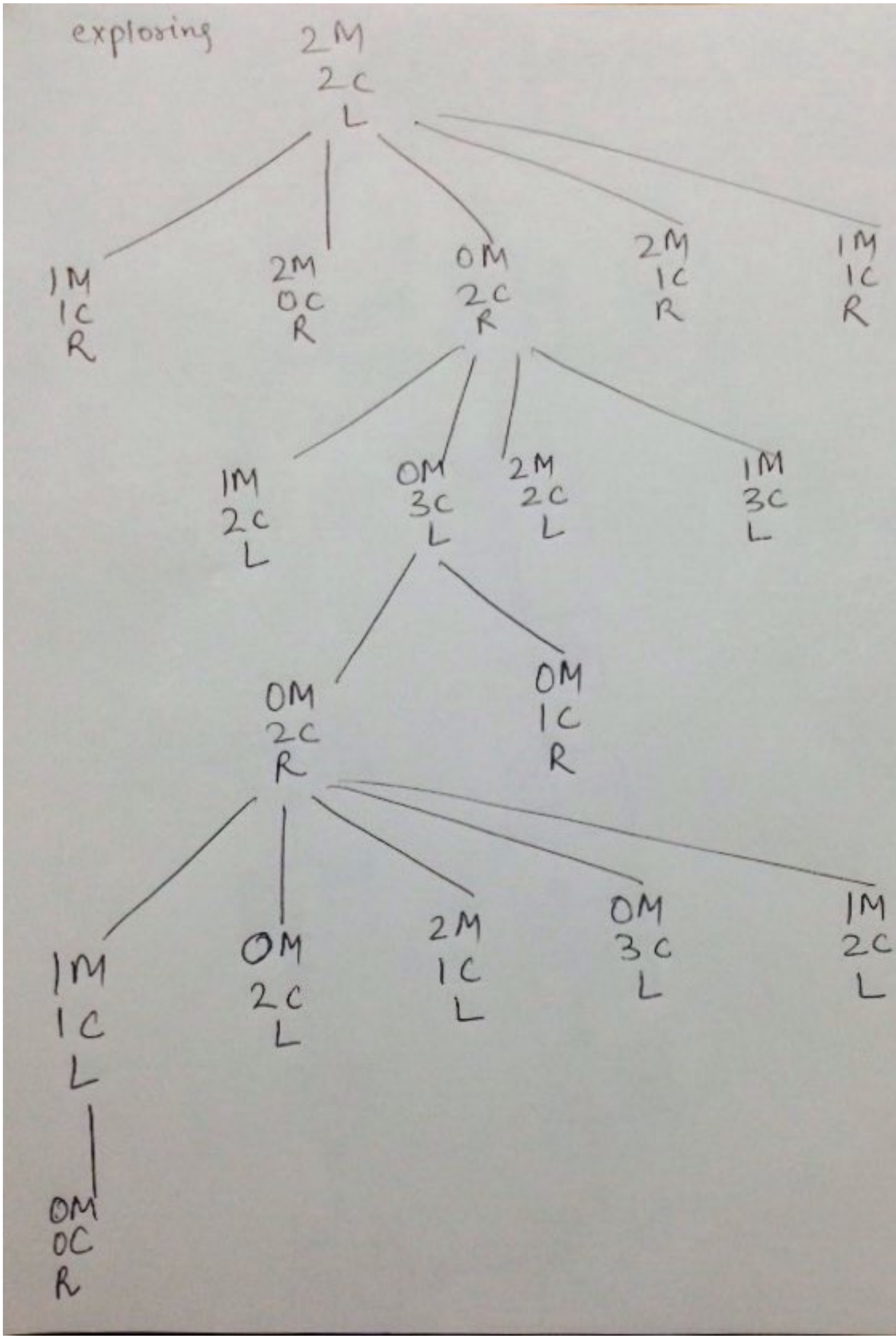


State Space



Continued

exploring

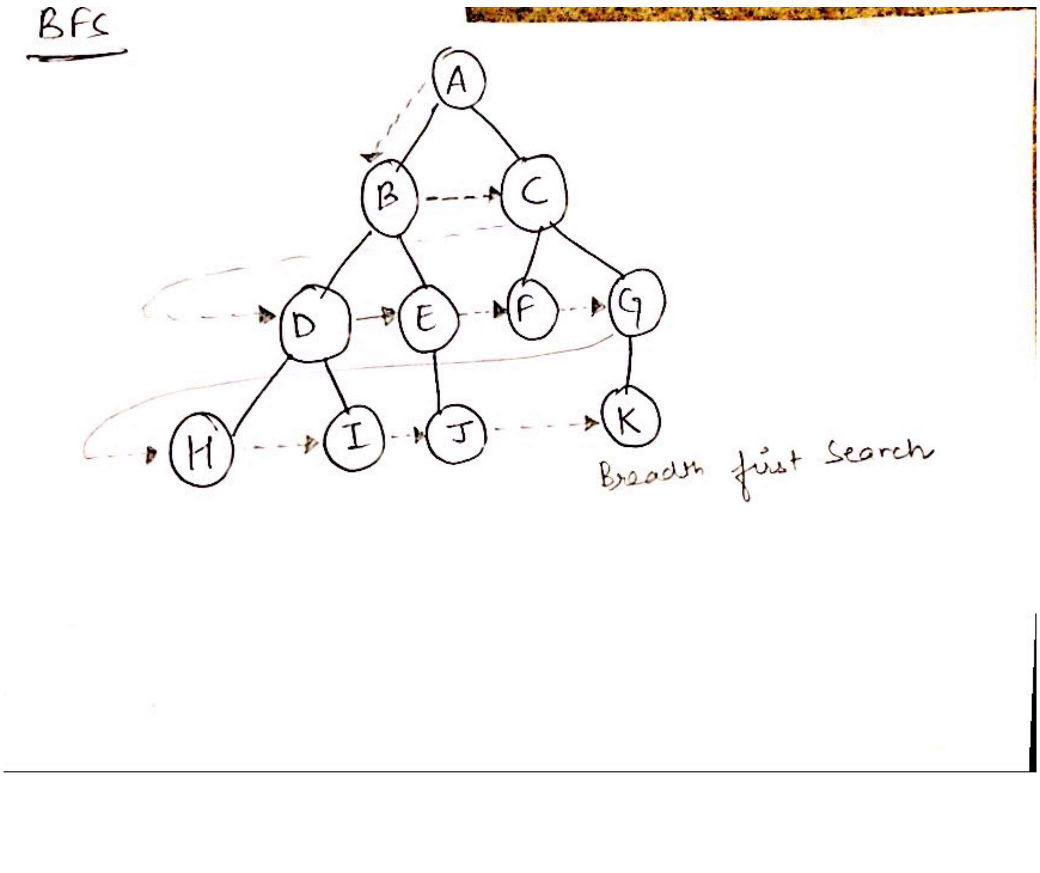


2M 2c L
1M 1c R
2M 0c R
0M 2c R
2M 1c R
1M 1c R
1M 2c L
0M 3c L
2M 2c L
1M 3c L
0M 2c R
0M 1c R
1M 1c L
0M 2c L
2M 1c L
0M 3c L
1M 2c L
0M 0C R

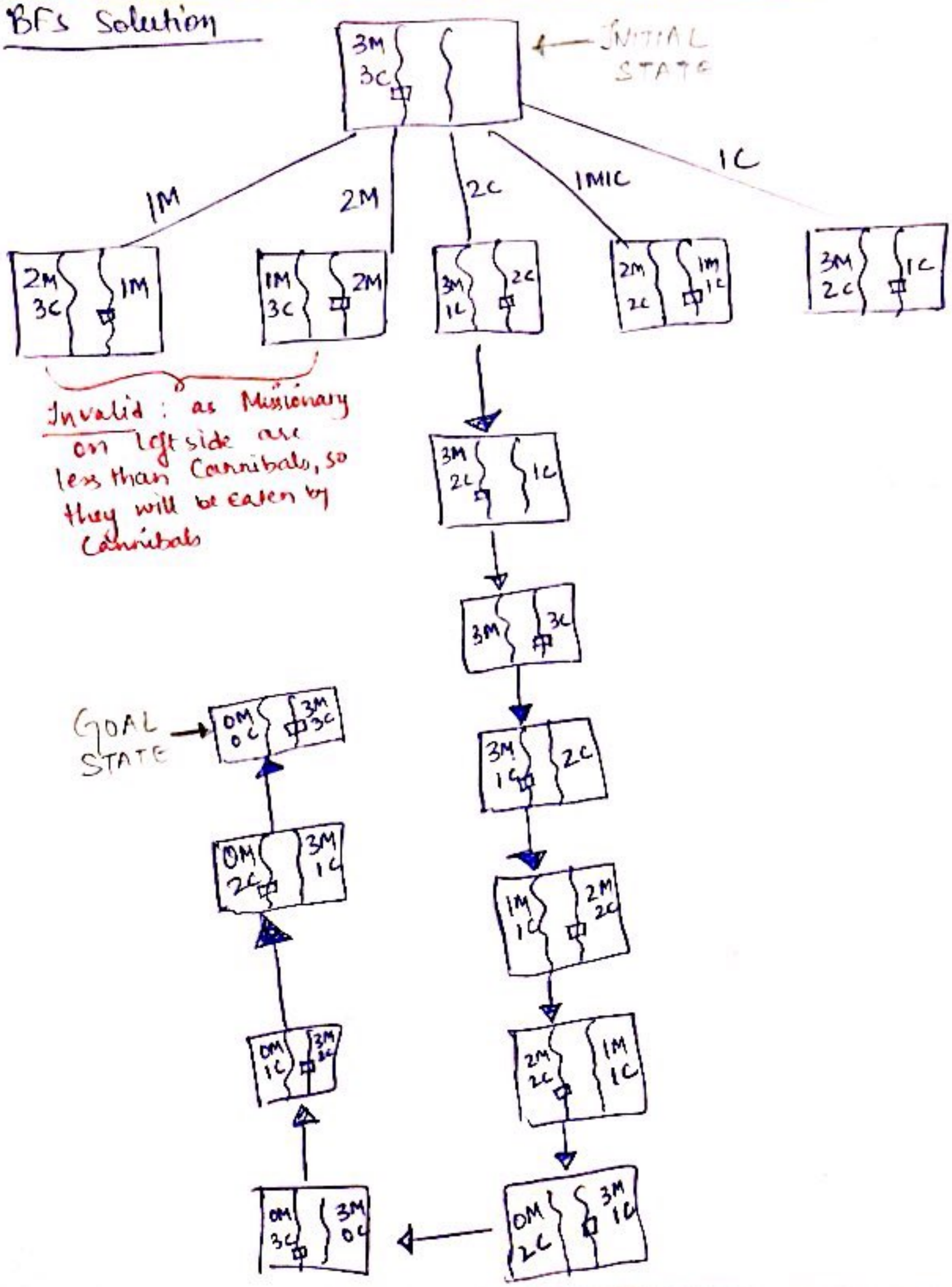
b) We have used Breadth first search algorithm because the graph is small and we have considered all step costs to be equivalent. Also, the best part of BFS is the optimality and completeness. It assures a complete solution when branching factor b is finite.

Constraints:

- Use boat to cross the river
- Boat cannot handle more than two people (missionary or cannibal)
- Missionary must never be less than cannibals; otherwise, cannibal will destroy the missionaries.



BFs Solution



Invalid : as Missionary on left side are less than Cannibals, so they will be eaten by Cannibals

For repeated states, it is a good idea to check for them since if we keep on exploring the same states, then it will give a redundant solution. For this purpose, we have used the concept of HashMap in our program in order to keep a track of explored states.

c) The problem seems simple when we devise the initial and final states. But going through intermediate states, we understand that the problem gives rise to large number of states and out of them some are invalid. So we need to keep a track of those and avoid falling in loop because of repeated states if we don't keep a check on them. Also, we can use any type of uninformed search algorithm to find a solution to this problem but every search algorithm may not yield an optimal solution or even complete the search.
This may be the reason why people have a hard time solving this puzzle.

2. Define in your own words the following terms

- **State** - State corresponds to a configuration of the world i.e. physical configuration. It contains all necessary information that is required to determine if the state is a goal state or not, based on our actions using the search strategy. For example - water jugs problem (3 gallons and 4 gallons), states are determined by the amount of water in each jug.

- **State Space** - State space is simply defined as the set of all states reachable from the initial state. The state space forms a graph in which the nodes are states and the arcs between nodes are actions.

- **Node** - Node is a data structure used to represent the search tree. It holds other things also like cost, path information, depth information along with the state. Also, two different nodes can contain the same world state if that state is generated via two different search paths.

  A node is considered to be a collection of 5 data structures:

a. State: represents the state in state space to which this node corresponds
b. Parent node: points to node that generated this node
c. Operator: operator applied that generated the node
d. Depth: no. of branches from the root
e. Path cost: path cost from initial state to this node

- **Search Tree** - A search tree is generated by the initial state and successor function that together define the state space.

- **Search Node** - The node that is currently being searched/expanded or the node that is in consideration is known as search node.

- **Goal** - The goal is the destination state, which the agent is trying to reach to. For N-Queens problem, goal is a configuration with non-attacking queens and we apply different search strategy and actions to reach the goal.

- **Action** - An action is a move/step that the agent chooses to take in order to reach to its goal state. For example – water jugs problem, all possible steps of filling and throwing water from jugs in order to put 2 gallons of water in 4-gallon jug, which is the goal.

- **Successor Function** - It is a description of possible actions, a set of operators and is a transformation function on a state representation, which converts it into another state. A successor function is needed to move between different states.

- **Branching Factor** - Branching factor b, is the maximum number of successors of any node.

3. Prove each of the following statements, or give a counter - example

- **Breadth-first search is a special case of uniform-cost search.**
  Answer - The given statement is true.
  Breadth first search selects the shallowest unexpanded node in search tree for expansion.
  Uniform cost search is an extension of BFS where we can find optimal solution with any step cost function. Instead of expanding the shallowest node like BFS, UCS expands the node n with the lowest path cost.
  If all the step costs are equal, UCS is equivalent to BFS.

- **Depth-first search is a special case of best-first tree search.**
  Answer – The given statement is true.
  Depth-first search is best-first search with f(n) = - depth(n);
  DFS always expands the deepest node in current fringe of search tree. It goes to the deepest level of search tree, where the nodes have no successors.

As those nodes are expanded, they are dropped from fringe, so then the search backs up to the next shallowest node that still has unexplored successors.

Best first search (BFS) uses an evaluation function f(n) for each node and it expands most desirable unexpanded node. So at f(n) = - depth(n), DFS behaves likes Best first search.

- **Uniform-cost search is a special case of A* search**

  Answer - Uniform-cost search is A* search with h(n) = 0.

  A* uses an evaluation function f(n) which has two components:
  f(n)=g(n)+h(n) where n is a node
  The given statement is true if its heuristic is a constant function. If A* is used with a monotonic heuristic, then it can be turned into UCS by subtracting from each edge cost the decrease in heuristic value along that edge.