# Assignment No: 2

Q.1) Explain why problem formulation must follow goal formulation.

→ Goal formulation:

- This is a technique for an agent to conclude which goals need to be achieved.
- The most complicated task during goal formulation is to develop representations for agents to reason about and find out when the new goals need to be formulated due to plan failures or opportunities.

* Problem formulation:

- In this technique, the problem is precisely defined and the definition should also consist of clear specification of the initial situation as well the find situation with acceptable solutions to the problem.
- The major step is to select the best problem-solving technique and apply ist to the particular problem.

- When it comes to goal formulation, user gets to decide which aspects of the world we are interested in and which can be ignored or abstracted away.
- In problem formulation, it is decided how to manipulate the important aspects.

- If the problem formulation is performed before goal formulation then it is not possible to know what to include and what to exclude.
- There may be a cycle of iterations between goal formulation, problem formulation, and problem solving until one arrives at a sufficiently useful and efficient solution.

Q.2) Define in your own words the following terms: state, state space, search tree, goal node, action, transition model.

→ State: A state is a situation in which an agent can find itself. Stat is distinguished as world state and representational state

State Space: This is a graph whose nodes are the set of all states, and whose links are actions that transform one state into another.

Search tree: This is a tree in which a robot node is the start state, and the set of children for each node consists of the states reachable by taking any actions.

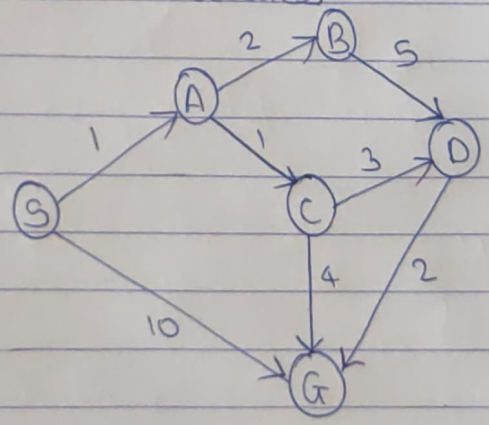
Search node: This is also a node in the search tree.

Goal node: This is a state that the agent is trying to reach.

Action: This is what the agent can choose to do in the state.

Trasition model: This descvibes the agent's options given a state. It returns a set of (action, state) pairs where each state is the state reachable by taking the action.

Q.3) Trace the operation of A* search using heuristic values.




| State | h(n) |
|---|---|
| S | 5 |
| A | 3 |
| B | 4 |
| C | 2 |
| D | 6 |
| G | 0 |

→ S → A, G

$f(n) = h(n) + g(n)$

$f(A) = h(A) + g(A)$

$= 3 + 1$

$= 4$

$f(G) = h(G) + g(G)$

$= 0 + 10$

$= 10$

$A \rightarrow B, C$

$f(B) = h(B) + g(B)$
$= 4 + 3$
$= 7$

$f(C) = h(C) + g(C)$
$= 2 + 2$
$= 4$

$C \rightarrow D, G$
$f(D) = h(D) + g(D)$
$= 6 + 5$
$= 11$

$f(G) = h(G) + g(G)$
$= 0 + 6$
$= 6$

$S \rightarrow A \rightarrow C \rightarrow G$

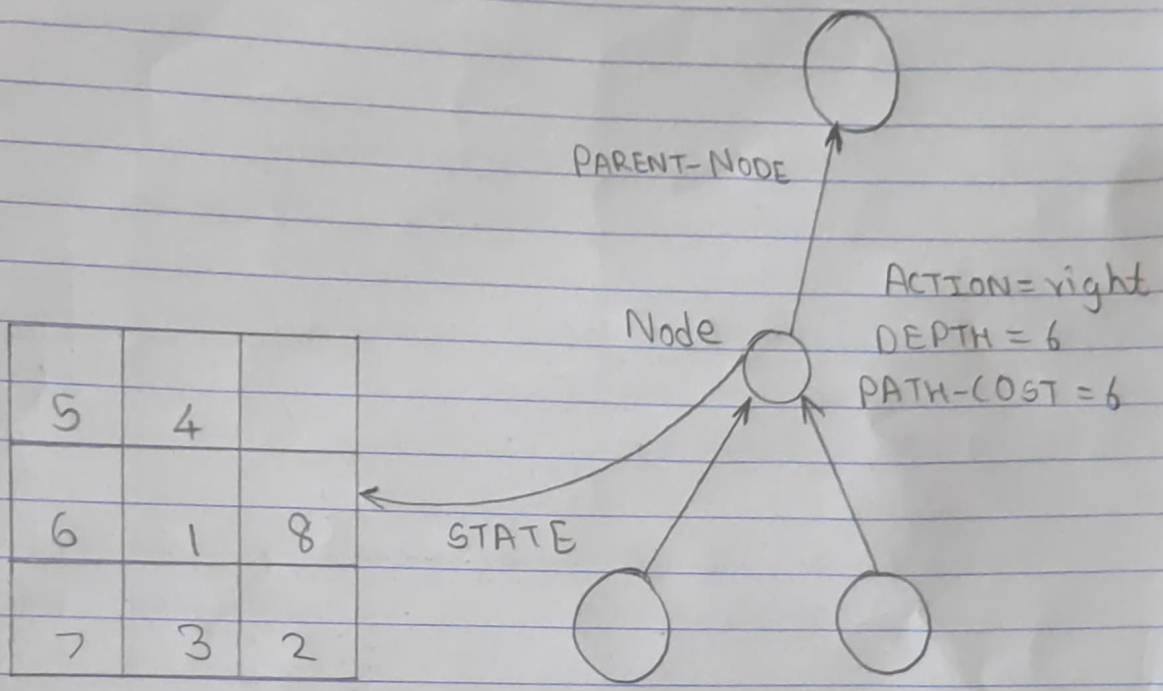
Q.4) Write a pseudocode for child node in infrastructure of search tree.

→ Search algorithms require a data structure to keep track of the search tree that is being constructed. For each node node n of the tree, we will have a structure that contains the following four components:

- n.State: the state in the space to which the node corresponds;
- n.Parent: the node in the search tree that generated this node;
- n.Action: the action that was applied to the parent to generate the node;
- n.PATH-COST: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers.

Given the components for a parent node, it is easy to see how to compute the necessary components for a child node. The function CHILD-NODE takes a parent node and an action and returns the resulting child node:

```
function CHILD-NODE(problem, parent, action)
        returns a node
returns a node with
    STATE = problem.RESULT(parent, STATE, action),
    PARENT = parent, ACTION = action,
    PATH-COST = parent.PATH-COST + problem.STEP-
                COST(parent.STATE, action)
```


PARENT-NODE
ACTION = right
Node
DEPTH = 6
PATH-COST = 6
STATE
5 4
6 1 8
7 3 2

Q 5) Differentiate between Informed search & uninformed search.

->

| Informed Search | Uninformed search |
|---|---|
| 1) It uses knowledge for the searching process. | 1) It doesn't use knowledge for searching process. |
| 2) If finds solution more quickly. | 2) It finds solution slow as compared to informed search. |
| 3) It may or may not be complete. | 3) It is always complete. |
| 4) Cost is low | 4) Cost is high. |
| 5) It consumes less time | 5) It consumes moderate time. |
| 6) It provides the direction regarding the solution | 6) No suggestion given regarding the solution in it. |
| 7) It is less lengthy while implementation. | 7) It is more lengthy while implemenetation. |
| 8) Greedy Search, A* Search, Graph Search | 8) Depth First Search, Breadth First Search. |

Q.6) Explain admissible heuristic with any example.

→ Admissible heuristics:-

- A heuristic $h(n)$ is admissible if for every node $n$, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from $n$.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, $A^*$ using TREE-SEARCH is optimal.

★ Example: Manhattan distance

- The Manhattan distance of a puzzle is defined as:

$$h(n) = \sum_{\text{all tiles}} \text{distance(tile, correct position)}$$

- Consider, the puzzle below in which the player wishes to move each tile such that the numbers are ordered.
- The Manhattan distance is an admissible heuristic in this case because every tile will have to be moved at least the number of spots in between itself and its correct position.

| | | | |
|---|---|---|---|
| $4_3$ | $6_1$ | $3_0$ | $8_1$ |
| $7_2$ | $12_3$ | $9_3$ | $14_4$ |
| $15_3$ | $13_2$ | $1_4$ | $5_4$ |
| $2_4$ | $10_1$ | $11_1$ | |

- The subscripts show the Manhattan distance for each tile. The total Manhattan distance for the shown puzzle. is:

$$= 3 + 4 + 3 + 2 + 4 + 4 + 4 + 1 + 1$$
$$= 36$$