Unit 2: Problem Solving

Poorva Agrawal

Disclaimer

- Some contents in this presentation are retrieved from the following original sources:
- 1. Washington State University: Artificial Intelligence Course
- 2. Introduction to Artifcial Intelligence by Prof. Bojana Dalbelo Basic and Assoc. Prof. Jan Snajder, University of Zagreb, Faculty of Electrical Engineering and Computing
- 3. https://www.educba.com/intelligent-agents/
- 4. https://www.geeksforgeeks.org/artificial-intelligence-an-introduction/

PROBLEM

- A goal and a set of means to achieve the goal is called a Problem.

- Process of exploring what the means can do is known as Searching.

Search

- Search permeates all of AI
- What choices are we searching through?
  - Problem solving
    Action combinations (move 1, then move 3, then move 2...)
  - Natural language
    Ways to map words to parts of speech
  - Computer vision
    Ways to map features to object model
  - Machine learning
    Possible concepts that fit examples seen so far
  - Motion planning
    Sequence of moves to reach goal destination
- An intelligent agent is trying to find a set or sequence of actions to achieve a goal
- This is a goal-based agent

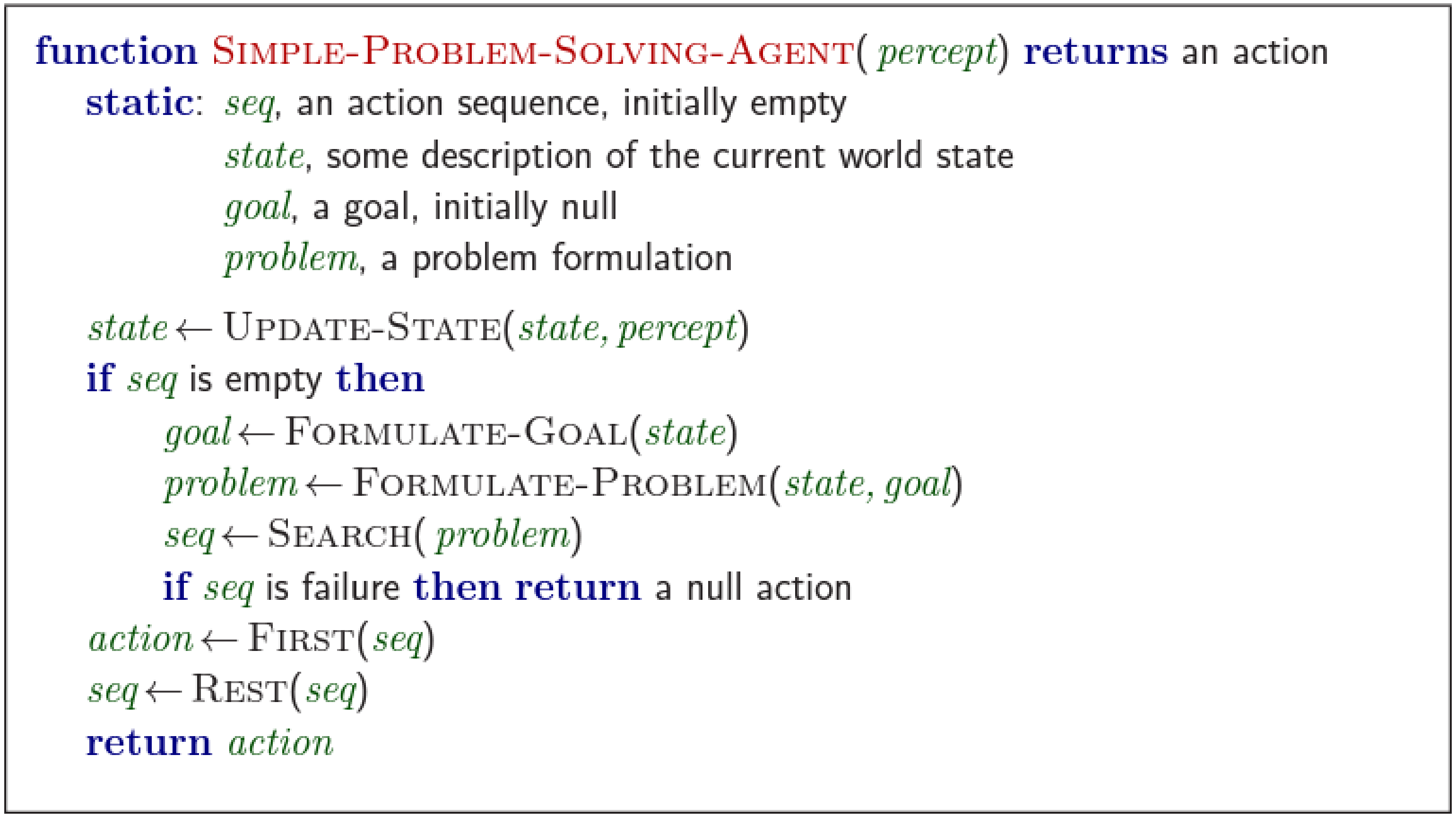
Well-defined problem

- The solution of many problems (e.g. noughts and crosses, timetabling, chess) can be described by finding a sequence of actions that lead to a desirable goal. Each action changes the *state* and the aim is to find the sequence of actions and states that lead from the initial (start) state to a final (goal) state.

- A well-defined problem can be described by:

- **Initial state**: starting state of the problem
- **Goal state**: solution of the problem (or) where we are stopping the process.
- **Operator or successor function** - for any state x returns s(x), the set of states reachable from x with one action
- **State space** - all states reachable from initial by any sequence of actions
- **Path** - sequence through state space
- **Path cost** - function that assigns a cost to a path. Cost of a path is the sum of costs of individual actions along the path
- **Goal test** - test to determine if at goal state

Problem-solving Agent



```
function SIMPLE-PROBLEM-SOLVING-AGENT( percept) returns an action
    static: seq, an action sequence, initially empty
            state, some description of the current world state
            goal, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, percept)
    if seq is empty then
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH( problem)
        if seq is failure then return a null action
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

Assumptions

We will assume environment is

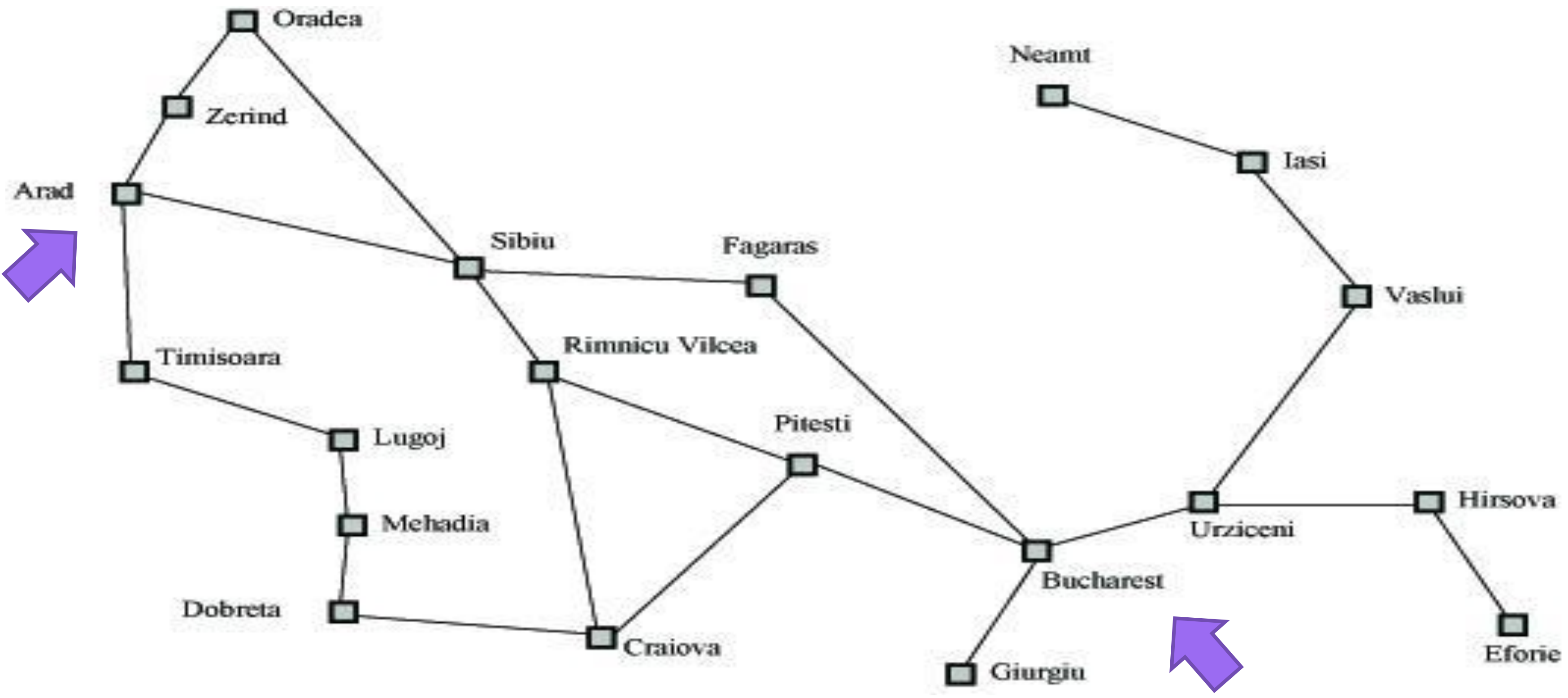
- Static
- Fully Observable
- Deterministic
- Discrete

The rational agent designer's goal

Goal of AI practitioner who designs rational agents:
given a PEAS task environment

1. Construct agent function f that maximizes (the expected value of) the performance measure,
2. Design an agent program that implements f on a particular architecture

Search Example: Romania



Search example: Romania

- On holiday in Romania; currently in Arad
  - Flight leaves tomorrow from Bucharest
- Formulate goal
  - Be in Bucharest
- Formulate search problem
  - States: various cities
  - Actions: drive between cities
  - Performance measure: minimize distance
- Find solution:
  - Sequence of cities; e.g. Arad, Sibiu, … Fagaras , Bucharest,

Search Space Definitions

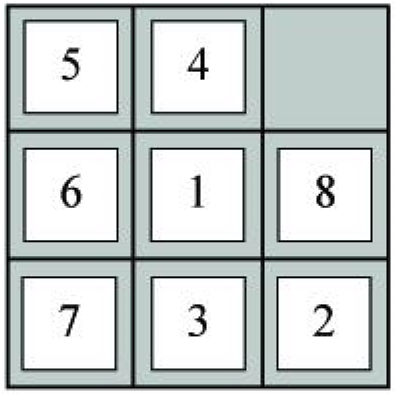
- Problem formulation
  - Describe a general problem as a search problem
- Solution
  - Sequence of actions that transitions the world from the initial state to a goal state
- Solution cost (additive)
  - Sum of the cost of operators
  - Alternative: sum of distances, number of steps, etc.
- Search
  - Process of looking for a solution
  - Search algorithm takes problem as input and returns solution
  - We are searching through a space of possible states
- Execution
  - Process of executing sequence of actions (solution)

Problem Formulation

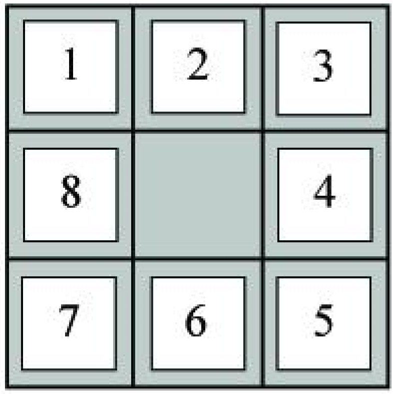
A search problem is defined by the

1. Initial state (e.g., Arad)
2. Operators (e.g., Arad -> Zerind, Arad -> Sibiu, etc.)
3. Goal test (e.g., at Bucharest)
4. Solution cost (e.g., path cost)

Example Problems – Eight Puzzle



Start State



Goal State

States: tile locations

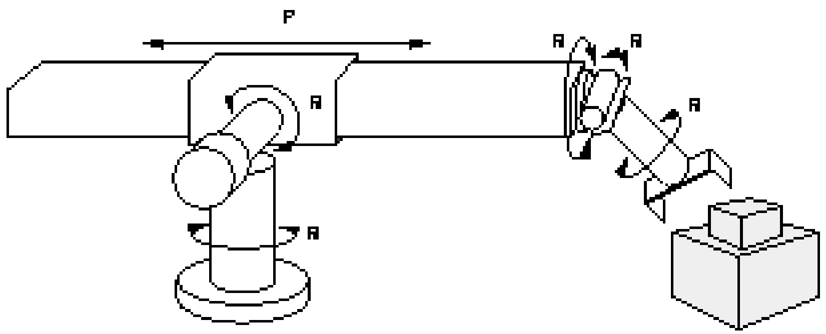
Initial state: one specific tile configuration

Operators: move blank tile left, right, up, or down

Goal: tiles are numbered from one to eight around the square

Path cost: cost of 1 per move (solution cost same as number of most or path length)

Example Problems – Robot Assembly



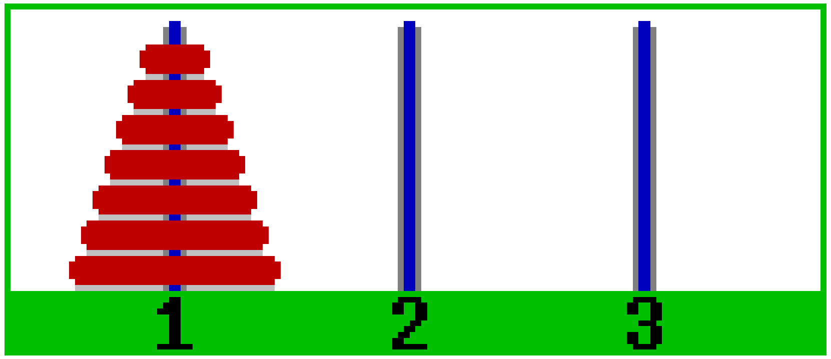
States: real-valued coordinates of
• robot joint angles
• parts of the object to be assembled

Operators: rotation of joint angles

Goal test: complete assembly

Path cost: time to complete assembly

Example Problems – Towers of Hanoi



States: combinations of poles and disks

Operators: move disk x from pole y to pole z subject to constraints
• cannot move disk on top of smaller disk
• cannot move disk if other disks on top

Goal test: disks from largest (at bottom) to smallest on goal pole

Path cost: 1 per move

Example Problems – Rubik's Cube

States: list of colors for each cell on each face

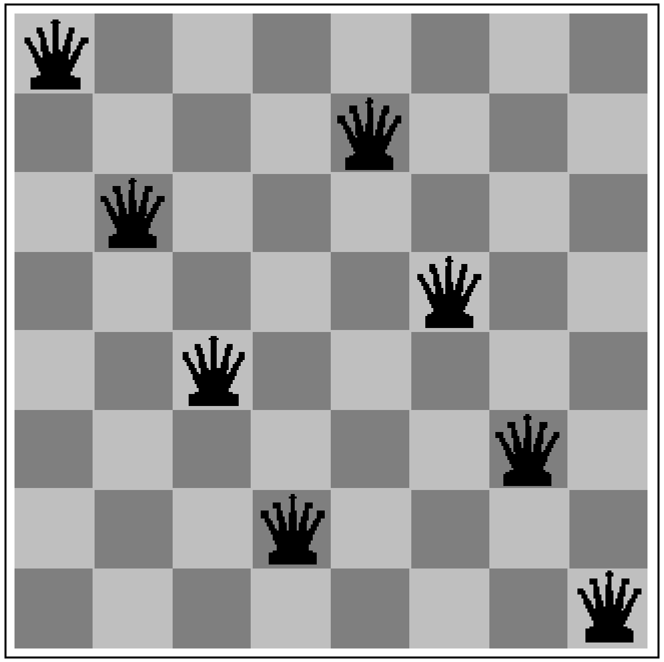
Initial state: one specific cube configuration

Operators: rotate row x or column y on face z direction a

Goal: configuration has only one color on each face

Path cost: 1 per move

Example Problems – Eight Queens



States: locations of 8 queens on chess board

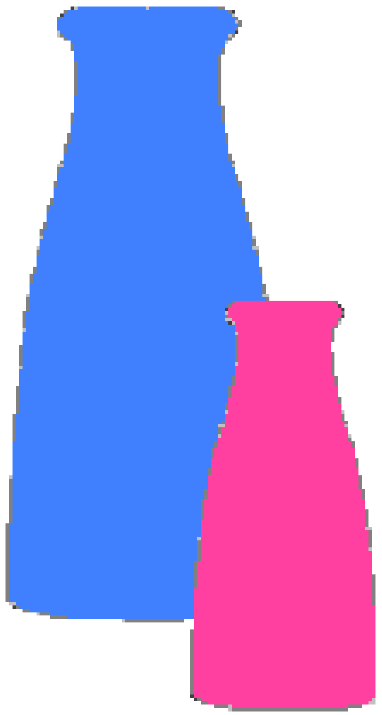
Initial state: one specific queens configuration

Operators: move queen x to row y and column z

Goal: no queen can attack another (cannot be in same row, column, or diagonal)

Path cost: 0 per move

Example Problems –Water Jug



States: Contents of 4-gallon jug and 3-gallon jug

Initial state: (0,0)

Operators:
- fill jug x from faucet
- pour contents of jug x in jug y until y full
- dump contents of jug x down drain

Goal: (2,n)

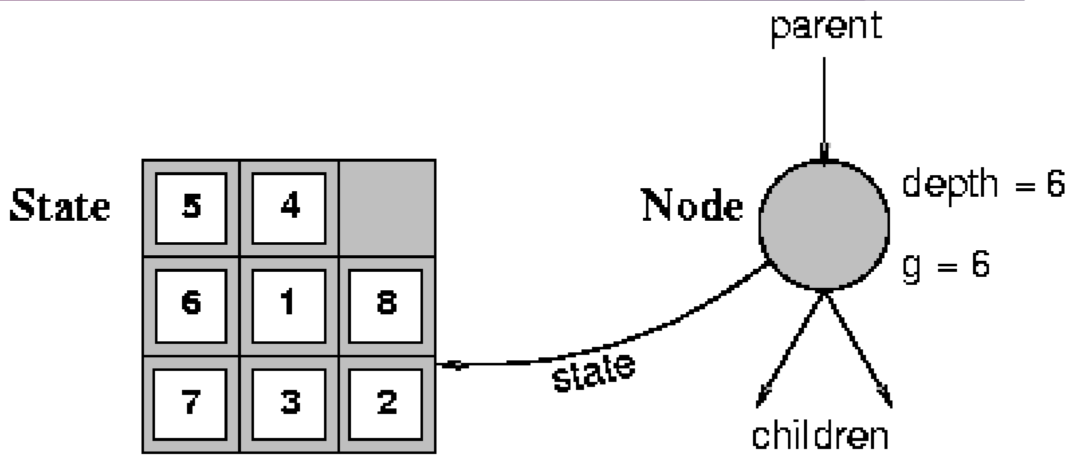
Path cost: 1 per fill

Sample Search Problems

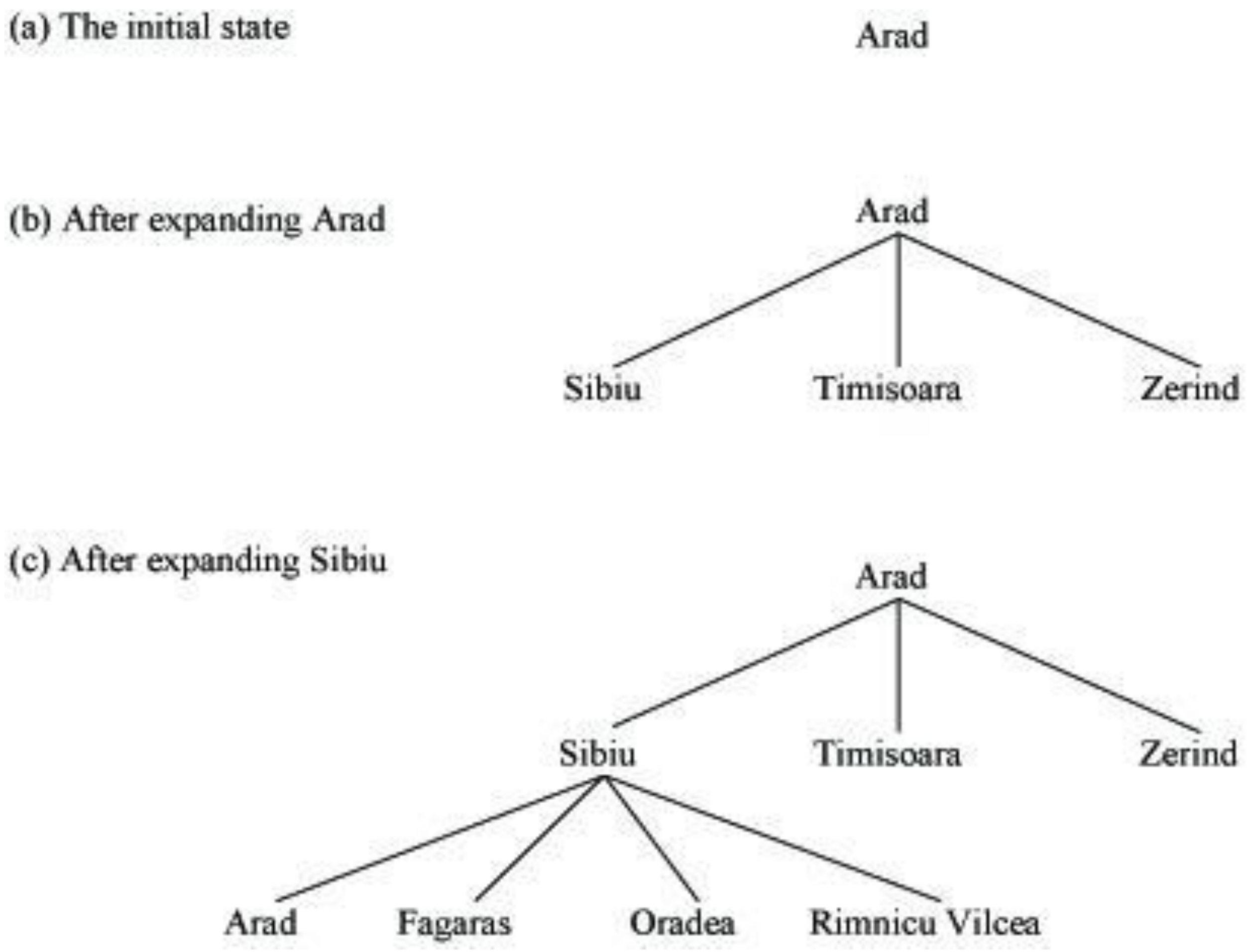
- Graph coloring
- Protein folding
- Game playing
- Airline travel
- Proving algebraic equalities
- Robot motion planning

Visualize Search Space as a Tree

- States are nodes
- Actions are edges
- Initial state is root
- Solution is path from root to goal node
- Edges sometimes have associated costs
- States resulting from operator are children



Search Problem Example (as a tree)



Introduction to Search Algorithms in AI

Introduction to Search Algorithms in AI

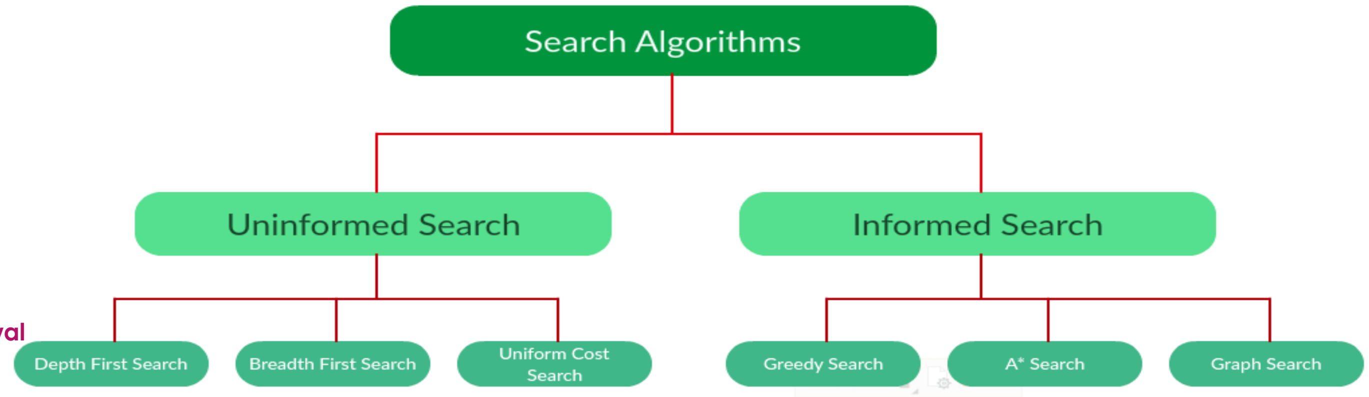
- Artificial Intelligence is basically the replication of human intelligence through computer systems or machines. It is done through the process of acquisition of knowledge or information and the addition of rules that are used by information, i.e. learning, and then using these rules to derive conclusions (i.e. reasoning) and then self- correction.

- **Properties of Search Algorithms**

- **Completeness:** A search algorithm is complete when it returns a solution for any input if at least one solution exists for that particular input.
- **Optimality:** If the solution deduced by the algorithm is the best solution, i.e. it has the lowest path cost, then that solution is considered as the optimal solution.
- **Time and Space Complexity:** Time complexity is the time taken by an algorithm to complete its task, and space complexity is the maximum storage space needed during the search operation. A good search algorithm takes less time and space to do its work.

Search Strategies

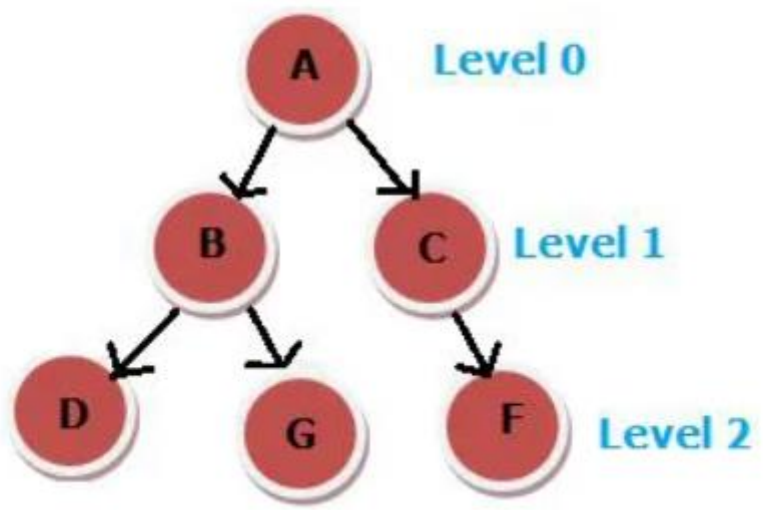
- Search strategies differ only in QueuingFunction
- Features by which to compare search strategies
  - Completeness (always find solution)
  - Cost of search (time and space)
  - Cost of solution, optimal solution
  - Make use of knowledge of the domain
    - "uninformed search" vs. "informed search"



Uninformed Search Algorithms

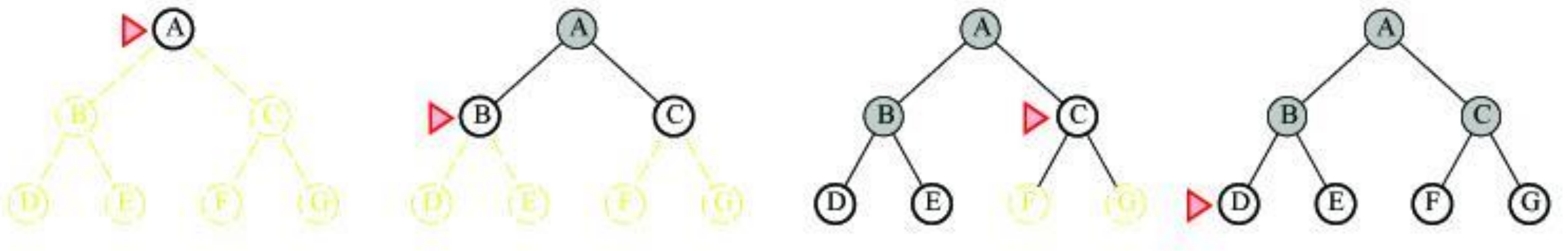
- Also known as blind search algorithm
- It has no additional information about goal state other than the one provided in the problem definition.
- It just knows how to traverse or visit the nodes in the tree.
- In such algorithm, machine blindly follows the technique irrespective of whether it is right or wrong, efficient or inefficient.

Breadth-First Search(BFS)



- In breadth-first search, the tree or the graph is traversed breadthwise, i.e. it starts from a node called search key and then explores all the neighbouring nodes of the search key at that depth-first and then moves to the next level nodes.
- It is implemented using the queue data structure that works on the concept of first in first out (FIFO). It is a complete algorithm as it returns a solution if a solution exists.
- The time complexity for breadth-first search is b^d where b (branching factor) is the average number of child nodes for any given node and d is depth.
- The disadvantage of this algorithm is that it requires a lot of memory space because it has to store each level of nodes for the next one. It may also check duplicate nodes.
- **Example:** If the search starts from root node A to reach goal node G, then it will traverse A-B-C-D-G. It traverses level wise, i.e. explores the shallowest node first.

BFS Examples



Poorva Agrawal

Analysis

- Assume goal node at level d with constant branching factor b

- Time complexity (measured in #nodes generated)
  - 1 (1st level) + b (2nd level) + b^2 (3rd level) + … + b^d (goal level) + (b^{d+1} – b) = O(b^d+1)

- This assumes goal on far right of level
- Space complexity
  - At most majority of nodes at level d + majority of nodes at level d+1 = O(b^{d+1})
  - Exponential time and space

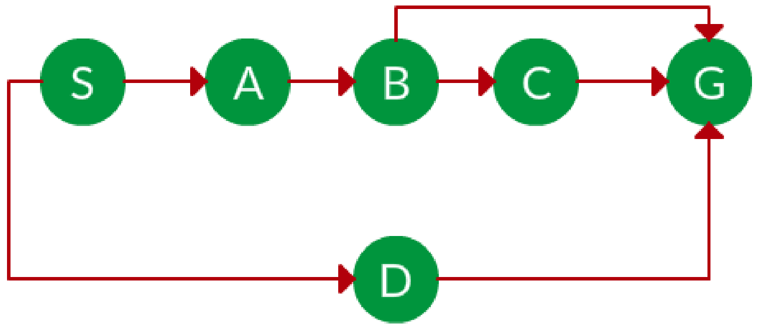
- Features
  - Simple to implement
  - Complete
  - Finds shortest solution (not necessarily least-cost unless all operators have equal cost)

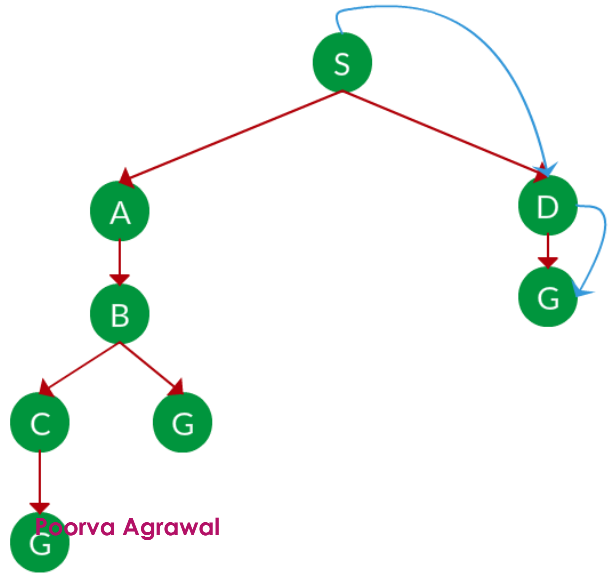
Analysis

- See what happens with b=10
  - expand 10,000 nodes/second
  - 1,000 bytes/node

Depth	Nodes	Time	Memory
2	1110	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
15	10^{15}	3,523 years	1 exabyte

Example





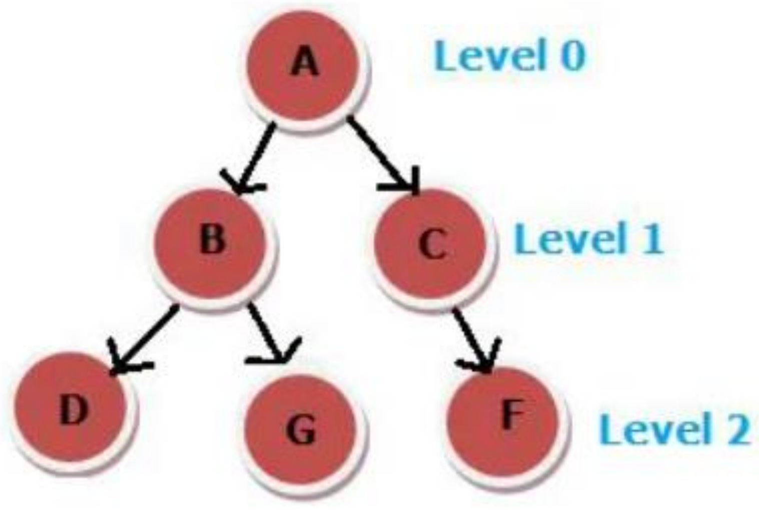
Which solution would BFS find to move from node S to node G if run on the graph below?

Solution. The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.

Path: S -> D -> G

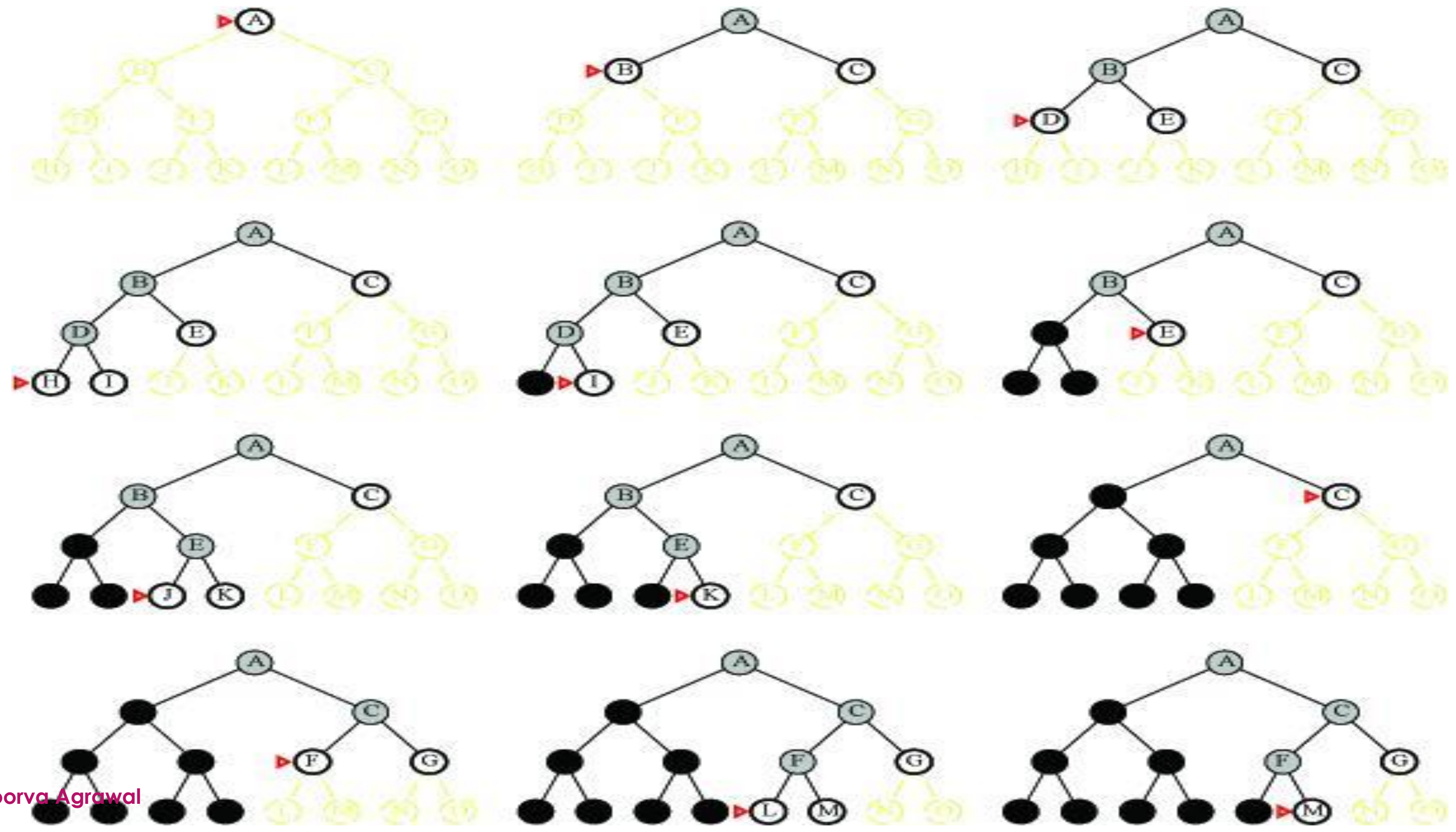
Source: https://www.geeksforgeeks.org/search-algorithms-in-ai/

Depth-First Search(DFS)



- In depth-first search, the tree or the graph is traversed depth-wise, i.e. it starts from a node called search key and then explores all the nodes along the branch then backtracks. It is implemented using a stack data structure that works on the concept of last in first out (LIFO).
- The time complexity for breadth-first search is b^d where b (branching factor) is the average number of child nodes for any given node and d is depth.
- It stores nodes linearly hence less space requirement.
- The major disadvantage is that this algorithm may go in an infinite loop.
- **Example:** If the search starts from root node A to reach goal node G, then it will traverse A-B-D-G. It traverses depth-wise, i.e. explores the deepest node first.

DFS Examples



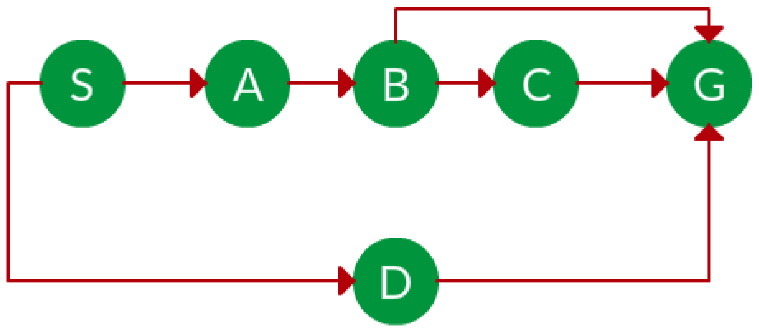
Analysis

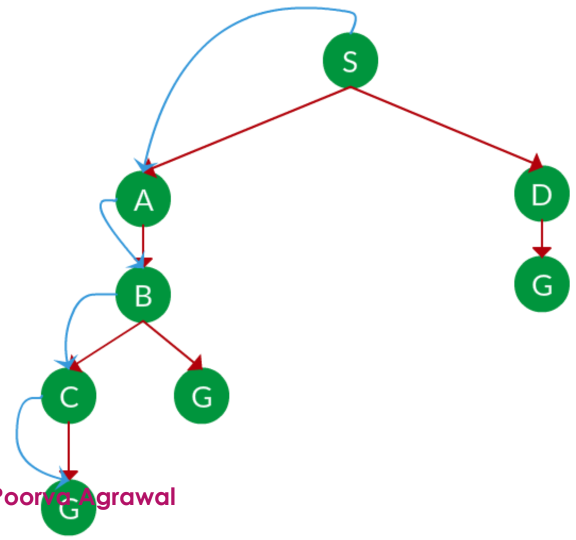
- Time complexity
  - In the worst case, search entire space
  - Goal may be at level d but tree may continue to level m, m>=d
  - $O(b^m)$
  - Particularly bad if tree is infinitely deep

- Space complexity
  - Only need to save one set of children at each level
  - 1 + b + b + … + b (m levels total) = O(bm)
  - For previous example, DFS requires 118kb instead of 10 petabytes for d=12 (10 billion times less)

- Benefits
  - May not always find solution
  - Solution is not necessarily shortest or least cost
  - If many solutions, may find one quickly (quickly moves to depth d)
  - Simple to implement
  - Space often bigger constraint, so more usable than BFS for large problems

Example





Which solution would DFS find to move from node S to node G if run on the graph below?

Solution: The equivalent search tree for the above graph is as follows. As DFS traverses the tree "deepest node first", it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.

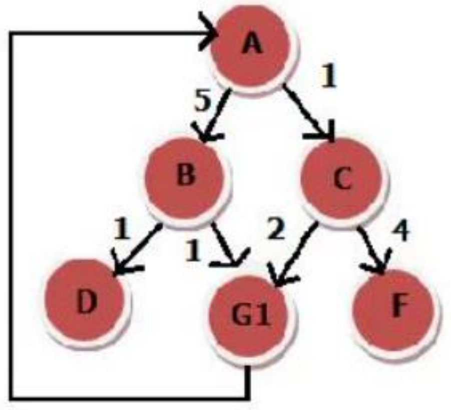
Path: S -> A -> B -> C -> G

Source: https://www.geeksforgeeks.org/search-algorithms-in-ai/

Depth-First Search

- QueueingFn adds the children to the front of the open list
- BFS emulates FIFO queue
- DFS emulates LIFO stack
- Net effect
  - Follow leftmost path to bottom, then backtrack
  - Expand deepest node first

Uniformed Cost Search



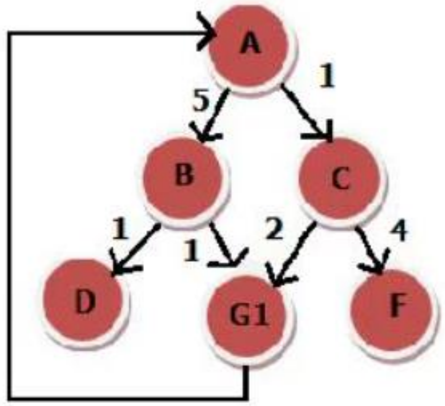
- Uniform cost search is different from both DFS and BFS. In this algorithm, the cost comes into the picture. There may be different paths to reach the goal, so the path with the least cost (cumulative sum of costs) is optimal. It traverses the path in the increasing order of cost. It is similar to the breadth-first search if the cost is the same for each transition.

- **Example:** If the search starts from node A of the graph to reach goal node G, then it will traverse A-C-G1. The cost will be 3.

Cost of a node is defined as:

```
cost(node) = cumulative cost of
all nodes from root
cost(root) = 0
```

Uniformed Cost Search



Graph Details : Nodes: {A, B, C, D, G1, F}

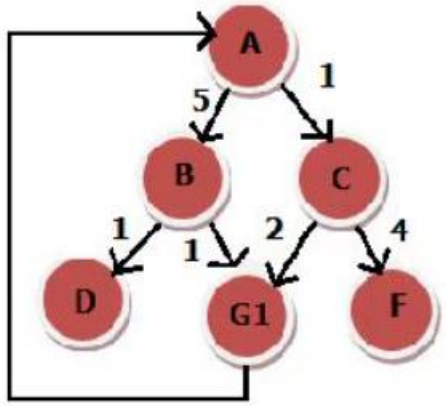
Edges with Costs:

- A → B (5) A → C (1) B → D (1)
- B → G1 (1) C → G1 (2) C → F (4)

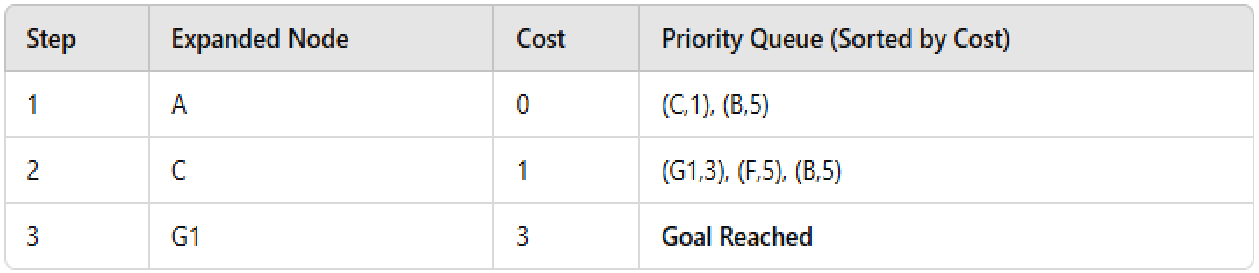
Steps of UCS using a Priority Queue :

- **Step 1:** Initialize the Priority Queue (PQ) with (A, 0).
- **Step 2**: Expand **A** (cost = 0), add neighbors to PQ:
  - (B, 5)
  - (C, 1) ✓ **(Lowest cost, expand next)**
- **Step 3:** Expand **C** (cost = 1), add neighbors:
  - (G1, 3) ✓
  - (F, 5)
  - PQ: (B, 5), (G1, 3), (F, 5)
- **Step 4:** Expand **G1** (cost = 3) ✓ **(Goal Found!)**
- **Final Path Found**, Optimal path: **A → C → G1** , Total cost: **3**

Uniformed Cost Search



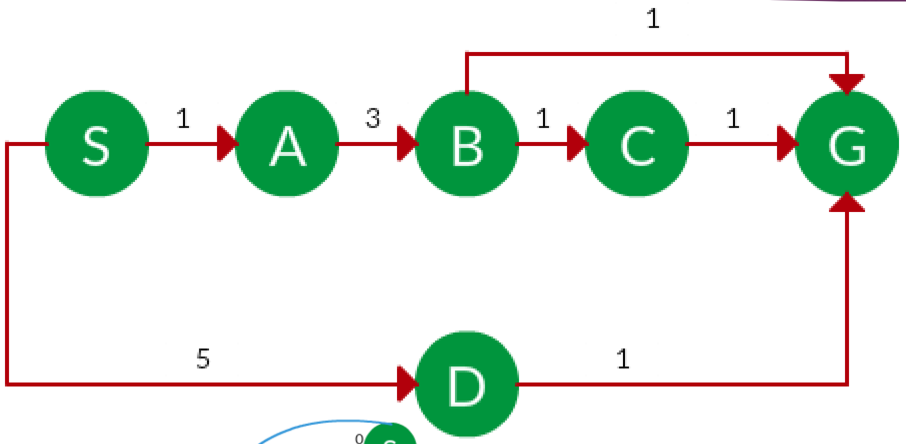
- Step-by-Step UCS Execution Table

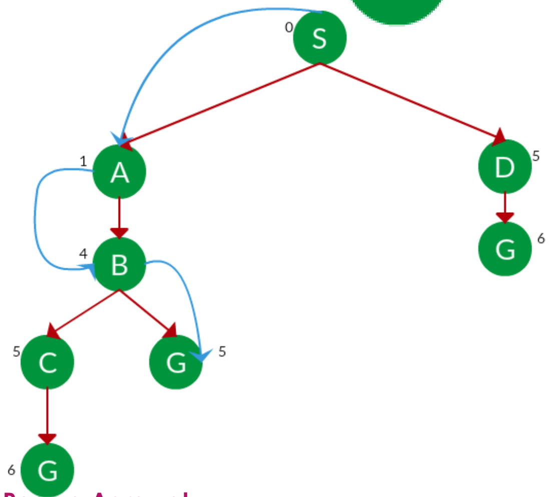


Step	Expanded Node	Cost	Priority Queue (Sorted by Cost)
1	A	0	(C,1), (B,5)
2	C	1	(G1,3), (F,5), (B,5)
3	G1	3	Goal Reached

- **Optimal Path Found**
- **Path:** A → C → G1
  Total Cost: 3
- Since the goal node **G1** is reached at step 3 with the lowest cost, the search terminates. UCS guarantees that this is the optimal solution.

Example





Which solution would UCS find to move from node S to node G if run on the graph below?

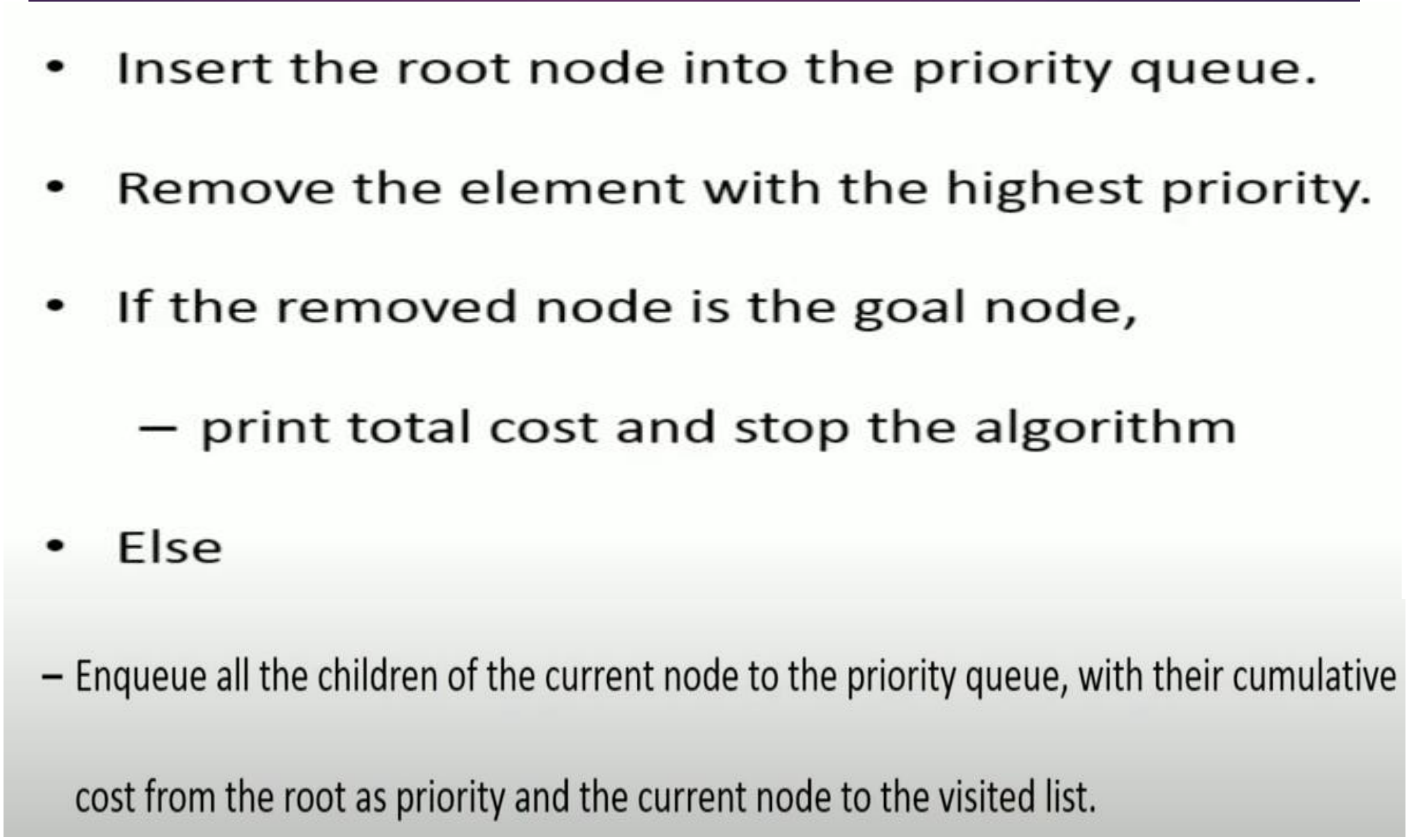
Solution: The equivalent search tree for the above graph is as follows. The cost of each node is the cumulative cost of reaching that node from the root. Based on the UCS strategy, the path with the least cumulative cost is chosen. Note that due to the many options in the fringe, the algorithm explores most of them so long as their cost is low, and discards them when a lower-cost path is found; these discarded traversals are not shown below. The actual traversal is shown in blue.

Path: S -> A -> B -> G
Cost: 5

Poorva Agrawal

Source: https://www.geeksforgeeks.org/search-algorithms-in-ai/

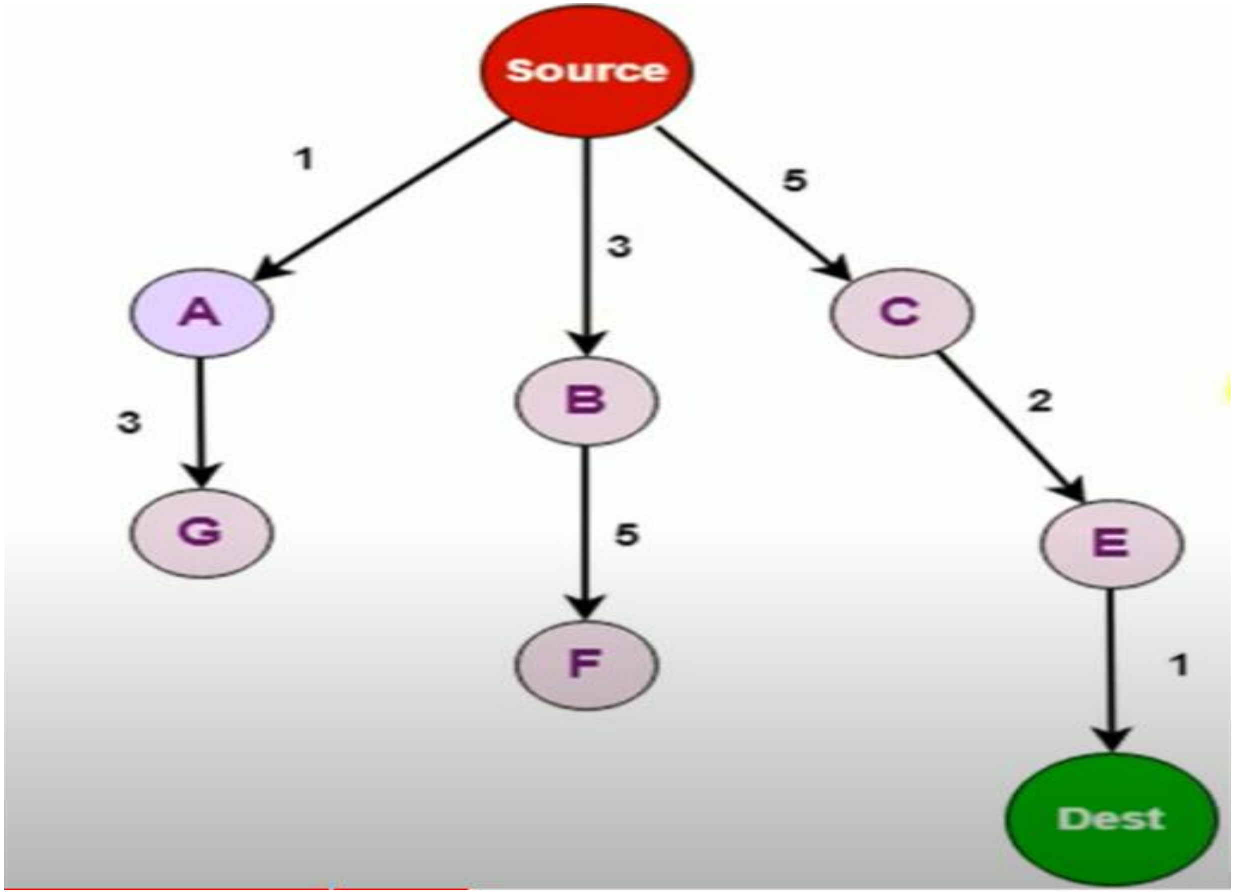
UCS Algorithm



- Insert the root node into the priority queue.
- Remove the element with the highest priority.
- If the removed node is the goal node,
  - print total cost and stop the algorithm
- Else

– Enqueue all the children of the current node to the priority queue, with their cumulative cost from the root as priority and the current node to the visited list.

Example of UCS



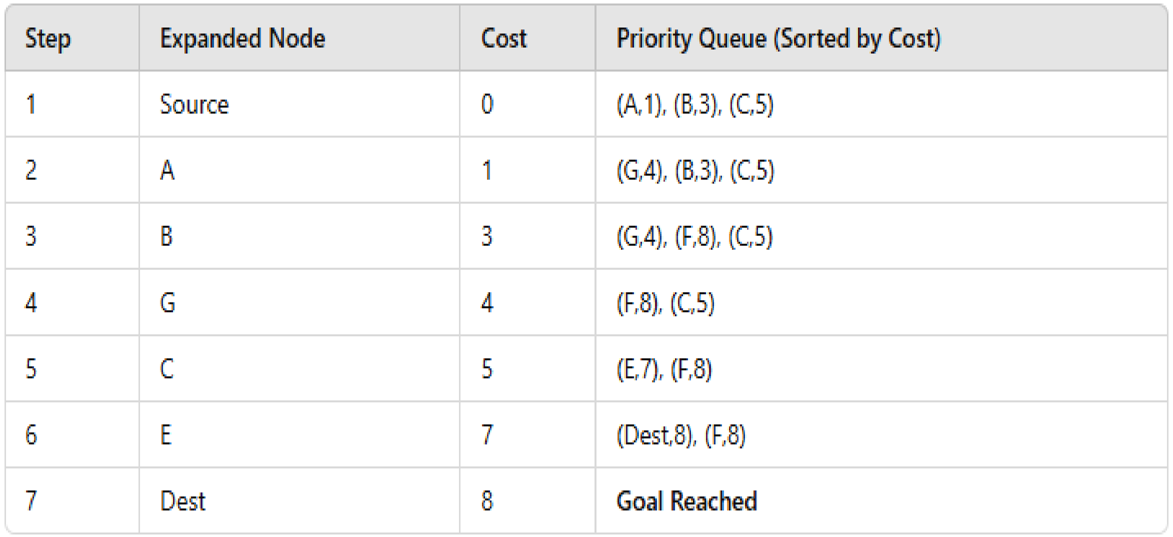
Step-by-Step UCS Execution

Graph Details

Nodes: {Source, A, B, C, G, F, E, Dest}

Edges with Costs:

- Source → A (1)
- Source → B (3)
- Source → C (5)
- A → G (3)
- B → F (5)
- C → E (2)
- E → Dest (1)



Step	Expanded Node	Cost	Priority Queue (Sorted by Cost)
1	Source	0	(A,1), (B,3), (C,5)
2	A	1	(G,4), (B,3), (C,5)
3	B	3	(G,4), (F,8), (C,5)
4	G	4	(F,8), (C,5)
5	C	5	(E,7), (F,8)
6	E	7	(Dest,8), (F,8)
7	Dest	8	**Goal Reached**

- **Optimal Path Found**
- **Path:** Source → C → E → Dest, **Total Cost: 8**
- Since the goal node **Dest** is reached at step 7 with the lowest cost, the search terminates. UCS guarantees this is the optimal solution.

Difference between Uninformed and Informed Search

Parameters	Informed Search	Uninformed Search
Known as	It is also known as Heuristic Search.	It is also known as Blind Search.
Using Knowledge	It uses knowledge for the searching process.	It doesn't use knowledge for the searching process.
Performance	It finds a solution more quickly.	It finds solution slow as compared to an informed search.
Completion	It may or may not be complete.	It is always complete.
Cost Factor	Cost is low.	Cost is high.
Time	It consumes less time because of quick searching.	It consumes moderate time because of slow searching.
Direction	There is a direction given about the solution.	No suggestion is given regarding the solution in
Examples of Algorithms	Greedy Search, A* Search, AO* Search, Hill Climbing Algorithm	DFS, BFS, Branch and Bound

Informed Search Algorithms

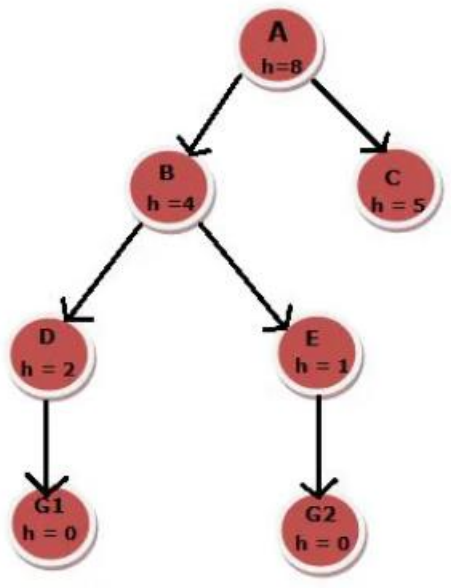
- Informed search algorithms have domain knowledge.
- It contains the problem description as well as extra information like how far is the goal node. It is also called the Heuristic search algorithm.
- It might not give the optimal solution always, but it will definitely give a good solution in a reasonable time.
- It can solve complex problems more easily than uninformed.
- Here, the algorithms have information on the goal state, which helps in more efficient searching. This information is obtained by something called a *heuristic.*

Search Heuristics

- In an informed search, a heuristic is a *function* that estimates how close a state is to the goal state.
- For example – Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal.)
- Different heuristics are used in different informed algorithms discussed below.
- In this section, we will discuss the following search algorithms.

1. Greedy Search
2. A* Tree Search

Greedy Search



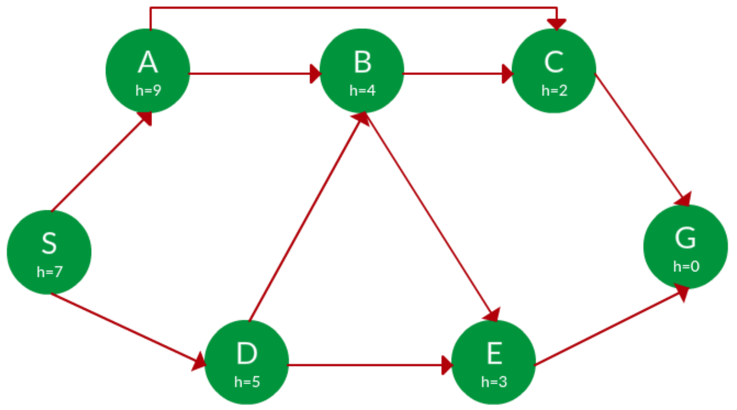
- In greedy search, we expand the node closest to the goal node. The "closeness" is estimated by a heuristic h(x).
- **Heuristic:** A heuristic h is defined as-
  h(x) = Estimate of distance of node x from the goal node.
  Lower the value of h(x), closer is the node from the goal.
- **Strategy:** Expand the node closest to the goal state, *i.e.* expand the node with a lower h value.
- This algorithm is implemented through the priority queue. It is not an optimal algorithm. It can get stuck in loops.
- **Example**: If we need to find the path from root node A to any goal state having minimum cost using greedy search, then the solution would be A-B-E-G2. It will start with B because it has less cost than C, then E because it has less cost than D and then G2.

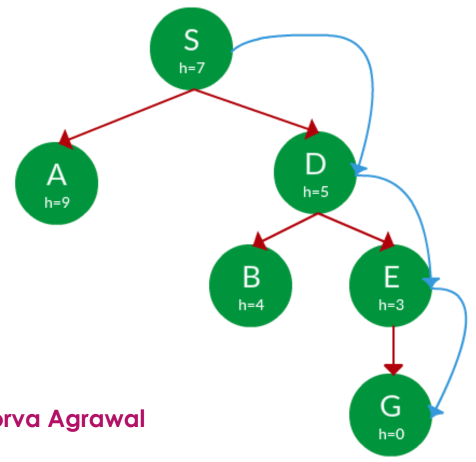
Source: https://www.educba.com/search-algorithms-in-ai/

Search Function – Uninformed Searches

```
Open = initial state                // open list is all generated states
                                    // that have not been "expanded"
While open not empty                // one iteration of search algorithm
  state = First(open)               // current state is first state in open
  Pop(open)                         // remove new current state from open
  if Goal(state)                    // test current state for goal condition
    return "succeed"                // search is complete
                                    // else expand the current state by
                                    // generating children and
                                    // reorder open list per search strategy
  else open = QueueFunction(open, Expand(state))
Return "fail"
```

Example





Question. Find the path from S to G using greedy search. The heuristic values h of each node below the name of the node.

Solution. Starting from S, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with a lower heuristic cost. Finally, from E, we go to G(h=0). This entire traversal is shown in the search tree below, in blue.

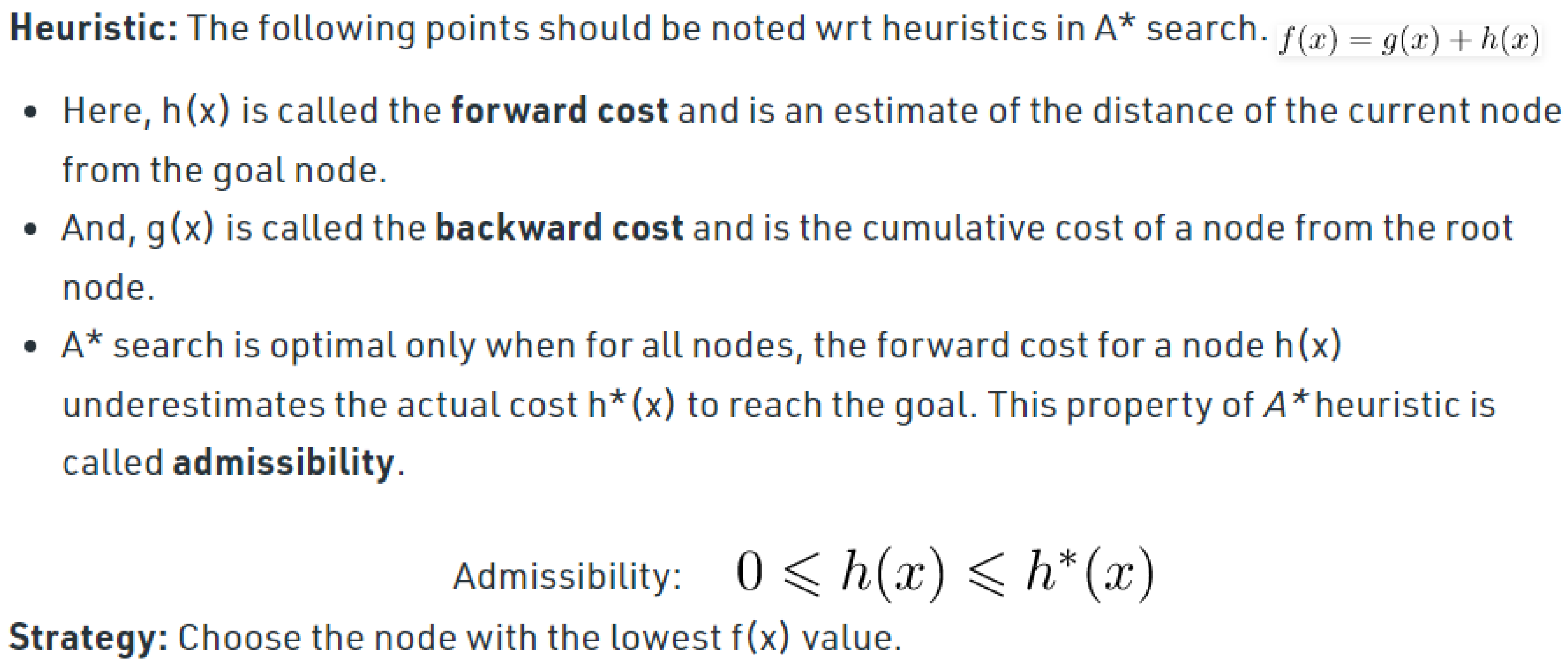
Path: S -> D -> E -> G

Source: https://www.geeksforgeeks.org/search-algorithms-in-ai/

A* Search

- A* Tree Search, or simply known as A* Search, combines the strengths of uniform-cost search and greedy search.
- In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in the greedy search, denoted by h(x). The summed cost is denoted by f(x).
- In this algorithm, the total cost (heuristic) which is denoted by f(x) is a sum of the cost in uniform cost search denoted by g(x) and cost of greedy search denoted by h(x).
- $f(x) = g(x) + h(x)$
- In this g(x) is the backward cost which is the cumulative cost from the root node to the current node and h(x) is the forward cost which is approximate of the distance of goal node and the current node.

A* Search Heuristics



Heuristic: The following points should be noted wrt heuristics in A* search. $f(x) = g(x) + h(x)$

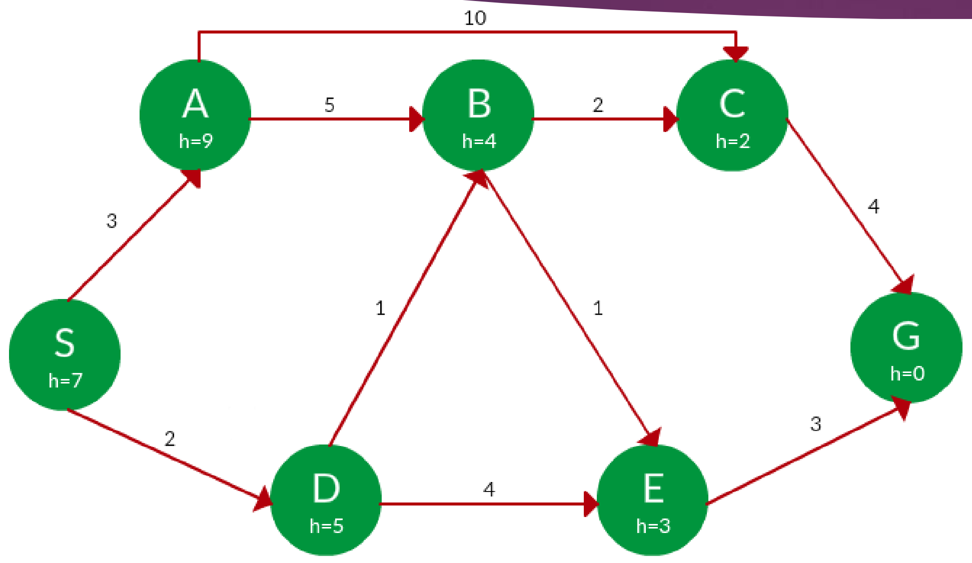
- Here, h(x) is called the **forward cost** and is an estimate of the distance of the current node from the goal node.
- And, g(x) is called the **backward cost** and is the cumulative cost of a node from the root node.
- A* search is optimal only when for all nodes, the forward cost for a node h(x) underestimates the actual cost h*(x) to reach the goal. This property of *A** heuristic is called **admissibility**.

Admissibility: $0 \leqslant h(x) \leqslant h^*(x)$

Strategy: Choose the node with the lowest f(x) value.

Source: https://www.geeksforgeeks.org/search-algorithms-in-ai/

Example

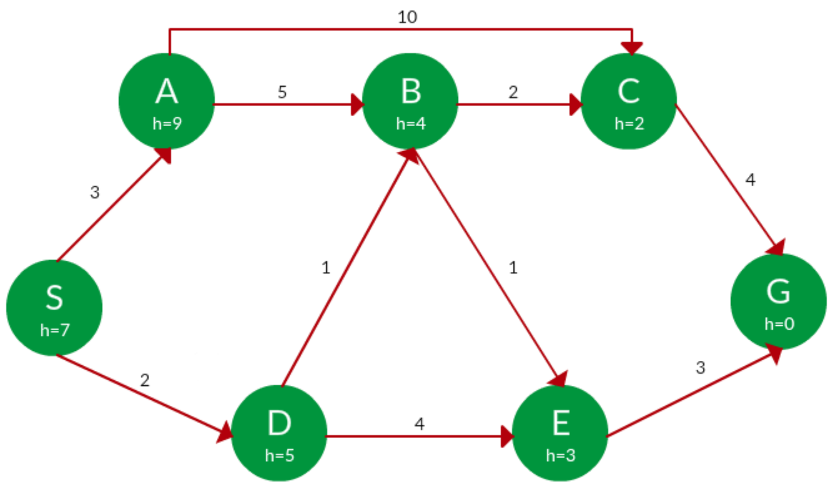


- **Question.** Find the path to reach from S to G using A* search.
- **Solution.** Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum. The entire work is shown in the table below.

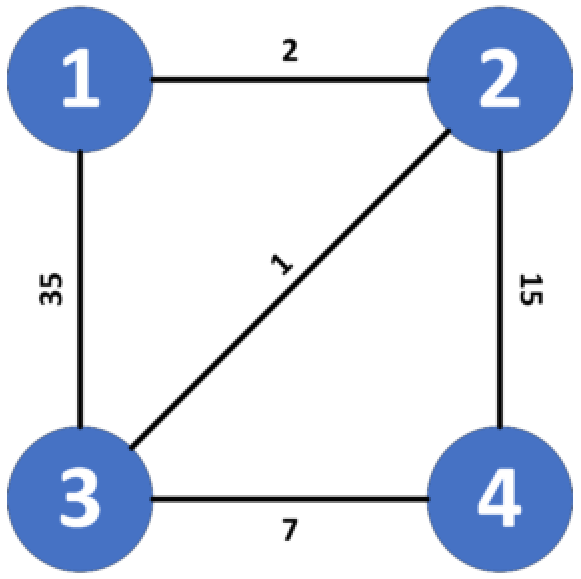
Example Continued

Path	h(x)	g(x)	f(x)
S	7	0	7
S -> A	9	3	12
S -> D	5	2	7
S -> D -> B	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C	2	3 + 2 = 5	7
S -> D -> B -> E	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G	**0**	**4 + 3 = 7**	**7**

- **Path:** S -> D -> B -> E -> G
  Cost: 7



Practice Numerical: UCS

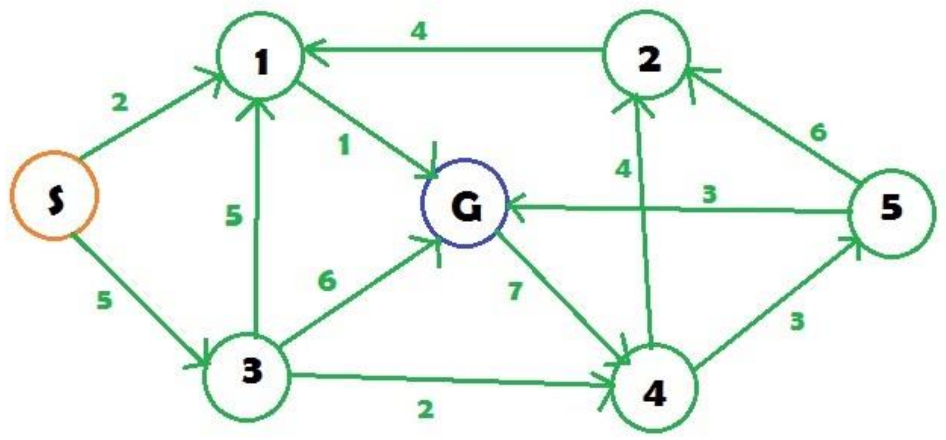


To go from node 1 to 4, there are four possible paths:

- 1,2,4: Cost: 17
- 1,3,4: Cost: 42
- 1,3,2,4: Cost: 51
- **1,2,3,4: Cost: 10**

Source: https://www.baeldung.com/cs/find-path-uniform-cost-search

Practice Numerical: UCS

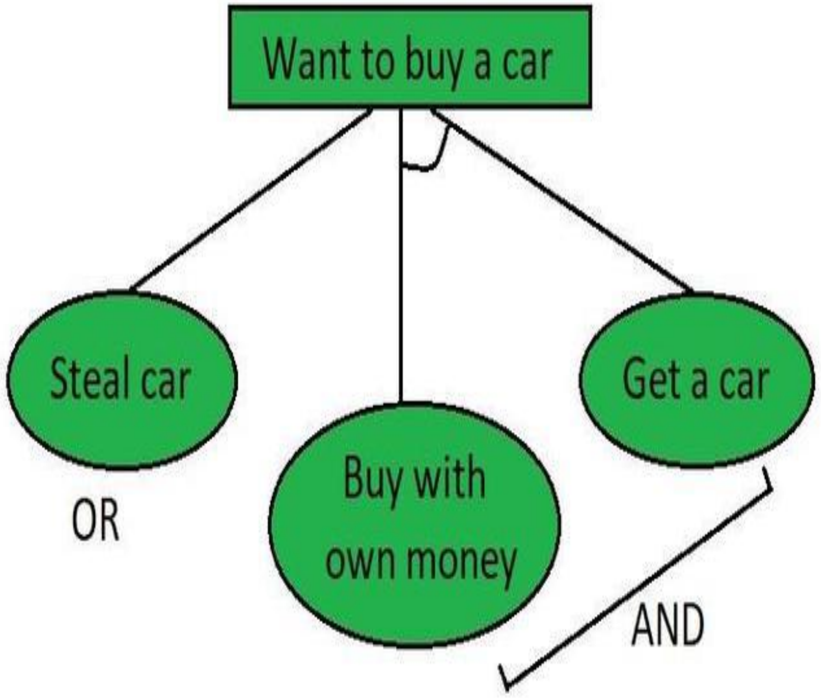


S is the starting state
G is the goal state

Source: https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/#:~:text=Uniform-Cost%20Search%20is%20mainly%20used%20in%20Artificial%20Intelligence.,Minimum%20cost%20from%20S%20to%20G%20is%20%3D3

AO * Algorithm

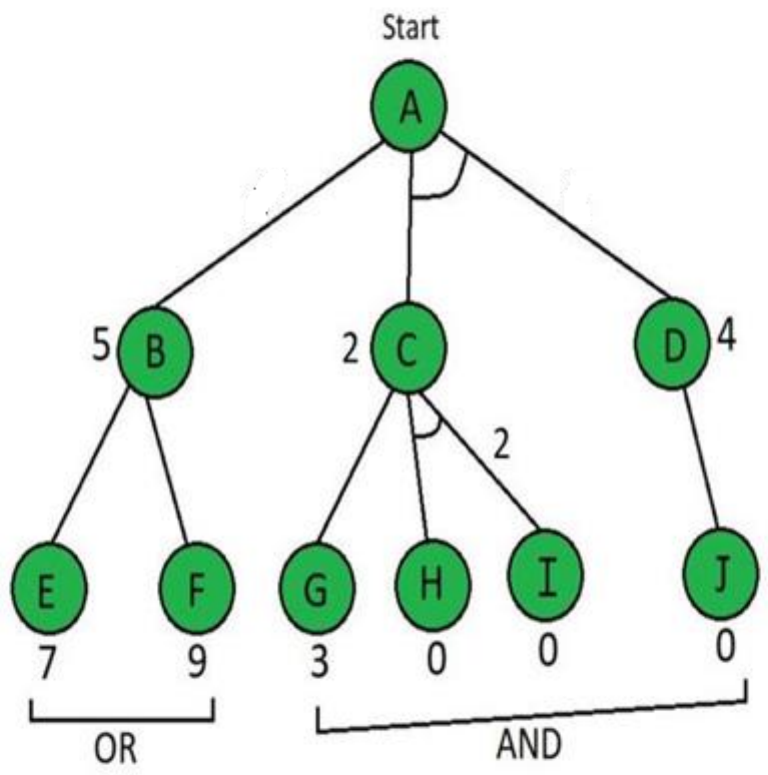
- The AO* method **divides** any given difficult **problem into a smaller group** of problems that are then resolved **using the AND-OR** graph concept.
- AND OR graphs are specialized graphs that are used in problems that can be divided into smaller problems.
- The AND side of the graph represents a set of tasks that must be completed to achieve the main goal, while the OR side of the graph represents different methods for accomplishing the same main goal.



Working of AO* algorithm:

- The informed search technique considerably reduces the algorithm's **time complexity**.
- The AO* algorithm is far more effective in searching AND-OR trees **than** the A* algorithm.
- The evaluation function in AO* is: **f(n) = g(n) + h(n)**
  f(n) = Actual cost + Estimated cost
  here,
  f(n) = The actual cost of traversal.
  g(n) = the cost from the initial node to the current node.
  h(n) = estimated cost from the current node to the goal state.

Practice Problem



- Node has the heuristic value i.e h(n). Edge length is considered as 1.
- With help of **f(n) = g(n) + h(n)** evaluation function,

Step 1: Initial Evaluation Using f(n) = g(n) + h(n)

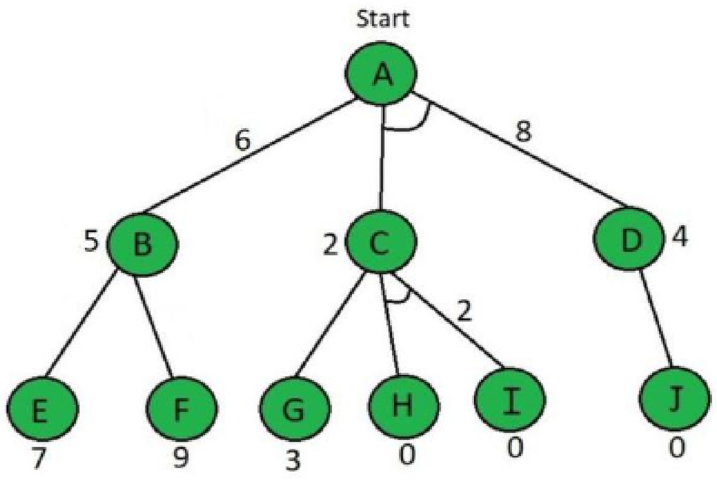
Starting from node **A**, we use the evaluation function:

```
f(A -> B) = g(B) + h(B)
          = 1 + 5  (g(n) = 1 is the default path cost)
          = 6
```

For the path involving **AND** nodes (**C** and **D**):

```
f(A -> C + D) = g(C) + h(C) + g(D) + h(D)
              = 1 + 2 + 1 + 4  (C & D are AND nodes)
              = 8
```

Since f(A -> B) = 6 is smaller than f(A -> C + D) = 8, we select the path A -> B.



AO* Algorithm (Step-1)

Source: https://www.geeksforgeeks.org

Step 2: Explore Node B

Next, we explore node **B**, and calculate the values for nodes **E** and **F**:

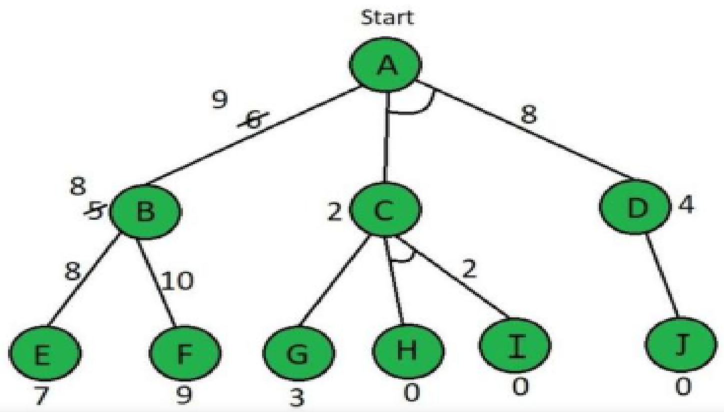
```
f(B -> E) = g(E) + h(E)
          = 1 + 7
          = 8

f(B -> F) = g(F) + h(F)
          = 1 + 9
          = 10
```

So, by above calculation B⇢E path is chosen which is minimum path, i.e f(B⇢E) because B’s heuristic value is different from its actual value The heuristic is updated and the minimum cost path is selected. The minimum value in our situation is 8. Therefore, the heuristic for A must be updated due to the change in B’s heuristic.

So we need to calculate it again.

*Thus, **f(B -> E) = 8** is chosen as the optimal path. Since **B**’s heuristic value differs from its actual cost, we update the heuristic for **A**.*

```
f(A -> B) = g(B) + updated h(B)
          = 1 + 8
          = 9
```



Now, we compare **f(A -> B) = 9** with **f(A -> C + D) = 8**. Since **f(A -> C + D)** is smaller, we explore this path and move to node **C**.

For node **C**:

```
f(C -> G) = g(G) + h(G)
          = 1 + 3
          = 4

f(C -> H + I) = g(H) + h(H) + g(I) + h(I)
              = 1 + 0 + 1 + 0  (H & I are AND nodes)
              = 2
```

f(C⇢H+I) is selected as the path with the lowest cost and the heuristic is also left unchanged because it matches the actual cost. Paths H & I are solved because the heuristic for those paths is **0,** but Path **A⇢D** needs to be calculated because it has an **AND**.

*The path **f(C -> H + I) = 2** is selected. Since the heuristic for **H** and **I** matches the actual cost (both are 0), these paths are considered solved. Next, we calculate the value for **A -> D** as it also has an **AND** node.*

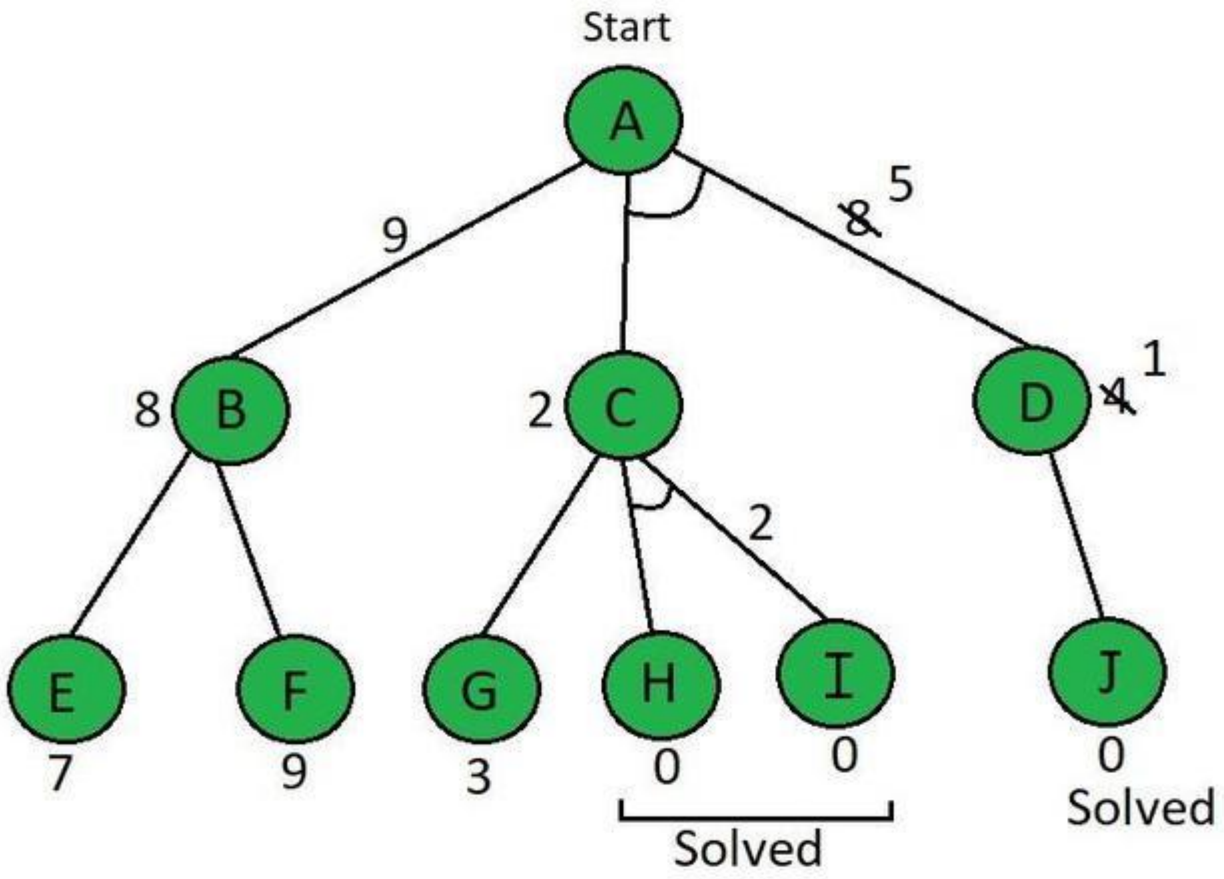
For node **D**:

```
f(D -> J) = g(J) + h(J)
          = 1 + 0
          = 1
```

After updating the heuristic for **D**, we recalculate:

```
f(A -> C + D) = g(C) + h(C) + g(D) + h(D)
              = 1 + 2 + 1 + 1
              = 5
```

Solved



THANK YOU!