

Problem Solving as Search & State Space Search

Dr. Mohamed Mostafa

AI 511 Foundations of Artificial Intelligence

Lecture 5



- **Uniform-Cost Strategy**

- **Heuristic Function**

- **Definition**
- **Examples of Evaluation function**
- **8-Puzzle**

- **More on Heuristic Search & Functions**

- **Heuristic Reduction**
 - A* Algorithm
 - Hill Climbing Strategy

Practice Exercises

Resources for this lecture

- **Chapter 3 (Structures & Strategies for State Space Search; sections 3.2, and 3.3) and Chapter 4 (Heuristic Search) from George F. Luger, "Artificial Intelligence: Structures and strategies for complex problem solving", " Sixth edition (2009), Pearson Education Limited..**

.. AND ..

- **Introduction to Artificial Intelligence Course,. Dr. Amr S. Ghoneim**

Uniform-Cost Strategy

- The goal of the Uniform-Cost Strategy is to find the lowest-cost path from a starting node to a goal node in a weighted graph or tree.
- Unlike other search algorithms like Breadth-First Search (BFS) or Depth-First Search (DFS), UCS prioritizes nodes based on their cumulative cost from the starting node.

cost of the path from the starting node to node

$C(n)$ =cost of the path from the starting node to node n

Uniform-Cost Strategy

Algorithm 1: Uniform-Cost Search

Data: s : the start node, $goal$: a function that can check if a node is a goal node, $successors$: a function that returns the nodes whose states we obtain by applying an action to the input node's state, c : a function that returns the cost of an edge between two nodes.

Result: The optimal path to a goal or *failure* if no node passes the goal test.

$g(s) \leftarrow 0$

// $g(node)$ is the cost of the path from s to the node.

$frontier \leftarrow$ a min-priority queue ordered by g , containing only s

$expanded \leftarrow$ an empty set

while *true* **do**

if $frontier$ is empty **then**

return *failure*

end

$v \leftarrow$ pop the lowest-cost node from $frontier$

if $goal(v) = true$ **then**

return v

end

$expanded \leftarrow expanded \cup \{v.state\}$

for $w \in successors(v)$ **do**

if $w.state \notin expanded$ and no frontier node represents $w.state$ **then**

$g(w) \leftarrow g(v) + c(v, w)$

 Insert w in $frontier$

else if there's a frontier node u such that $w.state = u.state$ and $g(v) + c(v, w) < g(u)$ **then**

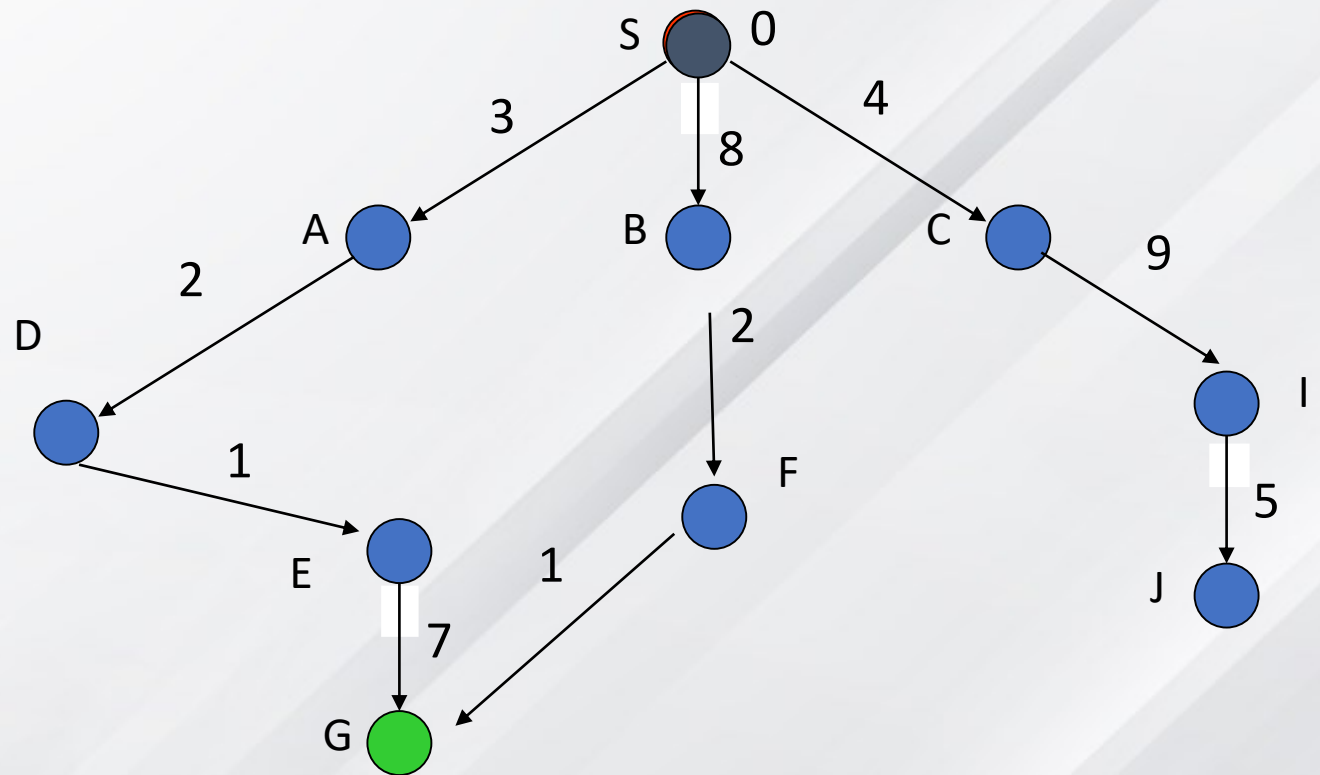
 Remove u from $frontier$

 Insert w in $frontier$

end

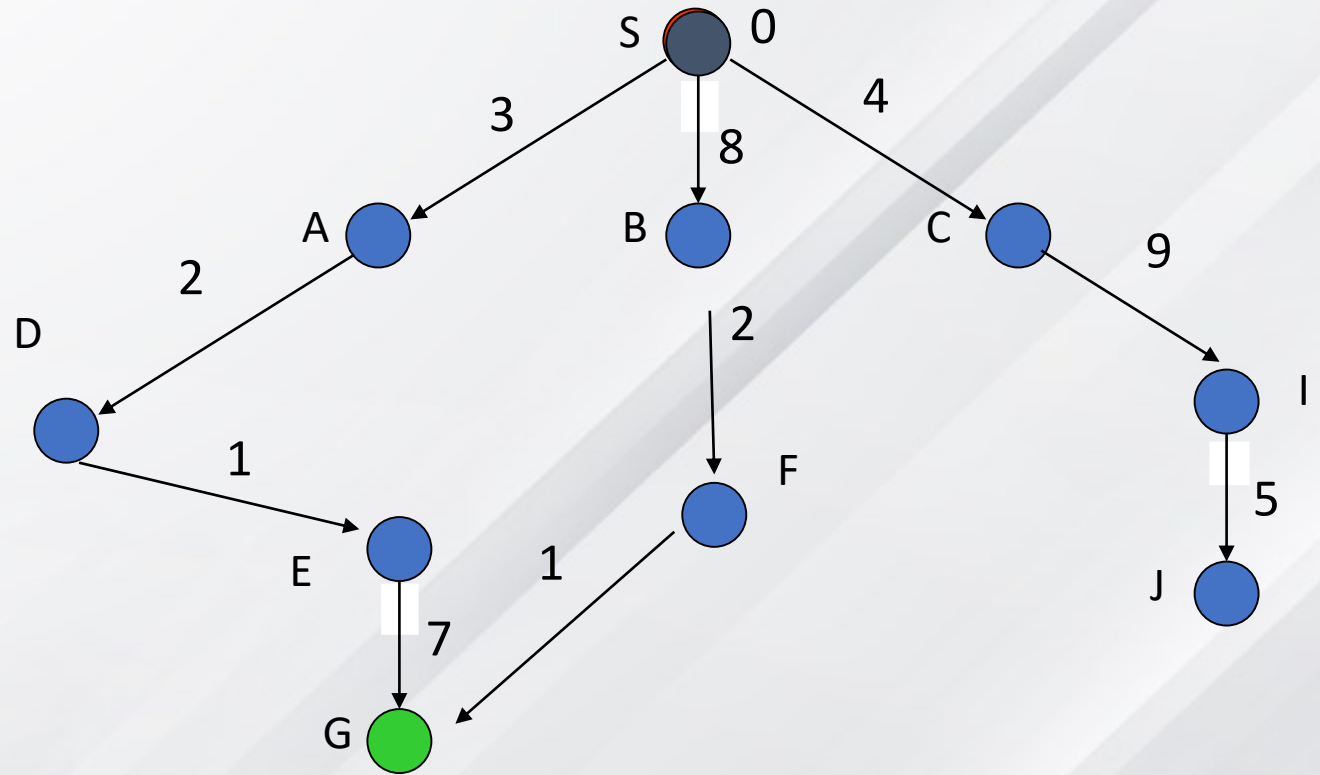
end

Uniform-Cost Strategy



Uniform-Cost Strategy

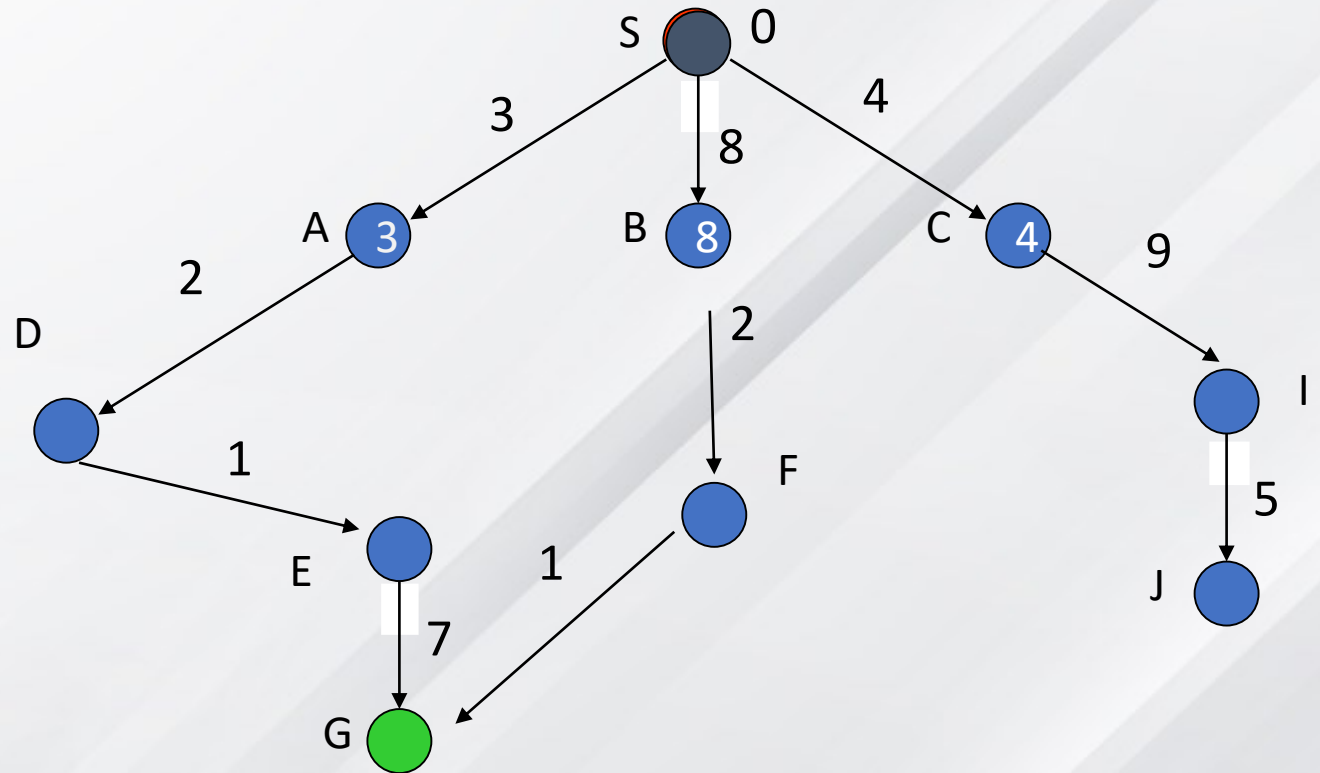
S0



Uniform-Cost Strategy

S0

S0
A3 B8 C4



Uniform-Cost Strategy

S0

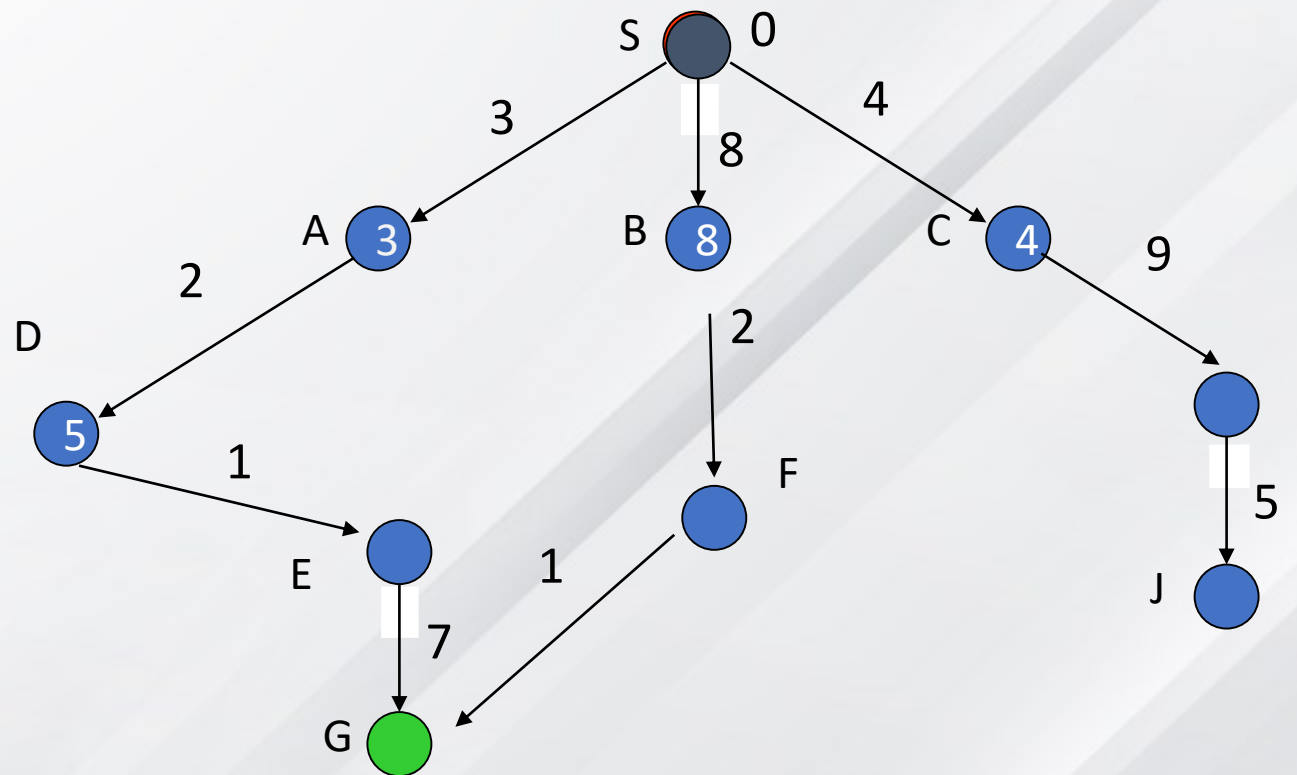
A3 B8 C4

D5 B8 C4

C4 D5 B8

S0
A3

Need to be sorted

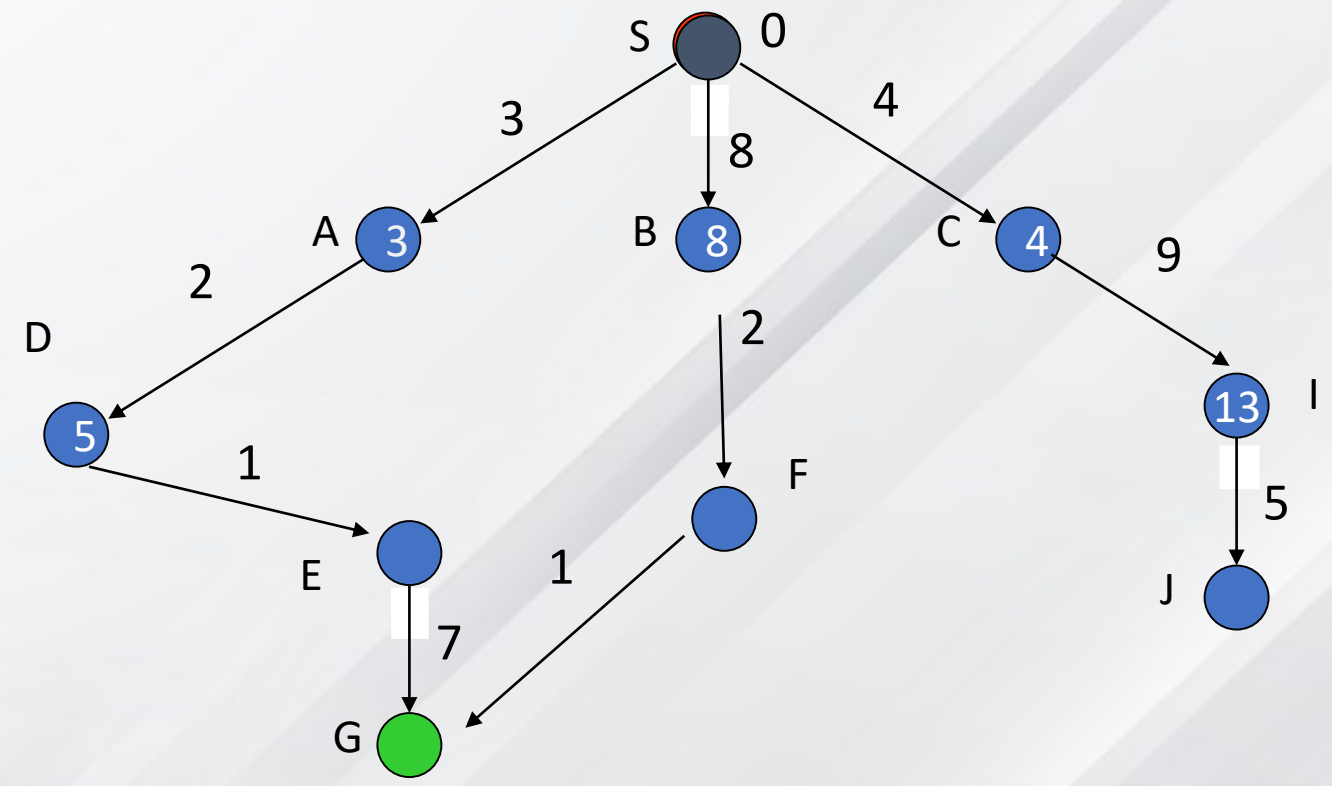


Uniform-Cost Strategy

S0
A3
C4

S0
A3 B8 C4
D5 B8 C4
C4 D5 B8
I13 D5 B8
D5 B8 I13

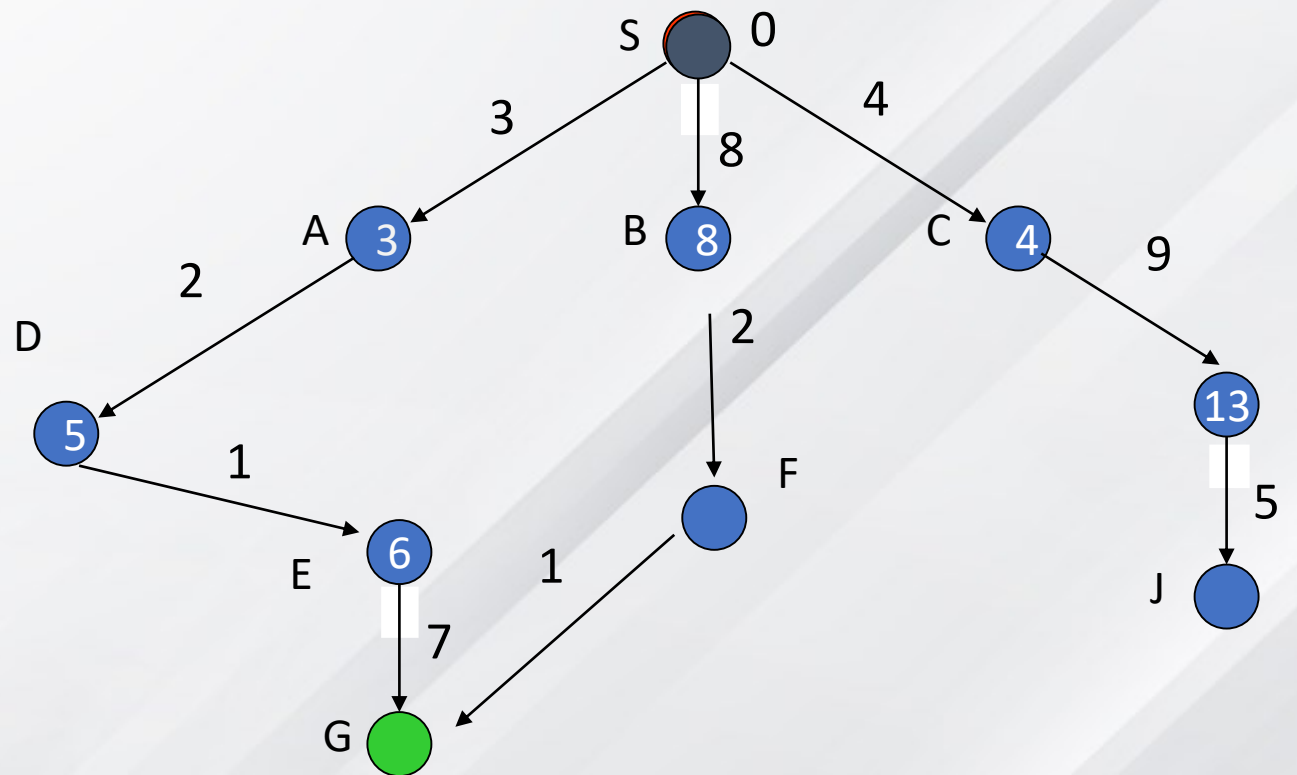
Need to be sorted



Uniform-Cost Strategy

S0
A3
C4
D5

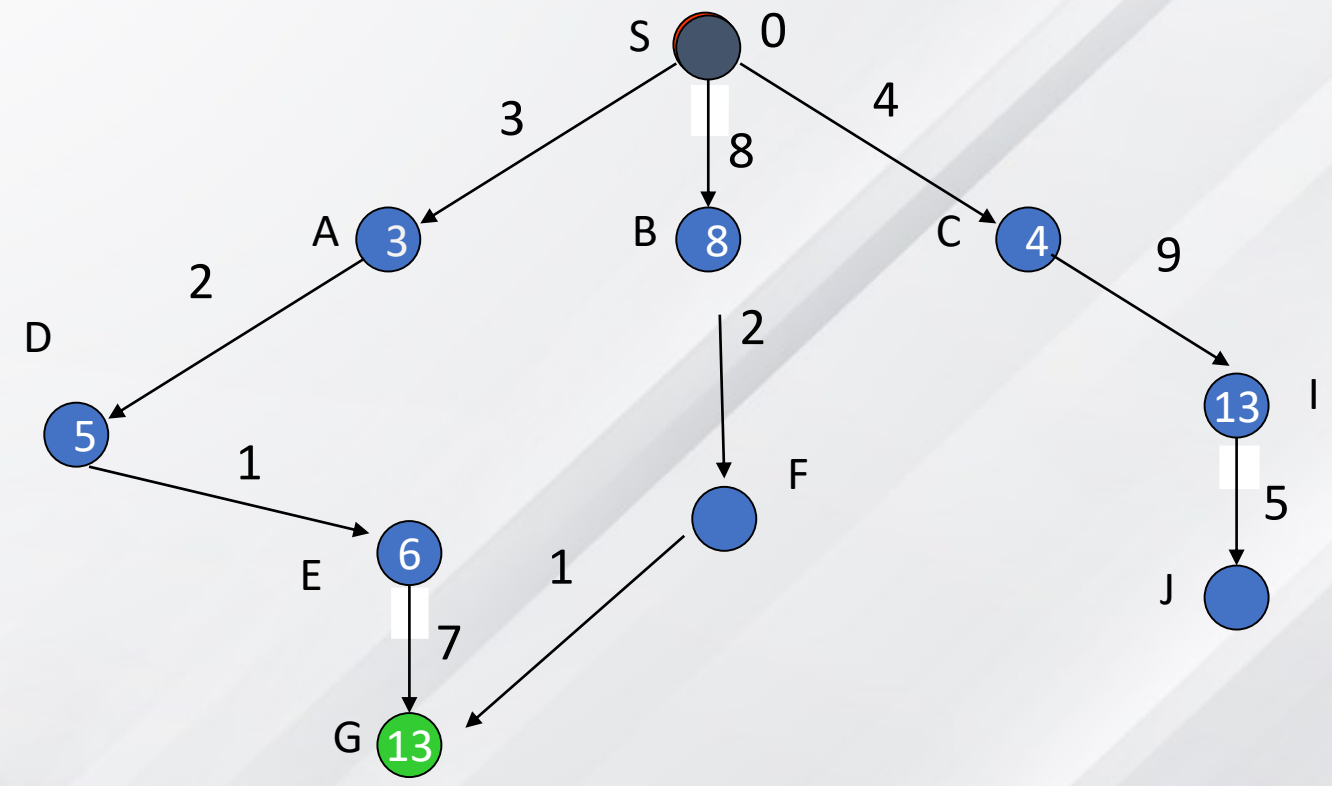
S0
A3 B8 C4
D5 B8 C4
C4 D5 B8
I13 D5 B8
D5 B8 I13
E6 B8 I13

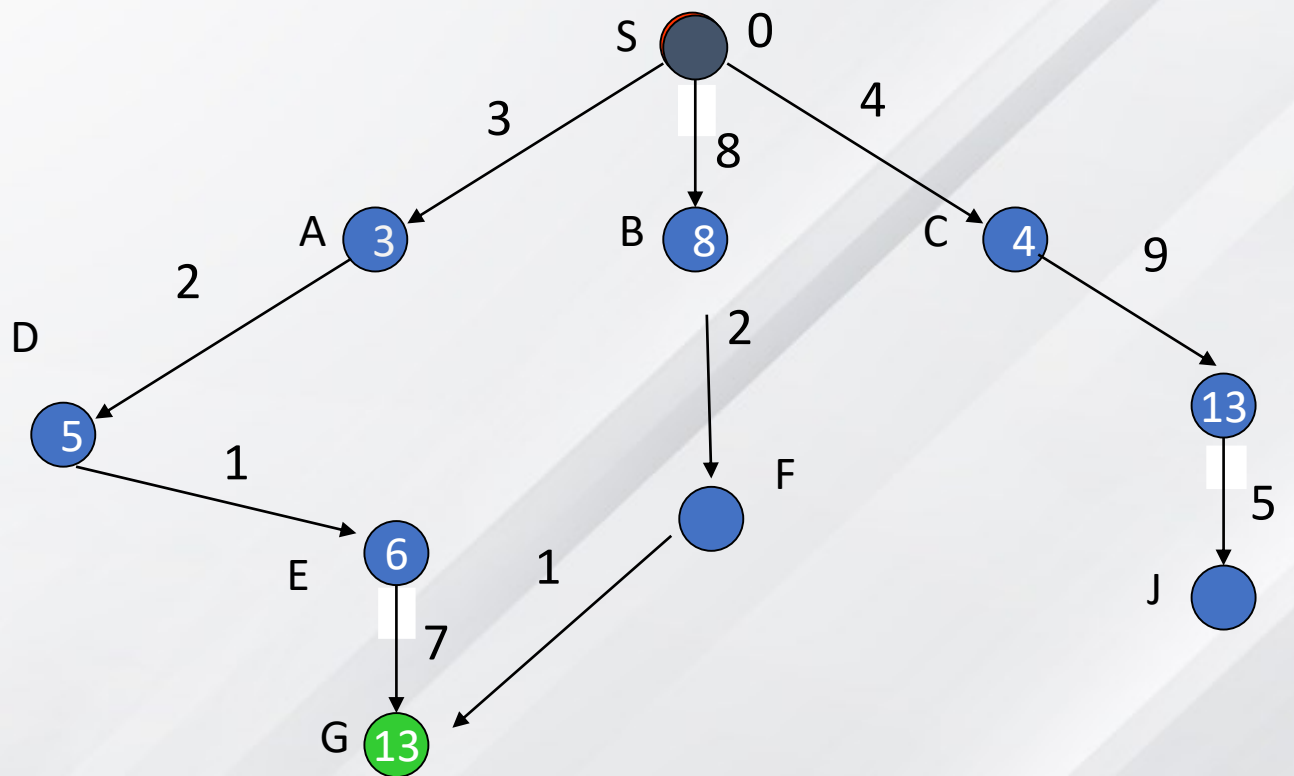


Uniform-Cost Strategy

S0
A3
C4
D5
E6

S0
A3 B8 C4
D5 B8 C4
C4 D5 B8
I13 D5 B8
D5 B8 I13
E6 B8 I13
G13 B8 I13





Uniform-Cost Strategy

S0

A3 B8 C4

D5 B8 C4

C4 D5 B8

I13 D5 B8

D5 B8 I13

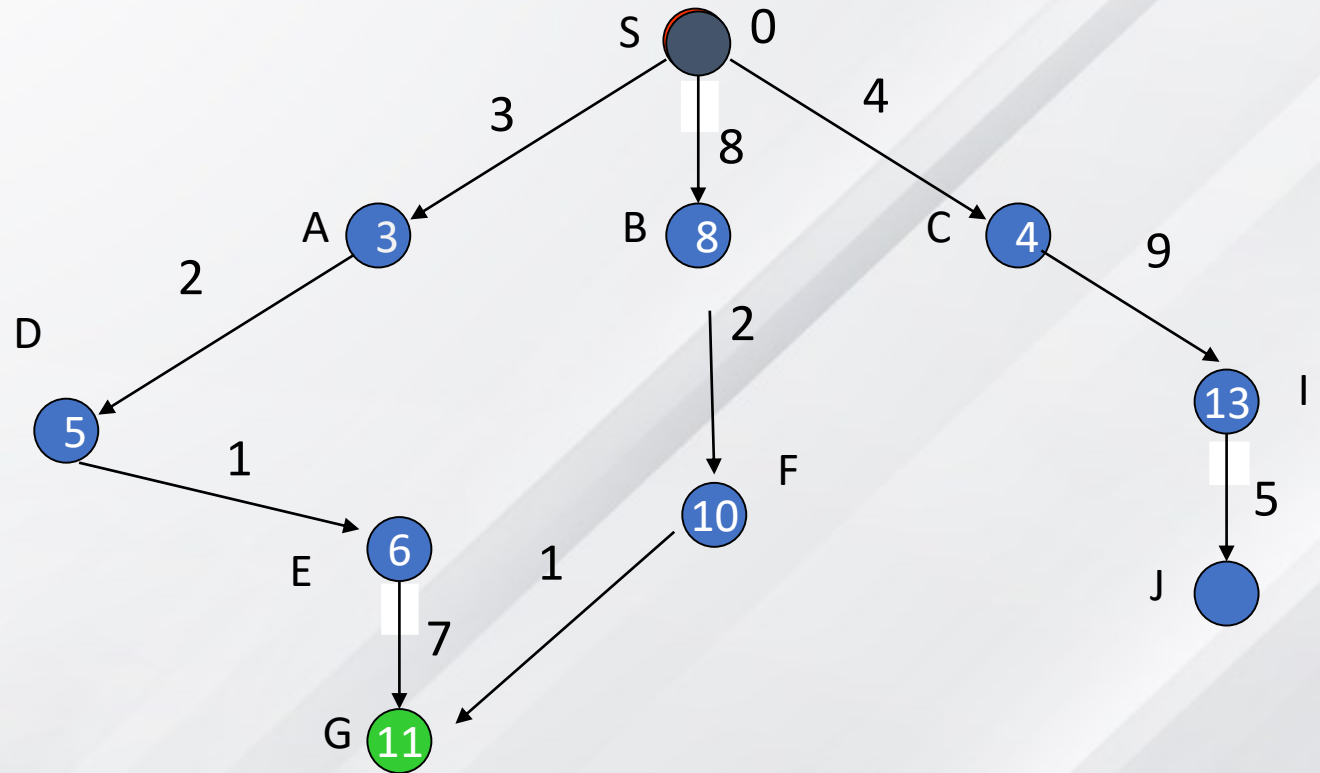
E6 B8 I13

G13 B8 I13

B8 G13 I13

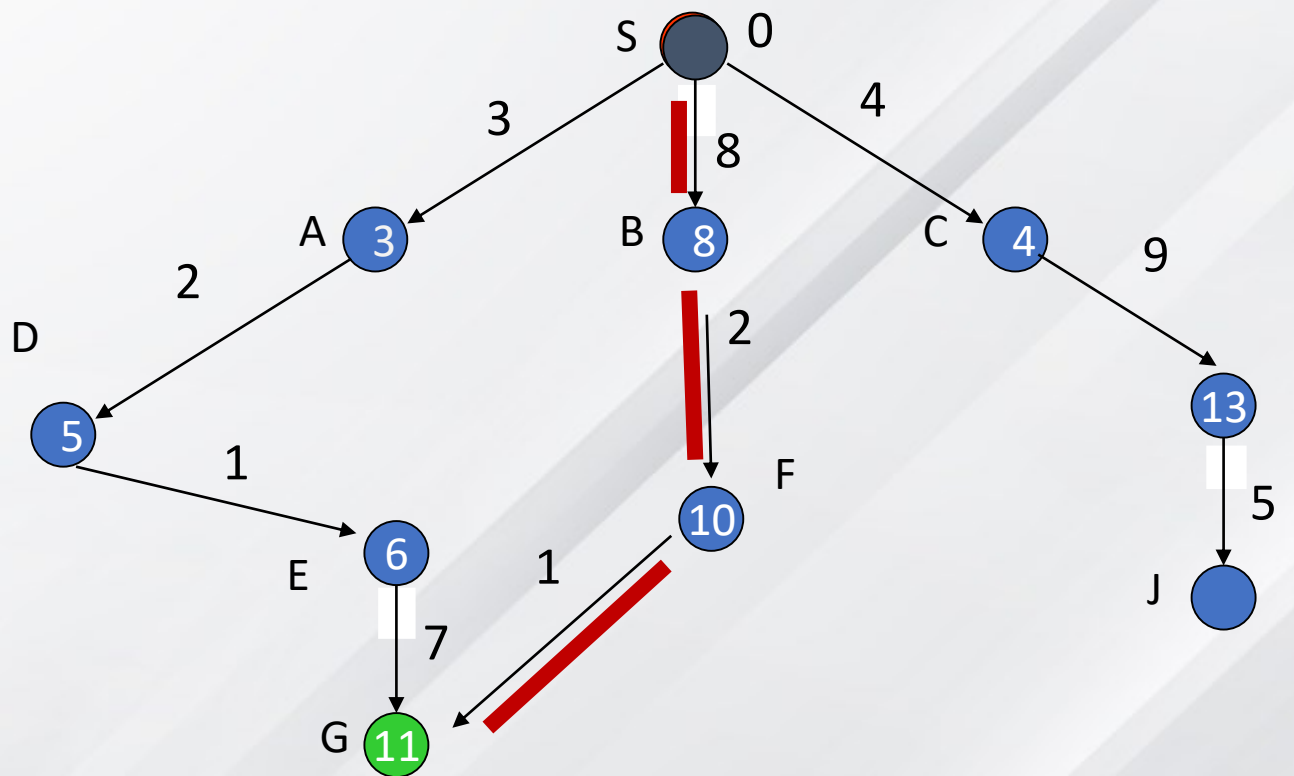
F10 G13 I13

G11 G13 I13

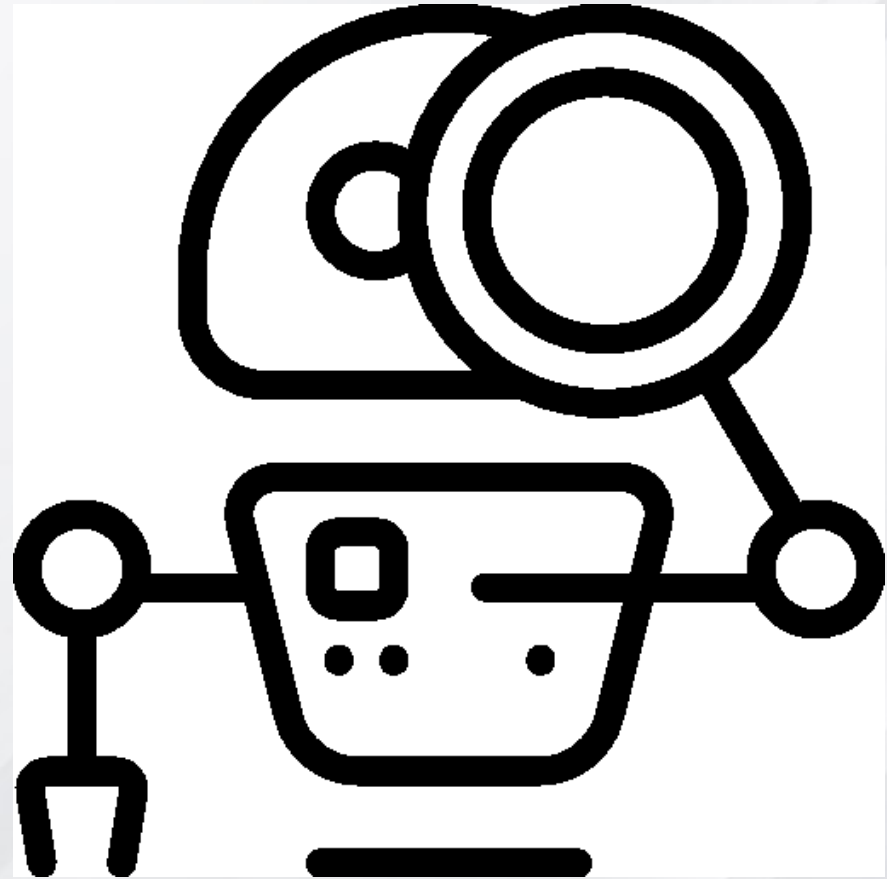


Uniform-Cost Strategy

S0
 A3 B8 C4
 D5 B8 C4
 C4 D5 B8
 I13 D5 B8
 D5 B8 I13
 E6 B8 I13
G13 B8 I13
 B8 G13 I13
 F10 G13 I13
G11 G13 I13



Informed Search [Heuristic Search]



Heuristic Function

- **Purpose:** The main purpose of a heuristic function $h(N)$ is to guide an algorithm or decision-making process by providing an estimate of the solution's quality or the cost associated with reaching it.
- $f(N)$ can be any function you want, but will it work?
- **Examples:** Heuristic functions $h(N)$ can take various forms depending on the problem domain:
 - In pathfinding problems, a common heuristic is the Euclidean distance between two points in a grid or the Manhattan distance.
 - In scheduling problems, a heuristic might prioritize tasks based on their deadlines or resource requirements.

Heuristic Function

- Function $h(N)$ that estimate the cost of the cheapest path from node N to goal node.
- Example: 8-puzzle

5		8
4	2	1
7	3	6

N

1	2	3
4	5	6
7	8	

goal

$$h(N) = \text{number of misplaced tiles} \\ = 6$$

Heuristic Function

- Function $h(N)$ that estimate the cost of the cheapest path from node N to goal node.
- Example: 8-puzzle

5		8
4	2	1
7	3	6

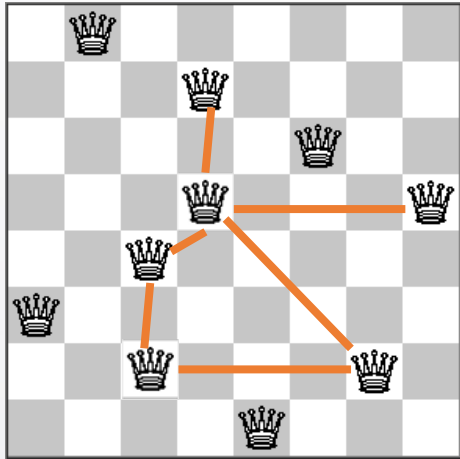
N

1	2	3
4	5	6
7	8	

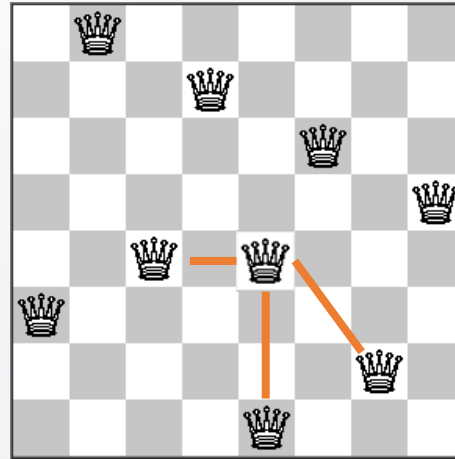
goal

$$\begin{aligned}
 h(N) &= \text{sum of the distances of} \\
 &\quad \text{every tile to its goal position} \\
 &= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 \\
 &= 13
 \end{aligned}$$

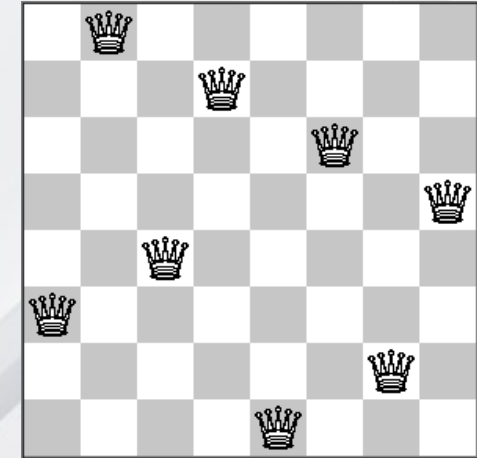
Heuristic Function



$$h(N) = 6$$



$$h(N) = 3$$



$$h(N) = 0$$

$h(N)$ = Count the number of pairs in attack

Best First Search

- The best first search uses the concept of a priority queue and heuristic search.
- The aim is to reach the goal from the initial state via the shortest path.

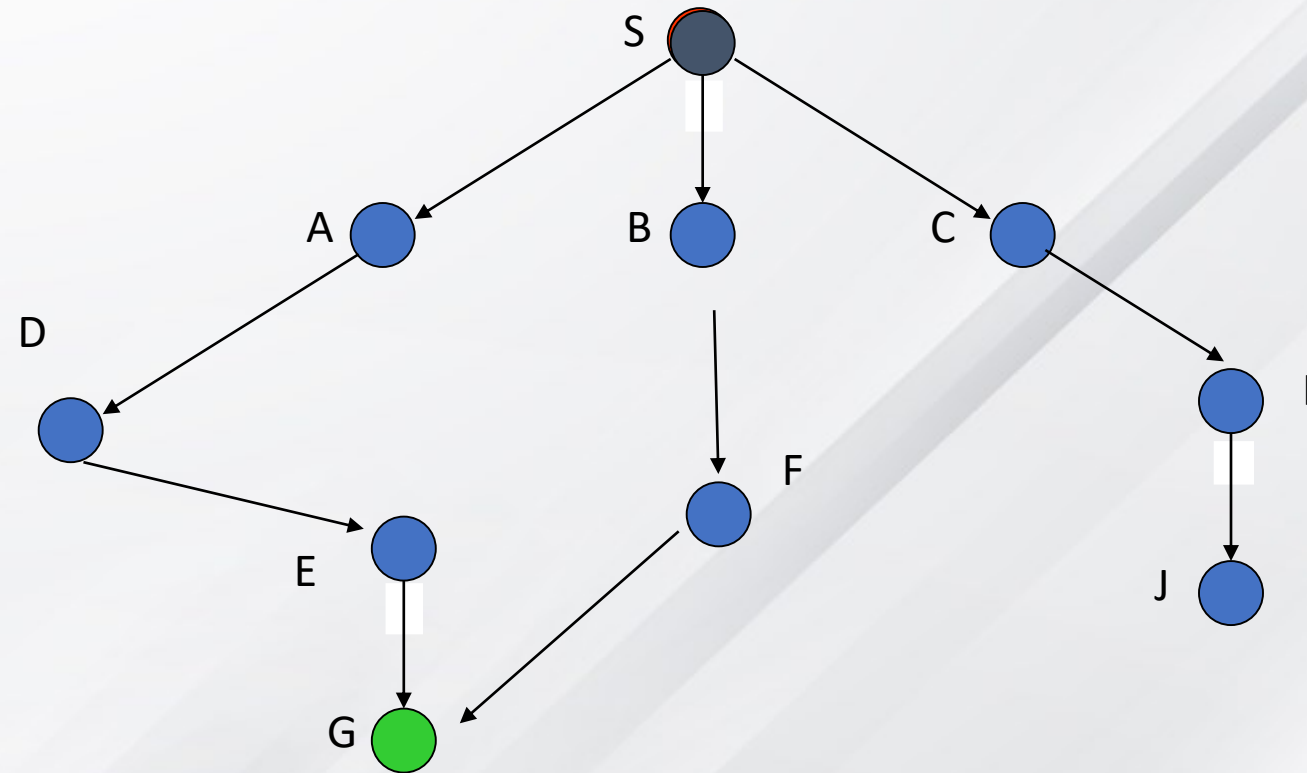
Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until the GOAL node is reached
 1. If the OPEN list is empty, then EXIT the loop returning 'False'
 2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
 3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
 4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
 5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function $f(n)$

Best First Search

Closed
List

Open List
S

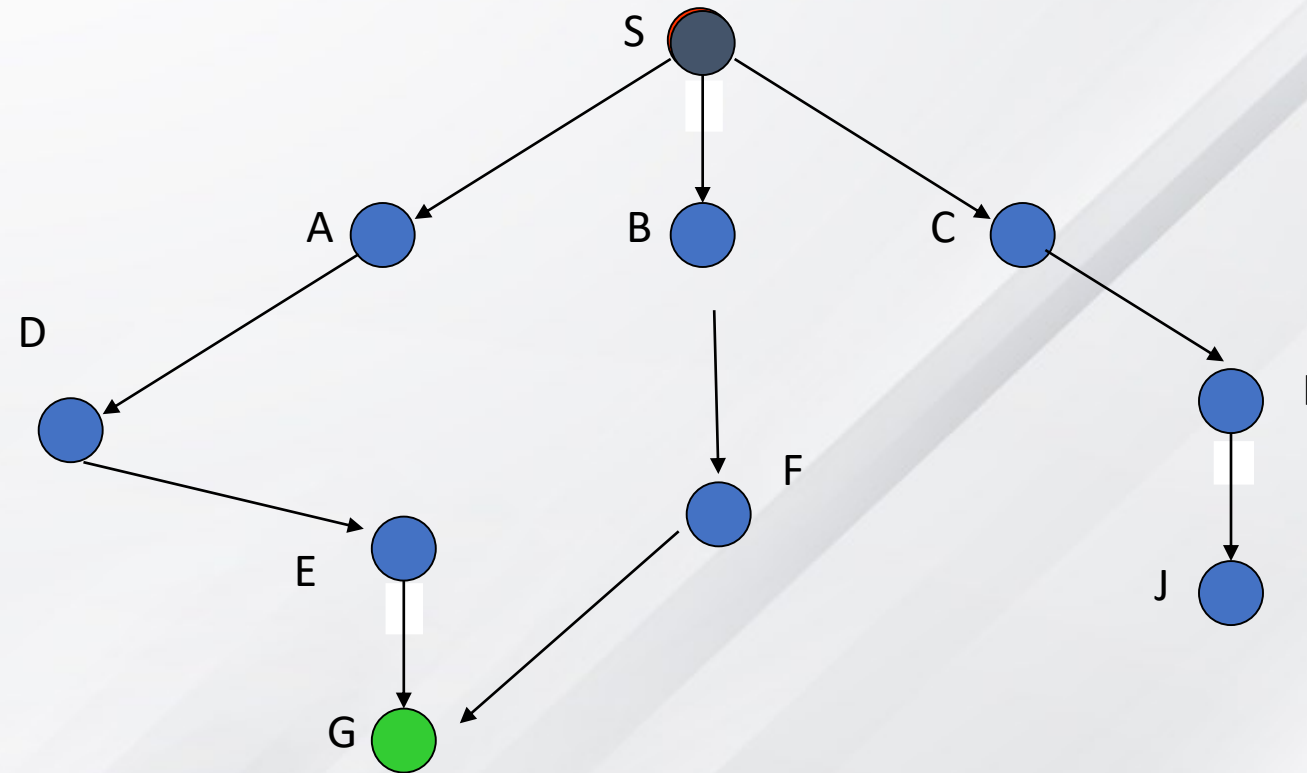


Node	$h(N)$
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

Best First Search

Closed List
S

Open List
S
A(7) B(4) C(16)

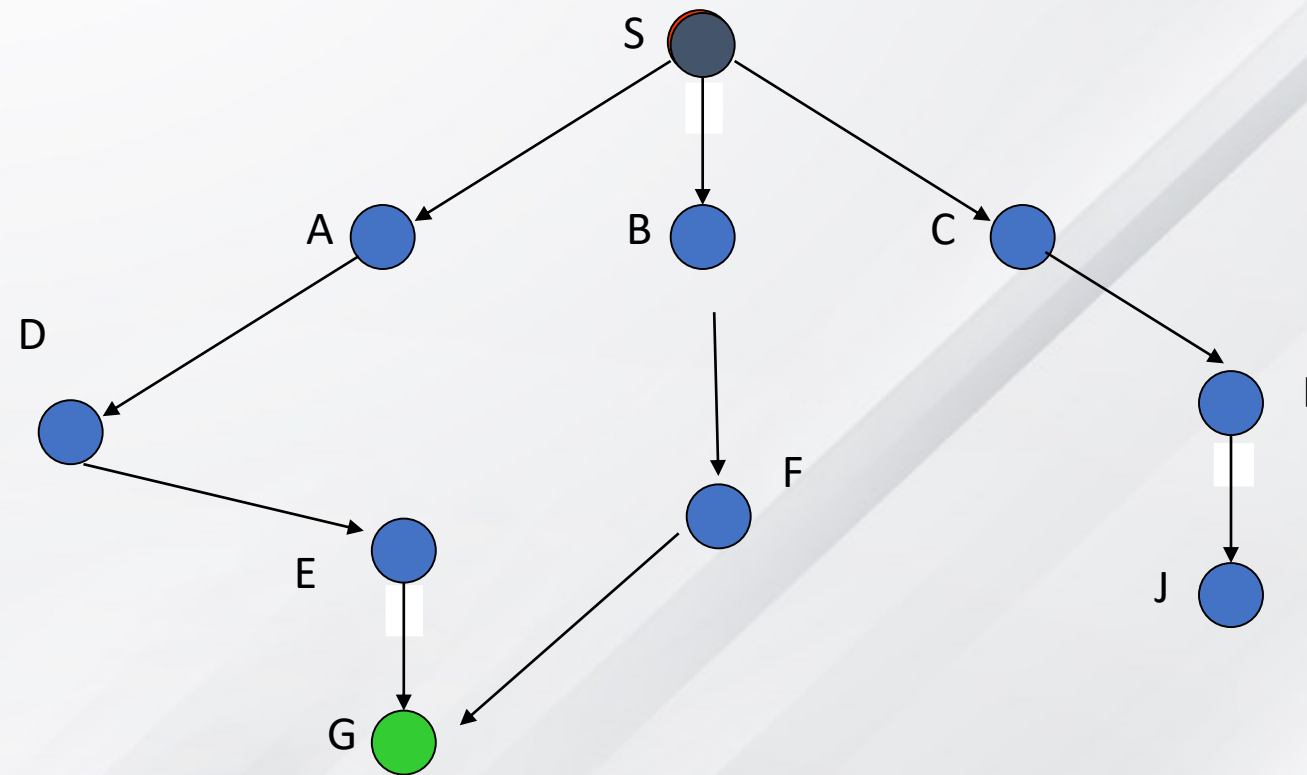


Node	$h(N)$
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

Best First Search

Closed List
S, B

Open List
S
A(7) C(16)
A(7) C(16) F(10)

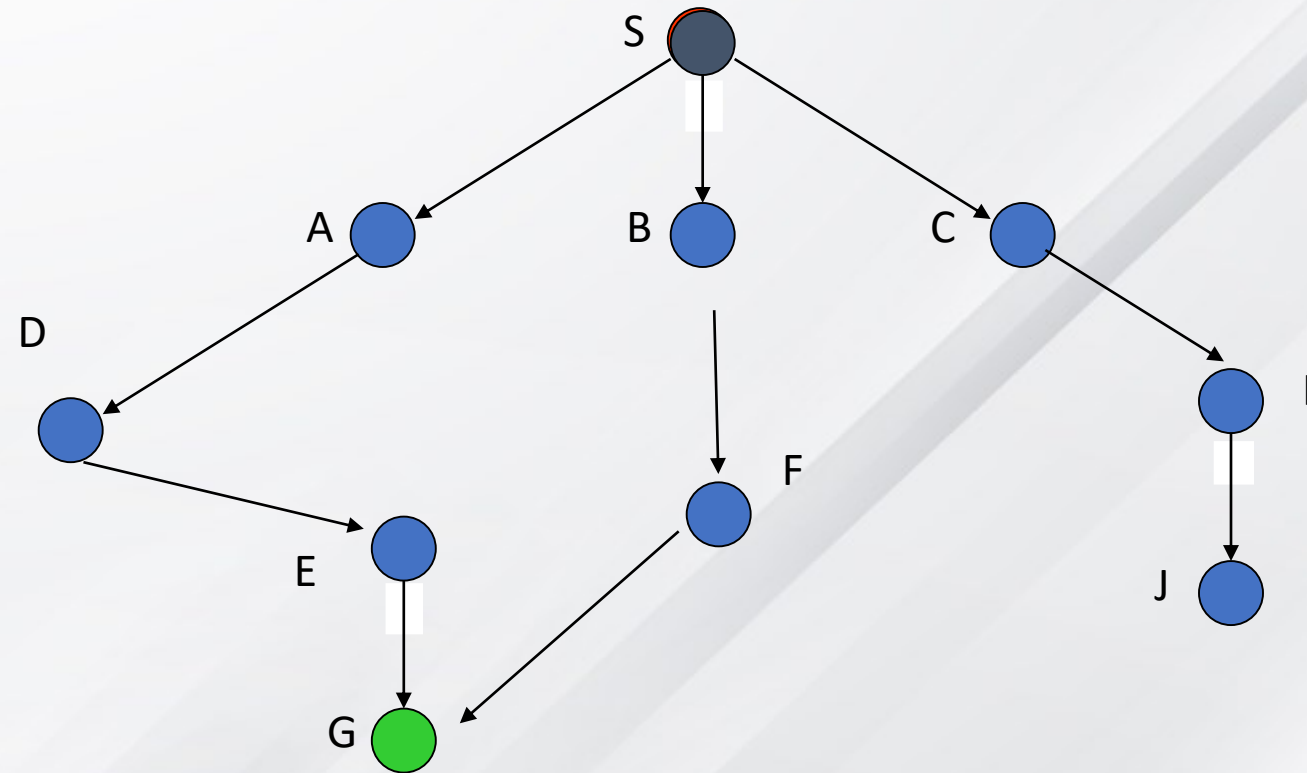


Node	$h(N)$
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

Best First Search

Closed List
S
S, B
S, B, A

Open List
S
A(7) C(16) F(10)

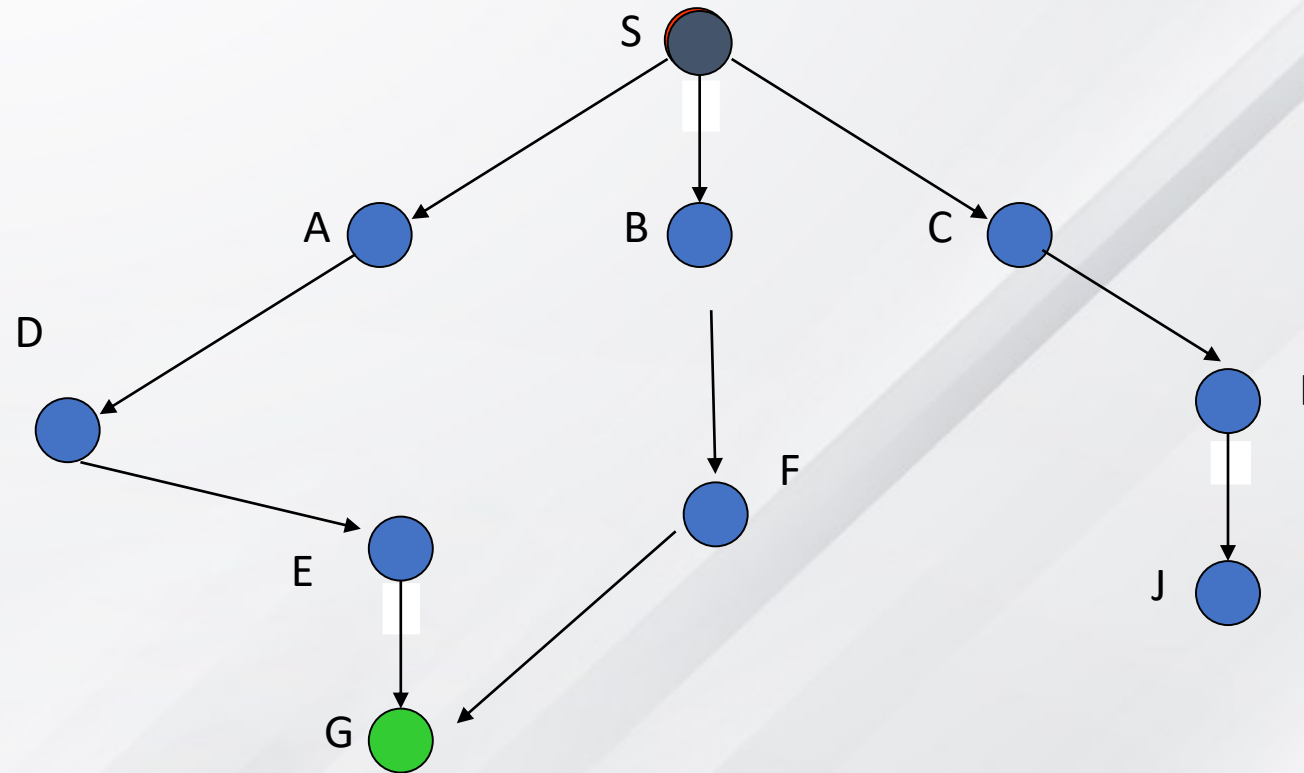


Node	$h(N)$
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

Best First Search

Closed List
S
S, B
S, B, A

Open List
S
A(7) C(16) F(10)
C(16) F(10) D(5)



Node	$h(N)$
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

Best First Search

Closed List

S

S, B

S, B, A

S, B, A, D

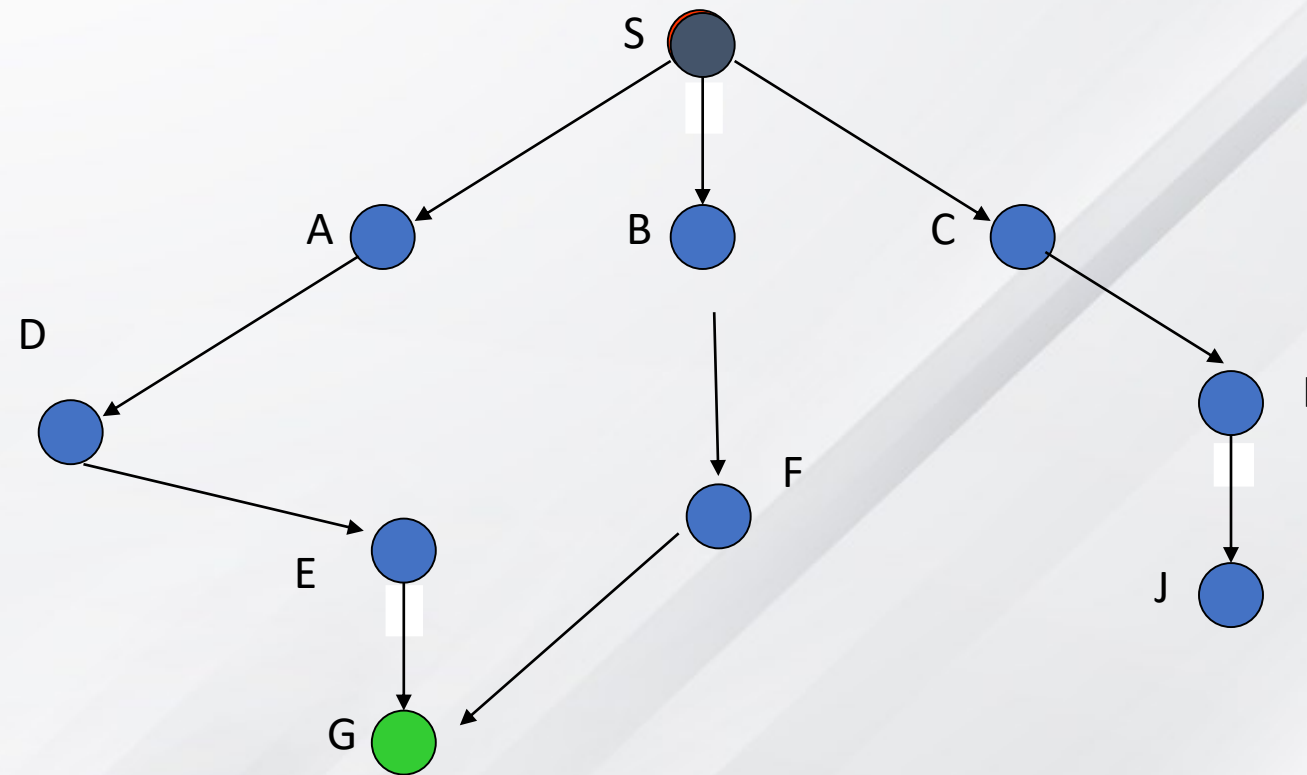
Open List

S

A(7) C(16) F(10)

C(16) F(10)

C(16) F(10) E(3)



Node	$h(N)$
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

Best First Search

Closed List

S
S, B
S, B, A

S, B, A, D

S, B, A, D, E

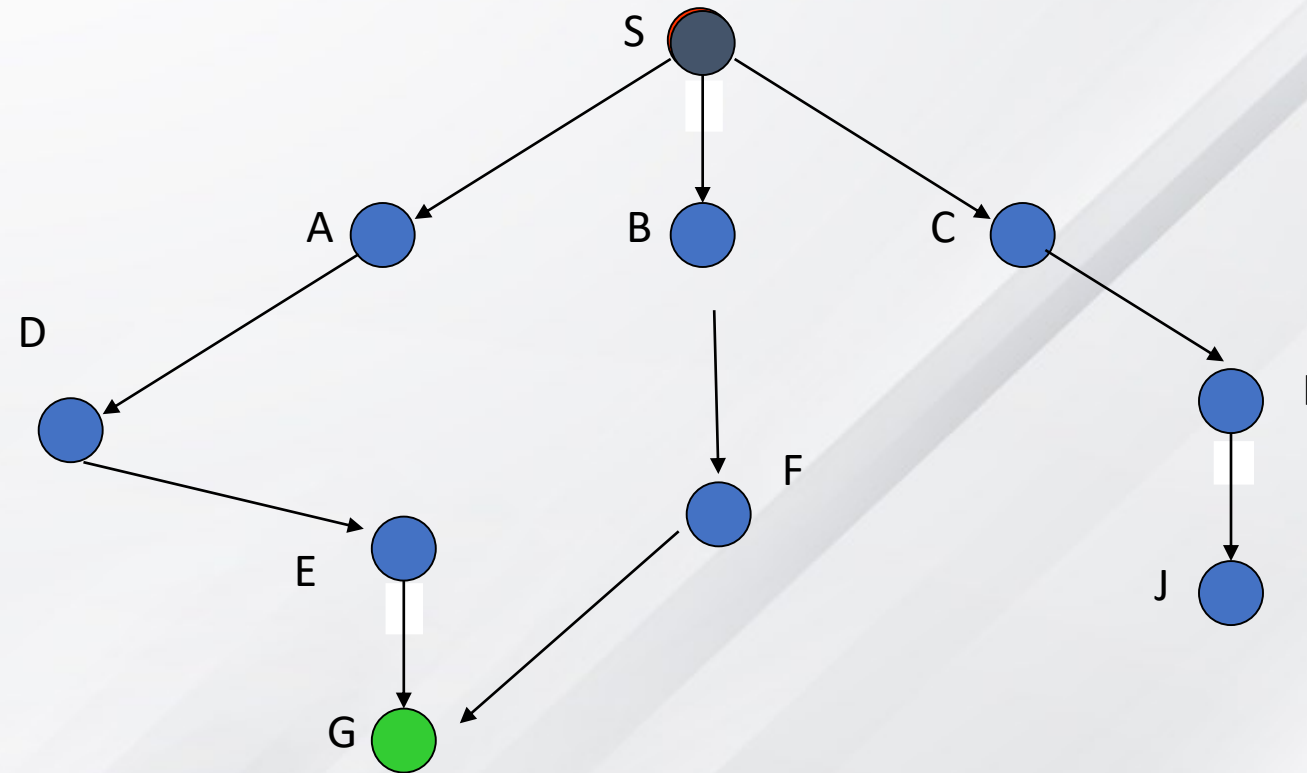
Open List

S
A(7) C(16) F(10)

C(16) F(10)

C(16) F(10)

C(16) F(10) G(0)



Node	h(N)
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

Best First Search

Closed
List

S
S, B
S, B, A

S, B, A, D

S, B, A, D,
E

S, B, A, D,
E, G

Open List

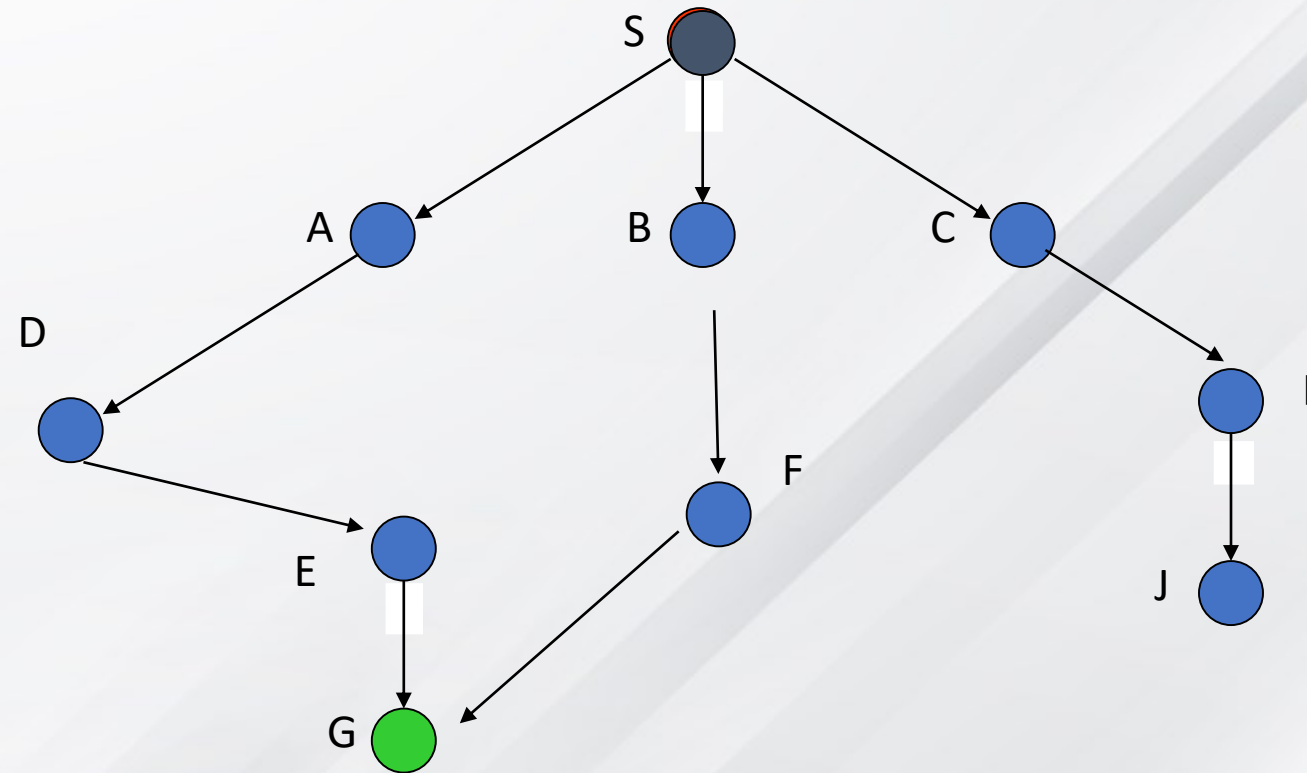
S
A(7) C(16) F(10)

C(16) F(10)

C(16) F(10)

C(16) F(10)

GOAL



$S \rightarrow A \rightarrow D \rightarrow E \rightarrow G$

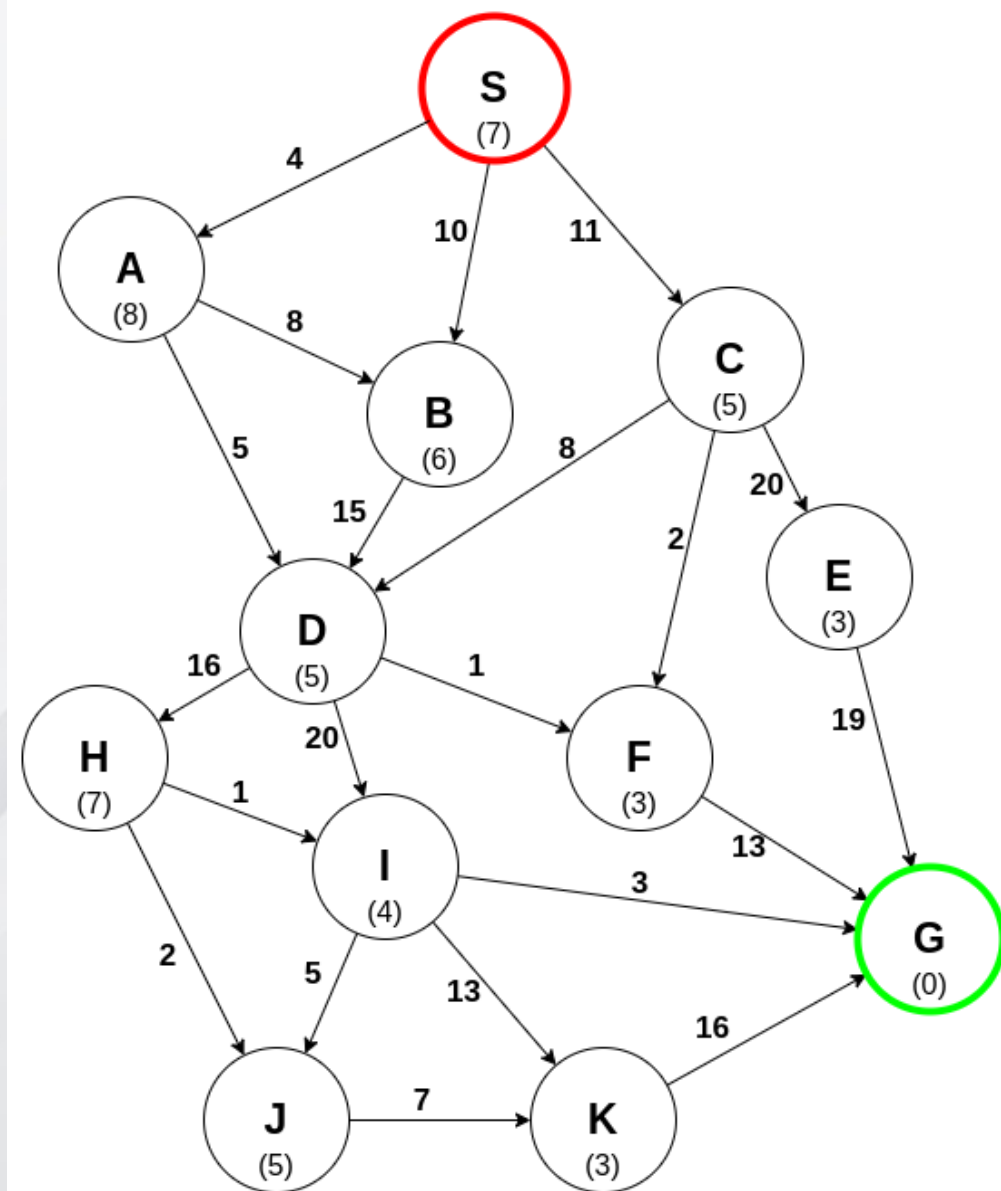
Node	h(N)
S	14
A	7
B	4
C	16
D	5
E	3
F	10
G	0
I	9
j	8

A* Algorithm

- It is an informed search algorithm that selects the most promising node based on a heuristic evaluation function.
- It is a searching algorithm that is used to find the shortest path between an initial and a final point.

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

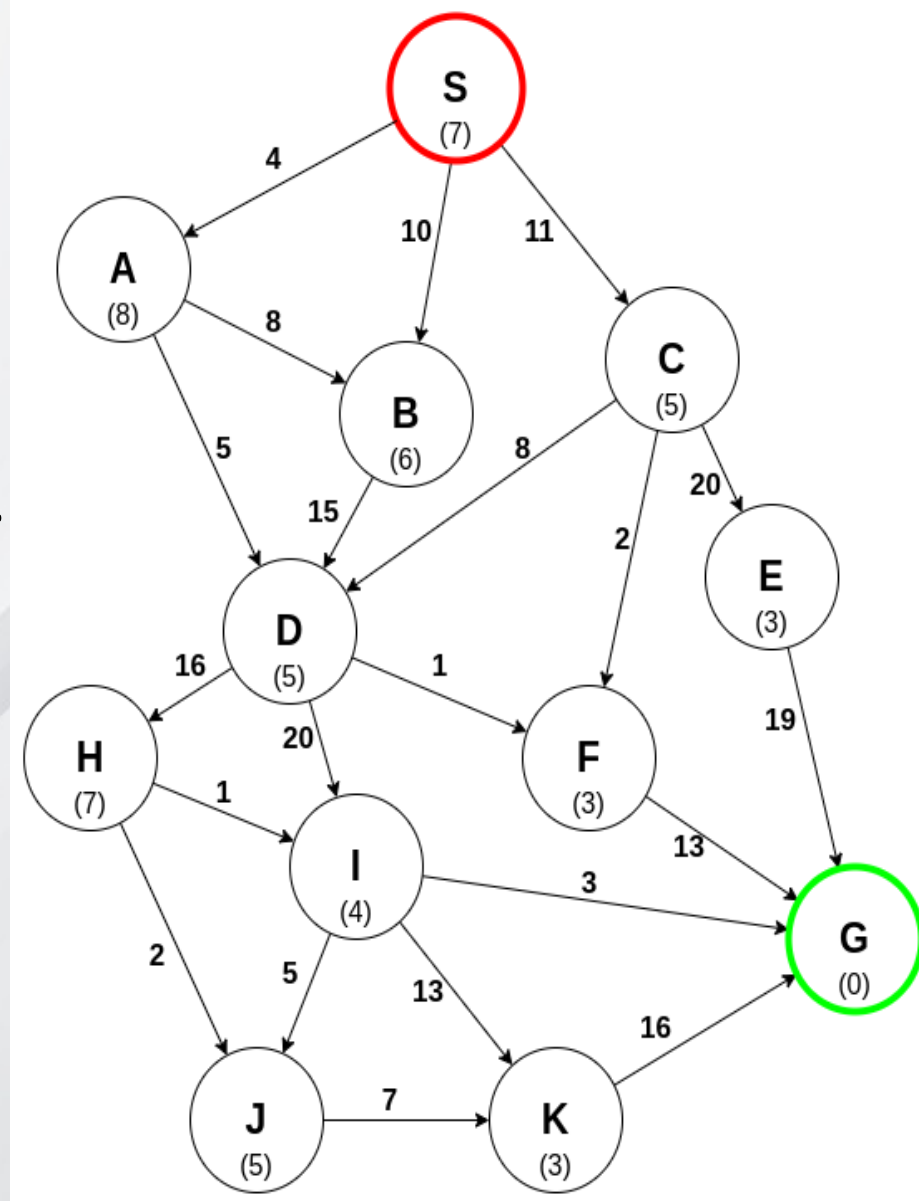
- $g(x)$ represents the cost to get to node x
- $h(x)$ represents the estimated cost to arrive at the goal node from node x .



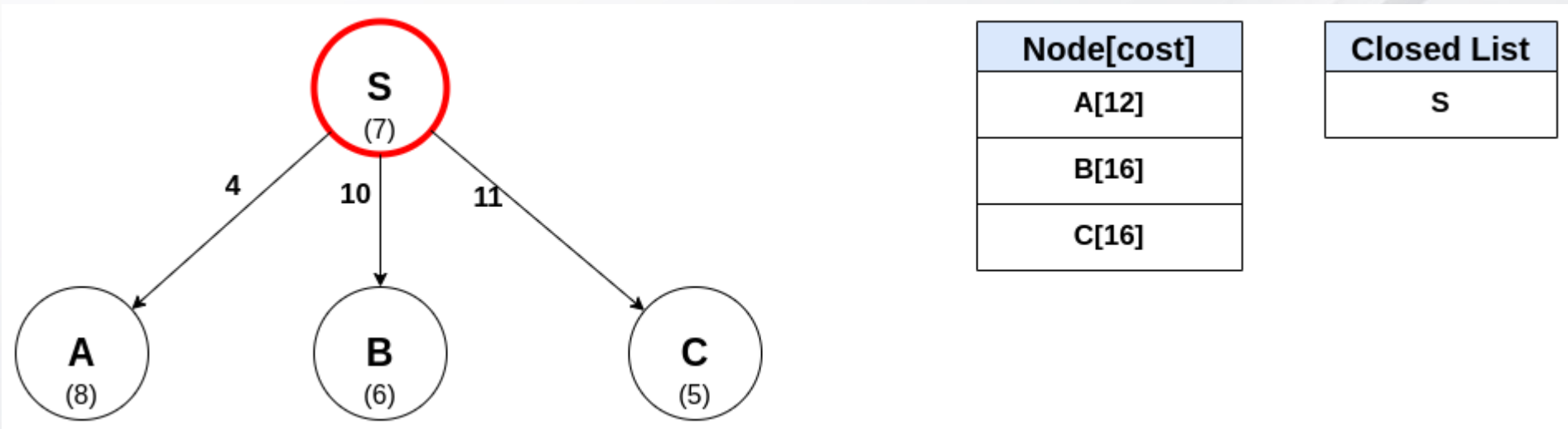
A* Algorithm

Algorithm Steps

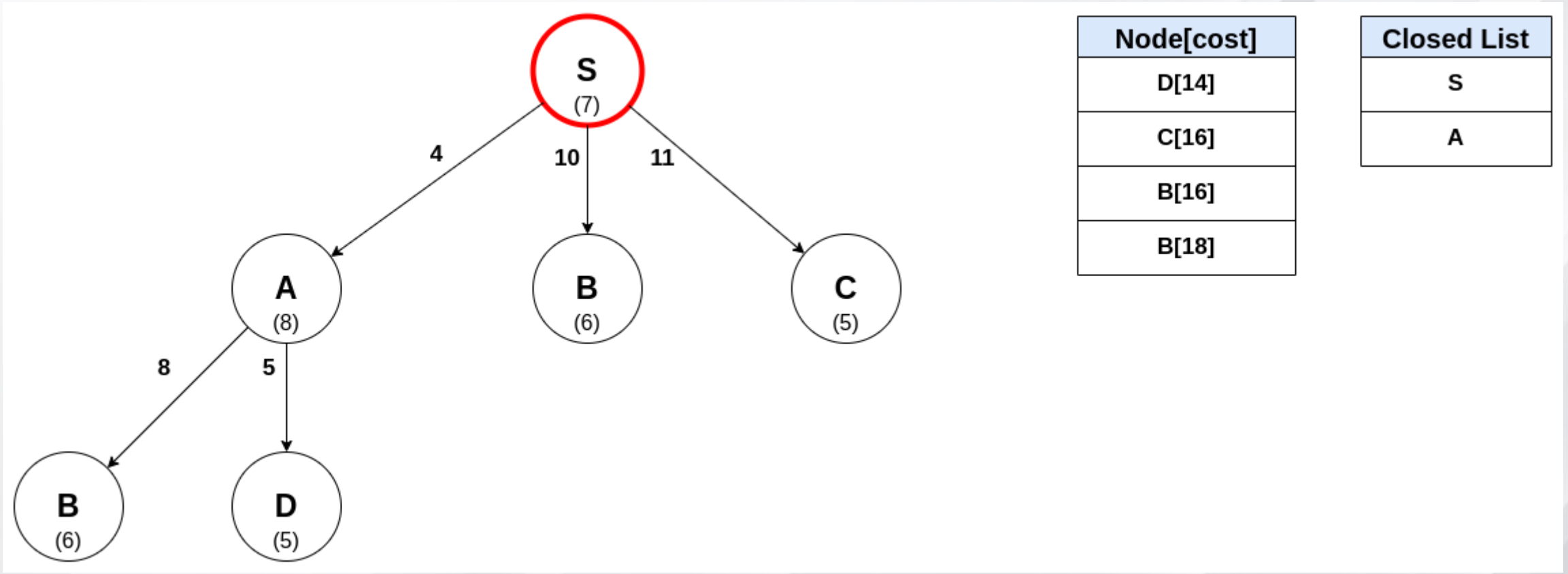
- Initialize a tree with the root node being the start node **S**.
- Remove the top node from the open list for exploration.
- Add the current node to the closed list.
- Add all nodes that have an incoming edge from the current node as child nodes in the tree.
- Update the lowest cost to reach the child node.
- Compute the evaluation function for every child node and add them to the open list.



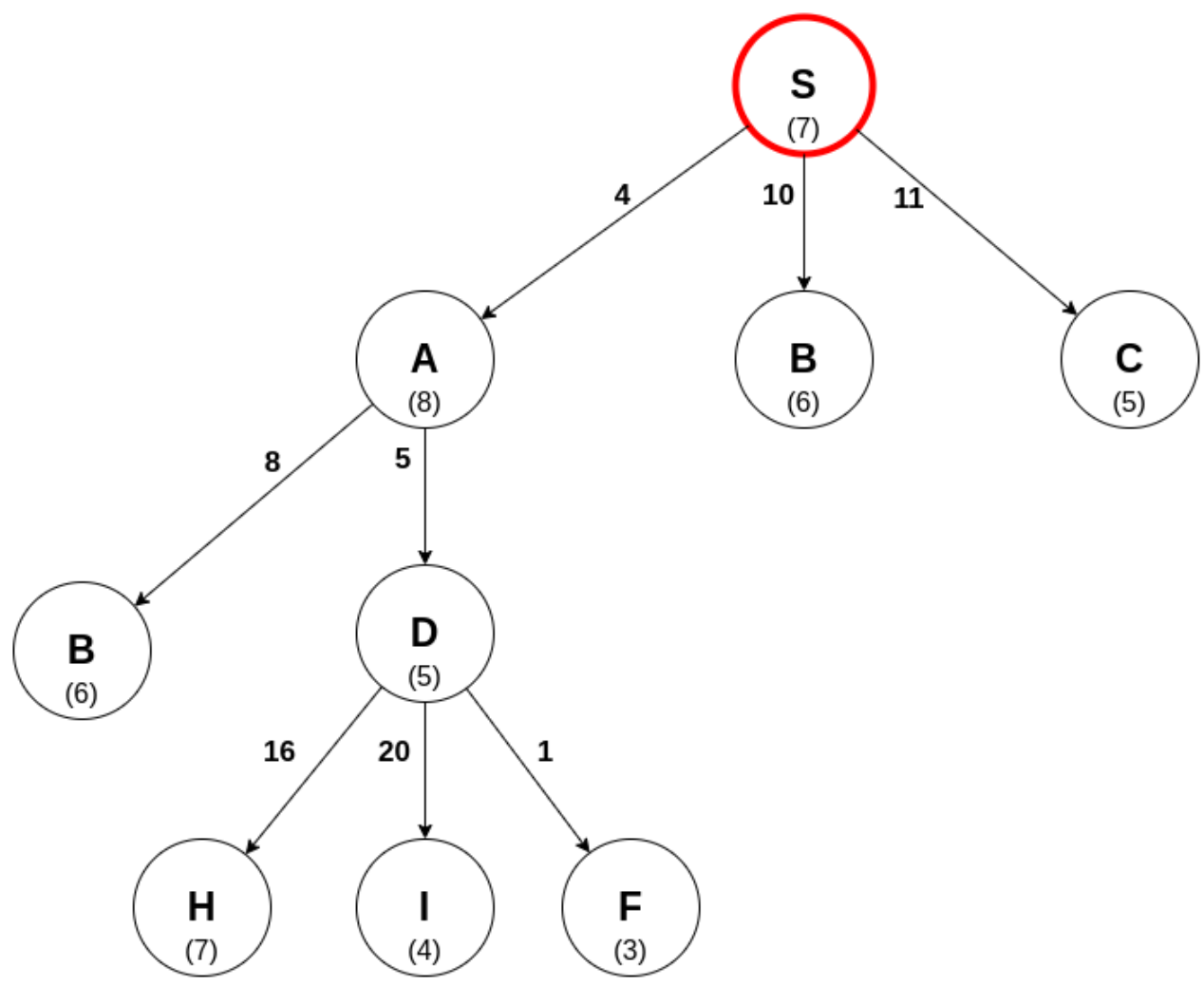
A* Algorithm



A* Algorithm



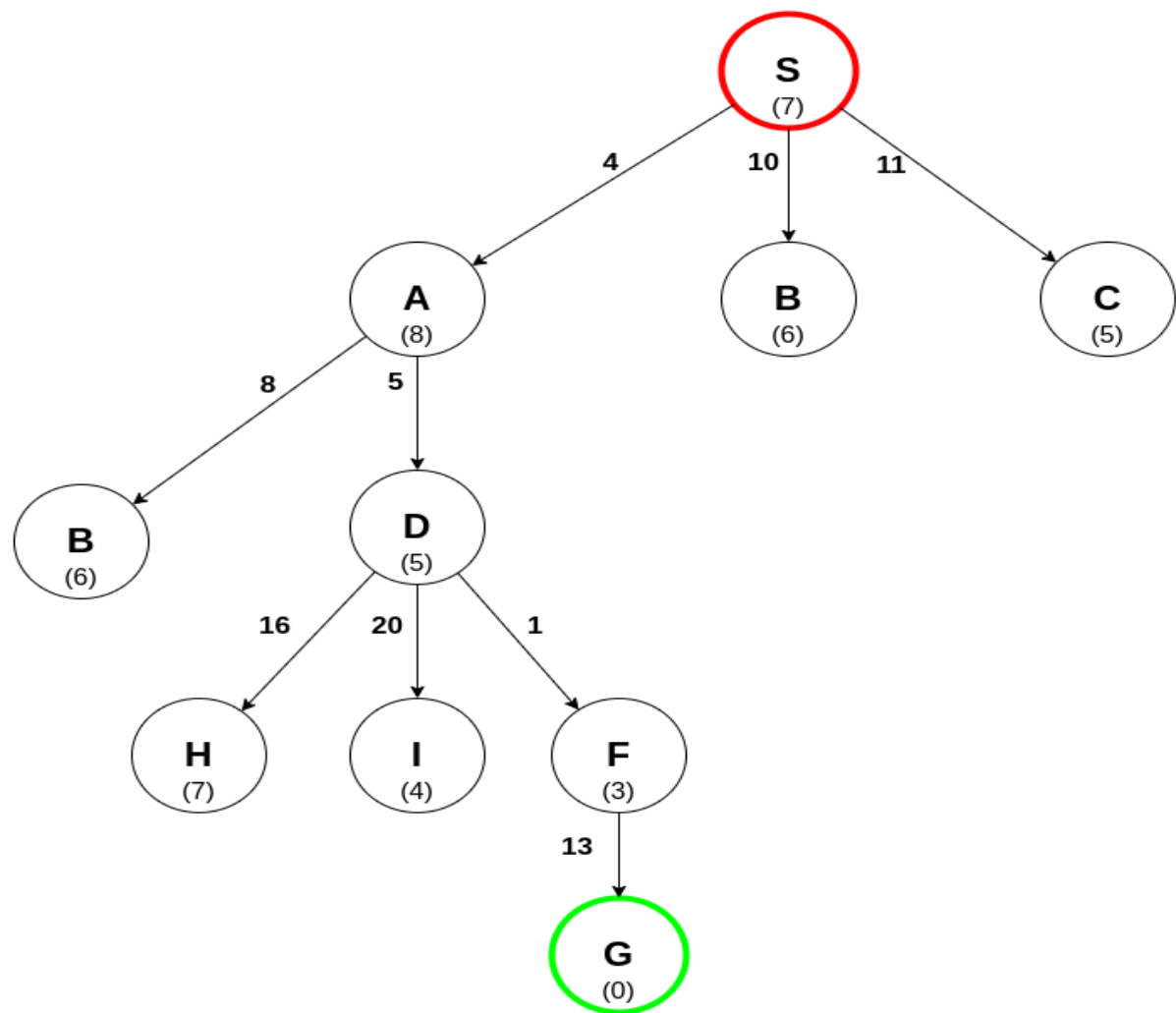
A* Algorithm



Node[cost]
F[13]
C[16]
B[16]
H[32]
I[33]
B[18]

Closed List
S
A
D

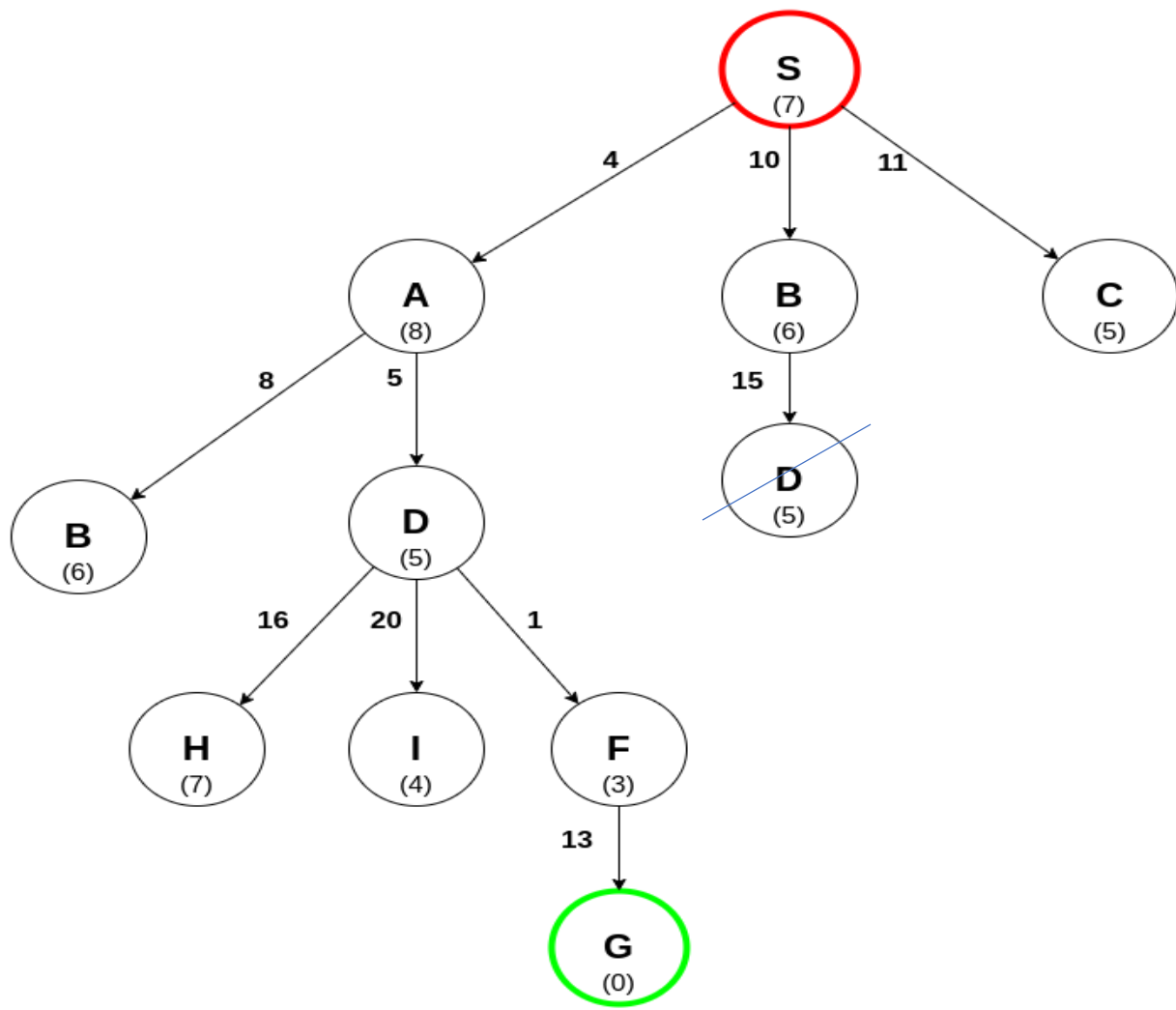
A* Algorithm



Node[cost]
B[16]
C[16]
B[18]
H[32]
I[33]
G[23]

Closed List
S
A
D
F

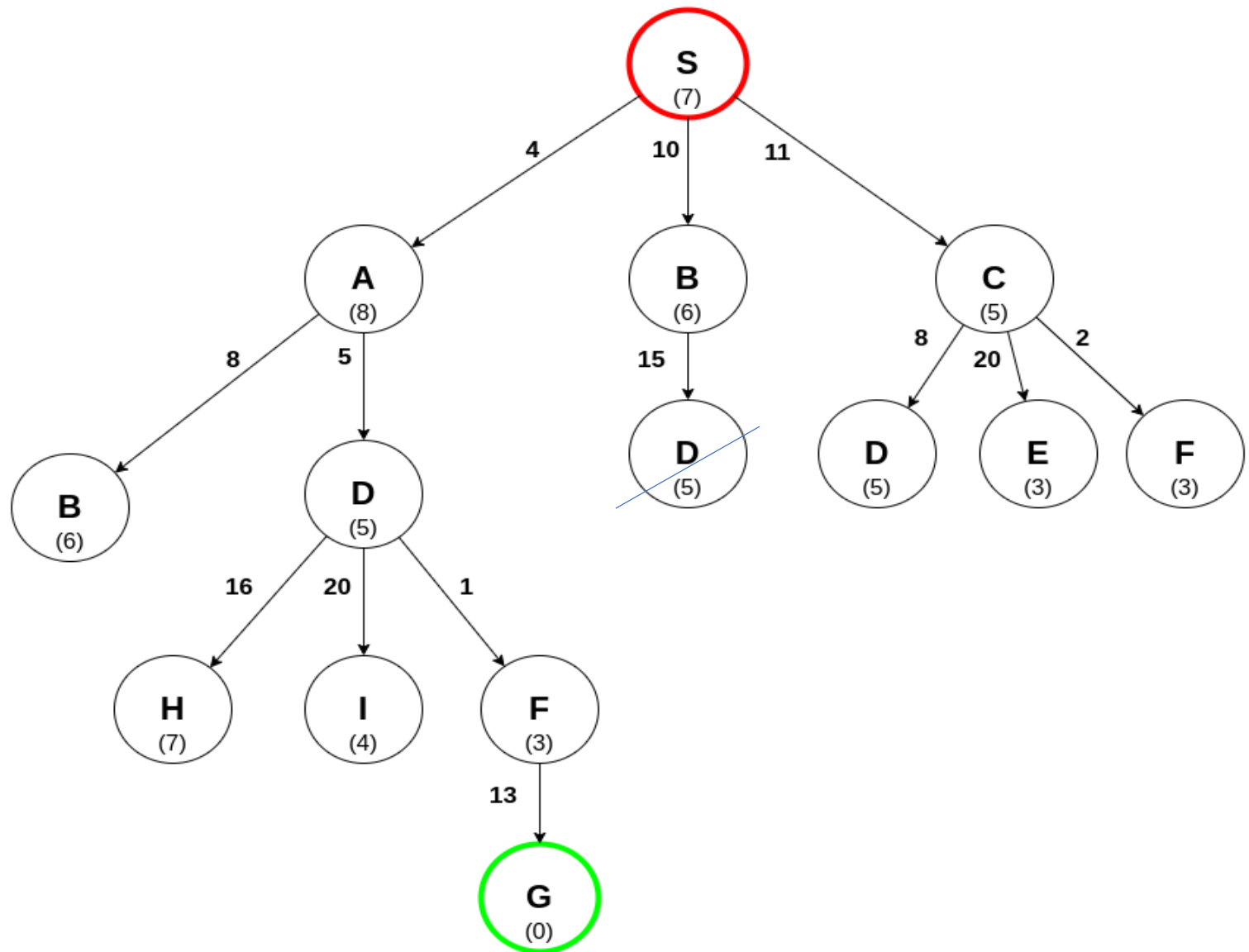
A* Algorithm



Node[cost]
C[16]
G[23]
B[18]
H[32]
I[33]

Closed List
S
A
D
F
B

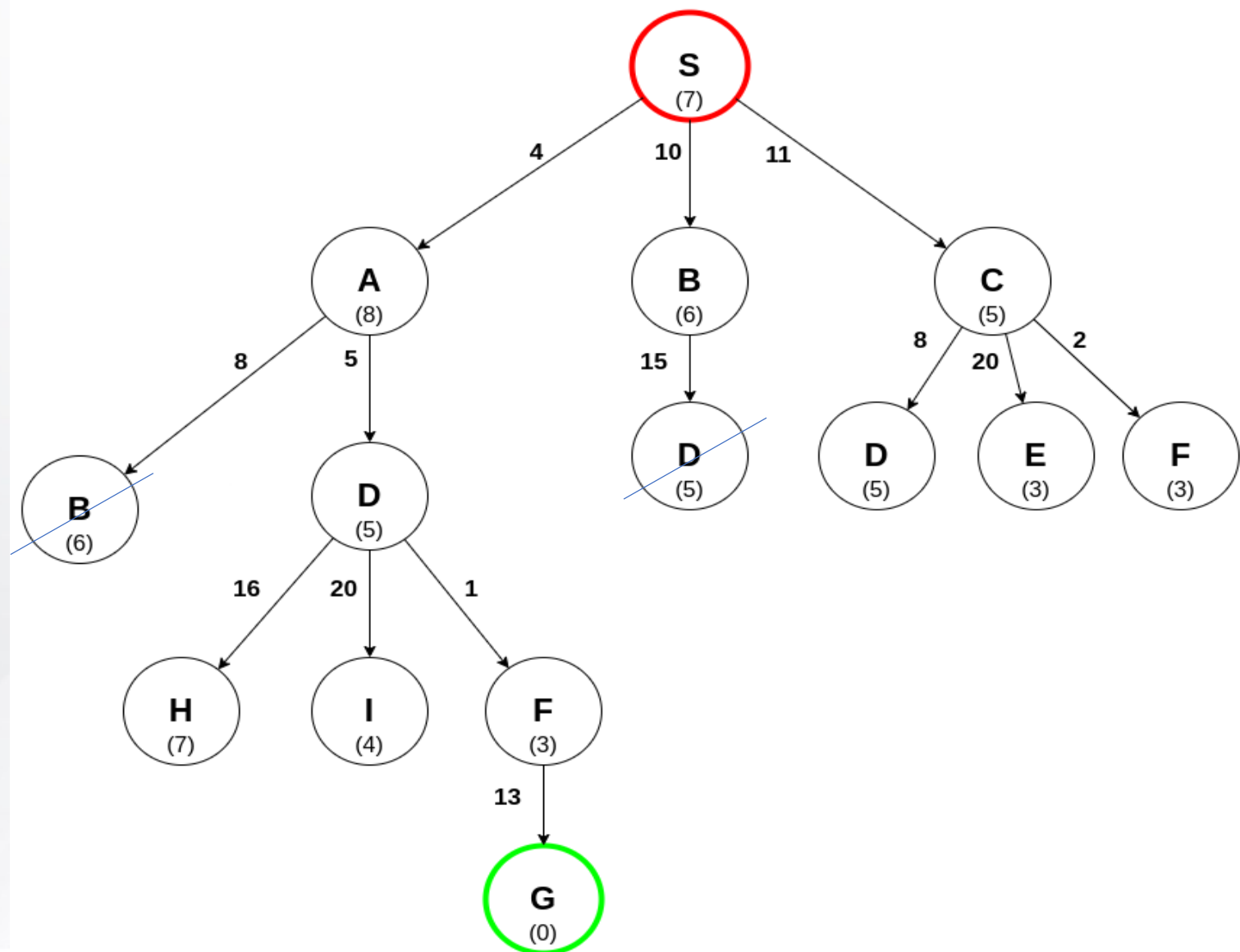
A* Algorithm



Node[cost]
B[18]
G[23]
I[33]
H[32]
E[36]

Closed List
S
A
D
F
B
C

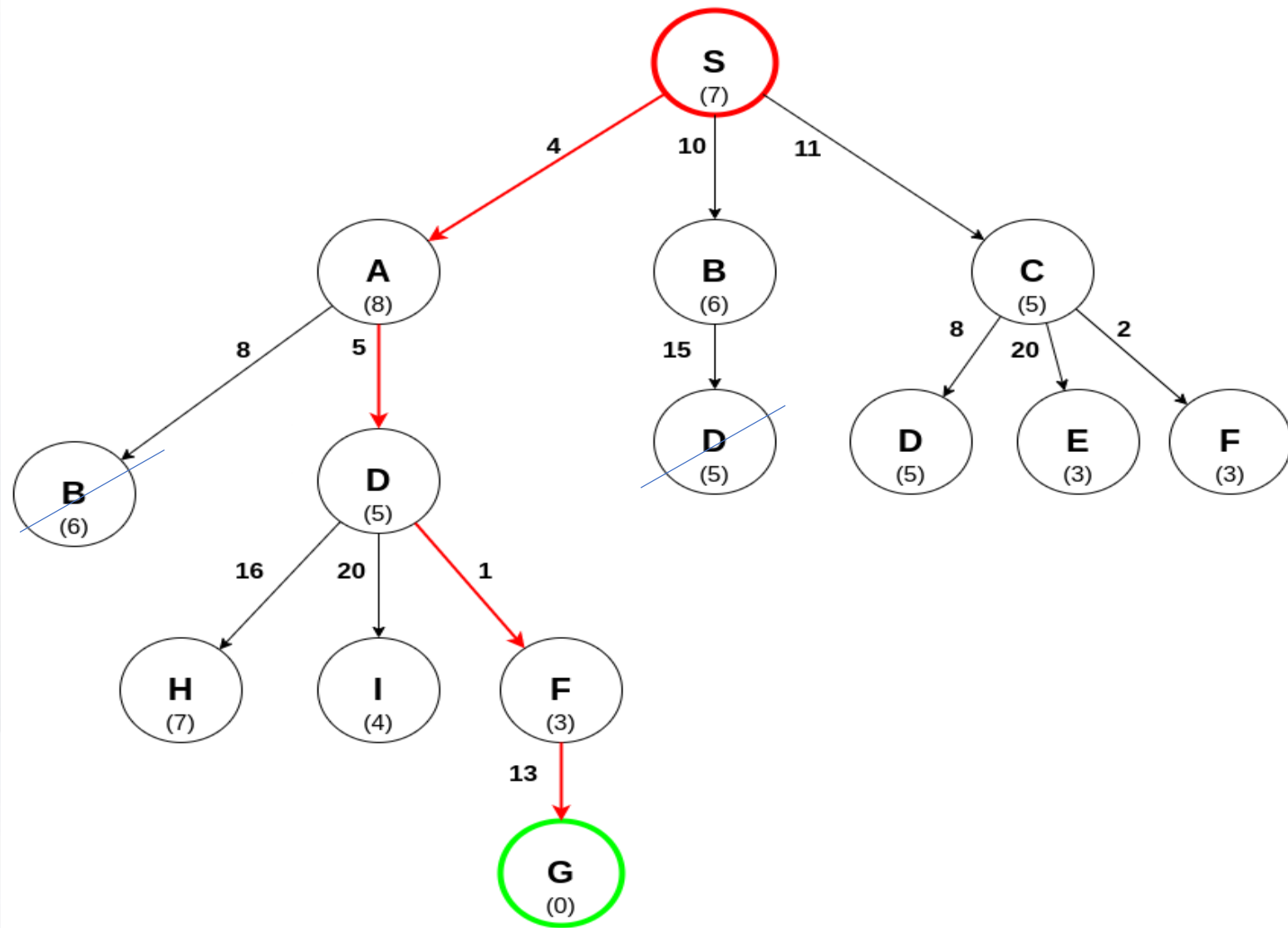
A* Algorithm



Node[cost]
G[23]
I[33]
H[32]
E[36]

Closed List
S
A
D
F
B
C

A* Algorithm



Node[cost]
I[33]
H[32]
E[36]

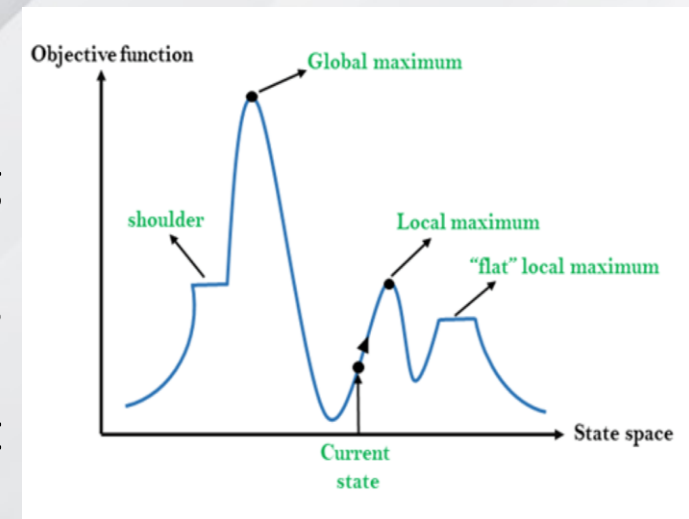
Closed List
S
A
D
F
B
C
G

Hill Climbing

The **hill climbing** search algorithm is a local search algorithm used for optimization problems. It is designed to find the highest point or the best solution within a given search space by iteratively exploring neighboring solutions.

Algorithm Steps:

- Initialization: Start with an initial solution within the search space.
- Evaluation: Evaluate the quality of the current solution using an objective function or fitness measure.
- Neighbor Generation: Generate neighboring solutions by making small modifications to the current solution.
- Selection: Choose the best neighboring solution based on its objective function value.
- Comparison: Compare the objective function value of the best neighboring solution with the current solution.
- Iteration: Repeat the process until a termination condition is met.



Hill Climbing

- Types of Hill Climbing Algorithm:

- Simple hill Climbing:

- Simple Hill Climbing is a basic variant of the hill climbing algorithm.

- Steepest-Ascent hill-climbing:

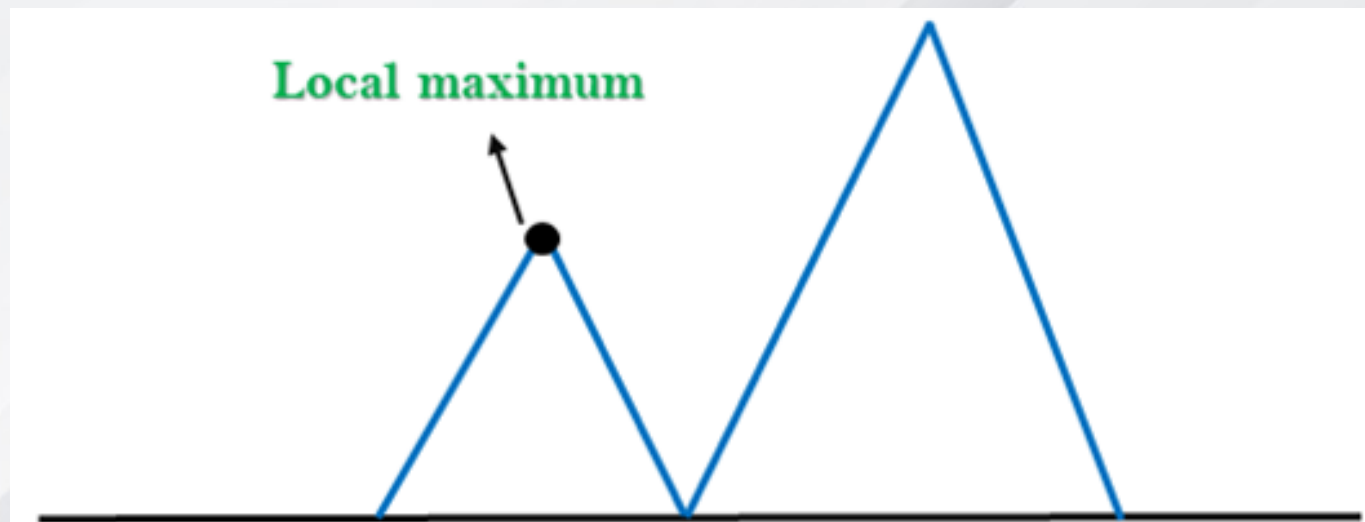
- represents an advancement over the basic hill climbing algorithm. In this approach, during each iteration, it thoroughly assesses all nearby solutions. Among these neighboring solutions, it picks the one that provides the most substantial enhancement in the objective function.

- Stochastic hill Climbing:

- Stochastic Hill Climbing is a probabilistic variant of the hill climbing algorithm.
 - Instead of always selecting the best neighboring solution, it randomly selects a neighboring solution with a probability based on its quality.
 - This introduces an element of randomness into the search process, allowing the algorithm to escape local optima and explore a wider range of solutions.
 - Stochastic Hill Climbing is particularly useful for optimization problems with a large search space or multiple local optima.

Hill Climbing

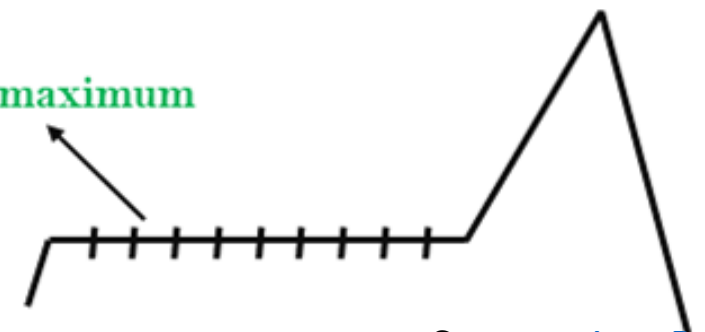
- Problems in Hill Climbing Algorithm:
- **1. Local Maximum:** A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.
- **Solution:** Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



Hill Climbing

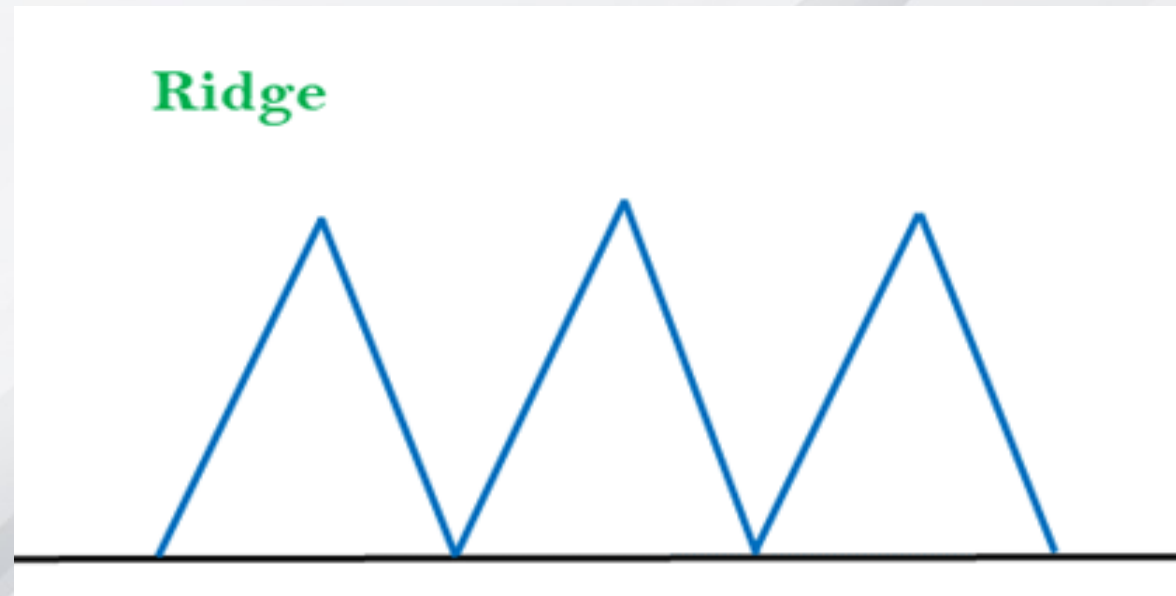
- Problems in Hill Climbing Algorithm:
- **2. Plateau:** A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.
- **Solution:** The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region

Plateau/Flat maximum

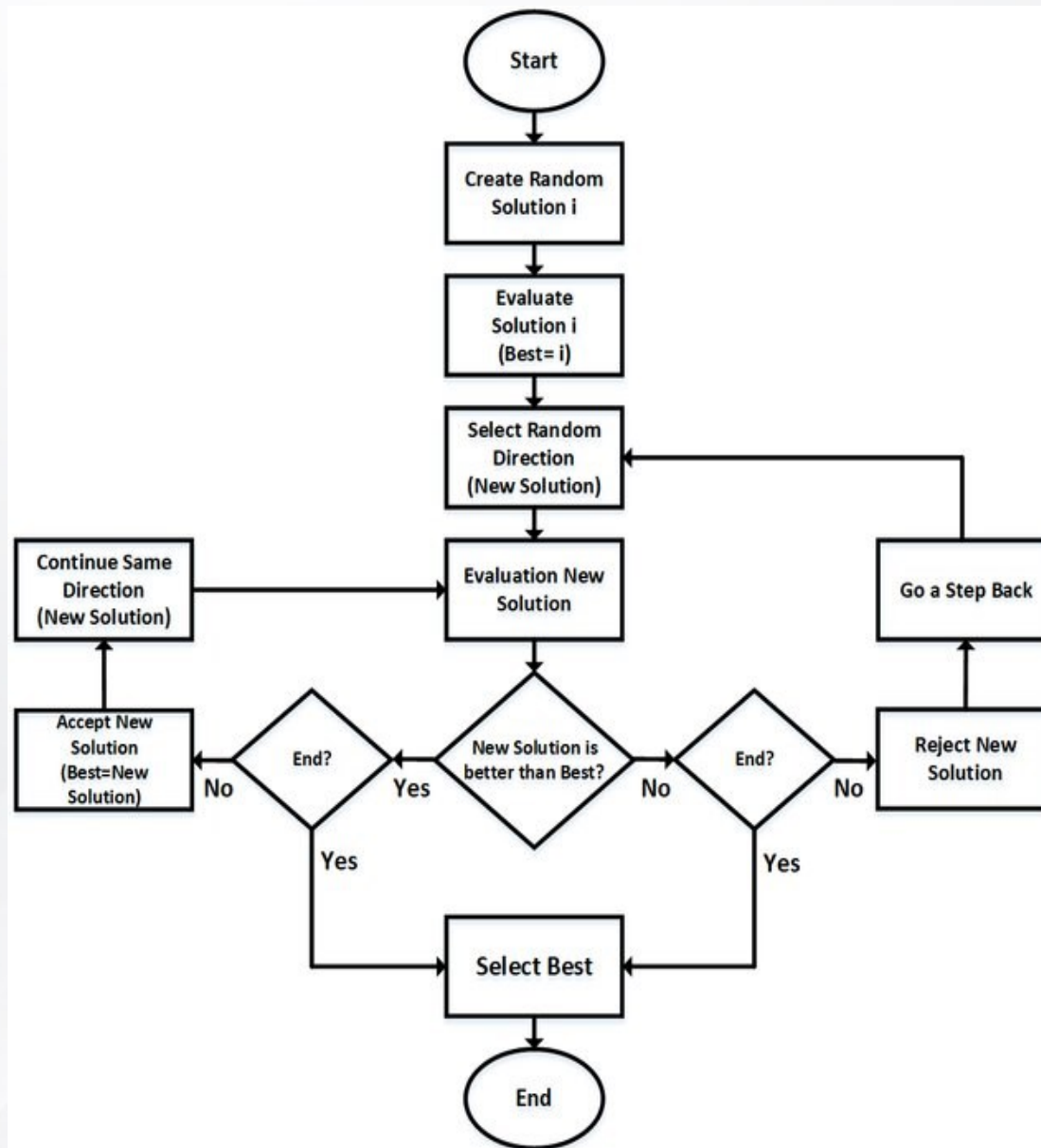


Hill Climbing

- Problems in Hill Climbing Algorithm:
- **3. Ridges:** A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.
- **Solution:** With the use of bidirectional search, or by moving in different directions, we can improve this problem.



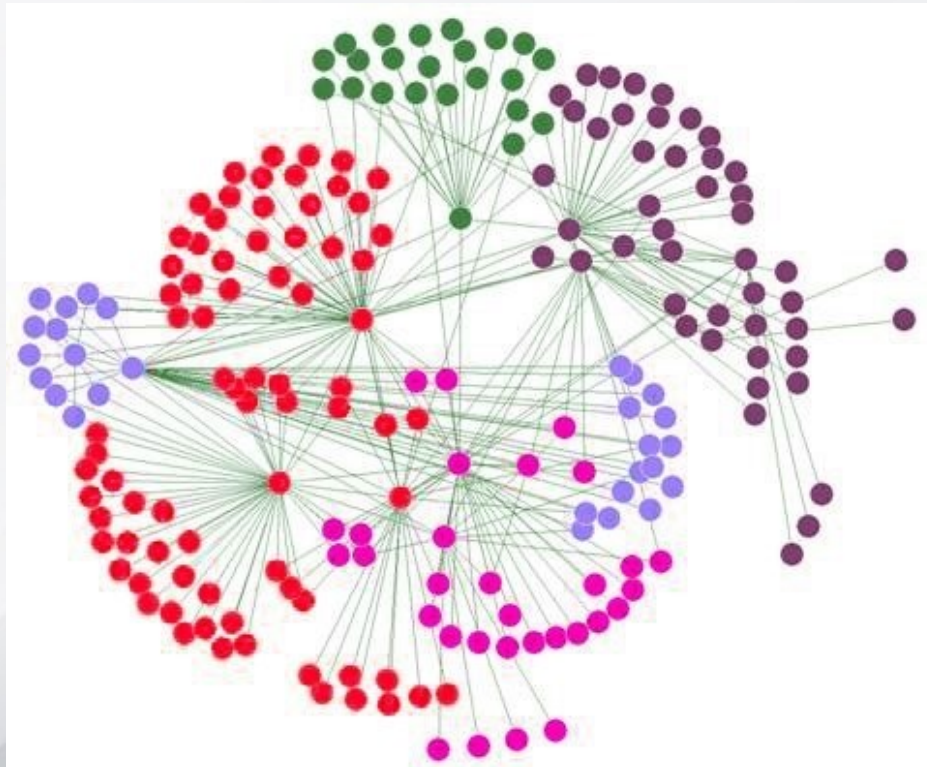
Hill Climbing Real-World Applications



Hill Climbing Real-World Applications

- **Network Routing**

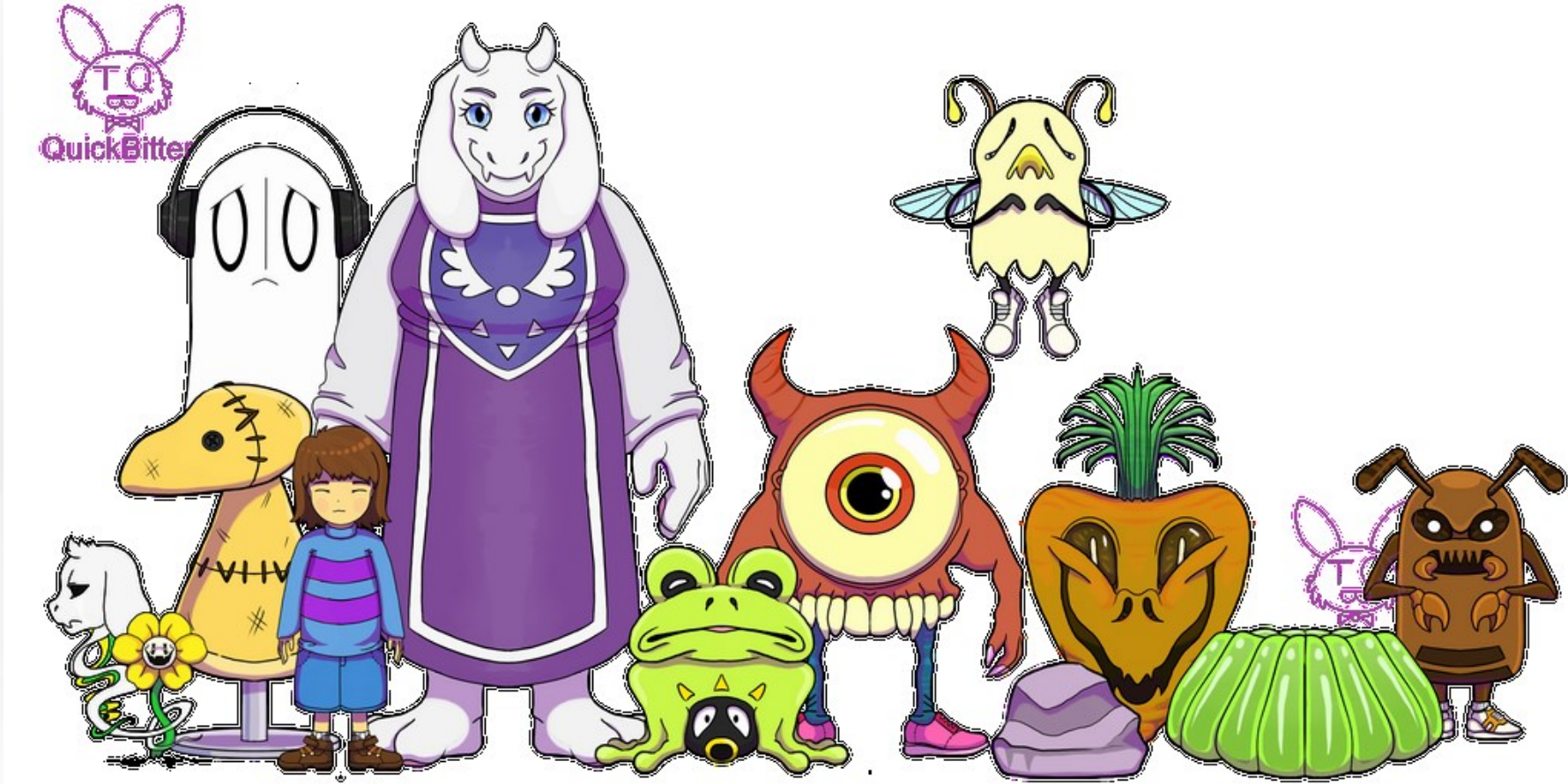
- Hill Climbing can be employed to enhance the efficiency of routing decisions, leading to optimized data packet traversal across the network. By continuously refining the selection of routes, Hill Climbing aids in minimizing congestion and reducing latency, ultimately improving the overall performance of the network.



Hill Climbing Real-World Applications

- **Game Strategy Optimization**

- Hill Climbing is used to optimize game strategies and decision-making processes. For example, it can fine-tune the behavior of non-player characters (NPCs) in video games to make them more challenging or responsive.



Additional Resources

- A* Search Algorithms

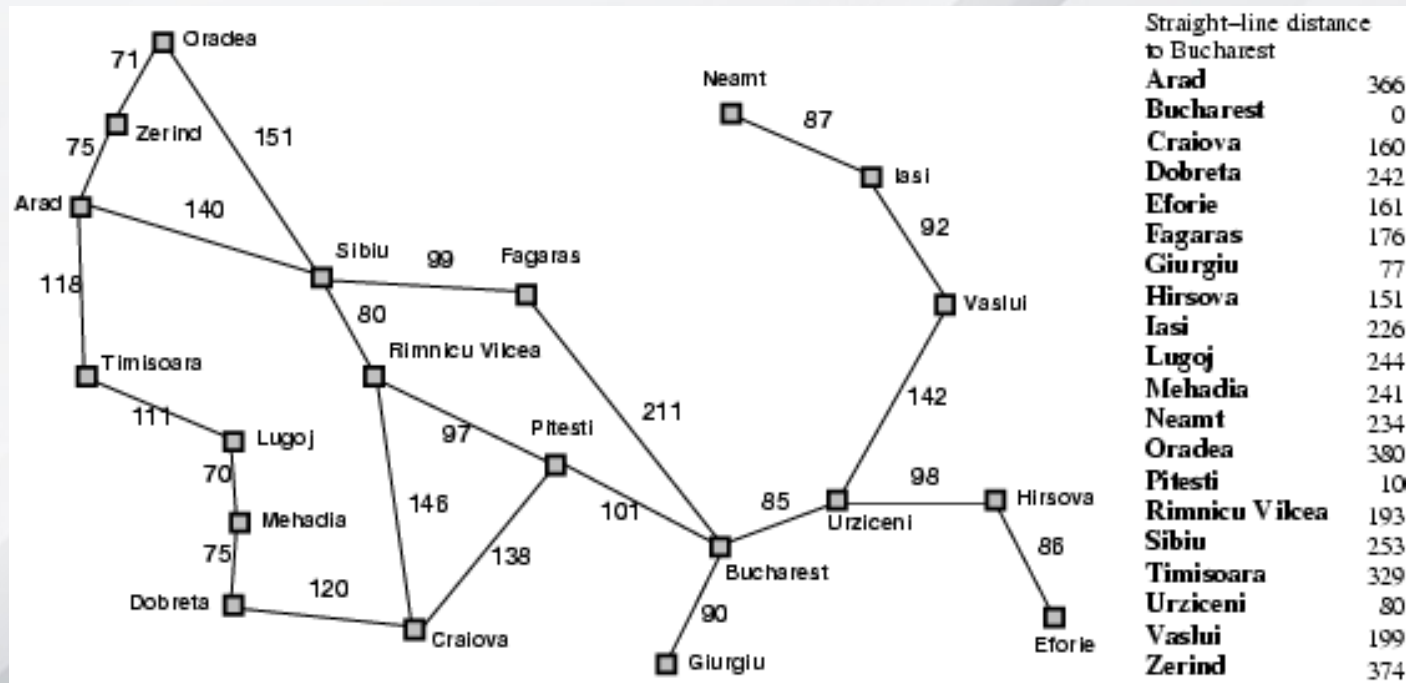
<https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search>

- Greedy-best-first algorithms

<https://www.codecademy.com/resources/docs/ai/search-algorithms/greedy-best-first-search>

Exercises

- Consider the following problem from Arad to Budapest :
- Perform a Uniform Cost Search to find the shortest path.





مجموعة جامعة
الجميع الذكية
للتعليم القابضة



Thank You

