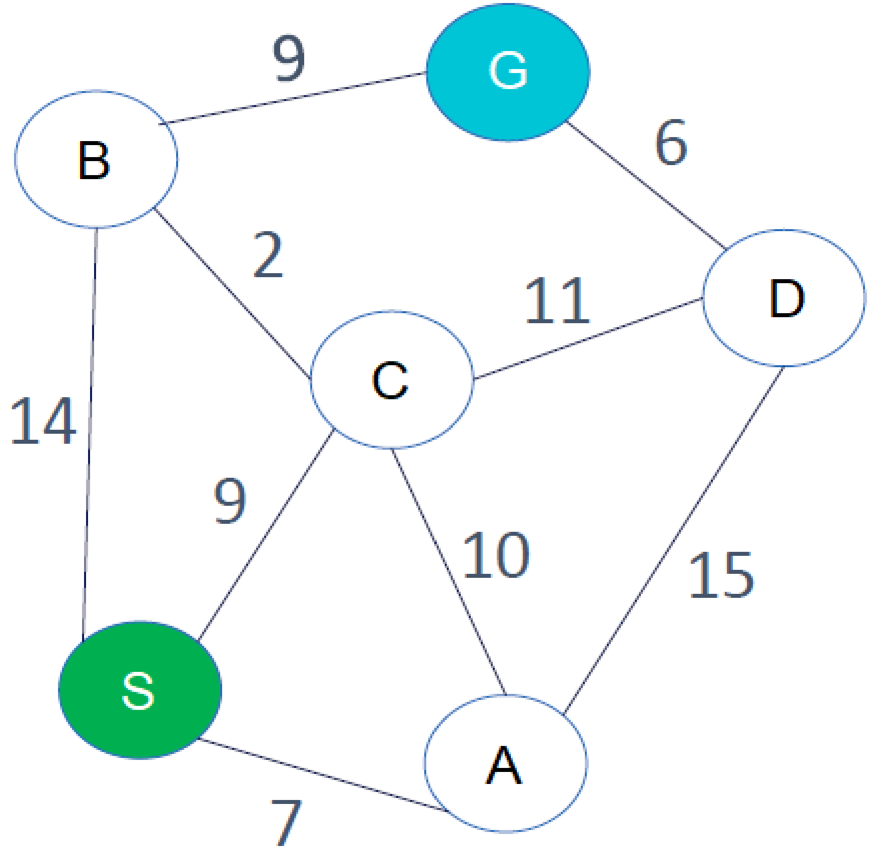
Introduction to
Artificial Intelligence in Games

Lecture 4

Uniform-Cost Search

Task

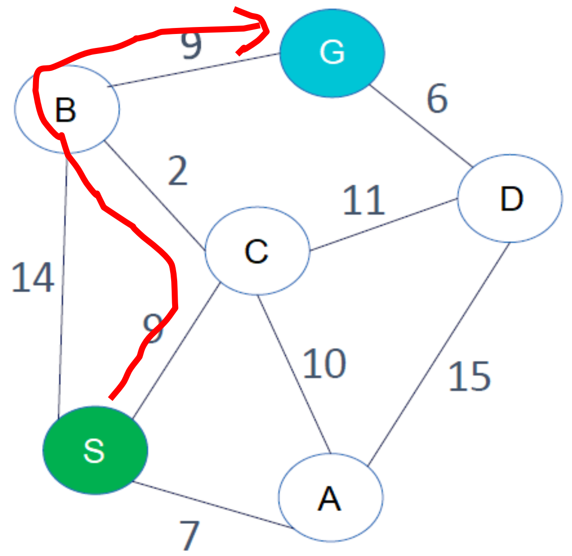
- Construct the search tree and find the optimum path from S to G using the uniform-cost search algorithm.

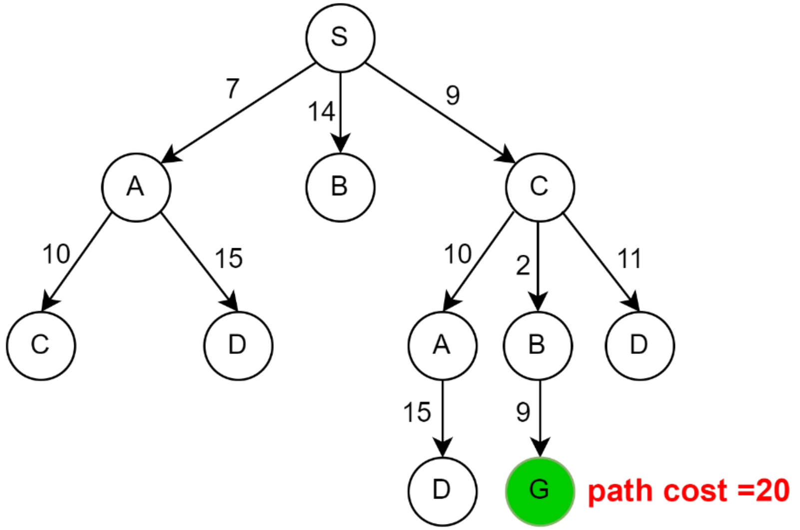


Solution

Expanded nodes list (sort then expand)

- S0 {A7, C9, B14 }
- A7 { C9, B14, ~~C17~~, D22 }
- C9 { B11, ~~B14~~, A19, D20, ~~D22~~ }
- B11 { A19, G20, D20 }
- A19 { G20, D20, ~~D34~~ }
- G20
- Solution path found is S → C → B → G, cost= 20





Informed Search Methods

Informed Search Algorithms

- In informed search algorithm, we have some knowledge about the goal, such as how far we are from the goal (straight line distance), how many wrong-placed tiles in 8-puzzule game, etc.
- This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- Informed search algorithm uses the idea of heuristic, so it is also called **Heuristic search**.
- In a heuristic search, each state is assigned a “heuristic value” (h-value) that the search uses in selecting the “best” next step.
- Heuristic h(n) gives an estimate of the distance from n to the goal.
- h(n)=0 for goal nodes

Greedy Best-first Search

- The basic idea of best first search is similar to uniform cost search.
- The only difference is that the “cost” (or in general, the evaluation function) is estimated based on some heuristics h(n) which represents how far we are from the goal, rather than the accumulated cost g(n) calculated during search.
- In the best first search algorithm, we expand the node which is closest to the goal node.
- Similar to the uniform cost search, the greedy best-first algorithm is implemented by the priority queue, with the node that has the lowest heuristic value h is processed first.

Greedy Best-first Search

Steps

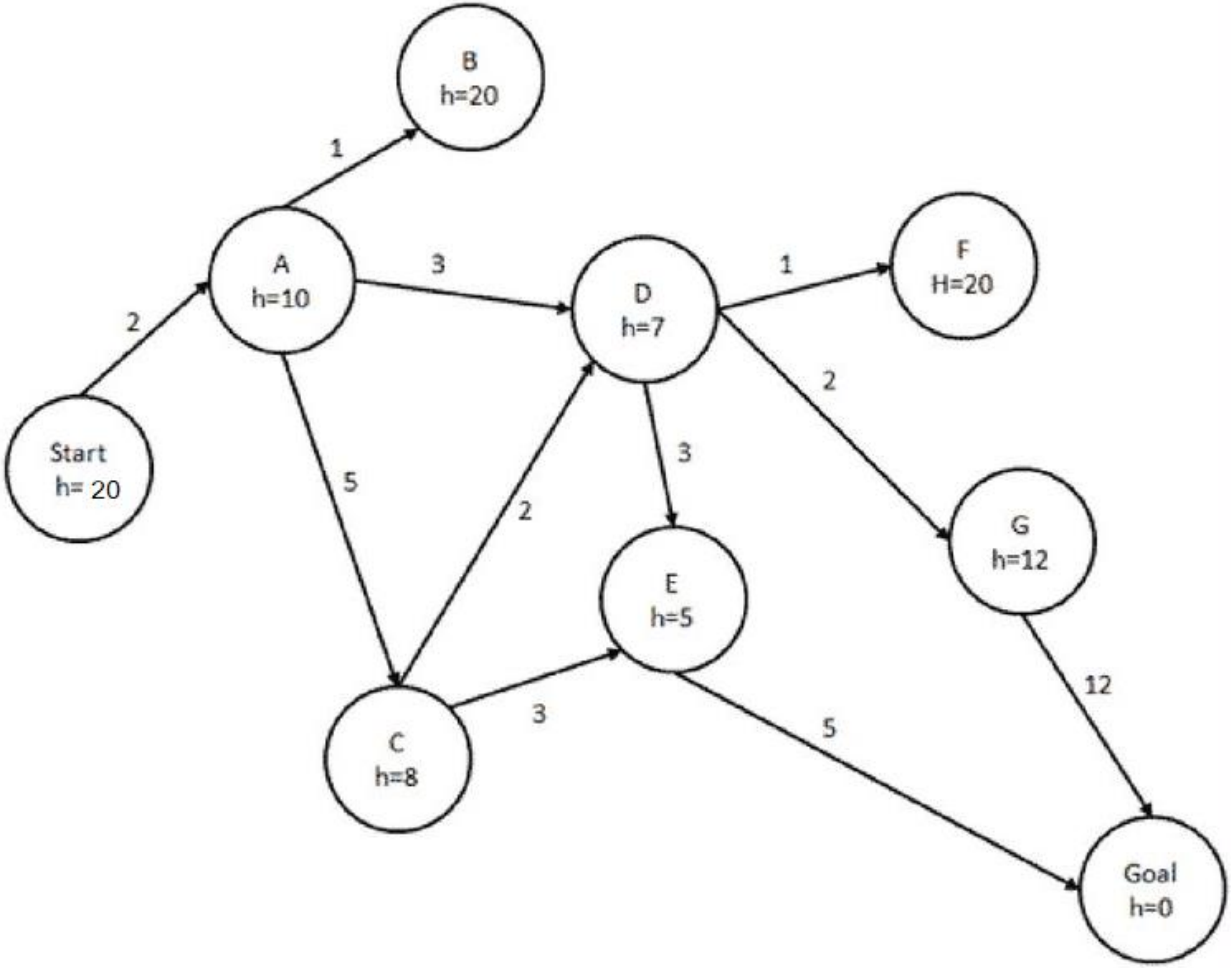
- Enqueue nodes by their heuristic value.
- That is, let h(n) = cost (straight line distance) of the path from the current node n to the goal node.
- Sort nodes by ascending value of h
- If there is more than one path to the same node, keep only the path with the lowest cost in the queue.
- Stop when the goal is the first node in the queue

  Else, expand nodes starting from the first node in the queue
- Ignore back paths to the start node.
- Ignore back paths to the parent node.

Greedy Best-first Search

Example

- Find the path from the start node to the goal node using the greedy best-first search.
- The heuristic value h (n) for each node is shown on the node.

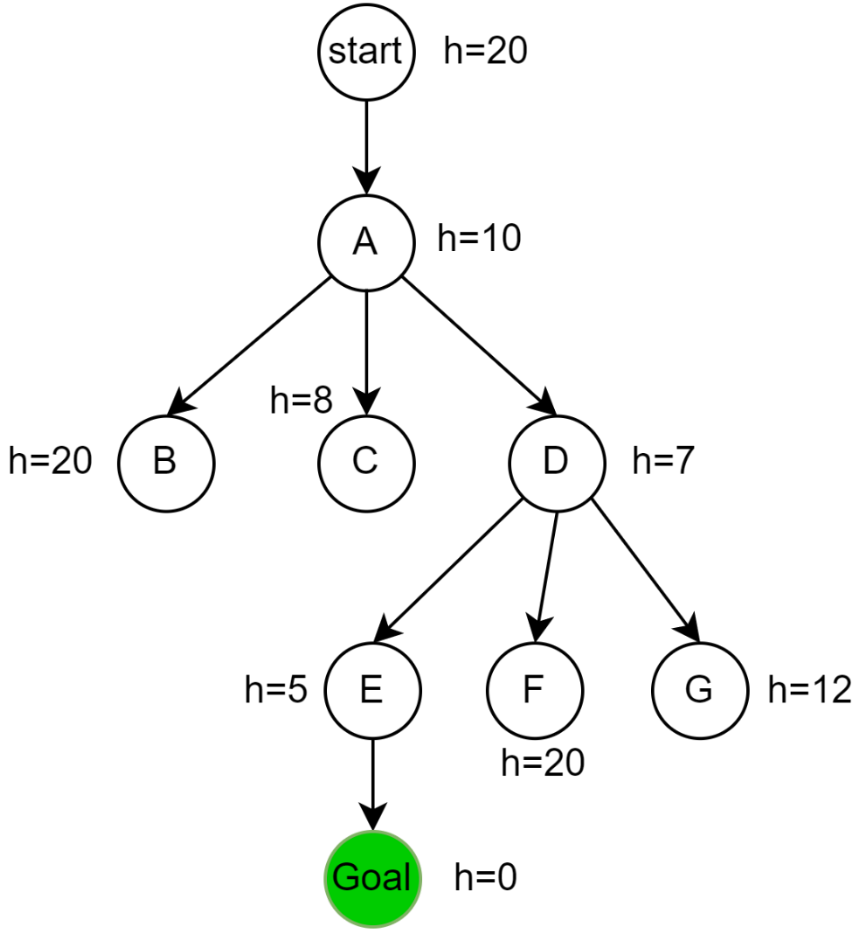


Greedy Best-first Search

Solution

Expanded nodes list (sort then expand)

- Start { A10 }
- A10 { D7, C8, B20}
- D7 { E5, C8, G12, B20, F20 }
- E5 { Goal }
- Goal
- Solution path found is Start → A → D → E → Goal



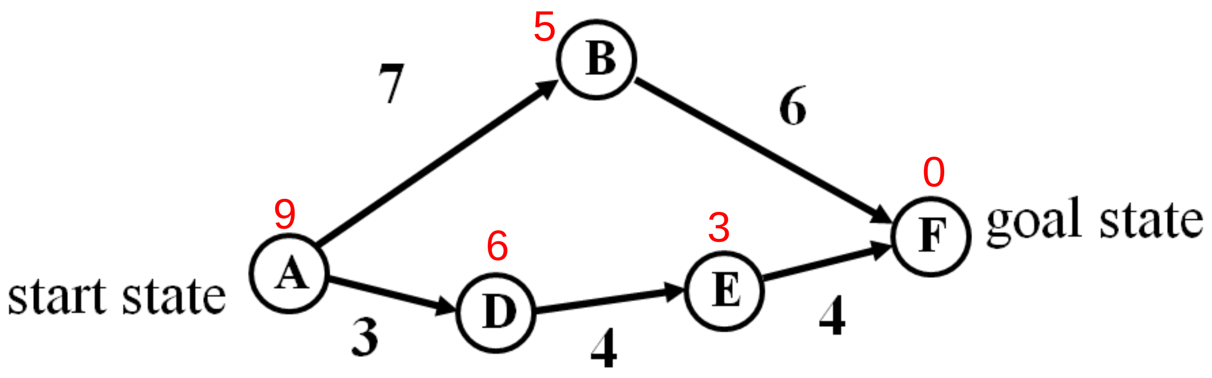
Greedy Best-first Search

Completeness

- Greedy best first search algorithm is complete as in worst case scenario it will search the whole space (worst option).

Optimality

- Greedy best first search algorithm is not optimal.
- A solution can be found in a longer path (lower h(n) with a higher g(n) value).

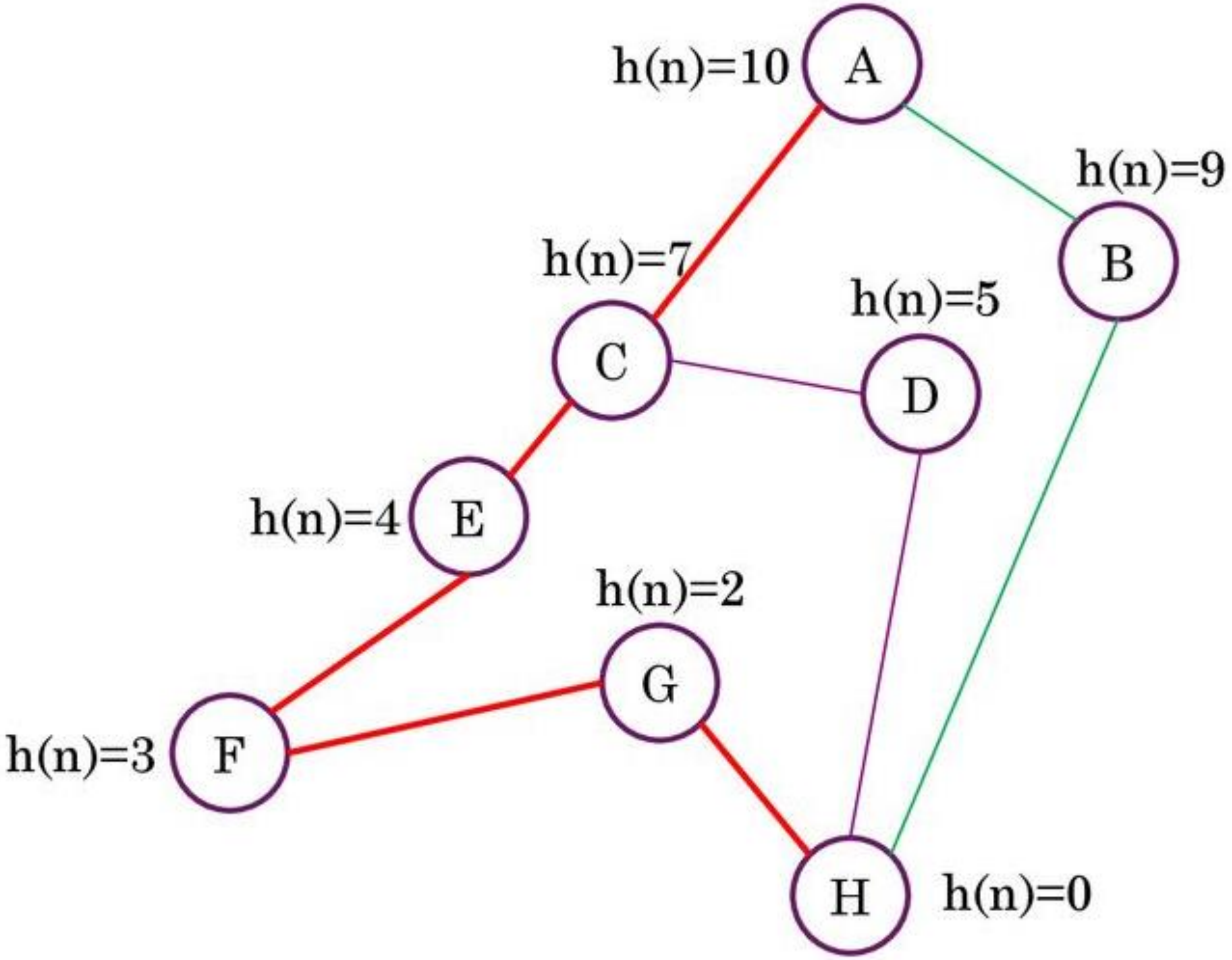


- Say: h(A) = 9, h(B) = 5, h(D) = 6, h(E) = 3, h(F) = 0
- It does not take the actual cost into account

Greedy Best-first Search

Another example

- The greedy best first search will find the solution path:
  A → C → E→ F→ G→ H(Goal)
- However, this path is the longest path in the graph to the goal and processes more nodes.



A* Search Algorithm

- A* Algorithm is one of the best and popular techniques used for path finding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently.
- It combines features of UCS and greedy best-first search, by which it solve the problem efficiently.
- It uses the evaluation function $f(n) = g(n) + h(n)$

  - $g(n)$ is the cumulative cost from the start node to the node *n*

  - $h(n)$ is the heuristic (estimated) cost from the node *n* to the goal

  - $f(n)$ is the estimated total cost of path through *n* to the goal

A* Search Algorithm

Advantages:

- The A* algorithm can obtain the best solution because it considers both the cost calculated up to now and the estimated future cost.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

A* Search Algorithm

Steps

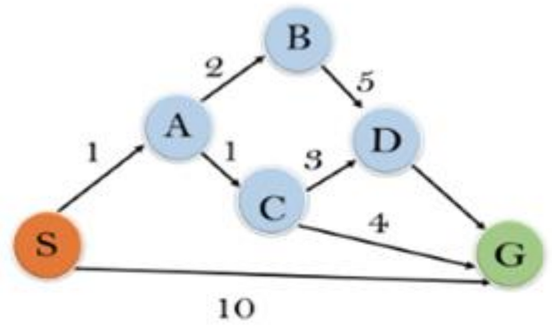
- Enqueue nodes by their evaluation function $f(n) = g(n) + h(n)$.
- Sort nodes by ascending value of f
- If there is more than one path to the same node, keep only the path with the lowest cost in the queue.
- Stop when the goal is the first node in the queue

  Else, expand nodes starting from the first node in the queue
- Ignore back paths to the start node.
- Ignore back paths to the parent node.

A* Search Algorithm

Example

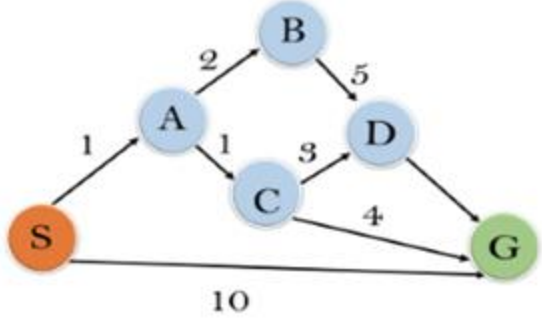
- In this example, we will traverse the given graph using the A* algorithm.
- The heuristic value of all states is given in the table.



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

A* Search Algorithm

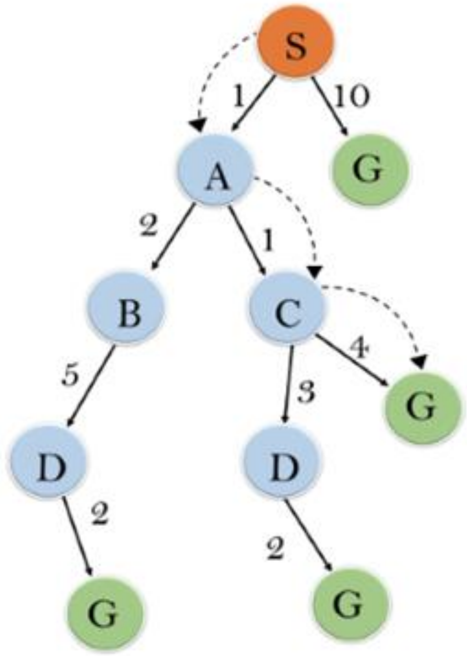
Solution



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

Expanded nodes list

- S { f(A), f(G) } not sorted
  f(A)=g(A)+h(A)=1+5=6
  f(G) = g(G)+h(G) = 10+0=10
  S { A6, G10 } sorted
- A6 { G10, f(B), f(C) } not sorted
  f(B)=g(B)+h(B)=3+4=7
  f(C) = g(C)+h(C) = 2+2=4
  A6 { C4, B7, G10 } sorted
- C4 { B7, G10, f(D), f(G)} not sorted
  f(D)=g(D)+h(D)=5+6=11
  f(G) = g(G)+h(G) = 6+0=6
  C4 { G6, B7, G10, D11 } sorted
- G6
- Solution path found is S → A →C → G



A* Search Algorithm

Practical Examples: 8-Puzzle

- Given an initial state of an 8-puzzle problem and final state to be reached.
- Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
- Consider the cost from one state to another = 1 (which will be used to get g(n)), and h(n) = Number of misplaced tiles.

2	8	3
1	6	4
7		5

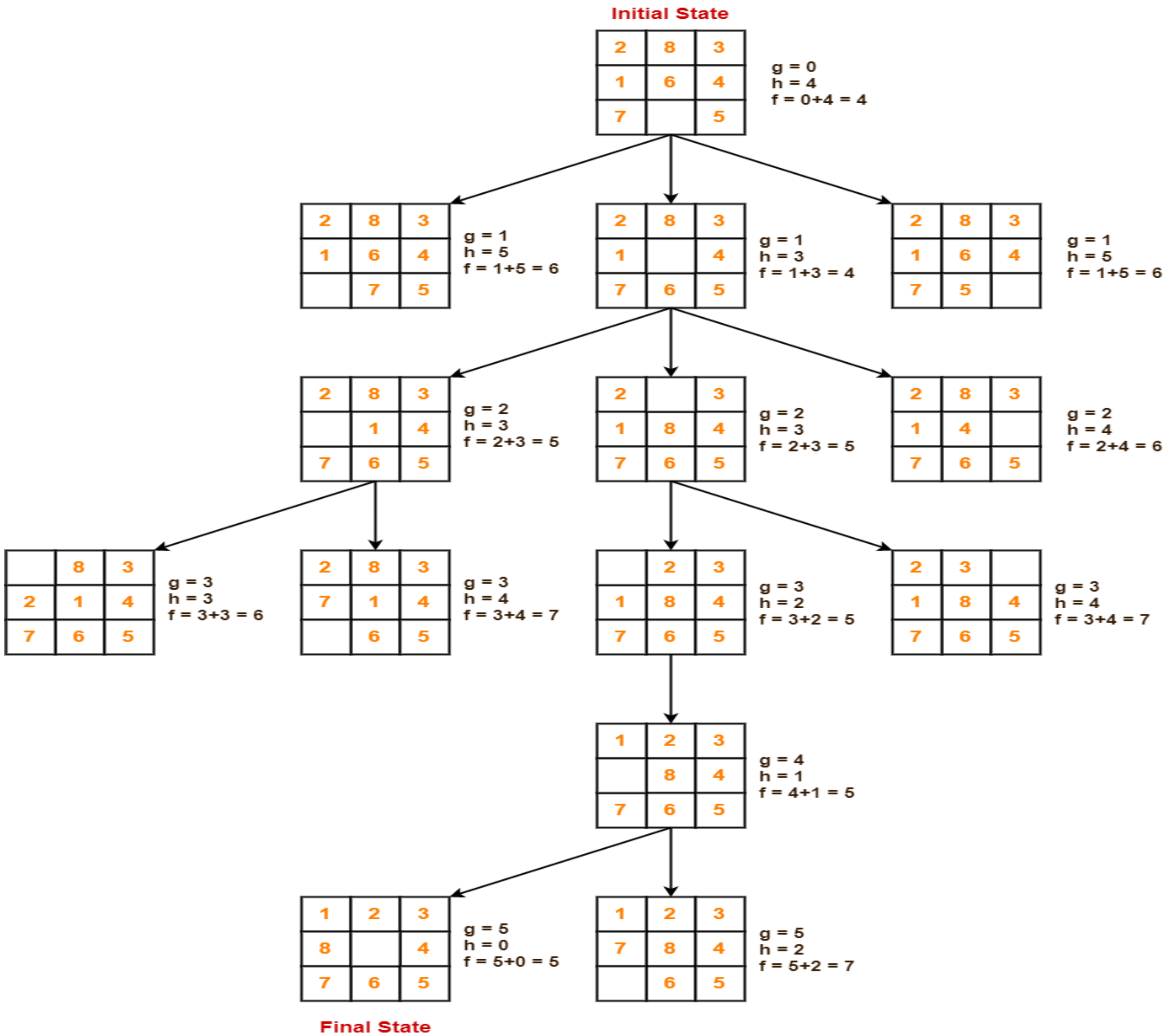
Initial State

1	2	3
8		4
7	6	5

Final State

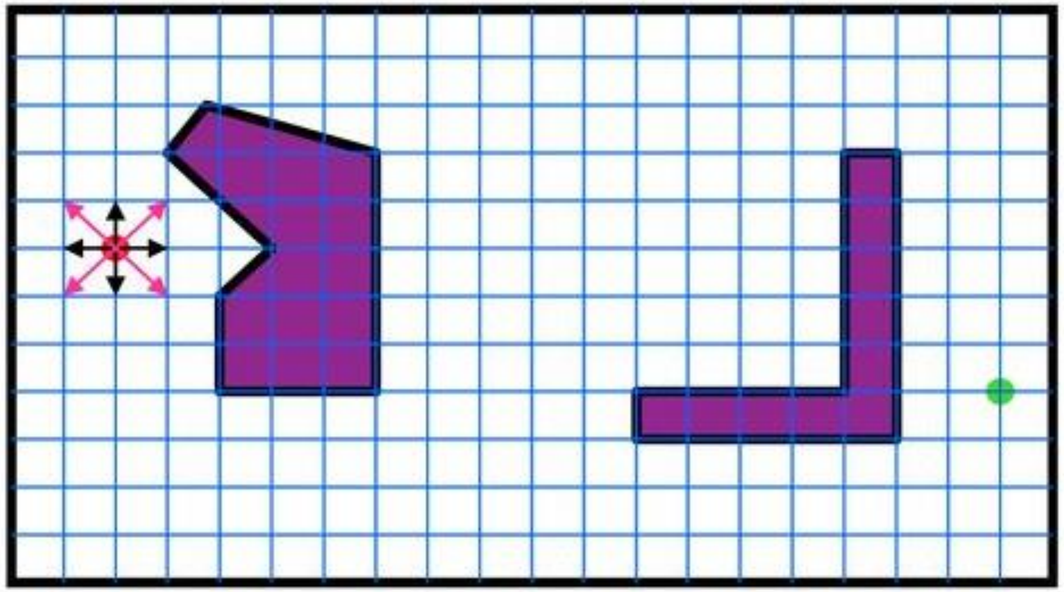
A* Search Algorithm

Solution



A* Search Algorithm

Practical Examples: Robot Navigation

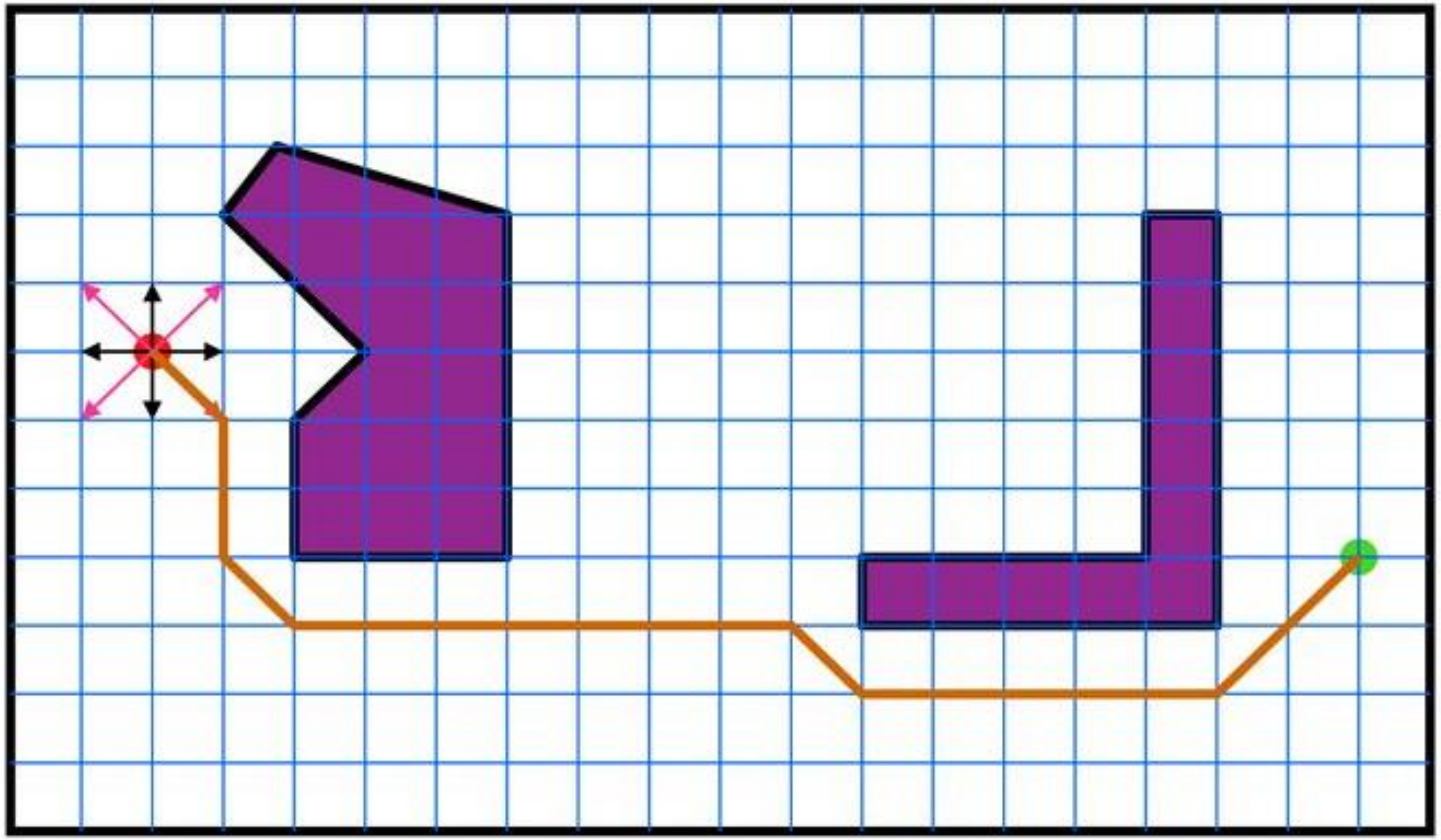


- $f(n) = g(n) + h(n)$, with $h(n)$ = straight-line distance from node n to goal
- $h(n) = \sqrt{(x_g - x_n)^2 + (y_g - y_n)^2}$
- $g(n)$: Cost of one horizontal/vertical step = 1

  Cost of one diagonal step = $\sqrt{2}$

A* Search Algorithm

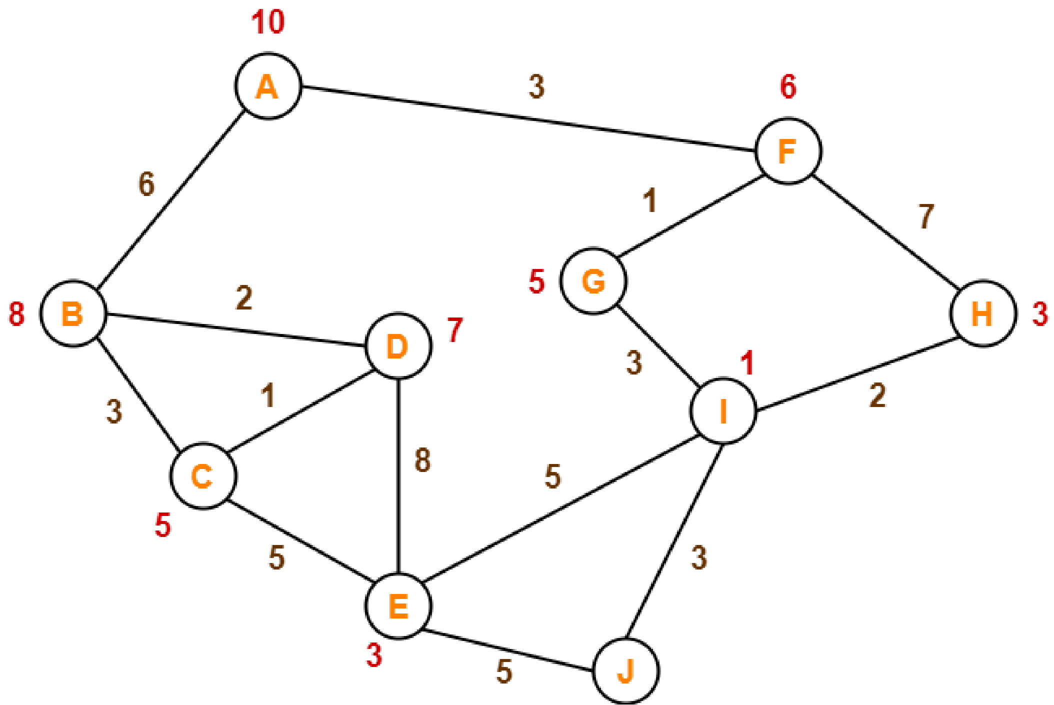
Solution: Robot Navigation



A* Search Algorithm

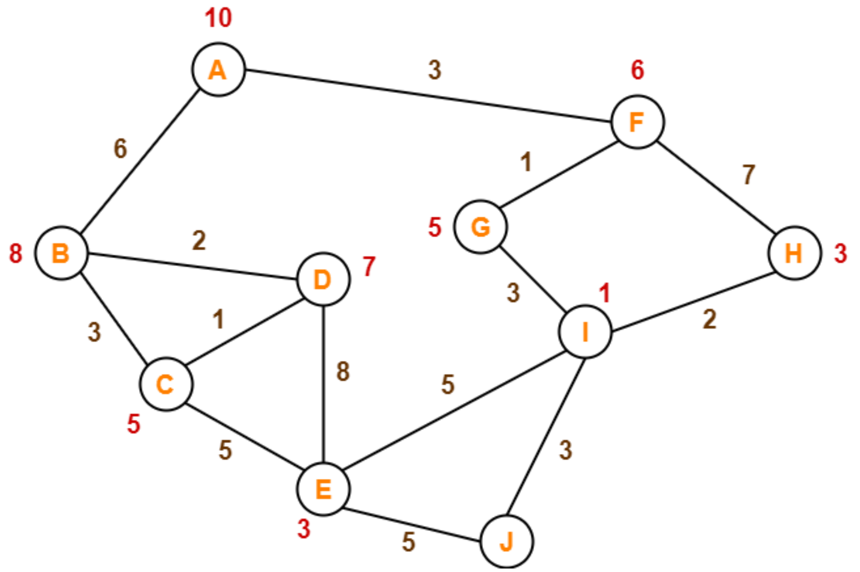
Practical Examples: Map

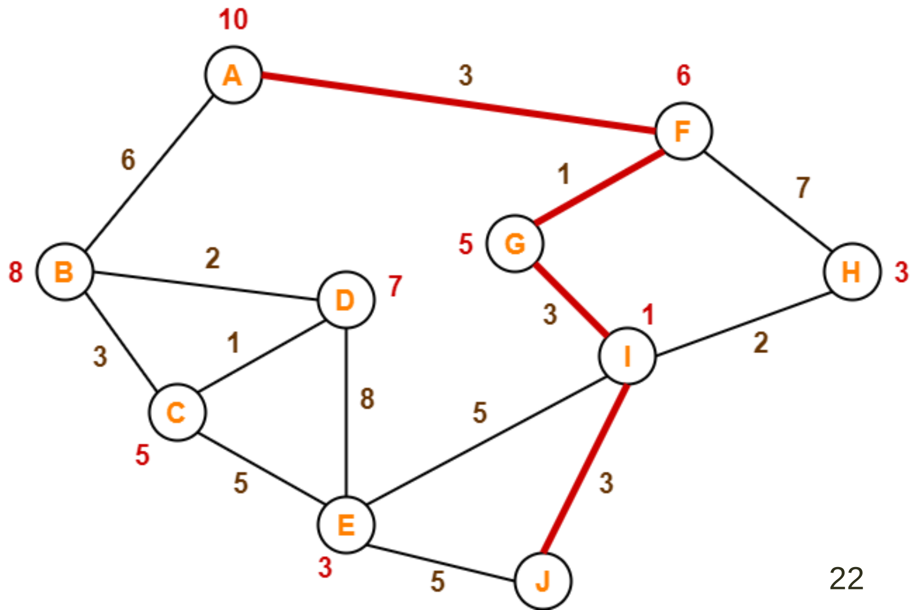
- Consider the following map of cities
- The numbers written on edges represent the distance between the cities.
- The numbers written on nodes represent the straight-line distances to the target city.
- Find the most cost-effective path to reach from start city A to target city J using A* Algorithm.



A* Search Algorithm

Solution

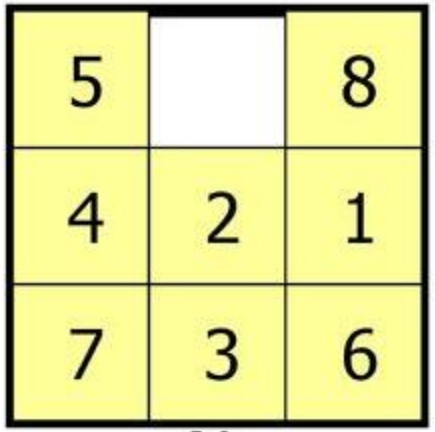




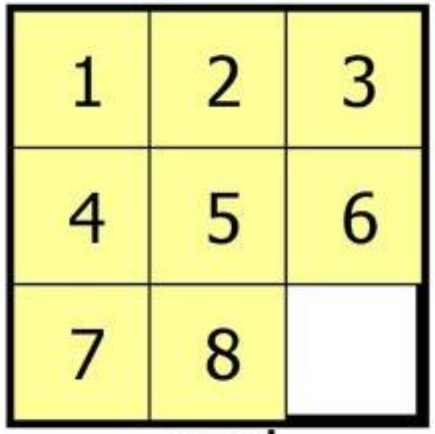
A* Search Algorithm

Practical Examples: 8-Puzzle

- Given an initial state of an 8-puzzle problem and final state to be reached.
- Find the most cost-effective path to reach the final state from initial state using A* Algorithm. Assume the cost of one move = 1



N



goal

- You can take the heuristic function h(n) as:

  - Number of misplaced tiles

  - Sum of distances of each tile to goal


Any Questions