# A* Search Algorithm

**Topic-1:** **Graph Representation**

-Adjacency Matrix
-Adjacency List

**Topic-2:** **Type of Searching Techniques**

-Uninformed Searching
  -BFS, DFS, etc.
  -Cons: Time Consuming & Complexity
-Informed Searching
  - A*, Heuristic DFS, Best First Search, etc.
  -Pros: Provides quick solution & less Complexity

**Topic-3:** **Heuristic Function & Heuristic Value**

**Topic-4:** **Admissibility**

**-Underestimation**
**-Overestimation**

A given heuristic function **h(n)** is **admissible** if it never **overestimates** the real distance between n and the goal node.

Therefore, for every node n the following formula applies:
$\mathbf{h(n) \leq h^*(n)}$

**Topic-5:** **Consistency**

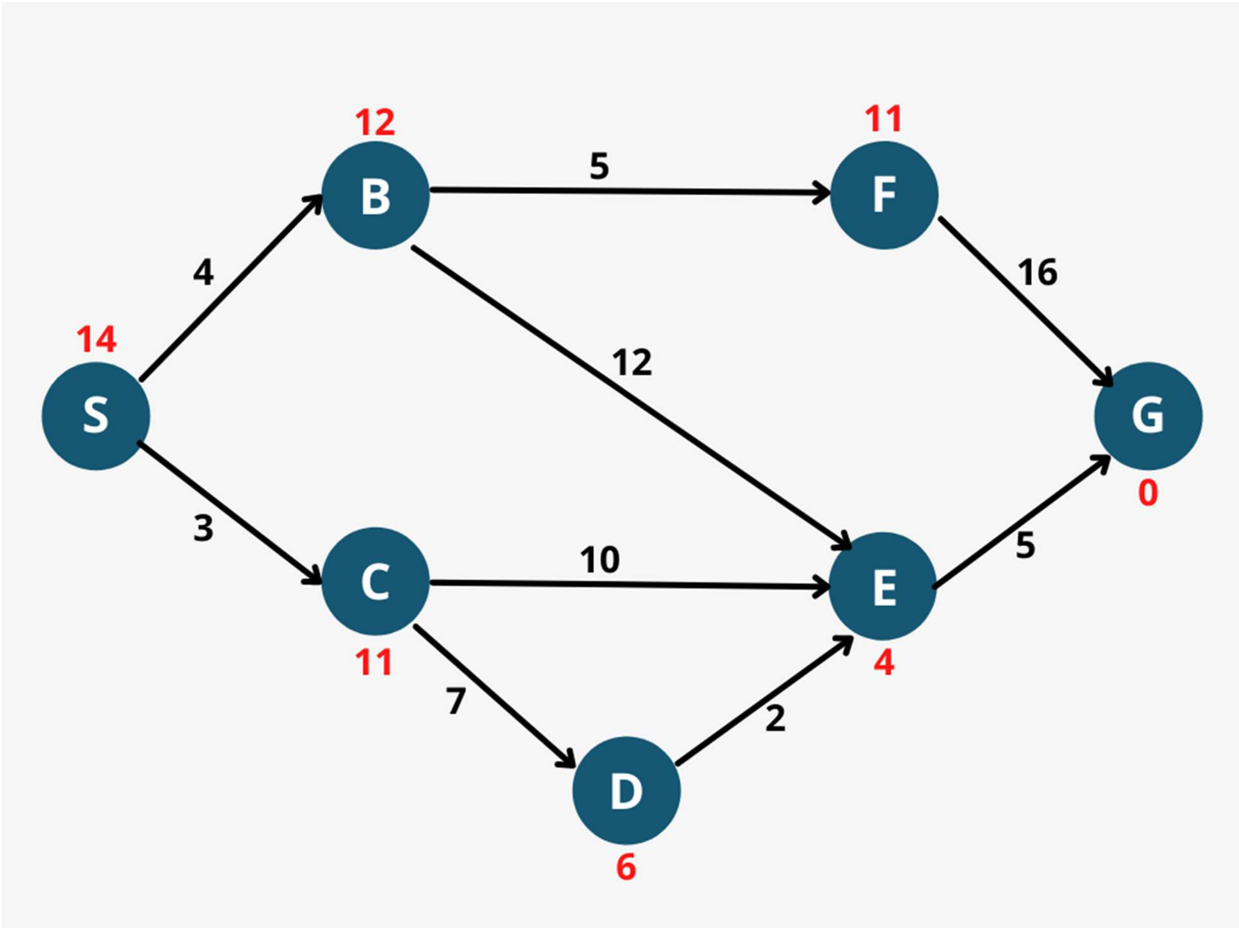
A given heuristic function h(n) is **consistent** if the estimate is always less than or equal to the estimated distance between the goal n and any given neighbor, plus the estimated cost of reaching that neighbor:
$\mathbf{c(n, m) + h(m) \geq h(n)}$

**Topic-6:** **A* Search Algorithm**

A* is based on using heuristic methods to achieve **optimality** and **completeness**, and is a variant of the **best-first algorithm**.

When a search algorithm has the property of optimality, **it means it is guaranteed to find the best possible solution**, in our case the shortest path to the finish state. When a search algorithm has the property of completeness, **it means that if a solution to a given problem exists, the algorithm is guaranteed to find it**.

# A* Search Algorithm




Initialize a min priority queue **Q** *ordered based on* ***f(n) = (g(n) + h(n))***

g ← 0
h ← euclidean_distance(**start_node**, **goal_node**)
f ← g(n) + h(n)
create **start_state**(**start_node**, **g(n)**, **f(n)**, parent=None)
insert **start_state** to **Q**

while **Q** not empty:
    **curr_state** ← extract_min(**Q**)
    if **curr_state.node** is the goalnode then
        print solution path and cost and **return**

    for each node **M** adjacent to node **curr_state.node**:

        g(n) ← **curr_state.g(n)** + edge_cost(**curr_state.node**, **M**)
        h(n) ← euclidean_distance(**M**, **goal_node**)
        f(n) ← g(n)+h(n)
        create **new_state**(**M**, **g**, **f**, parent=**curr_state**)
        Insert **new_state** to **Q**

# A* Search Algorithm

**Sample Input:**

7 9 (no_of _vertices no_of_ edges)

0 1 4 (starting_vertex_of_an_edge ending_vertex_of_an_edge weight_of_the_edge)

0 2 3

1 4 12

1 5 5

2 3 7

2 4 10

3 4 2

4 6 5

5 6 16

14 12 11 6 4 11 0

**Sample Output:**

**Path**: 0 --> 2 --> 3 --> 4 --> **6**

**Total Actual Cost**: 17

**References:**

https://stackabuse.com/basic-ai-concepts-a-search-algorithm/

https://www.pythonpool.com/a-star-algorithm-python/

**Tutorials:**

https://www.youtube.com/watch?v=vP5TkF0xJgI&ab_channel=GeeksforGeeks

https://www.youtube.com/watch?v=tvAh0JZF2YE&ab_channel=GateSmashers