Informed Search/ Heuristic search

- Problem information is available which can guide the search.
- Informed search strategies can find a solution more efficiently than an uninformed search strategy.
- Informed search is also called a Heuristic search.
- Heuristic function facilitates in imparting the additional knowledge of the problem to the algorithm in finding the way to the goal through the various neighboring nodes.

Heuristic Function

- A heuristic function h(n), is a function that calculates an approximate cost to a problem.
- Example: shortest driving distance
- A heuristic cost would be the straight line distance to the point.
- It is *simple and quick* to calculate. The true distance would likely be higher as we have to stick to roads and is much harder to calculate.
- Heuristic functions are often used in combination with search algorithms.
- You may also see the term *admissible*, which means the heuristic never overestimates the true cost i.e. the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path.
- *Admissibility* can be an important quality and is required for some search algorithms like A*

Best First Search

- A node is selected for expansion based on evaluation function f(n).
- Lowest evaluation is selected for the explanation
- Implemented via a priority queue
- Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path
- At each step it tries to get as close to the goal as it can

Properties of greedy best-first search

- Complete? No – It can start down an infinite path and never return to try other possibilities
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement. m is the max depth
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

A* search

- A* is a path-finding and graph traversals algorithm with completeness and optimality.
- Idea: avoid expanding paths that are already expensive
- ***A* search*** is a combination of lowest-cost-first and best-first searches that considers both path cost and heuristic information in its selection of which path to expand.
- *A** is implemented via priority queue

- Similar to Dijkstra's Algorithm with heuristic

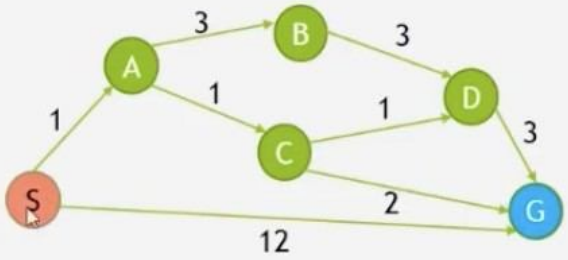
A* search

g(n)=cost of getting to node from start state

h(n)=heuristic function

h(n) = estimated cost of the cheapest path from node *n* to a goal state.

f(n)= estimated cost of the cheapest solution through n



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

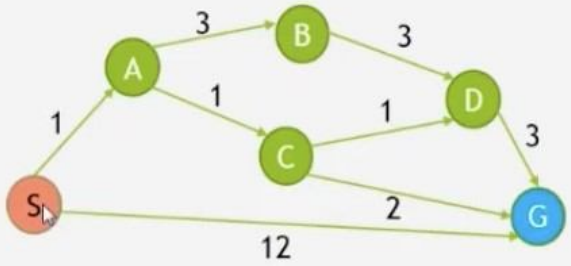
state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

A* search

g(n)=cost of getting to node from start state

h(n)=heuristic function

f(n)= estimated cost of the cheapest solution through n



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

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

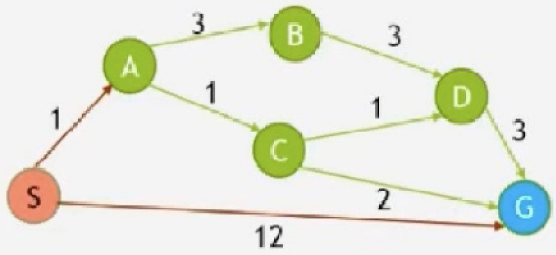
f(S)=g(S)+h(S)
=(0)+4
=4

A* search

g(n)=cost of getting to node from start state

h(n)=heuristic function

f(n)= estimated cost of the cheapest solution through n



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

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

f(S)=g(S)+h(S)
=(0)+4
=4

f(A)=g(A)+h(A)
=(1)+2
=3

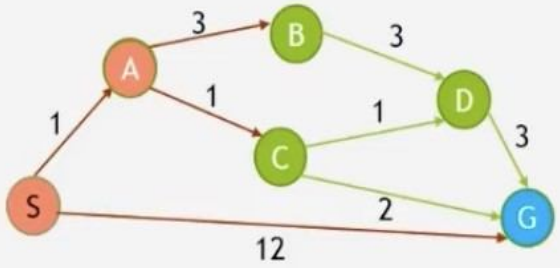
f(G)=g(G)+h(G)
=(12)+0
=12

A* search

g(n)=cost of getting to node from start state

h(n)=heuristic function

f(n)= estimated cost of the cheapest solution through n



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

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

Queue={ [S->A->C,4] , [S->A->B,10] , [S->G,12] }

f(S)=g(S)+h(S)
=(0)+4
=4

f(A)=g(A)+h(A)
=(1)+2
=3

f(G)=g(G)+h(G)
=(12)+0
=12

f(C)=g(C)+h(C)
=(1+1)+2
=4

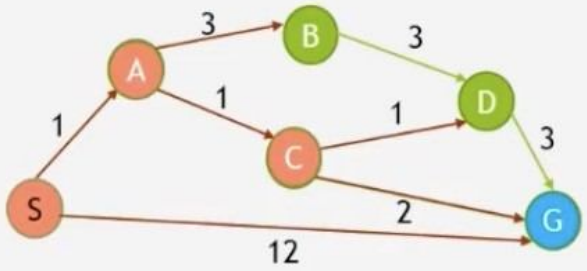
f(B)=g(B)+h(B)
=(1+3)+6
=10

A* search

g(n)=cost of getting to node from start state

h(n)=heuristic function

f(n)= estimated cost of the cheapest solution through n



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

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

Queue={ [S->A->C,4] , [S->A->B,10] , [S->G,12] }

Queue={ [S->A->C->G,4] , [S->A->C->D,6] , [S->G,12] , [S->A->B,10] , [S->G,12] }

f(S)=g(S)+h(S)
=(0)+4
=4

f(A)=g(A)+h(A)
=(1)+2
=3

f(G)=g(G)+h(G)
=(12)+0
=12

f(C)=g(C)+h(C)
=(1+1)+2
=4

f(B)=g(B)+h(B)
=(1+3)+6
=10

f(D)=g(D)+h(D)
=(1+1+1)+3
=6

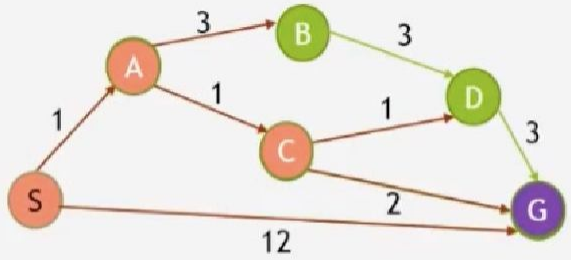
f(G)=g(G)+h(G)
=(1+1+2)+0
=4

A* search

g(n)=cost of getting to node from start state

h(n)=heuristic function

f(n)= estimated cost of the cheapest solution through n



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

state	h(n)
S	4
A	2
B	6
C	2
D	3
G	0

Queue={ [S,4] }

Queue={ [S->A,3] , [S->G,12] }

Queue={ [S->A->C,4] , [S->A->B,10] , [S->G,12] }

Queue={ [S->A->C->G,4] , [S->A->C->D,6] , [S->G,12] , [S->A->B,10] , [S->G,12] }

Final path : S->A->C->G with cost = 4

f(S)=g(S)+h(S)
=(0)+4
=4

f(A)=g(A)+h(A)
=(1)+2
=3

f(G)=g(G)+h(G)
=(12)+0
=12

f(C)=g(C)+h(C)
=(1+1)+2
=4

f(B)=g(B)+h(B)
=(1+3)+6
=10

f(D)=g(D)+h(D)
=(1+1+1)+3
=6

f(G)=g(G)+h(G)
=(1+1+2)+0
=4

Admissibility of *A**

- *A** always finds an optimal path, if one exists, and that the first path found to a goal is optimal is called the **admissibility** of *A**.

- Admissibility means that, even when the search space is infinite, if solutions exist, a solution will be found and the first path found will be an optimal solution - a lowest-cost path from a start node to a goal node.

Pseudocode for the A* algorithm is presented with Python-like syntax

make an openlist containing only the starting node https://brilliant.org/wiki/a-star-search/

```
make an empty closed list
while (the destination node has not been reached):
    consider the node with the lowest f score in the open list
    if (this node is our destination node) :
        we are finished
     if not:
        put the current node in the closed list and look at all of its neighbors
       for (each neighbor of the current node):
            if (neighbor has lower g value than current and is in the closed list) :
                replace the neighbor with the new, lower, g value
                 current node is now the neighbor's parent
                        else if (current g value is lower and this neighbor is in the open list ) :
               replace the neighbor with the new, lower, g value
                change the neighbor's parent to our current node
            else if this neighbor is not in both lists:
                add it to the open list and set its g
```