

# A\* Algorithm

# History

Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute  
(now SRI International) first published the algorithm in 1968.

It can be seen as an extension of Edsger Dijkstra's 1959 algorithm.

A\* achieves better performance by using heuristics to guide its search.

# concept

A\* traverses the graph, it follows a path of the lowest known cost, Keeping a sorted priority queue of alternate path along the way.

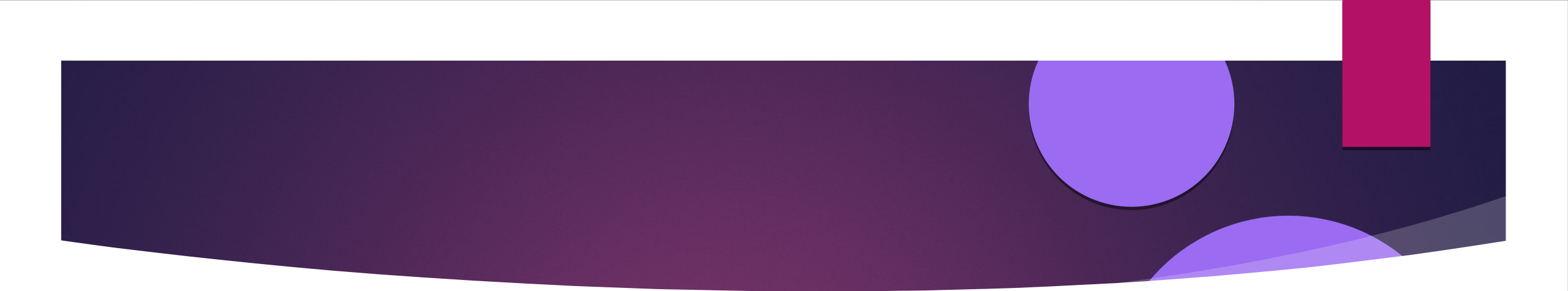
If, at any point, a segment of the path being traversed has a higher cost than another it abandons the higher-cost path segment and traverses the lower-cost path segment instead.

This process continues until the goal is reached.



## why we use this algorithm?

Because many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation) And flexible and can be used in a wide range of contexts.

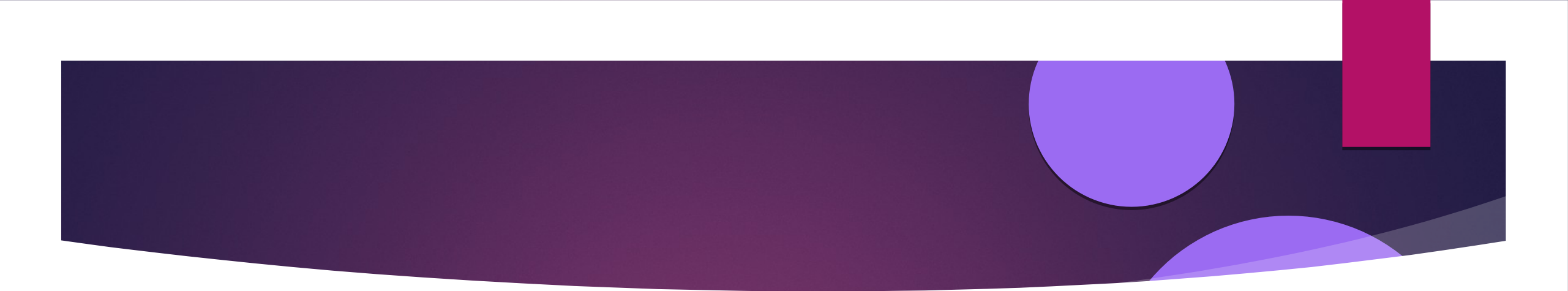


## what is A\* algorithm?

A\* is like Dijkstra's Algorithm in that it can be used to find a shortest path.

A\* is like Greedy Best-First-Search in that it can use a heuristic to guide itself.

In the simple case, it is as fast as Greedy Best-First-Search:



The secret to its success is that it combines the pieces of information that Dijkstra's Algorithm uses (favoring vertices that are close to the starting point) and information that Greedy Best-First-Search uses (favoring vertices that are close to the goal).



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

**$g(n)$** : represents the exact cost of the path from the starting point to any vertex

**$h(n)$** : represents the heuristic estimated cost from vertex  $n$  to the goal

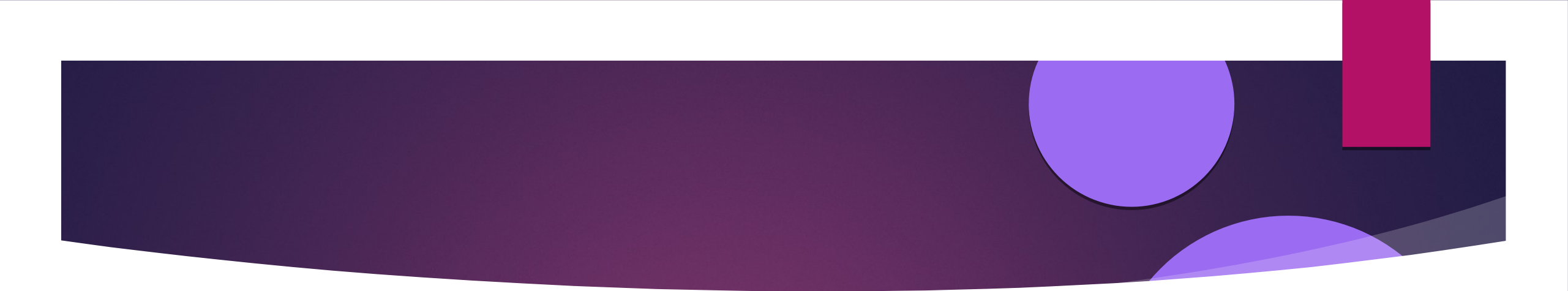
**$(h)$**  :represents vertices far from the goal and teal

**$(g)$**  :represents vertices far from the starting point

## A\*'s Use of the Heuristic:

The heuristic can be used to control A\*'s behavior.

- 1- At one extreme, if  $h(n)$  is 0, then only  $g(n)$  plays a role, and A\* turns into Dijkstra's Algorithm, which is guaranteed to find a shortest path.
- 2- If  $h(n)$  is always lower than (or equal to) the cost of moving from  $n$  to the goal, then A\* is guaranteed to find a shortest path. The lower  $h(n)$  is, the more node A\* expands, making it slower.
- 3- If  $h(n)$  is exactly equal to the cost of moving from  $n$  to the goal, then A\* will only follow the best path and never expand anything else, making it very fast. Although you can't make this happen in all cases, you can make it exact in some special cases.  
It's nice to know that given perfect information, A\* will behave perfectly.



4-If  $h(n)$  is sometimes greater than the cost of moving from  $n$  to the goal, then  $A^*$  is not guaranteed to find a shortest path, but it can run faster.

5-At the other extreme, if  $h(n)$  is very high relative to  $g(n)$ , then only  $h(n)$  plays a role, and  $A^*$  turns into Greedy Best-First-Search

So we have an interesting situation in that we can decide what we want to get out of  $A^*$ . With 100% accurate estimates, we'll get shortest paths really quickly. If we're too low, then we'll continue to get shortest paths, but it'll slow down. If we're too high, then we give up shortest paths, but  $A^*$  will run faster.





**Dijkstra's Algorithm** works well to find the shortest path,

but it wastes time exploring in directions that aren't promising.

Greedy Best First Search explores in promising directions but it may not find the shortest path.

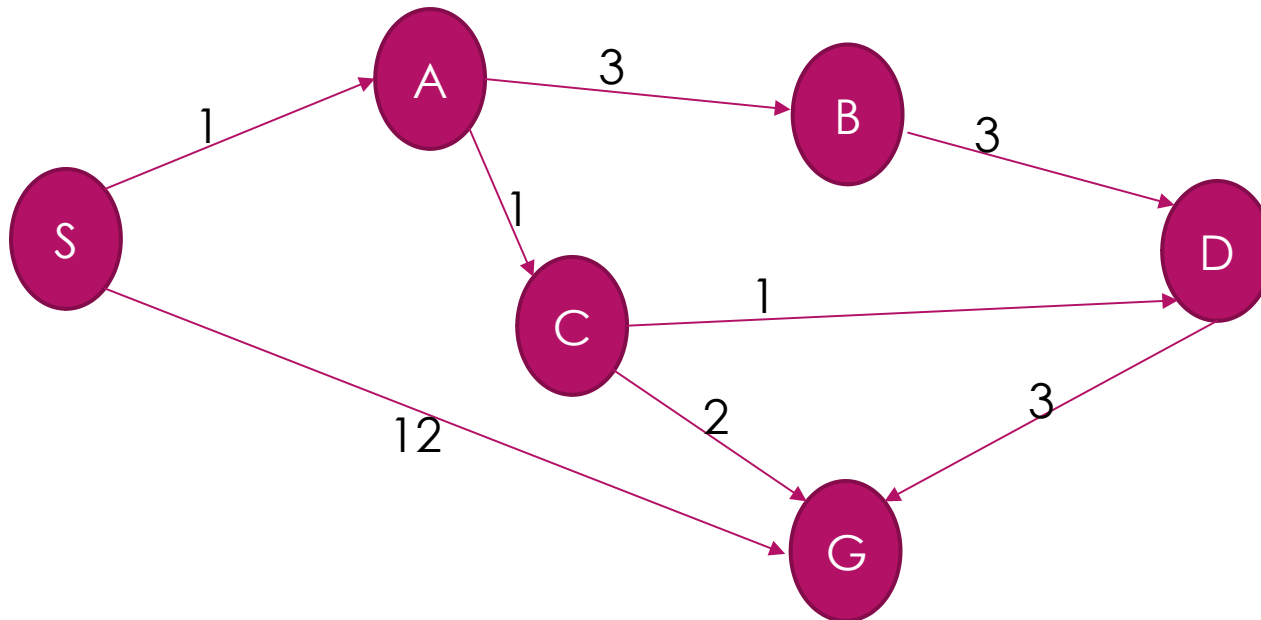
The A\* algorithm uses both the actual distance from the start and the estimated distance to the goal.



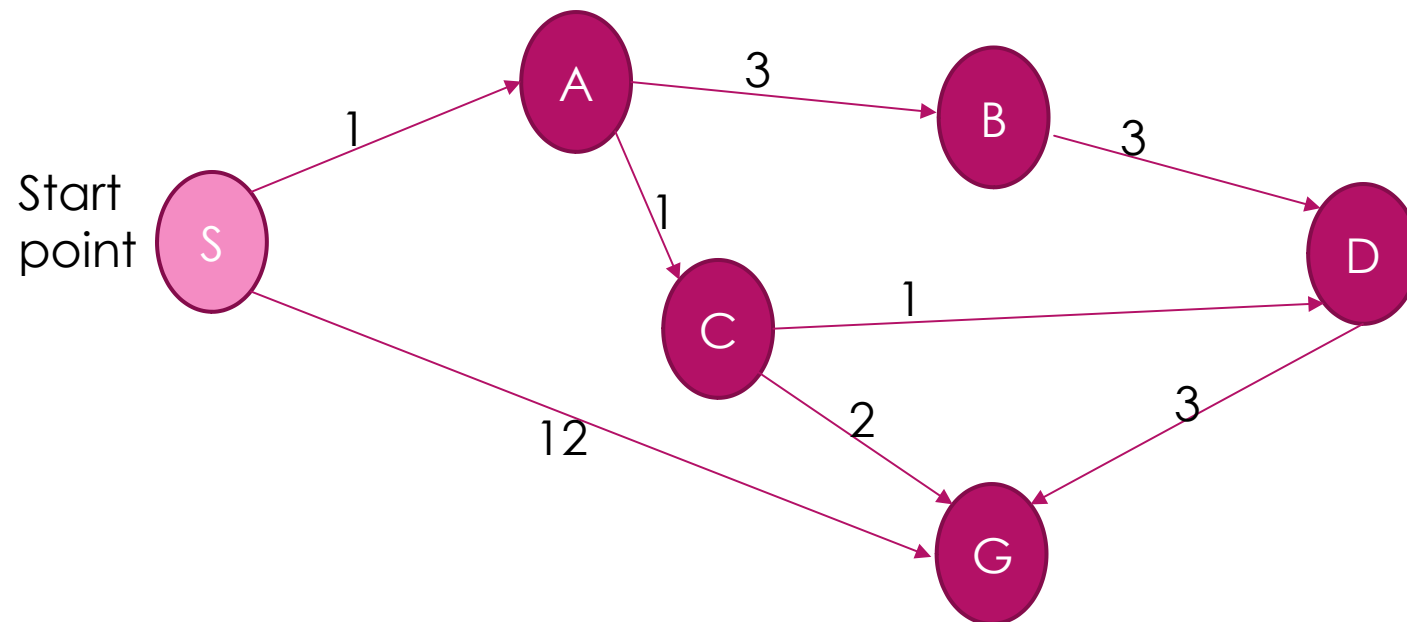
Example

# Exempel 1

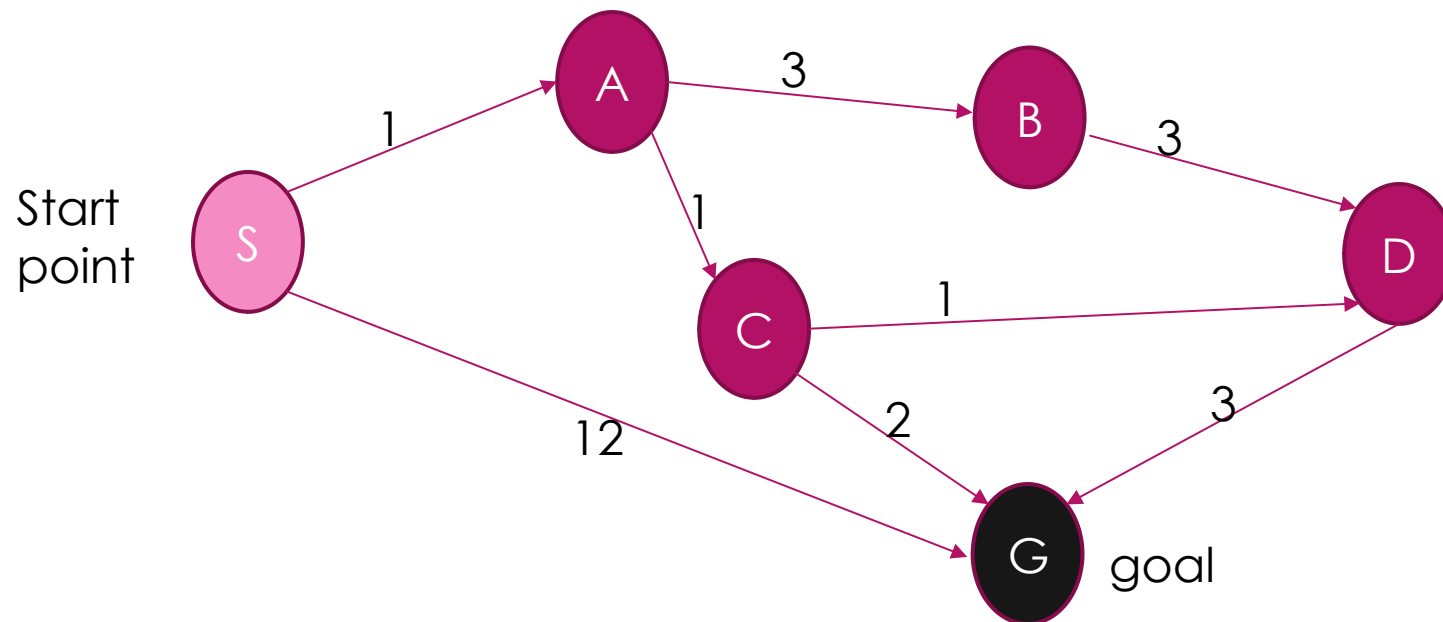
G	D	C	B	A	S	state
0	3	2	6	2	4	H(n)



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



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



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

1-By Tree

$$F(n) = g(n) + h(n)$$

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

$h(n)$  = heuristic function

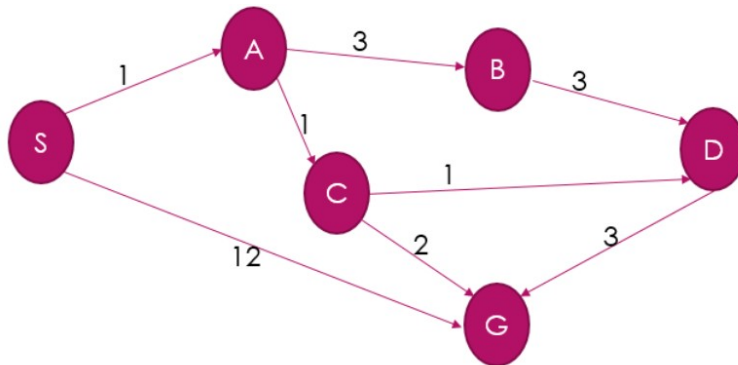
G	D	C	B	A	S	state
0	3	2	6	2	4	H(n)

$$F(n) = g(n) + h(n)$$

S

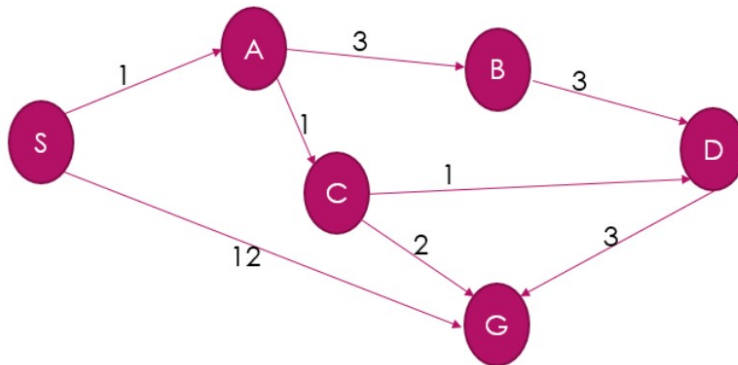
$$S$$

$$g+h=0+4=4$$

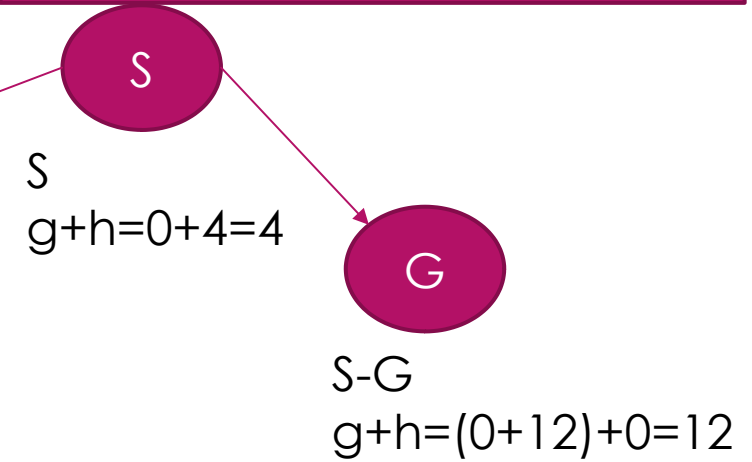


G	D	C	B	A	S	state
0	3	2	6	2	4	H(n)

$$F(n) = g(n) + h(n)$$



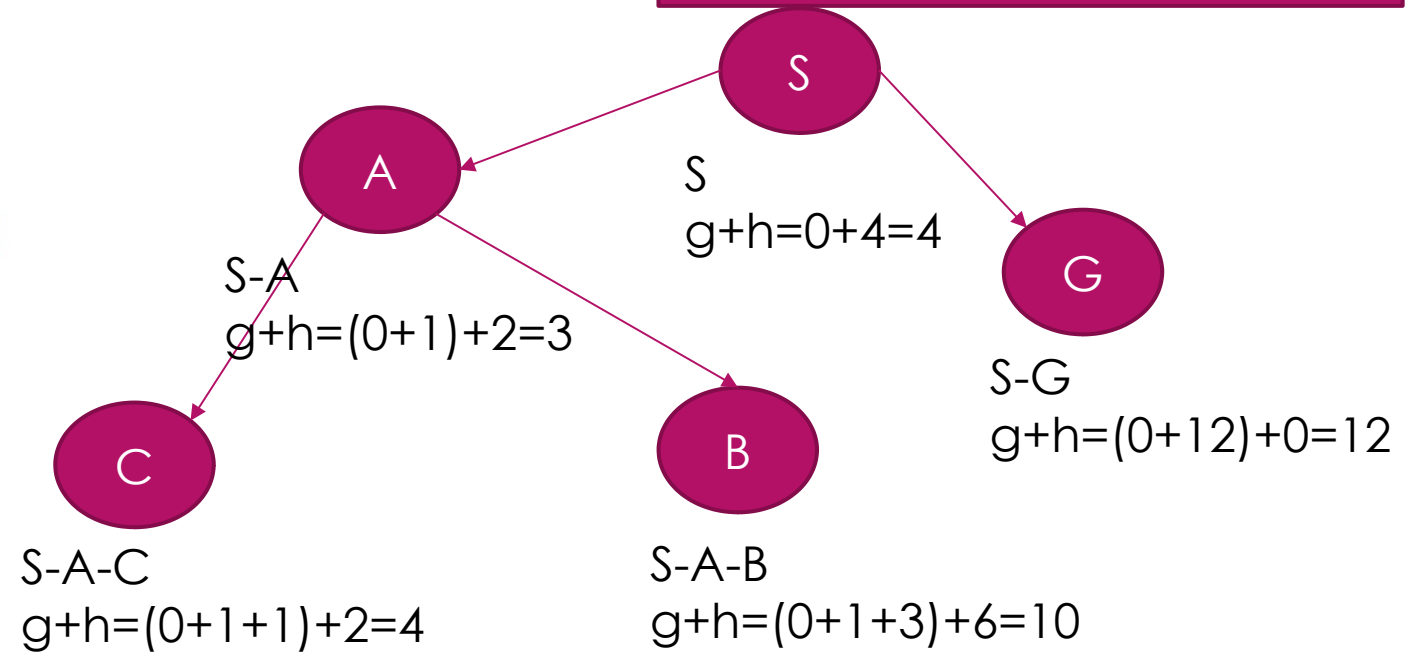
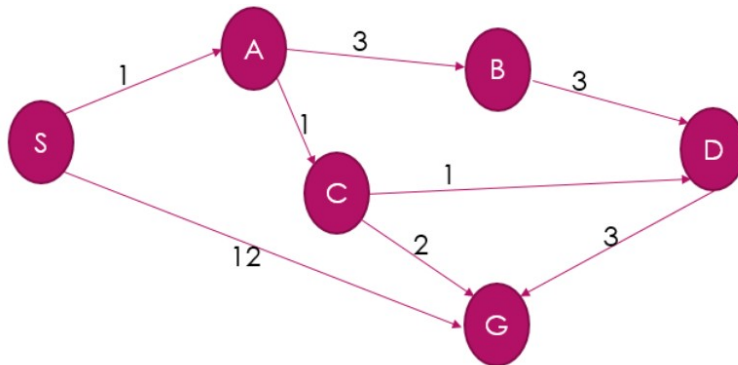
S-A  
 $g+h=(0+1)+2=3$





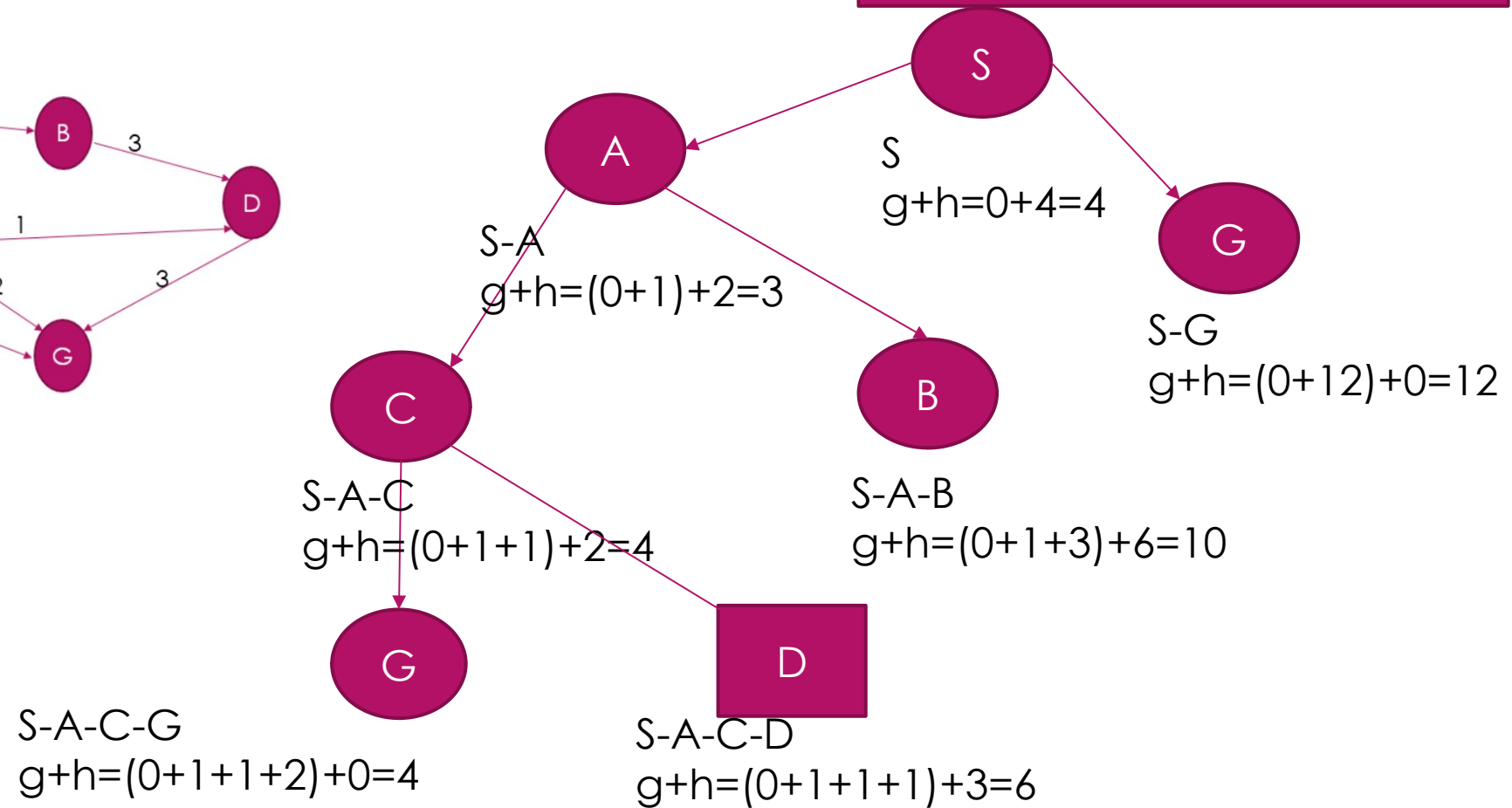
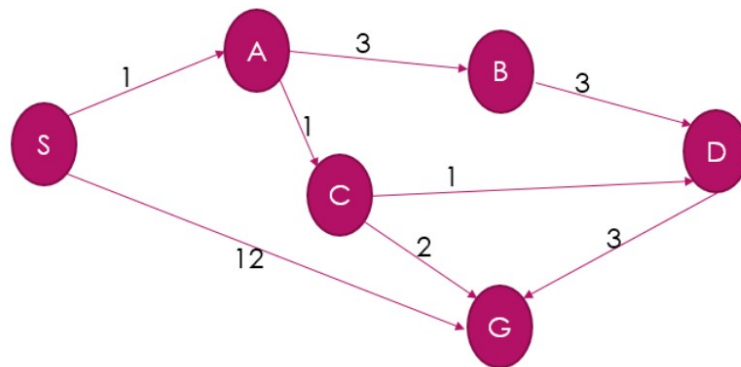
G	D	C	B	A	S	state
0	3	2	6	2	4	H(n)

$$F(n) = g(n) + h(n)$$



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

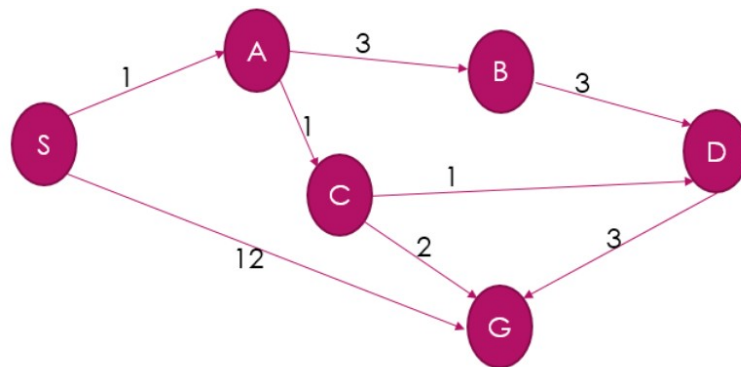
$$F(n) = g(n) + h(n)$$



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

$$F(n) = g(n) + h(n)$$

Final Path : S-A-C-G  
 $g+h=(0+1+1+2)+0=4$



S-A  
 $g+h=(0+1)+2=3$

S-G  
 $g+h=(0+12)+0=12$

S-A-C  
 $g+h=(0+1+1)+2=4$

S-A-B  
 $g+h=(0+1+3)+6=10$

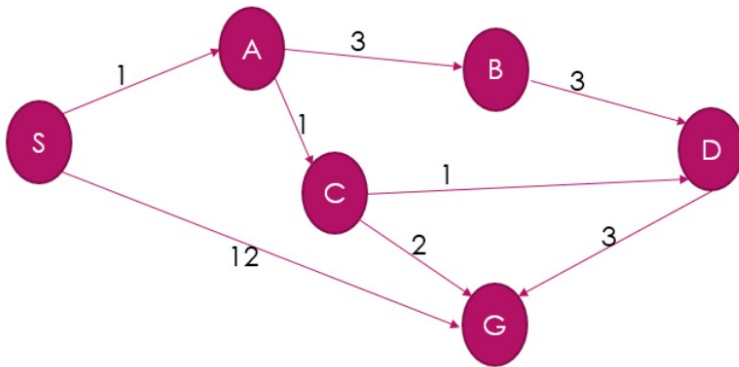
S-A-C-G  
 $g+h=(0+1+1+2)+0=4$

S-A-C-D  
 $g+h=(0+1+1+1)+3=6$

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

$$F(n) = g(n) + h(n)$$

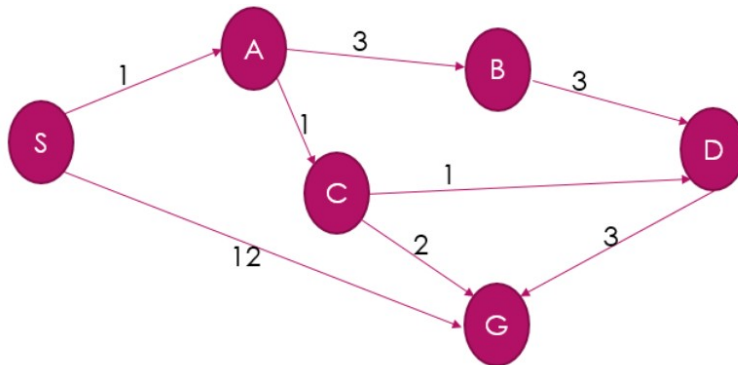
2-By priority queue



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

$$F(n) = g(n) + h(n)$$

$$F(s) = g(s) + h(s) = (0) + 4 = 4$$



Queue = {[s,4]}

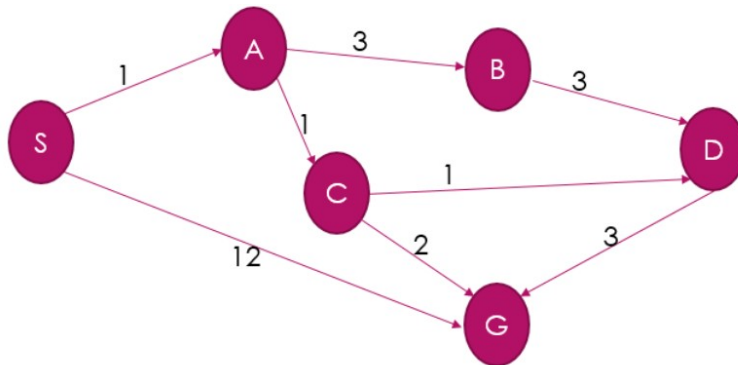
G	D	C	B	A	S	state
0	3	2	6	2	4	H(n)

$$F(n) = g(n) + h(n)$$

$$F(s) = g(s) + h(s) = (0) + 4 = 4$$

$$F(A) = g(A) + h(A) = (1) + 2 = 3$$

$$F(G) = g(G) + h(G) = (12) + 0 = 12$$



Queue = {[s,4]}

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

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

$$F(n) = g(n) + h(n)$$

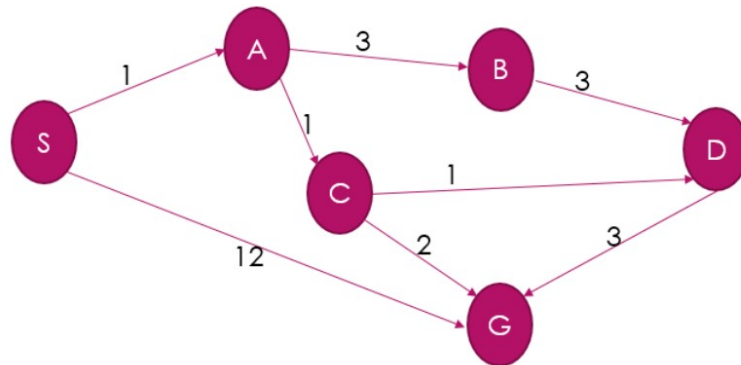
$$F(s) = g(s) + h(s) = (0) + 4 = 4$$

$$F(A) = g(A) + h(A) = (1) + 2 = 3$$

$$F(G) = g(G) + h(G) = (12) + 0 = 12$$

$$F(C) = g(C) + h(C) = (1+1) + 2 = 4$$

$$F(B) = g(B) + h(B) = (1+3) + 6 = 10$$



Queue = {[s,4]}

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

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

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

$$F(n)=g(n)+h(n)$$

$$F(s)=g(s)+h(s)=\\(0)+4=4$$

$$F(A)=g(A)+h(A)=\\(1)+2=3$$

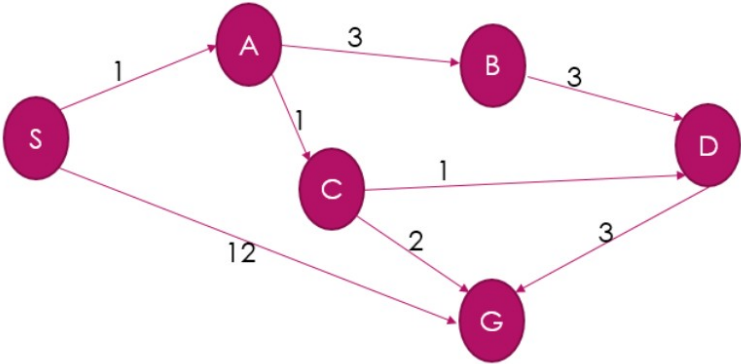
$$F(G)=g(G)+h(G)=\\(12)+0=12$$

$$F(C)=g(C)+h(C)=\\(1+1)+2=4$$

$$F(B)=g(B)+h(B)=\\(1+3)+6=10$$

$$F(D)=g(D)+h(D)=\\(1+1+1)+3=6$$

$$F(G)=g(G)+h(G)=\\(1+1+2)+0=4$$



Queue={[s,4]}

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

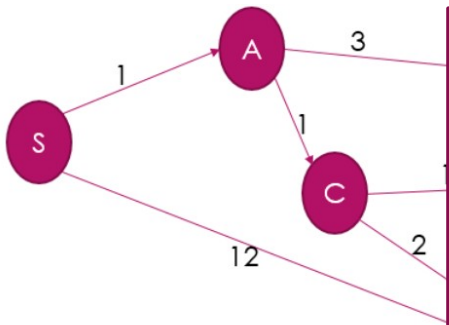
Queue={ [S->A->C,4], [S->A->B,10], [S->G,12]}

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



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

$$F(n)=g(n)+h(n)$$



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

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->A->B,10], , [S->G,12] }

$$F(C)=g(C)+h(C)= (1+1)+2=4$$

$$F(D)=g(D)+h(D)= (1+1+1)+3=6$$

$$F(B)=g(B)+h(B)= (1+3)+6=10$$

$$F(G)=g(G)+h(G)= (1+1+2)+0=4$$

$$F(G)=g(G)+h(G)= (12)+0=12$$

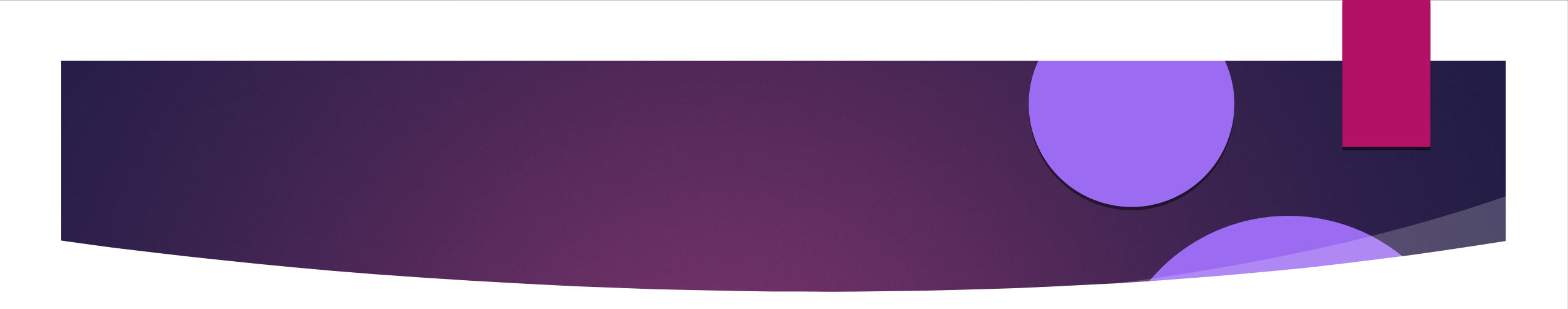
# Implementations

# Implementation

- The A\* algorithm is fairly simple. There are two sets, OPEN and CLOSED.
- The OPEN set contains those nodes that are candidates for examining.
- The CLOSED set contains those nodes that have already been examined.

## Pseudocode

```
make an open list containing only the starting node
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:
```



A\* will behave like depth-first search among equal cost paths  
(avoiding exploring more than one equally optimal solution).  
A\* reads from a priority queue, which has unvisited nodes.

## Special cases

Dijkstra's algorithm, as another example of a uniform-cost search algorithm, can be viewed as a special case of  $A^*$

where  $\{ \displaystyle h(x)=0 \} h(x) = 0$  for all  $x$ . [11][12] General

depth-first search can be implemented using  $A^*$  by considering that there is a global counter  $C$  initialized with a very large value. Every time we process a node we assign  $C$  to all

of its newly discovered neighbors. After each single assignment, we decrease the counter  $C$  by one.

Thus the earlier a node is discovered, the higher its  $\{ \displaystyle h(x) \} h(x)$  value.

Both Dijkstra's algorithm and depth-first search can be implemented more efficiently

without including an  $\{ \displaystyle h(x) \} h(x)$  value at each node.



→ Worst-case space complexity :  $O(|E|)=O(b^d)$

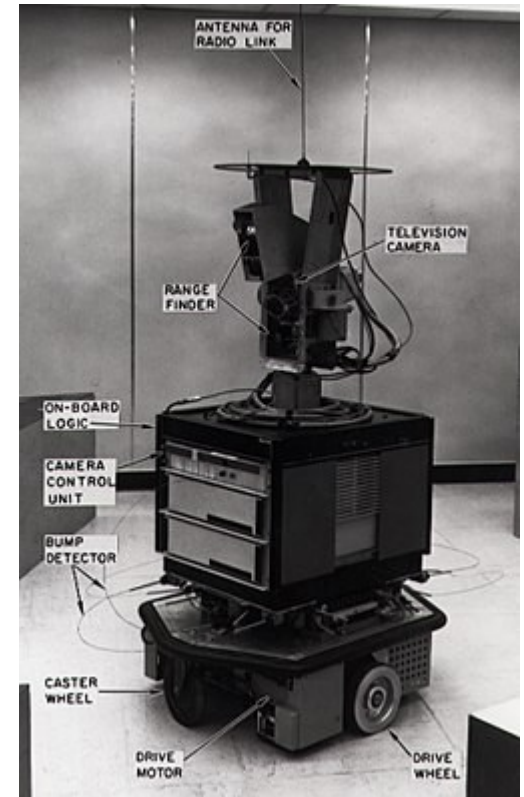
→ Worst-case performance :  $O(|E|)=O(b^d)$

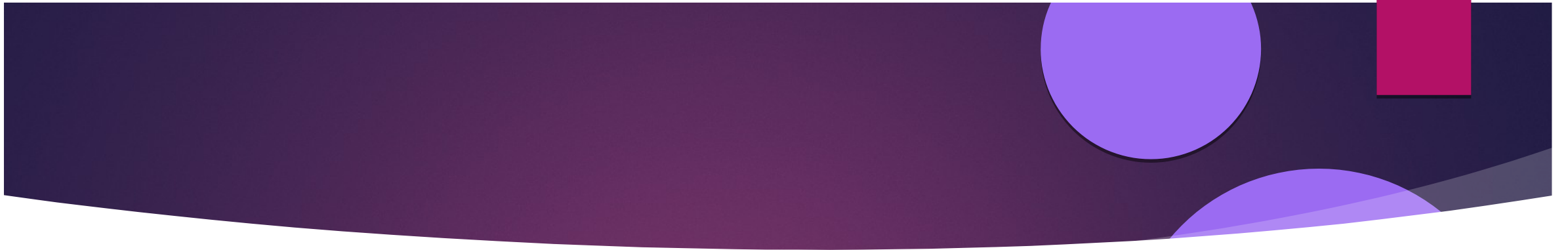
# Applications



## The Shakey robot :

The First AI Application based on A\* Search which had the aim of building a mobile robot that could plan its own actions.





These actions involved traveling from one location to another, turning the light switches on and off, opening and closing the doors, climbing up and down from rigid objects, and pushing movable objects around.

An operator types the command "push the block off the platform" at a computer console.

Shakey looks around, identifies a platform with a block on it, and locates a ramp in order to reach the platform.

Shakey then pushes the ramp over to the platform, rolls up the ramp onto the platform, and pushes the block off the platform.

Mission accomplished.

## Manhattan heuristic monotonic :

We want to go from yellow square (initial state) to green square (goal state) for each of them. We then select the neighbor with the lowest  $ff$  cost.

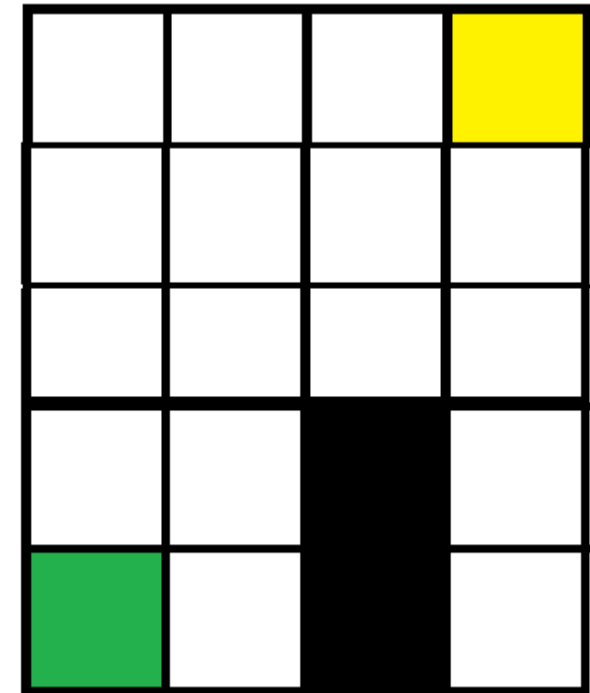
This is our new current cell and we then repeat the process above. (populate neighbors and compute  $ff$ ,  $gg$  and  $hh$  and choose the lowest ).

We do this until we are at the goal cell. The image below demonstrates how the search proceeds.

In each cell the respective  $ff$ ,  $hh$  and  $gg$  values are shown.

Remember  $gg$  is the cost that has been accrued in reaching the cell and

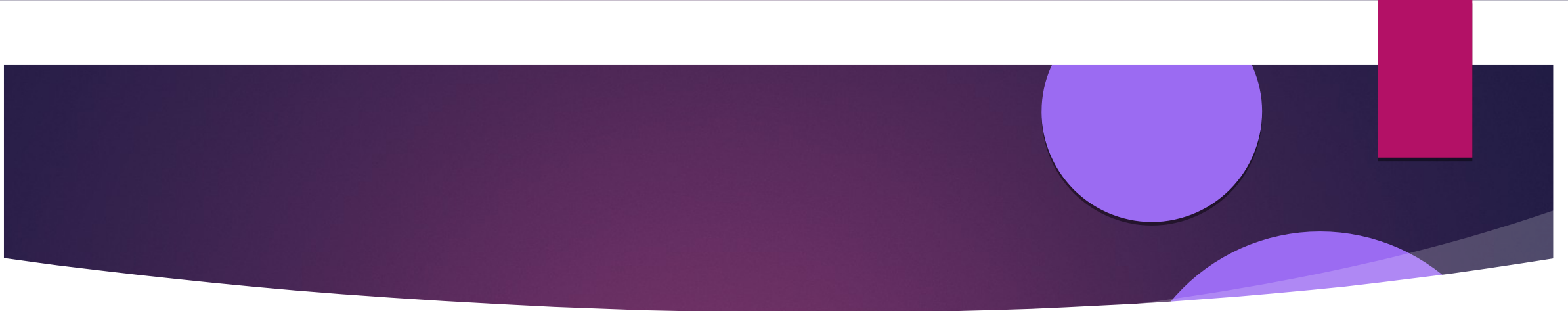
$hh$  is the Manhattan distance towards the yellow cell while  $ff$  is the sum of  $hh$  and  $gg$  .



# Warcraft

Strategy video game where the goal is to defend a player's territories or possessions by obstructing enemy attackers, usually achieved by placing defensive structures on or along their path of attack.





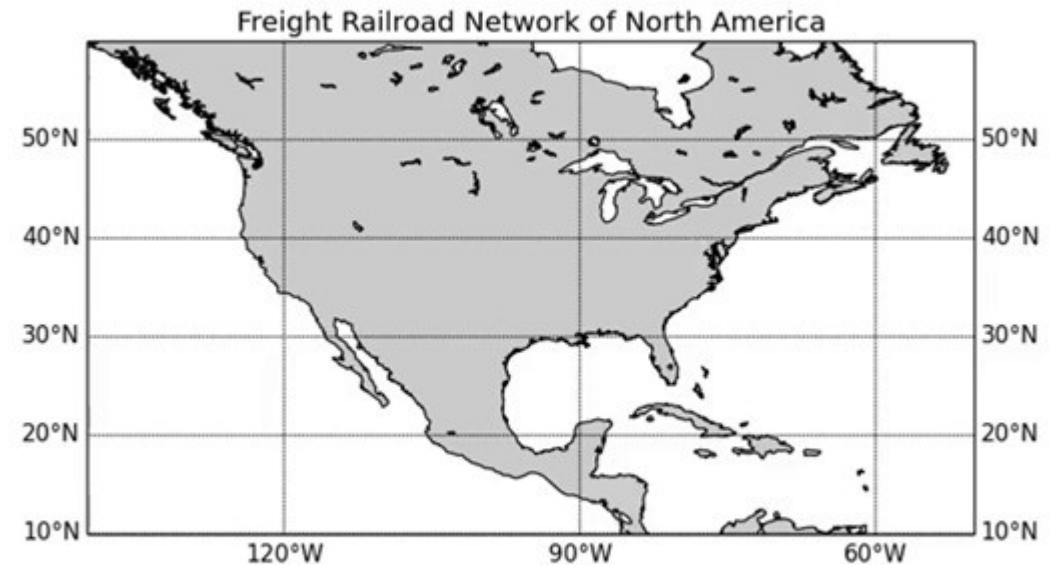
In the game, as in many real-time strategy (RTS) games, players collect resources, train individual units and heroes, and build bases in order to: achieve various goals (in single-player mode), or to defeat the enemy player. Four playable factions can be chosen from: Humans, Orcs, (both of which appeared in the previous games) and two new factions: the Night Elves and the Undead. Warcraft III's single-player campaign is laid out similarly to that of StarCraft, and is told through the races in a progressive manner. Players can also play matches against the computer, or against others—using local area networking (LAN) or Blizzard's Battle.net gaming platform.

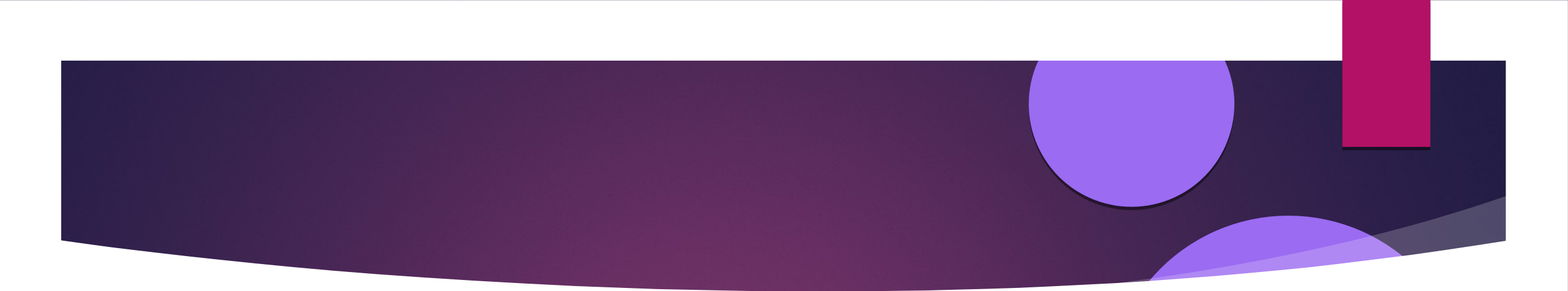


# Web-based maps :

It uses this algorithm to find the shortest path very efficiently :

[https://upload.wikimedia.org/wikipedia/commons/6/60/A%2A\\_Search\\_Example\\_on\\_North\\_American\\_Freight\\_Train\\_Network.gif](https://upload.wikimedia.org/wikipedia/commons/6/60/A%2A_Search_Example_on_North_American_Freight_Train_Network.gif)





A\* Search limits space complexity, as it stores all generated nodes in memory. Thus, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance .



## Advantages & Disadvantages



## Advantages :

- It is complete and optimal.
- It is the best one from other techniques. It is used to solve very complex problems.
- It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than  $A^*$ .

# Termination and Completeness

On finite graphs with non-negative edge weights  $A^*$  is guaranteed to terminate and is complete, i.e. it will always find a solution (a path from start to goal) if one exists.

On infinite graphs with a finite branching factor and edge costs that are bounded away from zero .

$A^*$  is guaranteed to terminate only if there exists a solution.

# Optimal Efficiency

The most interesting positive result they proved is that  $A^*$ , with a consistent heuristic, is optimally efficient with respect to all admissible  $A^*$ -like search algorithms on all "non-pathological" search problems. Roughly speaking, their notion of non-pathological problem is what we now mean by "up to tie-breaking".

This result does not hold if  $A^*$ 's heuristic is admissible but not consistent. In that case, Dechter and Pearl showed there exist admissible  $A^*$ -like algorithms that can expand arbitrarily fewer nodes than  $A^*$  on some non-pathological problems.

## Disadvantages :

- This algorithm is complete if the branching factor is finite and every action has fixed cost.
- The speed execution of A\* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute  $h(n)$ .
- It has complexity problems.

# Complexity

The time complexity of  $A^*$  depends on the heuristic. In the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path)  $d$ :  $O(b^d)$ , where  $b$  is the branching factor (the average number of successors per state).

## Relations to other algorithms:

### A\* Algorithm

- $f(n) = g(n) + h(n)$ .
- Each step expand the node with lowest value of  $f(n)$ .
- No other optimal algorithm is guaranteed to expand fewer nodes than A\*.

### Dijkstra's Algorithm

- $f(n) = g(n)$ , as a special case of A\* where  $h(n) = 0$ .
- Each step expand all closest unexamined nodes.
- Can be implemented more efficiently without a  $h(x)$  value at each node.

## Reference :

<https://www.redblobgames.com/pathfinding/a-star/implementation.html>

<https://www.redblobgames.com/pathfinding/a-star/implementation.html>

<http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html>

<https://www.redblobgames.com/pathfinding/a-star/implementation.html>

<https://www.geeksforgeeks.org/a-search-algorithm/>

<http://cs.indstate.edu>