A* Algorithm

Thomas Kappas
Applied Mathematical Sciences, NTUA

Athens, 4/12/2014

A* Algorithm

- A* is a computer algorithm that is widely used in pathfinding and graph traversal

A* Algorithm

- A* is a computer algorithm that is widely used in pathfinding and graph traversal
- Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described the algorithm in 1968

A* Algorithm

- A* is a computer algorithm that is widely used in pathfinding and graph traversal
- Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described the algorithm in 1968
- It is an extension of Edsger Dijkstra's 1959 algorithm

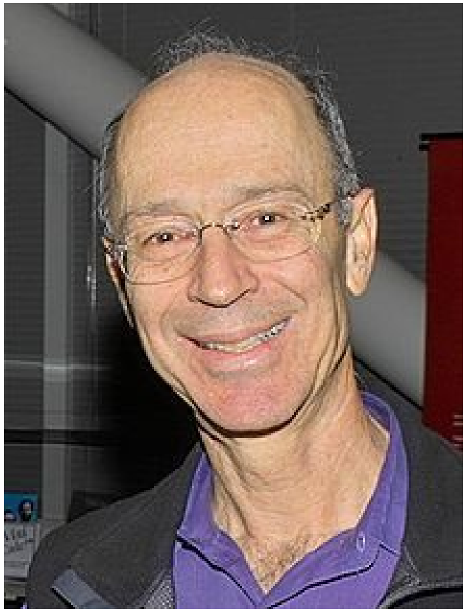
A* Algorithm

- A* is a computer algorithm that is widely used in pathfinding and graph traversal
- Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described the algorithm in 1968
- It is an extension of Edsger Dijkstra's 1959 algorithm
- A* uses a best-first search and finds a least-cost path from a given initial node to one goal node

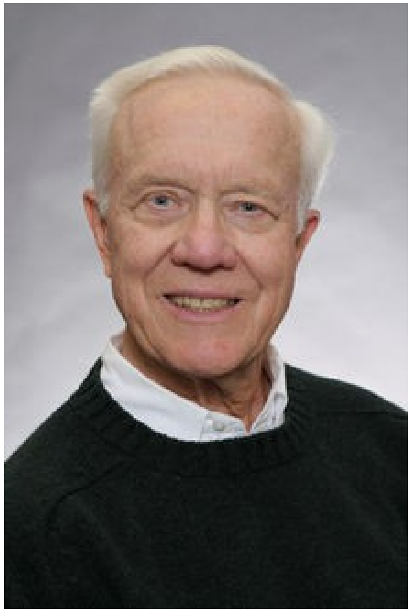
A* Algorithm

- A* is a computer algorithm that is widely used in pathfinding and graph traversal
- Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described the algorithm in 1968
- It is an extension of Edsger Dijkstra's 1959 algorithm
- A* uses a best-first search and finds a least-cost path from a given initial node to one goal node
- A* achieves better time performance by using heuristics

A* Algorithm



Peter Hart



Nils Nilsson

Bertram Raphael

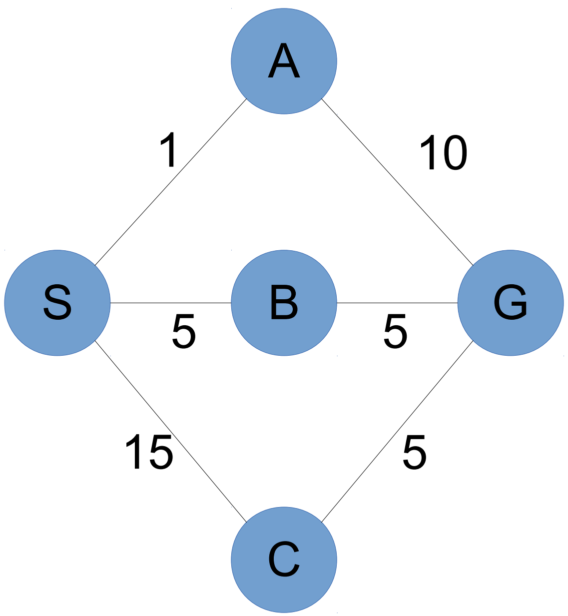
Uniform Cost Search (UCS)

Uniform Cost Search (UCS)

The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.

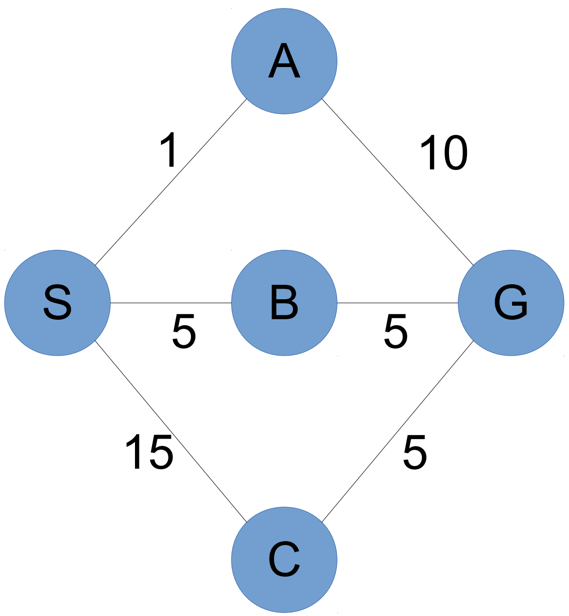
Uniform Cost Search (UCS)

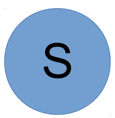
The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.



Uniform Cost Search (UCS)

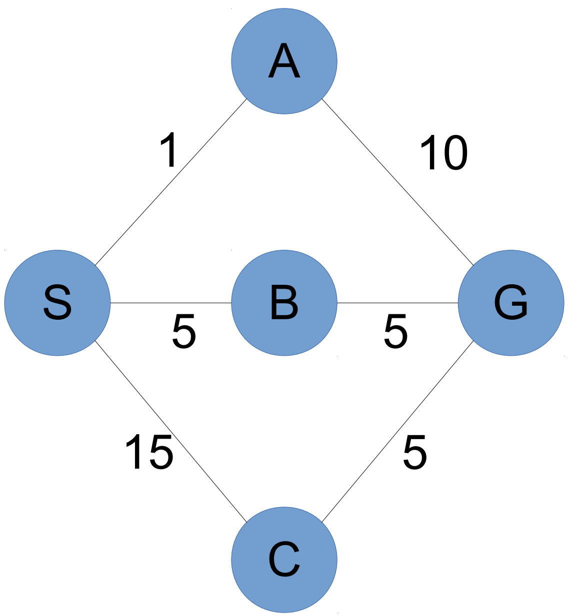
The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.

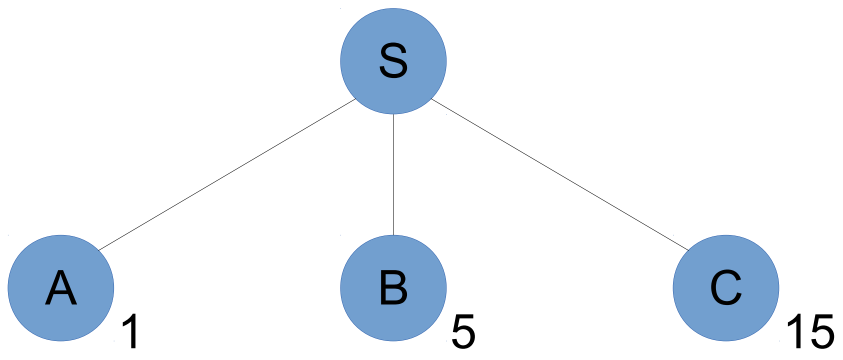




Uniform Cost Search (UCS)

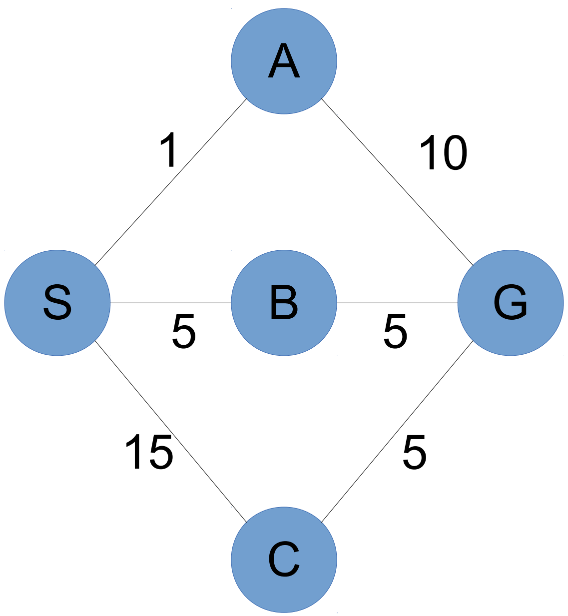
The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.

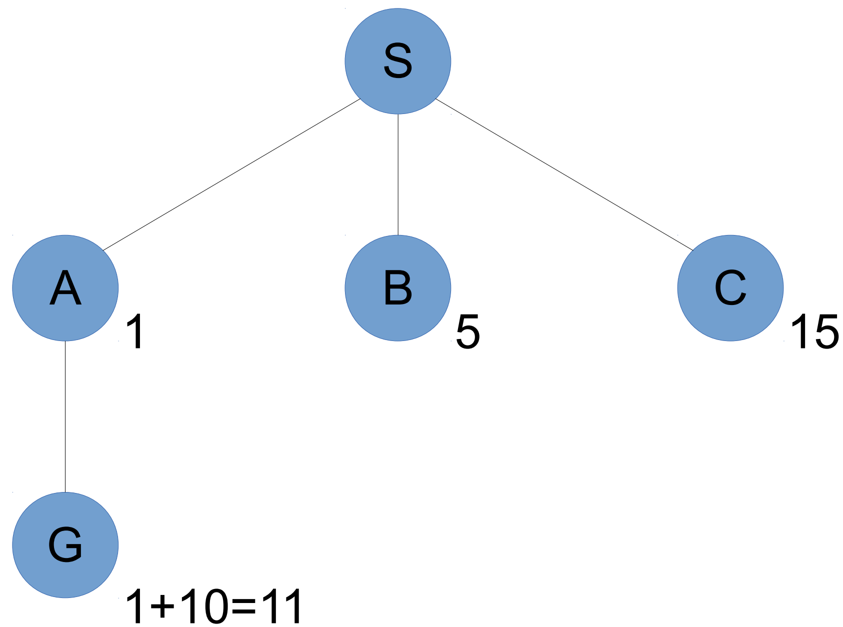




Uniform Cost Search (UCS)

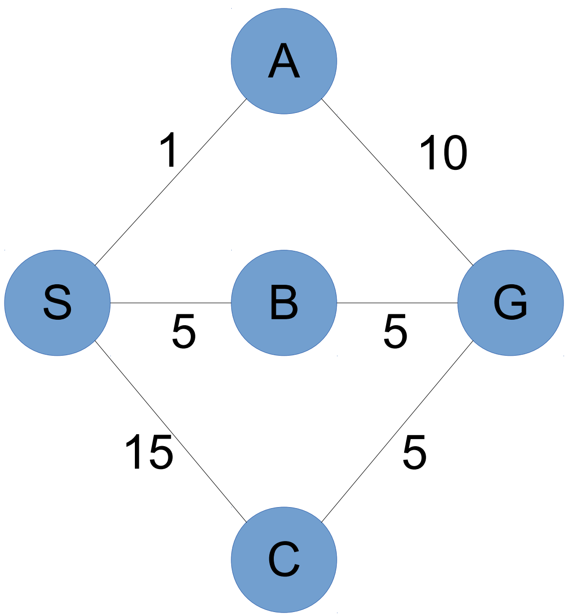
The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.

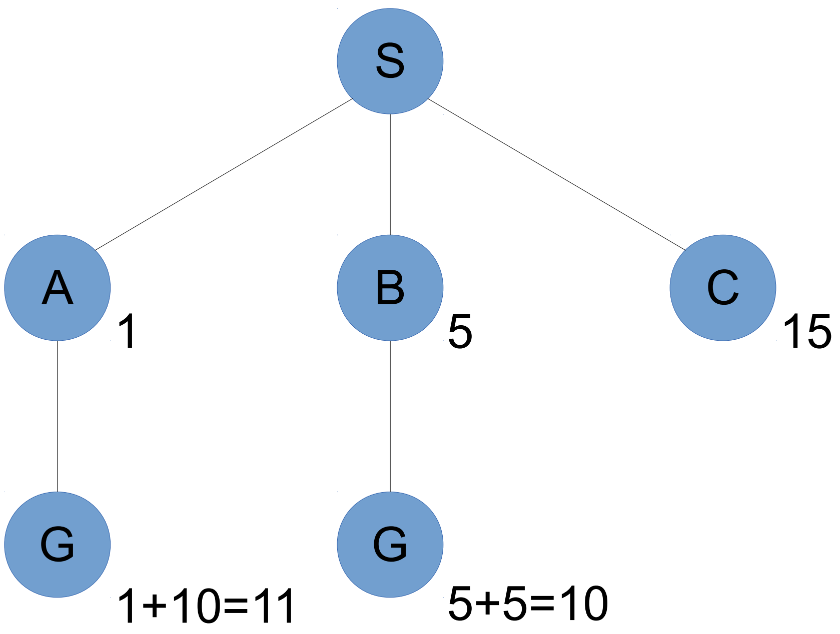




Uniform Cost Search (UCS)

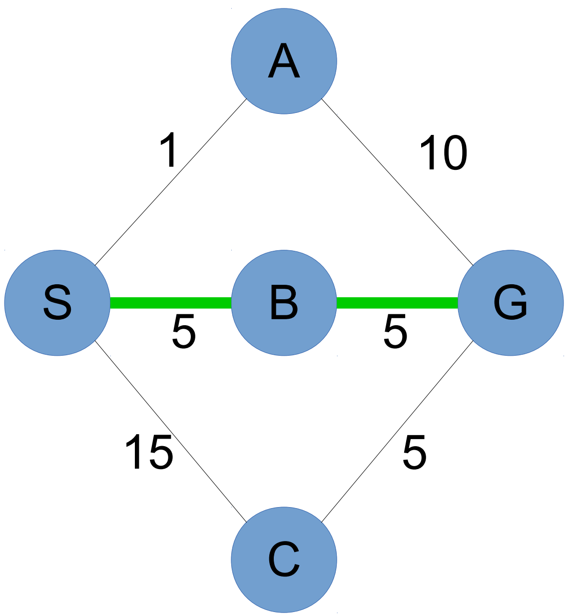
The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.

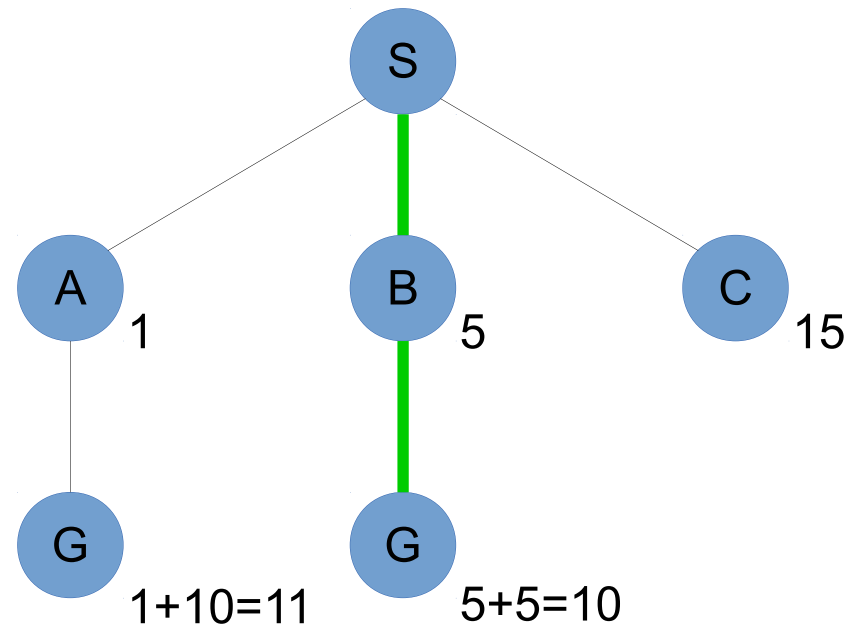




Uniform Cost Search (UCS)

The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.





Uniform Cost Search (UCS)

- Complete?

Uniform Cost Search (UCS)

- Complete? Yes*

Uniform Cost Search (UCS)

- Complete? Yes*
- Optimal?

Uniform Cost Search (UCS)

- Complete? Yes*
- Optimal? Yes*

Uniform Cost Search (UCS)

- Complete? Yes*
- Optimal? Yes*

*Only if branching factor is finite and the cost of each step exceeds some positive bound ε

Informed Search

Informed Search

- Use problem specific knowledge to pick which node to expand

Informed Search

- Use problem specific knowledge to pick which node to expand
- Involves a **heuristic function *h(n)*** estimating the cheapest path from *n* to a goal state

Informed Search

- Use problem specific knowledge to pick which node to expand
- Involves a **heuristic function *h(n)*** estimating the cheapest path from *n* to a goal state

Requirements

$h(n) \geq 0 \; \forall \; n$

$h(goal) = 0$

Greedy Best First Search

Greedy Best First Search

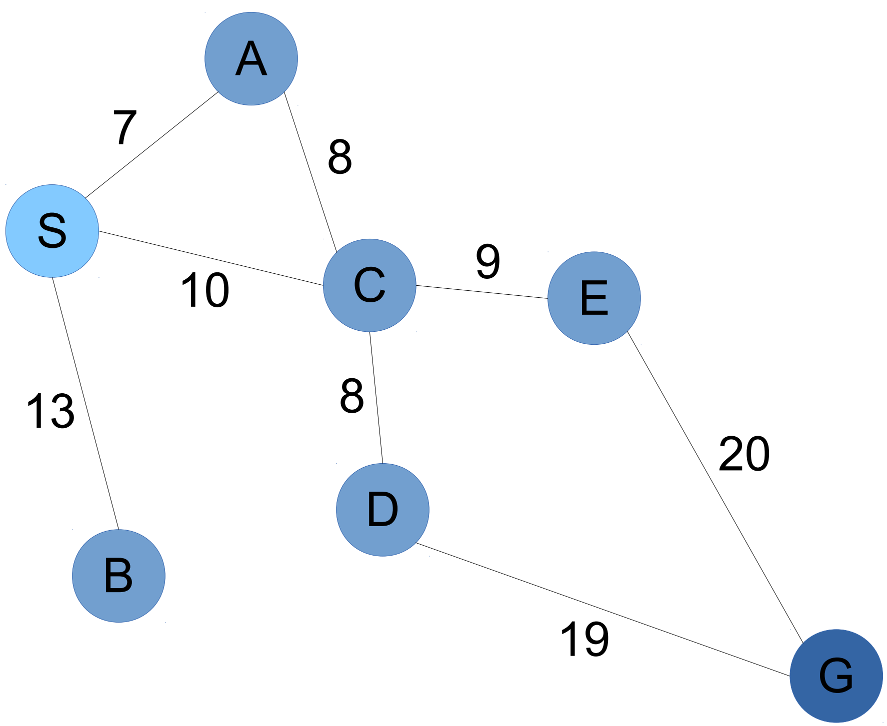
Evaluation function

f(n) = *h(n)* : Expand node that appears to be closest to the goal

Greedy Best First Search

Evaluation function

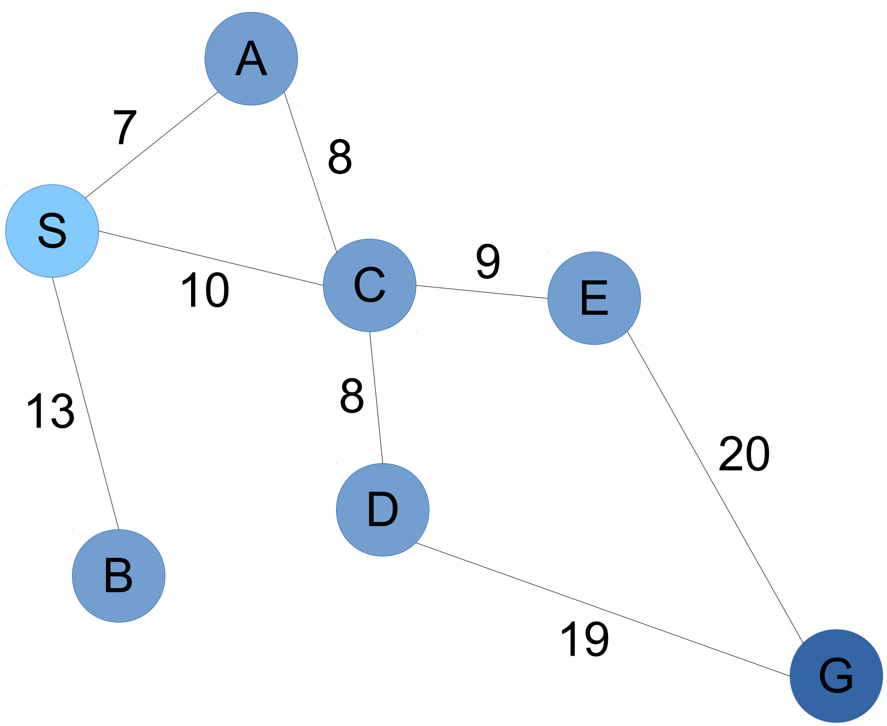
f(n) = h(n) : Expand node that appears to be closest to the goal



Greedy Best First Search

Evaluation function

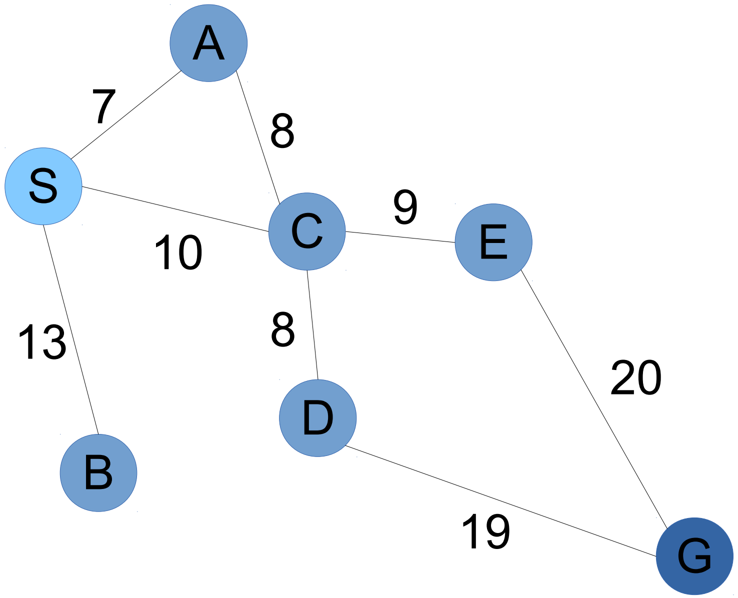
f(n) = *h(n)* : Expand node that appears to be closest to the goal



f(n) = *h(n)* = straight line distance from n to node G

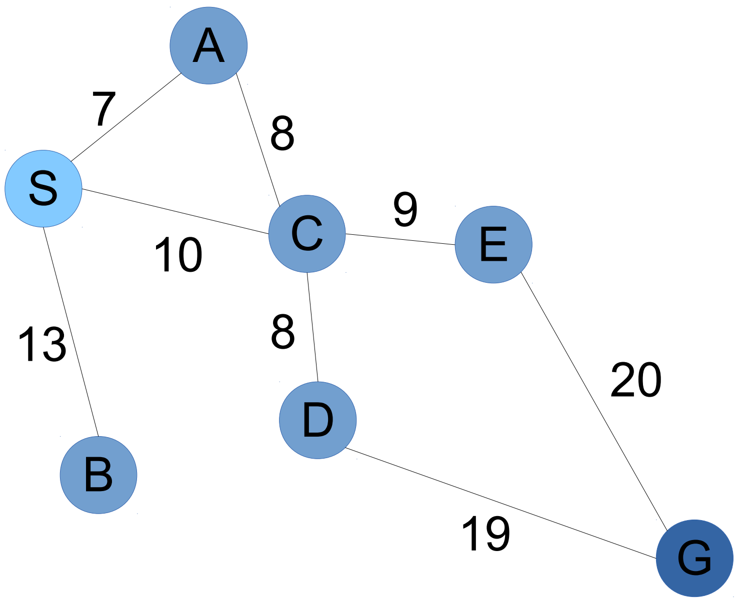
Node	*h(n)*
A	34
S	36
B	32
C	25
D	19
E	18
G	0

Greedy Best First Search

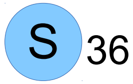


Node	*h(n)*
A	34
S	36
B	32
C	25
D	19
E	18
G	0

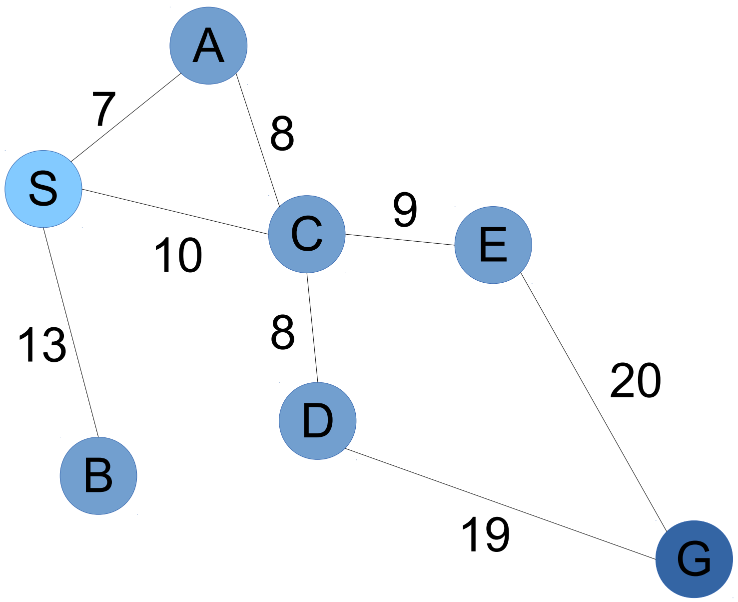
Greedy Best First Search

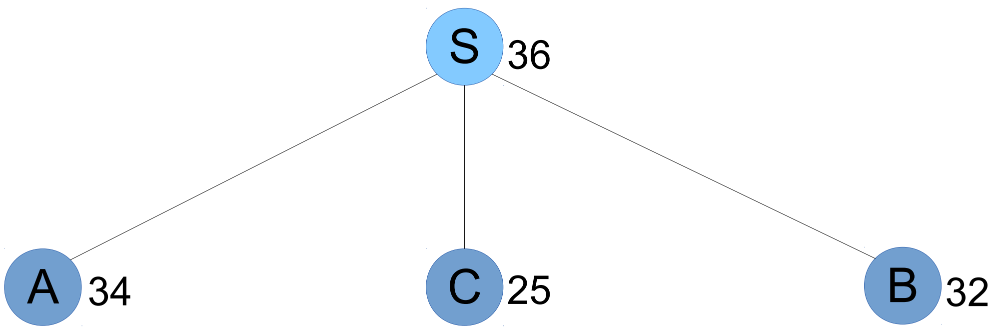


Node	*h(n)*
A	34
S	36
B	32
C	25
D	19
E	18
G	0



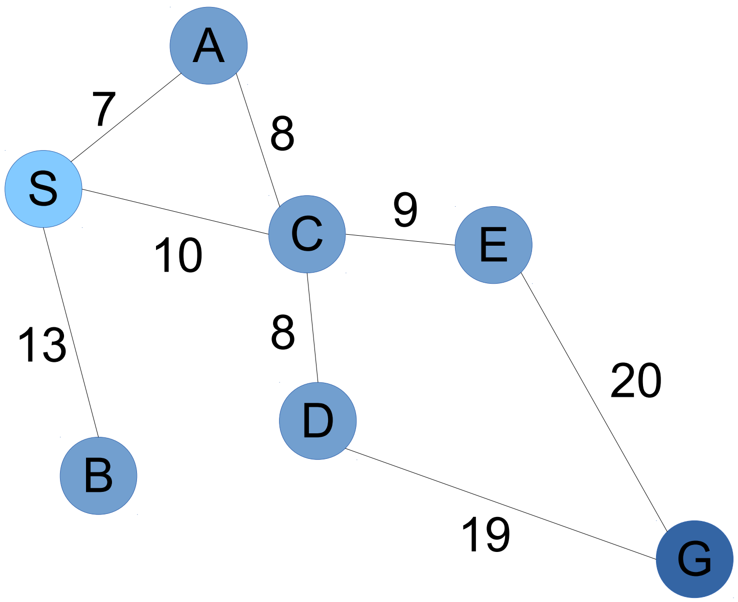
Greedy Best First Search

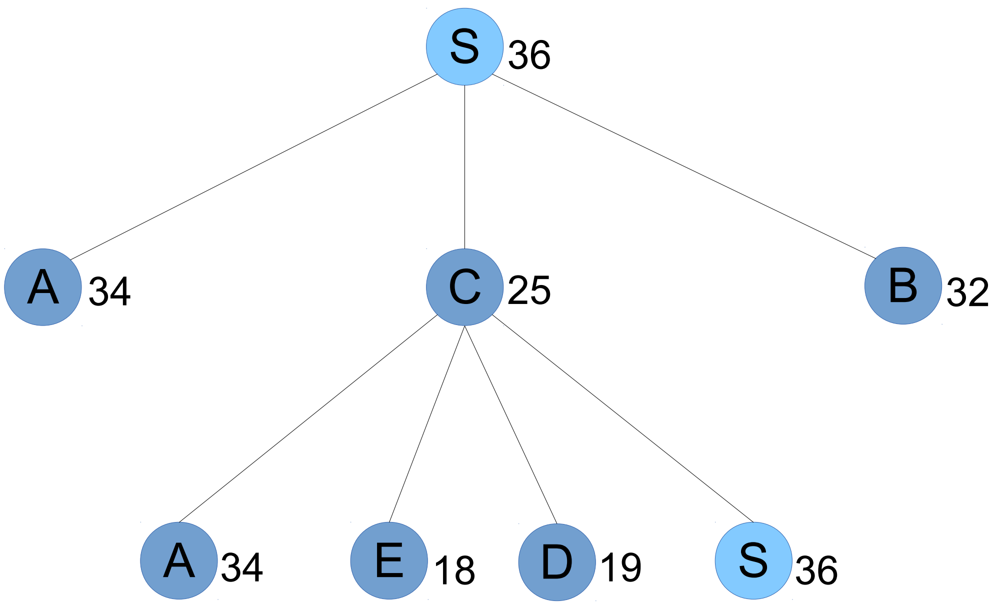




Node	*h(n)*
A	34
S	36
B	32
C	25
D	19
E	18
G	0

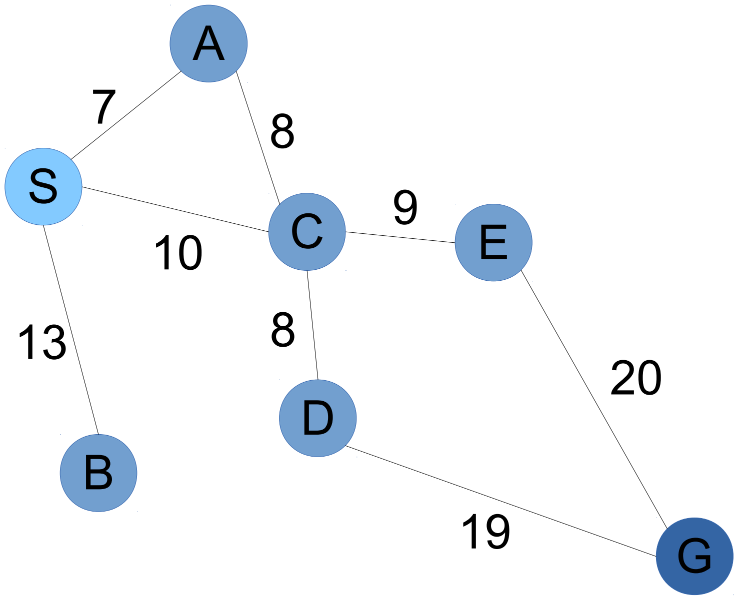
Greedy Best First Search

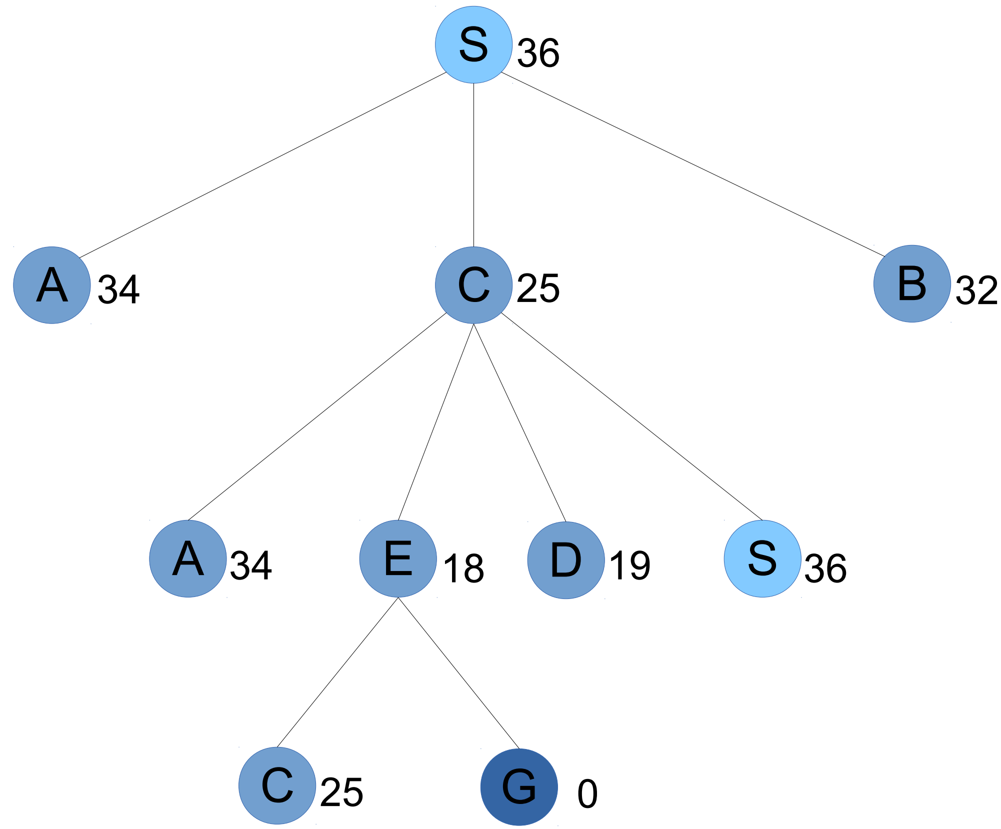




Node	*h(n)*
A	34
S	36
B	32
C	25
D	19
E	18
G	0

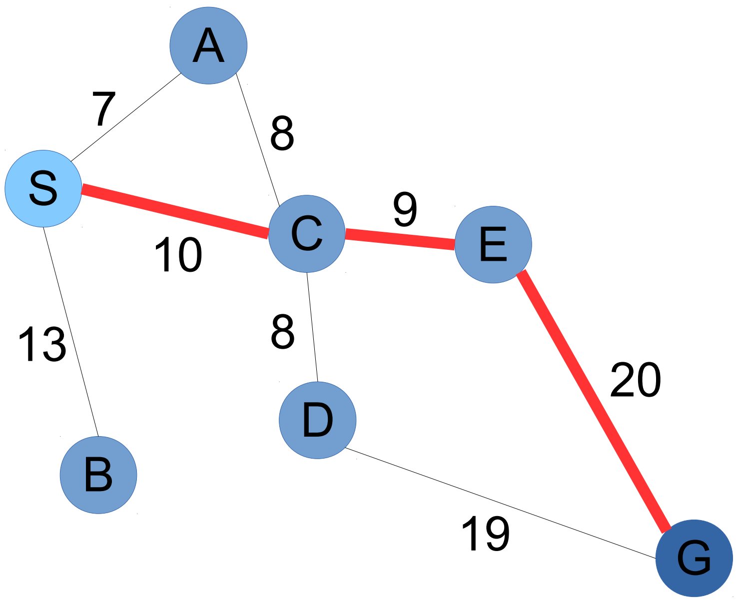
Greedy Best First Search



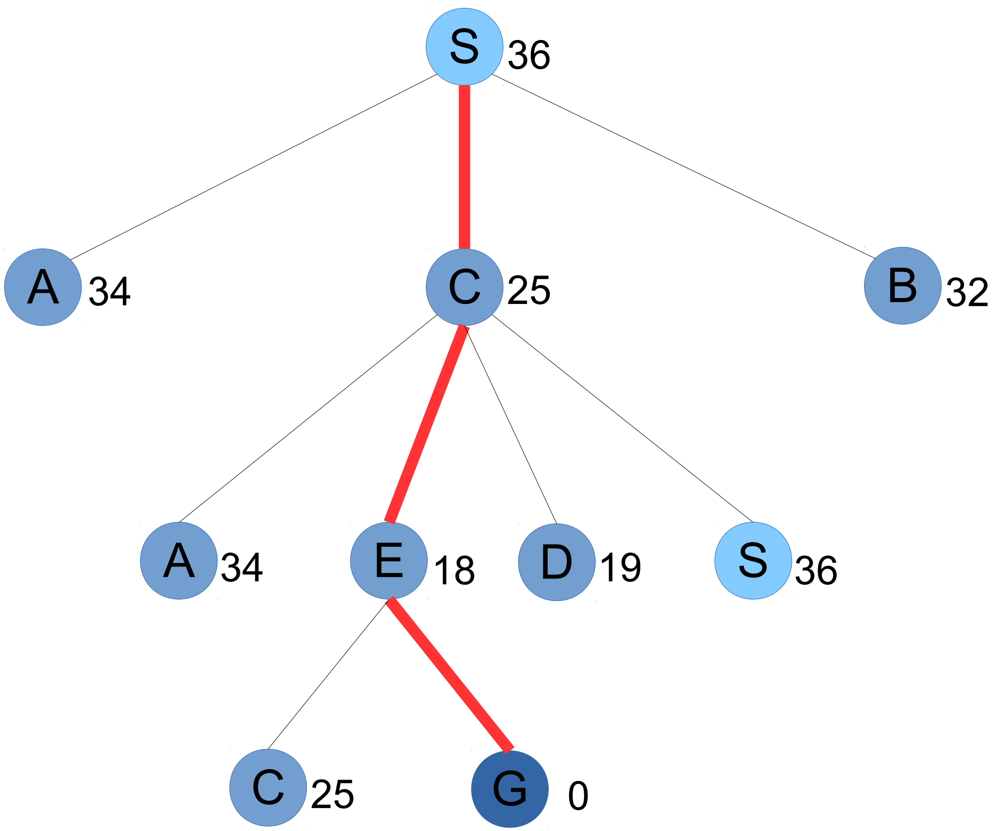


Node	*h(n)*
A	34
S	36
B	32
C	25
D	19
E	18
G	0

Greedy Best First Search



Node	*h(n)*
A	34
S	36
B	32
C	25
D	19
E	18
G	0



Greedy Best First Search

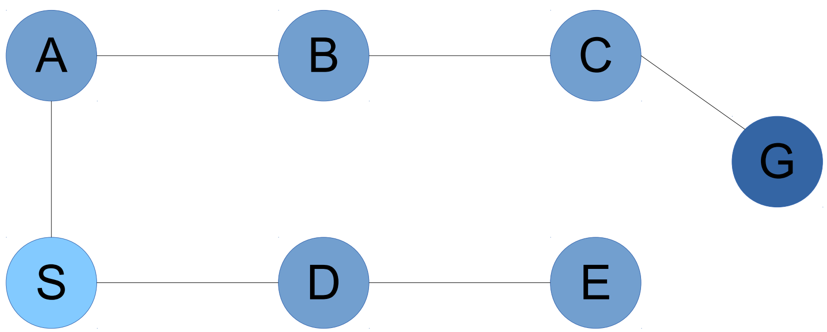
Complete?

Greedy Best First Search

Complete? No*

Greedy Best First Search

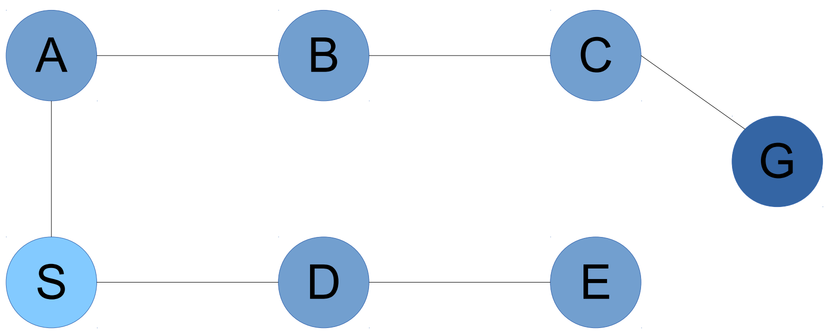
Complete? No*



Node	*h(n)*
S	6
A	7
B	4
C	2
D	4
E	2
G	0

Greedy Best First Search

Complete? No*



Node	*h(n)*
S	6
A	7
B	4
C	2
D	4
E	2
G	0

*Yes if we use Graph Search and branching factor is finite

Greedy Best First Search

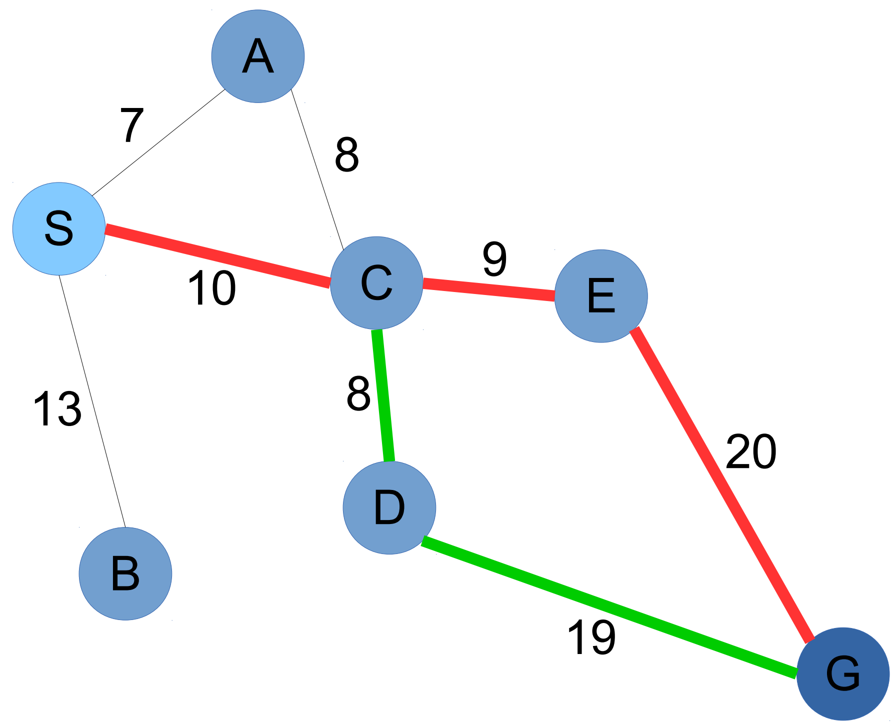
Optimal?

Greedy Best First Search

Optimal? No

Greedy Best First Search

Optimal? No



A* Search

A* Search

- Idea: Include cost of reaching node

A* Search

- Idea: Include cost of reaching node
- $g(n)$ = cost of reaching n

A* Search

- Idea: Include cost of reaching node
- $g(n)$ = cost of reaching n
- $h(n)$ = estimated cost of reaching goal from n

A* Search

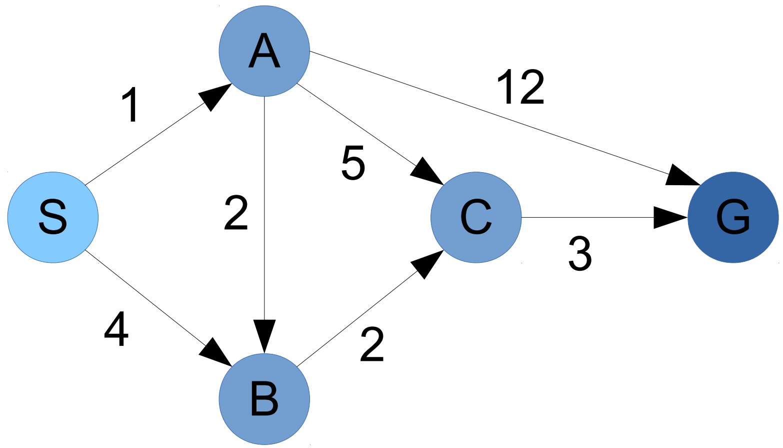
- Idea: Include cost of reaching node
- $g(n)$ = cost of reaching n
- $h(n)$ = estimated cost of reaching goal from n
- Evaluation function $f(n) = g(n) + h(n)$

A* Search

- Idea: Include cost of reaching node
- $g(n)$ = cost of reaching n
- $h(n)$ = estimated cost of reaching goal from n
- Evaluation function $f(n) = g(n) + h(n)$

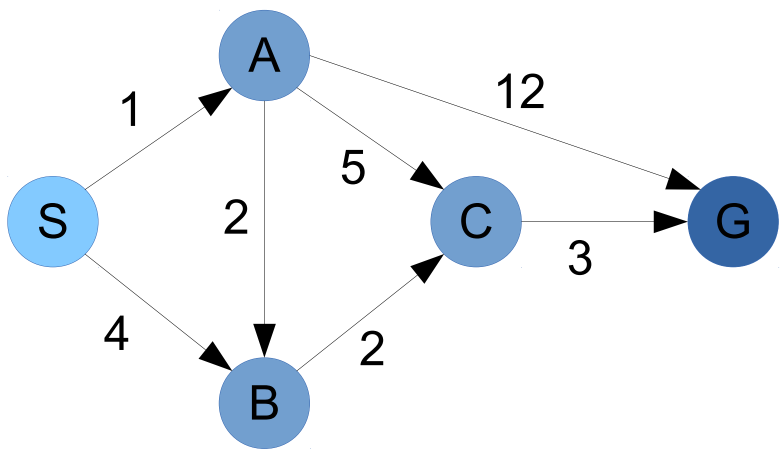
Uniform Cost Search had $f(n) = g(n)$
Greedy Best First Search had $f(n) = h(n)$

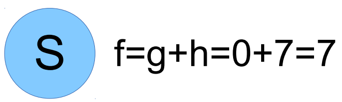
A* Tree Search



Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

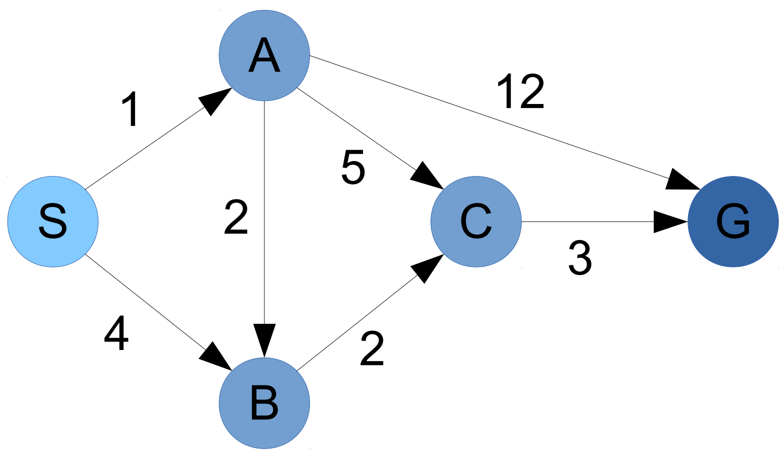
A* Tree Search

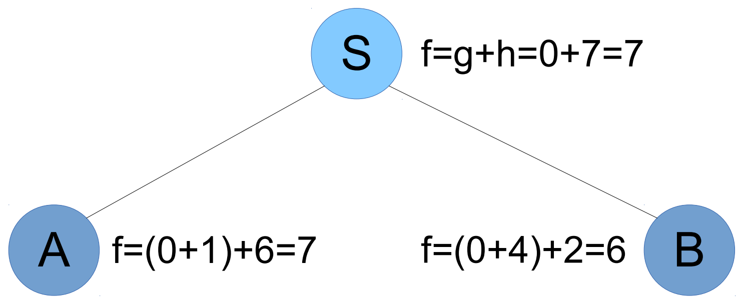




Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

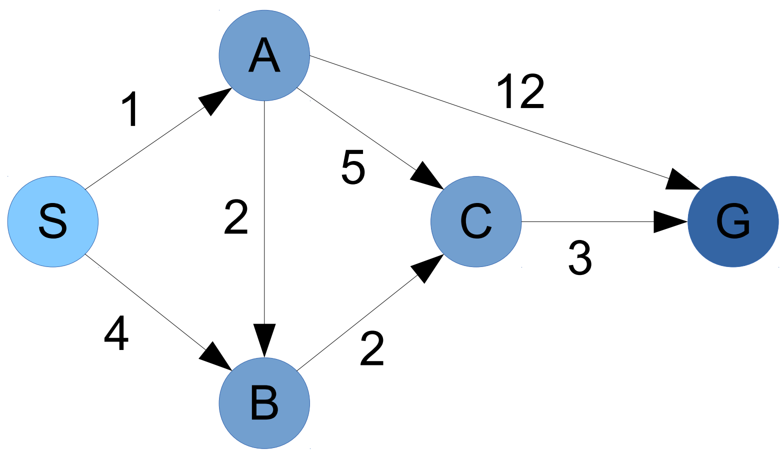
A* Tree Search

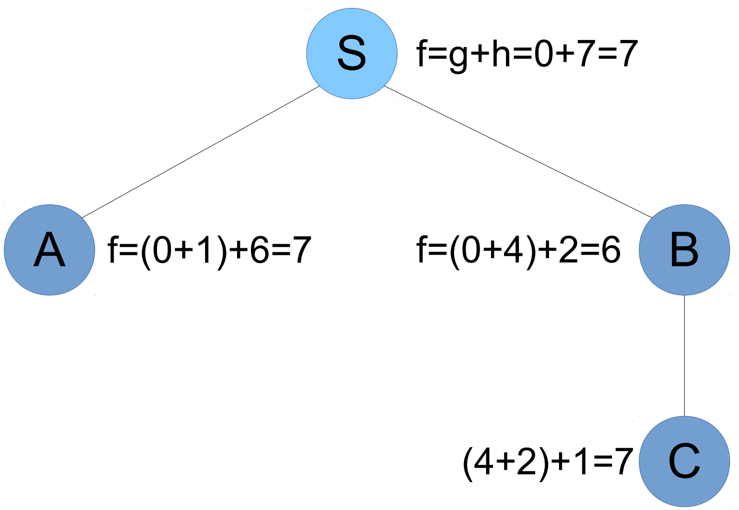




Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

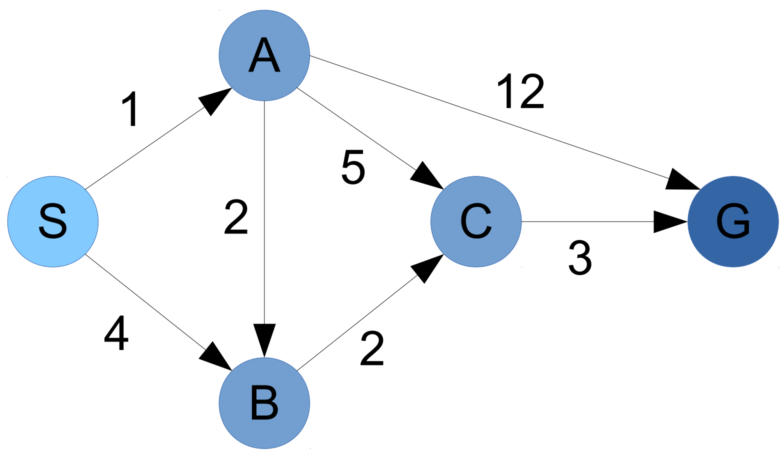
A* Tree Search

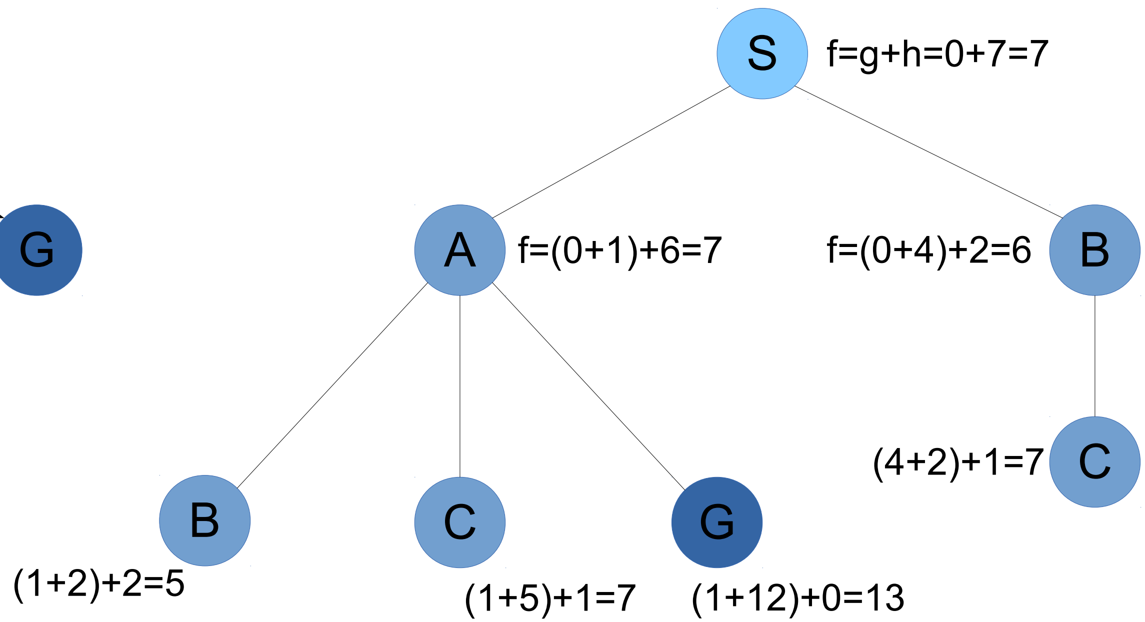




Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

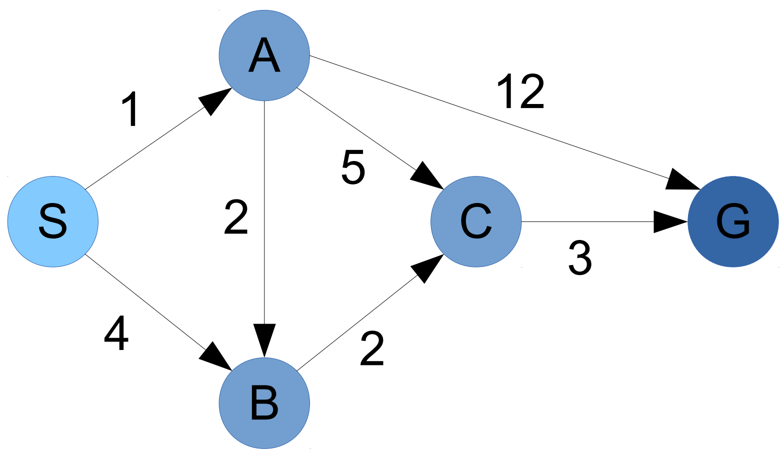
A* Tree Search

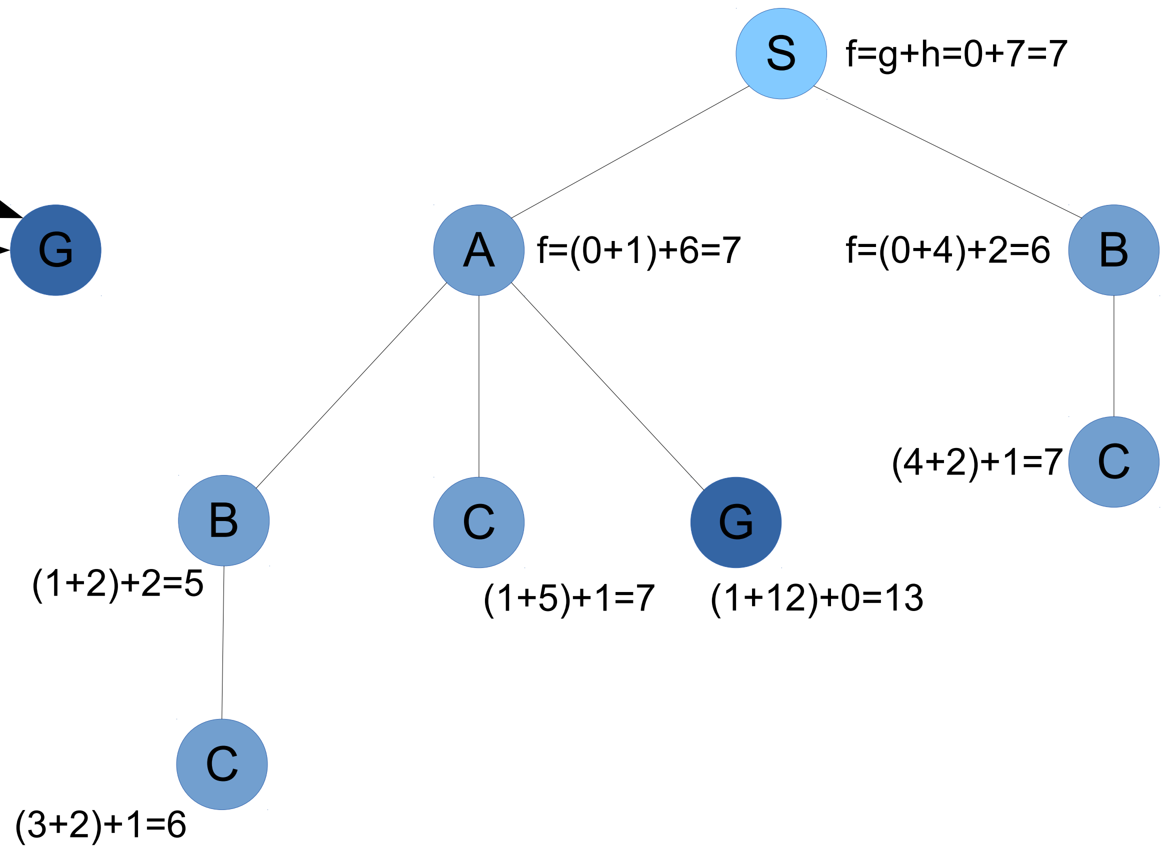




Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

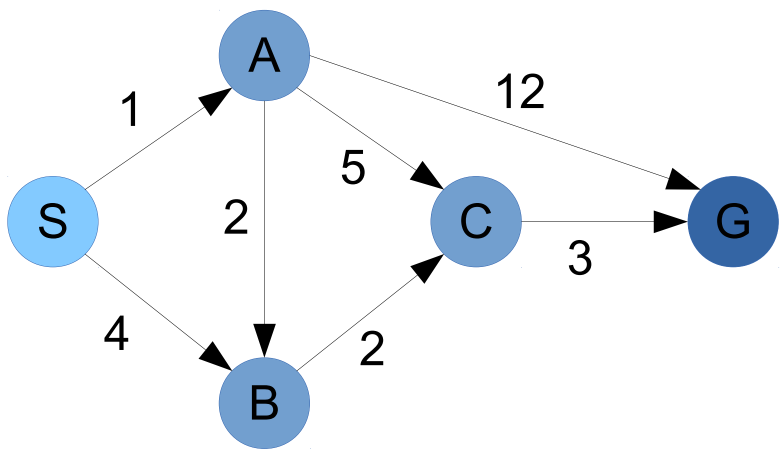
A* Tree Search



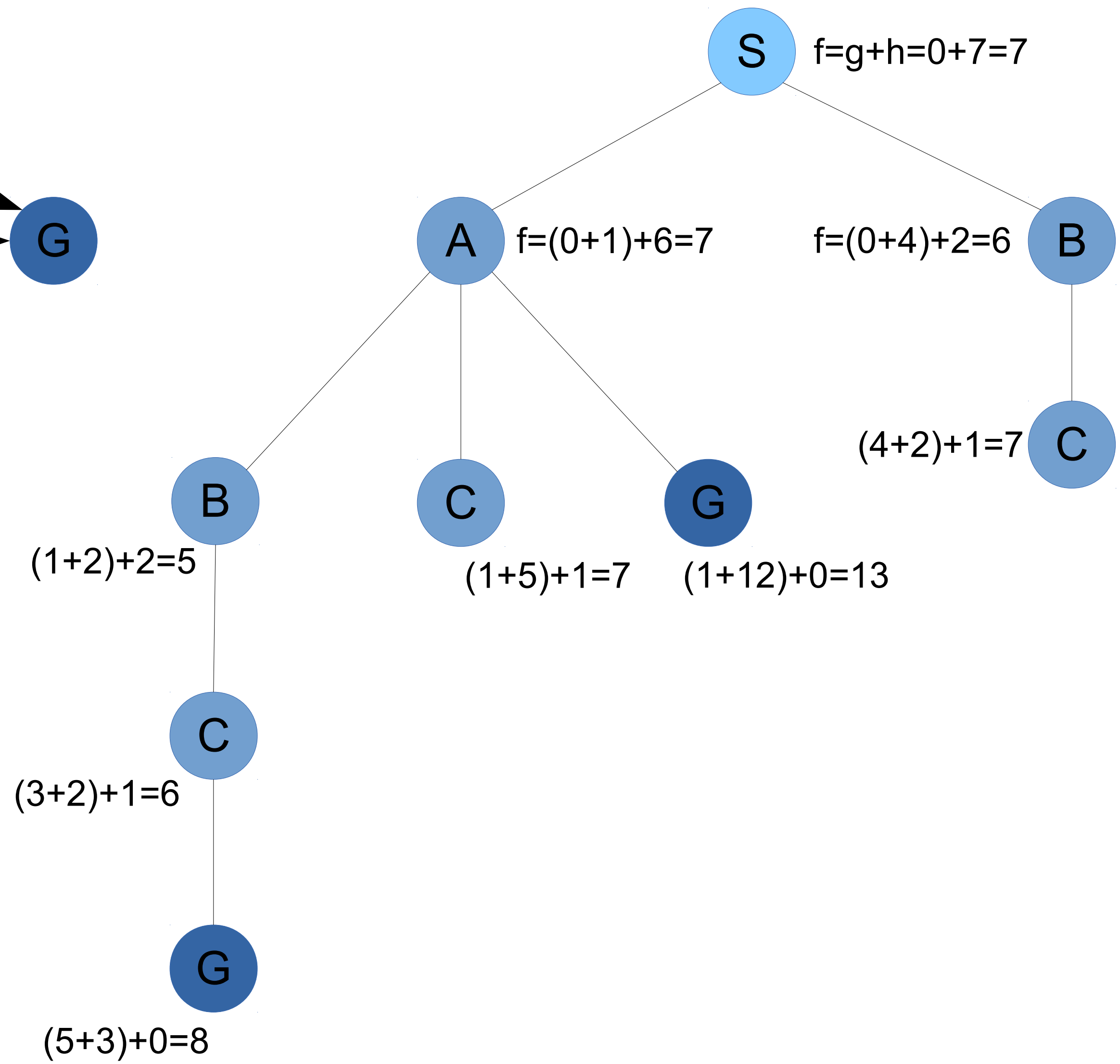


Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

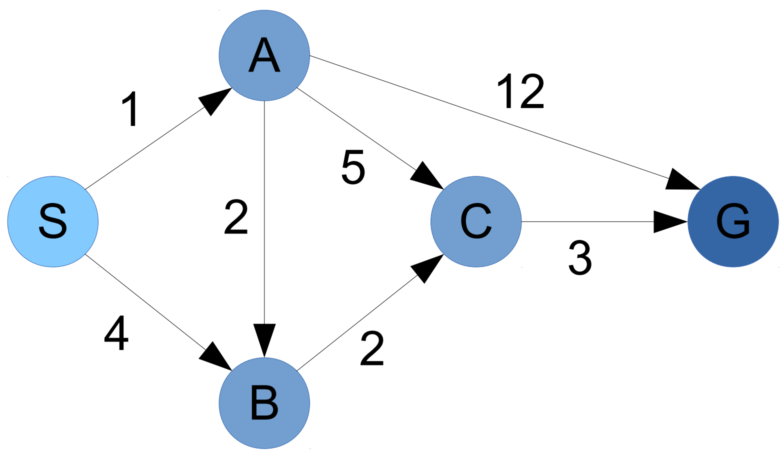
A* Tree Search



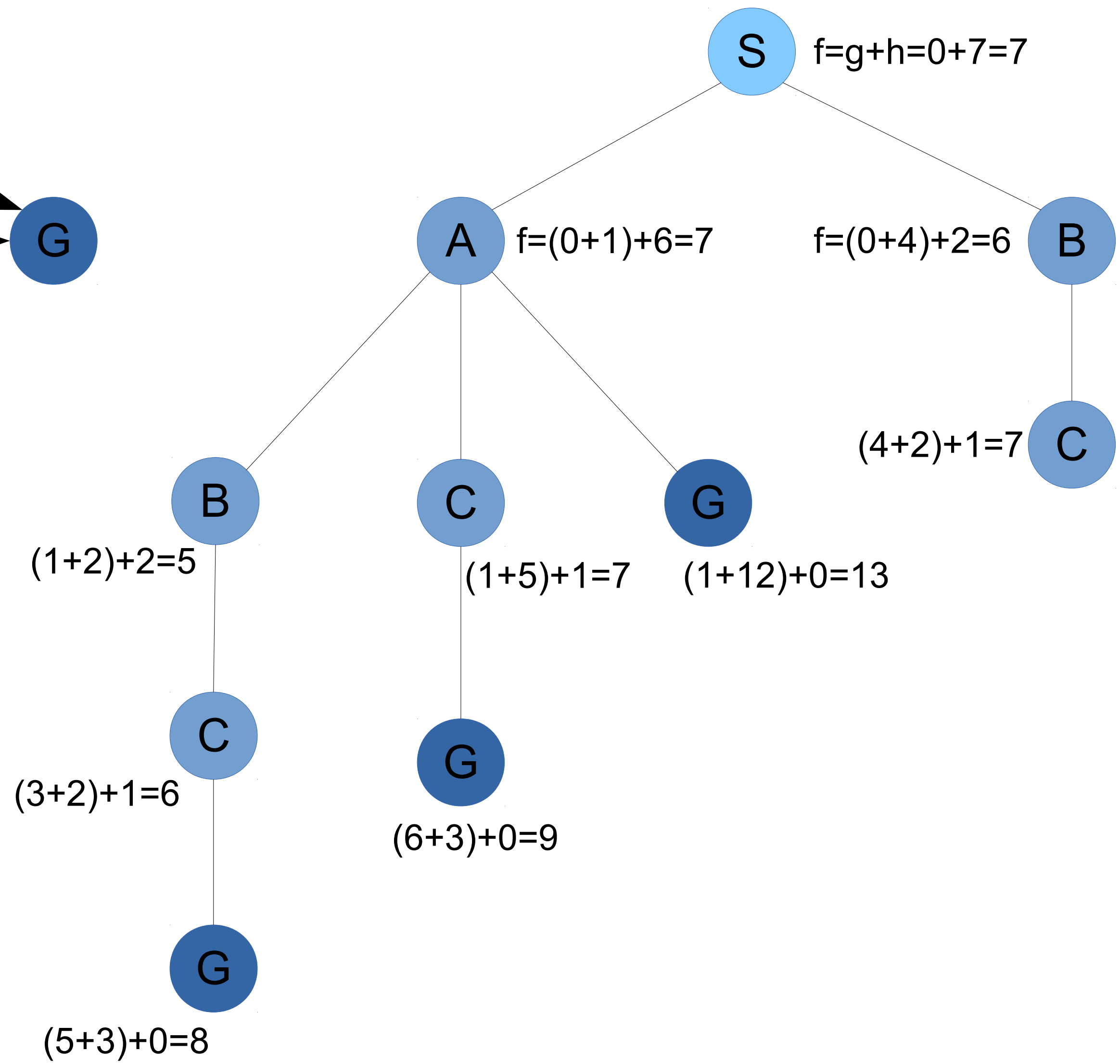
Node	*h(n)*
S	7
A	6
B	2
C	1
G	0



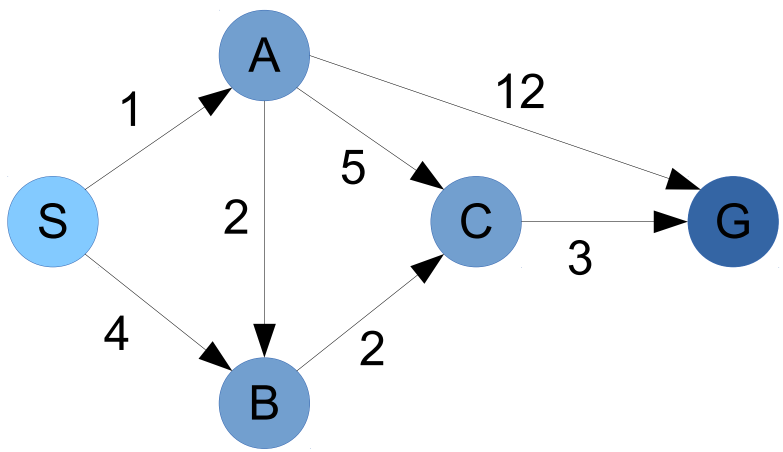
A* Tree Search



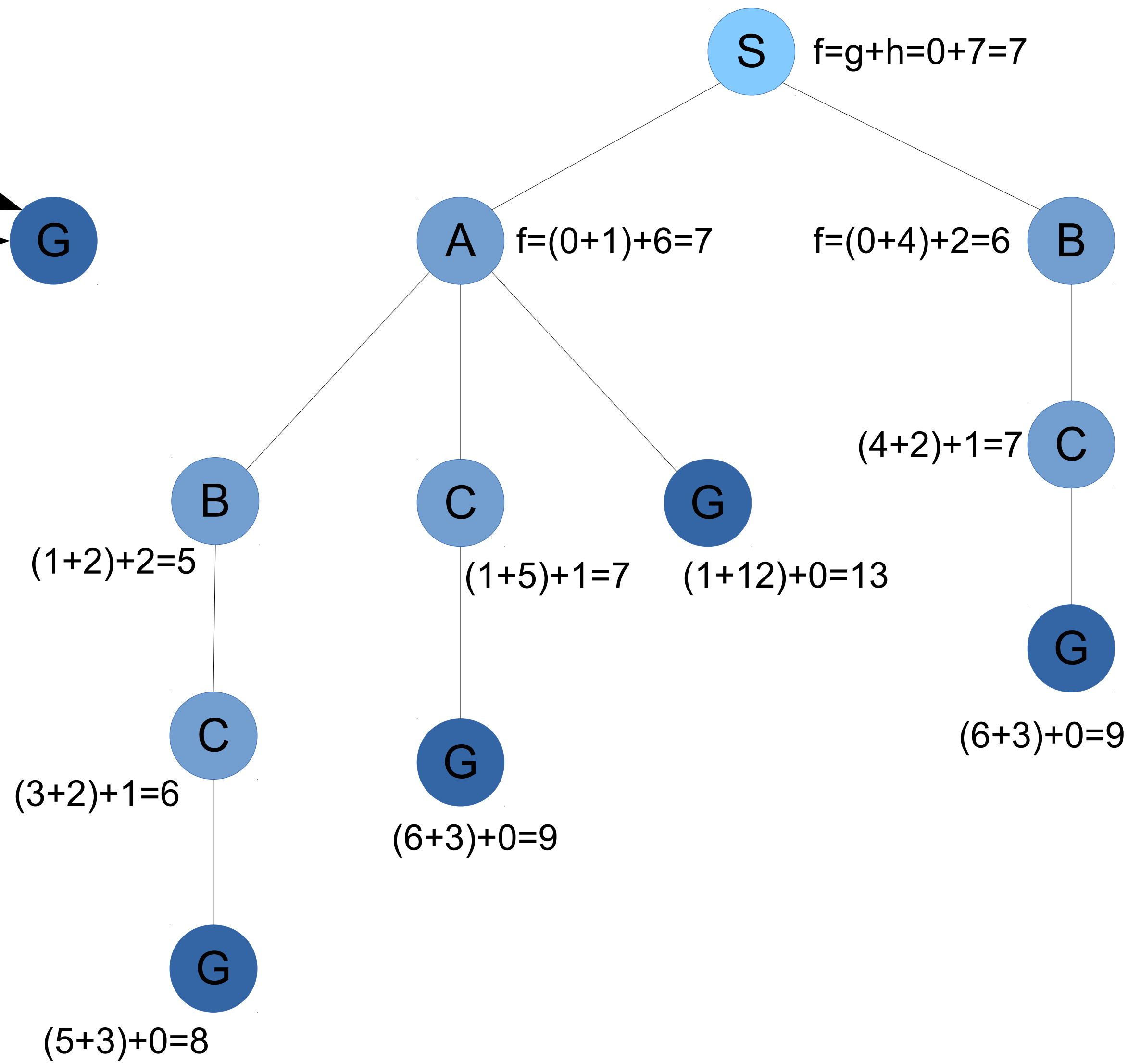
Node	*h(n)*
S	7
A	6
B	2
C	1
G	0



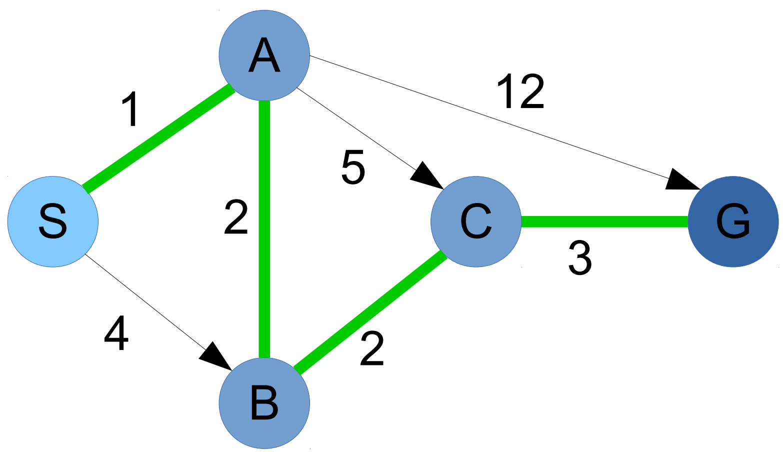
A* Tree Search



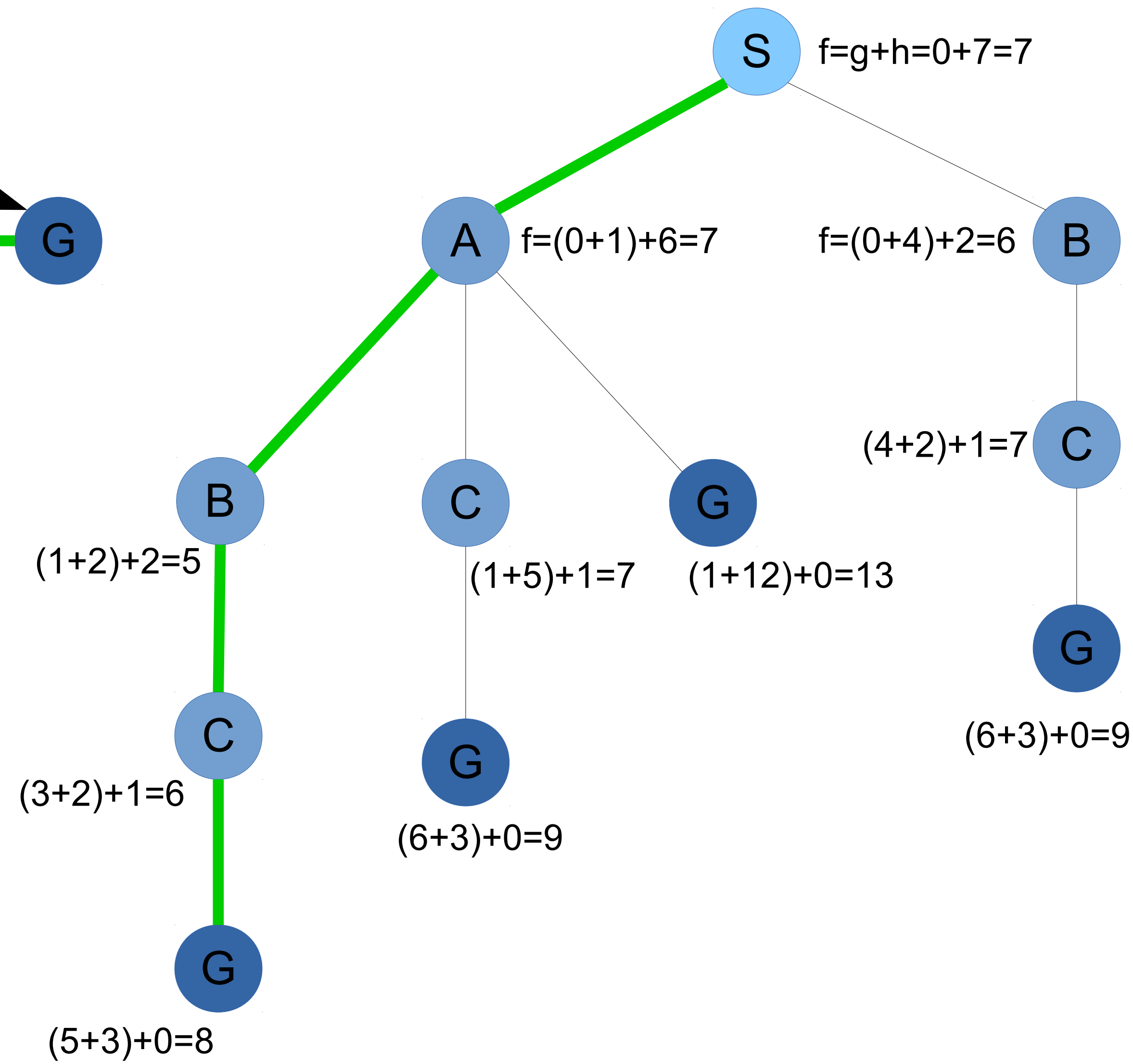
Node	*h(n)*
S	7
A	6
B	2
C	1
G	0



A* Tree Search



Node	*h(n)*
S	7
A	6
B	2
C	1
G	0



A* Tree Search

- Complete?

A* Tree Search

- Complete? Yes

A* Tree Search

- Complete? Yes
- Optimal?

A* Tree Search

- Complete? Yes
- Optimal? Yes if *h* is **admissible**

A* Tree Search

- Complete? Yes
- Optimal? Yes if *h* is **admissible**

A heuristic function is said to be **admissible** if it never overestimates the cost of reaching the goal, 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.

A* Tree Search

Theorem
If *h* is admissible then A* using Tree Search is optimal.

A* Tree Search

Theorem

If h is admissible then A* using Tree Search is optimal.

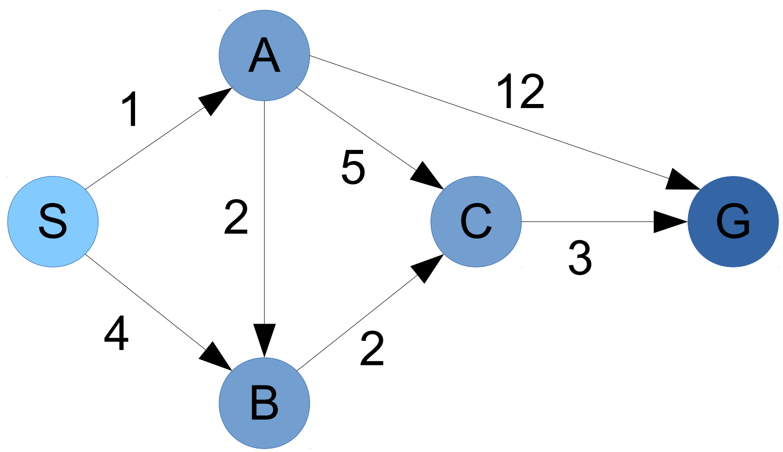
Proof

Suppose that the cost of the optimal solution is C^* and a goal node G has been generated and is in the fringe from a suboptimal path. Since the path is suboptimal we have $f(G) = g(G) + h(G) = g(G) > C^*$.

Let n be an unexpanded node in the fringe such that n is on a shortest path to G. Since h is admissible (never overestimates the cost to the goal) we have $f(n) = g(n) + h(n) \leq C^*$.

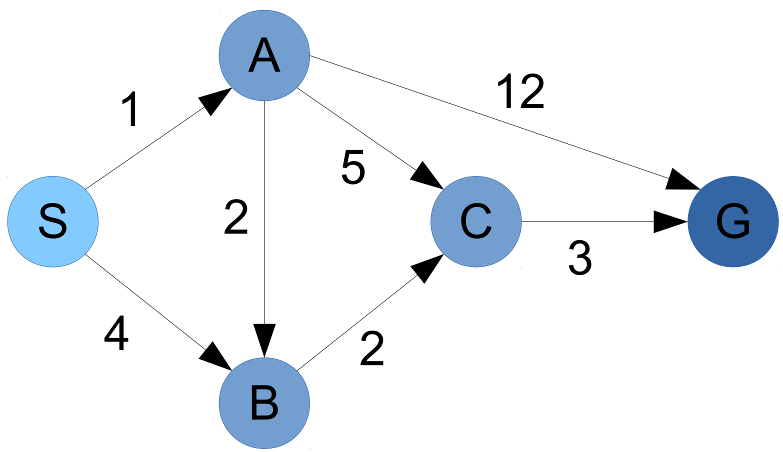
So $f(n) \leq C^* < f(G)$ and the algorithm will prefer to expand n over G.

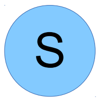
A* Graph Search



Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

A* Graph Search

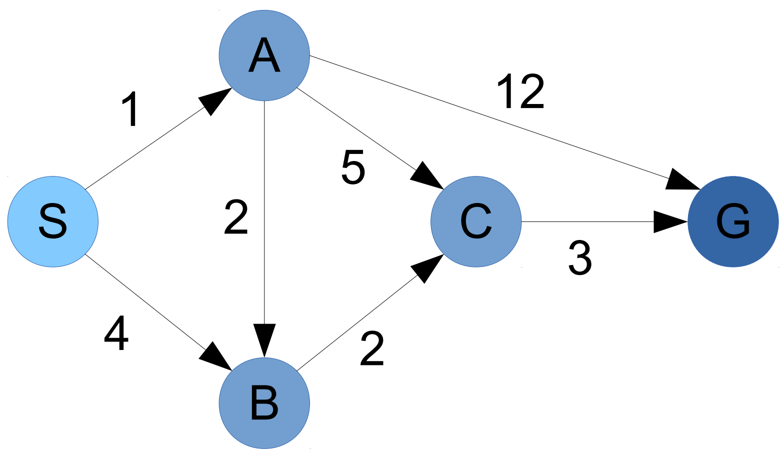


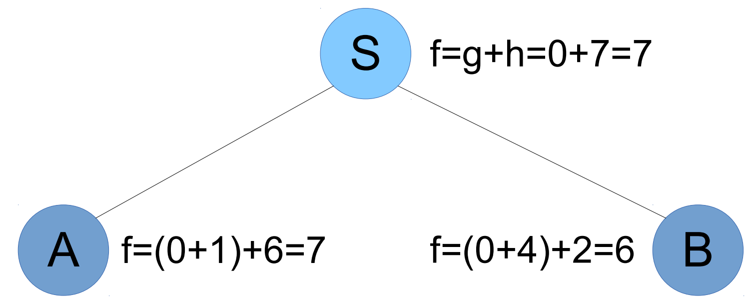


Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {}

A* Graph Search

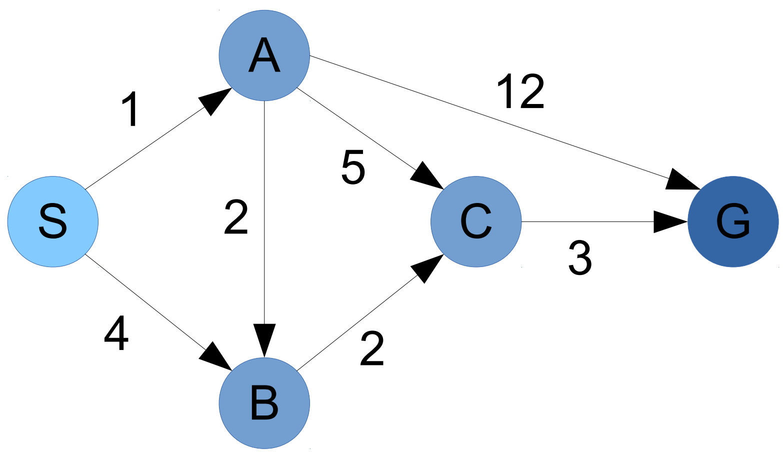


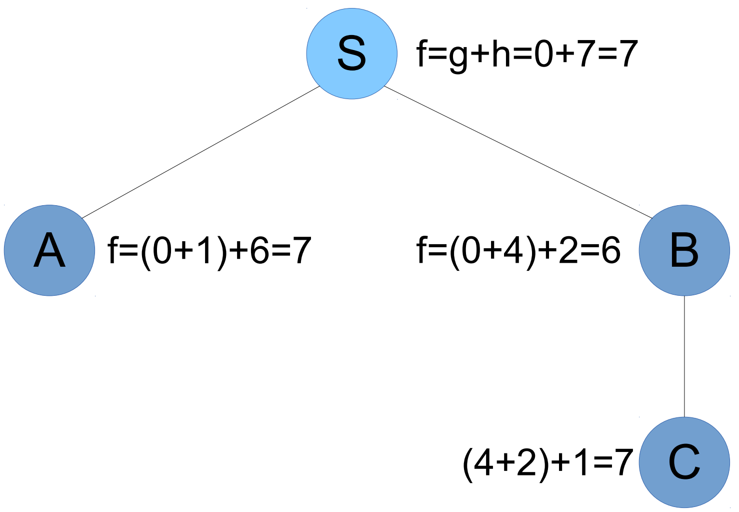


Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {S}

A* Graph Search

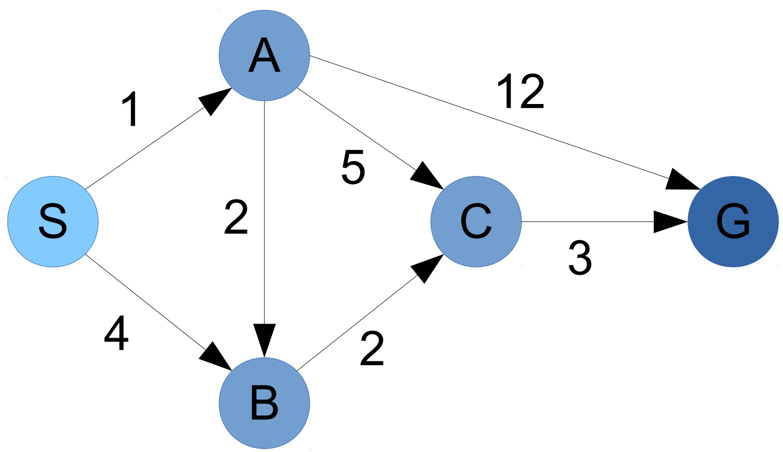


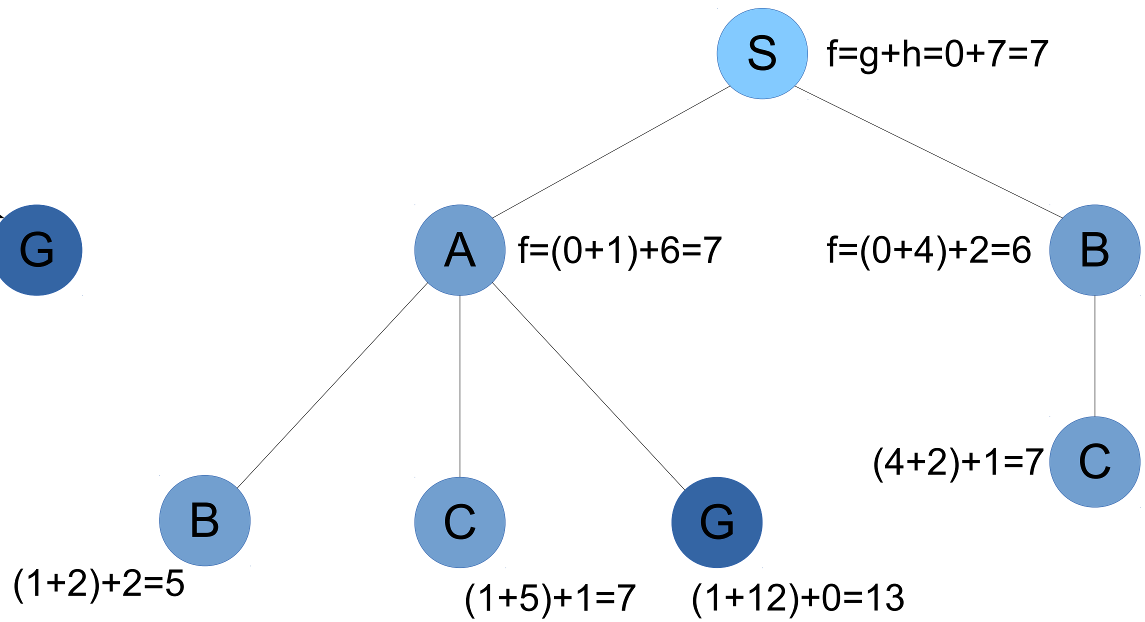


Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {S, B}

A* Graph Search

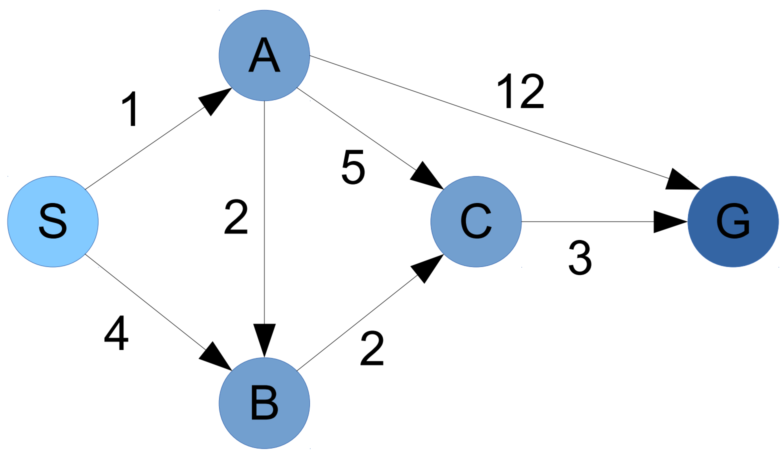


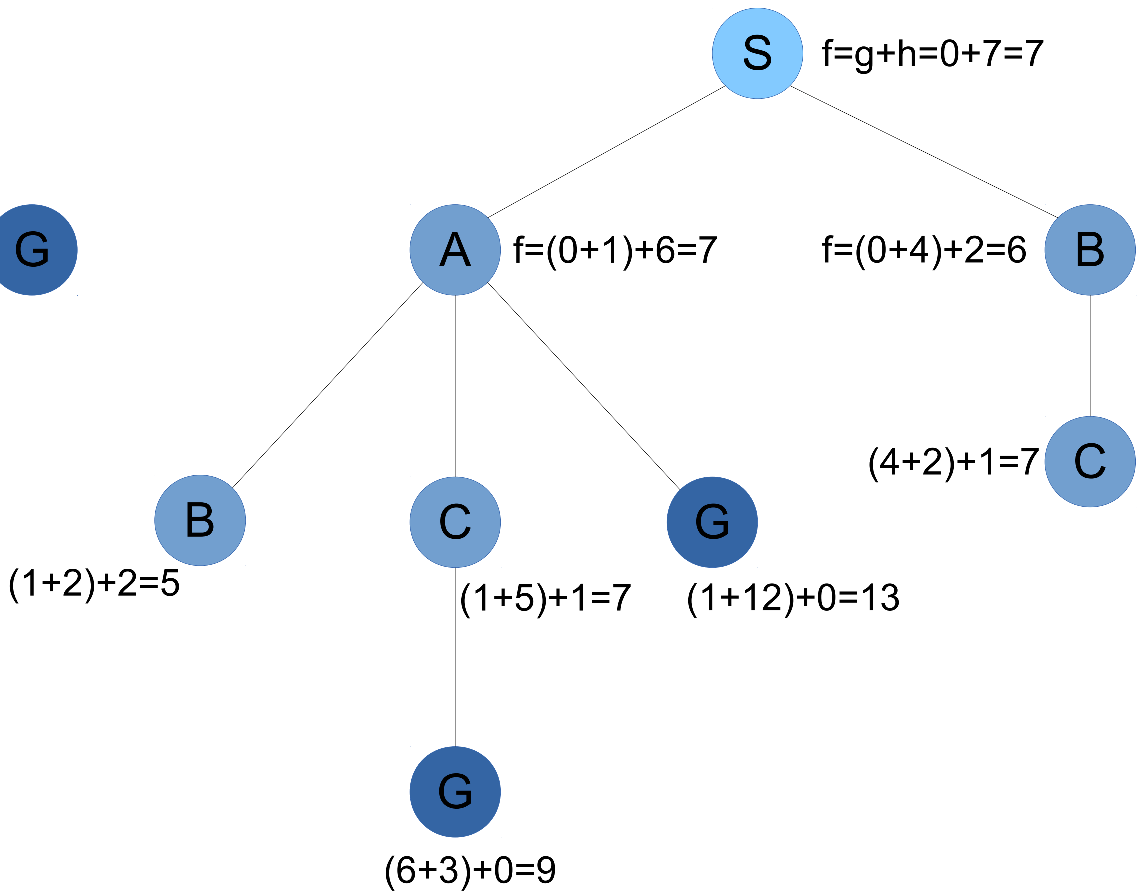


Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {S, B, A}

A* Graph Search

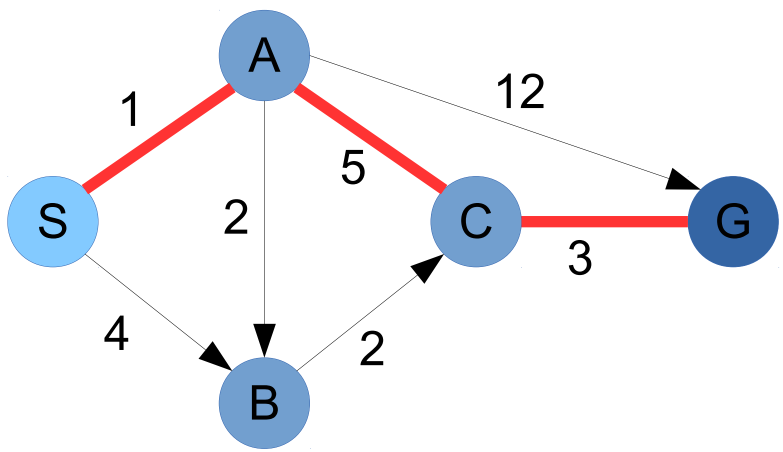


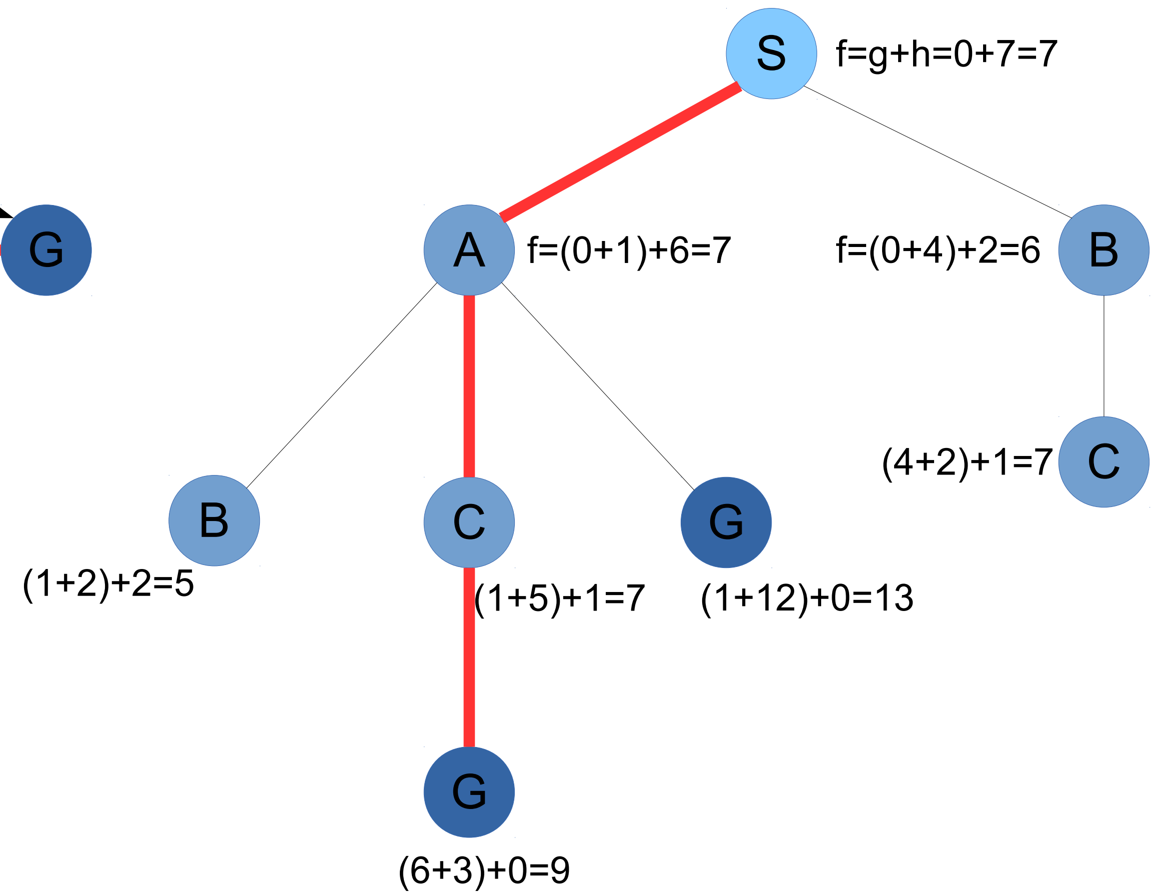


Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {S, B, A, C}

A* Graph Search





Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {S, B, A, C}

A* Graph Search

- A* using Graph Search is not optimal.

A* Graph Search

- A* using Graph Search is not optimal.
- If the optimal path contains a node *n* and *n* is first expanded as part of a suboptimal path then Graph Search will discard node *n* the second time (so it will discard the optimal path).

A* Graph Search

- A* using Graph Search is not optimal.
- If the optimal path contains a node *n* and *n* is first expanded as part of a suboptimal path then Graph Search will discard node *n* the second time (so it will discard the optimal path).
- We need an additional property for the heuristic function.

A* Graph Search

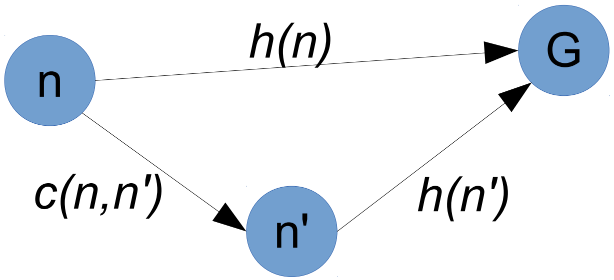
A heuristic function is **consistent** if for every node *n* and every successor *n'* of *n* generated by any action a, the estimated cost of reaching the goal from *n* is no greater than the step cost of getting to *n'* plus the estimated cost of reaching the goal from *n*. In other words:

$$h(n) \leq c(n,n') + h(n')$$

A* Graph Search

A heuristic function is **consistent** if for every node *n* and every successor *n′* of *n* generated by any action a, the estimated cost of reaching the goal from *n* is no greater than the step cost of getting to *n′* plus the estimated cost of reaching the goal from *n*. In other words:

$$h(n) \leq c(n,n') + h(n')$$

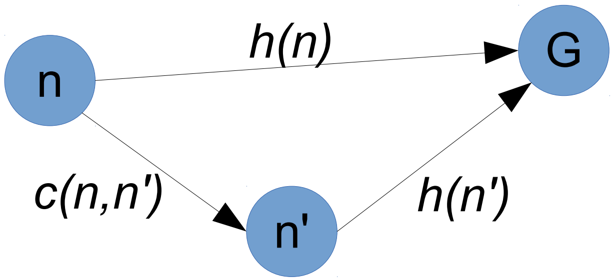


Triangle inequality

A* Graph Search

A heuristic function is **consistent** if for every node *n* and every successor *n'* of *n* generated by any action a, the estimated cost of reaching the goal from *n* is no greater than the step cost of getting to *n'* plus the estimated cost of reaching the goal from *n*. In other words:

$$h(n) \leq c(n,n') + h(n')$$



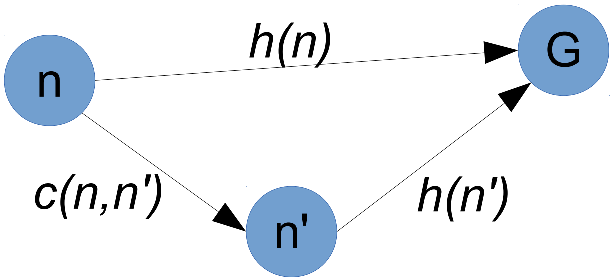
Triangle inequality

If h is consistent, we have:
$f(n') = g(n') + h(n') = g(n) + c(n,n') + h(n') \geq g(n) + h(n) = f(n)$
So $f(n)$ is non-decreasing along any path.

A* Graph Search

A heuristic function is **consistent** if for every node *n* and every successor *n'* of *n* generated by any action a, the estimated cost of reaching the goal from *n* is no greater than the step cost of getting to *n'* plus the estimated cost of reaching the goal from *n*. In other words:

$$h(n) \leq c(n,n') + h(n')$$



Triangle inequality

If h is consistent, we have:
$f(n') = g(n') + h(n') = g(n) + c(n,n') + h(n') \geq g(n) + h(n) = f(n)$
So $f(n)$ is non-decreasing along any path.

A consistent heuristic is also admissible.

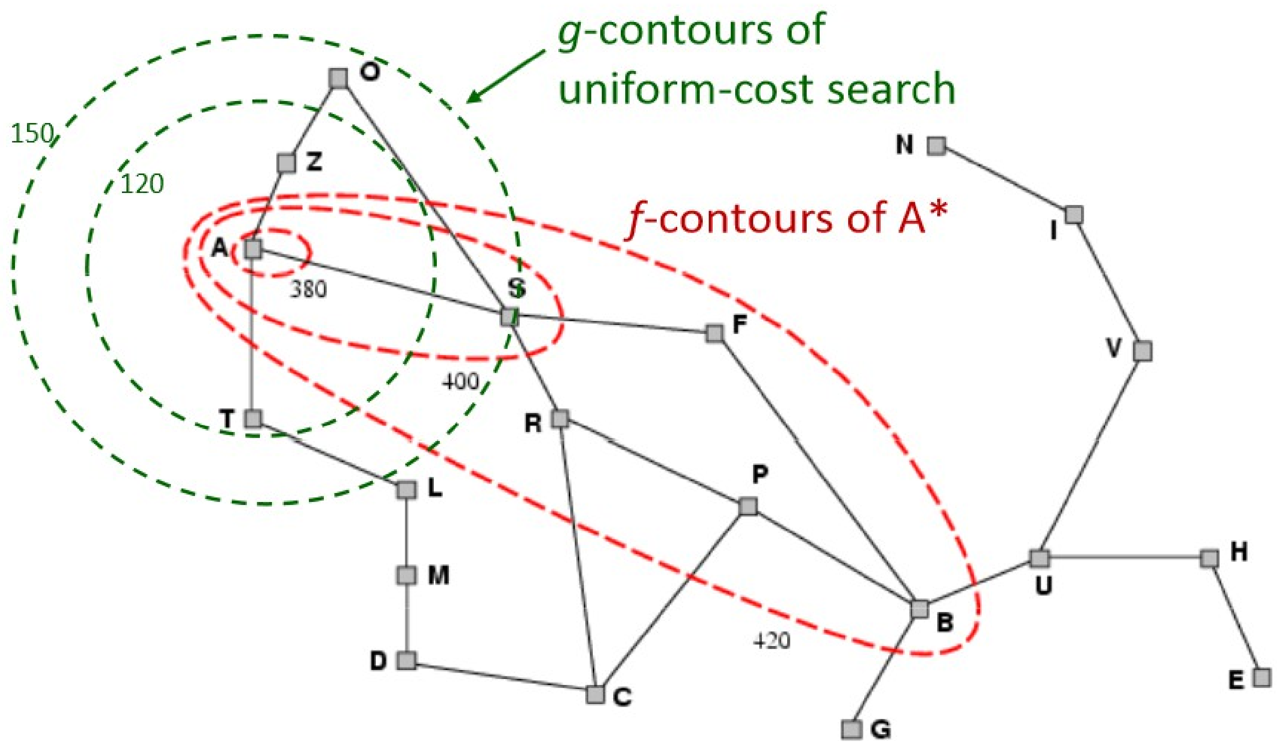
A* Graph Search

Theorem

If *h* is consistent, A* using Graph Search is optimal.

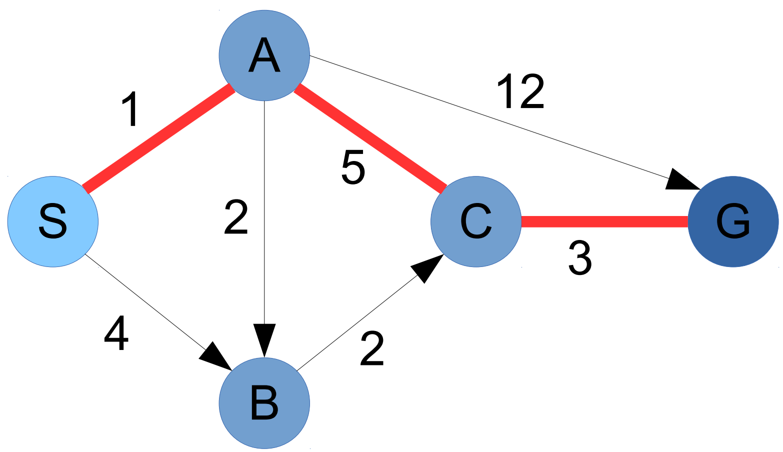
A* Graph Search

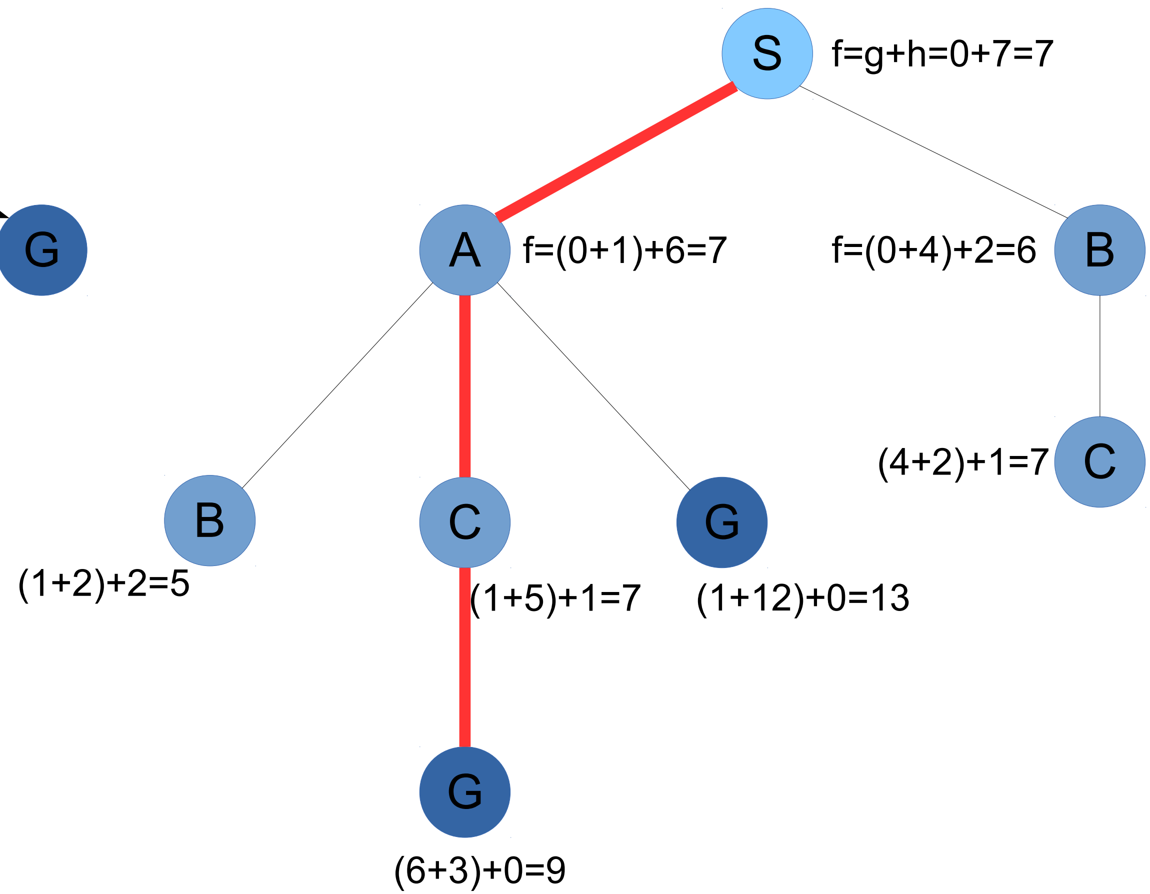
Theorem
If *h* is consistent, A* using Graph Search is optimal.



- A* Graph Search expands nodes of optimal paths in order of increasing *f*-value.
- It expands all nodes with *f(n)* < *C**.

A* Graph Search

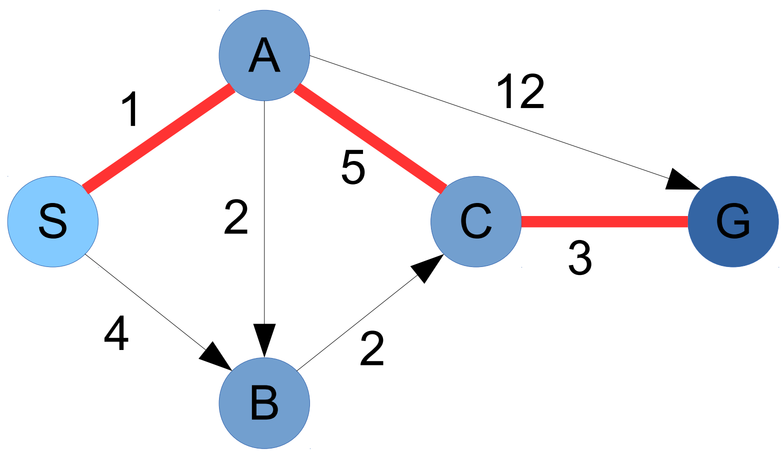


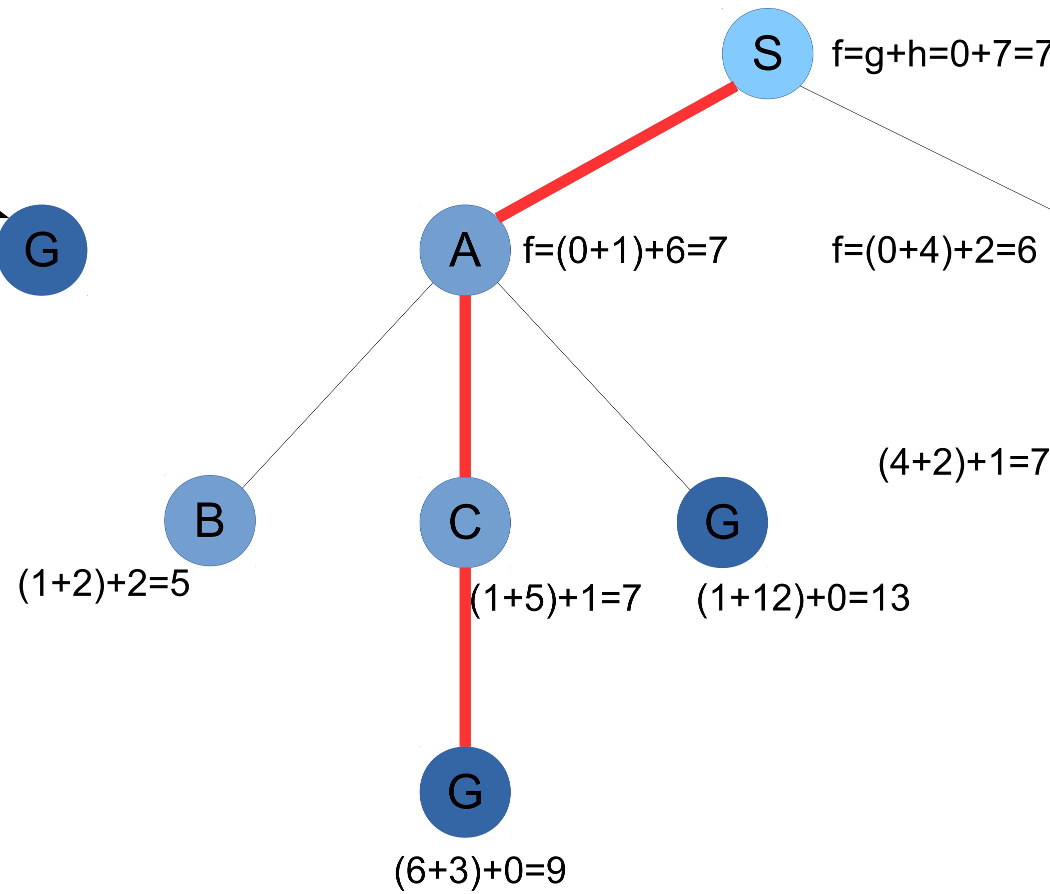


Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {S, B, A, C}

A* Graph Search



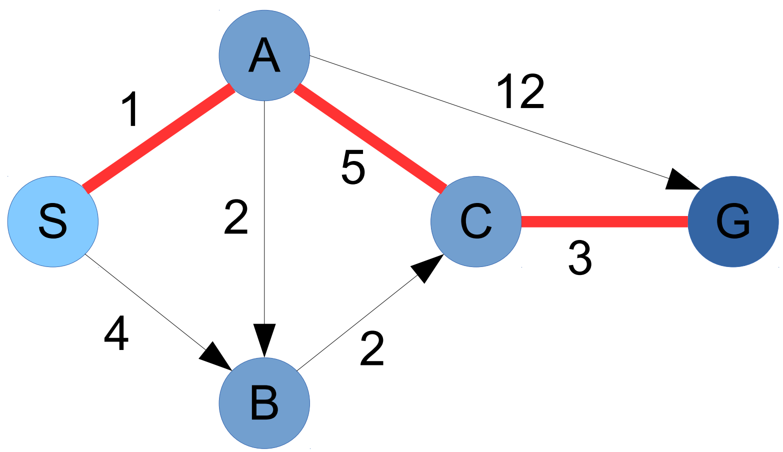


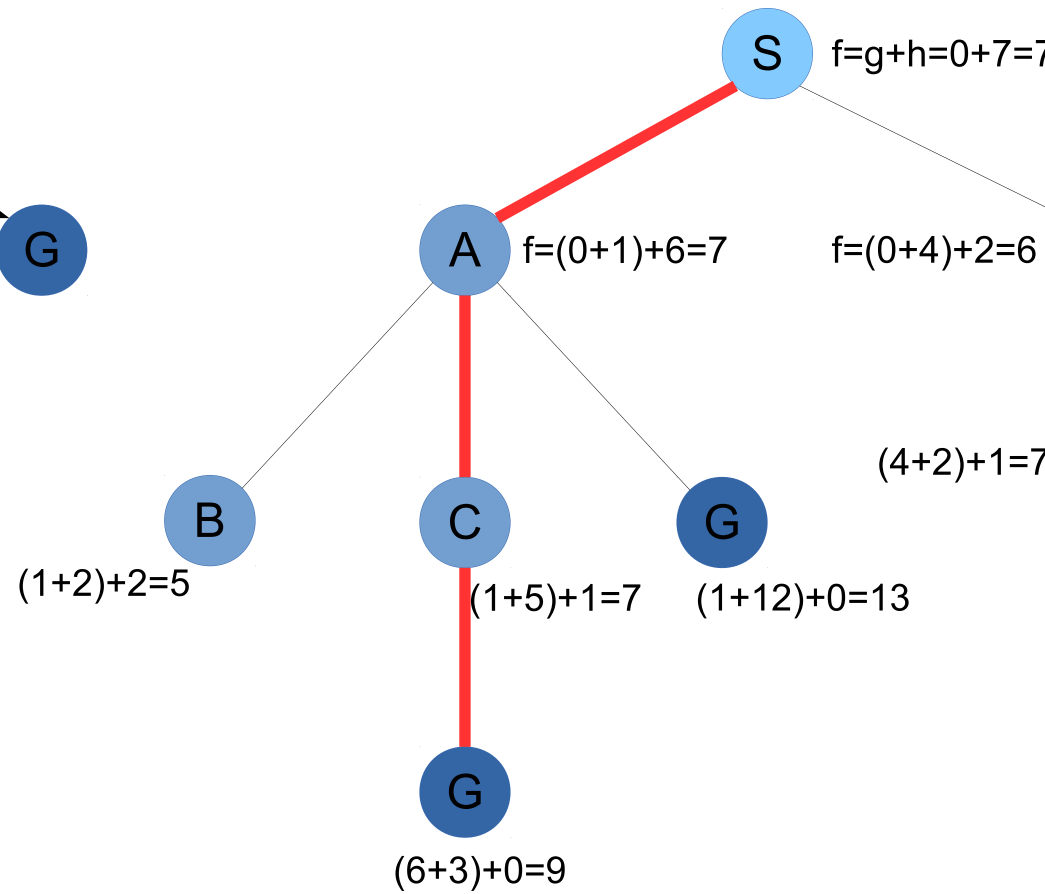
Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

$$h(n) \le c(n,n') + h(n')$$

ClosedSet = {S, B, A, C}

A* Graph Search





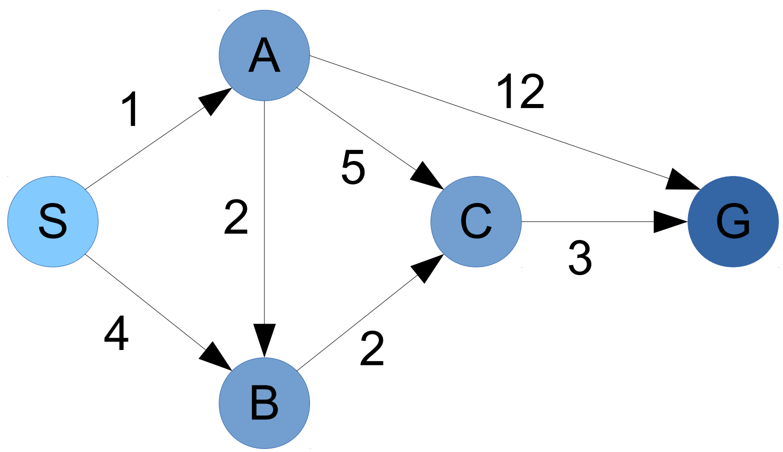
Node	*h(n)*
S	7
A	6
B	2
C	1
G	0

ClosedSet = {S, B, A, C}

$$h(n) \leq c(n,n') + h(n')$$

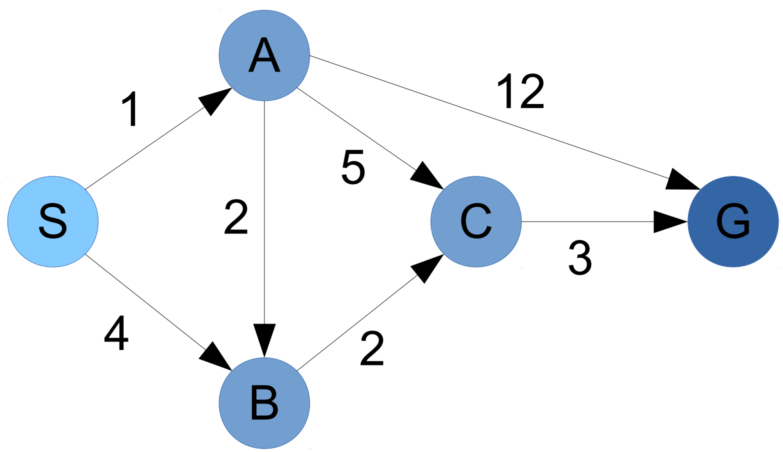
$$6 = h(A) > c(A,B) + h(B) = 2 + 2 = 4$$

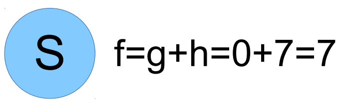
A* Graph Search



Node	*h(n)*
S	7
A	6
B	4
C	2
G	0

A* Graph Search

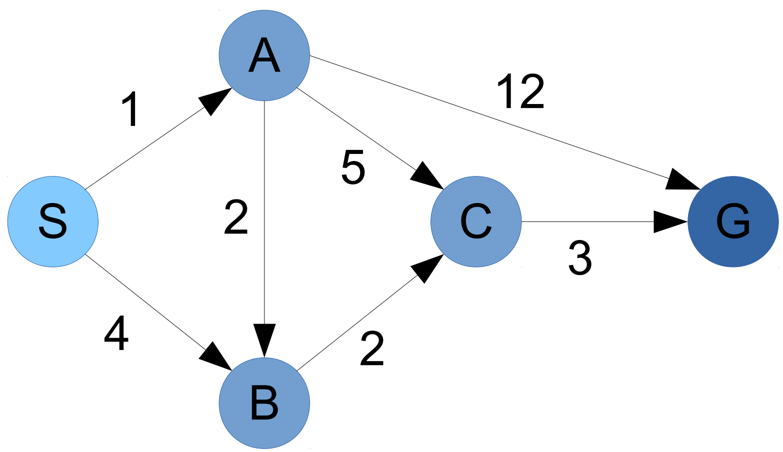


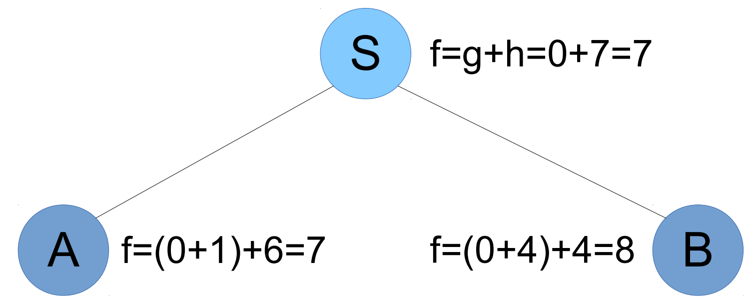


Node	*h(n)*
S	7
A	6
B	4
C	2
G	0

ClosedSet = {}

A* Graph Search

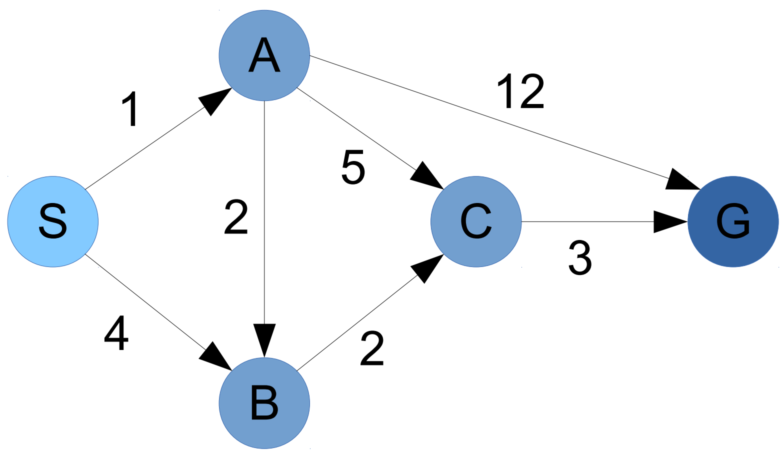


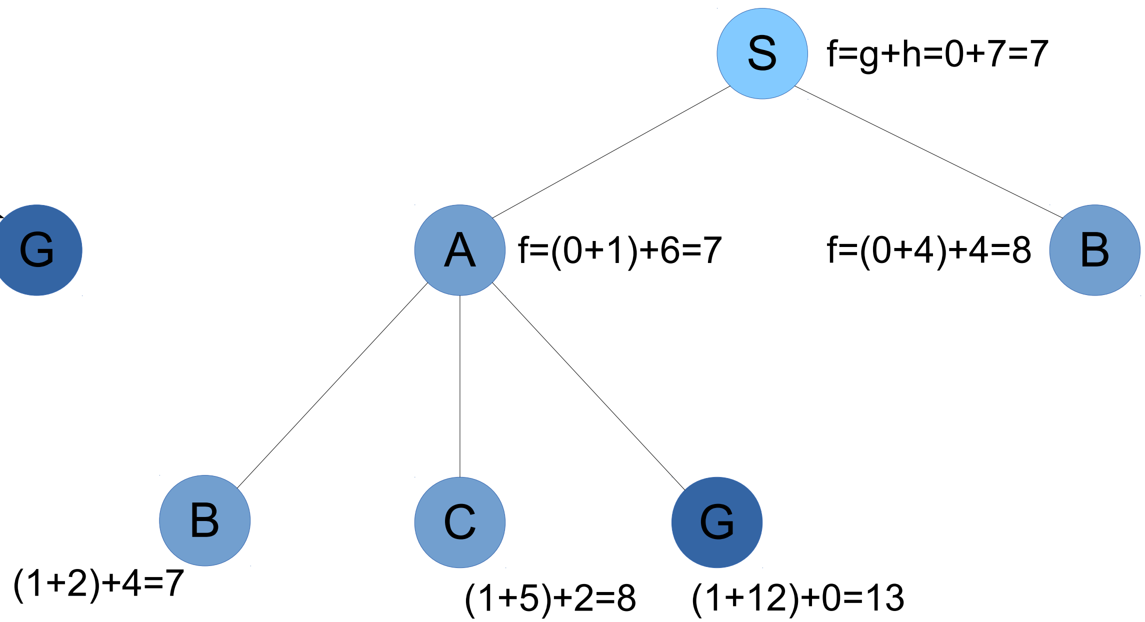


Node	*h(n)*
S	7
A	6
B	4
C	2
G	0

ClosedSet = {S}

A* Graph Search

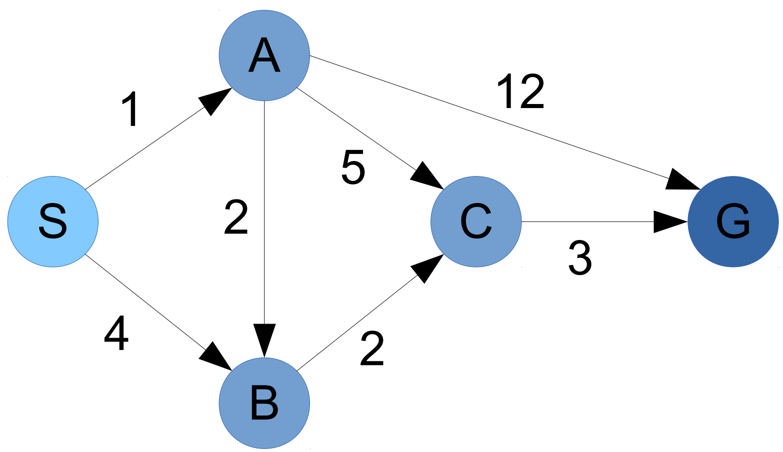


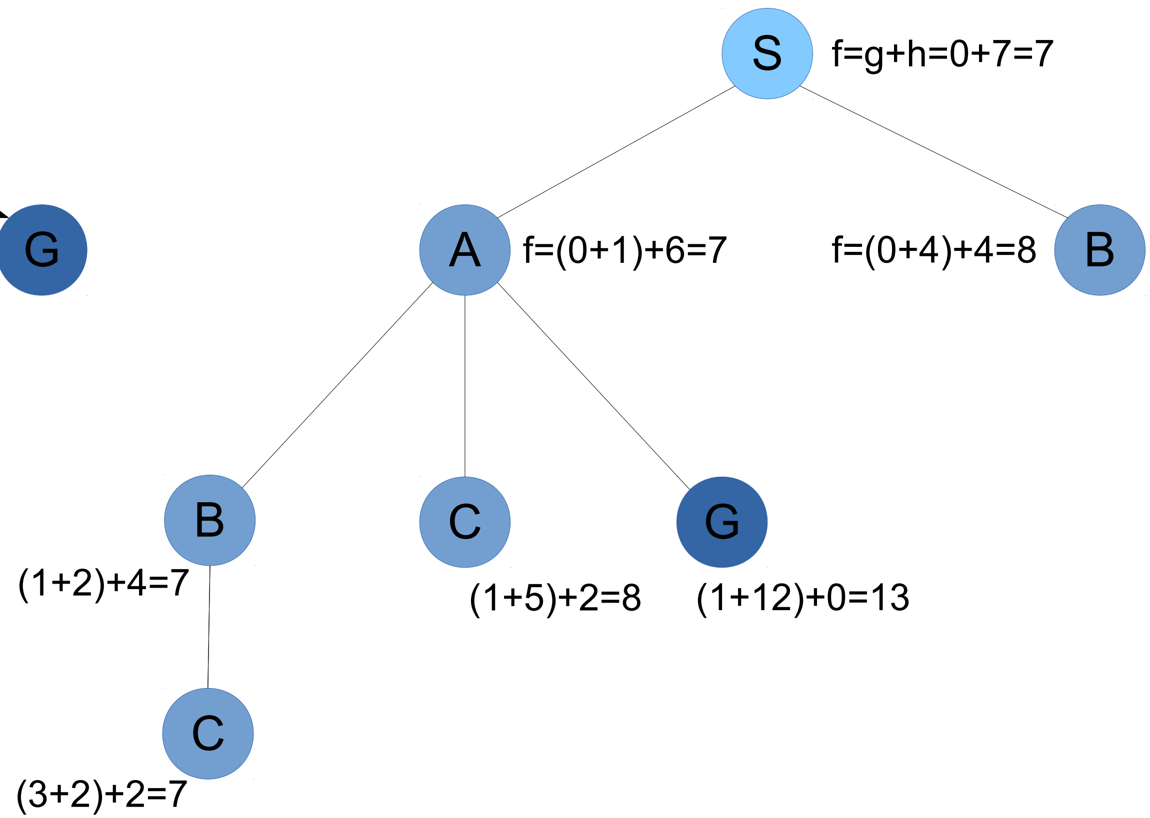


Node	*h(n)*
S	7
A	6
B	4
C	2
G	0

ClosedSet = {S, A}

A* Graph Search

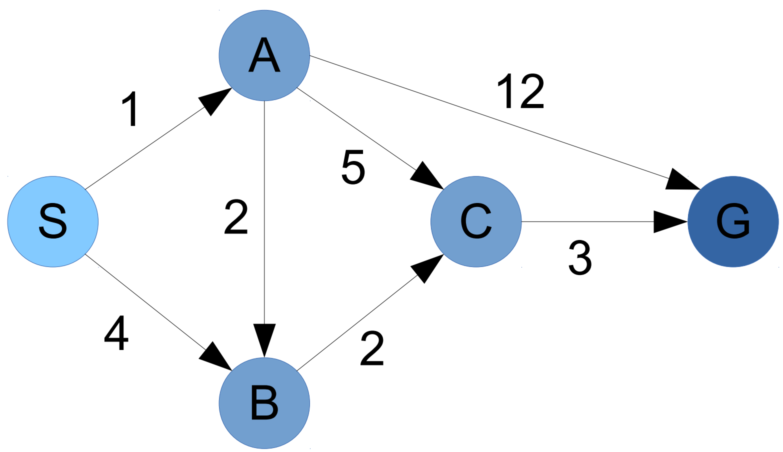




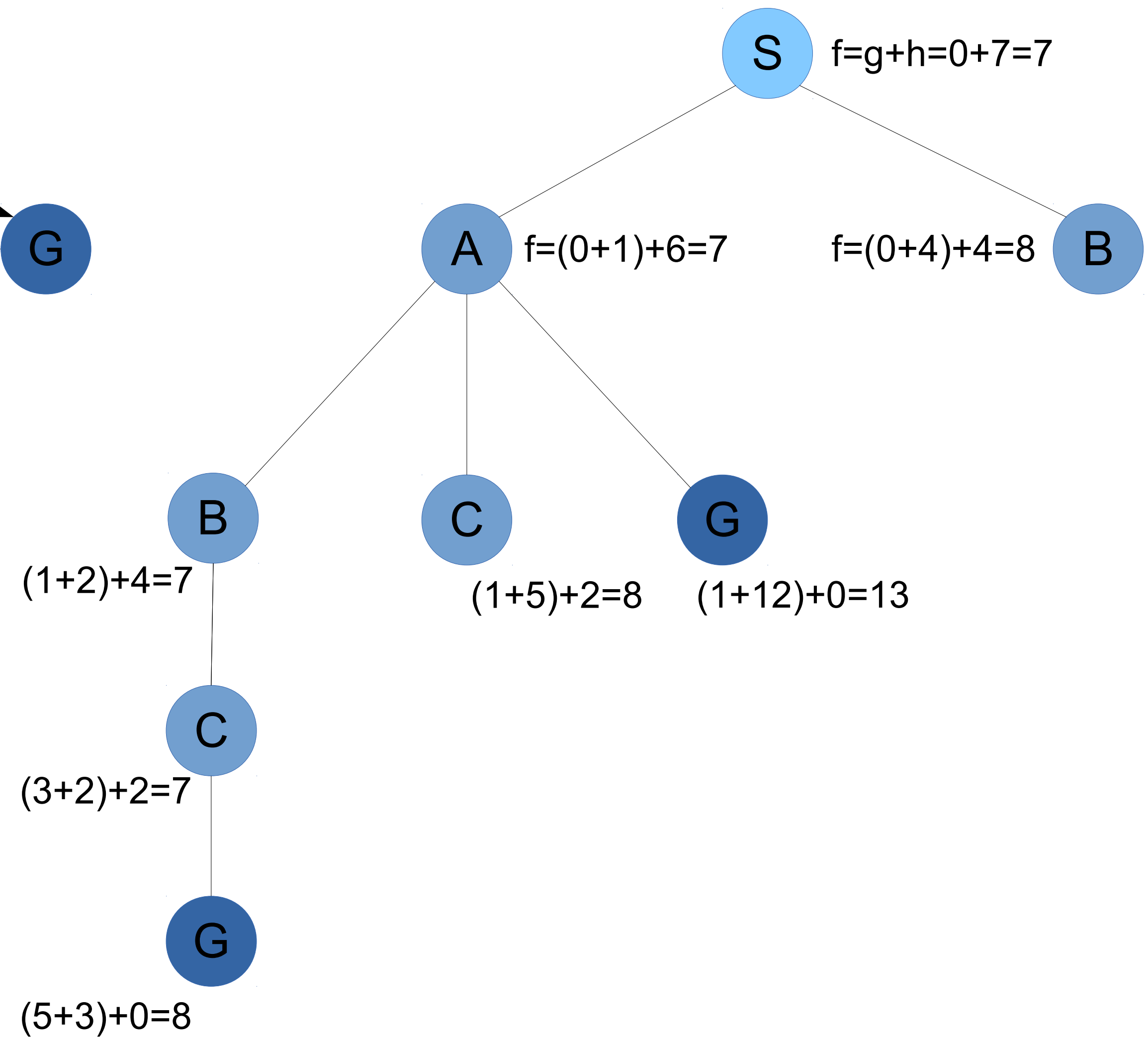
Node	*h(n)*
S	7
A	6
B	4
C	2
G	0

ClosedSet = {S, A, B}

A* Graph Search

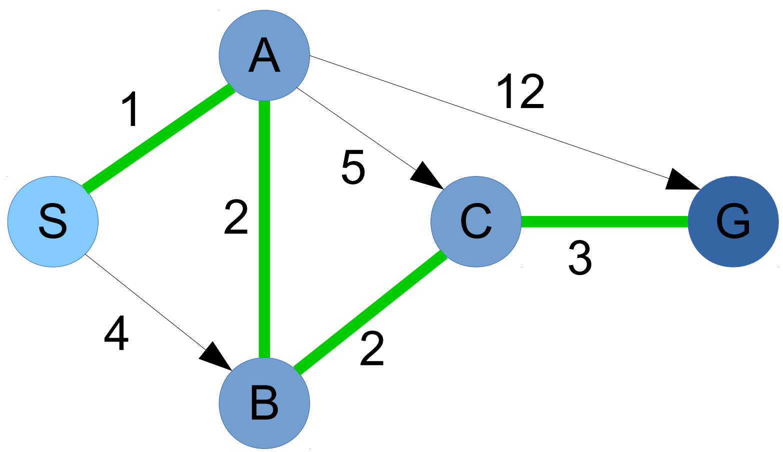


Node	*h(n)*
S	7
A	6
B	4
C	2
G	0

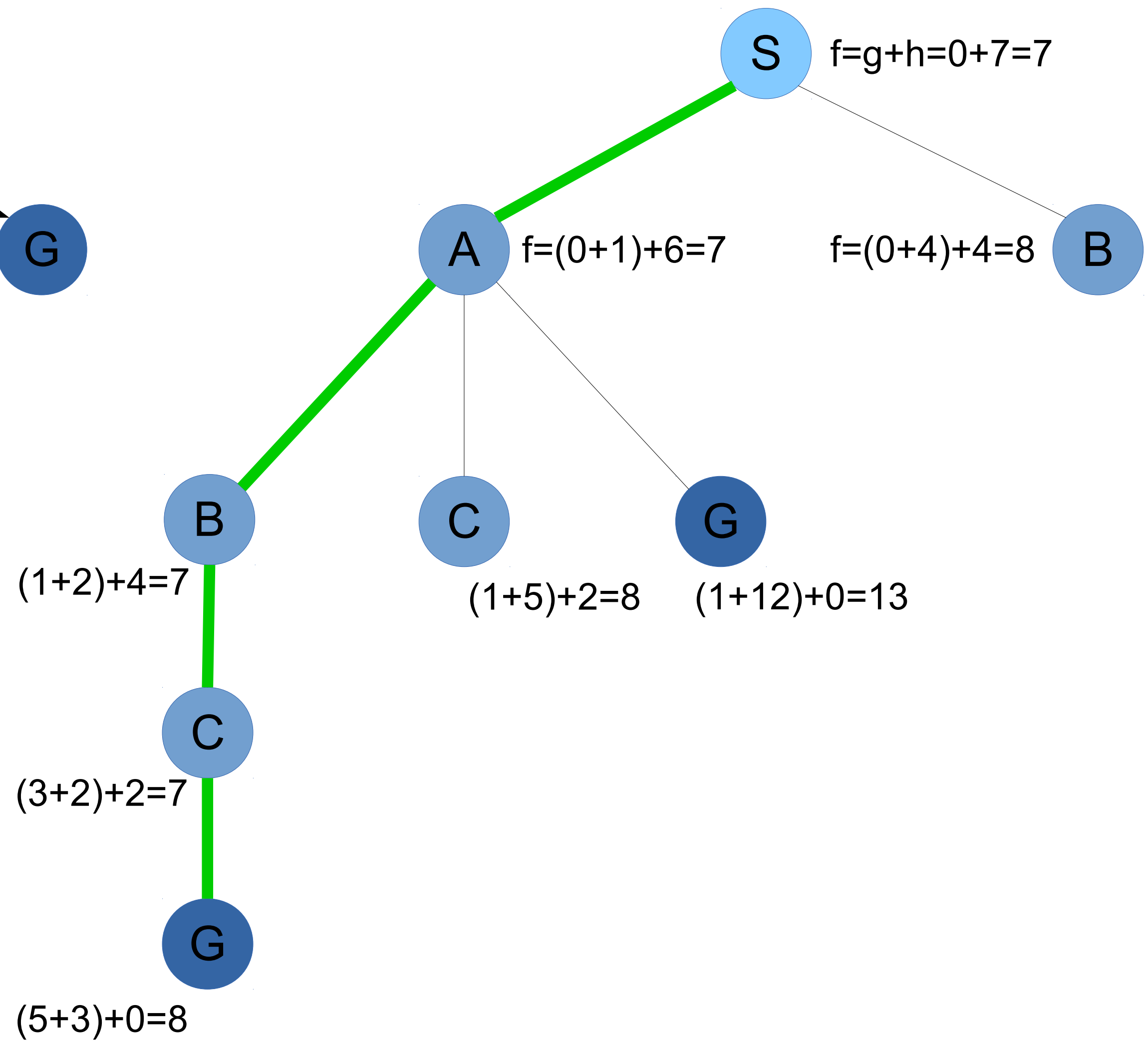


ClosedSet = {S, A, B, C}

A* Graph Search



Node	*h(n)*
S	7
A	6
B	4
C	2
G	0



ClosedSet = {S, A, B, C}

Heuristic functions

Heuristic functions

- How do we generate admissible heuristics?

Heuristic functions

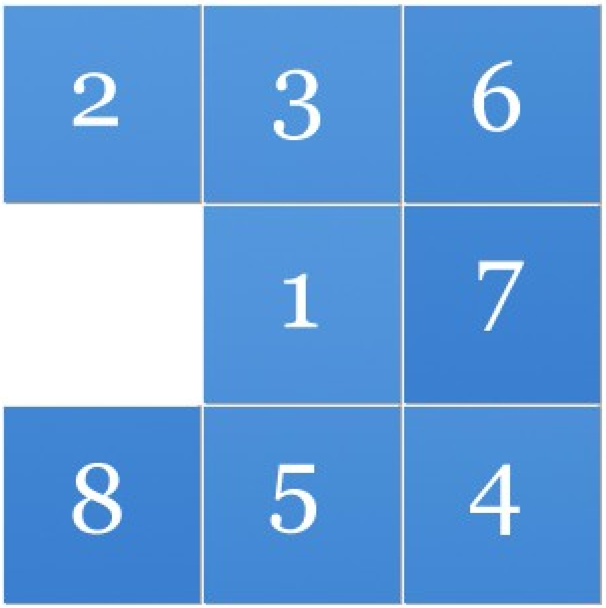
- How do we generate admissible heuristics?
- To come up with heuristic functions one can study **relaxed problems** from which some restrictions of the original problem have been removed.

Heuristic functions

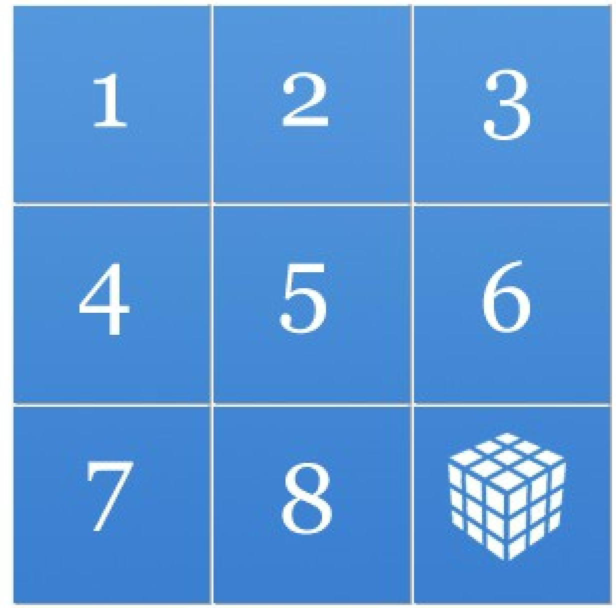
- How do we generate admissible heuristics?
- To come up with heuristic functions one can study **relaxed problems** from which some restrictions of the original problem have been removed.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem (does not overestimate).

Heuristic functions

e.g. for the 8-puzzle



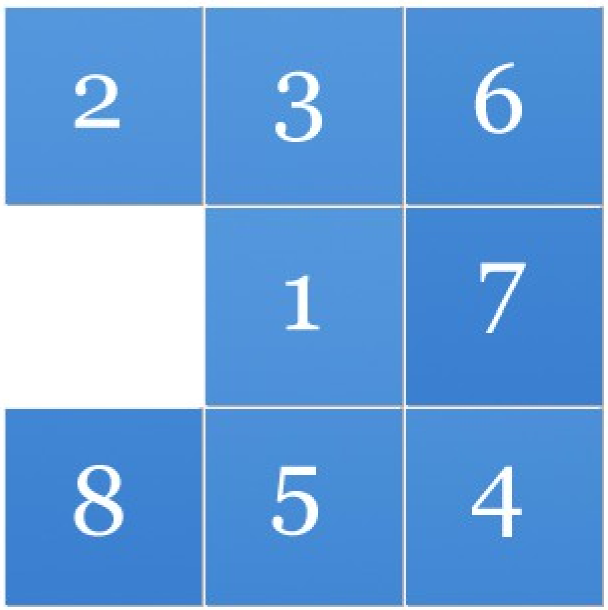
Start state



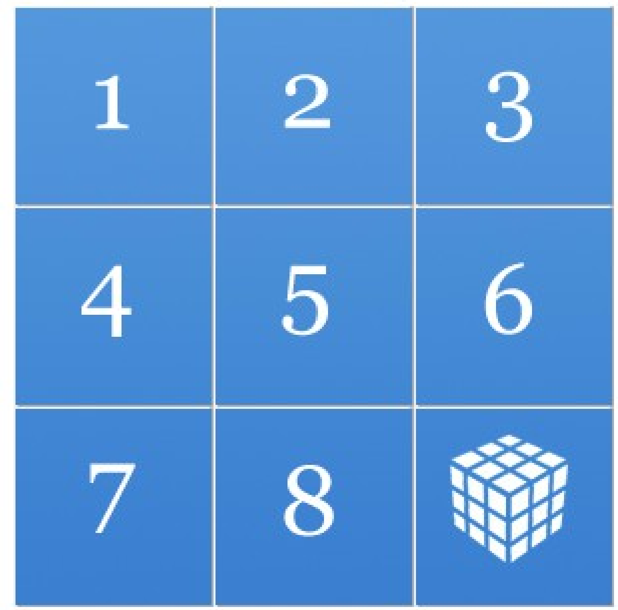
Goal state

Heuristic functions

e.g. for the 8-puzzle



Start state



Goal state

- $h_1(n)$ = number of misplaced tiles – Hamming distance ($h_1(start)$ = 8)
- $h_2(n)$ = sum of Manhattan distances to goal positions ($h_2(start)$ = 13)

Heuristic functions

- Let h_1 and h_2 be heuristic functions. If $h_2(n) \geq h_1(n)$ for all nodes n then we say that h_2 **dominates** h_1 (or h_2 is more informed than h_1).

Heuristic functions

- Let h_1 and h_2 be heuristic functions. If $h_2(n) \geq h_1(n)$ for all nodes n then we say that h_2 **dominates** h_1 (or h_2 is more informed than h_1).
- It is easy to see that for the previous heuristics for the 8-puzzle h_2 dominates h_1.

Heuristic functions

- Let h_1 and h_2 be heuristic functions. If $h_2(n) \geq h_1(n)$ for all nodes n then we say that h_2 **dominates** h_1 (or h_2 is more informed than h_1).
- It is easy to see that for the previous heuristics for the 8-puzzle h_2 dominates h_1.

Theorem

If h_2 dominates h_1 then A* with h_2 will expand less than or equal nodes of A* with h_1.

Heuristic functions

- Let h_1 and h_2 be heuristic functions. If $h_2(n) \geq h_1(n)$ for all nodes n then we say that h_2 **dominates** h_1 (or h_2 is more informed than h_1).
- It is easy to see that for the previous heuristics for the 8-puzzle h_2 dominates h_1.

Theorem

If h_2 dominates h_1 then A* with h_2 will expand less than or equal nodes of A* with h_1.

Proof

We said that A* expands all nodes with evaluation $f(n) < C^*$ where C^* is the cost of the optimal solution.
Equivalently A* expands all nodes with $h(n) < C^* - g(n)$.
Let m be a node which expands by A* with h_2. Then $h_2(m) < C^* - g(m)$ *and because* $h_1(m) \leq h_2(m)$ we have $h_1(m) < C^* - g(m)$.
Therefore m will be expanded by A* with h_1.

Heuristic functions

Among several admissible heuristic the one with highest value is the fastest.

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Thanks for Listening!

THE END