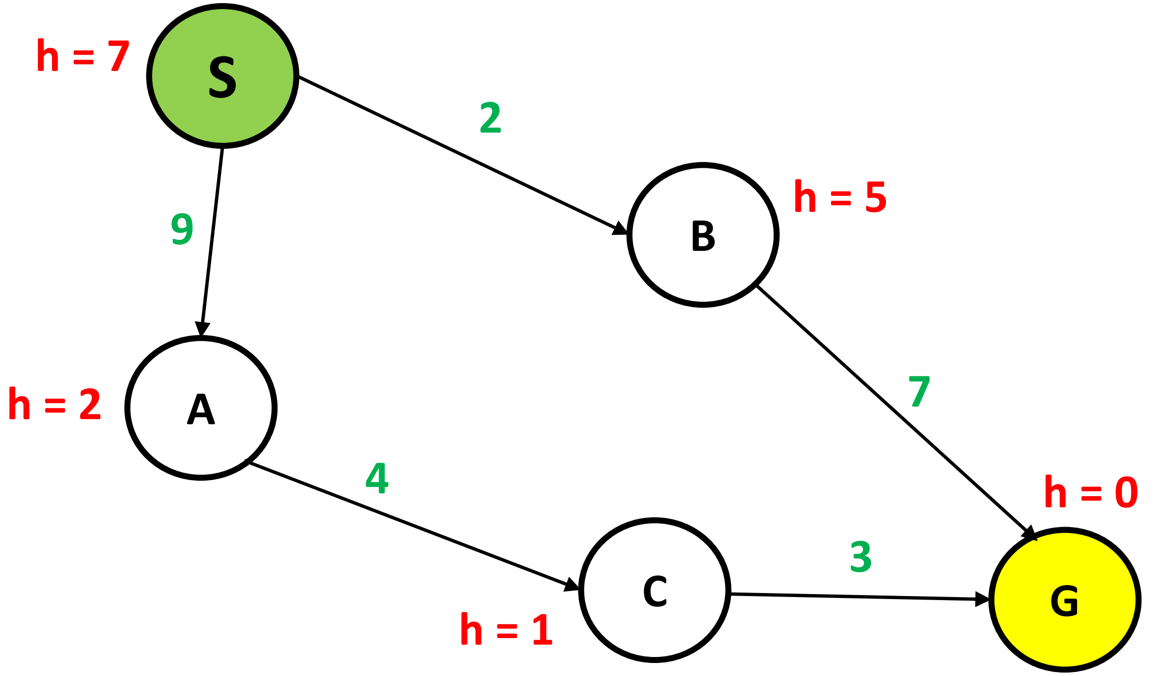
Informed Search

- **Informed:** Some additional information (heuristics) other than the problem description is there , which guides you proceeding from which path would be better
- **Heuristic** is a piece of domain specific knowledge that guides the search

Heuristic Function (h)

- g(n) : Actual path cost from the ***start node*** to node ***n***
- h(n) : Estimated cost of the cheapest path from ***n*** to the ***goal***

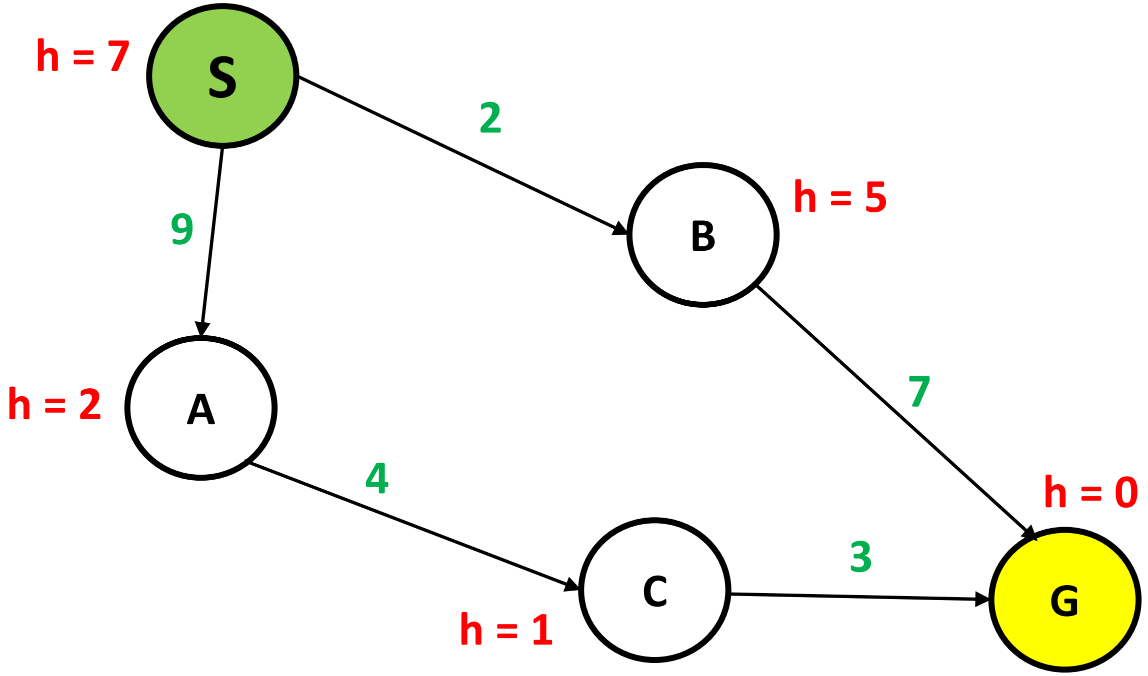


g (C)	
h (C)	

Note: Heuristic value is computed for each node from *domain knowledge* (directly given to you)

Heuristic Function (h)

- g(n) : Actual path cost from the ***start node*** to node ***n***
- h(n) : Estimated cost of the cheapest path from ***n*** to the ***goal***



g (C)	**13**
h (C)	**1**

Note: Heuristic value is computed for each node from *domain knowledge* (directly given to you)

Best First Search - Greedy Algorithm

- Expands the node that ***appears*** to be closest to the goal.
- **Implementation:** Maintain priority queue and pull off node with least heuristic.
- **Example:** Consider Travelling Salesman problem

Travelling Salesman Problem Optimality?

Straight Line Distance can be a good heuristic (h_{SLD})

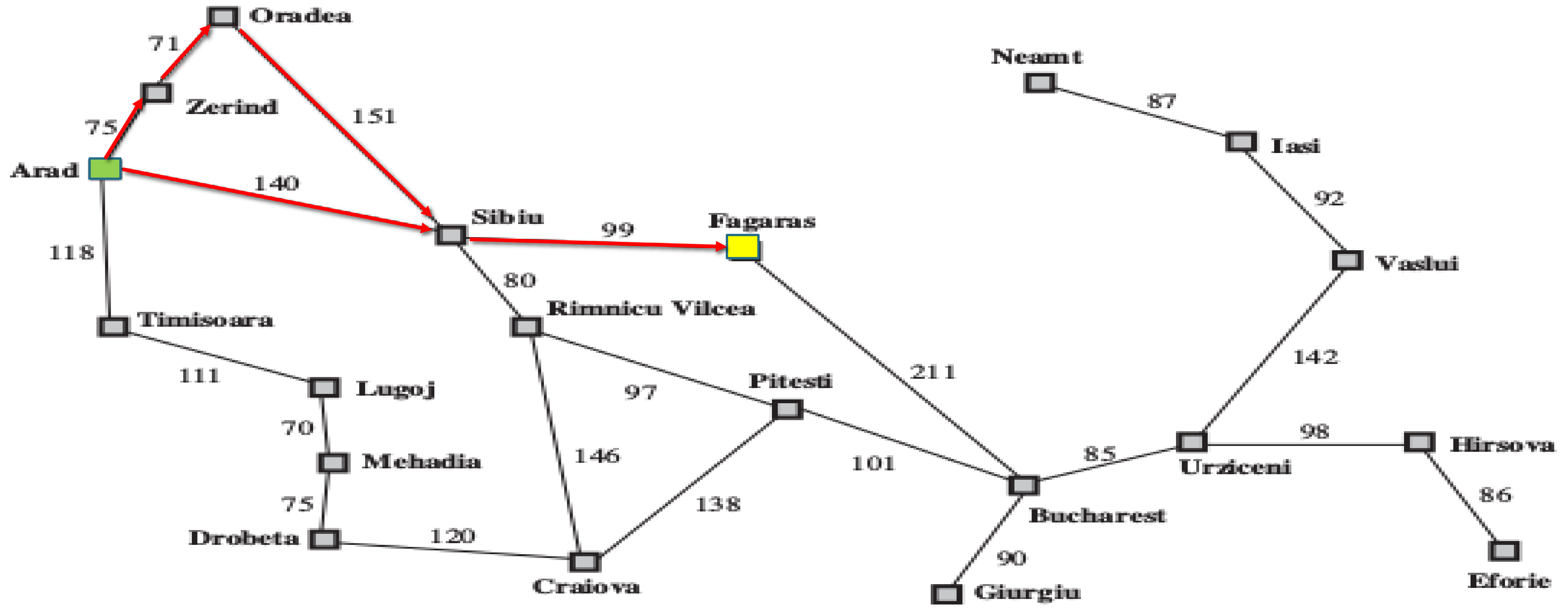


Figure 3.2 A simplified road map of part of Romania.

Does it guarantee to provide optimal solution?

Travelling Salesman Problem Optimality?

Straight Line Distance can be a good heuristic (h_{SLD})

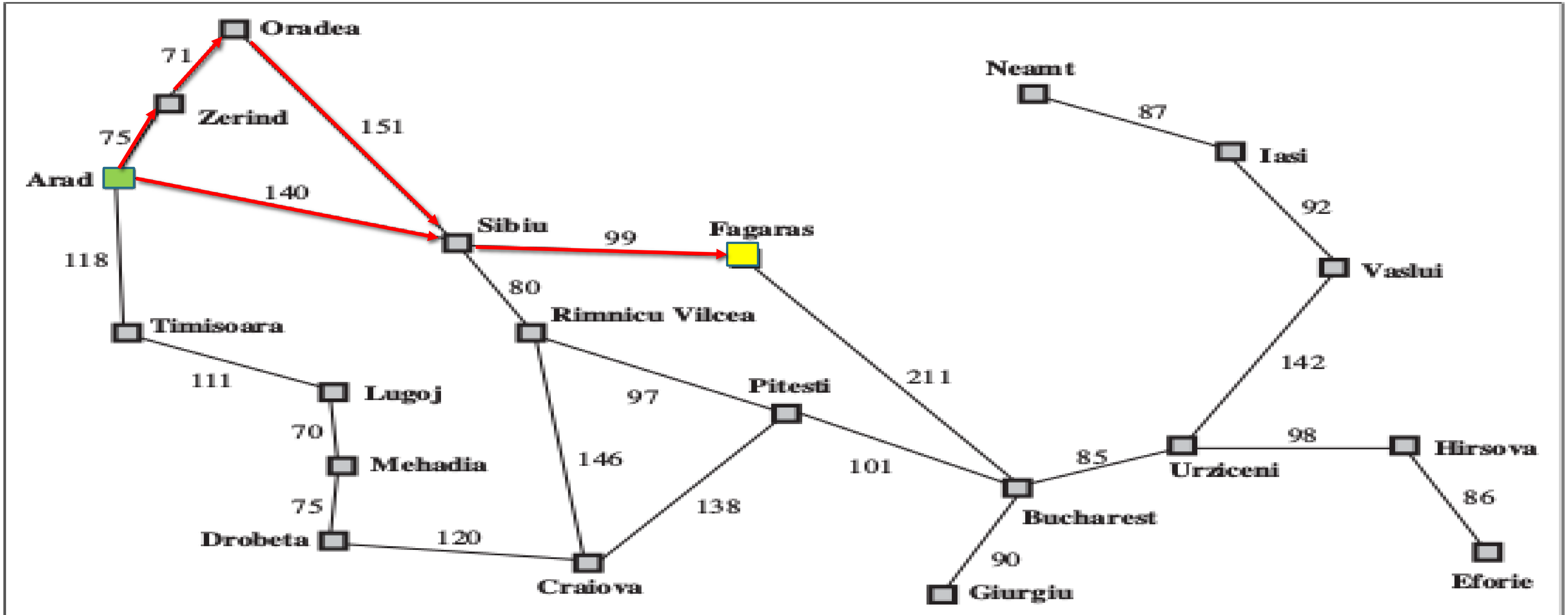
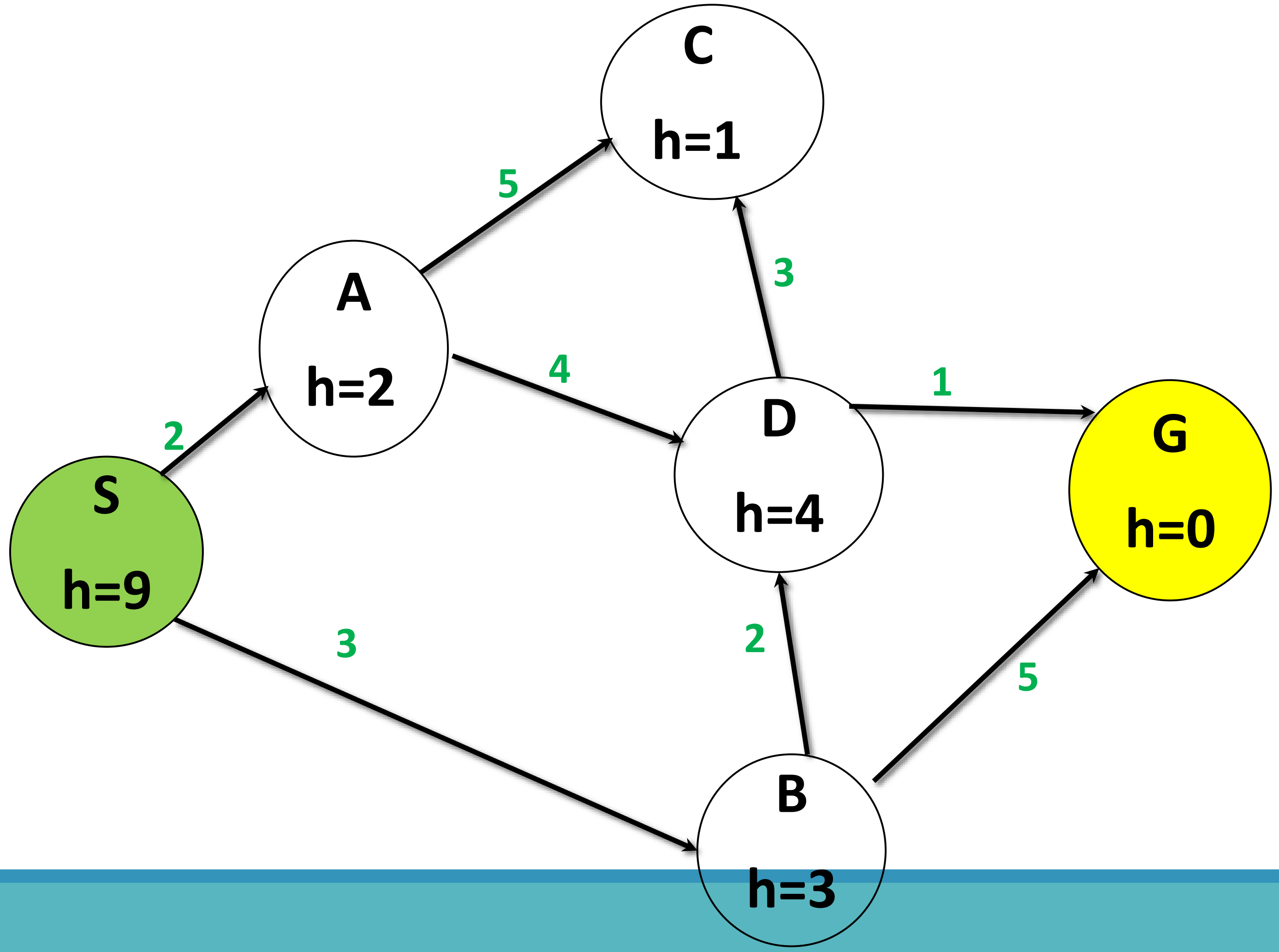


Figure 3.2 A simplified road map of part of Romania.

Does it guarantee to provide optimal solution?

Greed is curse

Greedy Algorithm - Example



Path : SBG

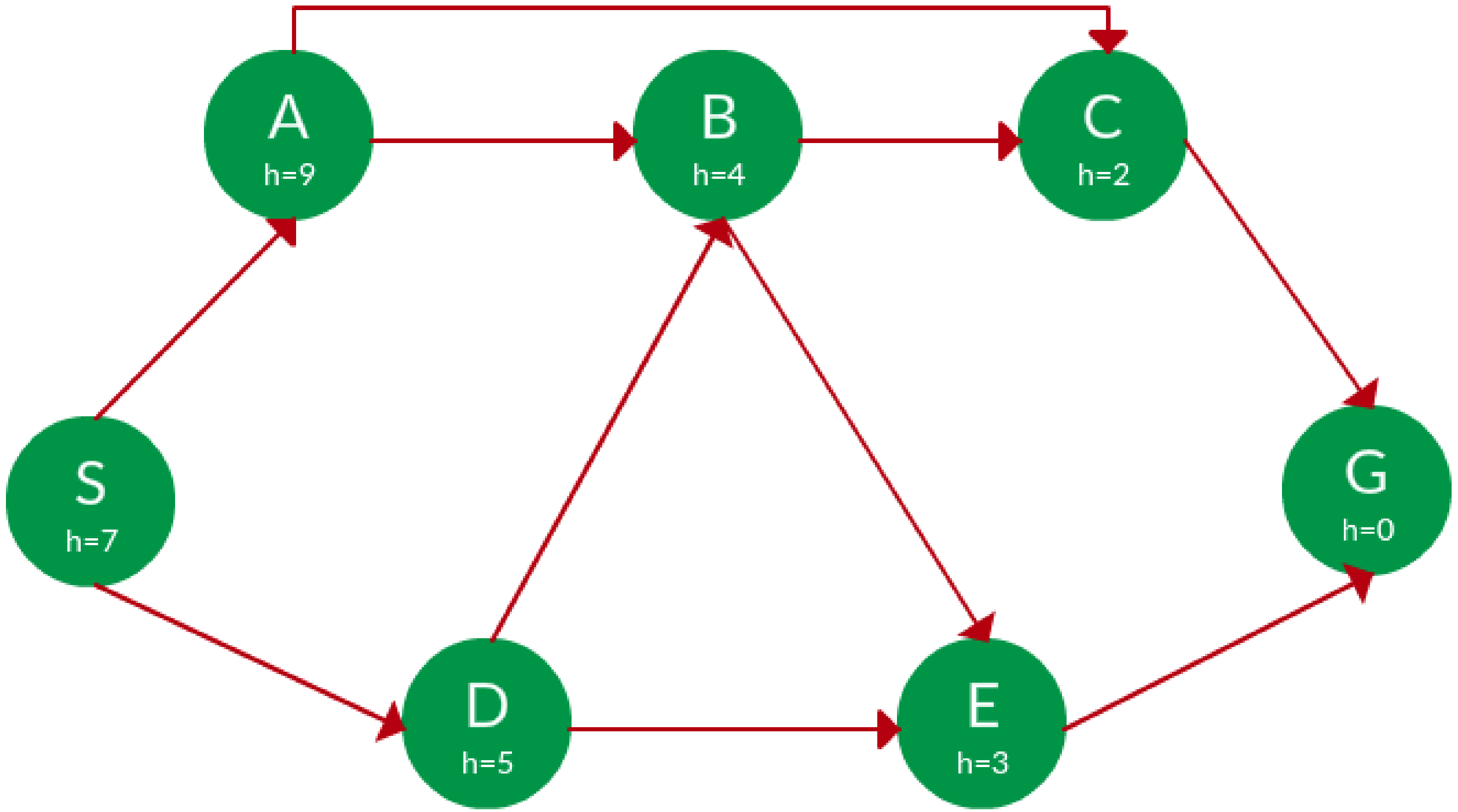
Solution

Step	F (priority Queue)	Visited	Remarks
1	(9,S)	S	
2	(2,SA) , (3,SB)	S, A, B	*remove S, S is not goal, add its successors*
3	(3,SB), (1,SAC) , (4,SAD)	S, A, B, C, D	*Remove A, A not goal, add its successors*
4	(3,SB) , (4, SAD)	S, A, B, C, D	*Remove C, C not goal, no successors of C*
5	(4, SAD) ,(0,SBG*)	S, A, B, C, D, G	*Remove B, B not goal, add successors, ~~SBD~~ as D is already visited*
6	(4, SAD)		*Remove G, goal state found*

***** 4 is estimated cheapest cost, actual cost of this path is 8.

Note: Optimal solution is SBDG with actual path cost of 6.

Homework : Greedy Algorithm



Performance Measures

1. Completeness

- Greedy Search is **incomplete**

Performance Measures

2. Optimality

- **No** – does not guarantee to provide optimal solution

Performance Measures

3. Time Complexity

- Like BFS & DFS, it may need to expand all the nodes
- Time Complexity : $\mathbf{O\ (b^{d+1})}$
- It also needs some time to look for the highest priority node in Q.

Performance Measures

4. Space Complexity

- Same as BFS
- Space Complexity : **$O\ (b^{d+1})$**

Admissibility of Heuristics

- A ***heuristic*** that never overestimates the cost to reach the goal is called admissible.
- **Admissible** heuristics are by nature **optimistic**, because;
- Admissible heuristics think **the cost of solving the problem is less than it actually is.**

Admissible Heuristic

- $h(n) \leq h^*(n)$
- $h(n)$ → estimated cost to reach the goal from n
- $h^*(n)$ → true cost to reach the goal from n
- Hence;
- $h(g) = 0$, for any goal node, g
- $h(n) = \infty$ if there is not path from n to a goal node

Admissible Heuristics – Some Examples

5	3	8
	2	6
7	4	1

Start

1	2	3
8		4
7	6	5

Goal

1. number of misplaced tiles

At least 1 move would be needed to move a misplaced tile towards the goal state – no overestimation - admissible

Admissible Heuristics – Some Examples

5	3	8
	2	6
7	4	1

Start

1	2	3
8		4
7	6	5

Goal

1. number of misplaced tiles

h = 7

At least 1 move would be needed to move a misplaced tile towards the goal state – no overestimation - admissible

Admissible Heuristics – Some Examples

5	3	8
	2	6
7	4	1

Start

1	2	3
8		4
7	6	5

Goal

2. Sum of **Manhattan Distances** of all misplaced tiles

h =

h =

Taxicab/ City-block /rectilinear distance b/w 2 points (x1 , y1) & (x2, y2) is **|x2 – x1| +|y2 – y2|**

Admissible Heuristics – Some Examples

5	3	8
	2	6
7	4	1

Start

1	2	3
8		4
7	6	5

Goal

2. Sum of **Manhattan Distances** of all misplaced tiles

h = **4 + 1 + 1 + 2 + 4 + 2 + 0 + 3**

h = **17**

Note: It can serve as a better estimate than simply number of misplaced tiles

Taxicab/ City-block /rectilinear distance b/w 2 points (x1 , y1) & (x2, y2) is **|x2 – x1| +|y2 – y2|**

Admissible Heuristics – Some Examples

5	3	8
	2	6
7	4	1

Start

1	2	3
8		4
7	6	5

Goal

3. Sum of permutation inversions

Explore yourself

A* Search

g(n) : actual path cost from the ***start node*** to node ***n***

h(n) : estimated cost of the cheapest path from ***n*** to the ***goal***

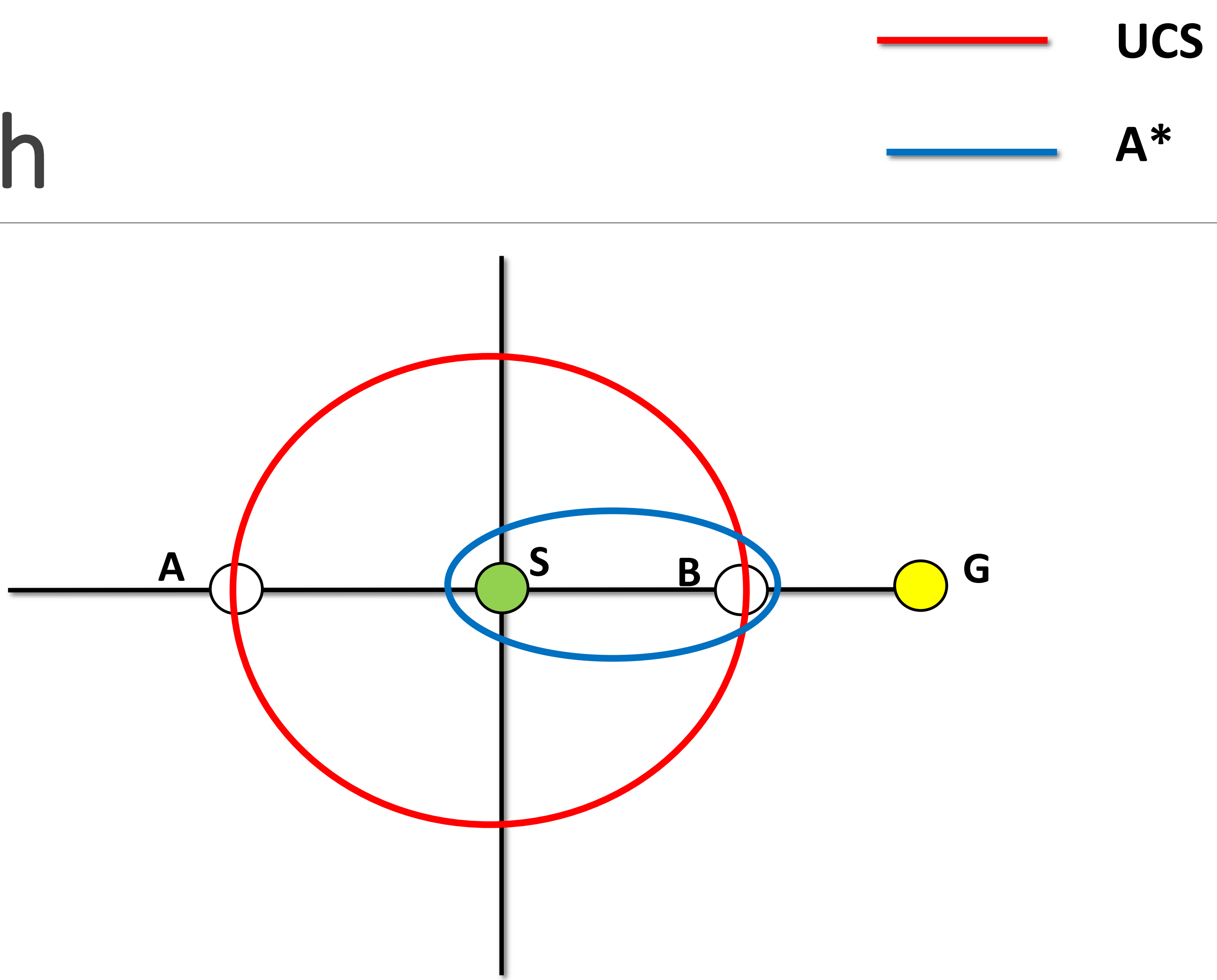
A* Search (A star)

- The most widely-known form of best-first search in AI
- UCS finds shortest path to every other node rather than focusing on the goal node
- A* uses heuristic to enumerates path length

f(n) = **g(n)** + **h(n)** where;

- **f(n)** : estimated cost of the cheapest solution through n (**estimated total path length**)
- In fact it is the ***best estimate*** of the **total** distance to the goal
- **Implementation:** Maintain priority queue and pull off node with ***least*** **f(n)**

A* Search

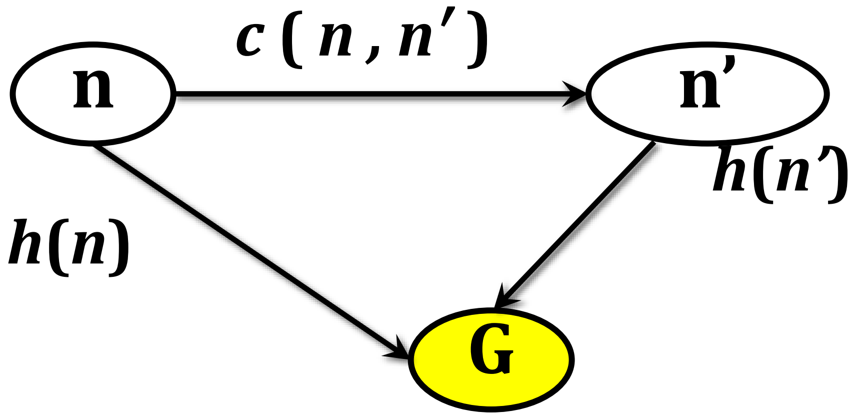


A* with Strict Expanded List/Closed List

- Use of Closed list saves us redundant effort of expanding longer, non-optimal paths.
- However, when Closed list is used with A*, only admissibility of heuristics does not guarantee its optimality
- Consistency of heuristic is also essential for optimality

Consistent Heuristic

- h is consistent if the heuristic function satisfies triangle inequality for every node & its child node n':
- H(n) <= h(n') + c(n, n')



- When h is consistent, the f values of nodes expanded by A* are never decreasing
- When A* selected n for expansion, it already found the shortest path to it
- When h is consistent, every node is expanded once.

Consistency is sometimes also called monotonicity

Consistent (Monotone) Heuristic

If h is consistent, we have

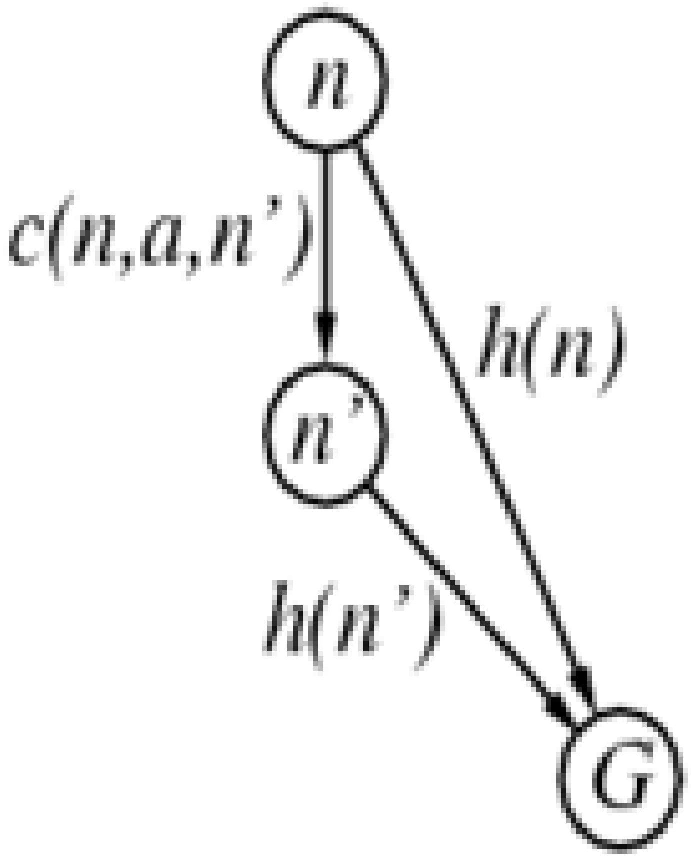
$f(n') = g(n') + h(n')$

$= g(n) + c(n, a, n') + h(n')$

$\geq g(n) + h(n)$

$\geq f(n)$

i.e. f(n) is non- decreasing along any path



A* with Strict Expanded List/Closed List

1. **Initialize** F ← (S) & set **Expanded = ( )**.

2. **If** F is empty, fail. **Else**, **apply** goal test to **least cost** search node N in F.

3. **If** state(N) == goal, **return** N. **Else** **remove** N from F.

4. **If** state(N) is in Expanded, **go to** step 2. **Else**, **add** state(N) to Expanded.

5. **Add** all successors of N, not in Expanded, to F **discarding** any longer path to the same state.

6. **Go to** step 2.

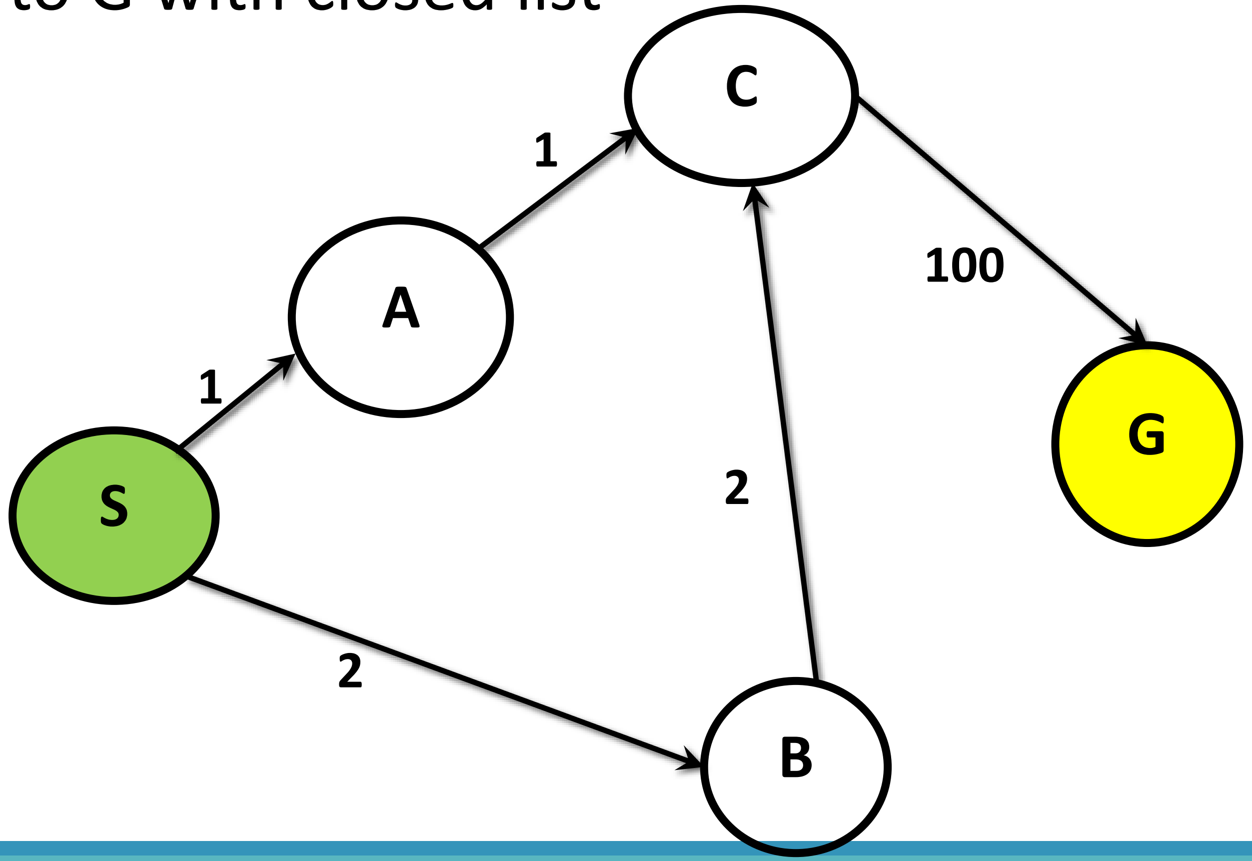
Least cost → having least total path length f(n) = g(n) + h(n)

Example- A*

(a) Simulate A* to find path from S to G with closed list

(b) Repeat without closed list

S	A	B	C	G
90	100	88	100	0

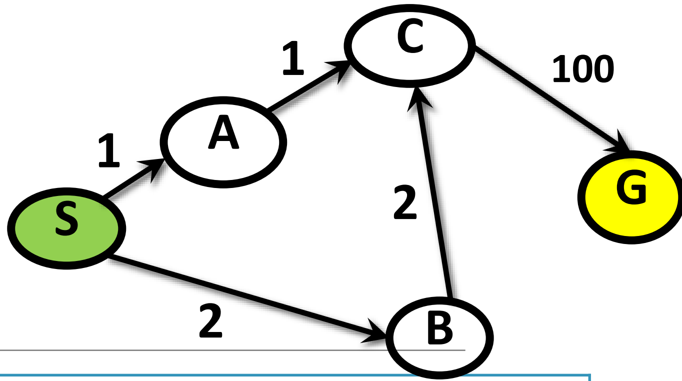


(a) With Closed List

Step	F (priority Queue)	Expanded	Remarks

(a) With Closed List

S	A	B	C	G
90	100	88	100	0



Step	F (priority Queue)	Expanded	Remarks
1	(90 , S)		
2	(101 , SA) , (90 , SB)*	S	
3	(101 , SA)* , (104 , SBC)	S , B	
4	~~(104 , SBC)~~ , (102, SAC)	S , A , B	longer path discarded
5	(102 , SACG)*	S , A , B , C	
			goal node found

Path : **(102 , SACG)**

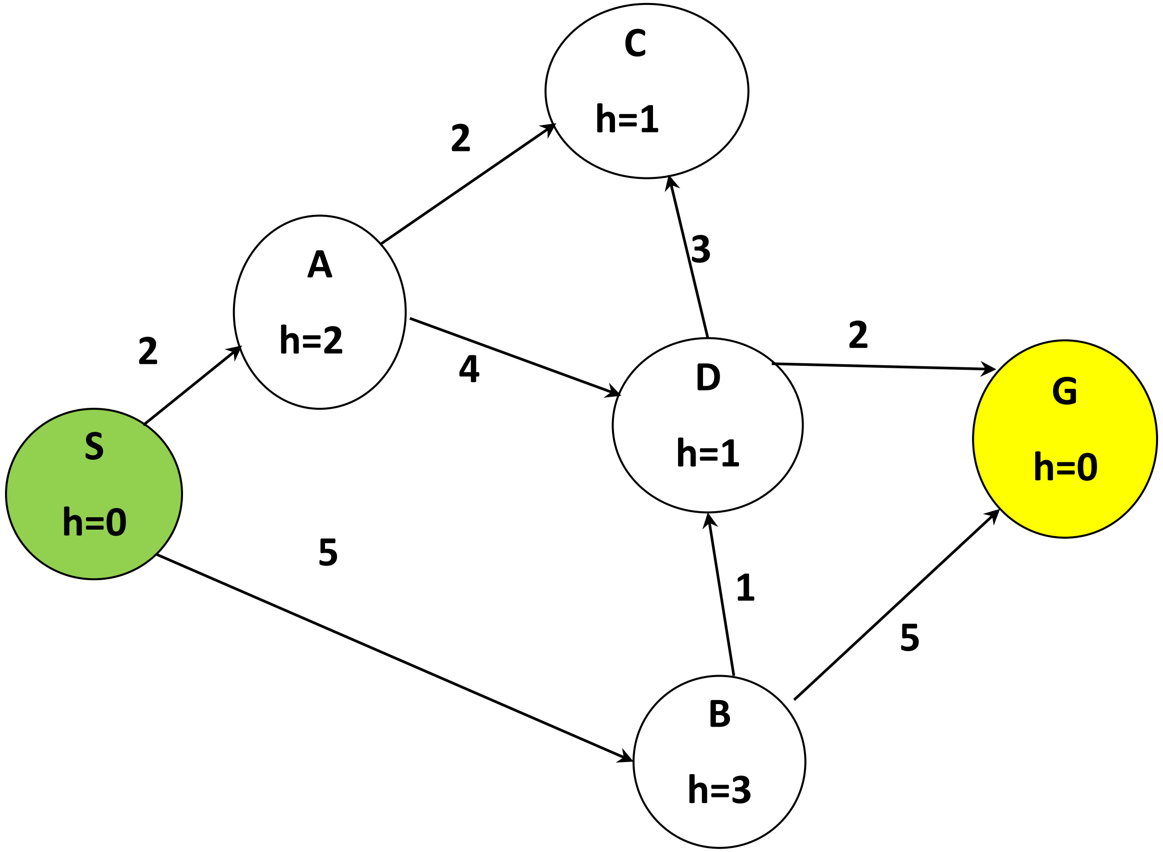
(b) Without Closed List

Step	F (priority Queue)	Remarks

Example- A*

(a) Simulate A* to find path from S to G without closed list

(b) Repeat with closed list



(a) Without Closed List

Step	F (priority Queue)	Remarks
1	(0, S)	
2	(4, SA), (8, SB)	
3	(8, SB), (5, SAC), (7, SAD)	
4	(8, SB), (7, SAD)	
5	(8, SB), (10, SADC), (8, SADG)	[**Note:** In case of tie, randomly break it]

(b) With Closed List

Step	F (priority Queue)	Expanded	Remarks
1	(0,S)	-	
2	(101, SA), (3, SB)	S	
3	(101, SA), (94, SBC)	S, B	
4	(101, SA), (104, SBCG)	S, B, C	
5	(104,SBCG)	S, B, C	C already expanded
6		S,B, C, G	

Homework Apply A* on given 8-puzzle

When;

1. h(n) = number of misplaced tiles

2. h(n) = sum of Manhattan distance

Step cost is 1

2	8	3		1	2	3
1	6	4		8		4
7		5		7	6	5
Initial State				Final State		

1. Completeness

- Complete if costs > 0 , above epsilon & branching factor is finite
- Even if h are not admissible, it is able to terminate with a solution path (though not necessarily the optimal one)
- **Proof:**
- The evaluation function f of nodes expanded must increase eventually (since paths are longer and more costly) until all the nodes on a solution path are expanded
- Note: A* is admissible if it uses admissible heuristics.

2. Optimality

- **Tree Search**
- - Admissible heuristic is needed
- **Graph Search with reopening closed nodes**
- Admissible heuristic is enough
- **Graph Search without reopening closed nodes**
- Consistent heuristic is needed

3. Space Complexity

- Exponential
- The space complexity of A∗ often makes it impractical to insist on finding an optimal solution.
- One can use variants of A∗ that find suboptimal solutions quickly, or
- one can sometimes design heuristics that are more accurate but not strictly admissible.
- In any case, the use of a good heuristic still provides enormous savings compared to the use of an uninformed search.

4. Time Complexity

- Exponential unless heuristic is very accurate.
- Computation time is not main drawback of A*
- Because it keeps all generated nodes in memory (as do all GRAPH-SEARCH algorithms),
- A∗ usually runs out of space long before it runs out of time.
- For this reason, A∗ is not practical for many large-scale problems.
- Recently developed algorithms have overcome the space problem without sacrificing optimality or completeness, at a small cost in execution time.

Explore yourself

- Simple Hill Climbing
- Steepest Ascent Hill Climbing

The End