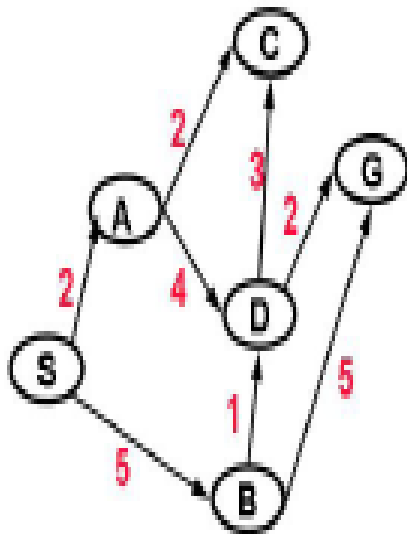
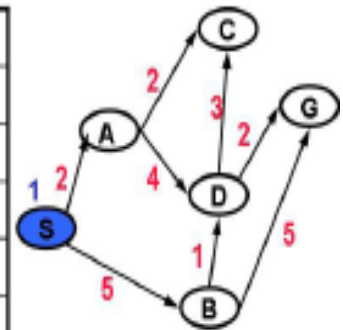


Uniform Search: Example



Pick best (by path length) element of Q; Add path extensions anywhere in Q

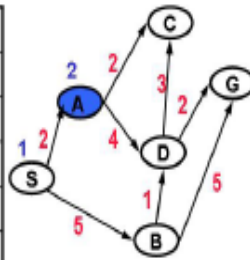
	Q
1	<u>(0 S)</u>



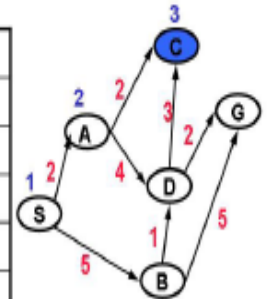
Added paths in **blue**; underlined paths are chosen for extension.
We show the paths in **reversed** order; the node's state is the first entry.

Pick best (by path length) element of Q; Add path extensions anywhere in Q

	Q
1	(0 S)
2	<u>(2 A S)</u> (5 B S)



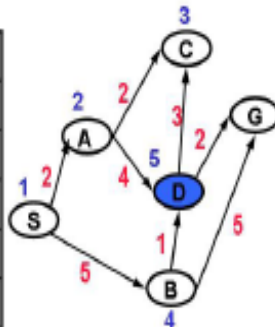
	Q
1	(0 S)
2	<u>(2 A S)</u> (5 B S)
3	<u>(4 C A S)</u> (6 D A S) (5 B S)



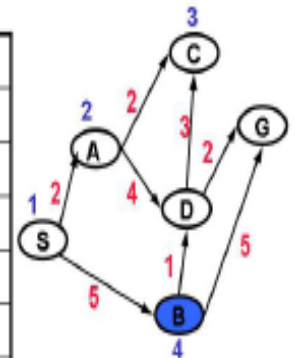
Added paths in blue; underlined paths are chosen for extension.
We show the paths in reversed order; the node's state is the first entry.



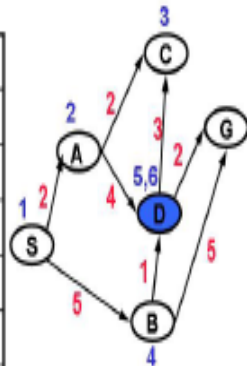
	Q
1	(0 S)
2	<u>(2 A S)</u> (5 B S)
3	<u>(4 C A S)</u> (6 D A S) (5 B S)
4	(6 D A S) <u>(5 B S)</u>
5	<u>(6 D B S)</u> (10 G B S) (6 D A S)



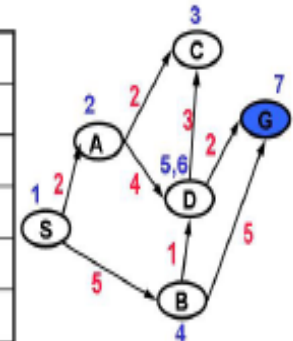
	Q
1	(0 S)
2	<u>(2 A S)</u> (5 B S)
3	<u>(4 C A S)</u> (6 D A S) (5 B S)
4	(6 D A S) <u>(5 B S)</u>



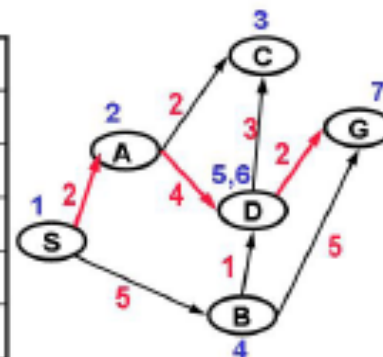
	Q
1	(0 S)
2	(2 A S) (5 B S)
3	(4 C A S) (6 D A S) (5 B S)
4	(6 D A S) (5 B S)
5	(6 D B S) (10 G B S) (6 D A S)
6	(8 G D B S) (9 C D B S) (10 G B S) (6 D A S)



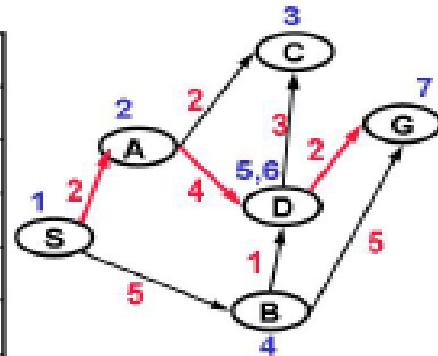
	Q
1	(0 S)
2	(2 A S) (5 B S)
3	(4 C A S) (6 D A S) (5 B S)
4	(6 D A S) (5 B S)
5	(6 D B S) (10 G B S) (6 D A S)
6	(8 G D B S) (9 C D B S) (10 G B S) (6 D A S)
7	(8 G D A S) (9 C D A S) (8 G D B S) (9 C D B S) (10 G B S)



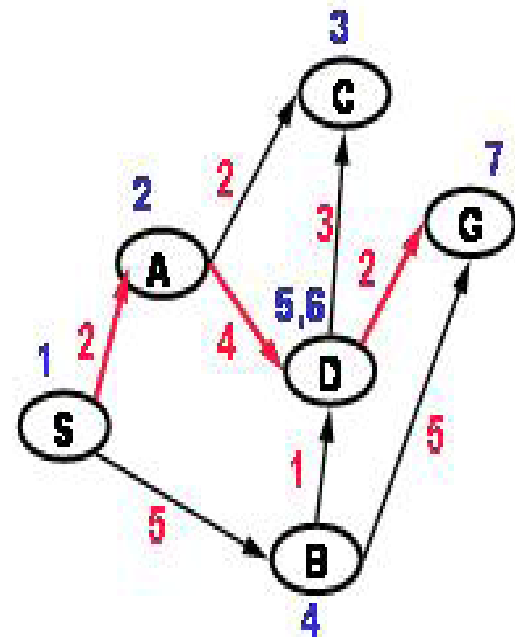
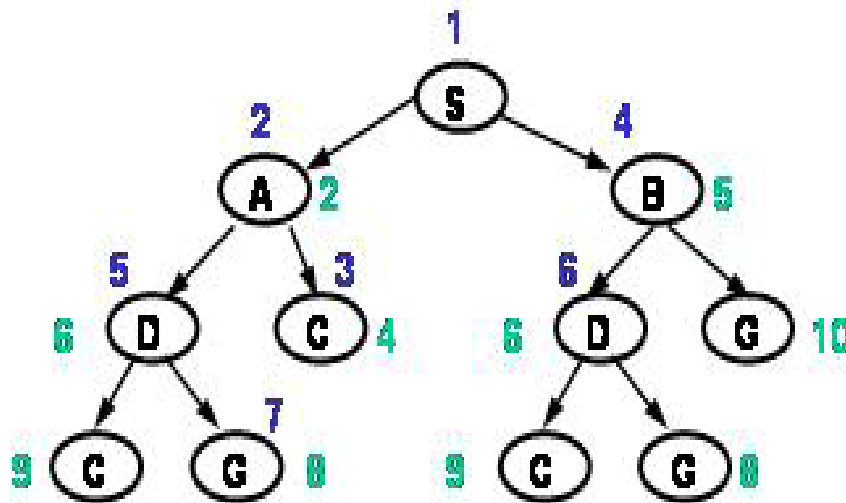
	Q
1	(0 S)
2	(2 A S) (5 B S)
3	(4 C A S) (6 D A S) (5 B S)
4	(6 D A S) (5 B S)
5	(6 D B S) (10 G B S) (6 D A S)
6	(8 G D B S) (9 C D B S) (10 G B S) (6 D A S)
7	(8 G D A S) (9 C D A S) (8 G D B S) (9 C D B S) (10 G B S)



	Q
1	(0 S)
2	(2 A S) (5 B S)
3	(4 C A S) (6 D A S) (5 B S)
4	(6 D A S) (5 B S)
5	(6 D B S) (10 G B S) (6 D A S)
6	(8 G D B S) (9 C D B S) (10 G B S) (6 D A S)
7	(8 G D A S) (9 C D A S) (8 G D B S) (9 C D B S) (10 G B S)

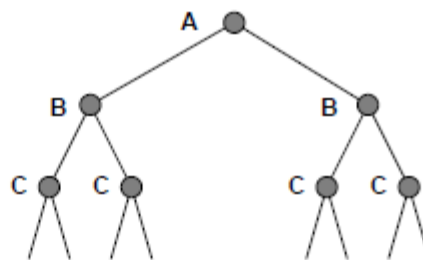
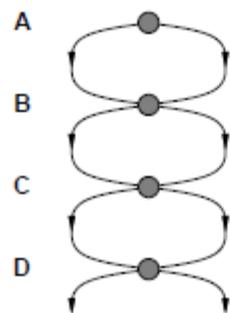


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100



Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!



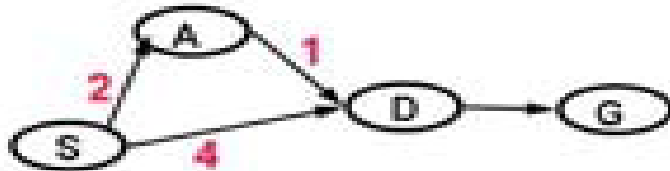
Use a Data Structure Visited as well as the usual Q

1. Initialize Q with start node S. Visited = {S}
2. If Q is empty, fail. Else pick a node N from Q //start of loop
3. If state(N) is Goal then exit loop with solution
4. Otherwise find all successors of state(N) not in Visited and add to Q and update the paths

Pros: Avoids repeated expansion of subtrees

No infinite paths because of loops

Cons: But does not give optimal solution. Example below:



Graph Search Schema

Maintain a new Data Structure Closed in addition to Q.
Closed is also referred to as Expanded

function GRAPH-SEARCH(*problem*, *fringe*) **returns** a solution, or failure

closed ← an empty set

fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)

loop do

if *fringe* is empty **then return** failure

node ← REMOVE-FRONT(*fringe*)

if GOAL-TEST(*problem*, STATE[*node*]) **then return** *node*

if STATE[*node*] is not in *closed* **then**

 add STATE[*node*] to *closed*

fringe ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

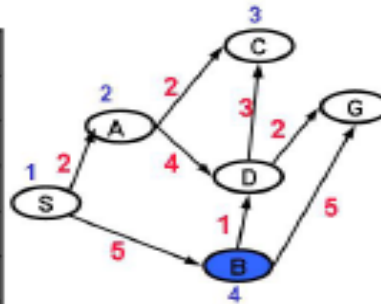
end

For optimal solution we should only keep least cost paths in Q.

Optimization – Only insert into fringe only those successors not in closed list

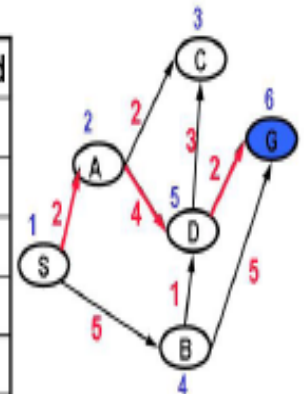
Implications of Graph Search

	Q	Expanded
1	(0 S)	
2	(2 A S) (5 B S)	S
3	(4 C A S) (6 D A S) (5 B S)	S, A
4	(6 D A S) (5 B S)	S, A, C



Uniform cost search

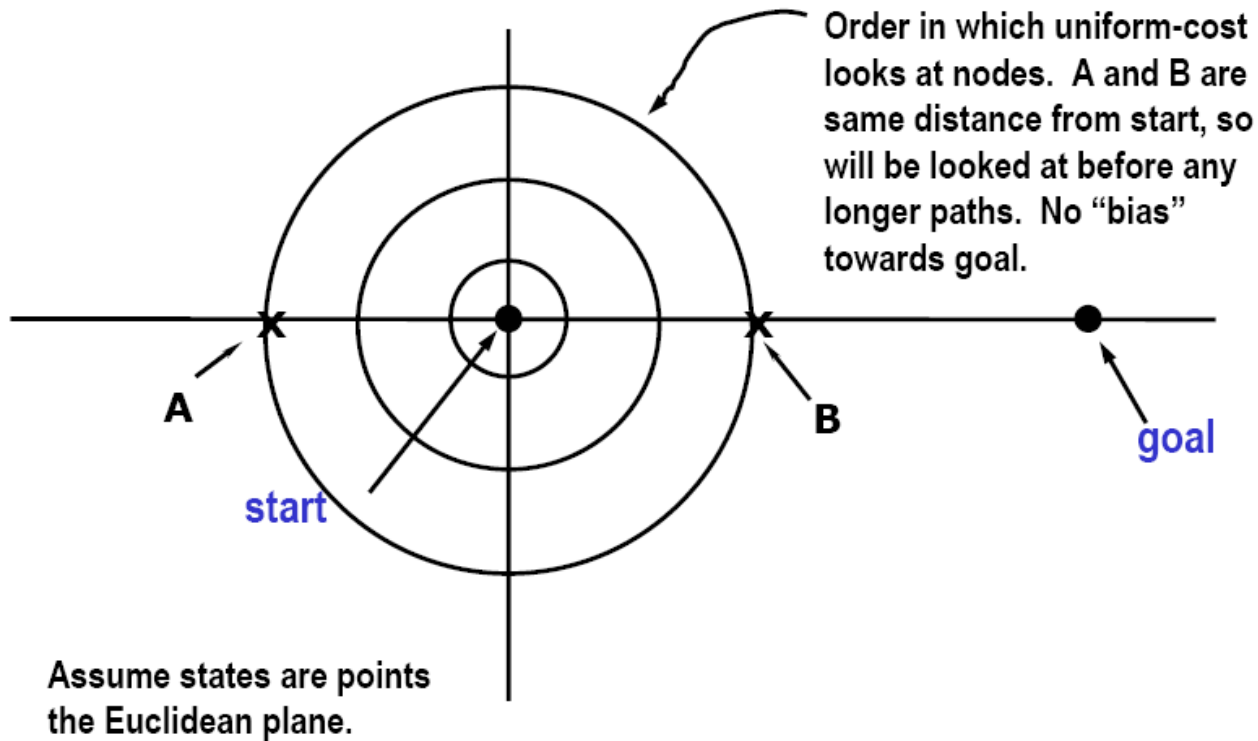
	Q	Expanded
1	(0 S)	
2	(2 A S) (5 B S)	S
3	(4 C A S) (6 D A S) (5 B S)	S, A
4	(6 D A S) (5 B S)	S, A, C
5	(6 D B S) (10 G B S) (6 D A S)	S, A, C, B
6	(8 G D A S) (9 C D A S) (10 G B S)	S, A, C, B, D



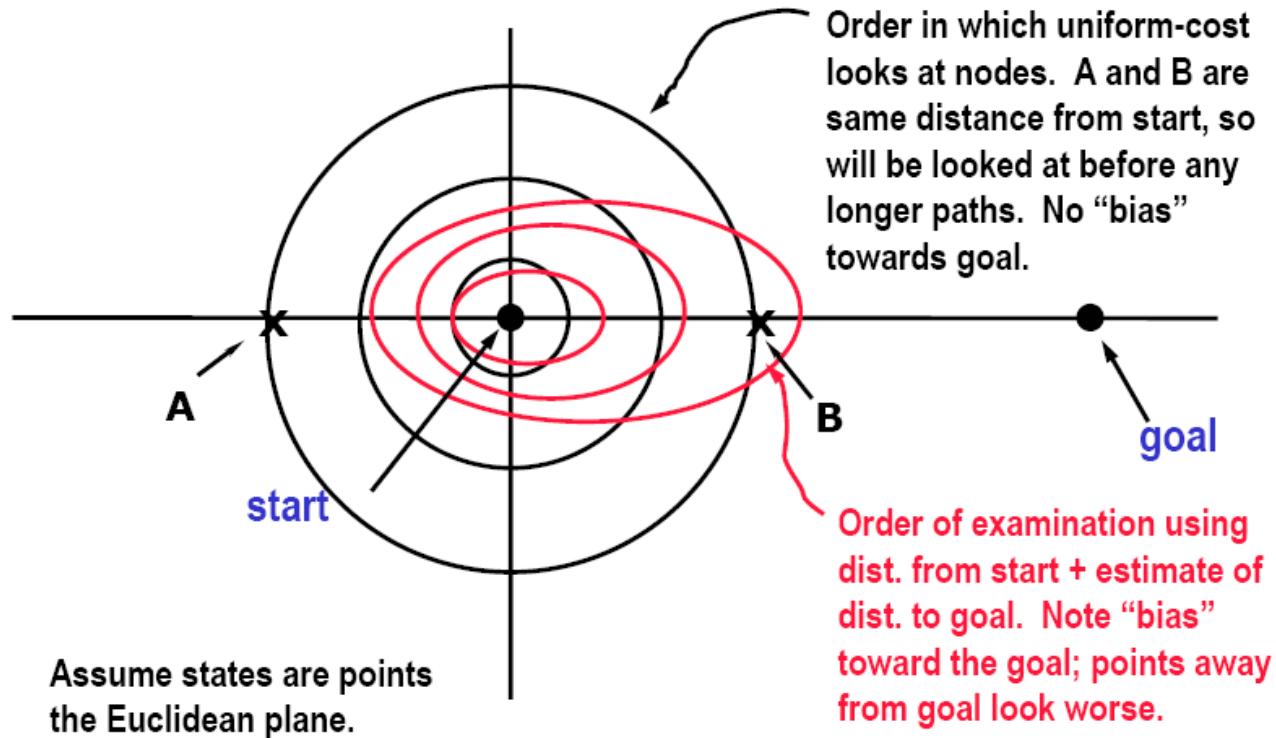
Uninformed vs. Informed Search

- Depth-first, breadth-first and uniform-cost searches are **uninformed**.
- In **informed** search there is an estimate available of the cost (distance) from each state (city) to the goal.
- This estimate (**heuristic**) can help you head in the right direction.
- Heuristic embodied in function $h(n)$, estimate of remaining cost from search node n to the least cost goal.
- Graph being searched is a graph of states. Search algorithm defines a tree of search nodes. Two paths to the same state generate two different search nodes.
- Heuristic could be defined on underlying state; the path to a state does not affect estimate of distance to the goal.

Why use estimate of goal distance?



Why use estimate of goal distance?

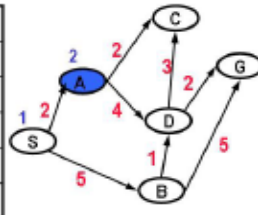


A* Search: Example

A*

Pick best (by path length+heuristic) element of Q; Add path extensions anywhere in Q

	Q
1	(0 S)
2	(4 A S) (8 B S)



Heuristic Values

A=2 C=1 S=0
B=3 D=1 G=0

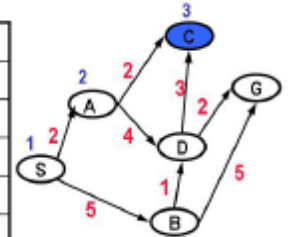
Added paths in blue; underlined paths are chosen for extension.

We show the paths in reversed order; the node's state is the first entry.

A*

Pick best (by path length+heuristic) element of Q; Add path extensions anywhere in Q

	Q
1	(0 S)
2	(4 A S) (8 B S)
3	(5 C A S) (7 D A S) (8 B S)

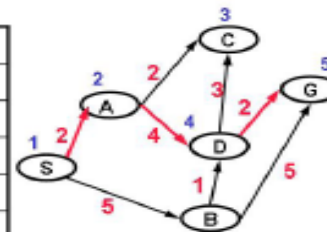


Heuristic Values

A=2 C=1 S=0
B=3 D=1 G=0

Pick best (by path length+heuristic) element of Q; Add path extensions anywhere in Q

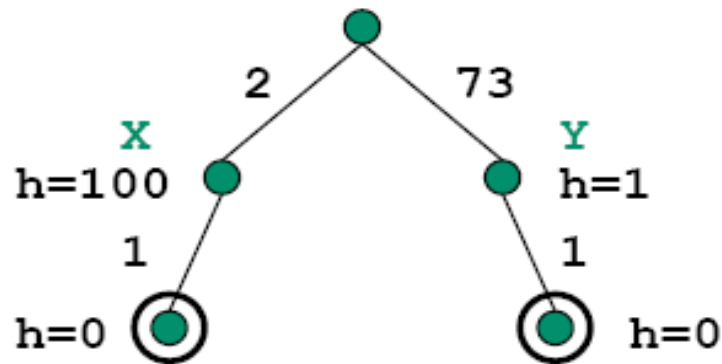
	Q
1	(0 S)
2	(4 A S) (8 B S)
3	(5 C A S) (7 D A S) (8 B S)
4	(7 D A S) (8 B S)
5	(8 G D A S) (10 C D A S) (8 B S)



Heuristic Values

A=2 C=1 S=0
B=3 D=1 G=0

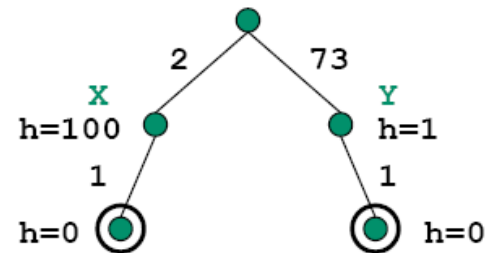
Issues in choosing $h(n)$ values



Running A* on the example will not give optimal solution!

Admissible Heuristic

- What must be true about h for A^* to find optimal path?
- A^* finds optimal path if h is admissible; h is **admissible** when it never overestimates.
- In this example, h is not admissible.

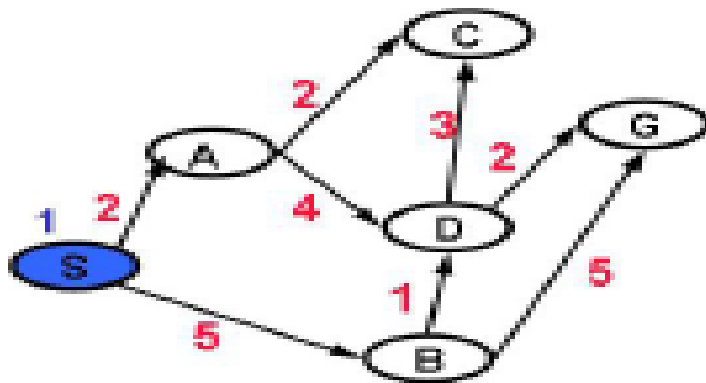


$$g(X) + h(X) = 102$$

$$g(Y) + h(Y) = 74$$

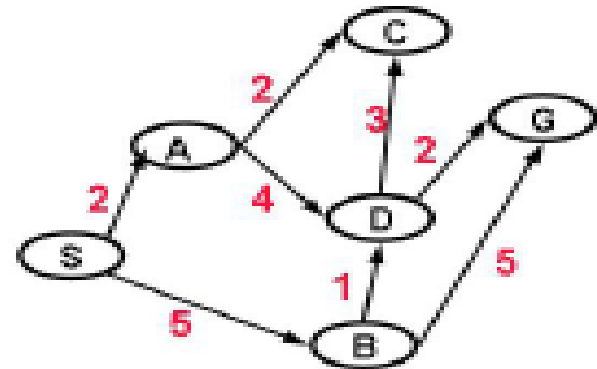
Optimal path is not found!

Are these admissible?



Heuristic Values

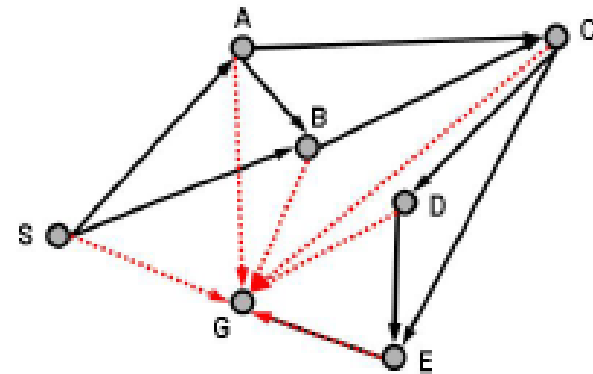
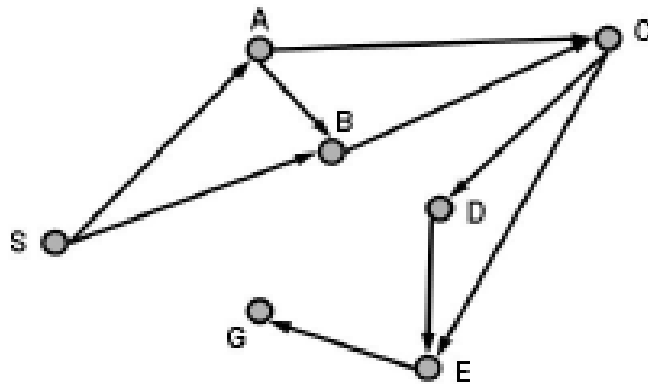
A=2	C=1	S=0
B=3	D=1	G=0



Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

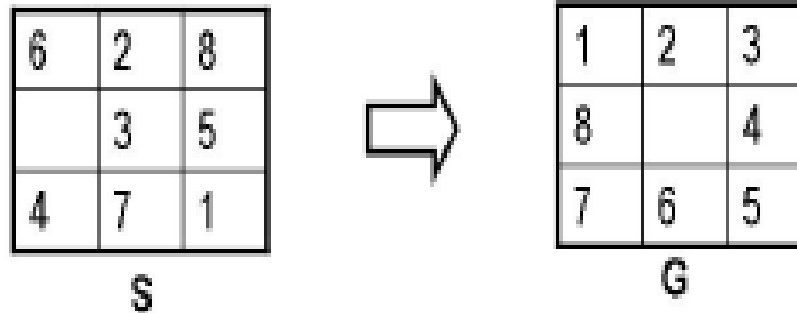
Admissible Heuristics: Straight-Line Estimate



We can use the straight-line distance (shown in red) as an underestimate of the actual driving distance between any city and the goal. The best possible distance between the cities cannot be better than the straight-line distance. But it can be worse.

Admissible Heuristics

8 Puzzle: Move tiles to reach goal. Think of a move as moving “empty” tile.

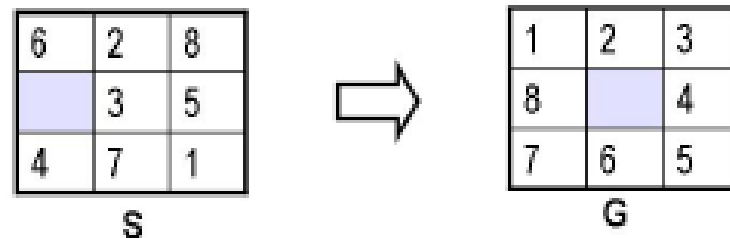


Alternative underestimates of “distance” (number of moves) to goal:

1. Number of misplaced tiles (7 in example above)

Admissible Heuristics

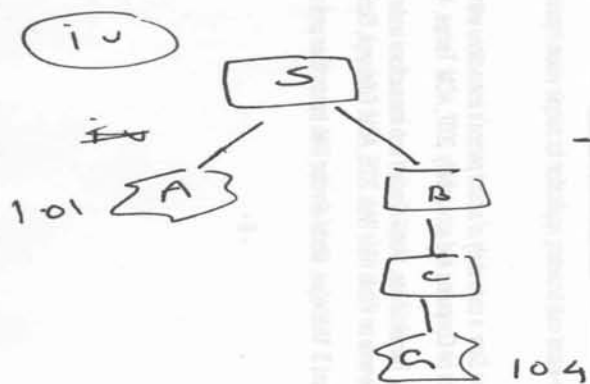
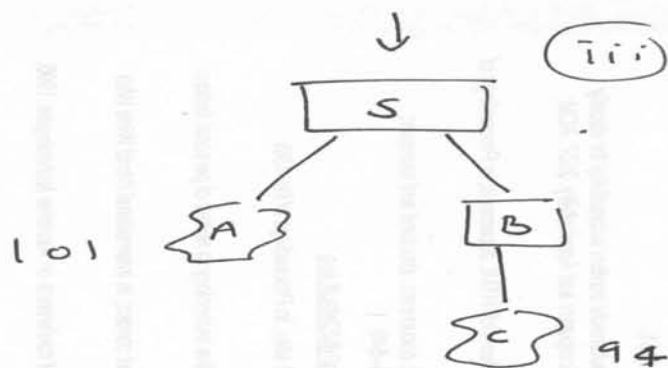
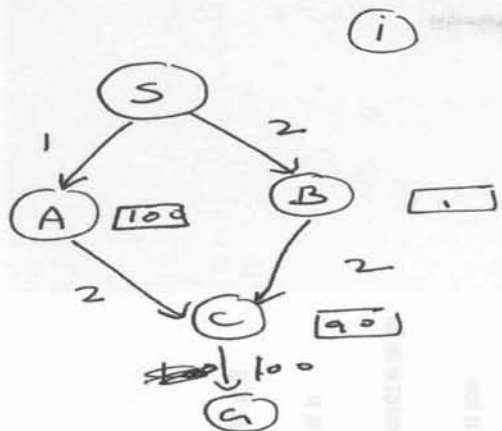
8 Puzzle: Move tiles to reach goal. Think of a move as moving “empty” tile.



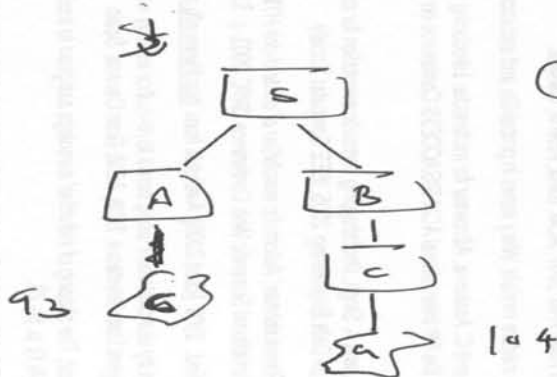
Alternative underestimates of “distance” (number of moves) to goal:

1. Number of misplaced tiles (7 in example above)
2. Sum of Manhattan distance of tile to its goal location (17 in example above). Manhattan distance between (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$. Each move can only decrease the distance of exactly one tile.

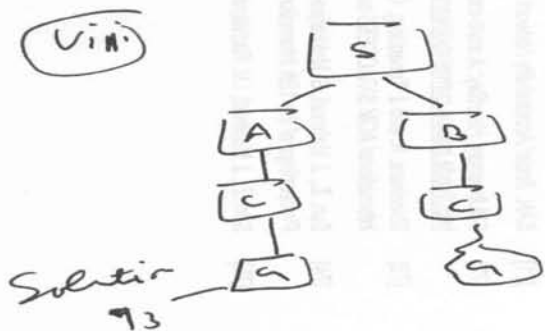
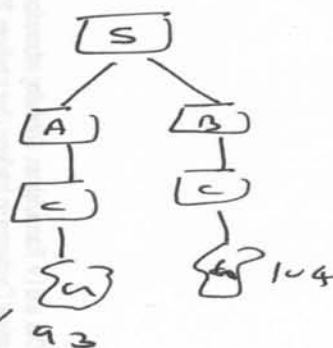
The second of these is much better at predicting actual number of moves.



(v)



(vi)



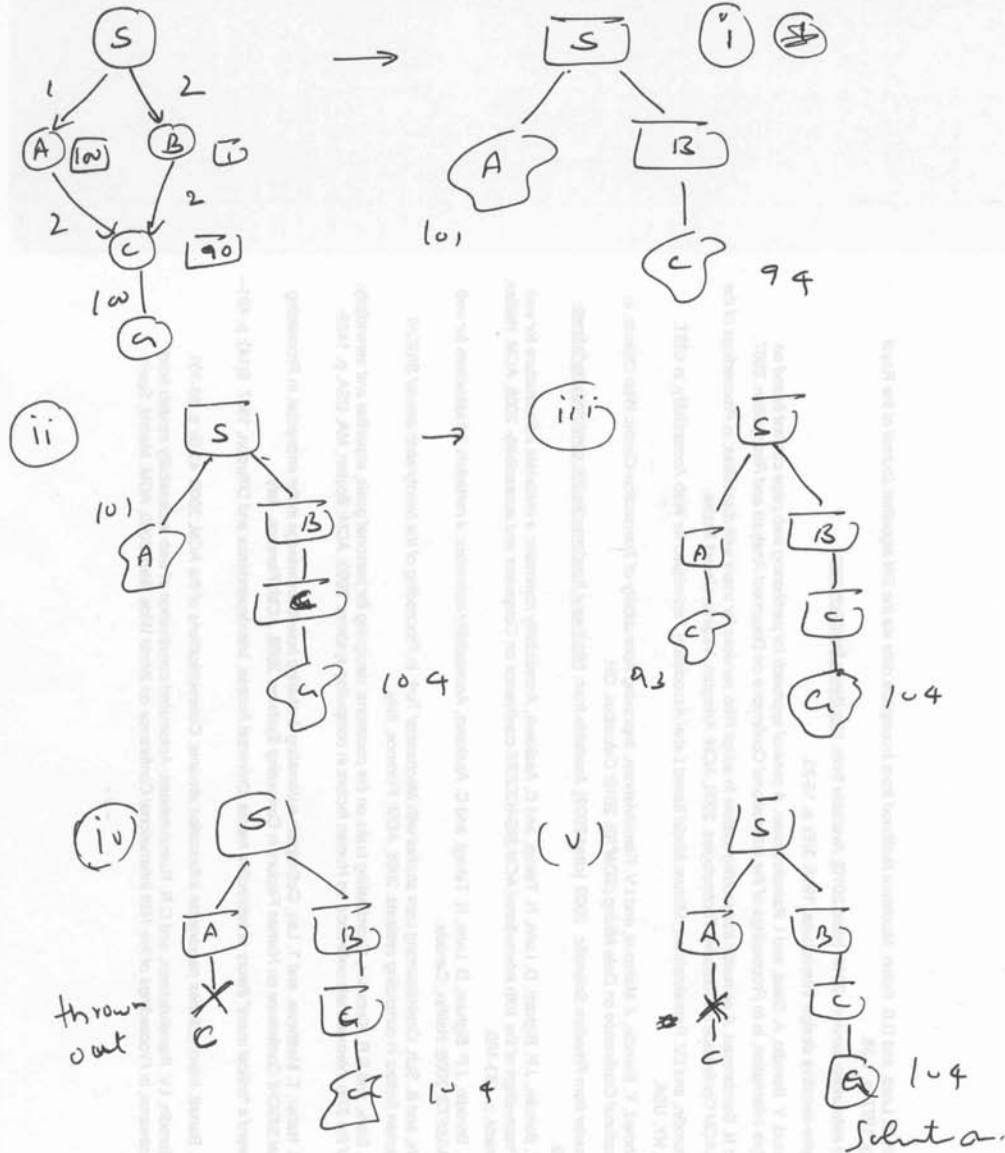
Tree Search

- Fringe node

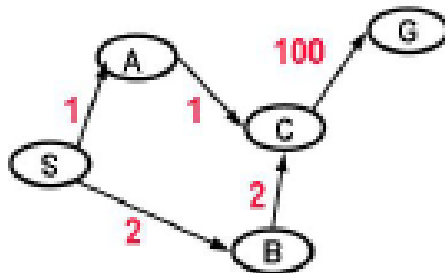
- Expanded node

Show Proof of Optimality of A^*
with Admissible Heuristic

A* Search – Graph Structure



Implications on A* Graph Search



Heuristic Values

A=100 C=90 S=0

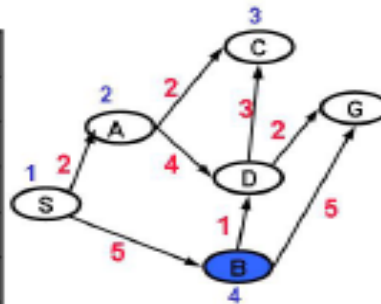
B=1 G=0

	Q	Expanded
1	<u>(0 S)</u>	
2	<u>(3 B S)</u> (101 A S)	S
3	<u>(94 C B S)</u> (101 A S)	B, S
4	<u>(101 A S)</u> (104 G C B S)	C, B, S
5	<u>(104 G C B S)</u>	A, C, B, S

Why: Because $f(n)$ does not monotonically increase on a path

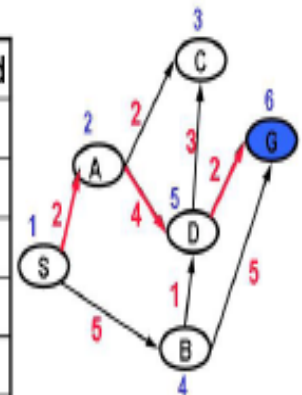
A* - Graph Search

	Q	Expanded
1	(0 S)	
2	<u>(2 A S)</u> (5 B S)	S
3	<u>(4 C A S)</u> (6 D A S) (5 B S)	S,A
4	(6 D A S) <u>(5 B S)</u>	S,A,C



Uniform cost search

	Q	Expanded
1	(0 S)	
2	<u>(2 A S)</u> (5 B S)	S
3	<u>(4 C A S)</u> (6 D A S) (5 B S)	S,A
4	(6 D A S) <u>(5 B S)</u>	S,A,C
5	(6 D B S) (10 G B S) (6 D A S)	S,A,C,B
6	<u>(8 G D A S)</u> (9 C D A S) (10 G B S)	S,A,C,B,D



Consistency Condition

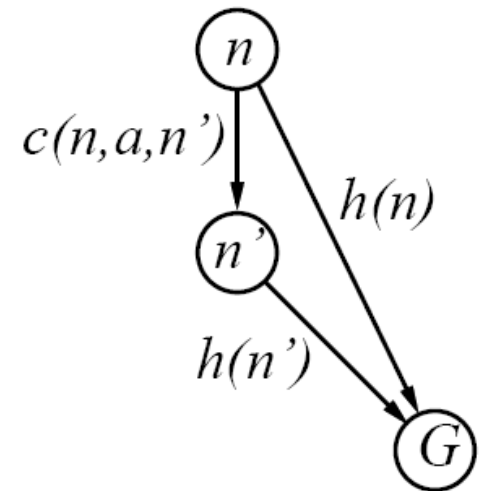
A heuristic is **consistent** if

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

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

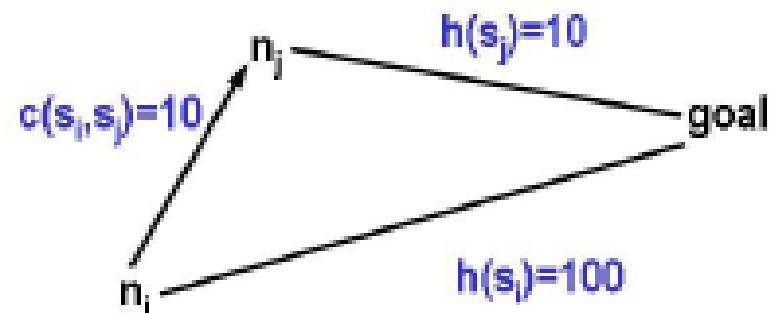
I.e., $f(n)$ is nondecreasing along any path.

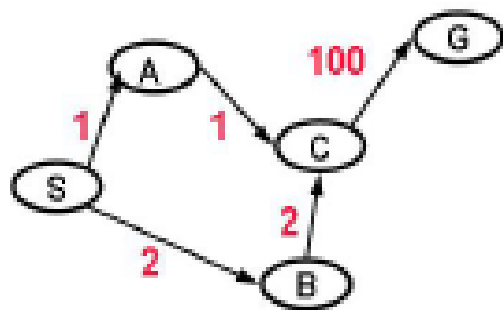


Consistent Heuristic

Consistency Violation

- A simple example of a violation of consistency.
- $h(s_i) - h(s_j) \cdot c(s_i, s_j)$
- In example, $100 - 10 > 10$
- If you believe goal is 100 units from n_i , then moving 10 units to n_j should not bring you to a distance of 10 units from the goal.

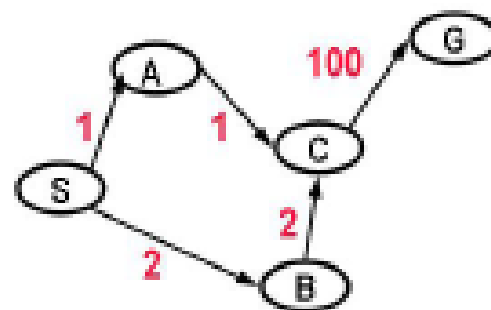




Heuristic Values

A=100	C=90	S=0
B=1		G=0

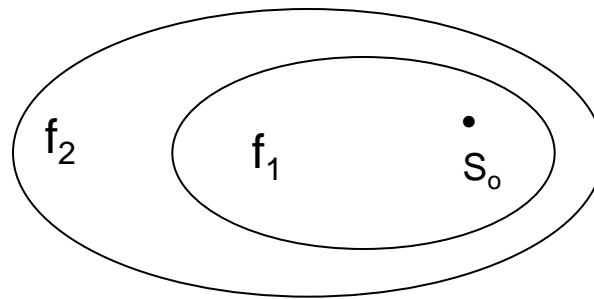
Not Consistent



Heuristic Values

A=100	C=100	S=90
B=88		G=0


Consistent

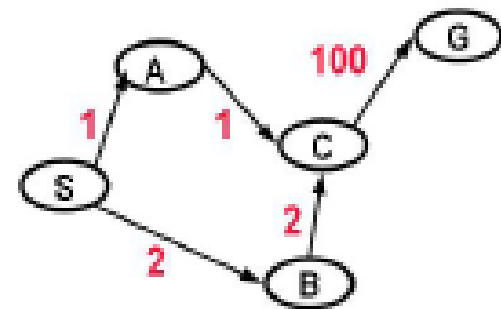


f_1 values $<$ f_2 values

Nodes expanded in the order of increasing f values

A* Graph Search with Consistent Heuristic

	Q	Expanded
1	(90 S)	
2	(90 B S) (101 A S)	S
3	(101 A S) (104 C B S)	A, S 
4	(102 C A S) (104 C B S)	C, A, S
5	(102 G C A S)	G, C, A, S



Heuristic Values

A=100 C=100 S=90

B=88

G=0

Comparing Heuristics

- $h_1(n)$ and $h_2(n)$ are 2 heuristics
 - Let h^* be the “perfect” heuristic – returns actual path cost to goal.
 - If $h_1(N) < h_2(N) \leq h^*(N)$ for all non-goal nodes, then h_2 is a better heuristic than h_1

Implications of better or more informed heuristic

- If A_1^* uses h_1 , and A_2^* uses h_2 , then every node expanded by A_2^* is also expanded by A_1^*