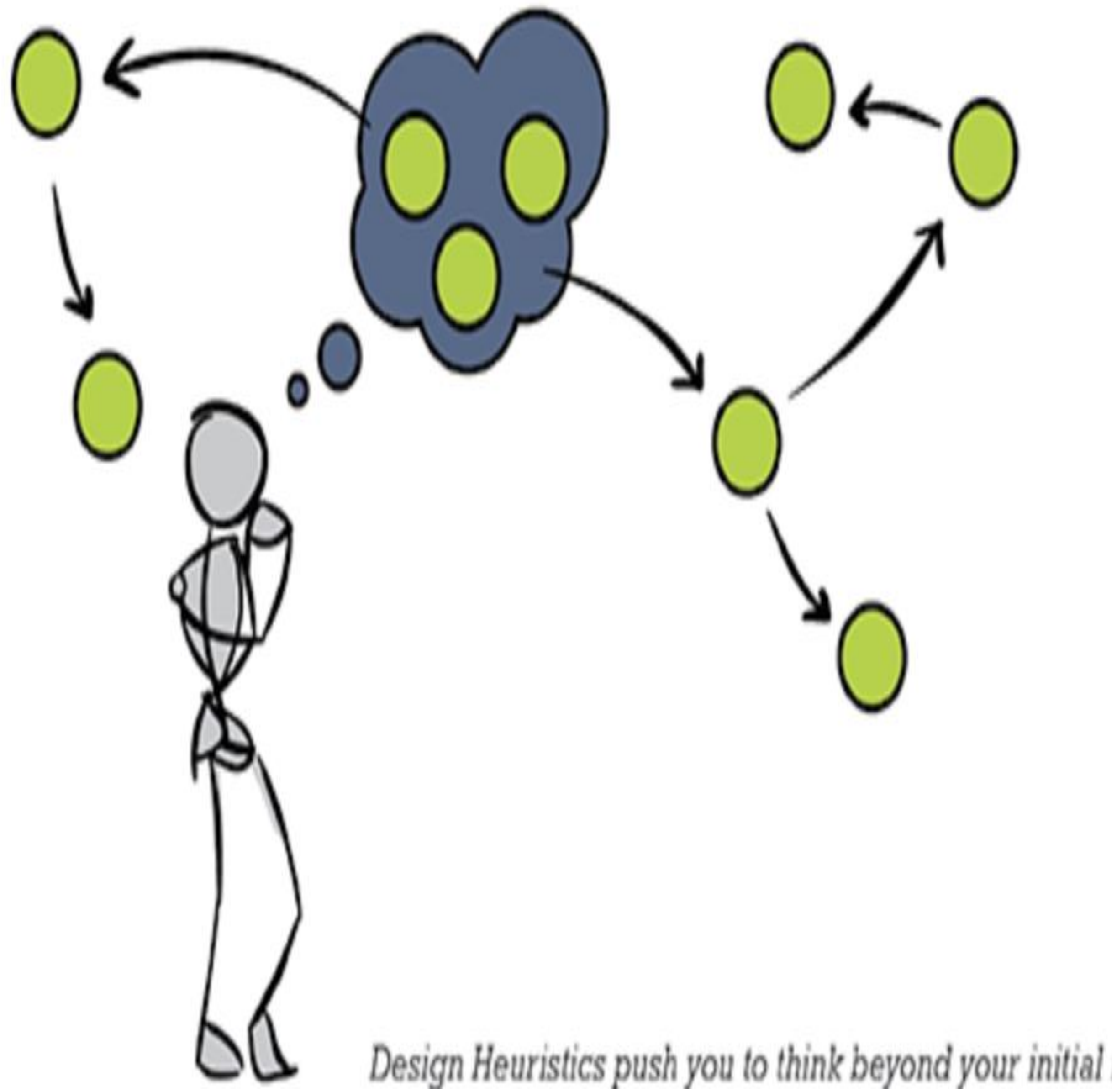
INFORMED SEARCH STRATEGIES



BEST-FIRST SEARCH

Best-first search is TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an evaluation function, **f (n)**

A whole ***family algorithms*** with different evaluation functions

Best-first search can be implemented using a ***priority queue***

- Order nodes on the nodes list by increasing value of an evaluation function, f(n).
- The "best" node according to the evaluation function is expanded.
- F(n) incorporates ***domain-specific information*** in the form of h(n) in some way

A key component of these algorithms is a heuristic function h(n):

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

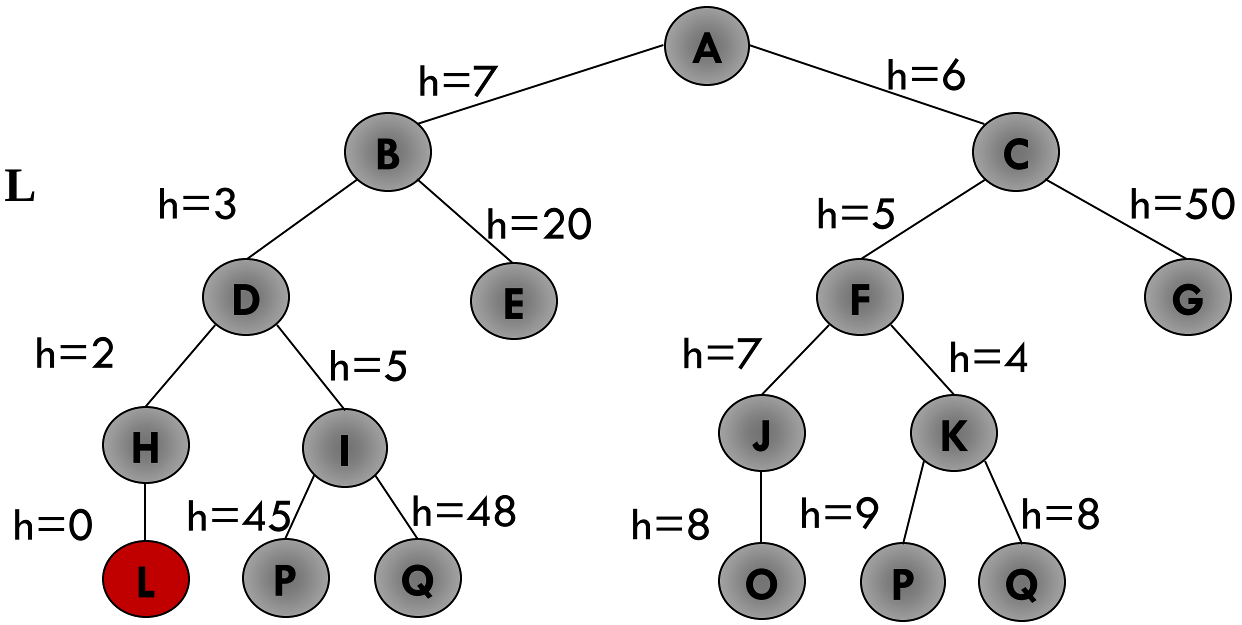
GREEDY SEARCH

Use as an evaluation function **f(n) = h(n)**.

Selects node to expand that are believed to be closest to a goal node (hence "greedy")

- Select node with smallest f value

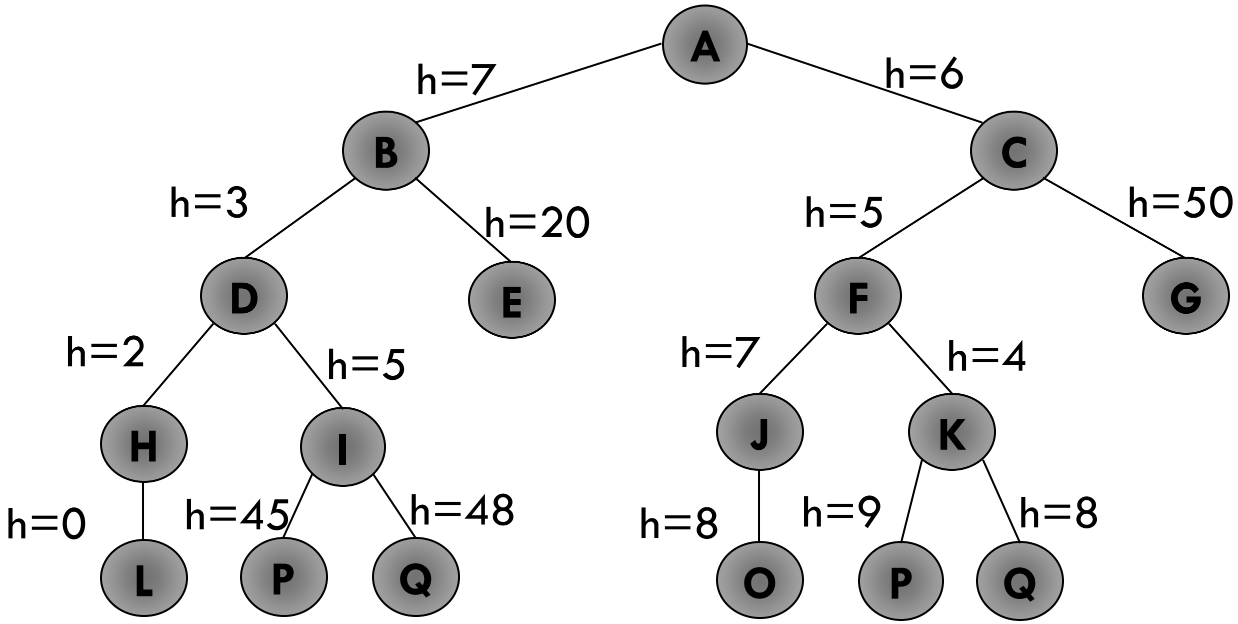
Search for L



GREEDY SEARCH

Best First Search

- A,C,F,K,J,B,D,H ➡ L
- A,C,F,K,B,D,H ➡ L



PROPERTIES OF GREEDY BEST-FIRST SEARCH

Complete: No – can get stuck in loops

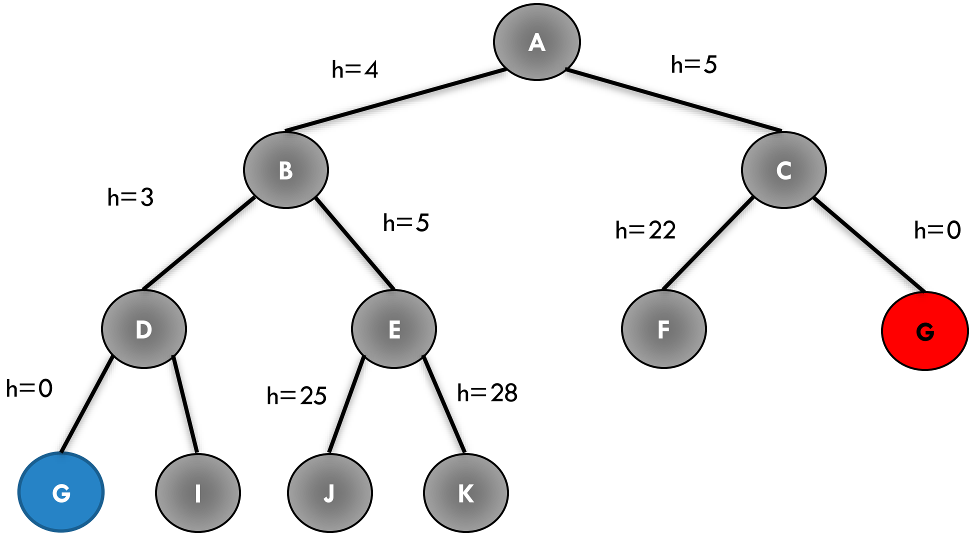
Optimal: No

Time Complexity: $O(b^m)$, but a good heuristic can give dramatic improvement

Space Complexity: $O(b^m)$ -- keeps all nodes in memory

RECALL

List the nodes expanded before the goal node “G” is found for both Breath-first search and Greedy Best first search in the tree below

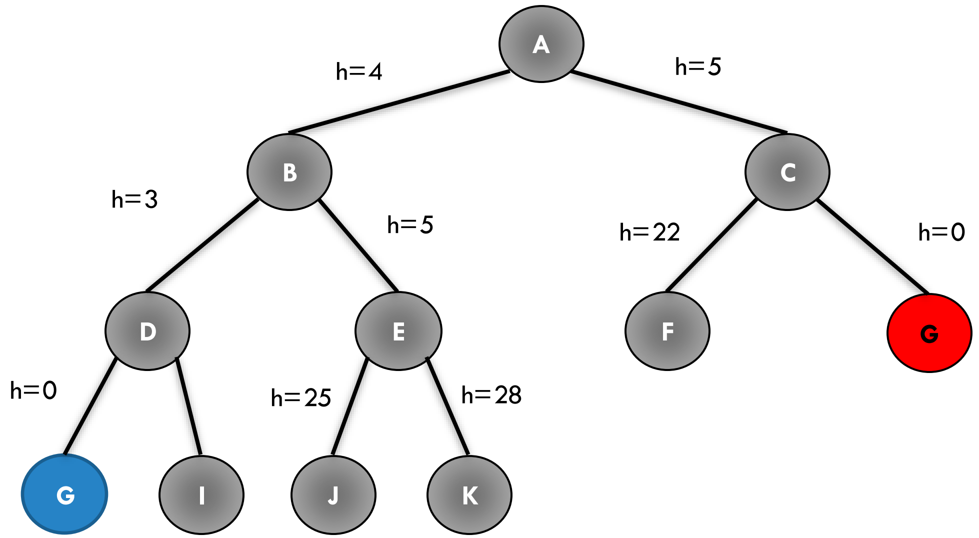


BEST FIRST SEARCH VS BREATH FIRST SEARCH

Assuming all arc costs are 1, the greedy search will find goal G (blue), which has a solution cost of 3, while the optimal solution is the path to goal G (Red) with cost 2.

- Breadth-First Search:
  - Nodes expanded: A,B,C,D,E,F ➡ G
  - Path Found: A-C-G
  - Path Cost: 2

- Greedy Best-First Search:
  - Nodes expanded: A,B,D ➡ G
  - Path Found: A-B-D-G
  - Path Cost: 3



BEAM SEARCH

Beam search

- **Tries to minimize the memory requirement of the breadth-first algorithm.**
- **Informed breadth-first algorithm.**
- Expands only the first ß promising nodes at each level.
- ß is called Beam Width

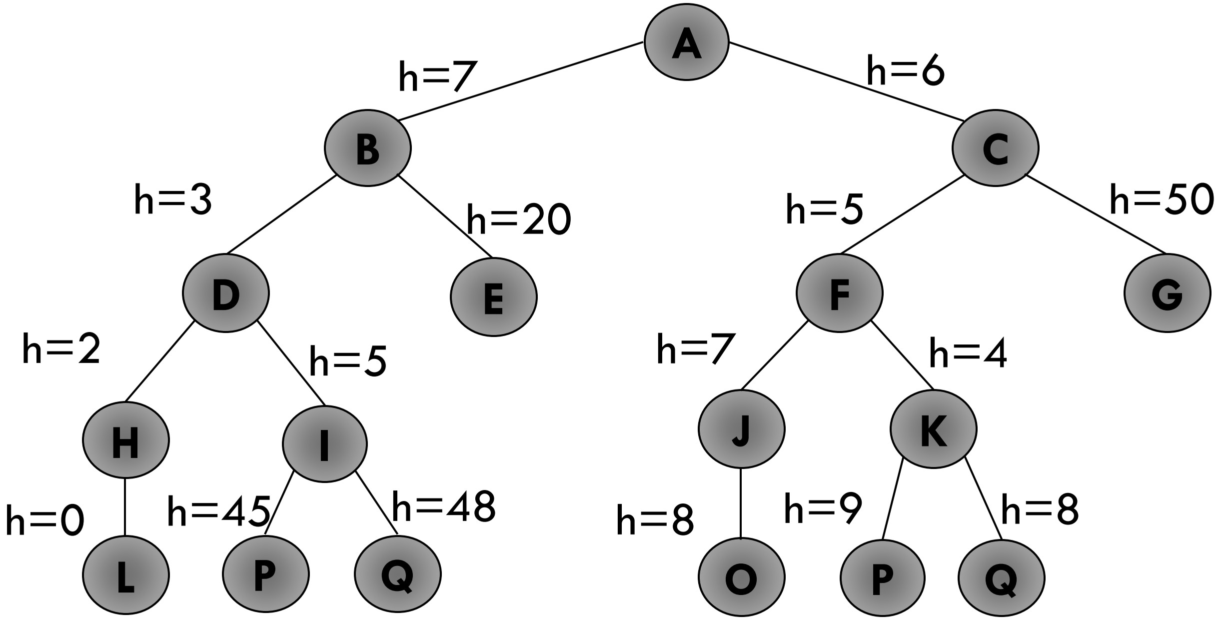
Uses an evaluation function f(n) = h(n), but the maximum size of the nodes list is ß.

Only keeps ß best nodes as candidates for expansion, and throws the rest away

More space efficient than greedy search, but may throw away a node that is on a solution path

BEAM SEARCH

ß = 2



Expanded nodes: A,B,C,D,F, H, K ➡ L

PROPERTIES OF BEAM SEARCH

Complete: No

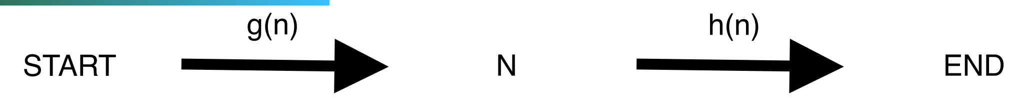
Optimal: No

Time Complexity: $O(ßd)$

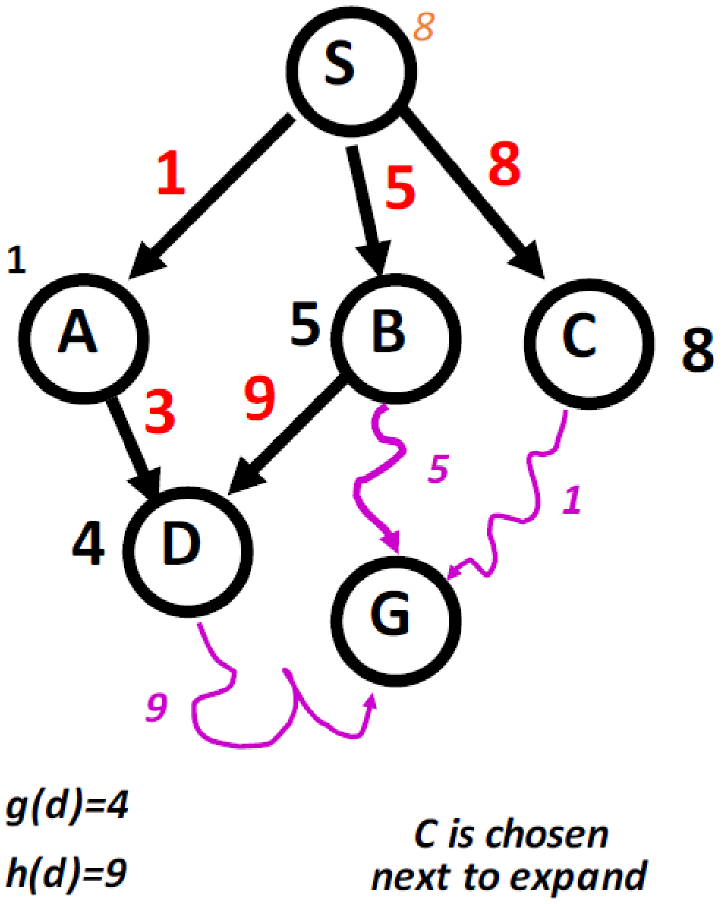
Space Complexity: $O(ßd)$

Not admissible

ALGORITHM A



- Use as an evaluation function
  f(n) = g(n) + h(n)
- g(n) = minimal-cost path from the start state to state n.
- The g(n) term adds a "breadth-first" component to the evaluation function.
- Ranks nodes on search frontier by ***estimated cost*** of solution from ***start node through the given node to goal***.
- Not complete if h(n) can equal infinity.
- Not admissible.



ALGORITHM A*

Algorithm A with constraint that **h(n) <= h*(n)**

h*(n) = true cost of the minimal cost path from n to a goal.

h is **admissible** when h(n) <= h*(n) holds.

Using an admissible heuristic guarantees that the first solution found will be an optimal one.

A* is **complete** whenever the branching factor is **finite,** and every operator has a **fixed positive** cost

A* is **admissible**

SOME OBSERVATIONS ON A

Perfect heuristic: If h(n) = h*(n) for all n:

then only the nodes on the optimal solution path will be expanded. So, no extra work will be per

Null heuristic: If h(n) = 0 for all n:

then this is an admissible heuristic and A* acts like Uniform-Cost Search.

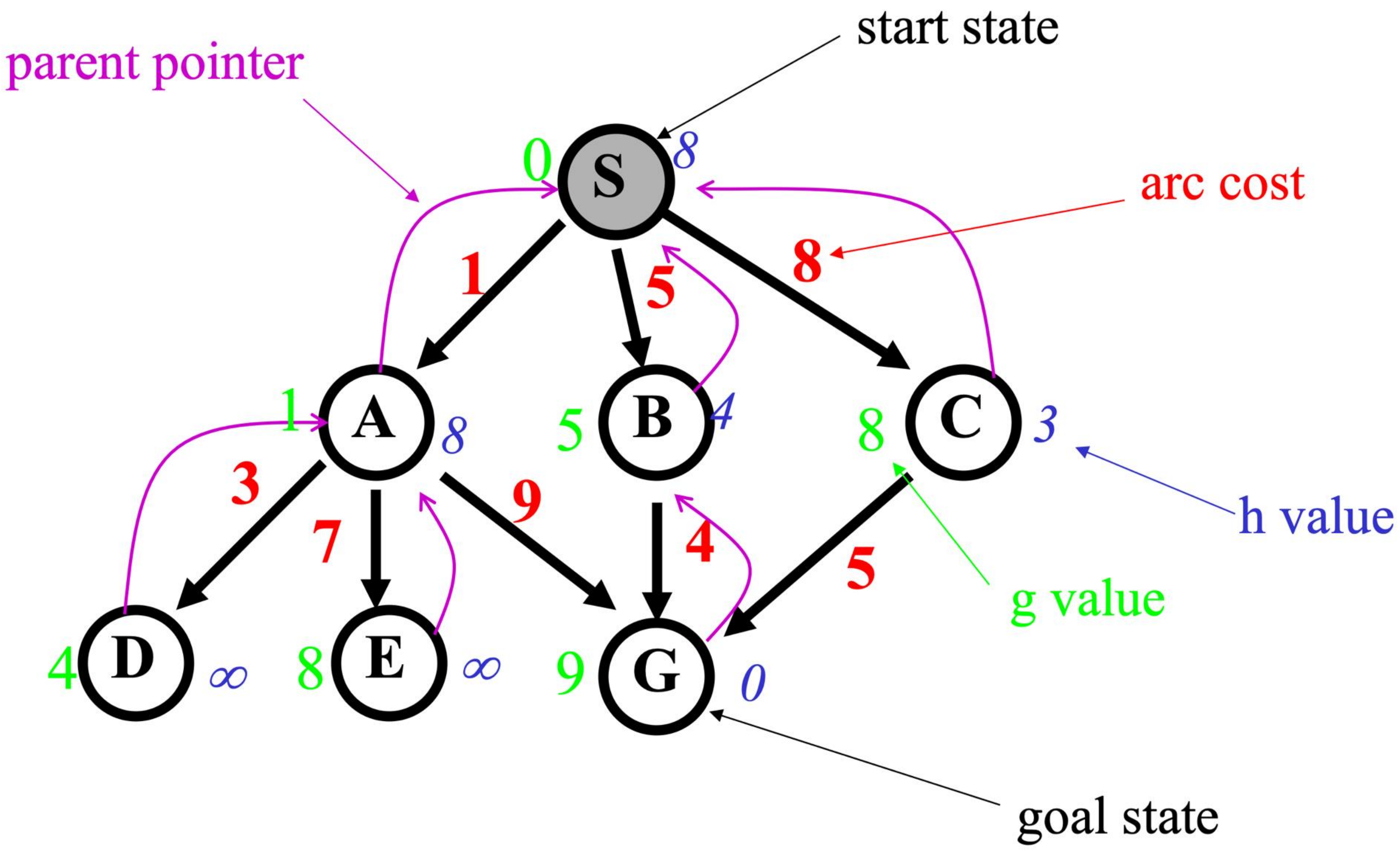
Better heuristic: If h1(n) < h2(n) <= h*(n) for all non-goal nodes:

then h2 is a better heuristic than h1

- If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*.
- In other words, A1 expands at least as many nodes as A2*.
- We say that A2* is better informed than A1*.

The closer h is to h*, the fewer extra nodes that will be expanded

Example search space

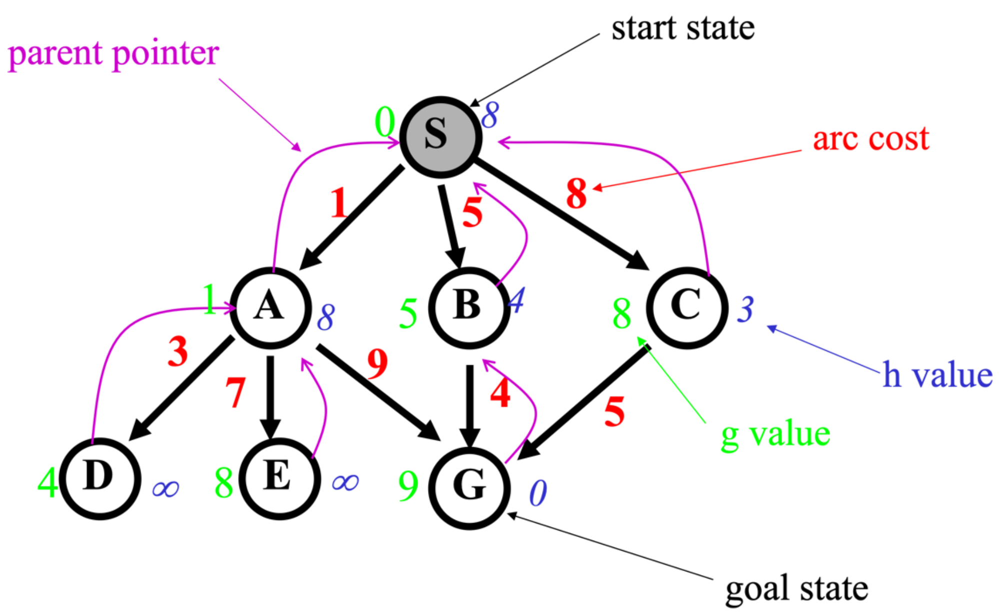


Example

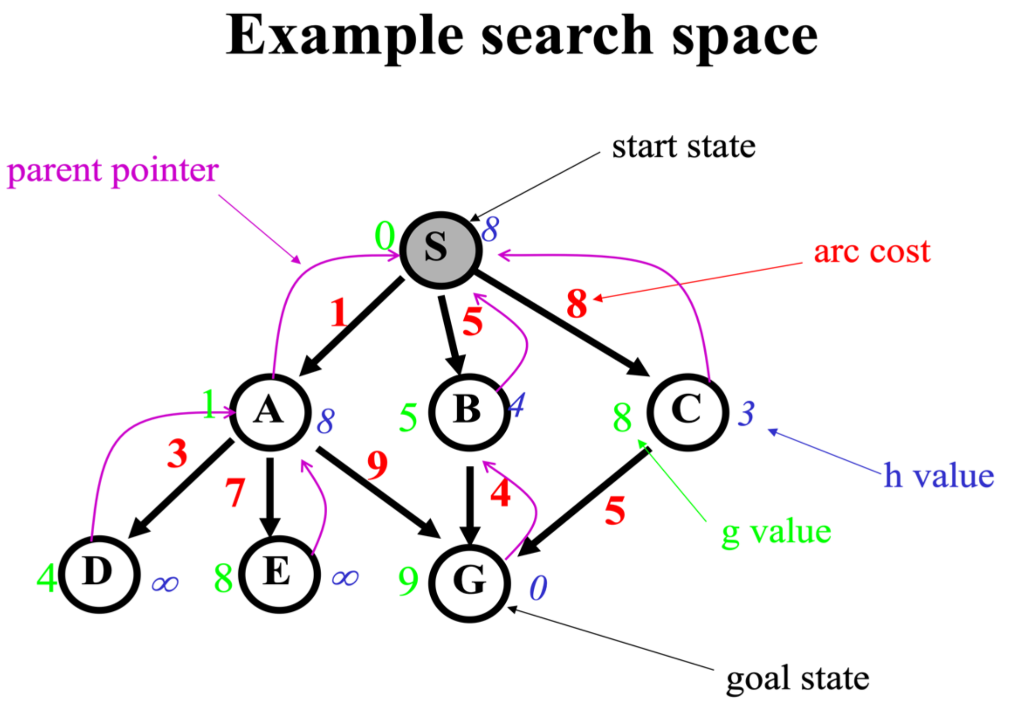
n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	9	0	9	0

- h*(n) is the (hypothetical) perfect heuristic.
- Since h(n) <= h*(n) for all n, h is admissible
- Optimal path = S B G with cost 9.

Example search space



A* search



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

```
node exp.      nodes list
               { S(8) }
 S             { A(9) B(9) C(11) }
 A             { B(9) G(10) C(11) D(inf) E(inf) }
 B             { G(9) G(10) C(11) D(inf) E(inf) }
 G             { C(11) D(inf) E(inf) }
```

- Solution path found is S B G, 4 nodes expanded..
- Still pretty fast. And optimal, too.