Artificial Intelligence

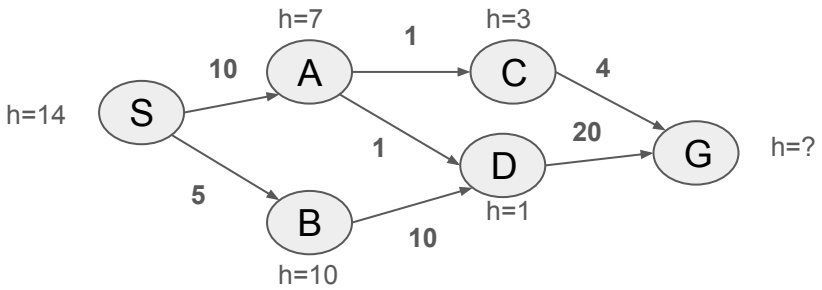
Lec 4 - Informed Search (contd.)

Pratik Mazumder

Content taken from various sources including
CS188 Intro to AI at UC Berkeley

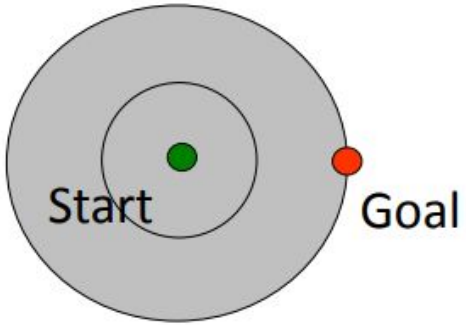
Practice

Is the heuristics function admissible?
Find the solution using A*

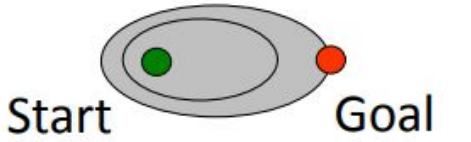


UCS vs A* Contours

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



Comparison

Greedy

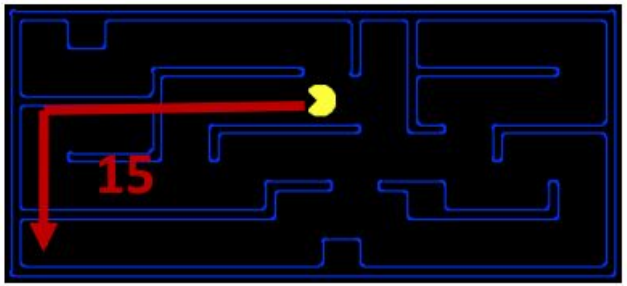
Uniform Cost

A*

Creating Admissible Heuristics

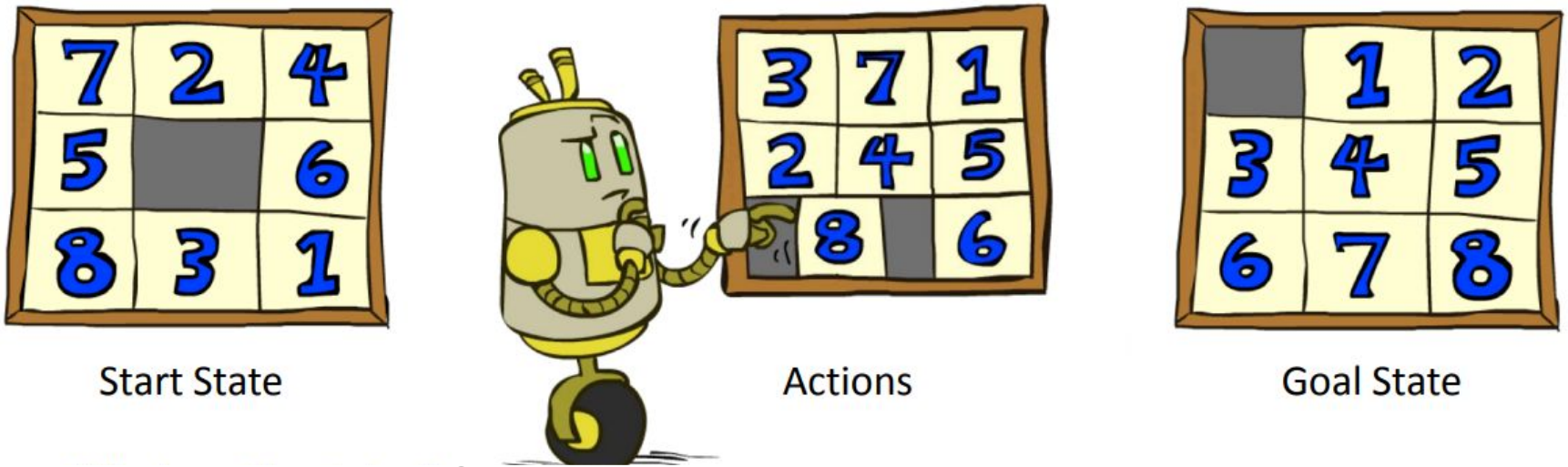
Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

Often, admissible heuristics are solutions to relaxed problems, where new actions are available



Inadmissible heuristics are often useful too, but may not be optimal

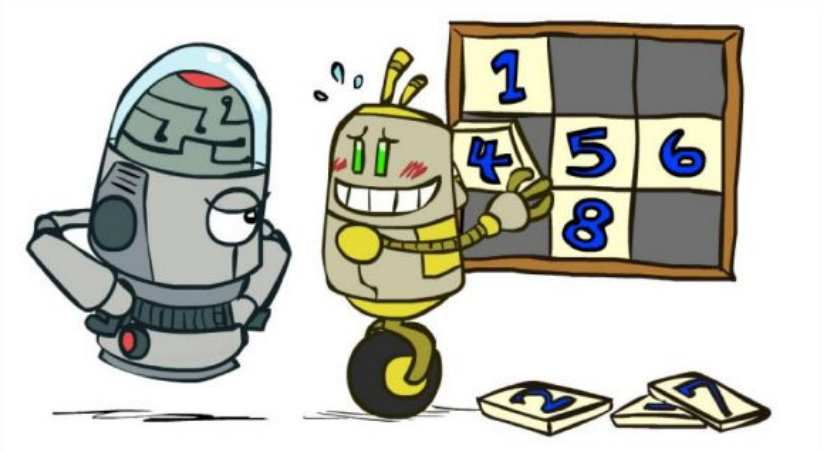
Example: 8 Puzzle

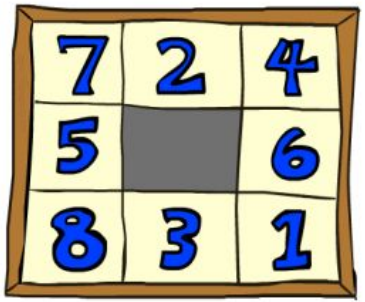


- **What are the states?**
- **What are the actions?**
- **How many successors from the start state?**
- **What should the costs be?**

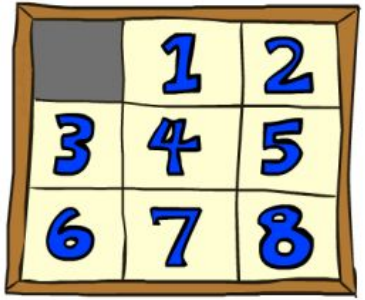
Example: 8 Puzzle - Heuristic 1

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic





Start State

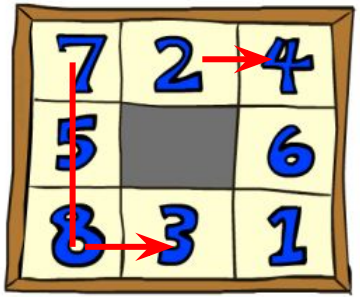


Goal State

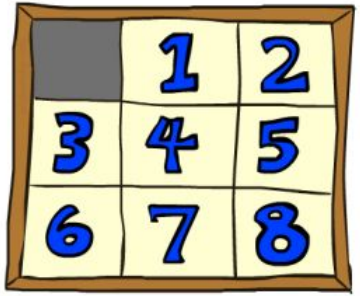
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

Example: 8 Puzzle - Heuristic 2

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- h(start) = 3 + 1 + 2 + … = 18



Start State

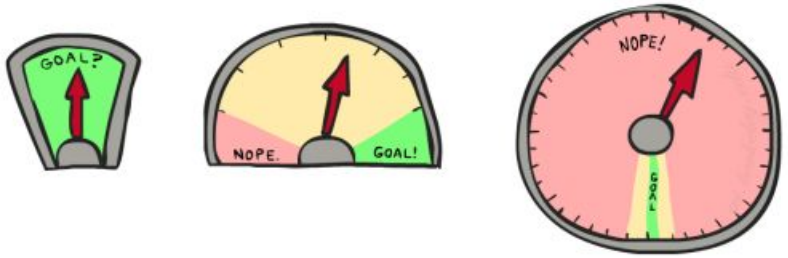


Goal State

	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

Example: 8 Puzzle - Heuristic 3

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Trivial Heuristics, Dominance

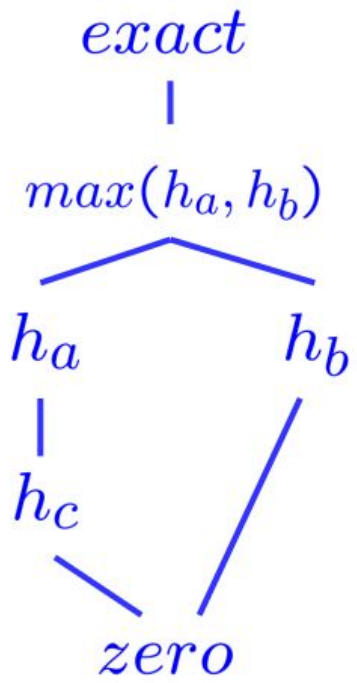
- Dominance: $h_a \geq h_c$ if (assuming both are admissible)

$$\forall n : h_a(n) \geq h_c(n)$$

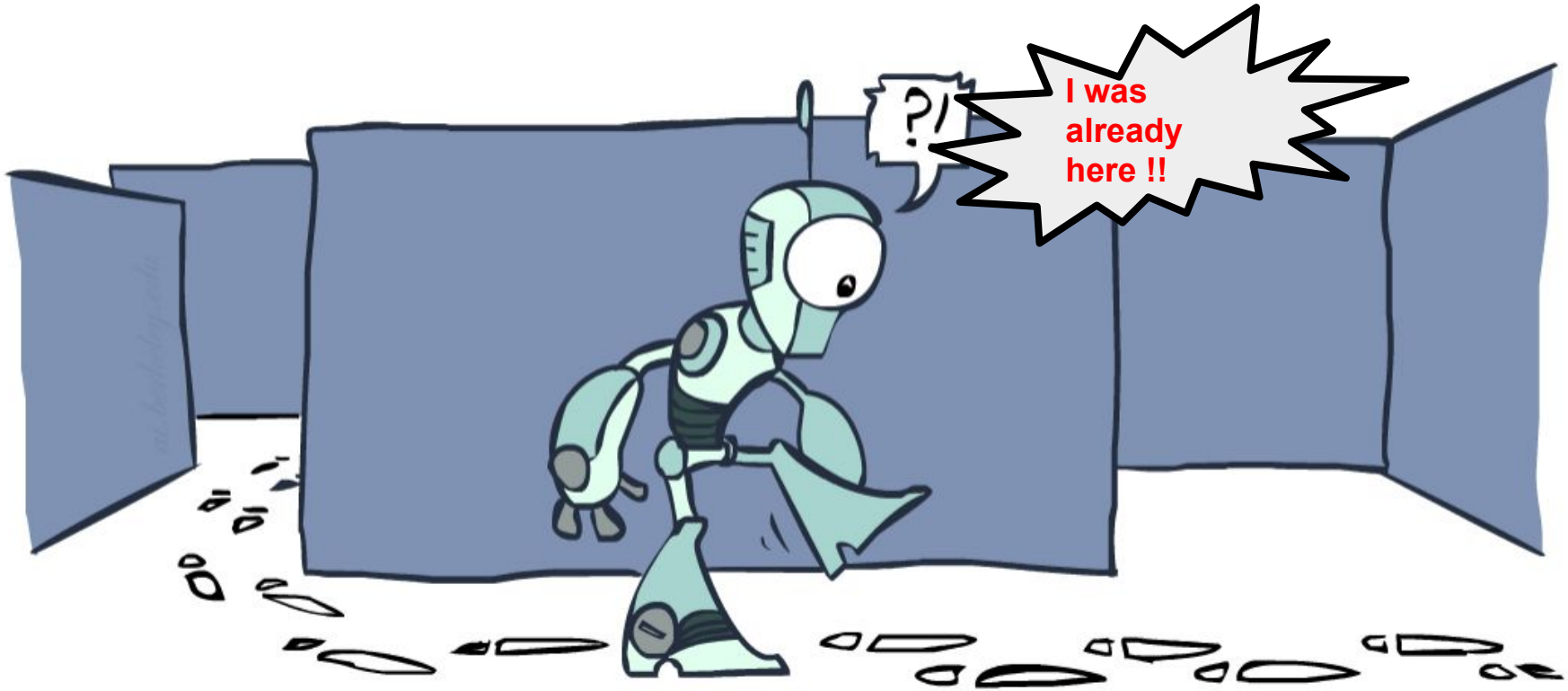
- Max of admissible heuristics is admissible

$$h(n) = max(h_a(n), h_b(n))$$

- Trivial heuristics
  - zero heuristic (worst)
  - exact heuristic (best)

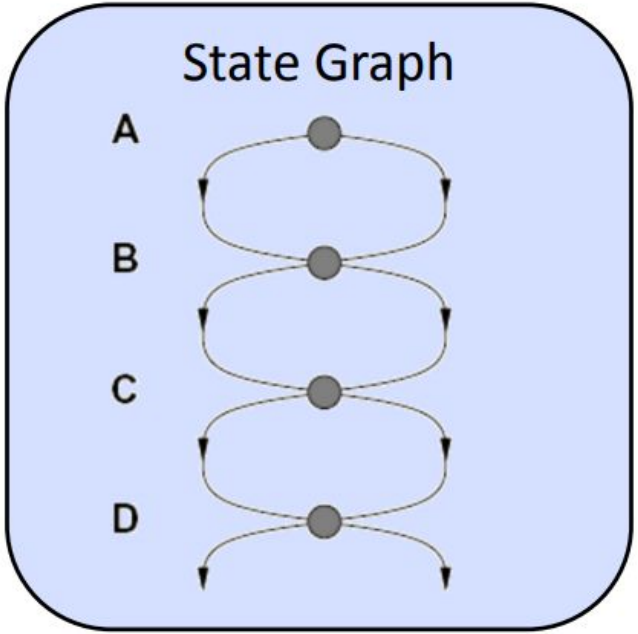


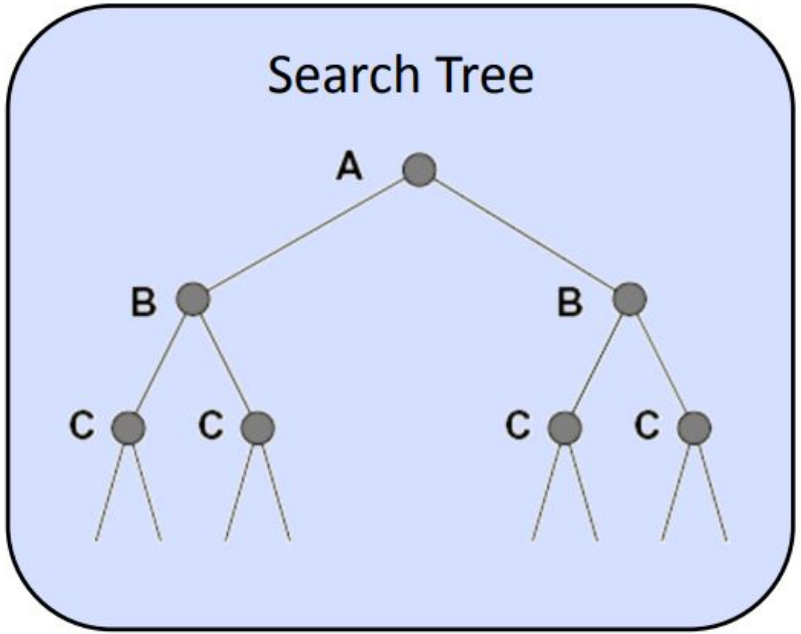
Graph Search



Graph Search

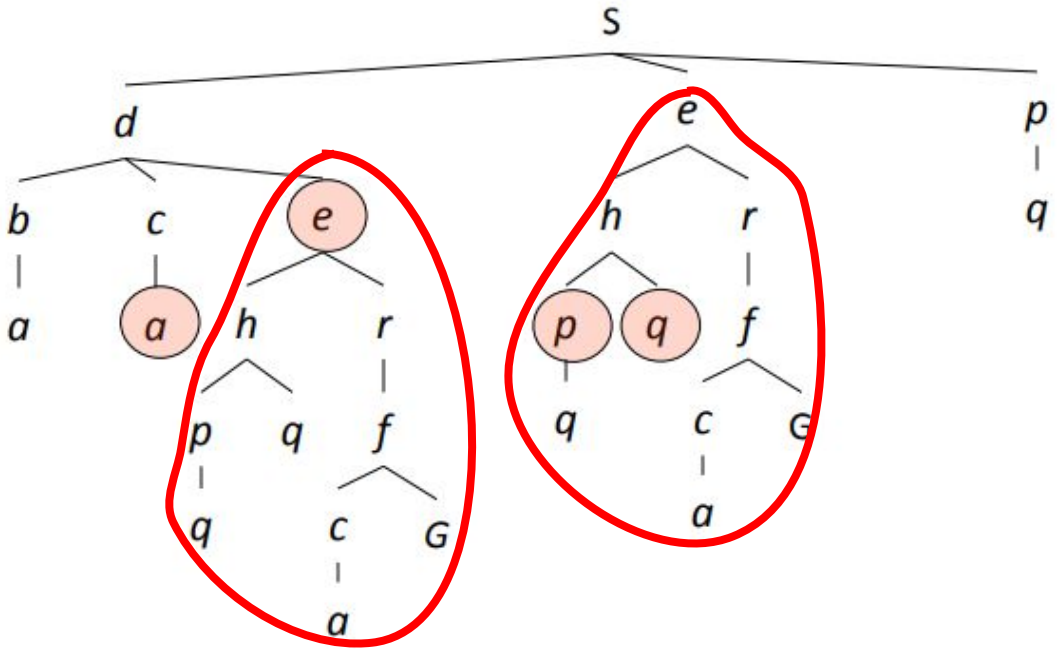
- Failure to detect repeated states can cause exponentially more work in Tree Search.





Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes



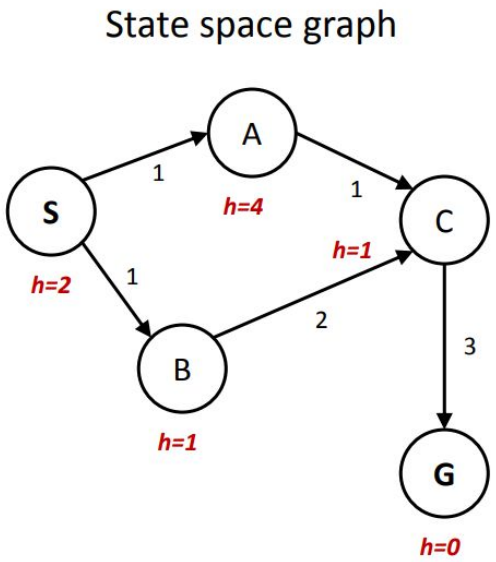
Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set

- Important: store the closed set as a set, not a list
  - Set as opposed to a list only contain unique elements

Graph Search

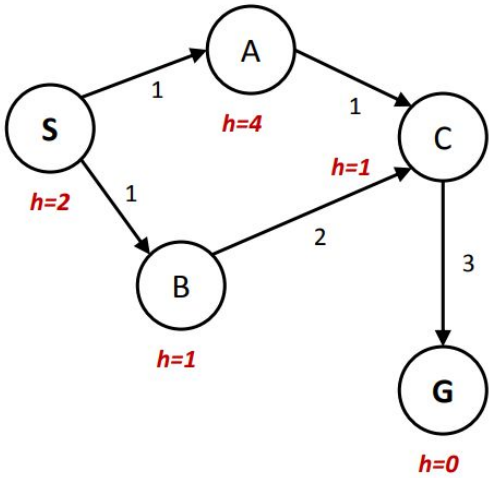
- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search



Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Search tree

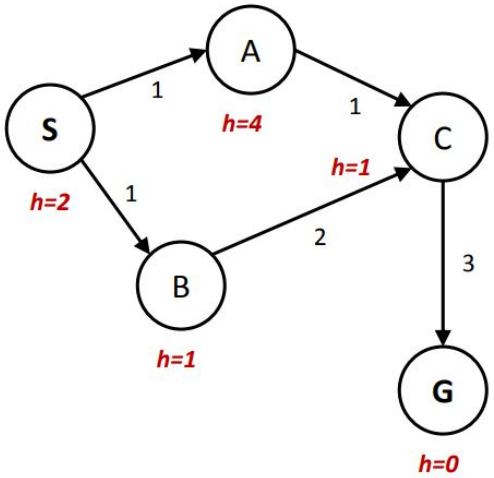
S (0+2)

Fringe
S,2

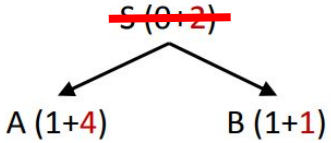
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Search tree



Fringe

~~S,f=2~~

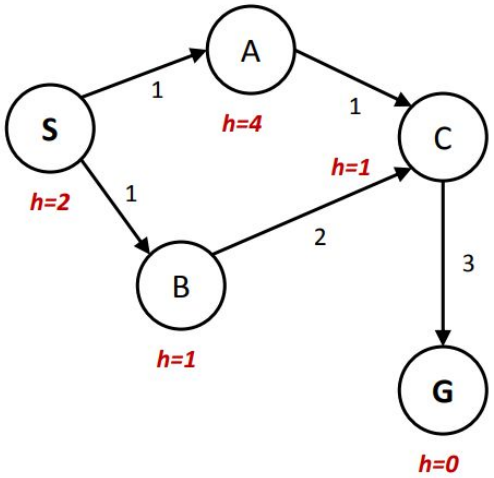
S→A,f=5

S→B,f=2

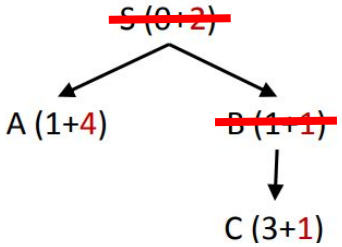
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Search tree



Fringe

~~S,f=2~~

S→A,f=5

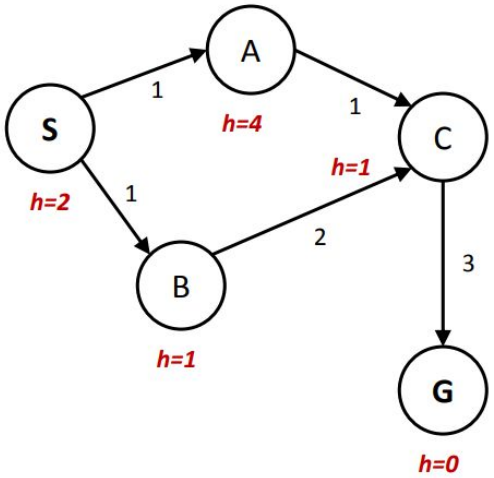
~~S→B,f=2~~

S→B→C,f=4

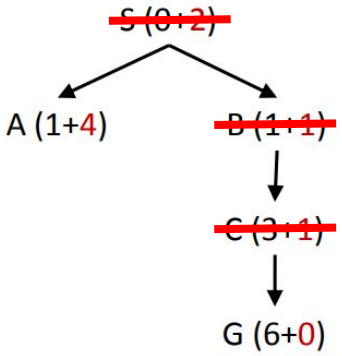
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Search tree



Fringe

~~S,f=2~~

S→A,f=5

~~S→B,f=2~~

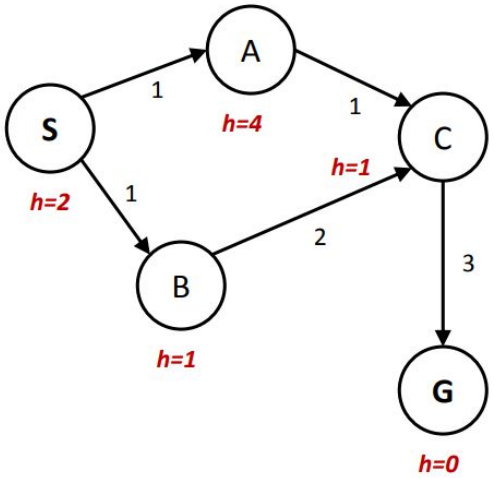
~~S→B→C,f=4~~

S→B→C→G,f=6

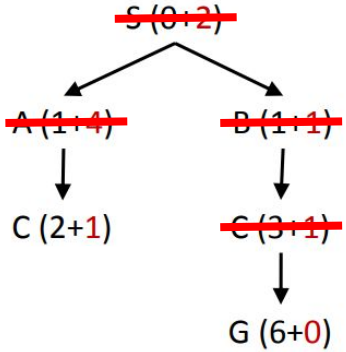
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Search tree



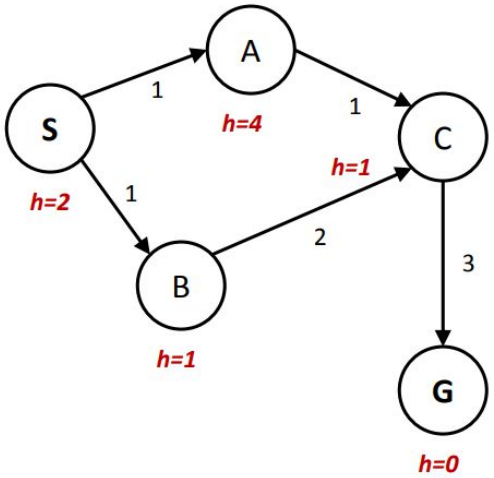
Fringe

~~S,f=2~~
~~S→A,f=5~~
~~S→B,f=2~~
~~S→B→C,f=4~~
S→B→C→G,f=6
S→A→C,f=3

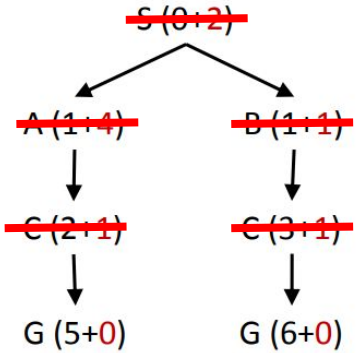
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Search tree



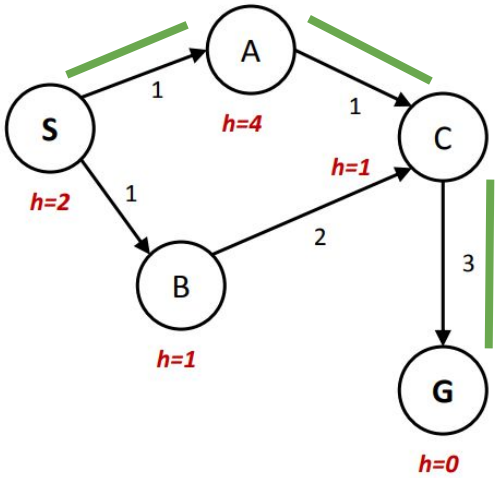
Fringe

~~S,f=2~~
~~S→A,f=5~~
~~S→B,f=2~~
~~S→B→C,f=4~~
S→B→C→G,f=6
~~S→A→C,f=3~~
S→A→C→G,f=5

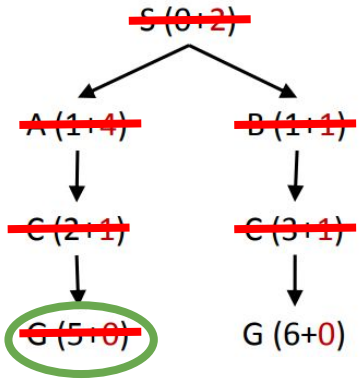
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Search tree



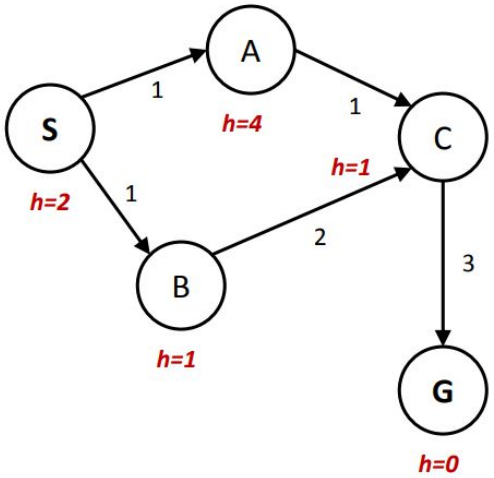
Fringe

~~S,f=2~~
~~S→A,f=5~~
~~S→B,f=2~~
~~S→B→C,f=4~~
S→B→C→G,f=6
~~S→A→C,f=3~~
S→A→C→G,f=5

Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Graph Search

S (0+2)

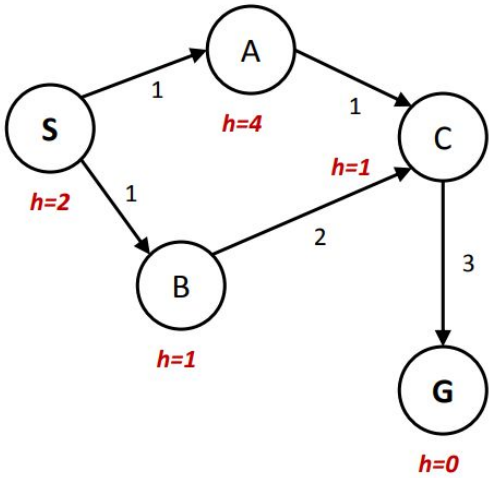
Fringe

S,f=2

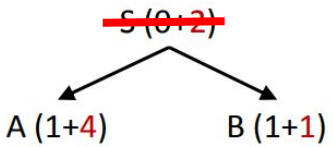
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Graph Search



S

Fringe

~~S,f=2~~

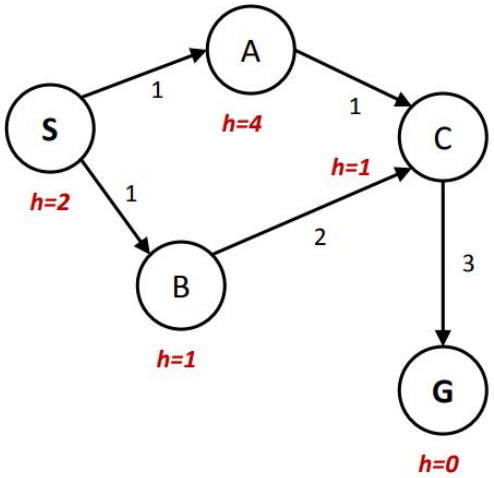
S→A,f=5

S→B,f=2

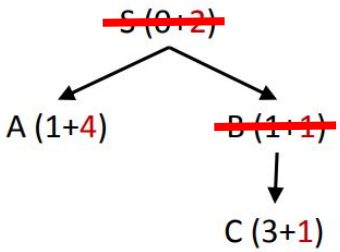
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Graph Search



S	B

Fringe

~~S,f=2~~

S→A,f=5

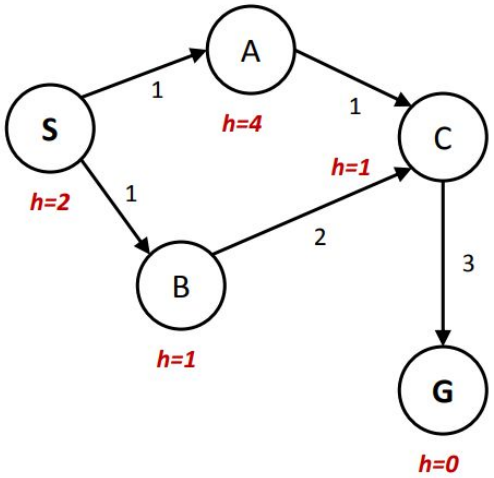
~~S→B,f=2~~

S→B→C,f=4

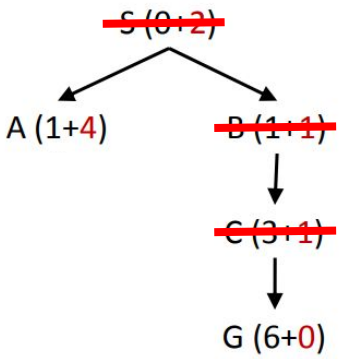
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Graph Search



S	B	C

Fringe

~~S,f=2~~

S→A,f=5

~~S→B,f=2~~

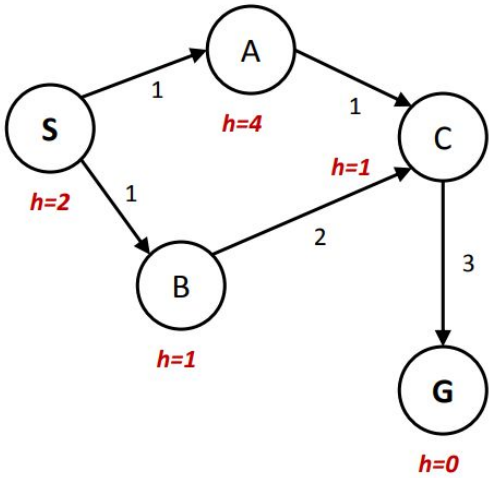
~~S→B→C,f=4~~

S→B→C→G,f=6

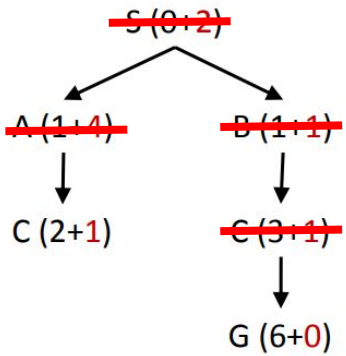
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Graph Search



S	B	C	A

Fringe

~~S,f=2~~

~~S→A,f=5~~

~~S→B,f=2~~

~~S→B→C,f=4~~

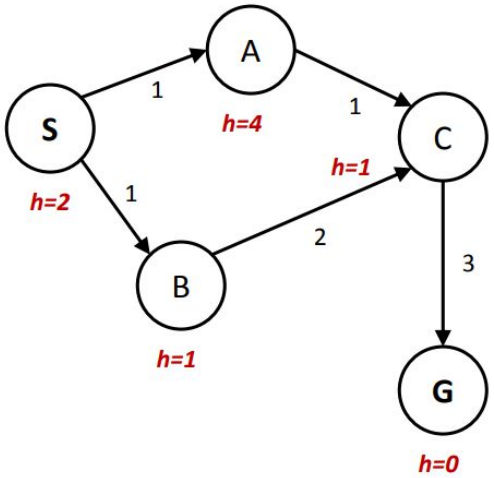
S→B→C→G,f=6

S→A→C,f=3

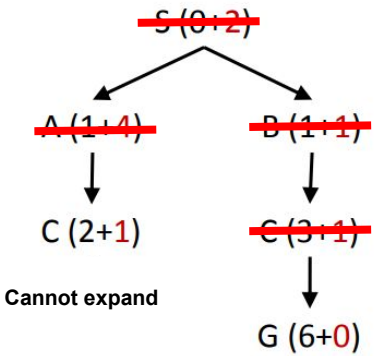
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Graph Search



S	B	C	A

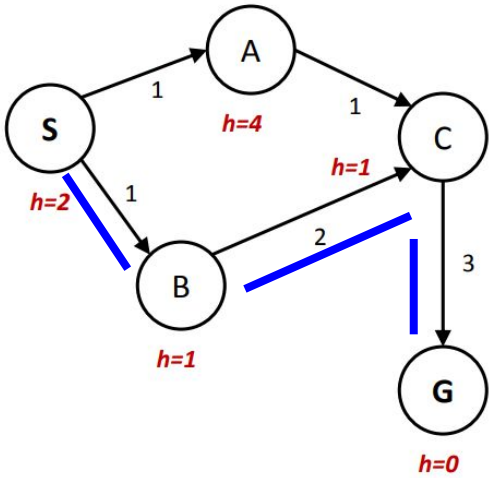
Fringe

~~S,f=2~~
~~S→A,f=5~~
~~S→B,f=2~~
~~S→B→C,f=4~~
S→B→C→G,f=6
S→A→C,f=3

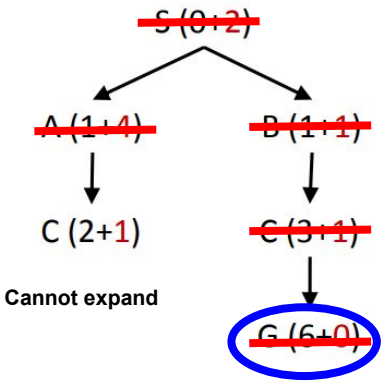
Graph Search

- Is it complete? i.e. Is it guaranteed to find a solution if one exists?
  - Yes. e.g. A* based graph search

State space graph



Graph Search



S	B	C	A	G

Fringe

~~S,f=2~~

~~S→A,f=5~~

~~S→B,f=2~~

~~S→B→C,f=4~~

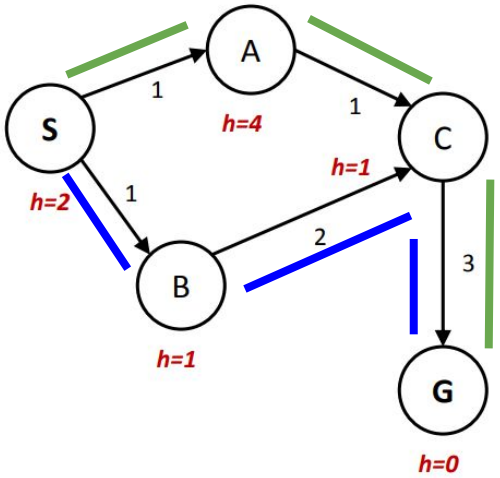
S→B→C→G,f=6

S→A→C,f=3

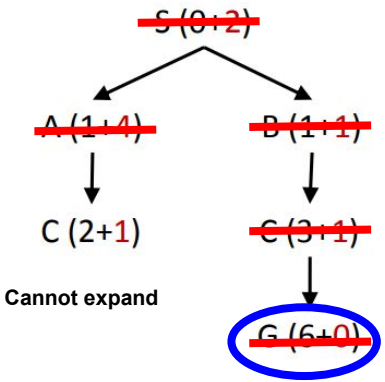
Graph Search

- Is it optimal? i.e. Guaranteed to find the least cost path?
  - No.

State space graph



Graph Search



S	B	C	A	G

Fringe

~~S,f=2~~

~~S→A,f=5~~

~~S→B,f=2~~

~~S→B→C,f=4~~

S→B→C→G,f=6

S→A→C,f=3

How to make Graph Search optimal

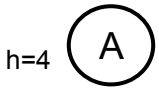
- We have to consider the Consistency of Heuristics

- Admissibility: heuristic cost ≤ actual cost to goal
  - h(A) ≤ actual cost from A to G

- Consistency: heuristic “arc” cost ≤ actual cost for each arc
  - A heuristic h(A) is **consistent** if, for every node A and every successor C of A generated by any action, the **estimated cost of reaching the goal from A** is **not greater** than the **step cost of getting to C plus the estimated cost of reaching the goal from C**
  - h(A) ≤ cost(A to C) + h(C)
  - h(A) – h(C) ≤ cost(A to C)

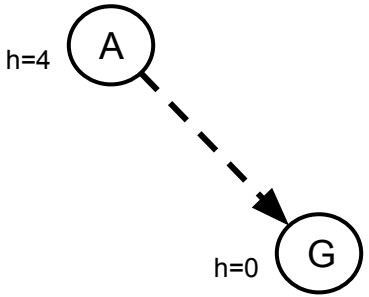
How to make Graph Search optimal

- Consistency: heuristic “arc” cost ≤ actual cost for each arc
  - h(A) – h(C) ≤ cost(A to C)



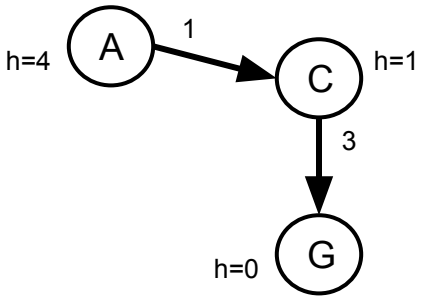
How to make Graph Search optimal

- Consistency: heuristic “arc” cost ≤ actual cost for each arc
  - h(A) – h(C) ≤ cost(A to C)



How to make Graph Search optimal

- Consistency: heuristic “arc” cost ≤ actual cost for each arc
  - h(A) – h(C) ≤ cost(A to C)

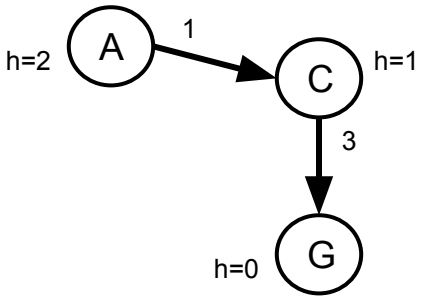


Is h admissible?
Yes

Is h consistent?
No

How to make Graph Search optimal

- Consistency: heuristic “arc” cost ≤ actual cost for each arc
  - h(A) – h(C) ≤ cost(A to C)



Is h admissible?
Yes

Is h consistent?
Yes

Optimality

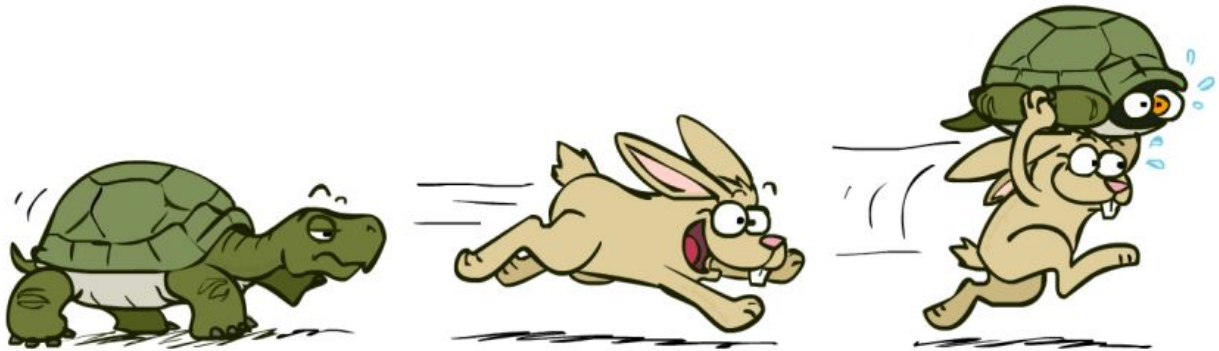
- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)

- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)

- Consistency implies admissibility but the opposite is not necessarily True
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Memory-bounded Heuristic Search

- A* can run out of space while running.
- If the search space is very large, the fringe can become too large to fit in memory, causing the search to fail.
- This is especially true if the heuristic function used is not very informative, causing the search to generate many nodes that are not on the optimal path to the goal.
- Use memory-bounded heuristic search algorithms

Memory-bounded Heuristic Search

Iterative Deepening A* (IDA*) search

- Standard Iterative Deepening performs DFS upto a cutoff depth.
- If no solution is found then the cutoff depth is increased for the next iteration.
- The main difference between IDA* and standard iterative deepening is that the cutoff used is the f-cost (g + h) rather than the depth.
- At each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.