



CS 540 Introduction to Artificial Intelligence
Search II: Informed Search
University of Wisconsin-Madison

Spring 2022

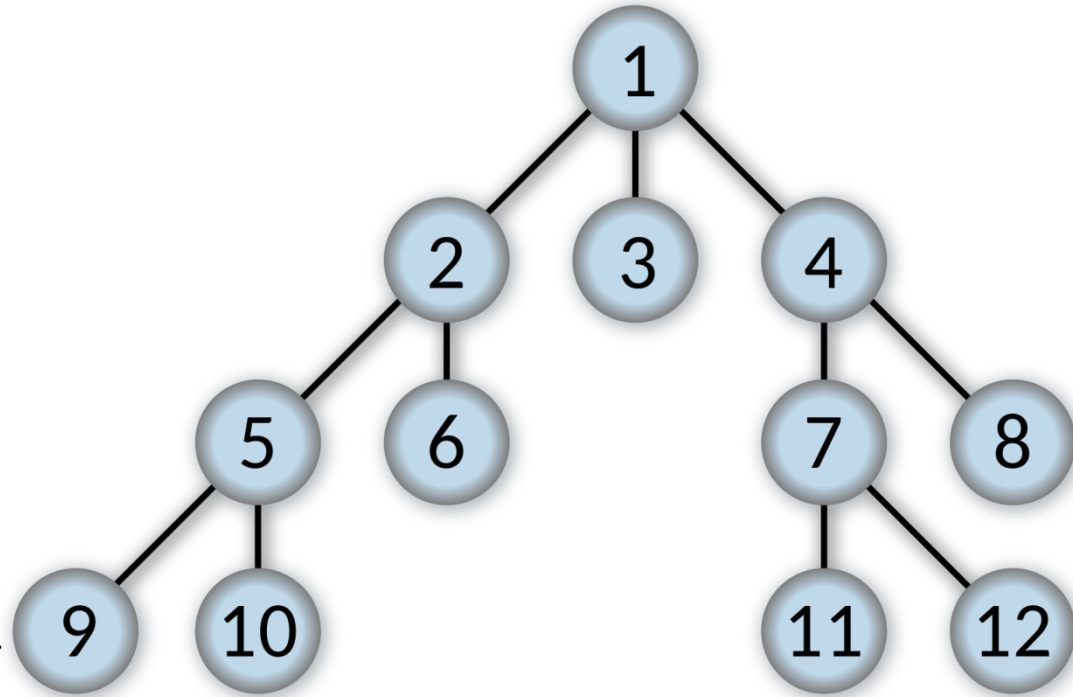
Outline

- Uninformed vs Informed Search
 - Review of uninformed strategies, adding heuristics
- A* Search
 - Heuristic properties, stopping rules, analysis
- Extensions: Beyond A*
 - Iterative deepening, beam search

Breadth-First Search

Recall:


- Data structure: queue
- **Properties:**
 - Complete
 - Optimal (if edge cost 1)
 - Time $O(b^d)$
 - ← Depth
 - ← Branching Factor
 - Space $O(b^d)$



Uniform Cost Search

Like BFS, but keeps track of cost

- Expand least cost node
- Data structure: priority queue
- **Properties:**
 - Complete
 - Optimal (if weight lower bounded by ϵ)
 - Time $O(b^{C^*/\epsilon})$
 - Space $O(b^{C^*/\epsilon})$



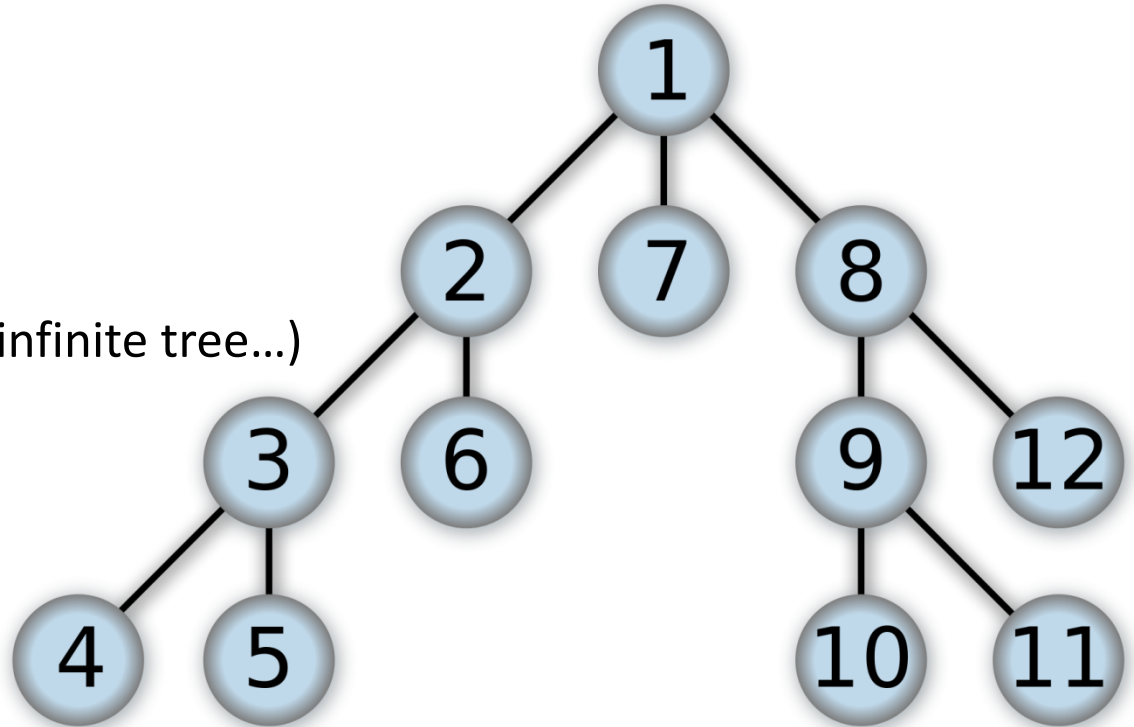
Optimal goal path cost

Depth-First Search

Recall:

- Data structure: stack
- **Properties:**
 - Incomplete (stuck in infinite tree...)
 - Suboptimal
 - Time $O(b^m)$
 - Space $O(bm)$

Max Depth



Iterative Deepening DFS

Repeated depth-limited DFS

- Search like BFS, fringe like DFS
- **Properties:**
 - Complete
 - Optimal (if edge cost 1)
 - Time $O(b^d)$
 - Space $O(bd)$

A good option!

Uninformed vs Informed Search

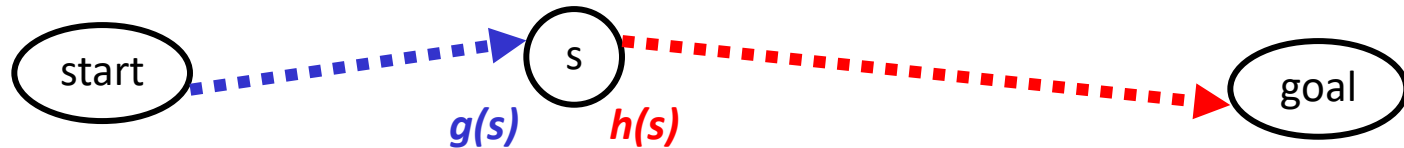
Uninformed search (all of what we saw). Knows:

- Path cost $g(s)$ from start to node s
- Successors.



Informed search. Knows:

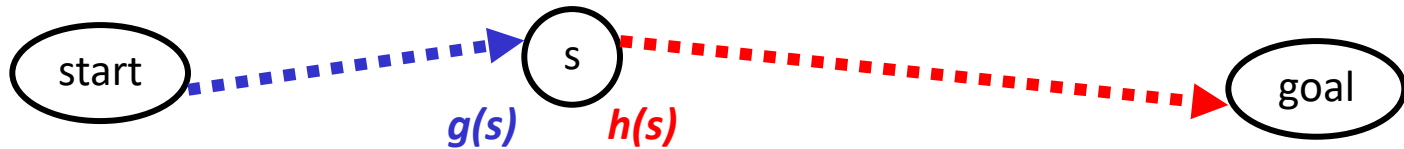
- All uninformed search properties, plus
- Heuristic $h(s)$ from s to goal (recall game heuristic)



Informed Search

Informed search. Know:

- All uninformed search properties, plus
- Heuristic $h(s)$ from s to goal (recall game heuristic)

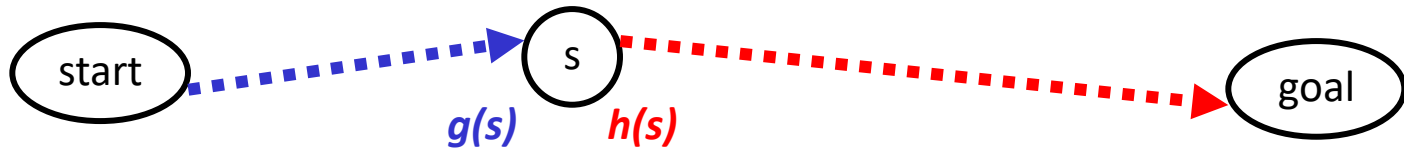


- Like in games, use information to **speed up search**.

Using the Heuristic

Back to uniform-cost search

- We had the priority queue
- Expand the node with the smallest $g(s)$
 - $g(s)$ “first-half-cost”

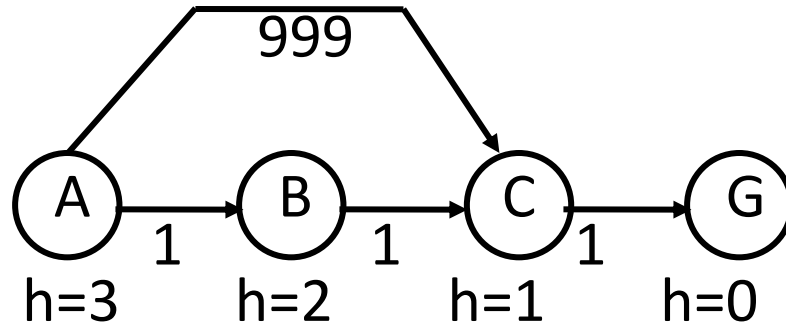


- Now let's use the heuristic (“second-half-cost”)
 - Several possible approaches: let's see what works

Attempt 1: Best-First Greedy

One approach: just use $h(s)$ alone

- Specifically, expand node with smallest $h(s)$
- This isn't a good idea. Why?

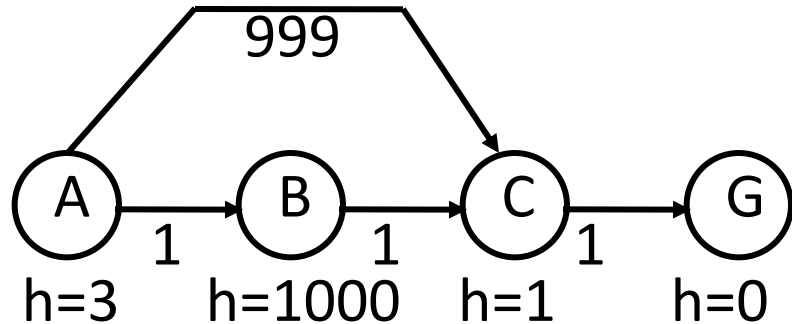


- Not optimal! **Get** $A \rightarrow C \rightarrow G$. **Want:** $A \rightarrow B \rightarrow C \rightarrow G$

Attempt 2: A Search

Next approach: use both $g(s) + h(s)$ alone

- Specifically, expand node with smallest $g(s) + h(s)$
- Again, use a priority queue
- Called “A” search



- **Still not optimal!** (Does work for former example).

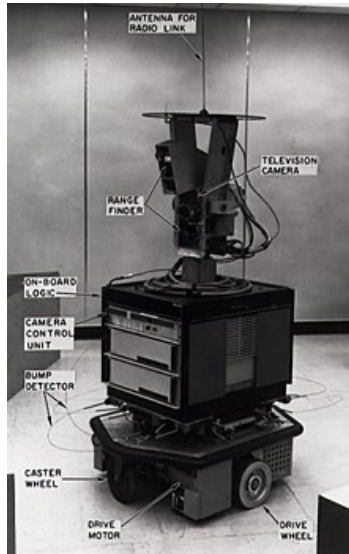
Attempt 3: A* Search

Same idea, use $g(s) + h(s)$, with one **requirement**

- Demand that $h(s) \leq h^*(s)$, for all s
- If heuristic has this property, “admissible”
 - Optimistic! Never over-estimates
- Still need $h(s) \geq 0$
 - Negative heuristics can lead to strange behavior
- This is **A* search**

Attempt 3: A* Search

Origins: robots and planning



Shakey the Robot,
1960's

Credit: Wiki



Animation: finding a path
around obstacle

Credit: Wiki

Admissible Heuristic Functions

Have to be careful to ensure admissibility (**optimism!**)

- Example: **8 Game**

Example
State

1		5
2	6	3
7	4	8

Goal
State

1	2	3
4	5	6
7	8	

- One useful approach: **relax constraints**
 - $h(s)$ = number of tiles in wrong position
 - allows tiles to fly to destination in a single step

Break & Quiz

Q 1.1: Consider finding the fastest driving route from one US city to another. Measure cost as the number of hours driven when driving at the speed limit. Let $h(s)$ be the number of hours needed to ride a bike from city s to your destination. $h(s)$ is

- A. An admissible heuristic
- B. Not an admissible heuristic

Break & Quiz

Q 1.1: Consider finding the fastest driving route from one US city to another. Measure cost as the number of hours driven when driving at the speed limit. Let $h(s)$ be the number of hours needed to ride a bike from city s to your destination. $h(s)$ is

- A. An admissible heuristic
- **B. Not an admissible heuristic**

Break & Quiz

Q 1.1: Consider finding the fastest driving route from one US city to another. Measure cost as the number of hours driven when driving at the speed limit. Let $h(s)$ be the number of hours needed to ride a bike from city s to your destination. $h(s)$ is

- A. An admissible heuristic **No: riding your bike take longer.**
- **B. Not an admissible heuristic**

Break & Quiz

Q 1.2: Which of the following are admissible heuristics?

(i) $h(s) = h^*(s)$

(ii) $h(s) = \max(2, h^*(s))$

(iii) $h(s) = \min(2, h^*(s))$

(iv) $h(s) = h^*(s) - 2$

(v) $h(s) = \text{sqrt}(h^*(s))$

- A. All of the above
- B. (i), (iii), (iv)
- C. (i), (iii)
- D. (i), (iii), (v)

Break & Quiz

Q 1.2: Which of the following are admissible heuristics?

(i) $h(s) = h^*(s)$

(ii) $h(s) = \max(2, h^*(s))$

(iii) $h(s) = \min(2, h^*(s))$

(iv) $h(s) = h^*(s) - 2$

(v) $h(s) = \text{sqrt}(h^*(s))$

- A. All of the above
- B. (i), (iii), (iv)
- **C. (i), (iii)**
- D. (i), (iii), (v)

Break & Quiz

Q 1.2: Which of the following are admissible heuristics?

(i) $h(s) = h^*(s)$

(ii) $h(s) = \max(2, h^*(s))$ No: $h(s)$ might be too big

(iii) $h(s) = \min(2, h^*(s))$

(iv) $h(s) = h^*(s) - 2$ No: $h(s)$ might be negative

(v) $h(s) = \text{sqrt}(h^*(s))$ No: if $h^*(s) < 1$ then $h(s)$ is bigger

- A. All of the above
- B. (i), (iii), (iv)
- **C. (i), (iii)**
- D. (i), (iii), (v)

Heuristic Function Tradeoffs

Dominance: h_2 dominates h_1 if for all states s ,

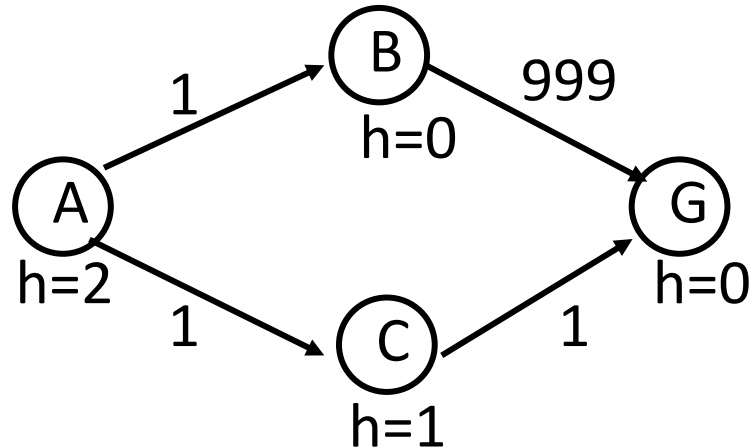
$$h_1(s) \leq h_2(s) \leq h^*(s)$$

- **Idea:** we want to be as close to h^* as possible
 - But not over!
- **Tradeoff:** being very close might require a very complex heuristic, expensive computation
 - Might be better off with cheaper heuristic & expand more nodes.

A* Termination

When should A* **stop**?

- One idea: as soon as we reach goal state?

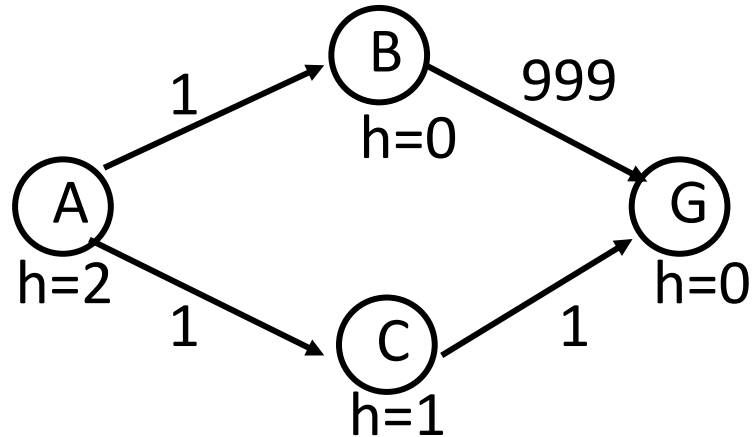


- *h* admissible, but note that we get $A \rightarrow B \rightarrow G$ (**cost 1000**)!

A* Termination

When should A* stop?

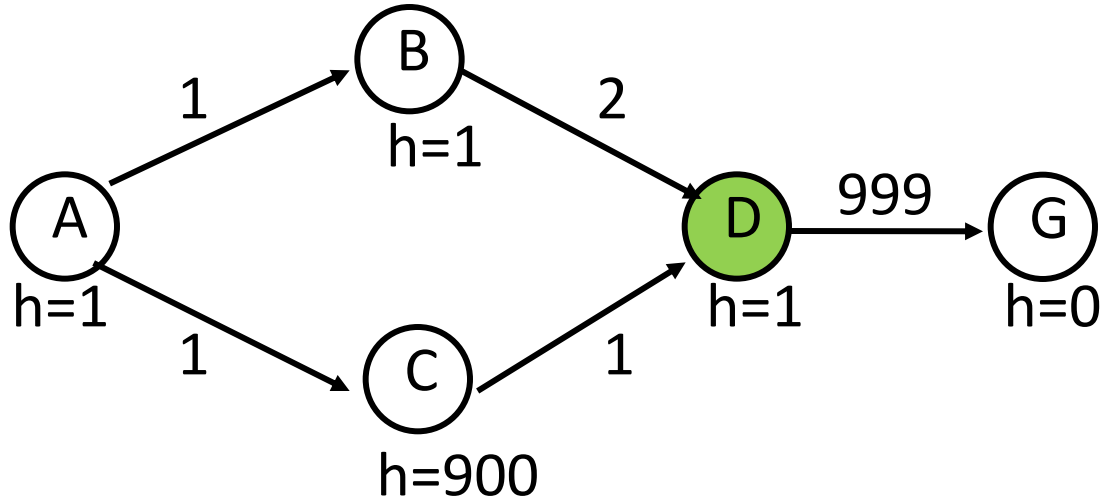
- **Rule:** terminate **when a goal is popped** from queue.



- Note: taking $h=0$ reduces to uniform cost search rule.

A* Revisiting Expanded States

Possible to revisit an expanded state, get a shorter path:



- Put D back into priority queue, smaller $g+h$

A* Full Algorithm

1. Put the start node S on the priority queue, called $OPEN$
2. If $OPEN$ is empty, exit with failure
3. Remove from $OPEN$ and place on $CLOSED$ a node n for which $f(n)$ is minimum (note that $f(n)=g(n)+h(n)$)
4. If n is a goal node, exit (trace back pointers from n to S)
5. Expand n , generating all successors and attach to pointers back to n . For each successor n' of n
 1. If n' is not already on $OPEN$ or $CLOSED$ estimate $h(n')$, $g(n')=g(n)+c(n,n')$, $f(n')=g(n')+h(n')$, and place it on $OPEN$.
 2. If n' is already on $OPEN$ or $CLOSED$, then check if $g(n')$ is lower for the new version of n' . If so, then:
 1. Redirect pointers backward from n' along path yielding lower $g(n')$.
 2. Put n' on $OPEN$.
 3. If $g(n')$ is not lower for the new version, do nothing.
6. Goto 2.

A* Analysis

Some properties:

- Terminates!
- A* can use **lots of memory**: $O(\# \text{ states})$.
- Will run out on large problems.
- Next, we will consider some alternatives to deal with this.

```
ubuntu@ip-172-31-46-218: ~
File Edit View Search Terminal Help
1 | [|||||] 93.5% 25 [|||||] 183.6% 49 [|||||] 100.0% 73 [|||||] 100.0%
2 | [|||||] 89.6% 26 [|||||] 176.7% 50 [|||||] 100.0% 74 [|||||] 100.0%
3 | [|||||] 66.4% 27 [|||||] 189.5% 51 [|||||] 100.0% 75 [|||||] 100.0%
4 | [|||||] 70.4% 28 [|||||] 189.5% 52 [|||||] 100.0% 76 [|||||] 100.0%
5 | [|||||] 93.5% 29 [|||||] 186.3% 53 [|||||] 100.0% 77 [|||||] 100.0%
6 | [|||||] 83.0% 30 [|||||] 194.1% 54 [|||||] 100.0% 78 [|||||] 100.0%
7 | [|||||] 76.4% 31 [|||||] 190.2% 55 [|||||] 100.0% 79 [|||||] 100.0%
8 | [|||||] 69.7% 32 [|||||] 183.0% 56 [|||||] 100.0% 80 [|||||] 100.0%
9 | [|||||] 62.7% 33 [|||||] 179.1% 57 [|||||] 100.0% 81 [|||||] 100.0%
10 | [|||||] 86.3% 34 [|||||] 156.9% 58 [|||||] 100.0% 82 [|||||] 100.0%
11 | [|||||] 86.9% 35 [|||||] 186.9% 59 [|||||] 100.0% 83 [|||||] 100.0%
12 | [|||||] 62.3% 36 [|||||] 183.7% 60 [|||||] 100.0% 84 [|||||] 100.0%
13 | [|||||] 184.2% 37 [|||||] 181.9% 61 [|||||] 100.0% 85 [|||||] 100.0%
14 | [|||||] 70.4% 38 [|||||] 170.0% 62 [|||||] 100.0% 86 [|||||] 100.0%
15 | [|||||] 70.6% 39 [|||||] 179.7% 63 [|||||] 100.0% 87 [|||||] 100.0%
16 | [|||||] 185.8% 40 [|||||] 177.9% 64 [|||||] 100.0% 88 [|||||] 100.0%
17 | [|||||] 85.7% 41 [|||||] 180.5% 65 [|||||] 100.0% 89 [|||||] 100.0%
18 | [|||||] 65.8% 42 [|||||] 185.1% 66 [|||||] 100.0% 90 [|||||] 100.0%
19 | [|||||] 80.5% 43 [|||||] 181.7% 67 [|||||] 100.0% 91 [|||||] 100.0%
20 | [|||||] 76.6% 44 [|||||] 190.1% 68 [|||||] 100.0% 92 [|||||] 100.0%
21 | [|||||] 85.7% 45 [|||||] 175.8% 69 [|||||] 100.0% 93 [|||||] 100.0%
22 | [|||||] 83.8% 46 [|||||] 183.0% 70 [|||||] 100.0% 94 [|||||] 100.0%
23 | [|||||] 88.9% 47 [|||||] 165.1% 71 [|||||] 100.0% 95 [|||||] 29.8%
24 | [|||||] 84.3% 48 [|||||] 186.9% 72 [|||||] 100.0% 96 [|||||] 100.0%
Mem [|||||] 177G/370G Tasks: 332, 49 thr, 749 kthr; 89 running
Swap [|||||] 0K/0K Load average: 157.55 143.11 124.84
Uptime: 13:52:44

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
3231 ubuntu 20 0 2384M 1925M 5820 R 102.0 0.5 10h48:36 /usr/lib/R/bin/exec/R
3232 ubuntu 20 0 2189M 1664M 5800 R 102.0 0.4 11h21:04 /usr/lib/R/bin/exec/R
3232 ubuntu 20 0 2384M 1925M 5820 R 102.0 0.5 10h38:29 /usr/lib/R/bin/exec/R
3176 ubuntu 20 0 2792M 2252M 5788 R 102.0 0.6 11h12:51 /usr/lib/R/bin/exec/R
3176 ubuntu 20 0 2384M 1925M 5800 R 102.0 0.5 11h37:07 /usr/lib/R/bin/exec/R
3154 ubuntu 20 0 1878M 1361M 5800 R 102.0 0.4 11h19:01 /usr/lib/R/bin/exec/R
3146 ubuntu 20 0 2588M 1943M 5788 R 102.0 0.5 11h18:23 /usr/lib/R/bin/exec/R
3208 ubuntu 20 0 2402M 1644M 5788 R 102.0 0.4 11h03:11 /usr/lib/R/bin/exec/R
3148 ubuntu 20 0 2922M 2367M 5788 R 102.0 0.6 11h38:07 /usr/lib/R/bin/exec/R
3230 ubuntu 20 0 1980M 1522M 5788 R 102.0 0.4 10h48:57 /usr/lib/R/bin/exec/R
3150 ubuntu 20 0 2588M 1985M 5788 R 102.0 0.5 11h42:22 /usr/lib/R/bin/exec/R
3207 ubuntu 20 0 2615M 2157M 5788 R 102.0 0.6 11h13:52 /usr/lib/R/bin/exec/R
3200 ubuntu 20 0 2996M 2337M 5788 R 101.0 0.6 11h46:18 /usr/lib/R/bin/exec/R
3157 ubuntu 20 0 2792M 2263M 5800 R 101.0 0.6 11h52:37 /usr/lib/R/bin/exec/R
3152 ubuntu 20 0 2591M 2068M 5788 R 101.0 0.5 11h37:38 /usr/lib/R/bin/exec/R
F1 Help F2 Setup F3 Search F4 Filter F5 Tree F6 SortBy F7 Nice F8 NICE F9 Kill F10 Quit
```

Break & Quiz

Q 2.1: Consider two heuristics for the 8 puzzle problem. h_1 is the number of tiles in wrong position. h_2 is the l_1 /Manhattan distance between the tiles and the goal location. How do h_1 and h_2 relate?

- A. h_2 dominates h_1
- B. h_1 dominates h_2
- C. Neither dominates the other

Break & Quiz

Q 2.1: Consider two heuristics for the 8 puzzle problem. h_1 is the number of tiles in wrong position. h_2 is the l_1 /Manhattan distance between the tiles and the goal location. How do h_1 and h_2 relate?

- **A. h_2 dominates h_1**
- B. h_1 dominates h_2
- C. Neither dominates the other

Break & Quiz

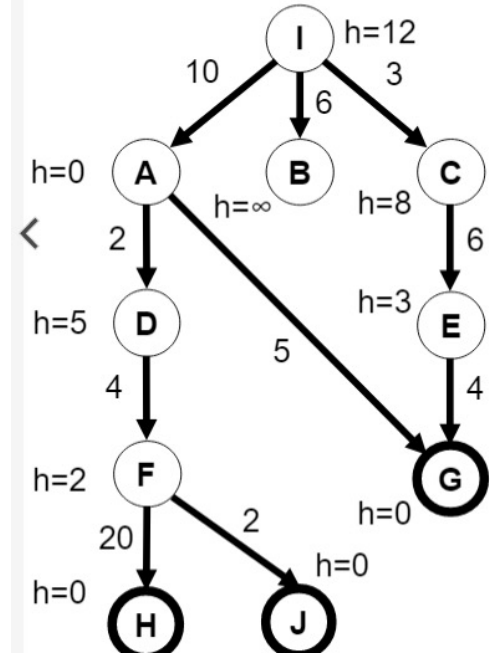
Q 2.1: Consider two heuristics for the 8 puzzle problem. h_1 is the number of tiles in wrong position. h_2 is the l_1 /Manhattan distance between the tiles and the goal location. How do h_1 and h_2 relate?

- **A. h_2 dominates h_1**
- **B. h_1 dominates h_2 (No: h_1 is a distance where each entry is at most 1, h_2 can be greater)**
- **C. Neither dominates the other**

Break & Quiz

Q 2.2: Consider the state space graph below. Goal states have bold borders. $h(s)$ is show next to each node. What node will be expanded by A* after the initial state I?

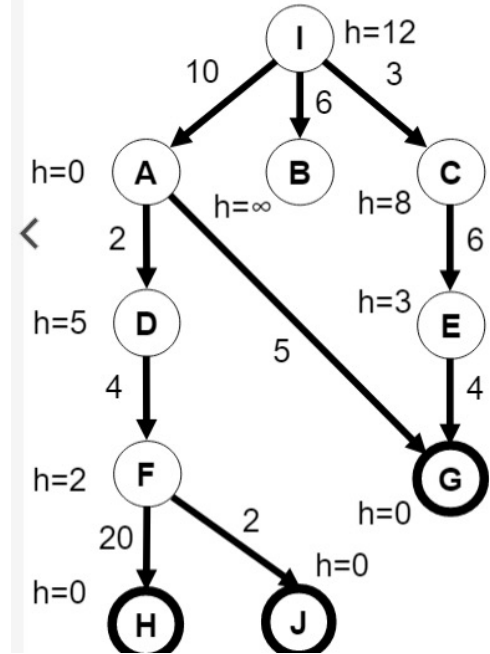
- A. A
- B. B
- C. C



Break & Quiz

Q 2.2: Consider the state space graph below. Goal states have bold borders. $h(s)$ is show next to each node. What node will be expanded by A* after the initial state I?

- **A. A**
- B. B
- C. C



IDA*: Iterative Deepening A*

Similar idea to our earlier iterative deepening.

- Bound the memory in search.
- At each phase, don't expand any node with $g(s) + h(s) > k$,
 - Assuming integer costs, do this for $k=0$, then $k=1$, then $k=2$, and so on
- Complete + optimal, might be costly time-wise
 - Revisit many nodes
- Lower memory use than A*

IDA*: Properties

How many restarts do we expect?

- With integer costs, optimal solution C^* , at most C^*

What about non-integer costs?

- Initial threshold k . Use the same rule for non-expansion
- Set new k to be the min $g(s) + h(s)$ for non-expanded nodes
- Worst case: restarted for each state

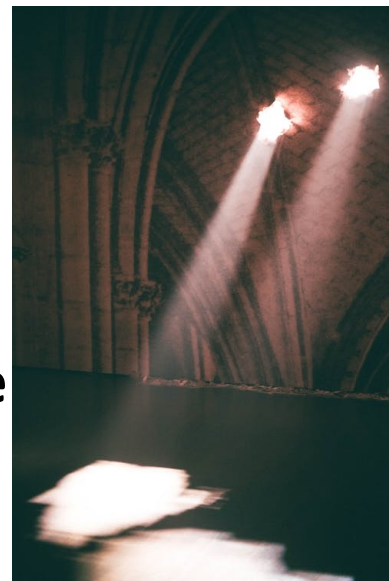
Beam Search

General approach (beyond A* too)

- Priority queue with fixed size k ; beyond k nodes, **discard!**
- **Upside**: good memory efficiency
- **Downside**: not complete or optimal

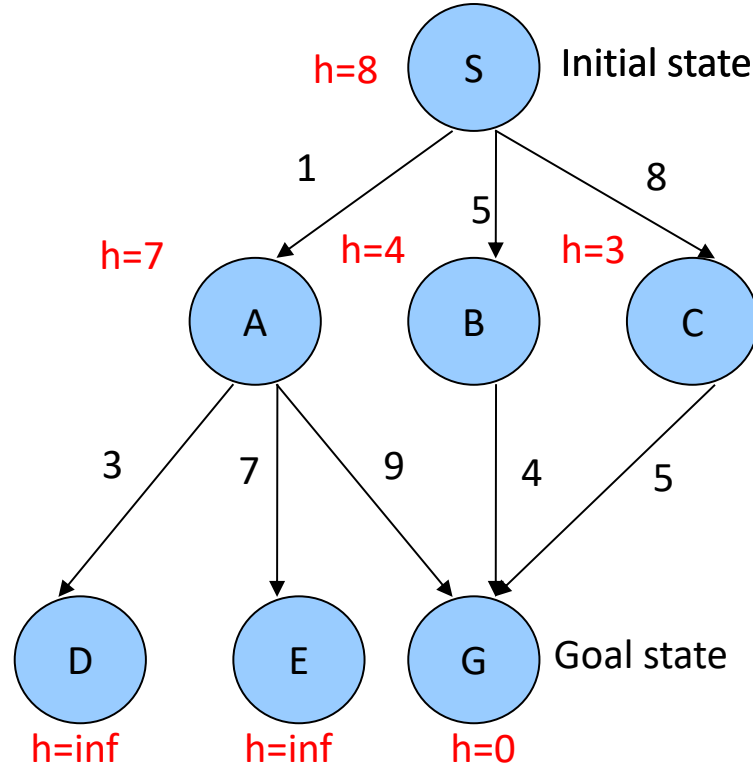
Variation:

- Priority queue with nodes that **are at most ϵ worse** than best node.



Recap and Examples

Example for A*:

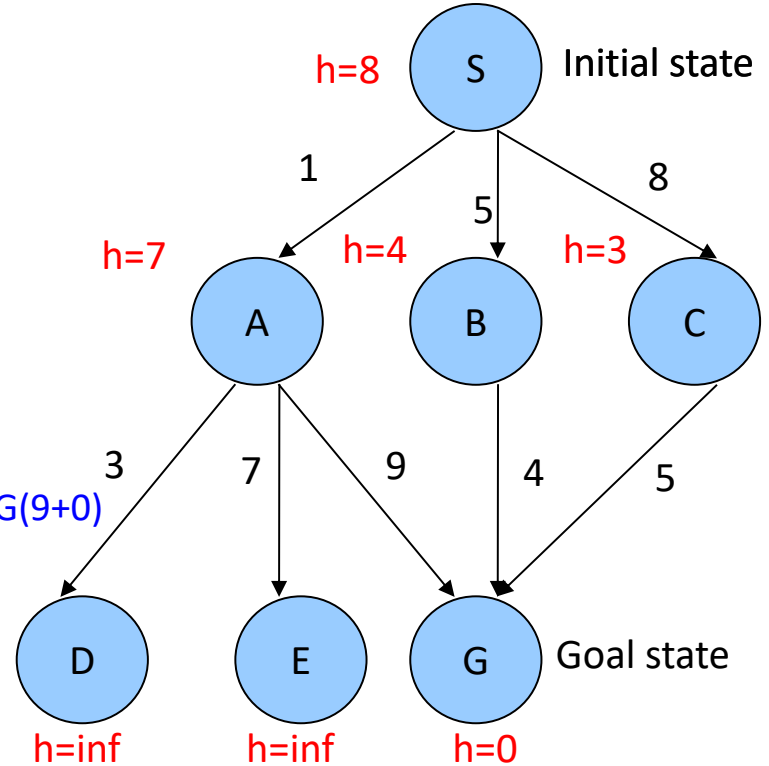


Recap and Examples

Example for A*:

OPEN	CLOSED
S(0+8)	-
A(1+7) B(5+4) C(8+3)	S(0+8)
B(5+4) C(8+3) D(4+inf) E(8+inf) G(10+0)	S(0+8) A(1+7)
C(8+3) D(4+inf) E(8+inf) G(9+0)	S(0+8) A(1+7) B(5+4)
C(8+3) D(4+inf) E(8+inf)	S(0+8) A(1+7) B(5+4) G(9+0)

G → B → S



Recap and Examples

Example for IDA*:

Threshold = 8

PREFIX

-

S

SA

SAH

SAHF

SAD

OPEN

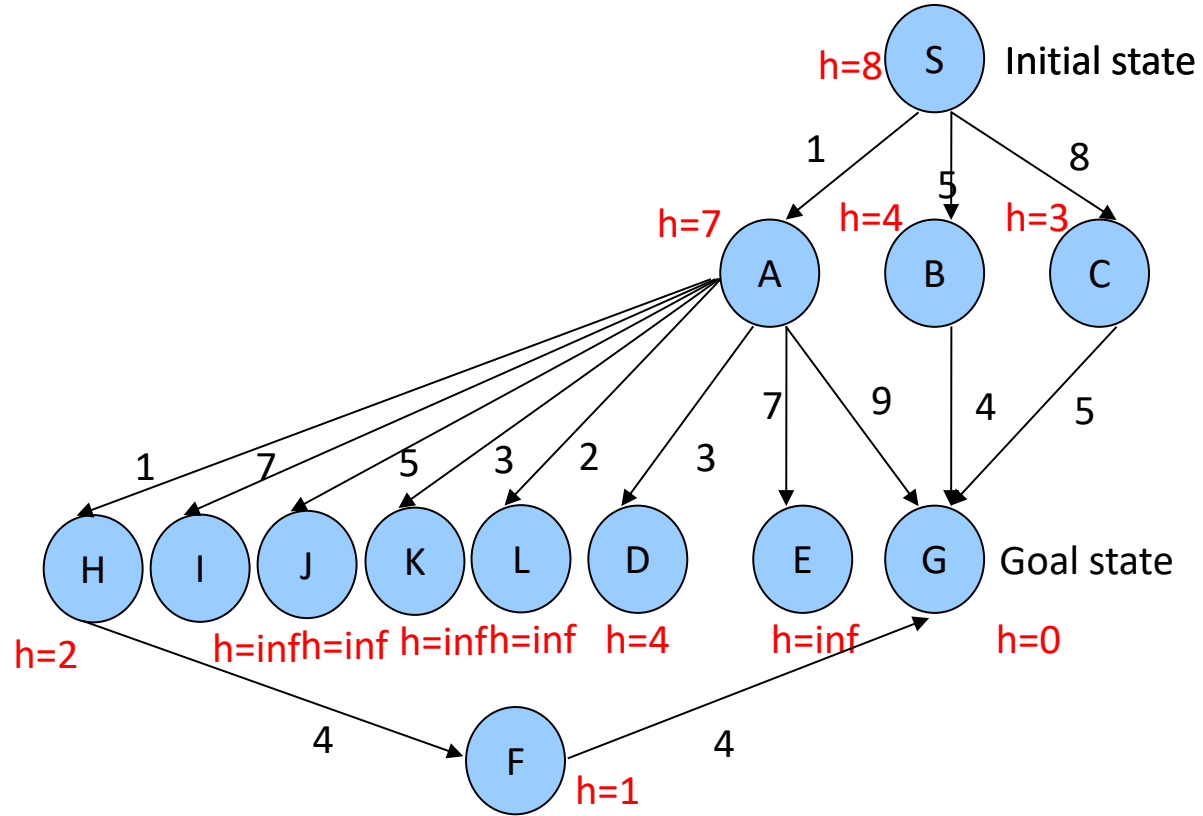
S(0+8)

A(1+7)

H(2+2) D(4+4)

D(4+4) F(6+1)

D(4+4)

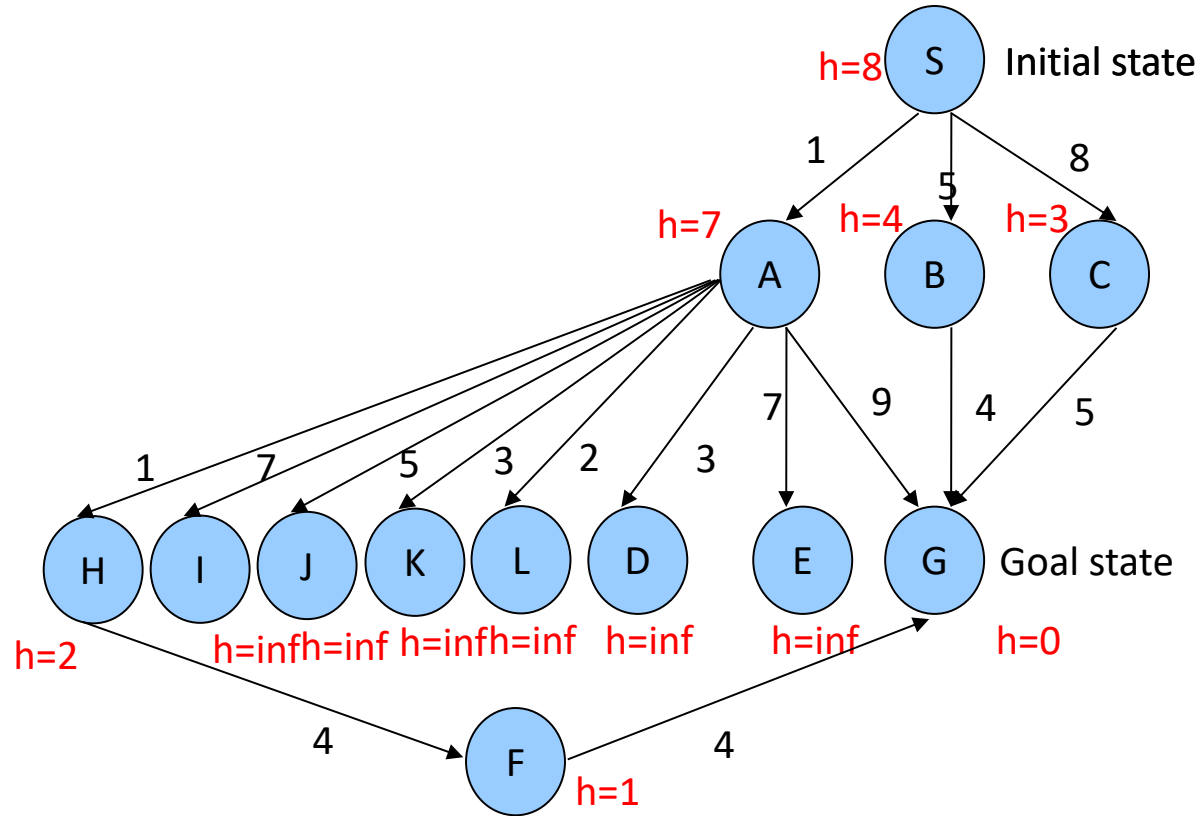


Recap and Examples

Example for IDA*:

Threshold = 9

PREFIX	OPEN
-	S(0+8)
S	A(1+7) B(5+4)
SA	B(5+4) H(2+2) D(4+4)
SAH	B(5+4) D(4+4) F(6+1)
SAHF	B(5+4) D(4+4)
SAD	B(5+4)
SB	G(9+0)
SBG	



Recap and Examples

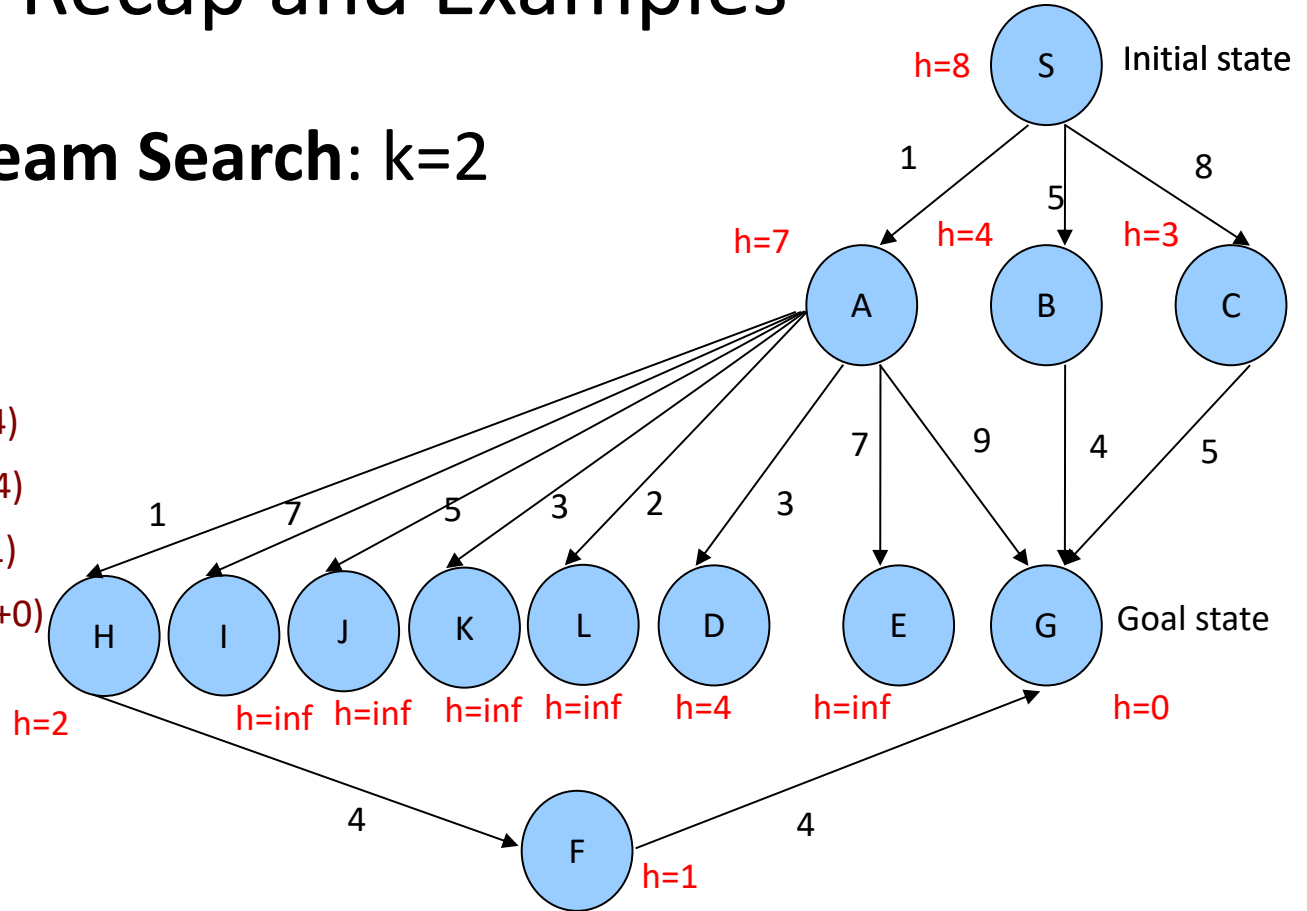
Example for Beam Search: $k=2$

CURRENT

-
S
A
H
F
D
G

OPEN

S(0+8)
A(1+7) B(5+4)
H(2+2) D(4+4)
D(4+4) F(6+1)
D(4+4) G(10+0)
G(10+0)



Summary

- Informed search: introduce heuristics
 - Not all approaches work: best-first greedy is bad
- A* algorithm
 - Properties of A*, idea of admissible heuristics
- Beyond A*
 - IDA*, beam search. Ways to deal with space requirements.