Informed Search

Stephen G. Ware

CSCI 4525 / 5525

Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be the frontier, initially containing
   only the initial state.
3. Loop:
4.   If F is empty, return failure.
5.   Choose a node n to remove from F.
6.   If n is a solution, return n.
7.   Add n to V.
8.   For every successor s of n not in V or F:
9.     Add s to F.
```

Uninformed vs. Informed

- Uninformed search considers only information about the past (work already done).
- Informed search also estimates information about the future (work still to be done).

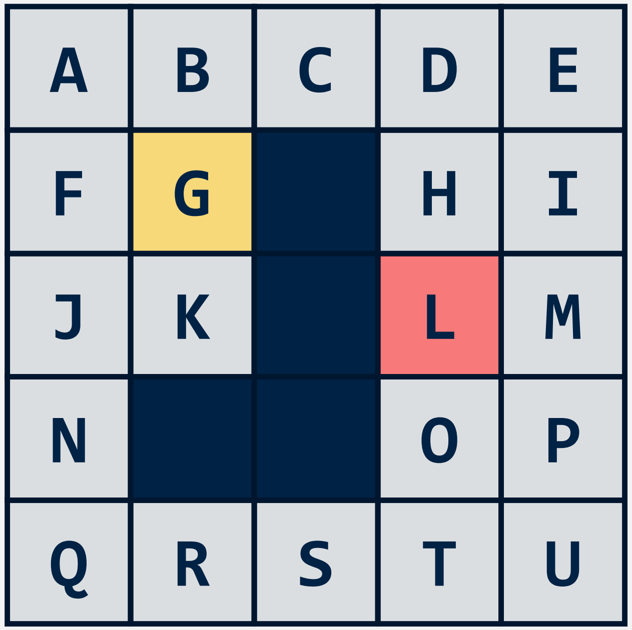
Heuristics

A **heuristic** is a function *h* which, given some state, estimates the amount of work left to do before reaching a goal state.

Consider driving from one city to another.

The straight line distance between two cities in miles (distance "as the crow flies") provides a good estimate of how many miles are left to drive on the actual roads before reaching the destination.

Grid World Heuristic



Given x and y coordinates:

A = (0,0)

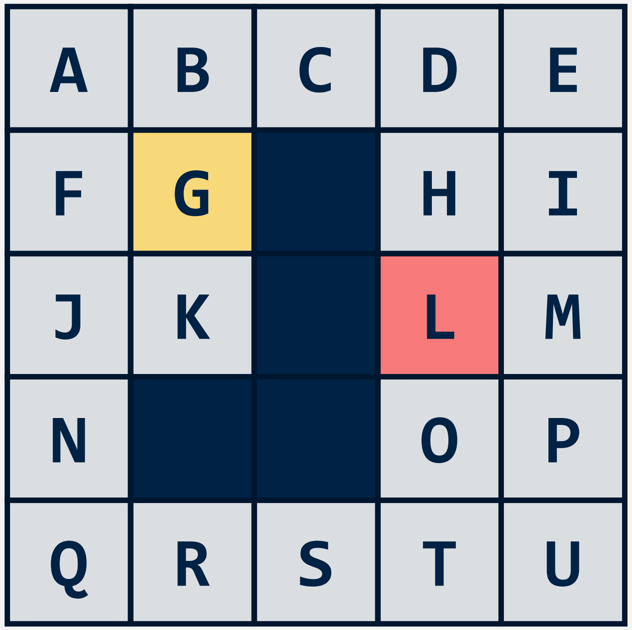
B = (0,1)

C = (0,2)

…

U = (4,4)

Grid World Heuristic



Manhattan Distance:

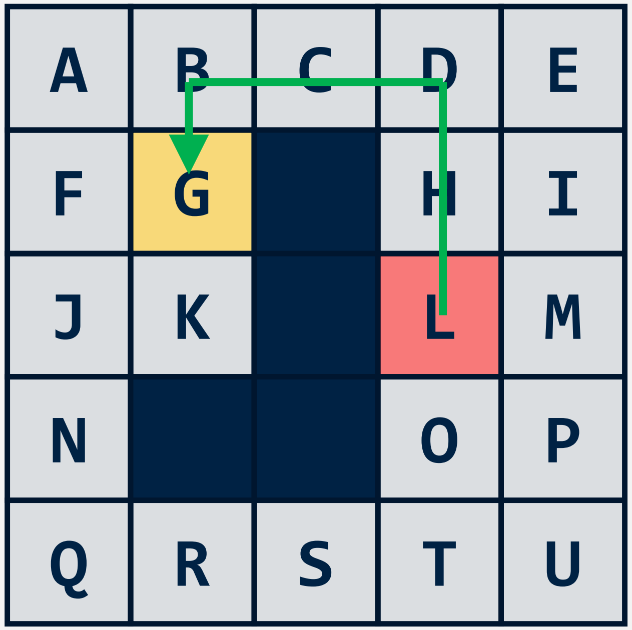
(also called taxicab distance)

Given two squares

$X = (x_1, y_1)$ and $Y = (x_2, y_2)$

$h(X,Y) = |x_1 - x_2| + |y_1 - y_2|$

Grid World Heuristic



Manhattan Distance:

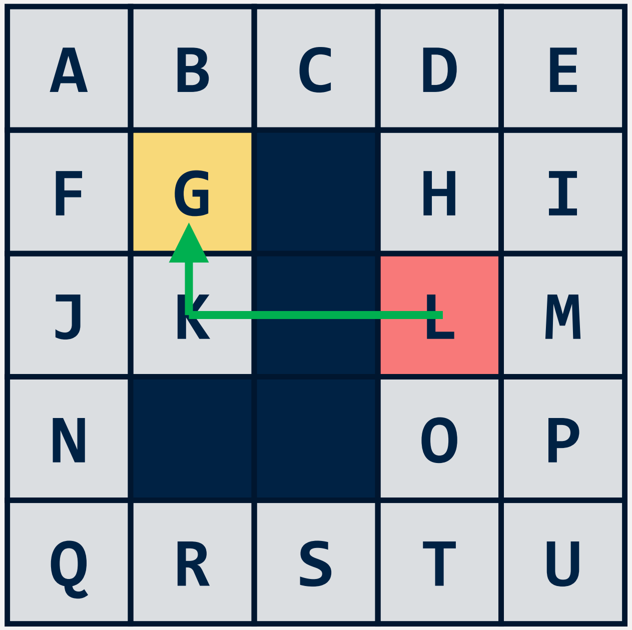
(also called taxicab distance)

Example:

L = (3,2) and **G** = (1,1)

d(**L**,**G**) = 5

Grid World Heuristic



Manhattan Distance:

(also called taxicab distance)

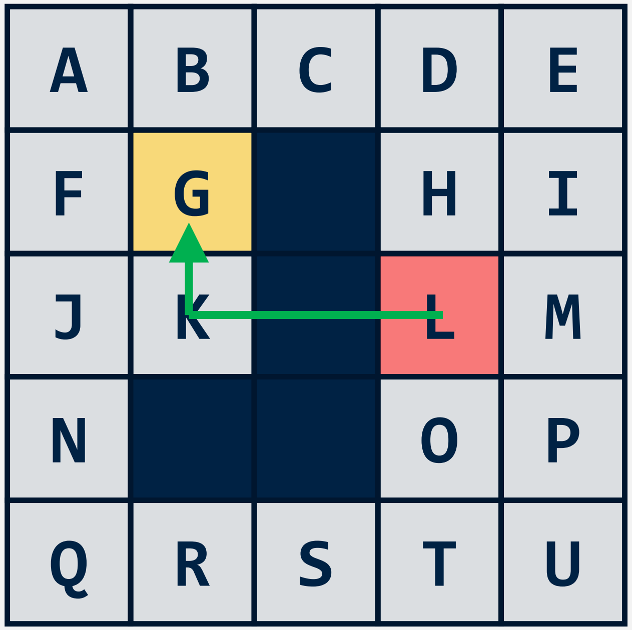
Example:

L = (3,2) and **G** = (1,1)

d(**L**,**G**) = 5

h(**L**,**G**) = |3-1| + |2-1| = 3

Grid World Heuristic



Manhattan Distance:

(also called taxicab distance)

- Ignores obstacles.
- May underestimate the actual distance, but that’s ok!

Best First Search

When choosing which state to consider next, use a heuristic to estimate how much work is left to do before the goal is reached.

Informed Search

- Greedy Search
- A* Search

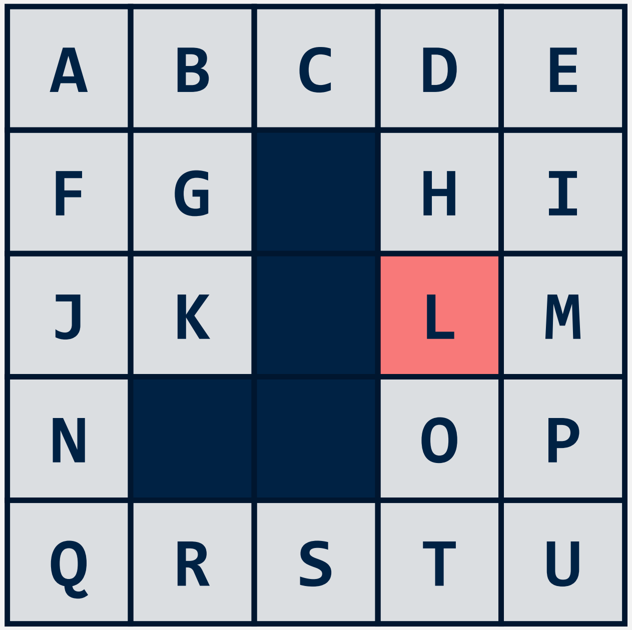
Both are kinds of Best First Search.

Greedy Search

When choosing the next state, always choose the one with the lowest heuristic value.

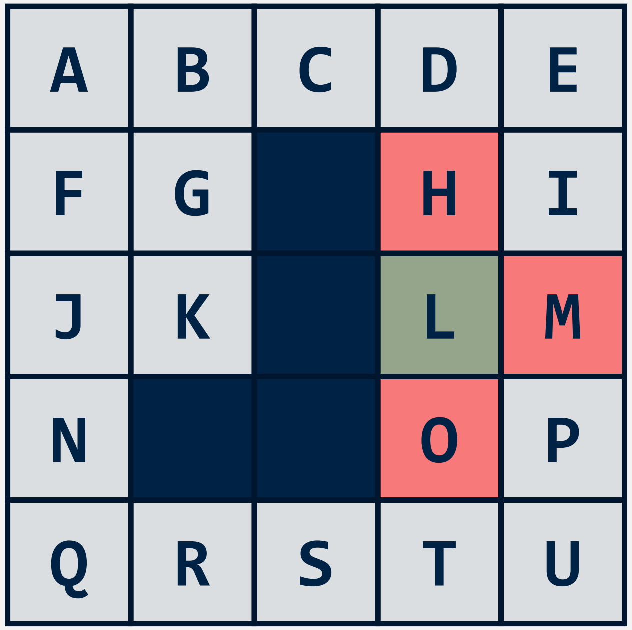
i.e. always choose the state which (we estimate) is closest to the goal.

Greedy Search



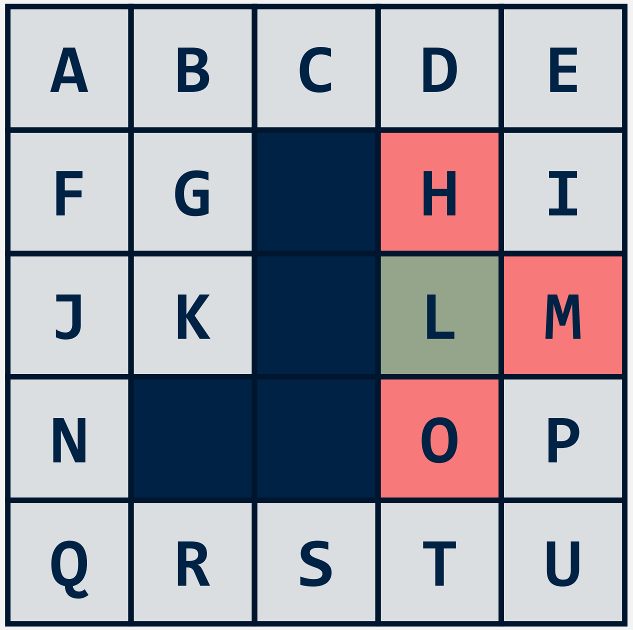
Visited:
Frontier: L

Greedy Search



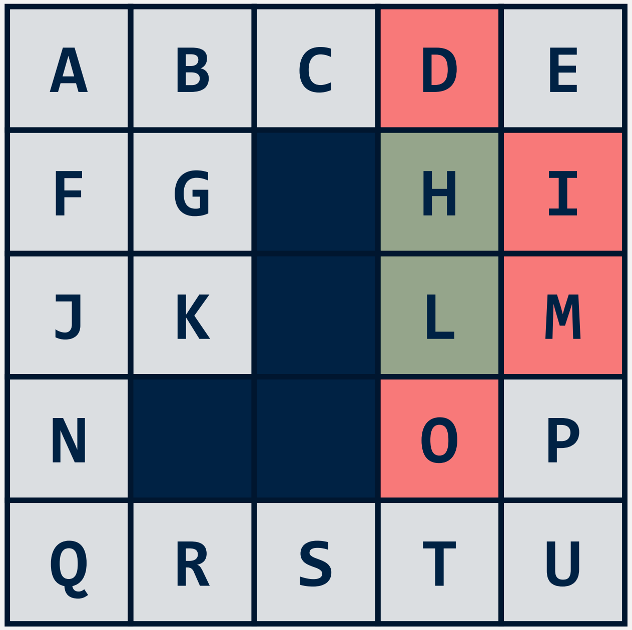
Visited: L
Frontier: H,M,O

Greedy Search



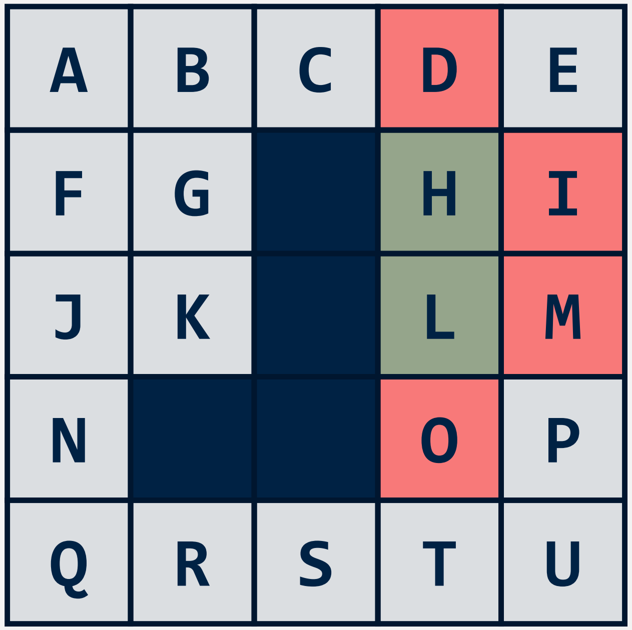
Visited: L
Frontier: H,M,O
h(H,G)=2
h(M,G)=4
h(O,G)=4

Greedy Search



Visited: L,H
Frontier: M,O,D,I

Greedy Search



Visited: L,H

Frontier: M,O,D,I

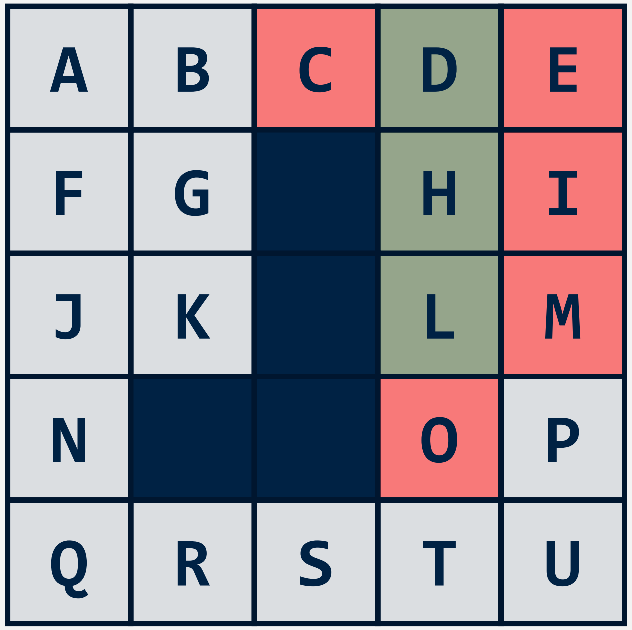
h(M)=4

h(O)=4

h(D)=3

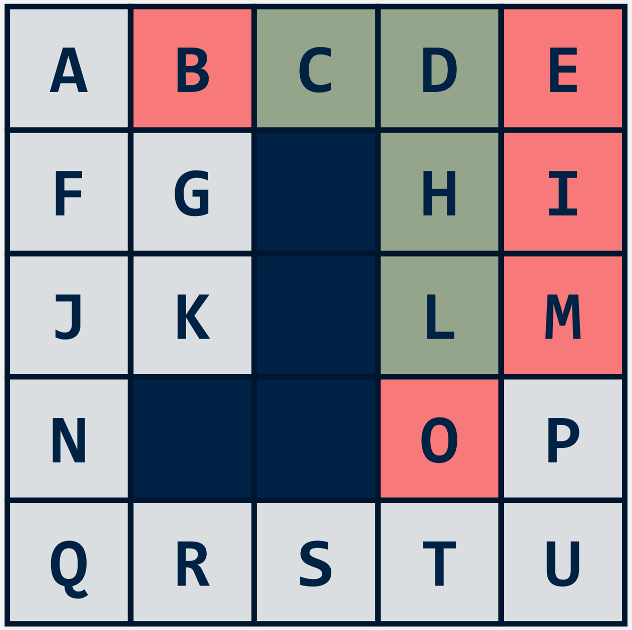
h(I)=3

Greedy Search



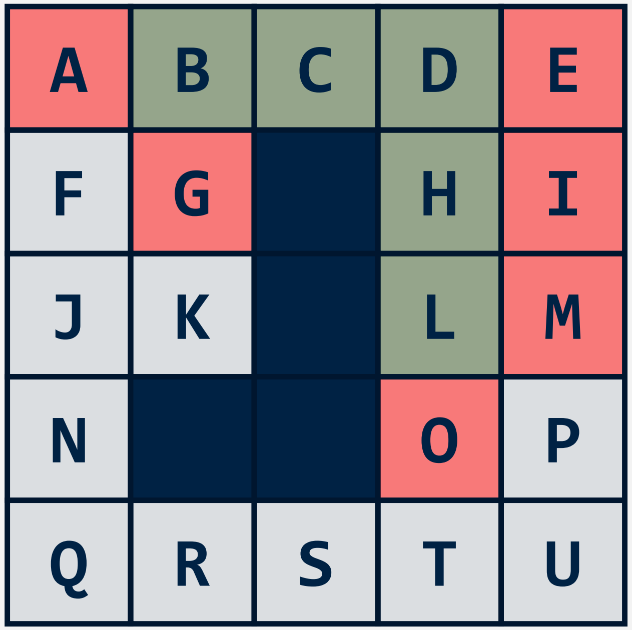
Visited: L,H,D
Frontier: M,O,I,C,E

Greedy Search



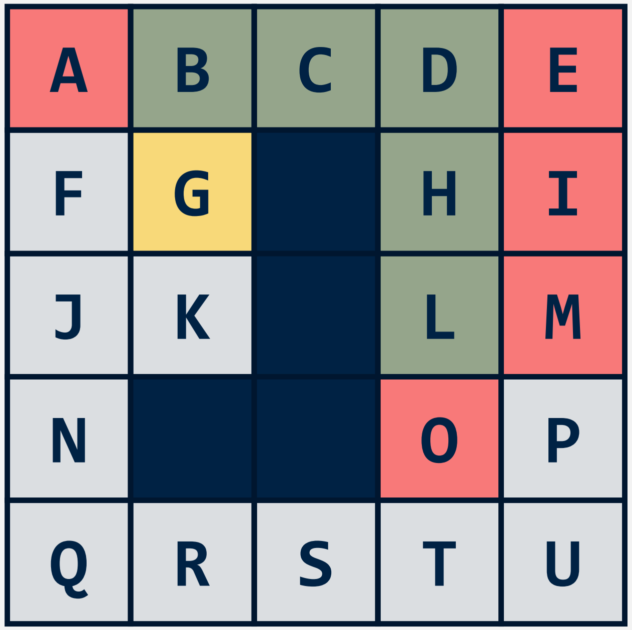
Visited: L,H,D,C
Frontier: M,O,I,E,B

Greedy Search



Visited: L,H,D,C,B
Frontier: M,O,I,E,A,G

Greedy Search



Visited: L,H,D,C,B
Frontier: M,O,I,E,A
Goal found!

Best First Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be the frontier, initially containing
   only the initial state.
3. Loop:
4.   If F is empty, return failure.
5.   Choose a node n to remove from F.
6.   If n is a solution, return n.
7.   Add n to V.
8.   For every successor s of n not in V or F:
9.     Add s to F.
```

Implement F as a min priority queue.

Properties of Greedy Search

If we don't keep track of visited states (i.e. we might repeat some steps), how will greedy search behave?

It may not find a solution!

In other words, greedy search is only **complete** if we make sure not to repeat steps.

In an infinite space, greedy is incomplete.

Complexity of Greedy Search

Given a state space in which...

- each node has b successors,
- the shortest path to the solution has d steps,
- the longest path in the space is m long

What is the time complexity?

$O(b^m)$

Complexity of Best First Search

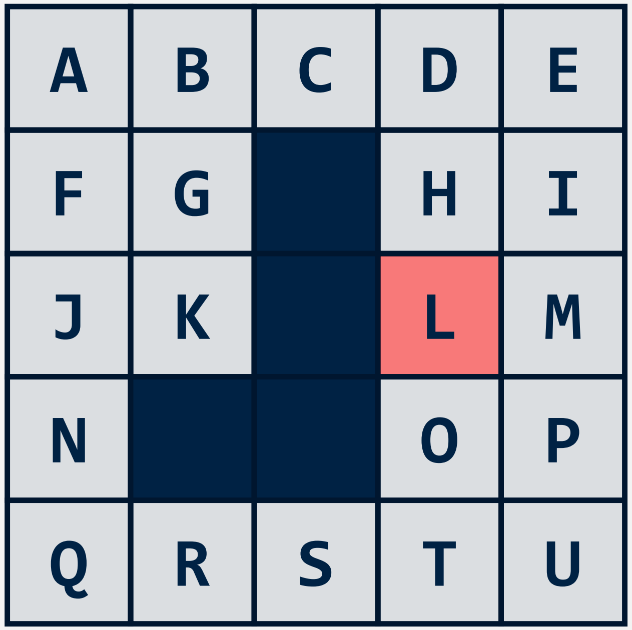
Though the worst case may not be better than uninformed search, in practice, best first search is usually better because the heuristic helps us avoid the worst case most of time.

A* Search

When choosing the next state, choose the one for which the sum of the work done so far plus the estimated work remaining is lowest.

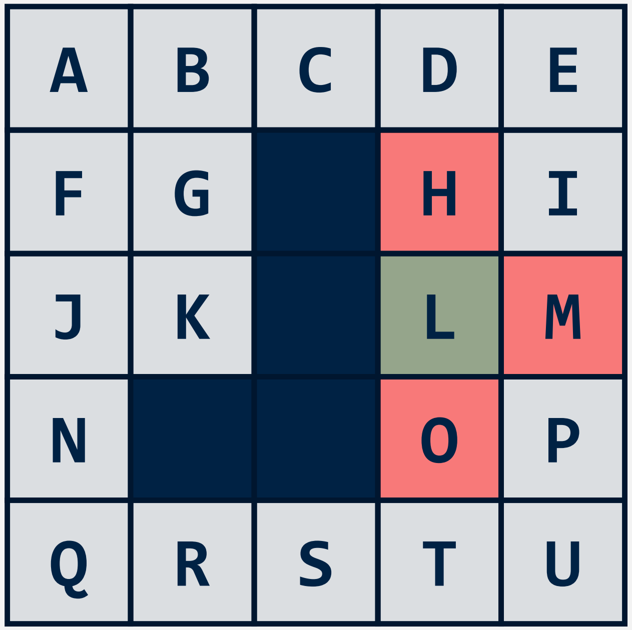
i.e. consider both how far we have come and how far we think we have left to go.

A* Search



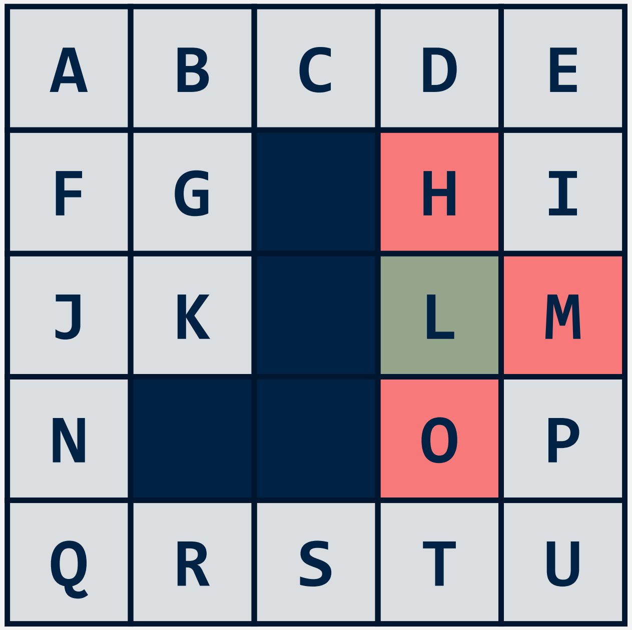
Visited:
Frontier: L

A* Search



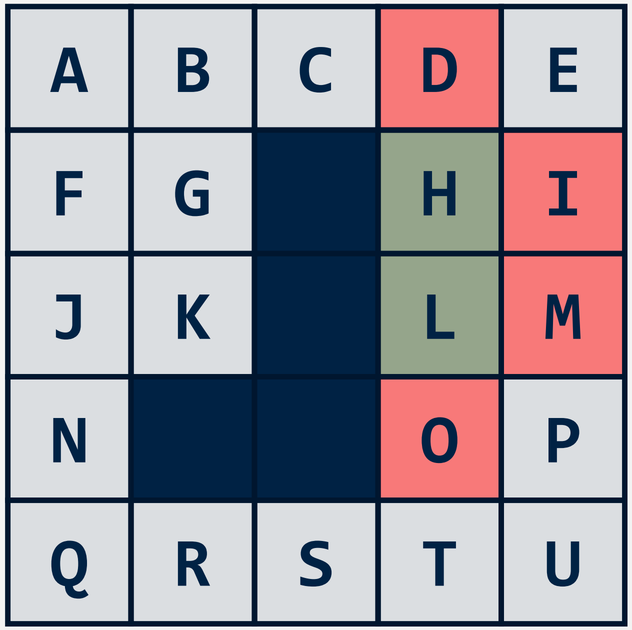
Visited: L
Frontier: H,M,O

A* Search



Visited: L
Frontier: H,M,O
d(L,H)=1 h(H,G)=2 d+h=3
d(L,M)=1 h(M,G)=4 d+h=5
d(L,O)=1 h(O,G)=4 d+h=5

A* Search

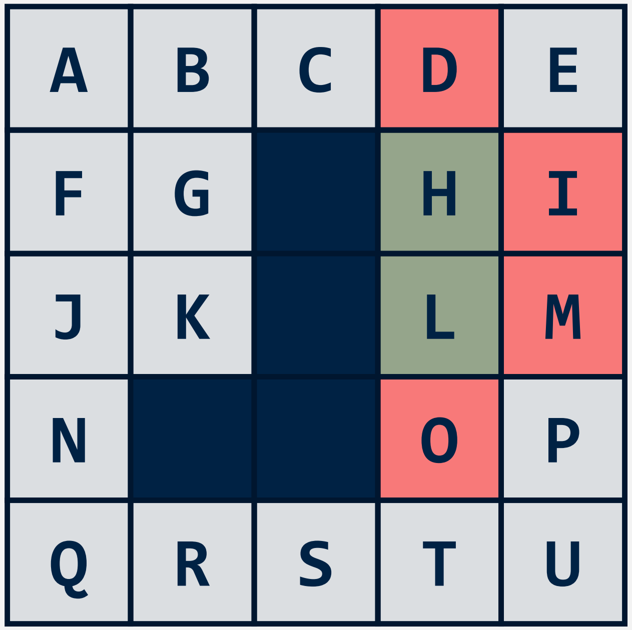


Visited: L,H

Frontier: M,O,D,I

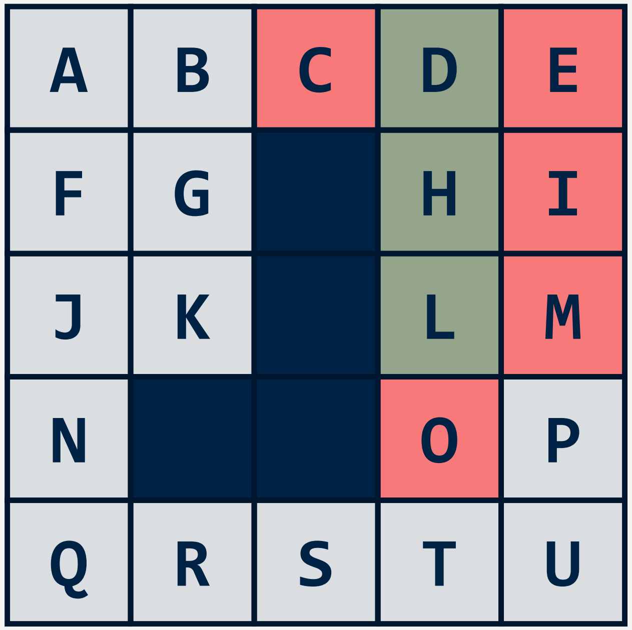
NIL

A* Search



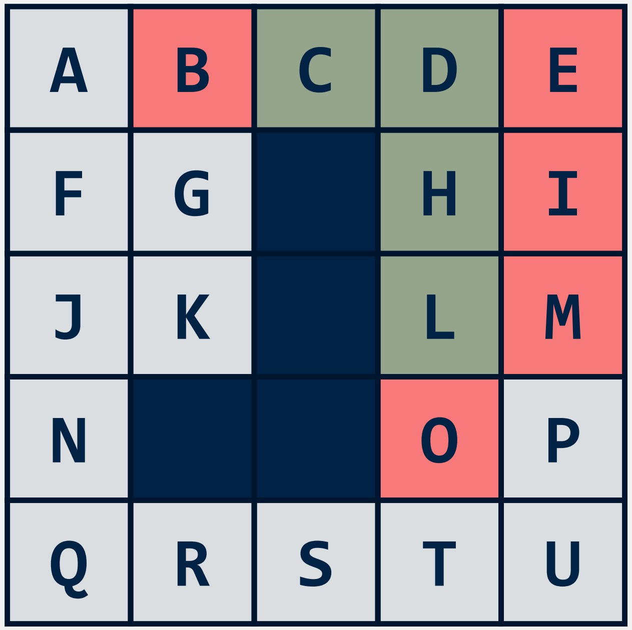
Visited: L,H
Frontier: M,O,D,I
d(L,M)=1 h(M,G)=4 d+h=5
d(L,O)=1 h(O,G)=4 d+h=5
d(L,D)=2 h(D,G)=3 d+h=5
d(L,I)=2 h(I,G)=3 d+h=5

A* Search



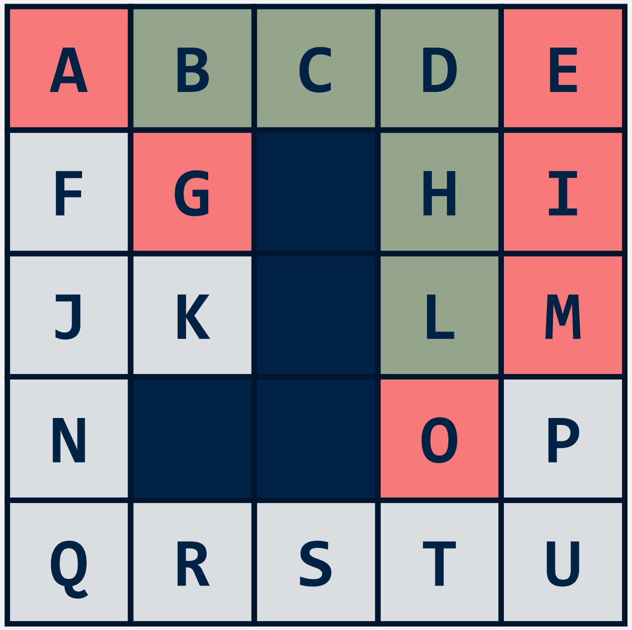
Visited: L,H,D
Frontier: M,O,I,C,E

A* Search



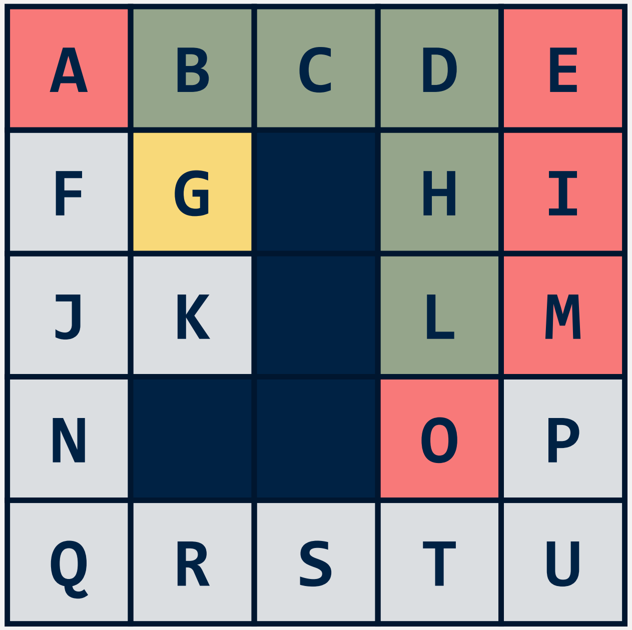
Visited: L,H,D,C
Frontier: M,O,I,E,B

A* Search



Visited: L,H,D,C,B
Frontier: M,O,I,E,A,G

A* Search



Visited: L,H,D,C,B
Frontier: M,O,I,E,A
Goal found!

A* and Its Heuristic

If a heuristic can guarantee that it will never overestimate, it is called **admissible**.

When used with an admissible heuristic, A*:

- is **complete** (even in infinite spaces)
- always returns the **optimal** solution

Designing Heuristics

When designing heuristics, we often face a tradeoff between the two most important features:

- **Accuracy:** How well it estimates remaining work.
- **Speed:** How fast it can be computed.

More accurate heuristics take longer to compute, while easily computed heuristic are often inaccurate.

Designing Heuristics

Most heuristics are based on solving a **relaxed** (simpler) version of the real problem.

For example, Manhattan distance ignores barriers in the search space. This means it may underestimate, but now it can be computed much faster than the actual distance to the goal.

Designing Heuristics

In large search spaces (i.e. most real world problems), accuracy is more valuable than speed, because the problem is intractable anyway.

Small improvements to the accuracy of the heuristic can save an exponential amount of work.

Designing Heuristics

Many heuristics sacrifice admissibility for accuracy and speed.

Such heuristics do not guarantee an optimal solution, but this is often acceptable, since an optimal solution would take too long to find anyway.

Satisficing = good enough

Best First Search

```
1. Let V be the set of visited nodes, empty.
2. Let F be the frontier, initially containing
   only the initial state.
3. Loop:
4.   If F is empty, return failure.
5.   Choose a node n to remove from F.
6.   If n is a solution, return n.
7.   Add n to V.
8.   For every successor s of n not in V or F:
9.     Add s to F.
```

Implement F as a min priority queue.

General Search Algorithm

If we use a min priority queue for the frontier, we can do most kinds of search by simply changing how we calculate the key of a node in the queue.

- d = number of steps taken so far (e.g. roads traveled)
- w = work done so far (e.g. miles traveled)
- h = estimated work remaining (e.g. miles remaining)

General Search Algorithm

How should we calculate the key of a node in the min priority queue to get…

• Breadth First Search?	d
• Depth First Search?	$-d$
• Uniform Cost Search?	w
• Greedy Search?	h
• A* Search?	$w + h$

Complexity of Best First Search

If we don't keep track of which nodes have been visited (i.e. we treat the search space as a tree), is A* still complete and optimal?

Yes!

Because A* considers the distance traveled so far, it naturally penalizes paths with loops. Thus, many implementations of A* do not use **V**.

Complexity of Best First Search

Even if we don't keep track of which nodes have already been visited, we still usually run out of memory before we run out of time.

How can we save memory?

Memory-Bounded A* Search

- Set a maximum size for the frontier.
- When the frontier gets full, remove the worst node (i.e. the nodes with the highest $d + h$ value).
- Before removing ("forgetting") a node, have its parent remember its value. A parent only remembers the value of its best forgotten child.

Memory-Bounded A* Search

- Later on, it might turn out that the best path is one that we have forgotten.
- By having the parent remember the value of its best forgotten child, we will know when this happens.
- We can always go back and re-explore the forgotten part of the space. This means some time is wasted re-doing past work (i.e. there is a tradeoff between space and time).