# Chapter 2 State-Space Search

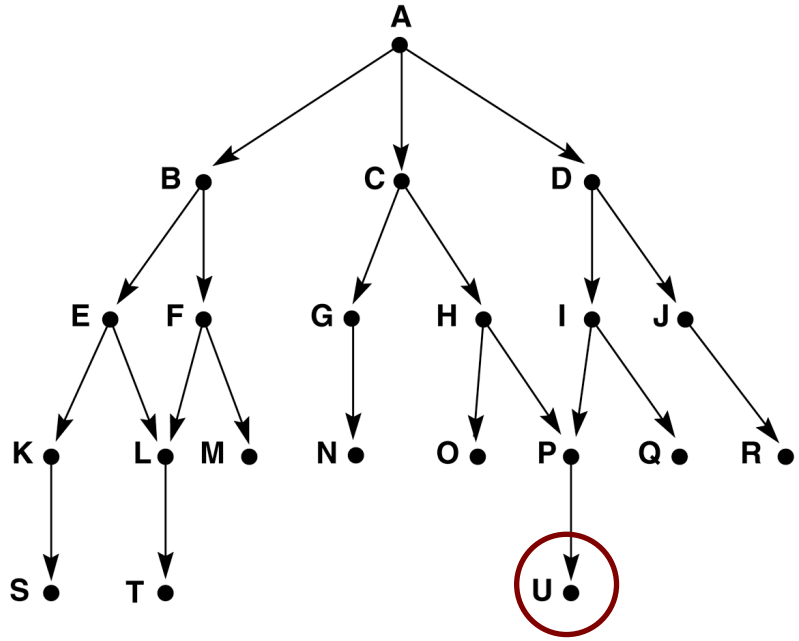
## COMP 6721 Introduction of AI

*Russell & Norvig – Chapter 3 & Section 4.11*

***Some slides from:*** robotics.stanford.edu/~latombe/cs121/2003/home.htm

# Breadth-First Search Example

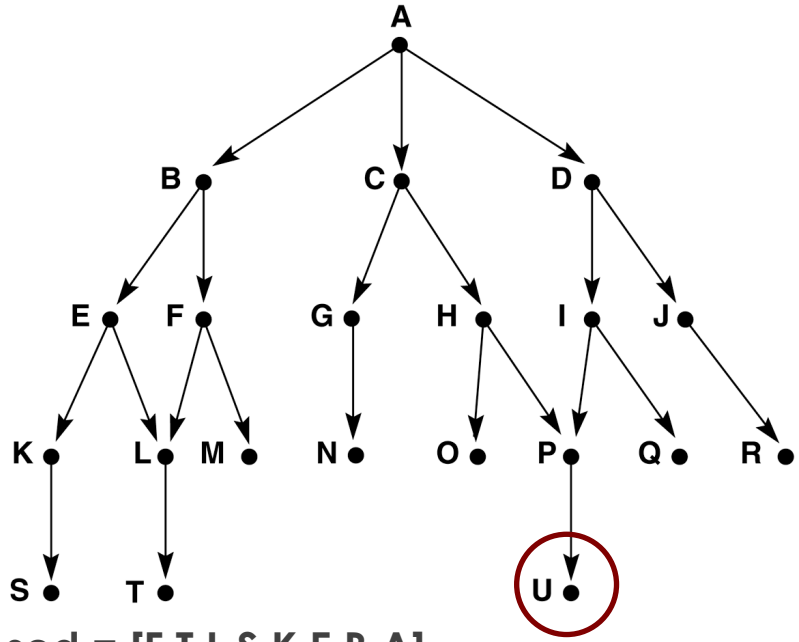
- BFS uses a ***queue***.
- Assume U is goal State.




1. open = [A-null] closed = []
2. open = [B-A C-A D-A] closed = [A]
3. open = [C-A D-A E-B F-B] closed = [B A]
4. open = [D-A E-B F-B G-C H-C] closed = [C B A]
5. open = [E-B F-B G-C H-C I-D J-D] closed = [D C B A]
6. open = [F-B G-C H-C I-D J-D K-E L-E] closed = [E D C B A]
7. open = [G-C H-C I-D J-D K-E L-E M-F] ***as L is already on open*** closed = [F E D C B A]
8. and so on until either U is found or open = []

# Depth-First Search Example

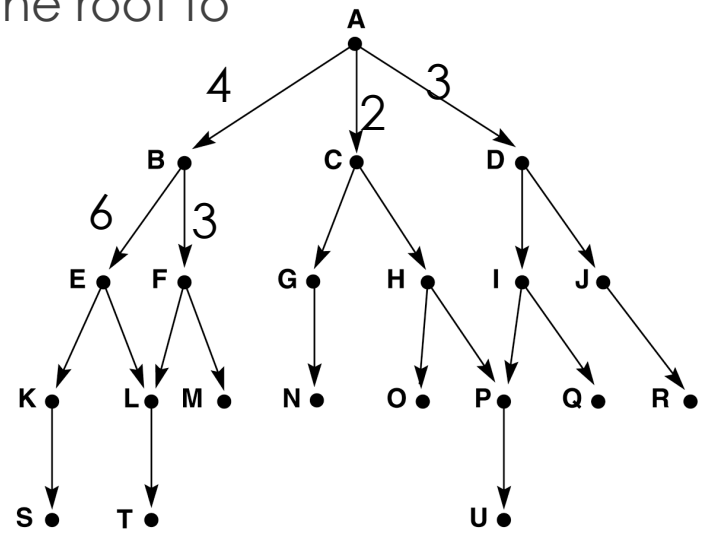
- DFS uses a ***stack***.
- Assume U is goal State.




1. open = [A-null] closed = []
2. open = [B-A C-A D-A] closed =[A]
3. open = [E-B F-B C-A D-A] closed = [B A]
4. open = [K-E L-E F-B C-A D-A] closed = [E B A]
5. open = [S-K L-E F-B C-A D-A] closed = [K E B A]
6. open = [L-E F-B C-A D-A] closed = [S K E B A]
7. open = [T-L F-B C-A D-A] closed = [L S K E B A]
8. open = [F-B C-A D-A] closed = [T L S K E B A]
9. open = [M-F C-A D-A] *as L is already on closed* closed = [F T L S K E B A]
10. open = [C-A D-A] closed = [M F T L S K E B A]
11. open = [G-C H-C D-A] closed = [C M F T L S K E B A]
12. open = [N-G, H-C, D-A] closed = [G C M F T L S K E B A]
13. open =[O-H, P-H, D-A] closed = [H N G C M F T L S K E B A]
14. open = [P-H, D-A] closed= [H N G C M F T L S K E B A]
15. open = [D-A] closed= [U P O H N G C M F T L S K E B A]

# Uniform Cost Search

- Breadth First Search
  - uses a priority queue sorted using the depth of the nodes from the root
  - guarantees to find the shortest solution path
- But what if all edges/moves do not have the same cost?
- Uniform Cost Search
  - uses a priority queue sorted using the cost from the root to node n – later called g(n)
  - guarantees to find the lowest cost solution path



# Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - Informed Search
    - Hill Climbing
    - Best – Frist
    - Design Heuristics
    - A*

# Informed Search (Heuristic Search)

- Most of the time, it is not feasible to do an exhaustive search, search space is too large
- so far, all search algorithms have been uninformed (general search)
- so need an ***informed/heuristic search***

- Idea:
  - choose "best" next node to expand
  - according to a selection function (heuristic)
- But: heuristic might fail

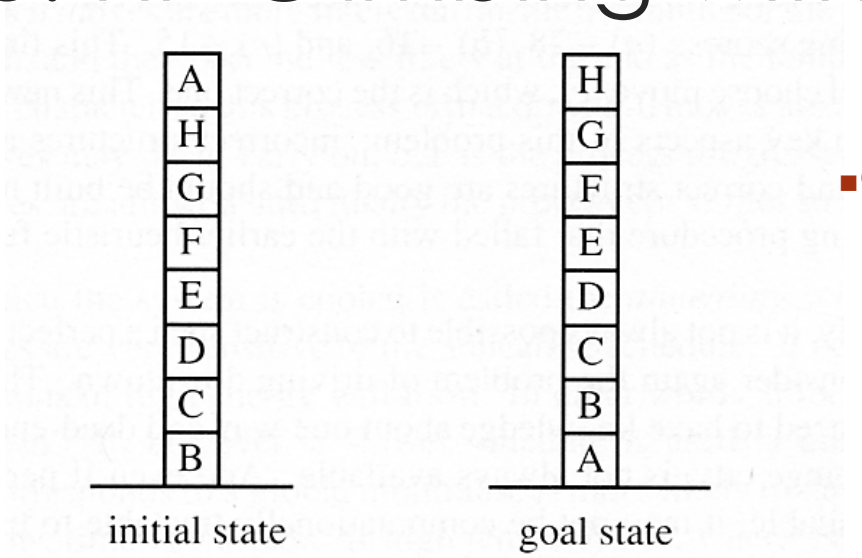
# Heuristic – Heureka!


- Heuristic:
  - a rule of thumb, a good bet
  - but has no guarantee to be correct whatsoever!
- Heuristic search:
  - A technique that improves the efficiency of search, possibly sacrificing on completeness
  - Focus on paths that seem most promising according to some function
  - Need an evaluation function (heuristic function) to score a node in the search tree
- Heuristic function h(n) = an ***approximation*** to a function that gives the true evaluation of the node n

# Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - Informed Search
    - ***Hill Climbing***
    - Best – Frist
    - Design Heuristics
    - A*

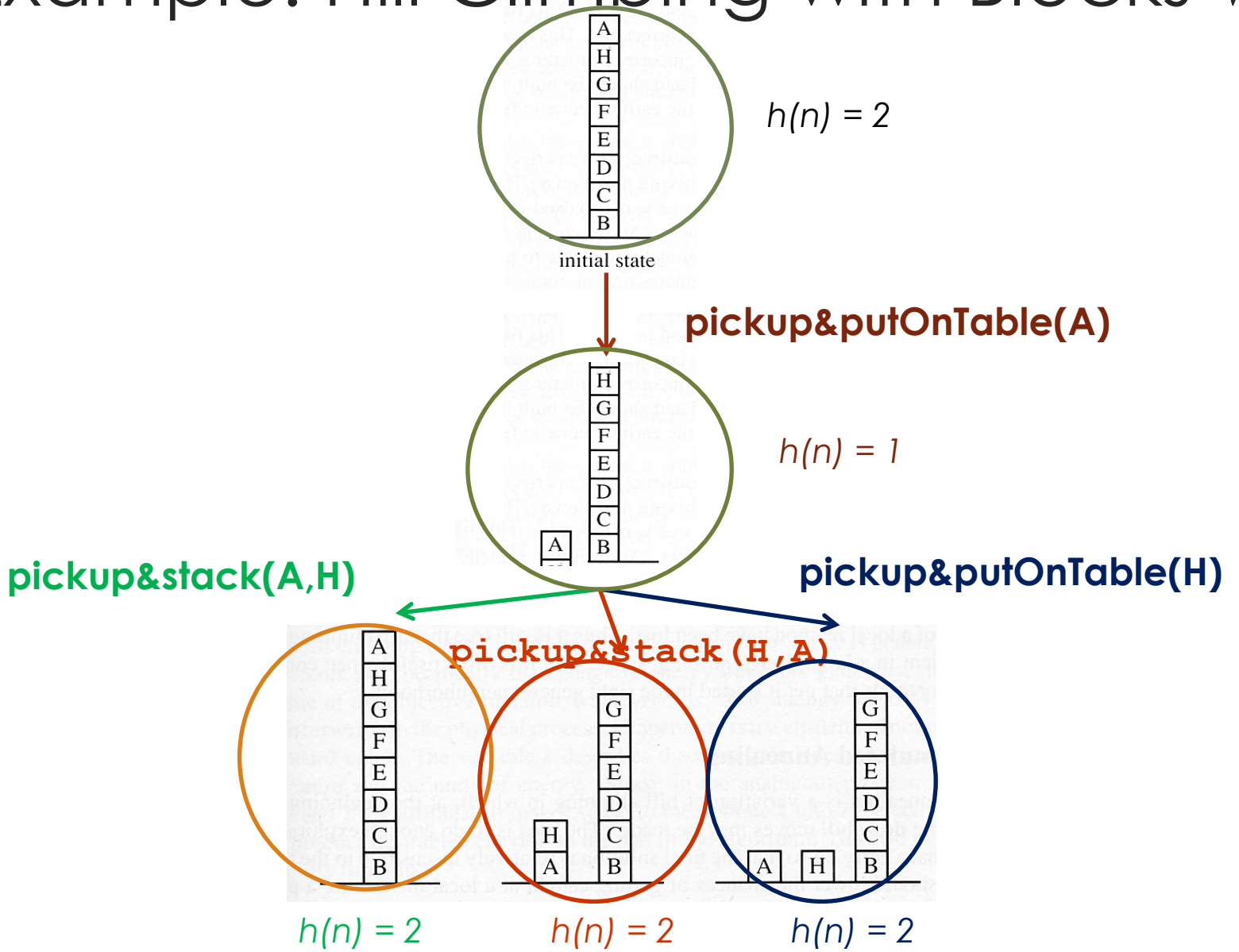
# Example: Hill Climbing with Blocks World




- Operators:
  - pickup&putOnTable(Block)
  - pickup&stack(Block1,Block2)

- Heuristic:
  - - 1pt if a block is sitting where it is supposed to sit
  - +1pt if a block is NOT sitting where it is supposed to sit
  - so lower h(n) is better
    - h(initial) = 2
    - h(goal) = 0

# Example: Hill Climbing with Blocks World



# Hill Climbing

- General hill climbing strategy:
  - as soon as you find a position that is better than the current one, select it.
  - Does not maintain a list of next nodes to visit (an *open* list)
  - Similar to climbing a mountain in the fog with amnesia … always go higher than where you are now,

    but never go back…


- Steepest ascent hill climbing:
  - instead of moving to the first position that

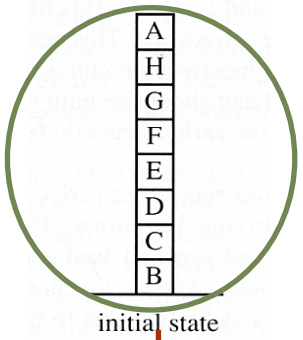
    is better than the current one
  - pick the best position out of all the next possible moves

# Steepest Ascent Hill Climbing

```
currentNode = startNode;
    loop do
        L = CHILDREN(currentNode);
        nextEval = -INFINITY;
        nextNode = NULL;

        for all c in L
          if (HEURISTIC-VALUE(c) < nextEval) // lower h is better
              nextNode = c;
              nextEval = HEURISTIC-VALUE(c);
         if nextEval >= HEURISTIC-VALUE(currentNode)
              return currentNode; // Return current node since no better child state exist
         currentNode = nextNode;
```

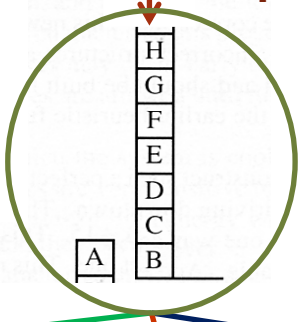
# Example: Hill Climbing with Blocks World




*h(n) = 2*

**pickup&putOnTable(A)**

hill-climbing will stop, because all children are higher than parent… --> local minimum

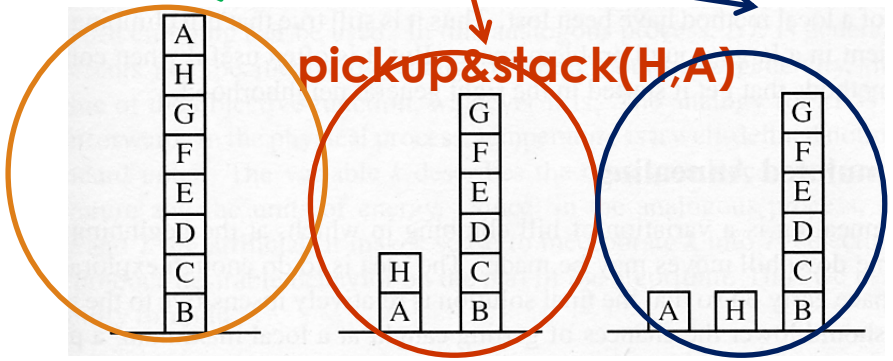



*h(n) = 1*

**pickup&stack(A,H)**

**pickup&putOnTable(H)**

**pickup&stack(H,A)**



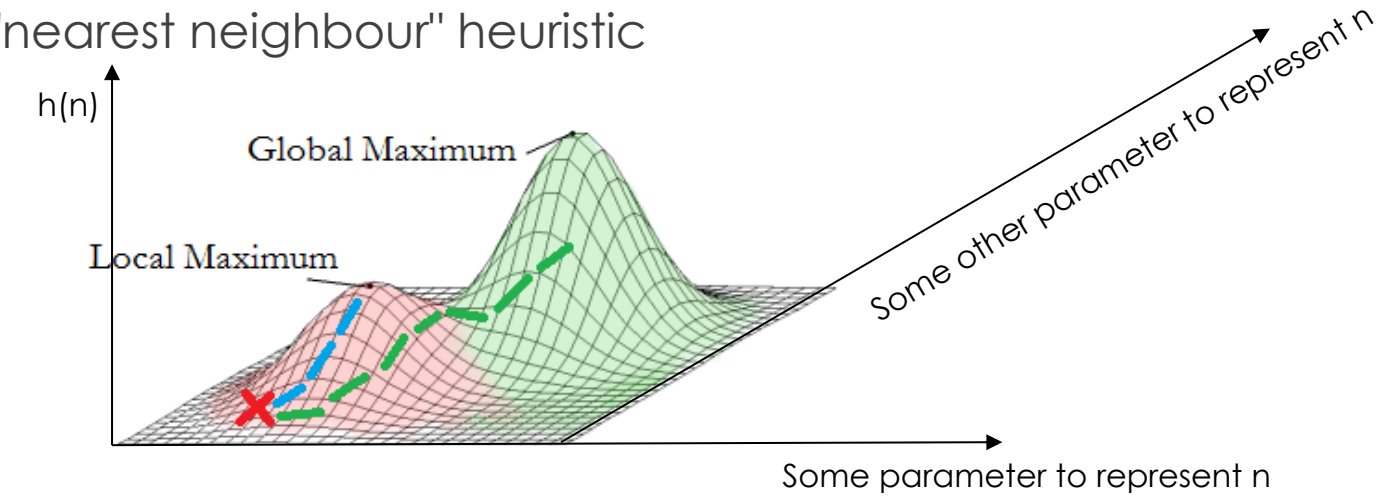

*h(n) = 2* *h(n) = 2* *h(n) = 2*

Don't be confused…
a lower h(n) is better…

So with lower h(n), *hill-climbing* should really be called *deep-diving* ☺

# Problems with Hill Climbing

- Foothills (or local maxima)
  - reached a local maximum, not the global maximum
  - a state that is better than all its neighbors but is not better than some other states farther away.
  - at a local maximum, all moves appear to make things worse.
  - ex: 8-puzzle: we may need to move tiles temporarily out of goal position in order to place another tile in goal position
  - ex: "nearest neighbour" heuristic

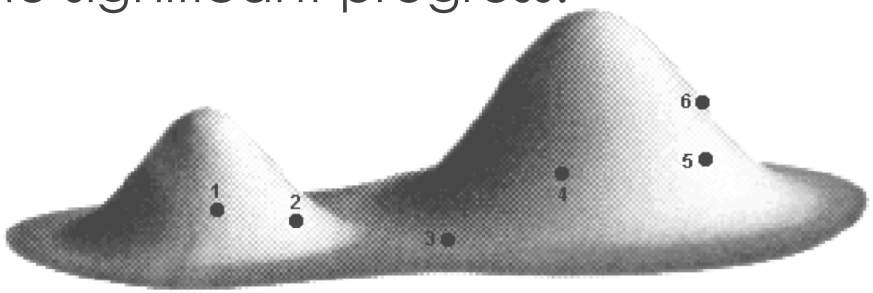


# Problems with Hill Climbing

- Plateau
  - a flat area of the search space in which the next states have the same value.
  - it is not possible to determine the best direction in which to move by making local comparisons.

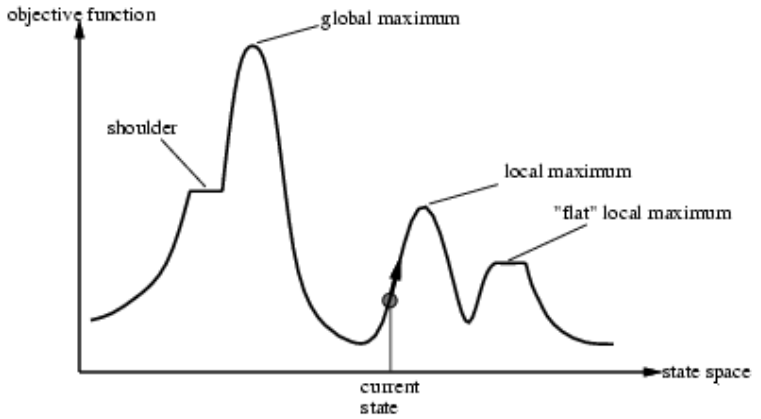
# Solutions to Hill Climbing

- Random-restart hill-climbing
  - random initial states are generated
  - run each until it halts or makes no significant progress.
  - the best result is then chosen.




- keep going even if the best successor has the same value as current node
  - works well on a "shoulder"
  - but could lead to infinite loop

    on a plateau



# Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - Informed Search
    - ***Hill Climbing***
    - ***Best – Frist***
    - Design Heuristics
    - A*

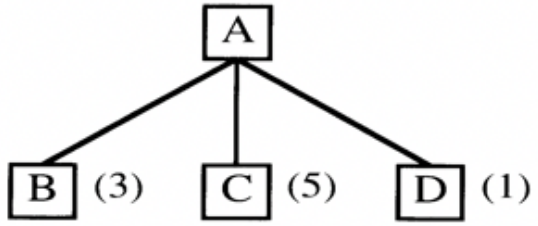
# Best – First Search

- Problem with hill-climbing:
  - one move is selected and all others are forgotten.
- Solution to hill-climbing:
  - use "open" as a priority queue
  - this is called best-first search
- Best-first search:
  - Insert nodes in *open* list so that the nodes are sorted in best (ascending/descending) **h(n)**
  - Always choose the next node to visit to be the one with the best h(n) -- regardless of where it is in the search space

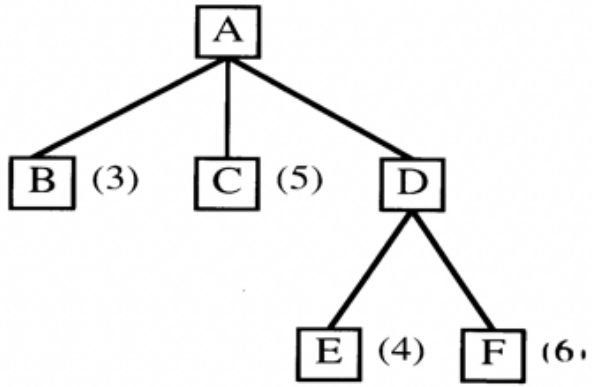
# Best – First Search Example

Step 1


Step 2

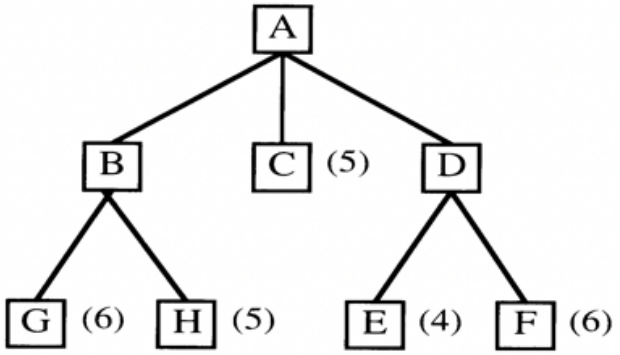



Step 3

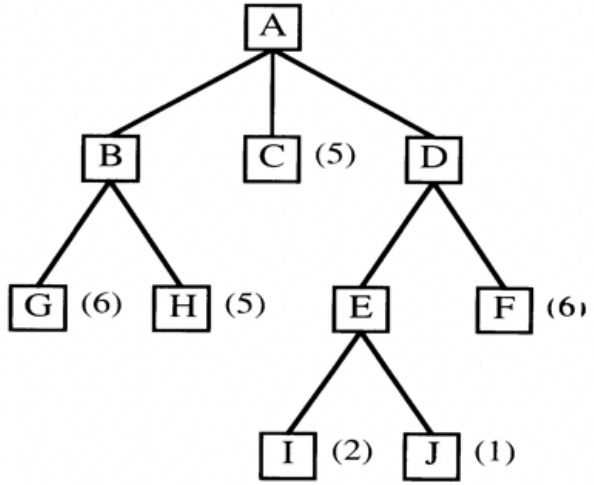



**Lower h(n) is better**

Step 4




Step 5



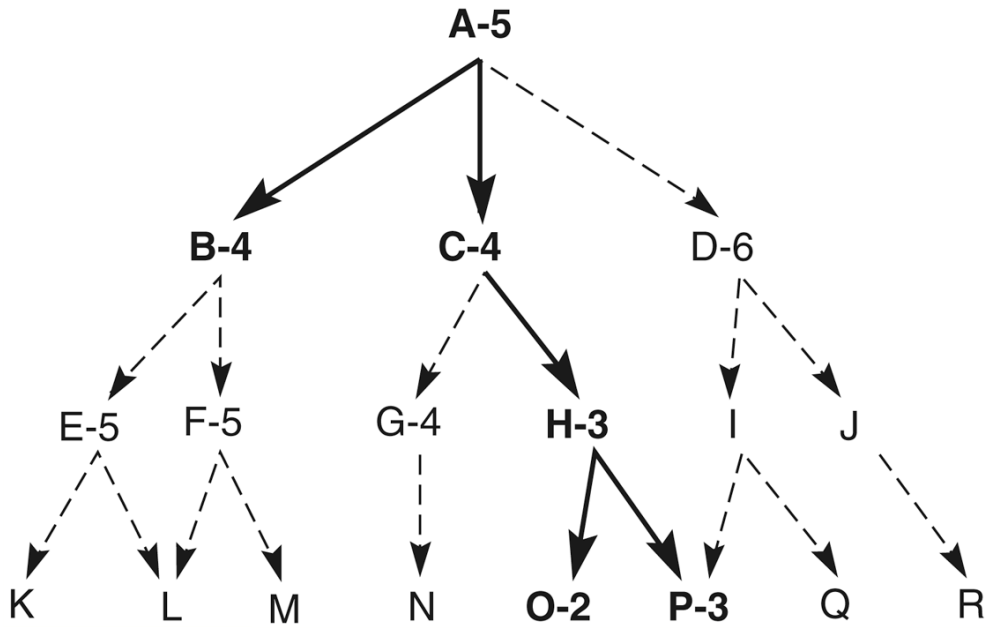
# Best – First Search

- If you have a good evaluation function,
  best-first can find the solution very quickly
- The first solution may not be the best,
  but there is a good chance of finding it quickly
- It is an exhaustive search …
  - will eventually try all possible paths

# Best – First Search Example

1. open = [A-null-5] closed = []
2. open = [B-A-4 C-A-4 D-A-6] *(arbitrary choice)* closed [A]
3. open = [C-A-4 E-B-5 F-B-5 D-A-6] closed = [B A]
4. open = [H-C-3 G-C-4 E-B-5 F-B-5 D-A-6] closed = [C B A]
5. open = [P-H-0 0-H-2 G-C-4 E-B-5 F-B-5 D-A-6] closed = [H C B A]
6. goal P found
7. solution path: A C H P

**Lower h(n) is better**



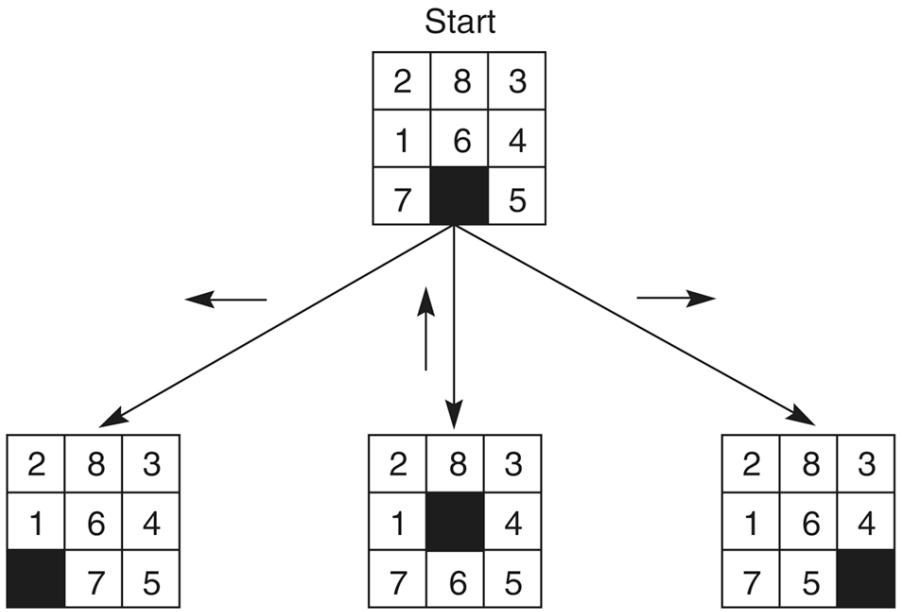
# Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - Informed Search
    - ***Hill Climbing***
    - ***Best – Frist***
    - ***Design Heuristics***
    - A*

# Design Heuristics

- Heuristic evaluation functions are highly dependent on the search domain
- In general: the more informed a heuristic is, the better the search performance
- Bad heuristics lead to frequent backtracking
- So how do we design a "good" heuristic?

# Example: 8 – Puzzle Heuristic 1




- $h_1$: Simplest heuristic
  - Hamming distance : count ***number of tiles out of place*** when compared with goal

| | | |
|---|---|---|
| 5 | | 8 |
| 4 | 2 | 1 |
| 7 | 3 | 6 |

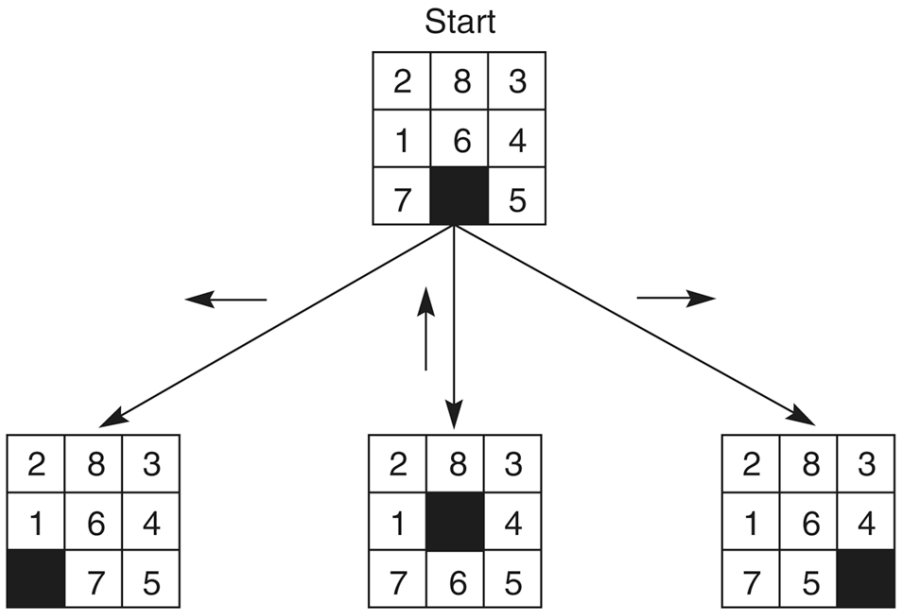
STATE n

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Goal state

- $h_1(n) = 6$
  - does not consider the distance tiles have to be moved

# Example: 8 – Puzzle Heuristic 2




- $h_2$: Better heuristic
  - Manhattan distance: sum up all the ***distances*** by which tiles ***are out of place***

| | | |
|---|---|---|
| 5 | | 8 |
| 4 | 2 | 1 |
| 7 | 3 | 6 |

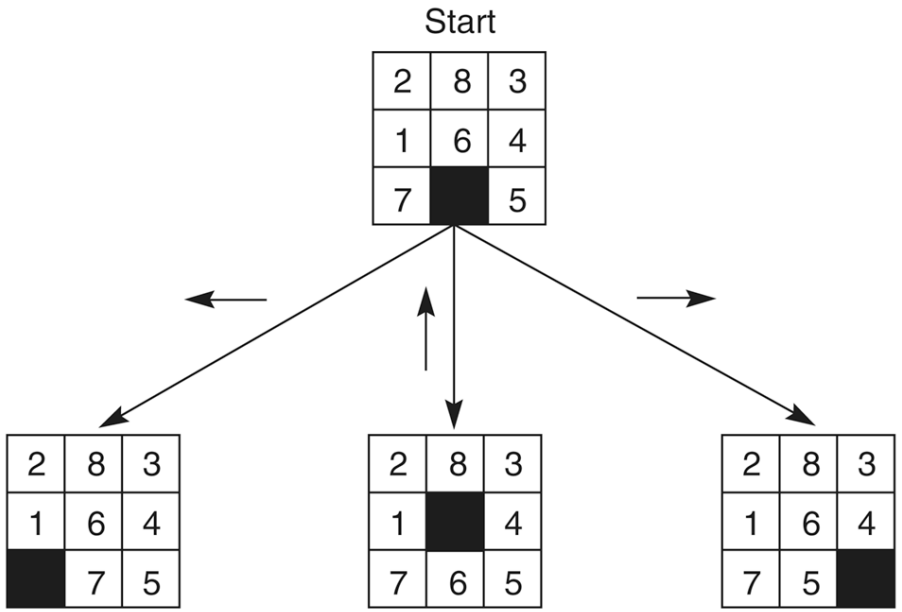
STATE n

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Goal state

- $h_2(n) = 2+3+0+1+3+0+3+1$
  $= 13$

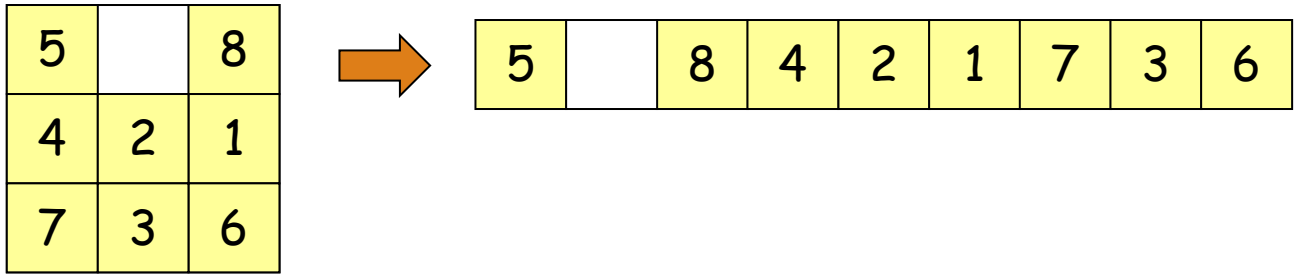
# Example: 8 – Puzzle Heuristic 3



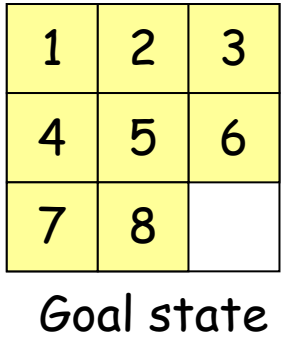

- $h_3$: Even Better
  - sum of permutation inversions
  - See next slide…

# Example: 8 – Puzzle Heuristic 3

- h3(N) = sum of permutation inversions

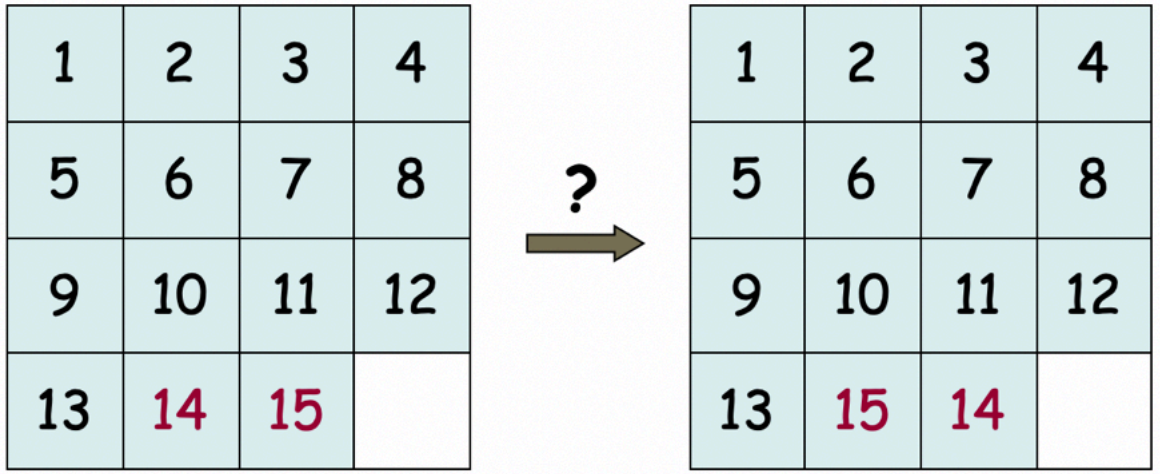



- For each numbered tile, count how many tiles on its right should be on its left in the goal state.
- $h_3(n) = n_5 + n_8 + n_4 + n_2 + n_1 + n_7 + n_3 + n_6$
  $= 4 + 6 + 3 + 1 + 0 + 2 + 0 + 0$
  $= 16$
- If this number is even, the puzzle is solvable.
- If this number is odd, the puzzle is not solvable.



# Example: 15 - Puzzle

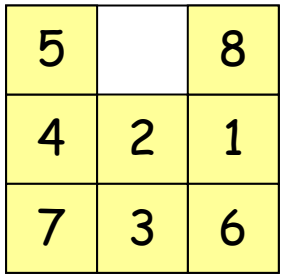
- Sam Loyd even offered $1,000 of his own money to the first person who would solve the following problem:



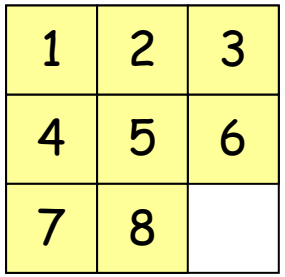

$h_3(n) = n_1 + n_2 + n_3 + n_4 + \ldots + n_{13} + n_{14} + n_{15}$
$= 0 + 0 + 0 + 0 + \ldots + 0 + 1 + 0$
$= 1$

→ the puzzle is not solvable.

# Heuristics for 8 - Puzzle




STATE n




Goal state

- $h_1(n)$ = misplaced numbered tiles (Hamming Distance)
  = 6

- $h_2(n)$ = Manhattan distance
  = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13

- $h_3(n)$ = sum of permutation inversions
  = $n_5 + n_8 + n_4 + n_2 + n_1 + n_7 + n_3 + n_6$
  = 4 + 6 + 3 + 1 + 0 + 2 + 0 + 0 = 16

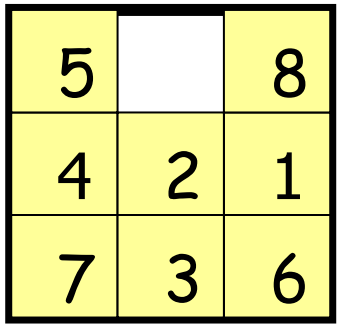
# Evaluation of Heuristics

1. Admissibility:
   - "optimistic"
   - never overestimates the cost of reaching the goal
   - guarantees to find the shortest solution path to the goal (if it exists)
   - Note: does not guarantee to find the shortest search path.
   - e.g.: breadth-first is admissible -- it uses f(n) = g(n) + 0
2. Monotonicity:
   - "local admissibility"
   - guarantees to find the shortest path to each state encountered in the search
3. Informedness:
   - measure for the "quality" of a heuristic
   - the more informed, the better

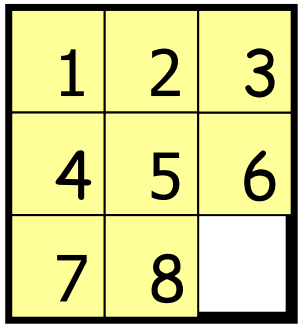
# Admissibility

- Evaluation function **f(n) = g(n) + h(n)** for node **n**:
  - **g(n)** current cost from start to node n
  - **h(n)** estimate of the cost from n to goal
  - **f(n)** estimate of the total cost of the solution path passing through n
- Now consider **f*(n) = g*(n) + h*(n)**:
  - **g*(n)** cost of shortest path from start to node n
  - **h*(n)** actual cost of shortest path from n to goal
  - **f*(n)** actual cost of shortest path from start to goal through n
- In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than or equal to the actual cost of reaching the goal state.

# Example: 8 - Puzzle




n




goal

- h1(n) = Hamming distance = number of misplaced tiles = 6

--> admissible

- h2(n) = Manhattan distance = 13

--> admissible

# Monotonicity (Consistent)

- An admissible heuristics may temporarily reach non-goal states along a suboptimal path
- A heuristic is monotonic if it always finds the optimal path to each state the 1st time it is encountered !
- h is monotonic if for every node n and every successor n' of n:
  - **$h(n) \leq c(n,n') + h(n')$**
- i.e. f(n) is non-decreasing along any path

# Informedness

- Intuition: number of misplaced tiles is less informed than Manhattan distance
- For two admissible heuristics $h_1$ and $h_2$
  - if $h_1(n) \leq h_2(n)$, for all states n
  - then $h_2$ is more informed than $h_1$
  - $h_1(n) \leq h_2(n) \leq h^*(n)$
  - **$h^*(n)$** actual cost of shortest path from n to goal
- More informed heuristics search smaller space to find the solution path
- However, you need to consider the computational cost of evaluating the heuristic…
- The time spent computing heuristics must be recovered by a better search

# Types of State-Space Search

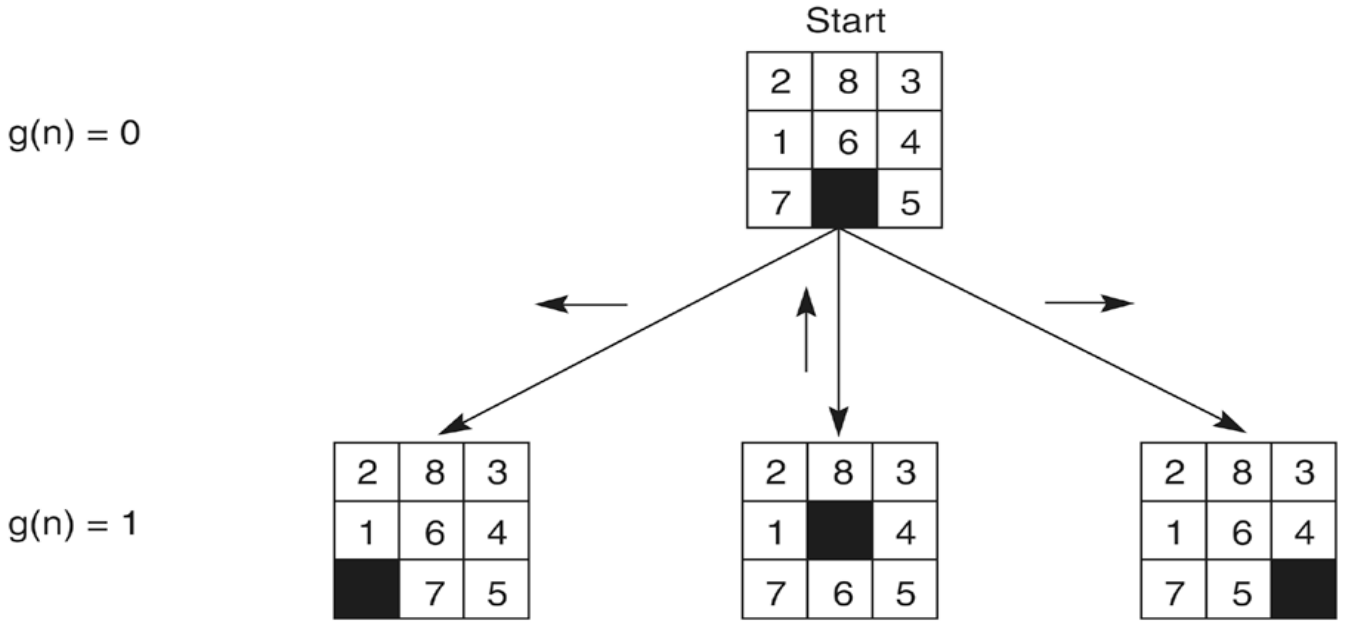
- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - Informed Search
    - ***Hill Climbing***
    - ***Best – Frist***
    - ***Design Heuristics***
    - ***A****

# Algorithm A

- Heuristics might be wrong:
  - so search could continue down a wrong path
- Solution:
  - Maintain depth/cost count, i.e., give preference to shorter/least expensive paths
- Modified evaluation function **f**:

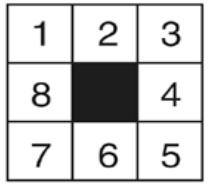
  **f(n) = g(n) + h(n)**
  - f(n) estimate of total cost along path through n
  - g(n) actual cost of path from start to node n
  - h(n) estimate of cost to reach goal from node n

# Algorithm A Example: 8-Puzzle

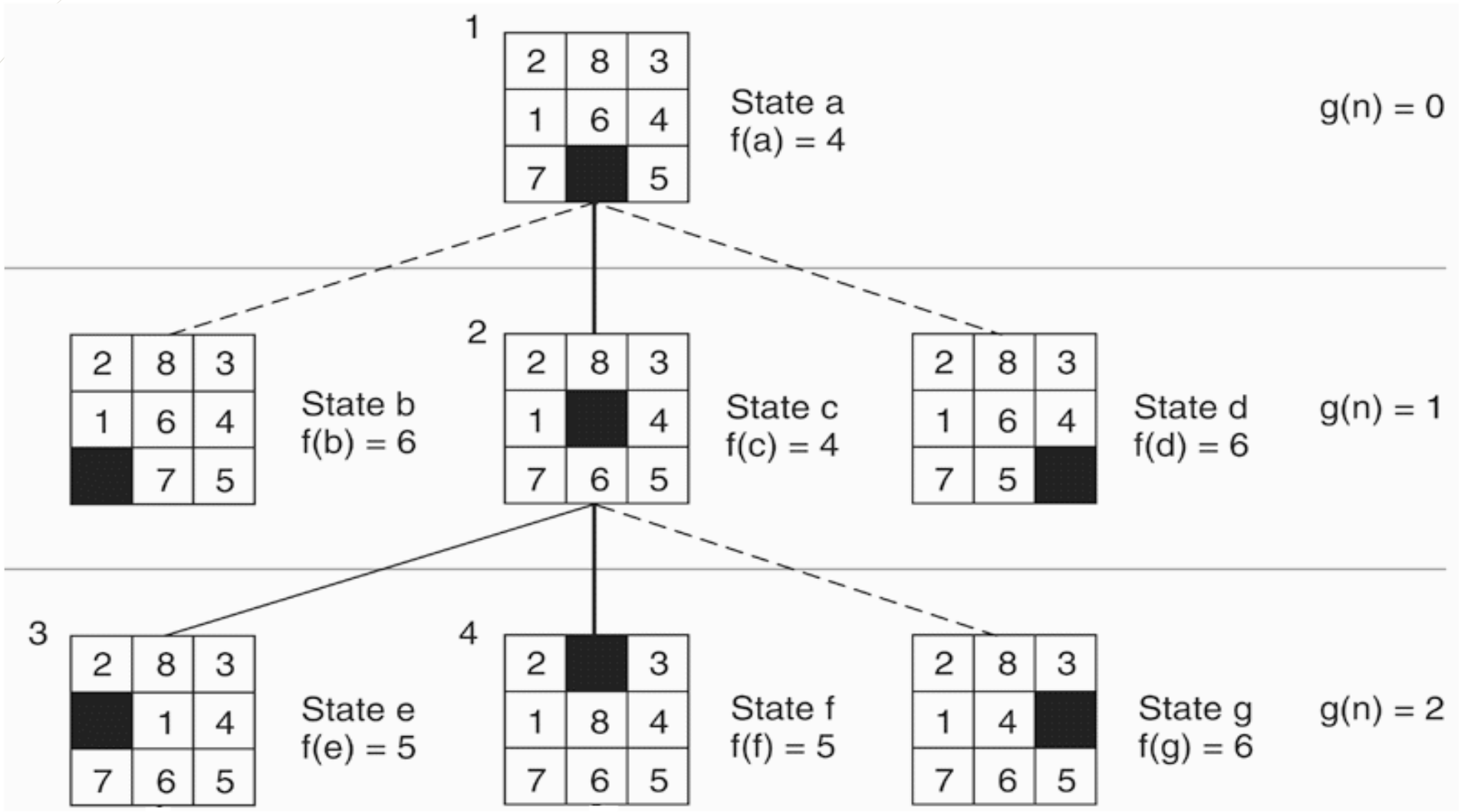



**Values of f(n) for each state,** **6** **4** **6**

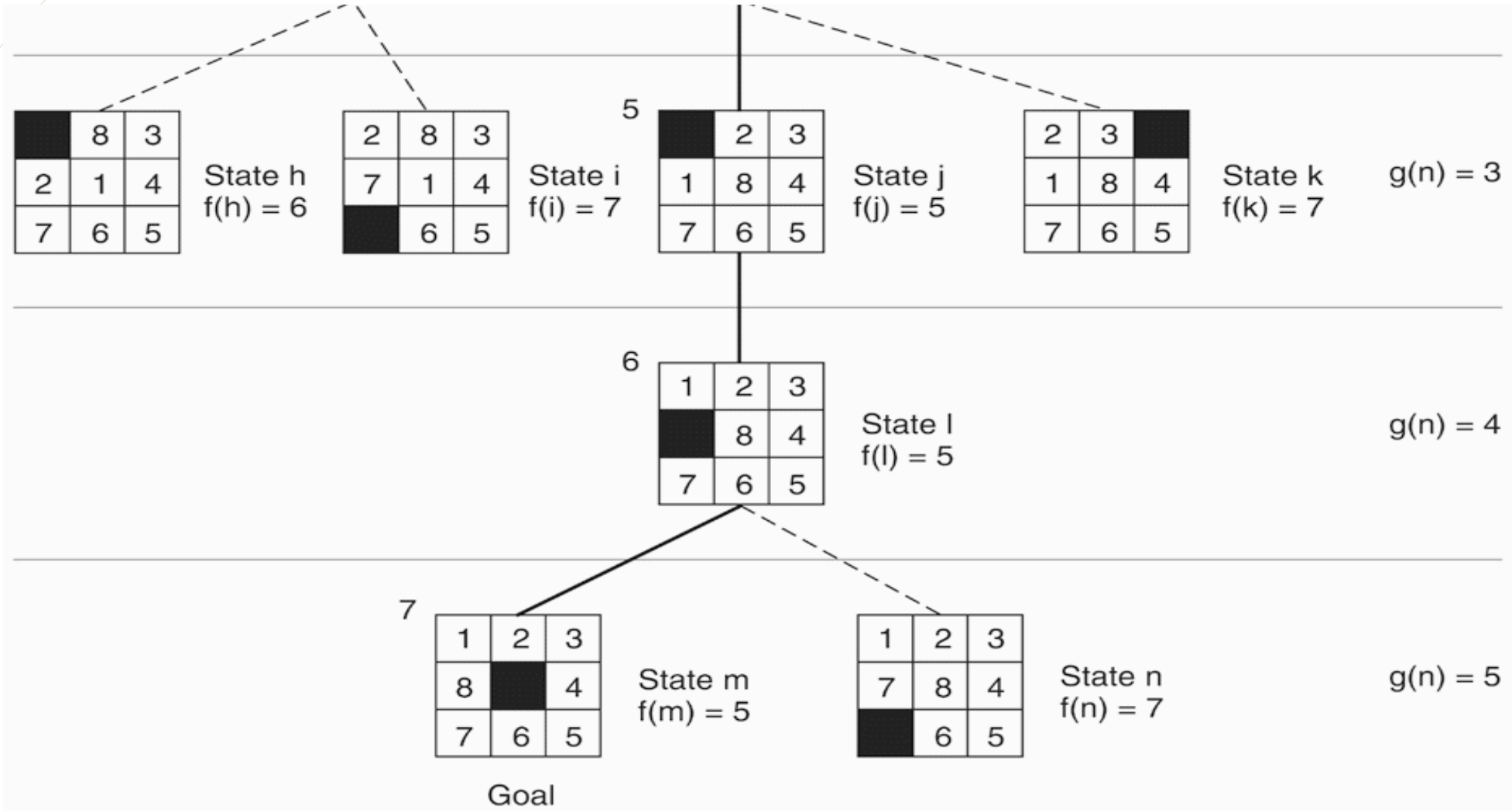
where:
f(n) = g(n) + h(n),
g(n) = actual distance from n to the start state, and
h(n) = number of tiles out of place.



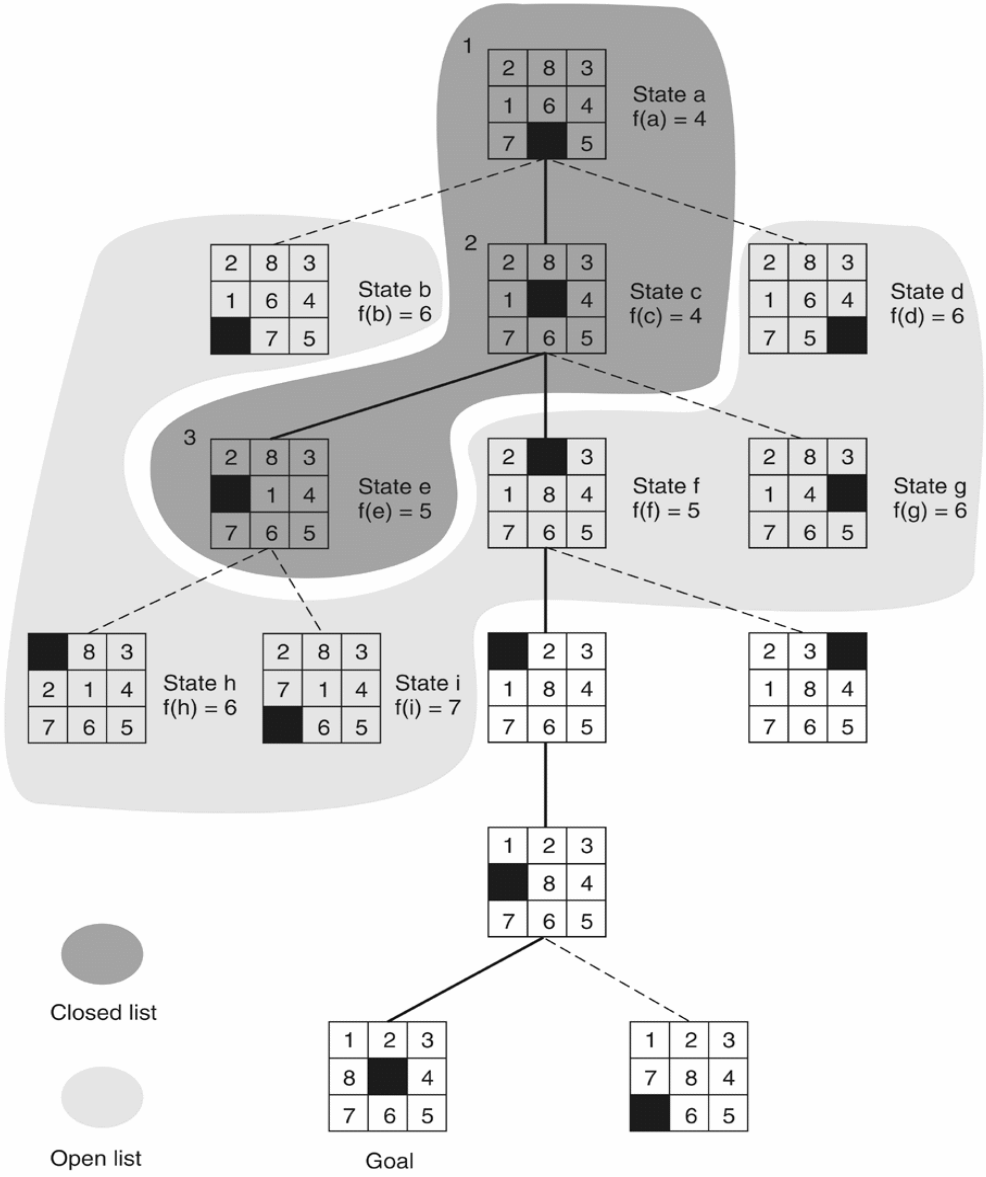
# Algorithm A Example: 8-Puzzle



# Algorithm A Example: 8-Puzzle



# Algorithm A Example: 8-Puzzle

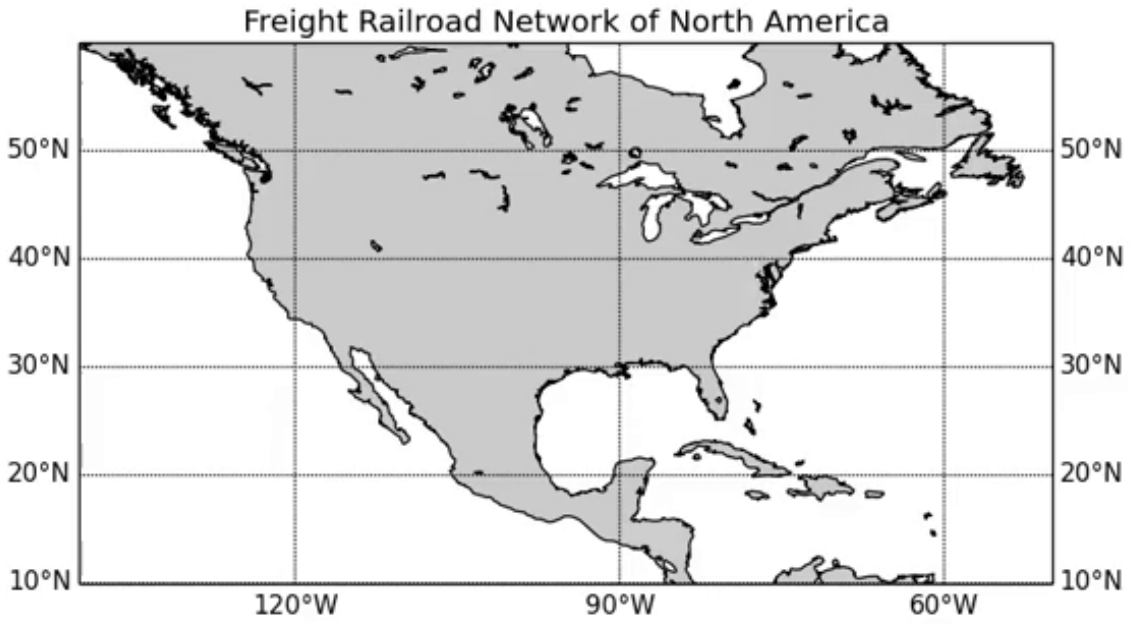


# Algorithm A Example: 8-Puzzle

- if **g(n) ≥ g*(n)** for all n
  - best-first used with such a g(n) is called "algorithm A"
- if **h(n) ≤ h*(n)** for all n
  - i.e. h(n) never overestimates the true cost from n to a goal
  - algorithm **A** used with such an h(n) is called "algorithm A*"
  - ➔ **an A* algorithm is admissible**
  - ➔ i.e it guarantees to find the lowest cost solution path from the initial state to the goal
- **g*(n)** cost of shortest path from start to node n
- **h*(n)** actual cost of shortest path from n to goal

# Algorithm A* Example

The A* algorithm also has real-world applications. In this example, edges are railroads and h(x) is the great-circle distance (the shortest possible distance on a sphere) to the target. The algorithm is searching for a path between Washington, D.C. and Los Angeles.




*Source from Wikipedia:* https://en.wikipedia.org/wiki/A*_search_algorithm

# Types of State-Space Search

- **State-Space Search**
  - ***Uninformed Search***
    - ***Breadth-first and Depth-first***
    - ***Depth-limit Search***
    - ***Iterative Deepening***
    - ***Uniform Cost***
  - ***Informed Search***
    - ***Hill Climbing***
    - ***Best – Frist***
    - ***Design Heuristics***
    - ***A****

# Summary

| Search | Uses h(n)? | Open is a… |
|---|---|---|
| Breadth-first | No | Queue |
| Depth-first | No | Stack |
| Depth-limited | No | Stack |
| Iterative Deepening | No | Stack |
| Uniform Cost | No | Priority queue sorted by g(n) |
| Hill climbing | Yes | none |
| Best-First | Yes | Priority queue sorted by h(n) |
| Algorithm A<br>- no constraints on h(n) | Yes | Priority queue sorted by f(n)<br>f(n) = g(n) + h(n) |
| Algorithm A*<br>- same as A, but h(n) must be admissible | Yes | Priority queue sorted by f(n)<br>f(n) = g(n) + h(n) |

# The End