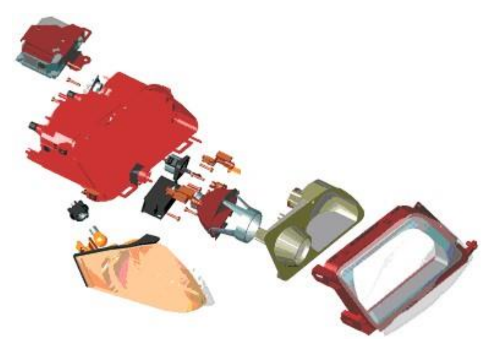
Search

Russell chap 3, 4, 5

Luger chap. 3

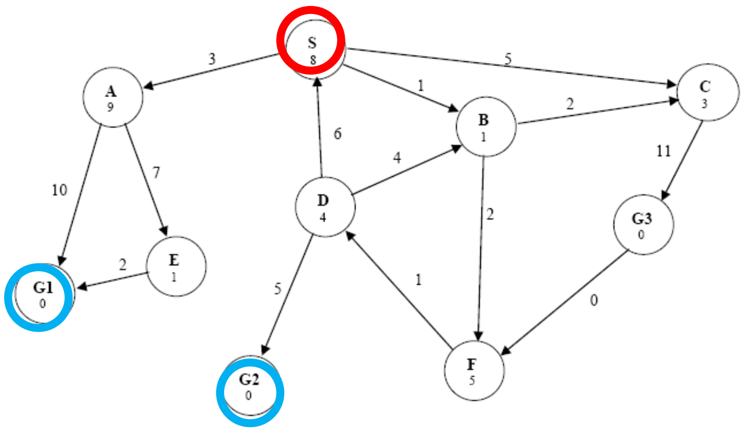
Search

- Problem solving
  - Searching among alternative choices to ***find a sequence of actions*** by which a goal can be achieved
  - Example
    - Route finding : finding a sequence of road
    - Assembly planning : finding a sequence of part assembly
    - Playing game : finding a sequence of move
    - Inference in FOPC : finding a sequence of applying M.P.



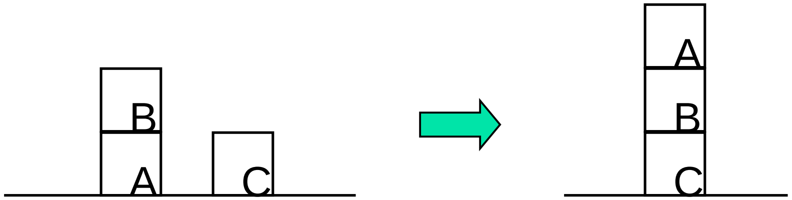
State Space

- State space representation
  - Represent problem space as a *graph*
  - Nodes: states
  - Arcs: Operations to other states
- State space
  - [N, A, S, G]
    - N – nodes(states)
    - A – arcs(operations)
    - S – start state
    - G – goal state



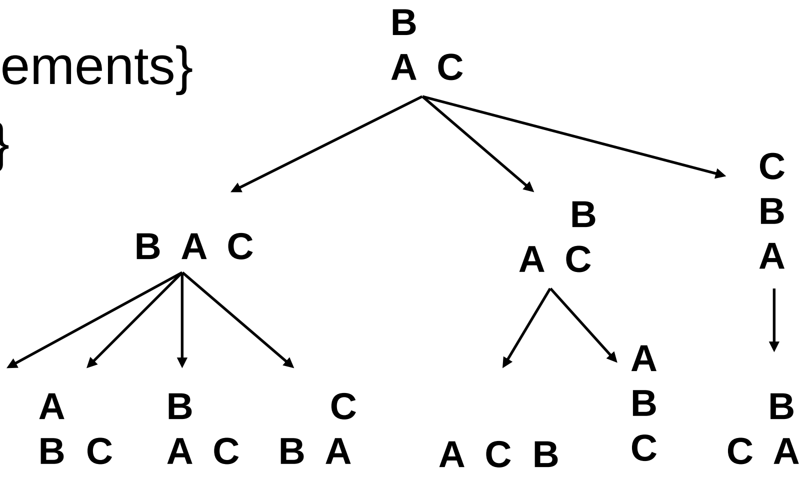
State Space

- Planning in a blocks world



N = {different arrangements}

A = {moving a block}



State Space

- 8-Puzzle

1	4	3
7		6
5	8	2

1	2	3
8		4
7	6	5

N = {different arrangements}

A = {moving the blank}

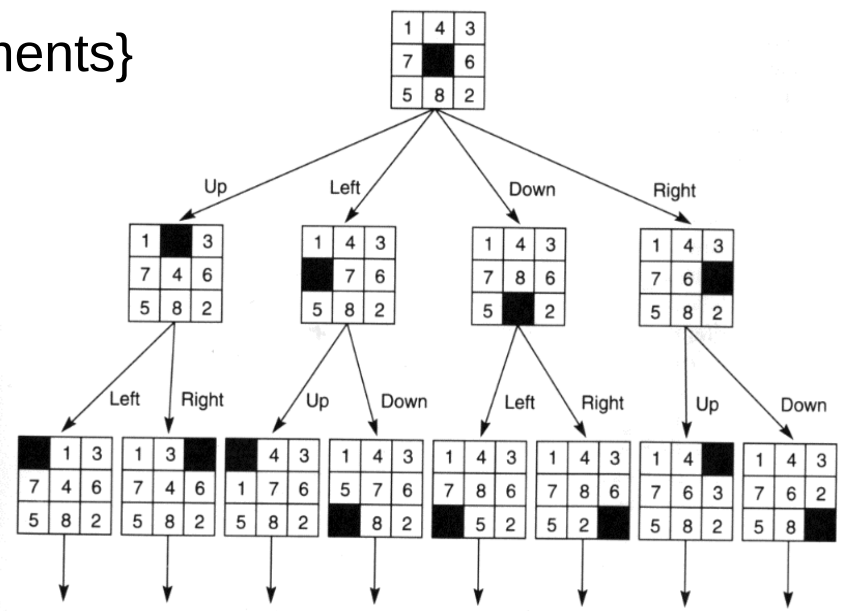


Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

State Space

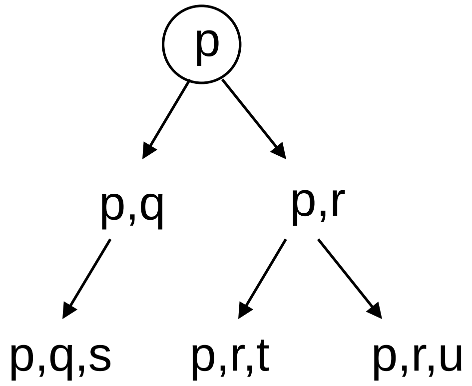
- Inference

$p \Rightarrow q$, $p \Rightarrow r$, $q \Rightarrow s$,
$r \Rightarrow t$, $r \Rightarrow u$, p

t ?

N = {true sentences}

A = {applying M.P.}



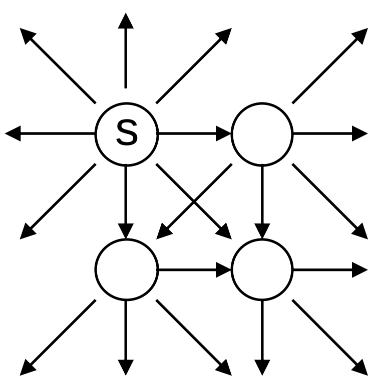
State Space

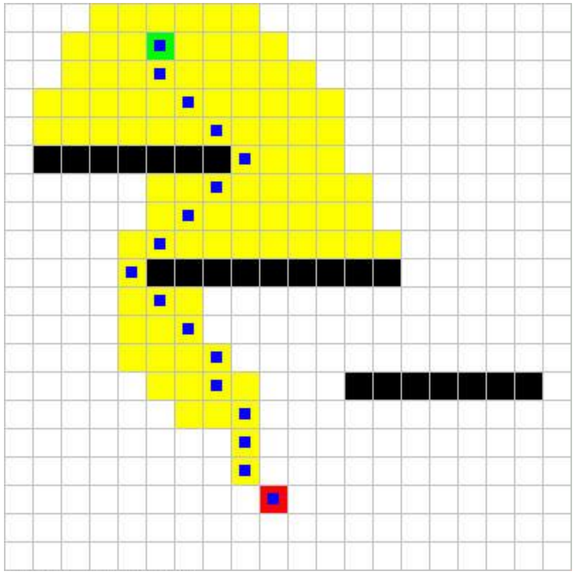
- Route finding

start location → goal location

N = {different locations}

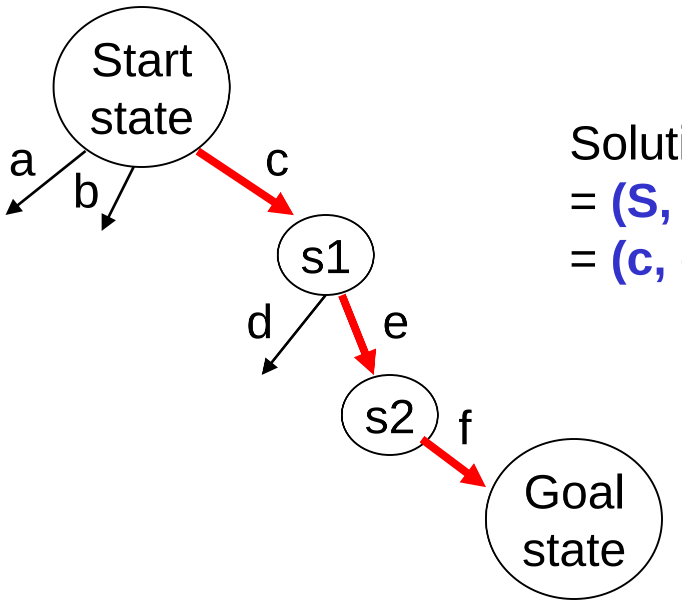
A = {move to the adjacent location}





State Space Search

- Search
  - Finding a solution path from S to G

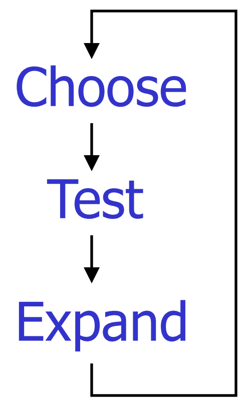


Solution path
= **(S, s1, s2, G)**
= **(c, e, f)**

Depth-First Search

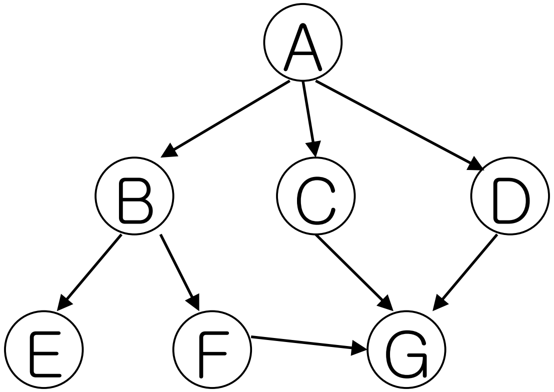
- Goes deeper whenever possible

```
open = {START}, closed = Ø
while (open ≠ Ø)
    remove leftmost state from open → X
    if (X is GOAL) return (success)
    else
        generate children of X
        put X into closed
        eliminate child if it is in open or close
        put remaining children into left of open(stack)
return (fail)
```



Depth-First Search

- Example



1. open = { }, closed = { }
2. X = , open = { }, closed = { }
3. X = , open = { }, closed = { }
4. X = , open = { }, closed = { }
5. X = , open = { }, closed = { }
6.

Depth-First Search

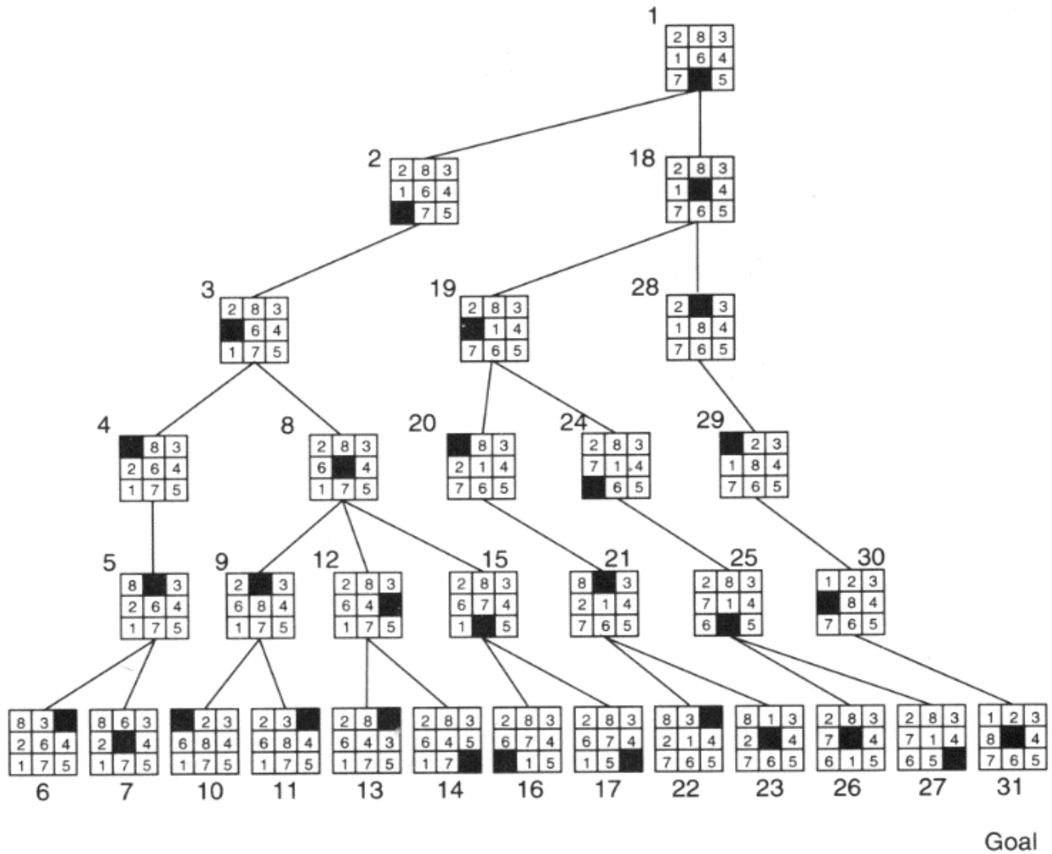
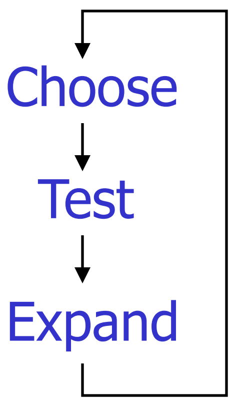


Figure 3.17 Depth-first search of the 8-puzzle with a depth bound of 5.

Breadth-First Search

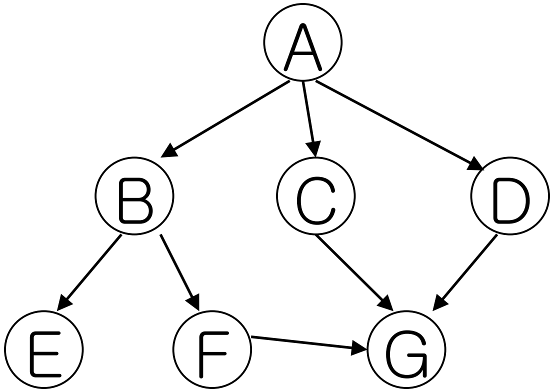
- Explore search space level by level

```
open = {START}, closed = Ø
while (open ≠ Ø)
    remove leftmost state from open → X
    if (X is GOAL) return (success)
    else
        generate children of X
        put X into closed
        eliminate child if it is in open or close
        put remaining children into right of open(queue)
return (fail)
```



Breadth-First Search

- Example



1. open = { }, closed = { }
2. X = , open = { }, closed = { }
3. X = , open = { }, closed = { }
4. X = , open = { }, closed = { }
5. X = , open = { }, closed = { }
6. X = , open = { }, closed = { }
7. X = , open = { }, closed = { }
8.

Breadth-First Search

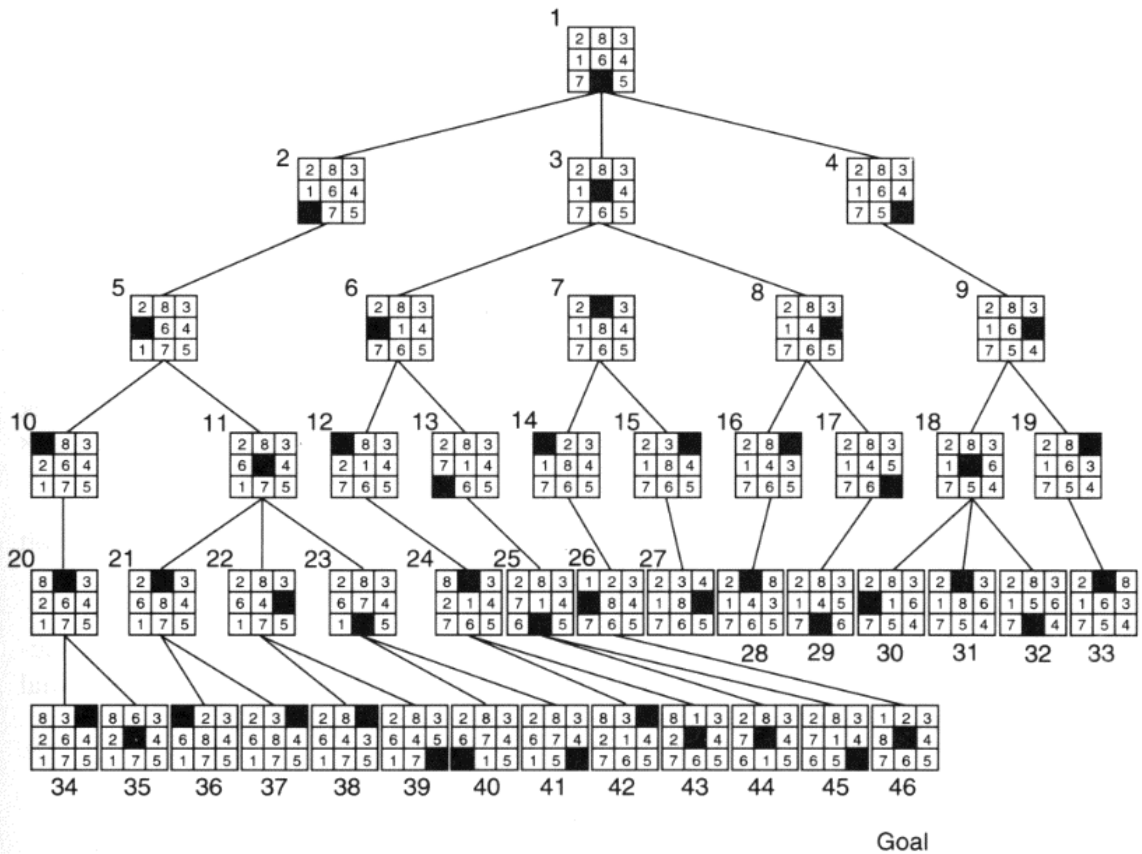


Figure 3.15 Breadth-first search of the 8-puzzle, showing order in which states were removed from open.

Solution Path

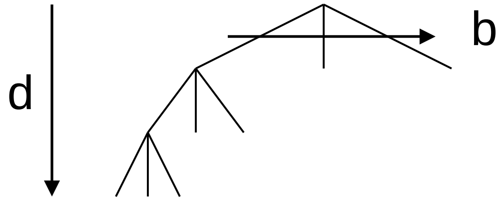
- Store parent information with the states
- Example
  - closed = { (A,0), (B,A), (C,A), (D,A), (E,B), (F,B) }
  - Goal = (G, C)

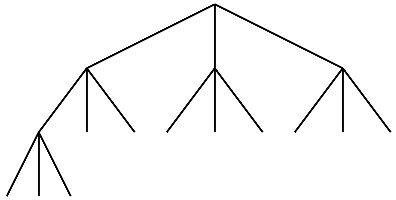
⇨ G ← C ← A

DFS vs. BFS

- Let
  - b: Average branching factor
  - d: Search depth

	OPEN	Time	Space	Optimal
DFS	stack	$O(b^d)$	$O(b \cdot d)$	No
BFS	queue	$O(b^d)$	$O(b^d)$	Yes





- DFS: $b + b + \ldots + b$ BFS: $b + b^2 + \ldots + b^{d-1}$

Heuristics

- Uninformed search (DFS, BFS)
  - Impractical for large state space
    - 15-puzzle: ~10^{12}, Chess: ~10^{120}, Baduk: ~10^{360}
  - Inefficient in terms of time and space
- Informed search
  - Use *heuristics*
    - Rules for choosing branches in a state space
    - Guide search to states that are most likely lead to the goal

⇨ *Informed search, heuristic search*

Heuristics

- Example
  - State to search next?

2 8 3 1 6 4 7 5	5	6	0
2 8 3 1 4 7 6 5	3	4	0
2 8 3 1 6 4 7 5	5	6	0
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

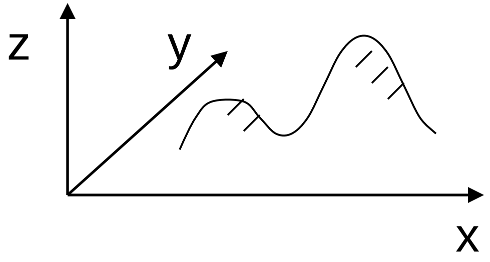
1	2	3
8		4
7	6	5

Goal

Hill Climbing

- Algorithm
  1. Expand current state
  2. Select best child state (use heuristics)
  3. Stop if current state is best, or goto 1

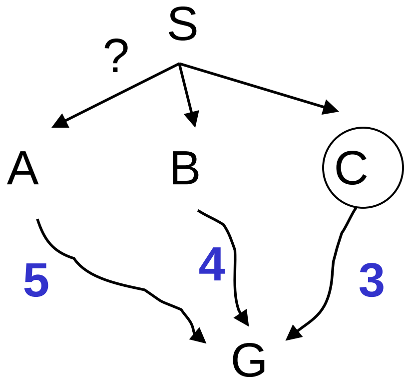
- Problem
  - It may stuck at local maximum
    - Find max z = f(x, y)   (S: (1,1), G: unknown)



Best-First Search

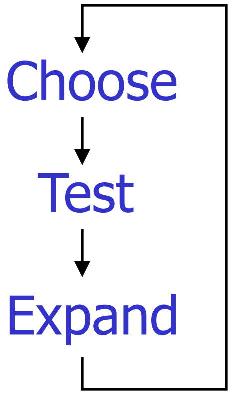
- Best-first
  - Use heuristic value f(n) to evaluate each state
  - Choose most promising state from OPEN

→ If heuristic value estimate the distance (cost) to the goal, choose smallest one

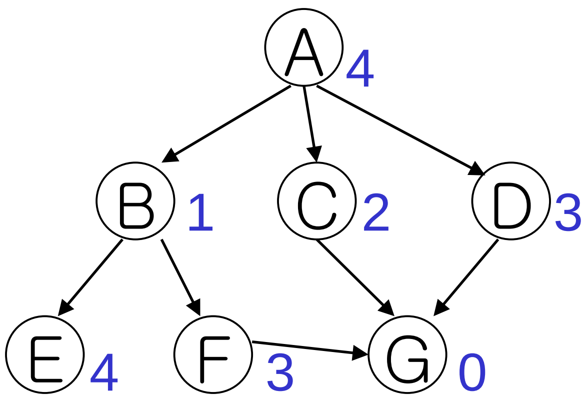


Best-First Search

```
open = {START}, closed = Ø
while (open ≠ Ø)
    remove leftmost state from open → X
    if (X is GOAL) return (success)
    else
        generate and evaluate children of X
        put X into closed
        for each child C
                if C is in open          update path
                if C is in closed and reached by shorter path
                                         remove C from closed, put into open
                else                     put C into open
        reorder open(priority queue)
 return (fail)
```

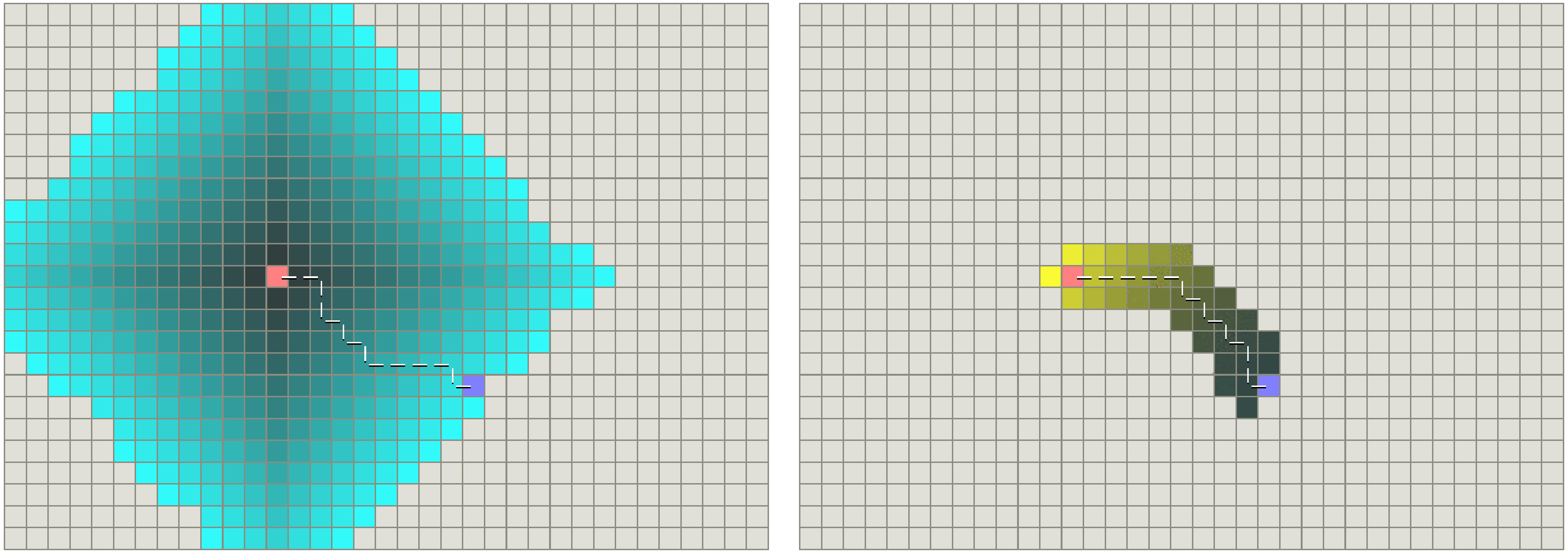


Best-First Search



1. open = { },
   closed = { }
2. X = , open = { },
   closed = { }
3. X = , open = { },
   closed = { }
4. X = , open = { },
   closed = { }
5.

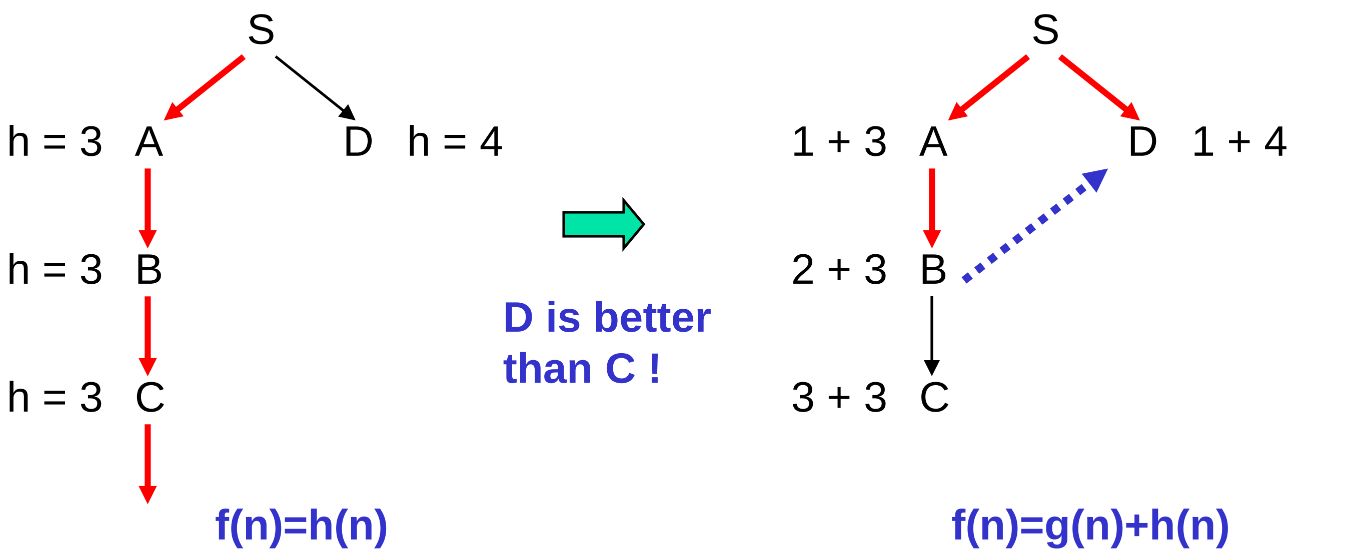
BFS vs. Best-First



Evaluation Function

- Heuristics are fallible
  - We want to find good path (not only fast search)

    → use (heuristic value to G + actual path length from S)



Evaluation Function

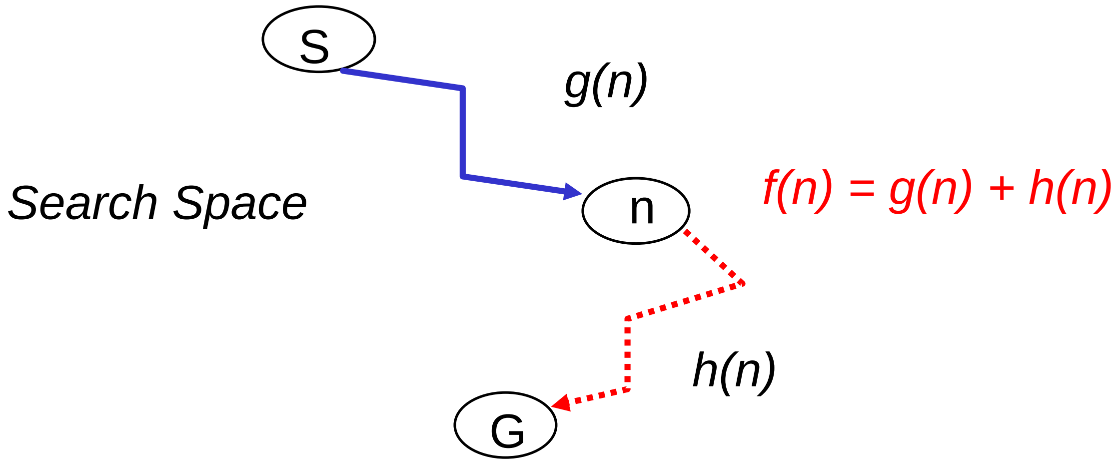
- Evaluation function

$$\boxed{f(n) = g(n) + h(n)}$$

g(n) : actual cost (distance) from S to n

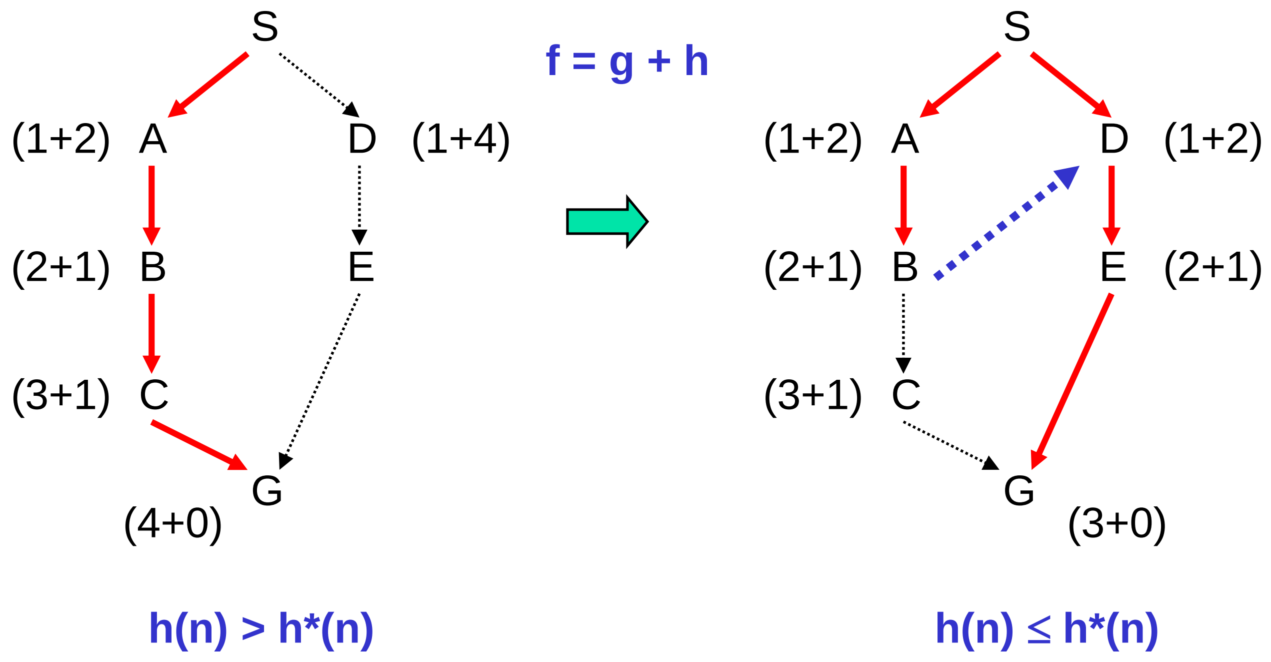
h(n) : estimated cost (distance) from n to G

- → **f(n)** is the estimated cost (distance) ***from S to G***



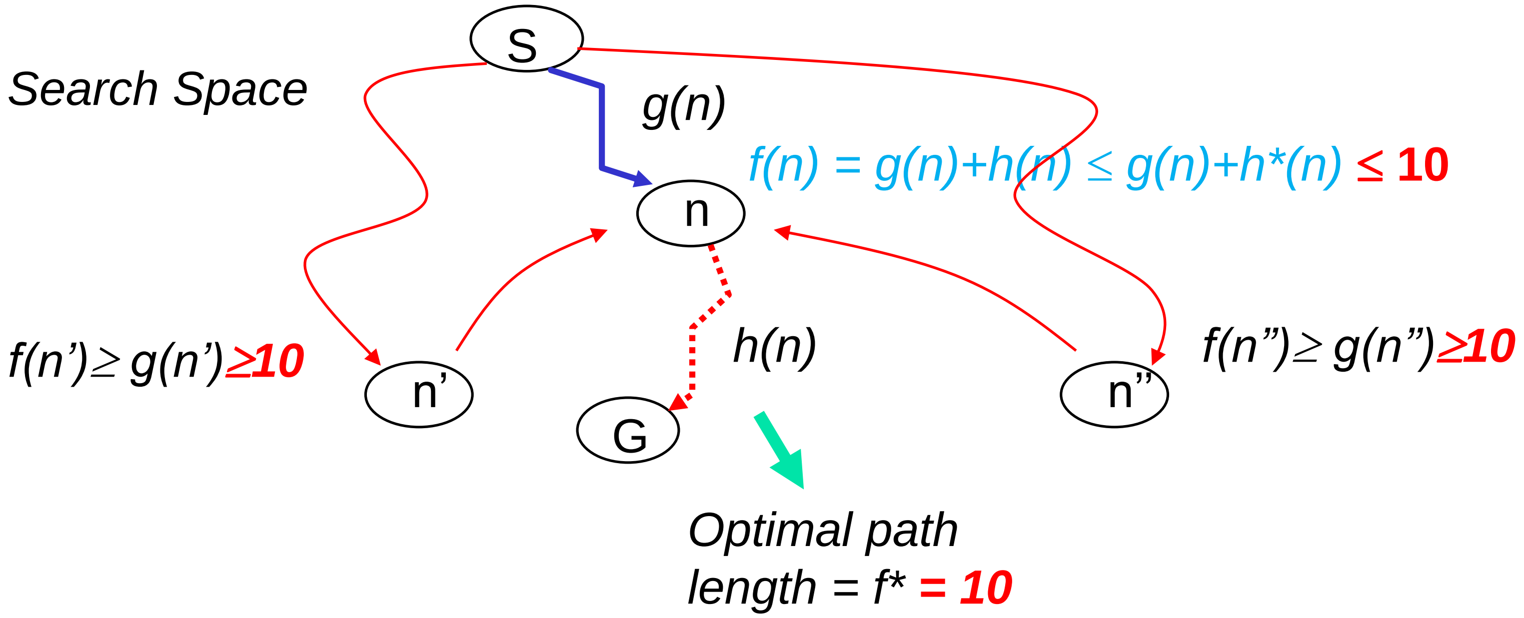
Under Estimation

- Heuristics are fallible
  - Optimal path may not be found
    → use heuristic value <u>smaller than real value (h*(n))</u>



Under Estimation

- $h(n) \leq h^*(n)$ guarantees optimal path
  - Example: optimal path length = f* = 10



A* Search

- A search algorithm is admissible
  - If it is guaranteed to find a minimal path to a solution whenever such a path exist
- A* algorithm
  - A best-first search with

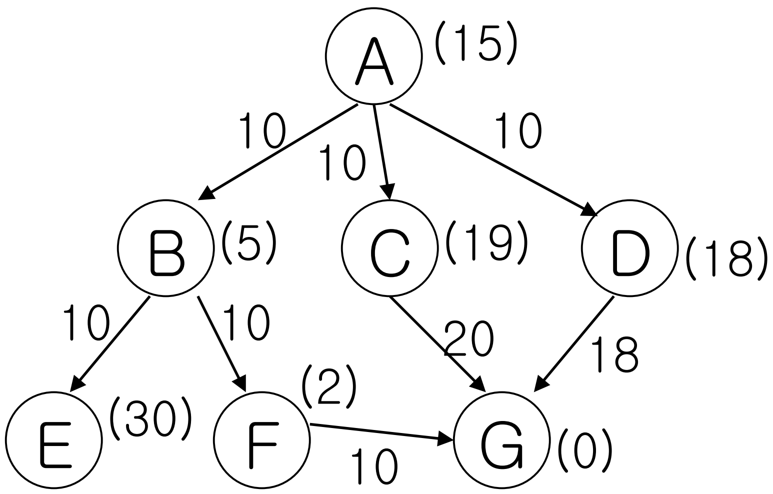
$$f(n) = g(n) + h(n)$$

where $\mathbf{h(n) \leq h^*(n)}$

⇨ *Admissible* !

($h^*(n)$: actual cost (distance) from n to G)

A* Search



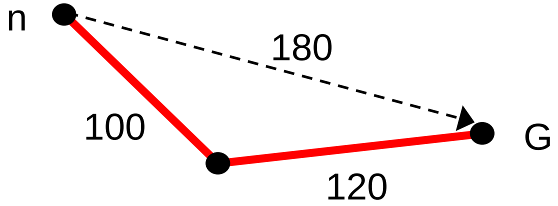
1. open = { },
   closed = { }
2. X = , open = { },
   closed = { }
3. X = , open = { },
   closed = { }
4. X = , open = { },
   closed = { }
5. X = , open = { } ,
   closed = { }
6.

Admissible Heuristics

- BFS
  - f(n) = g(n)

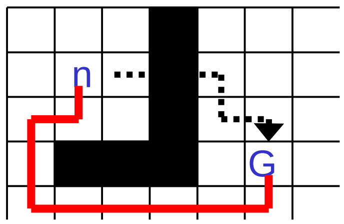
    → h(n) = 0 < h*(n)   ∴ BFS is an A* algorithm !
- Route-finding
  - h(n) = straight line distance to G

    ≤ h*(n)



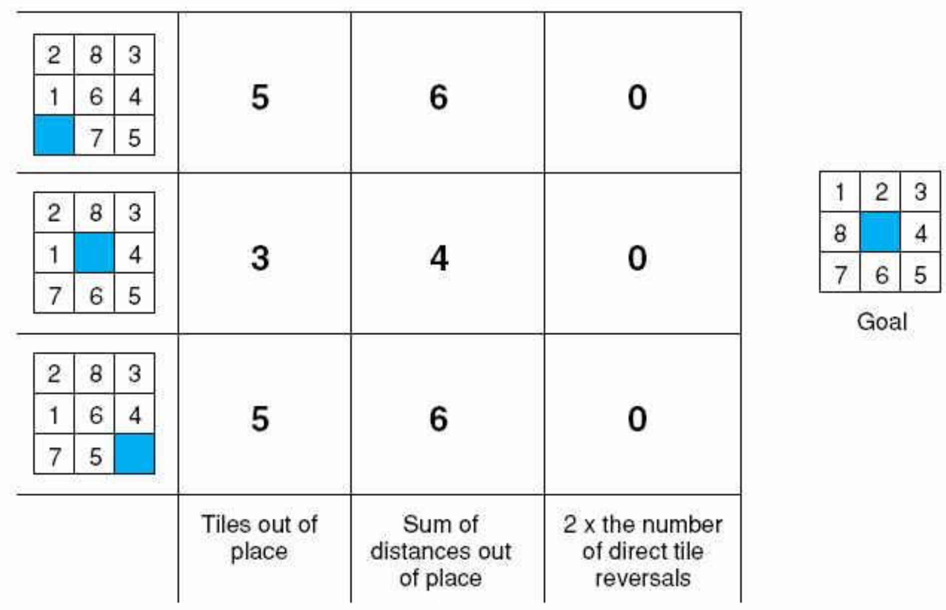
  - h(n) = Manhattan distance to G

    ≤ h*(n)



Admissible Heuristics

- 8-puzzle
  - $h_1(n)$ = # of tiles in wrong position $\leq h^*(n)$
  - $h_2(n)$ = Sum of Manhattan distance $\leq h^*(n)$



Admissible Heuristics

- Informedness
  - Accuracy of h(n)
  - $0 < h_1(n) < h_2(n) \leq h^*(n)$

    ⇨ h_2 is more accurate

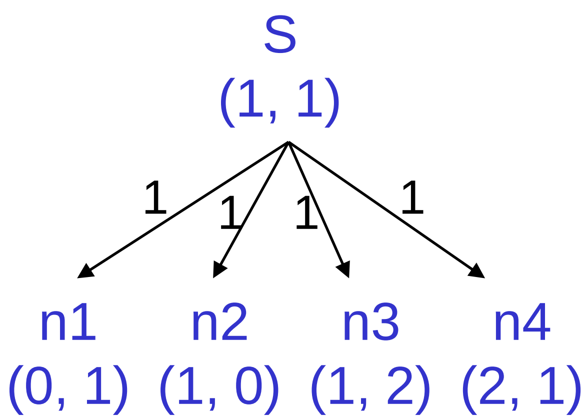
    ⇨ h_2 expands less nodes

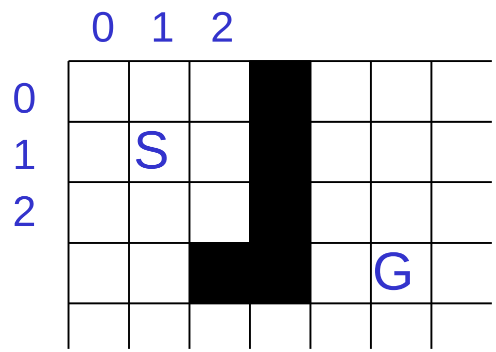
  - Example
    - $h_1(n)$ = # of tiles in wrong position < $h_2(n)$ = Manhattan distance

      → $h_2(n)$ is better !

A* Search - Example

- Route finding in grid space
  - h(n) : Manhattan distance



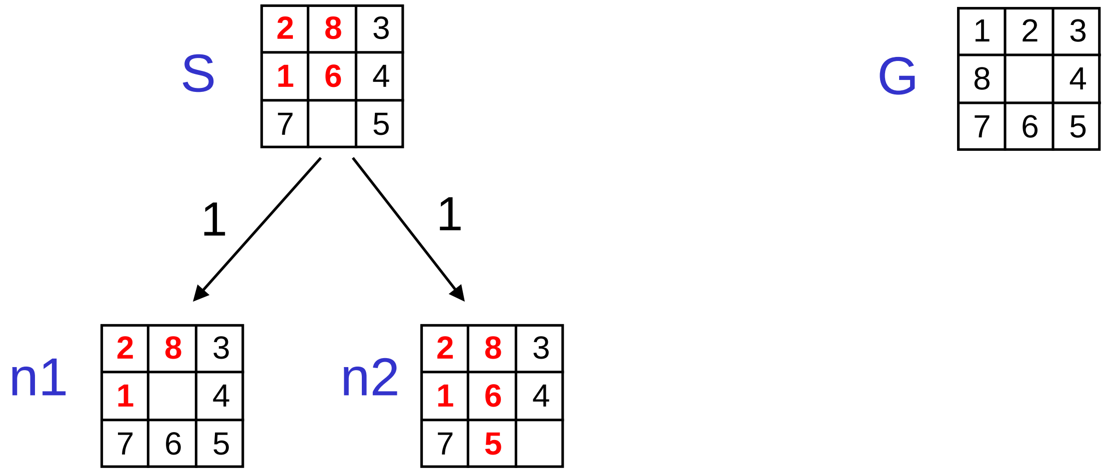


g(n1) = 1
h(n1) = 7
f(n1) = 1 + 7 = 8

g(n4) = 1
h(n4) = 5
f(n4) = 1 + 5 = 6

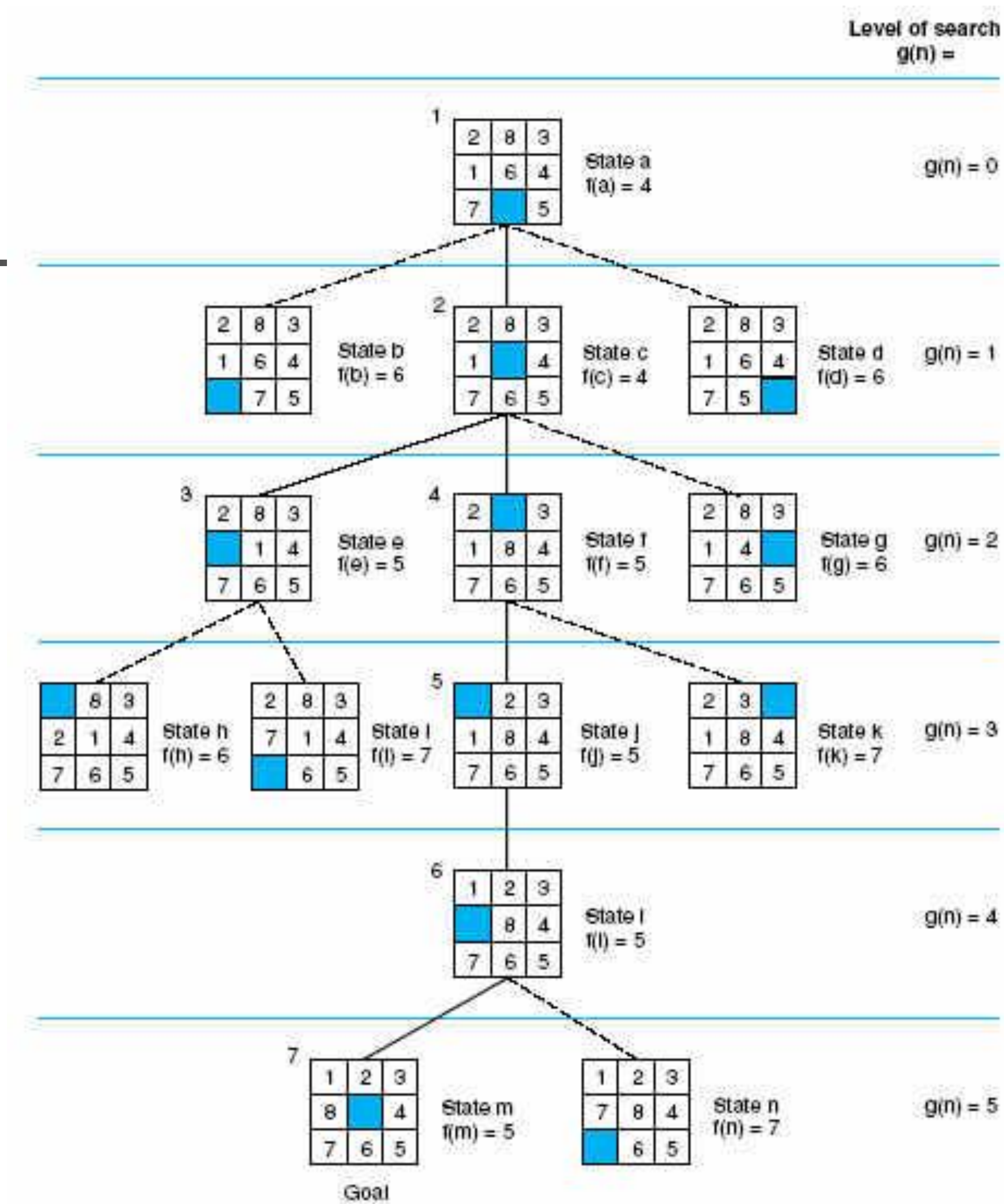
A* Search - Example

- 8-puzzle
  - h(n) : # of tiles out-of-place

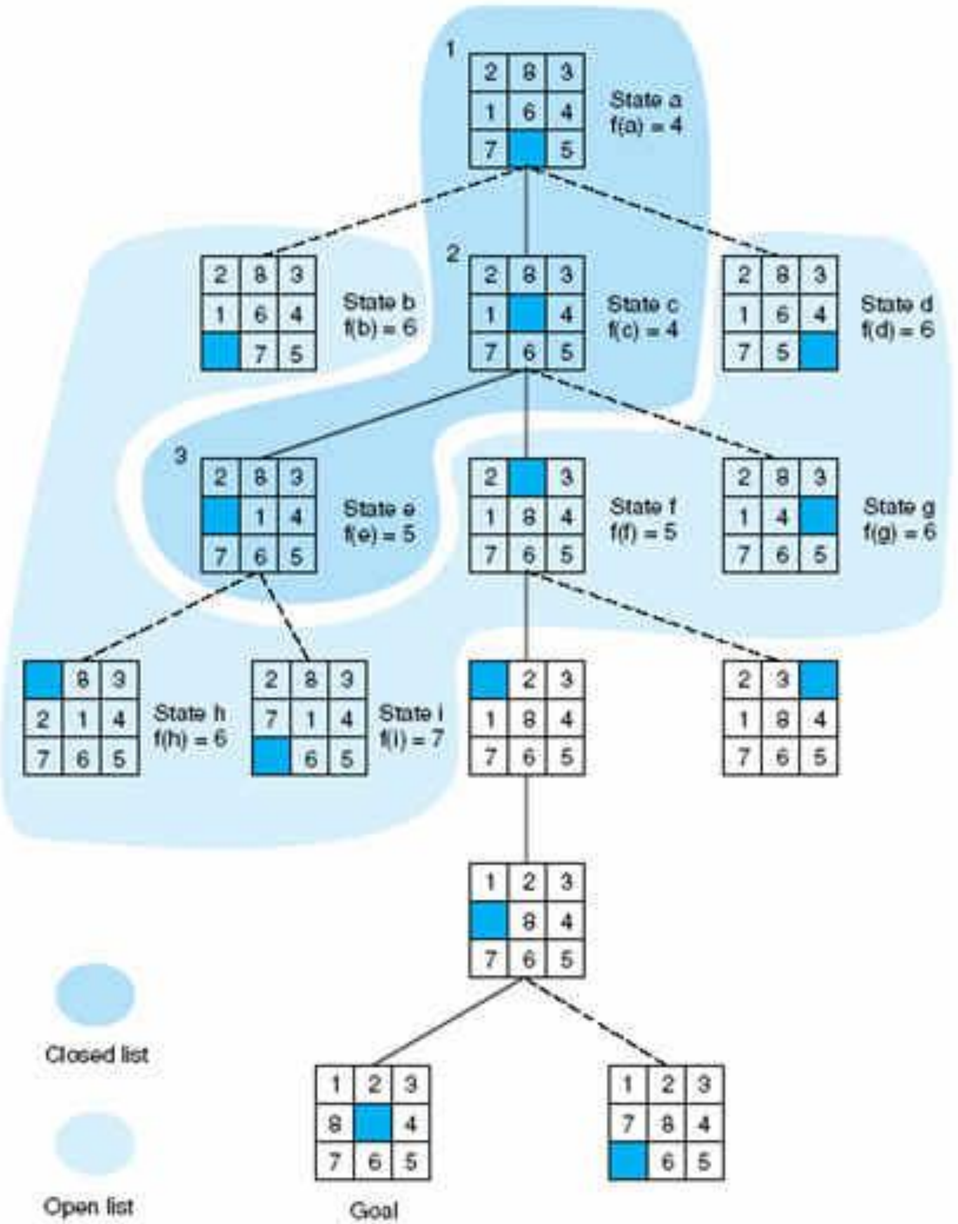


g(n1) = 1
h(n1) = 3
f(n1) = 1 + 3 = 4

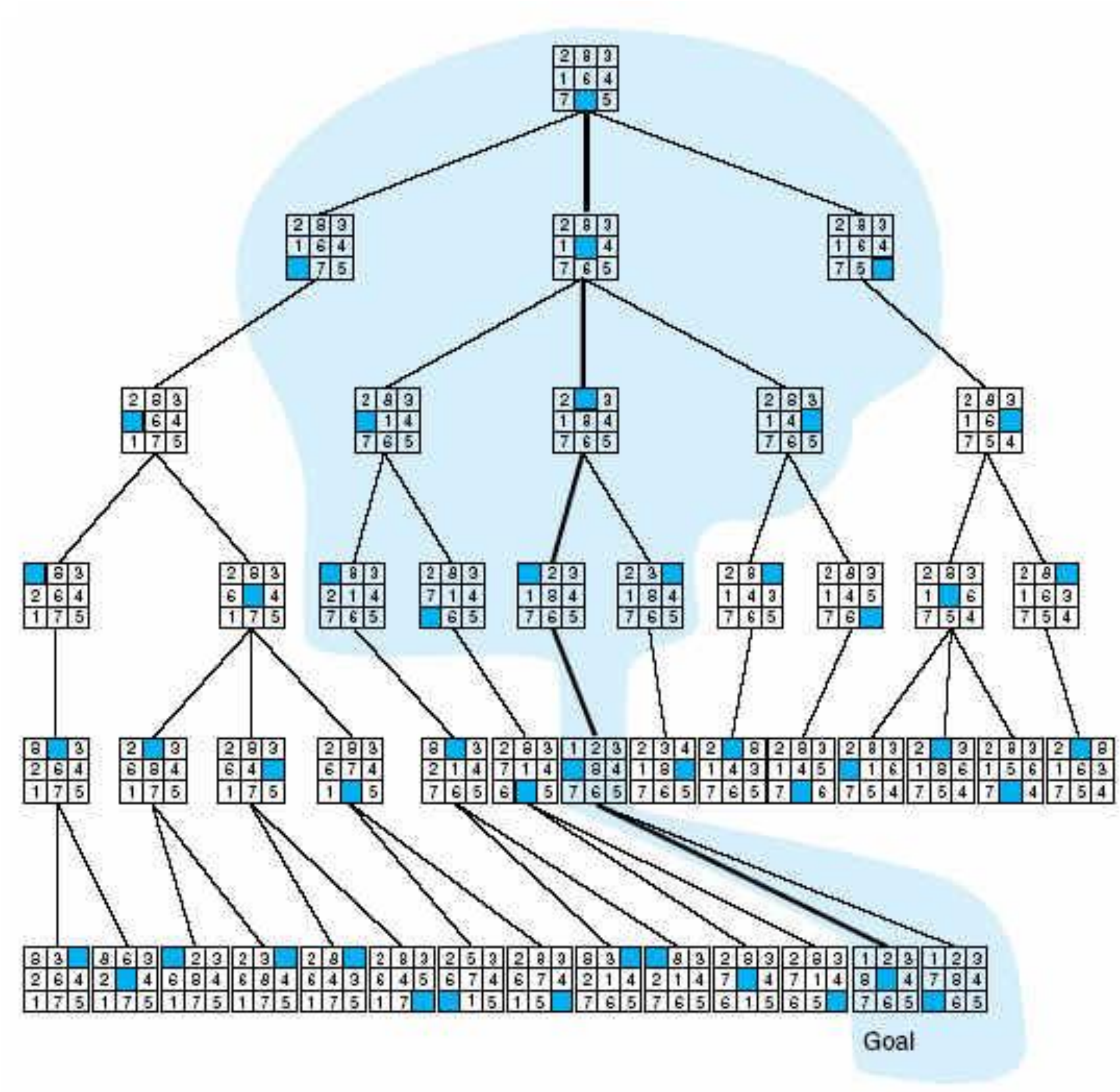
g(n2) = 1
h(n2) = 5
f(n2) = 1 + 5 = 6



Level of search
g(n) =
1
2 8 3
1 6 4
7 5
State a
f(a) = 4
g(n) = 0
2 8 3
1 6 4
7 5
State b
f(b) = 6
2
2 8 3
1 4
7 6 5
State c
f(c) = 4
2 8 3
1 6 4
7 5
State d
f(d) = 6
g(n) = 1
3
2 8 3
1 4
7 6 5
State e
f(e) = 5
4
2 3
1 8 4
7 6 5
State f
f(f) = 5
2 8 3
1 4
7 6 5
State g
f(g) = 6
g(n) = 2
8 3
2 1 4
7 6 5
State h
f(h) = 6
2 8 3
7 1 4
6 5
State i
f(i) = 7
5
2 3
1 8 4
7 6 5
State j
f(j) = 5
2 3
1 8 4
7 6 5
State k
f(k) = 7
g(n) = 3
6
1 2 3
8 4
7 6 5
State l
f(l) = 5
g(n) = 4
7
1 2 3
8 4
7 6 5
State m
f(m) = 5
Goal
1 2 3
7 8 4
6 5
State n
f(n) = 7
g(n) = 5

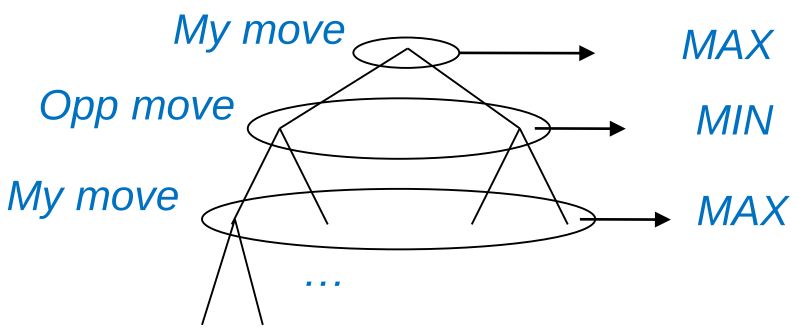


1
State a
f(a) = 4
2
State b
f(b) = 6
State c
f(c) = 4
State d
f(d) = 6
3
State e
f(e) = 5
State f
f(f) = 5
State g
f(g) = 6
State h
f(h) = 6
State i
f(i) = 7
Closed list
Open list
Goal


Goal

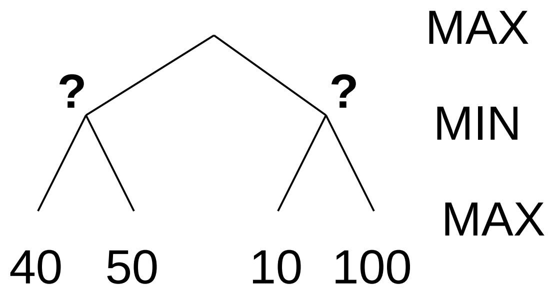
Using Heuristics in Game

- Search in 2-persons game
  - Chess, Baduk, …
- Assumption
  - I select a state that MAXimize my winning probability
  - Opponent select a state that MINimize my winning probability
- Divide states
  - MAX - my move
  - MIN - opponent's move

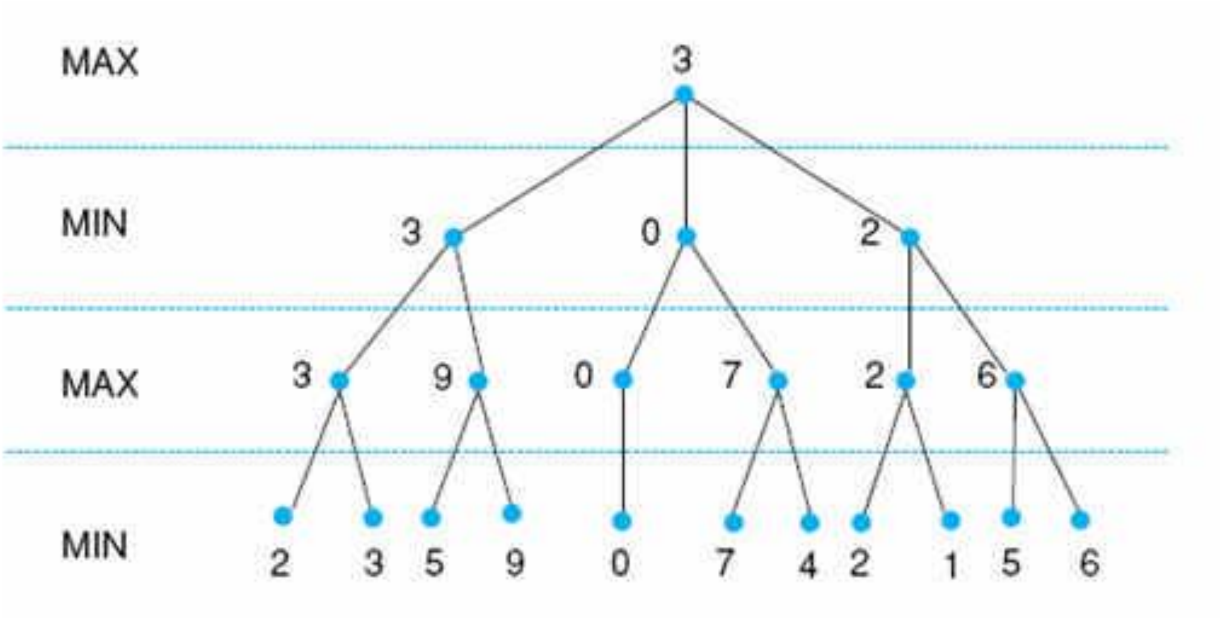


Minimax Procedure

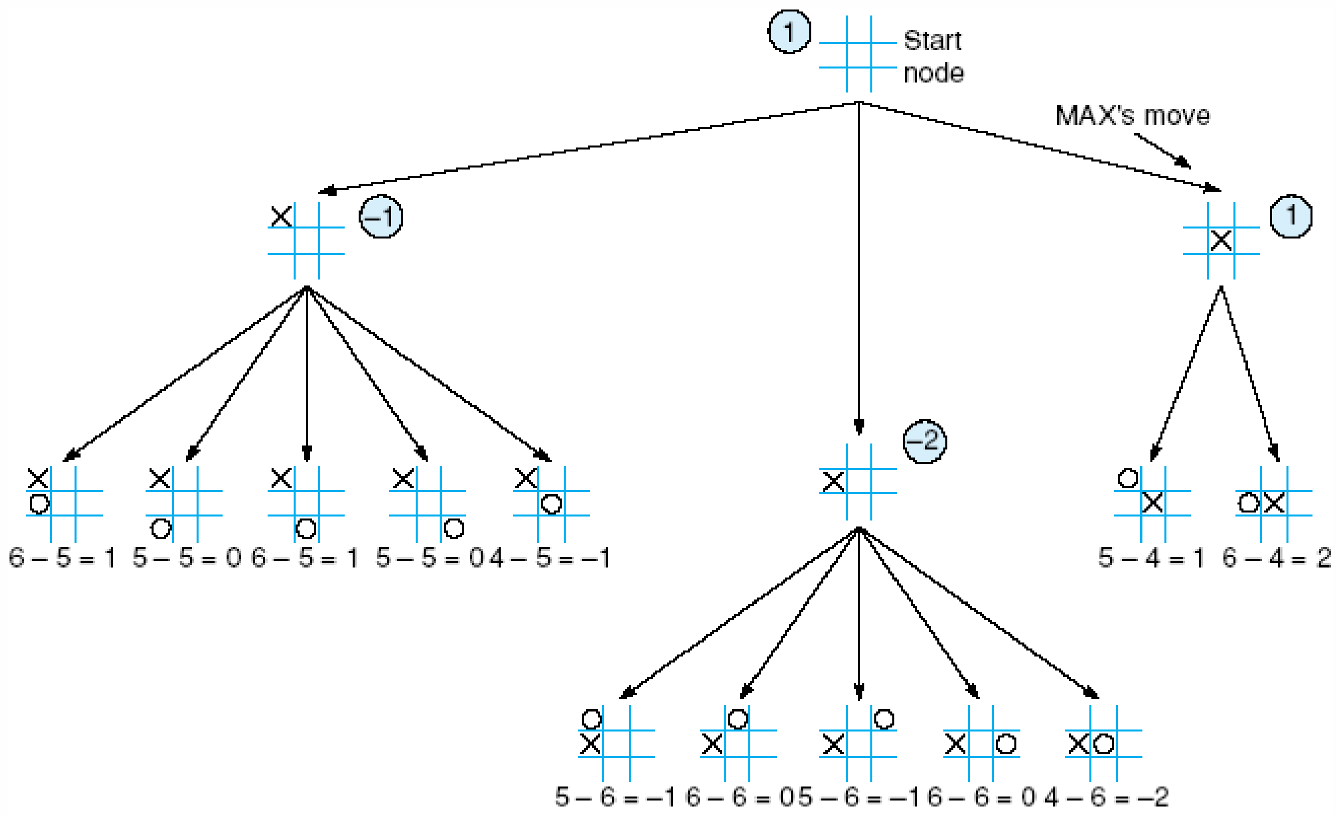
- Evaluation
  1. Search to certain depth
  2. Evaluate states (higher value means better state)
  3. In MAX node, value of node = max of children
  4. In MIN node, value of node = min of children



Minimax Procedure



Example: TicTacToe

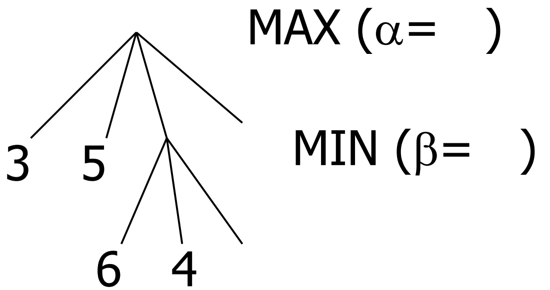


Alpha-Beta Pruning

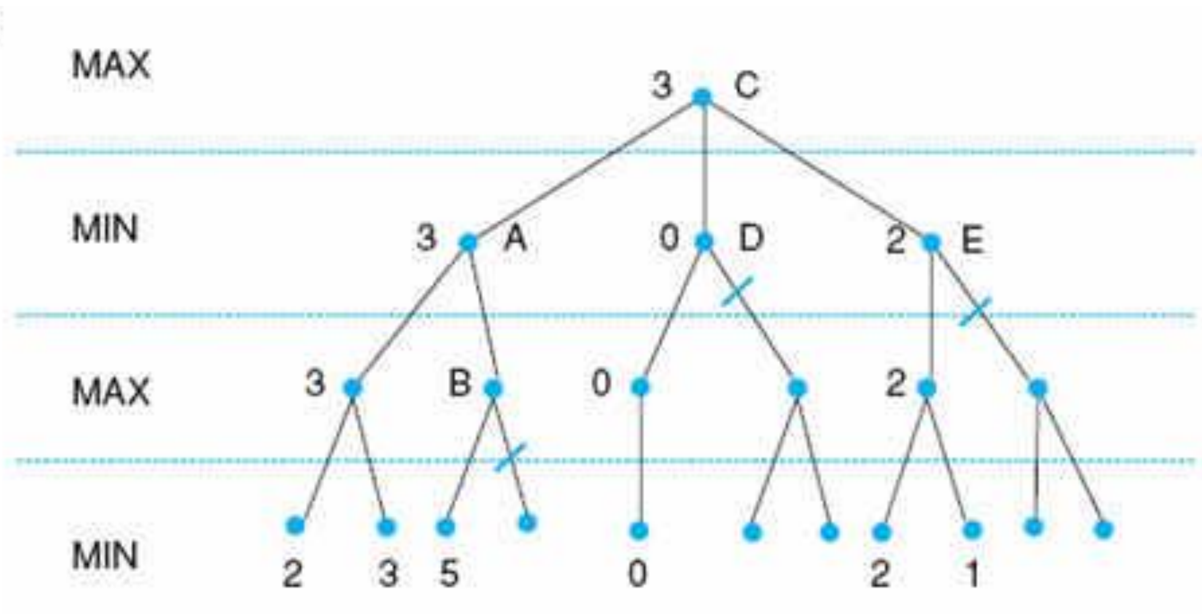
- Reduce the search space
- Algorithm
  - Search the game tree like DFS with depth bound
  - α - current maximum value of MAX node
  - β - current minimum value of MIN node
  - If $\beta \leq \alpha$ of parent, stop search
  - If $\alpha \geq \beta$ of parent, stop search

α always increase

β always decrease



Alpha-Beta Pruning

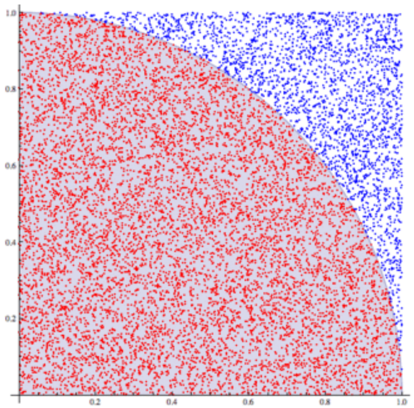


MCTS

- Monte Carlo Method
  - Algorithms rely on repeated *random sampling* to obtain numerical results
  - Example: computing π
    - Generate random (x, y) in [0,1] x [0,1]
    - Count the number of points inside the circle of radius 1

$$\frac{\#of\ inside\ points}{\#\ of\ total\ points} \approx$$

$$\frac{area\ of\ \frac{1}{4}\ circle}{area\ of\ square} = \frac{\frac{1}{4}\pi}{1}$$



MCTS

- MCTS: Monte Carlo Tree Search
  - A heuristic search algorithm for decision processes in game
    1. The game is played out to the end by selecting random moves
    2. Playout is then used to weight the nodes in the game tree
       → better nodes are more likely to be chosen in future play