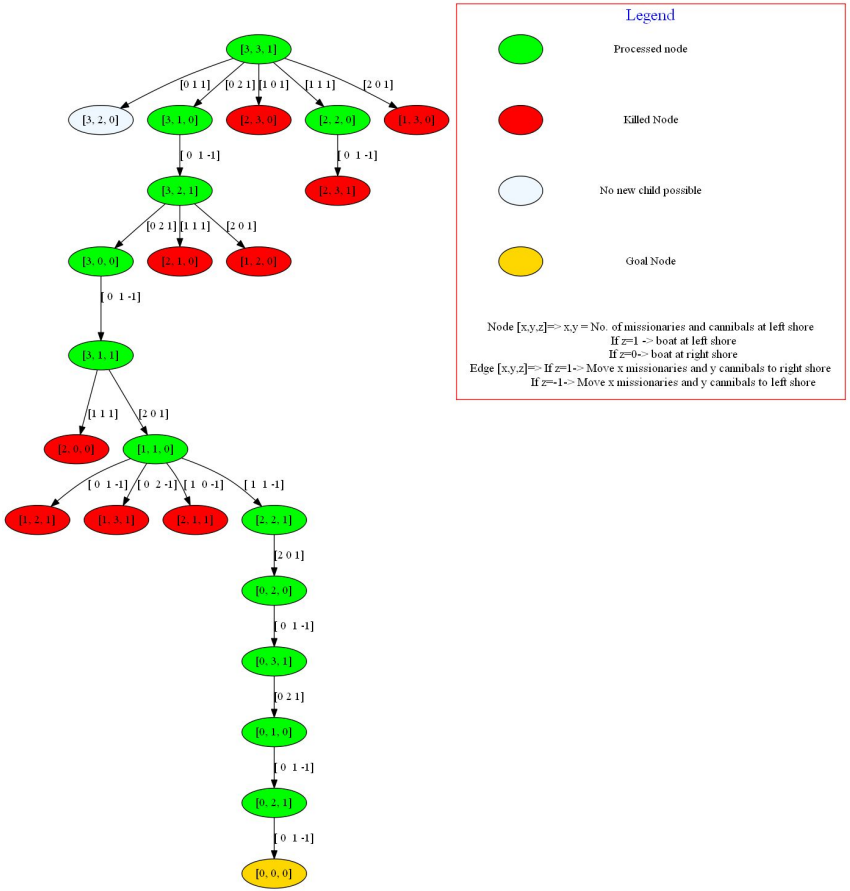
Search Strategies (Module 3)

Extra Examples…..

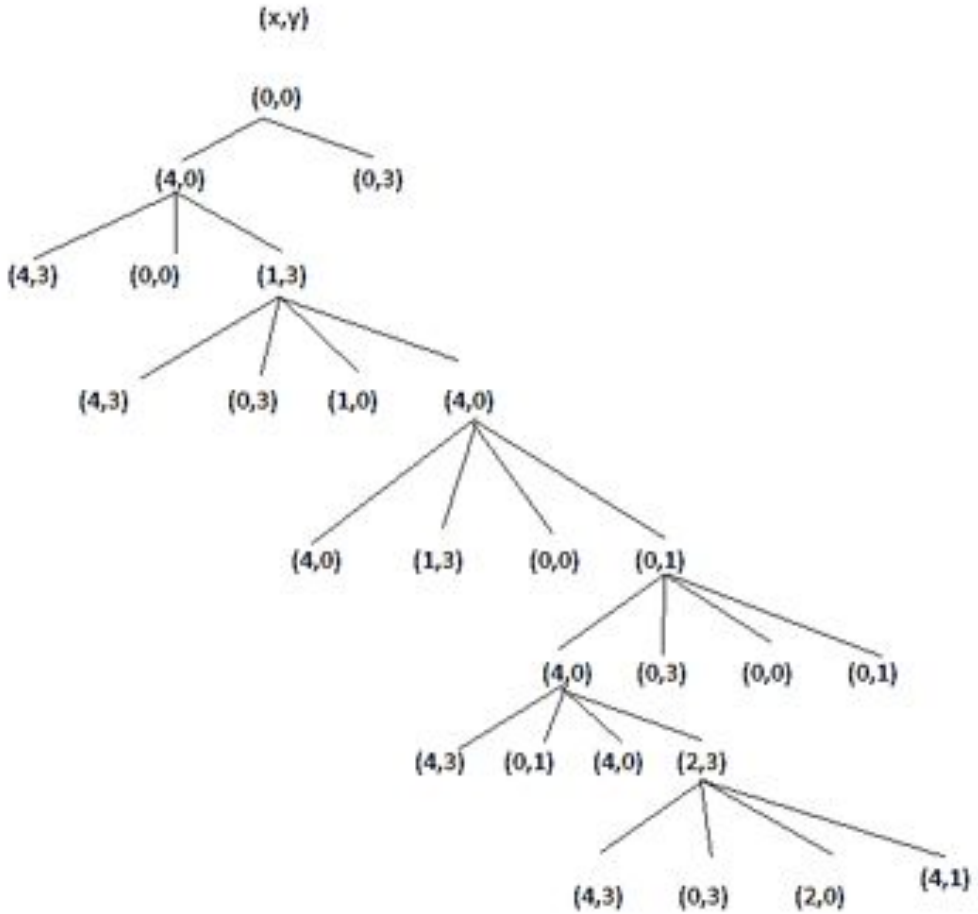
Dr. Ayesha Hakim

State space diagram for Missionary Cannibal Problem



Missionaries and Cannibals State Space

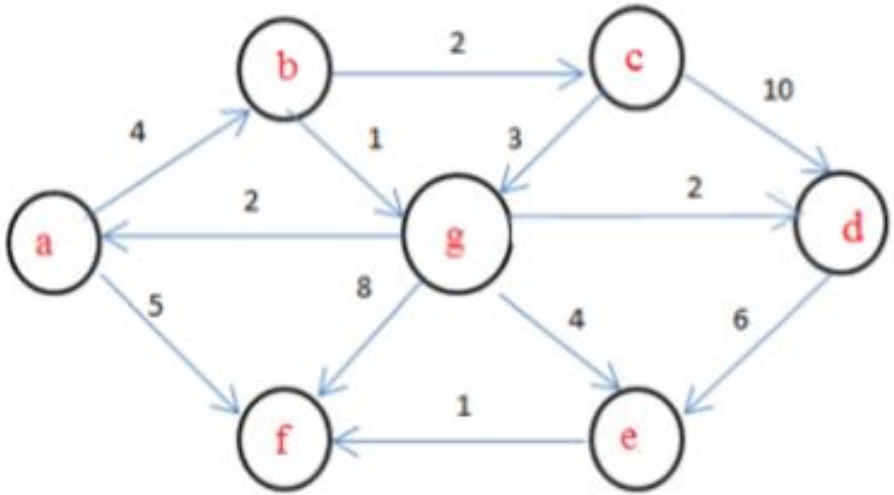
State space diagram for Water Jug Problem



Q. Consider the following graph:

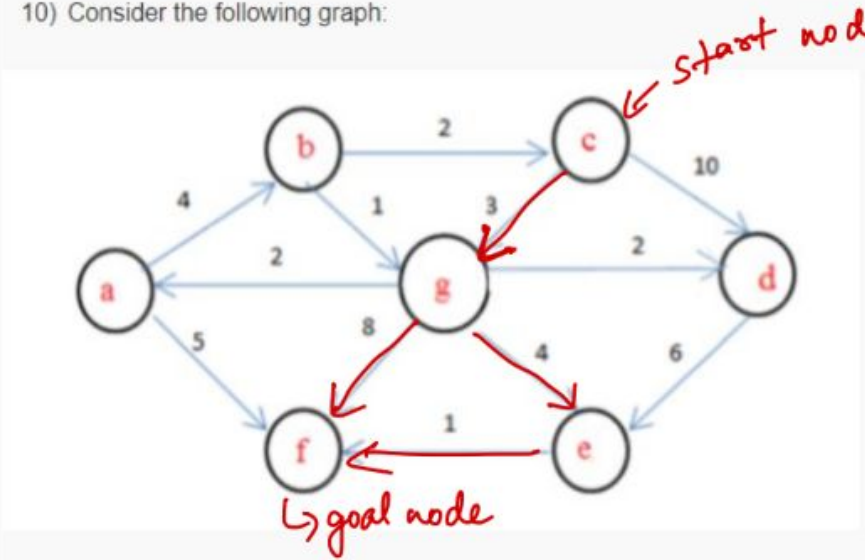
What is the minimum cost to reach vertex f starting from vertex c?

a. 11.0
b. 12.0
c. 6.0
d. 8.0



SOLUTION

10) Consider the following graph:



What is the minimum cost to reach vertex f starting from vertex c?

- 11.0
- 12.0
- 6.0
- 8.0 ✓

greedy:

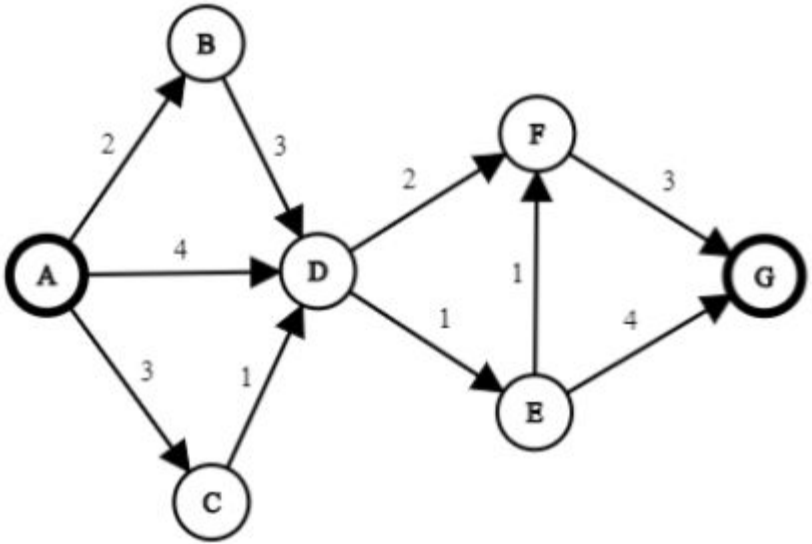
c → g, d
(3) (10)

g → f, e.
(3) (8), (4)

c → g → f = 11 ✓

c → g → e = 7 + 1 = 8.
↳ f(1)

Q. Consider the following directed graph, having A as the starting node and G as the goal node, with edge costs as mentioned, and the heuristic values for the nodes are given as {h(A)=7, h(B)=6, h(C)=5, h(D)=4, h(E)=3, h(F)=3, h(G)=0}



What is the order of exploration of nodes using A* algorithm in the above question?

Solved in class

	Path	h(x)	g(n)	f(n) = h(n) + g(n)
	A	7	0	7
	A-B	6	2	8
	A-C	5	3	8
	A-D ✓	4	4	8
	A-B-D	4	2+3	9
	A-C-D ✓	4	3+1	8
	A-D-B ✓	3	4+1	8
	A-D-F	3	4+2	9
	A-C-D-B ✓	3	3+1+1	8
	A-C-D-F	3	3+1+2	9
①	A-D-E-G ✓	0	4+1+4	9
	A-D-E-F ✓	3	4+1+1	9
②	A-C-D-E-G ✓	0	3+1+4	9
	A-C-D-E-F ✓	3	3+1+1+1	9
③	A-D-E-F-G ✓	0	4+1+3	9
④	A-C-D-E-F-G ✓	0	3+1+1+1+3	9

Q. In figure 1, illustrate uninformed algorithms BFS, DFS, DLS (L=1). Assume Initial node as **S** and goal node as **G**. Show the path traversed and cost.

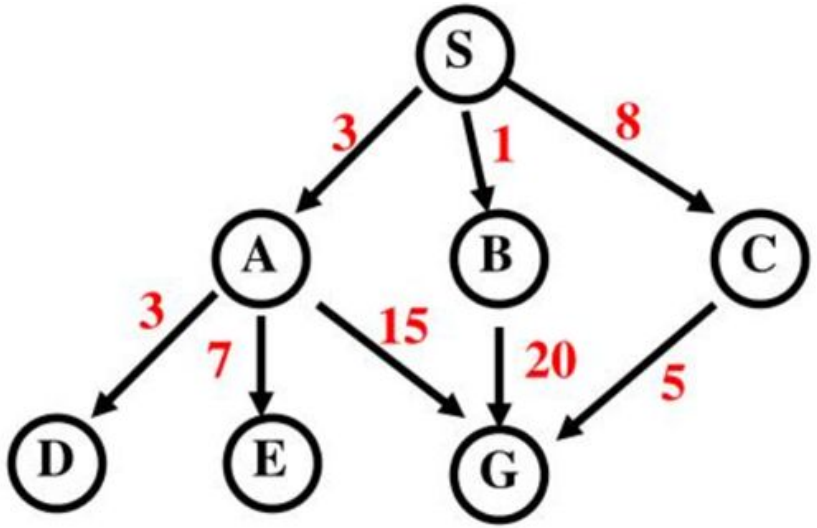
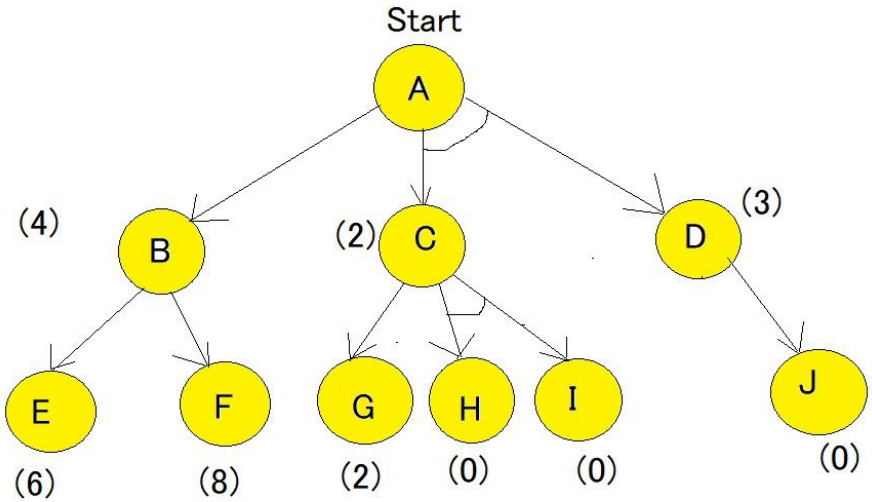
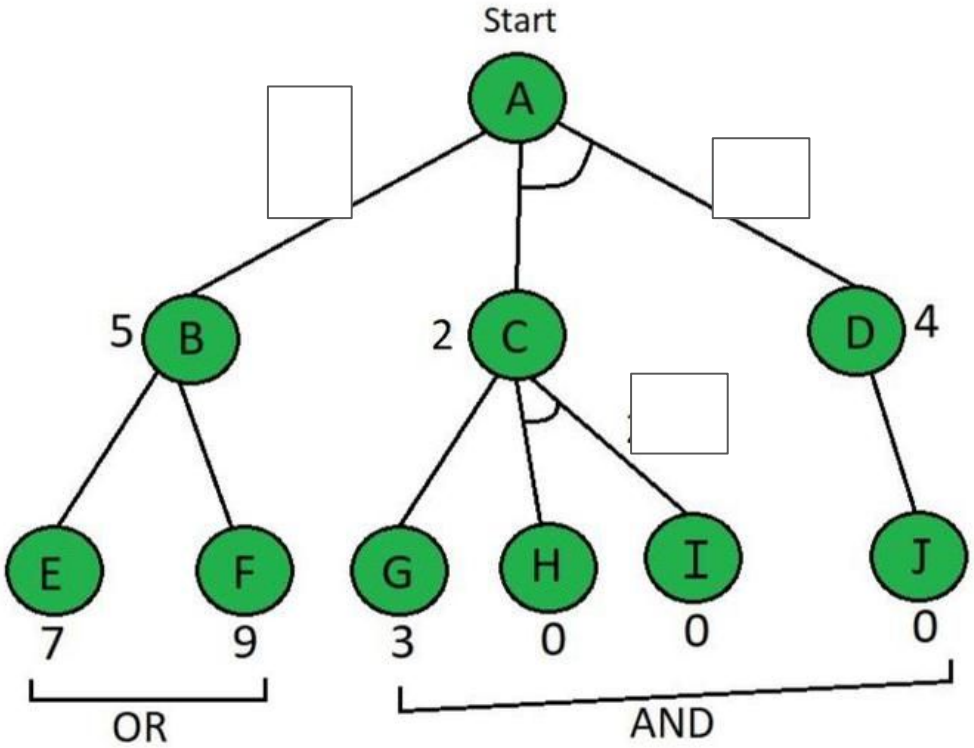


Figure 1

Q. Explain AO* step by step (**exploration**) to find the lowest cost path and the corresponding **lowest cost** from the starting node A to the goal node. All numbers in brackets are the heuristic values i.e., h(n). Consider each edge to have a value of 1.



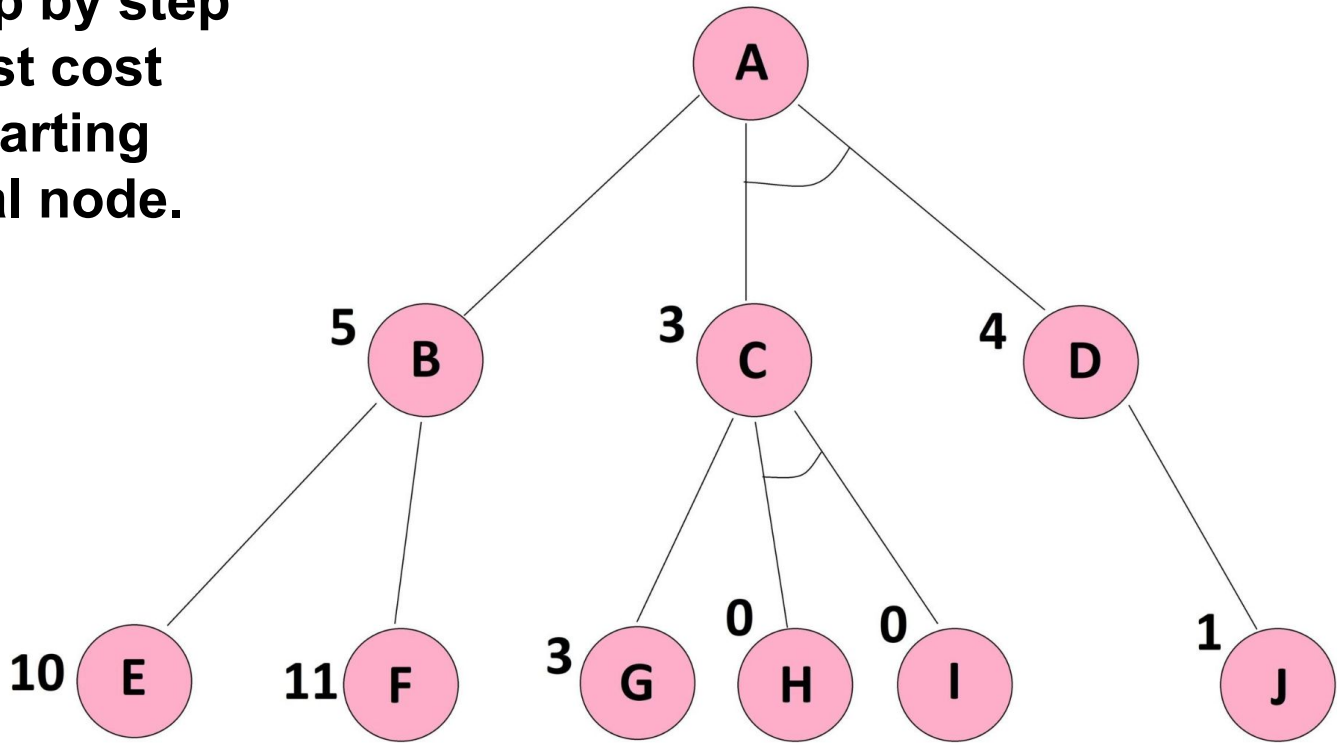
For solution check:
https://iq.opengenus.org/ao-algorithm/#:~:text=The%20AO*%20algorithm%20is%20a,to%20the%20informed%20search%20technique



Q. Below the Node, heuristic values i.e h(n) are given.
Edge length is considered as 1.
Get solved path using AO* search strategy.

Solution: https://www.geeksforgeeks.org/ao-algorithm-artificial-intelligence/

Explain AO* step by step to find the lowest cost path from the starting node to the goal node.



Step by step solution: https://www.baeldung.com/cs/ao-star-algorithm

Q. Given an initial state of a 8-puzzle problem and final state is to be reached. Find the most cost-effective path to reach the final state from initial state using A* Algorithm. Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

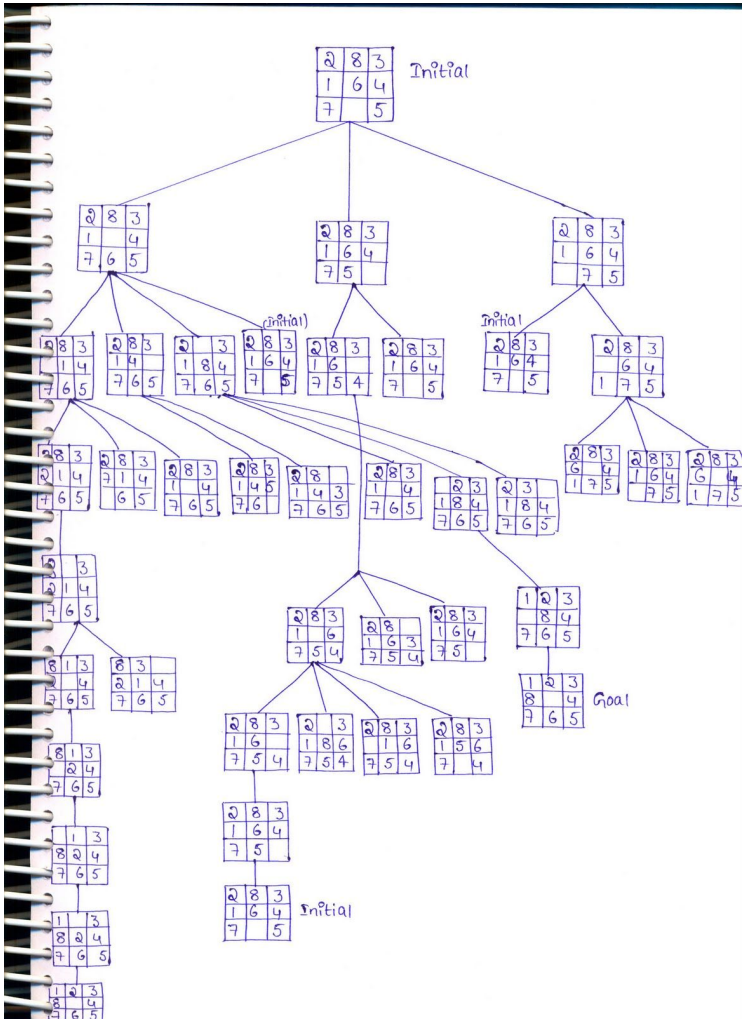
2	8	3
1	6	4
7		5

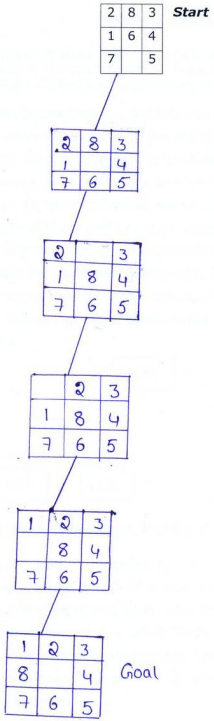
Initial State

1	2	3
8		4
7	6	5

Final State

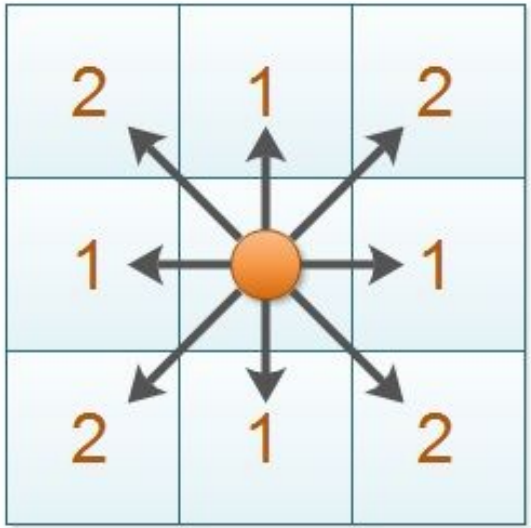
For solution check: https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/





Searrh tree for 8-puzzle

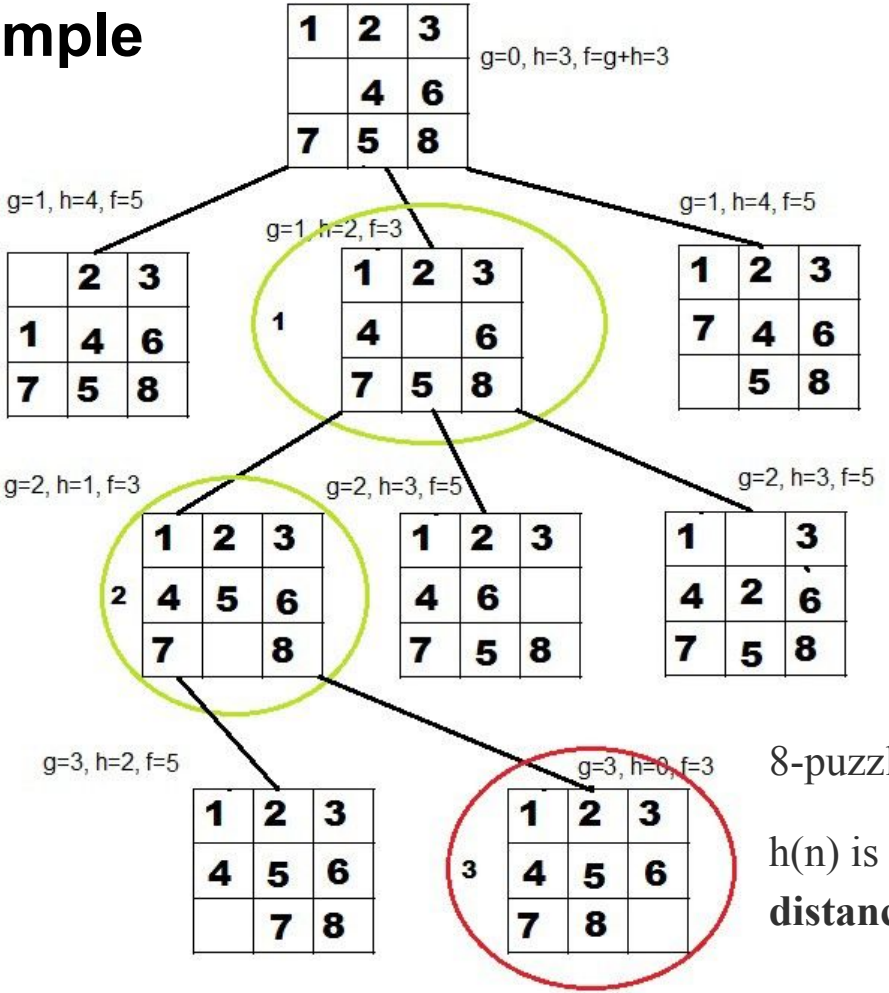
Manhattan Distance



$$|x_1 - x_2| + |y_1 - y_2|$$

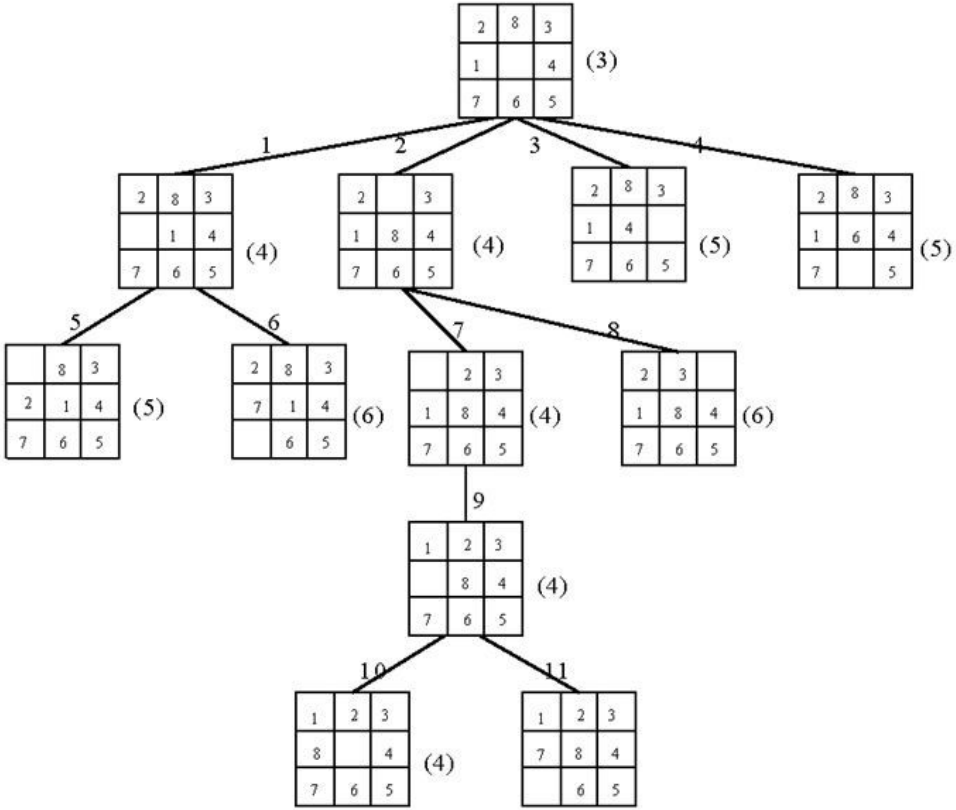
Required to calculate heuristics for the 8 puzzle problem**

Another Example



8-puzzle problem using A* algorithm

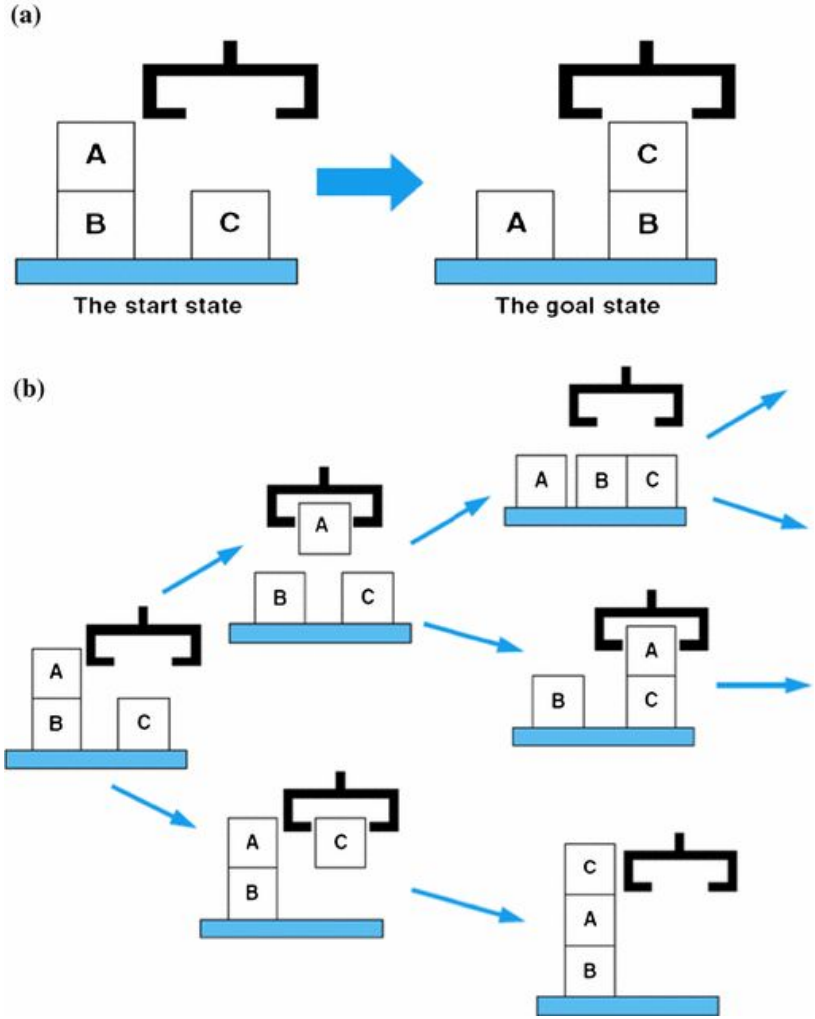
h(n) is calculated using **Manhattan distance**



An 8-puzzle problem solved by a best-first search scheme.

Robot block world

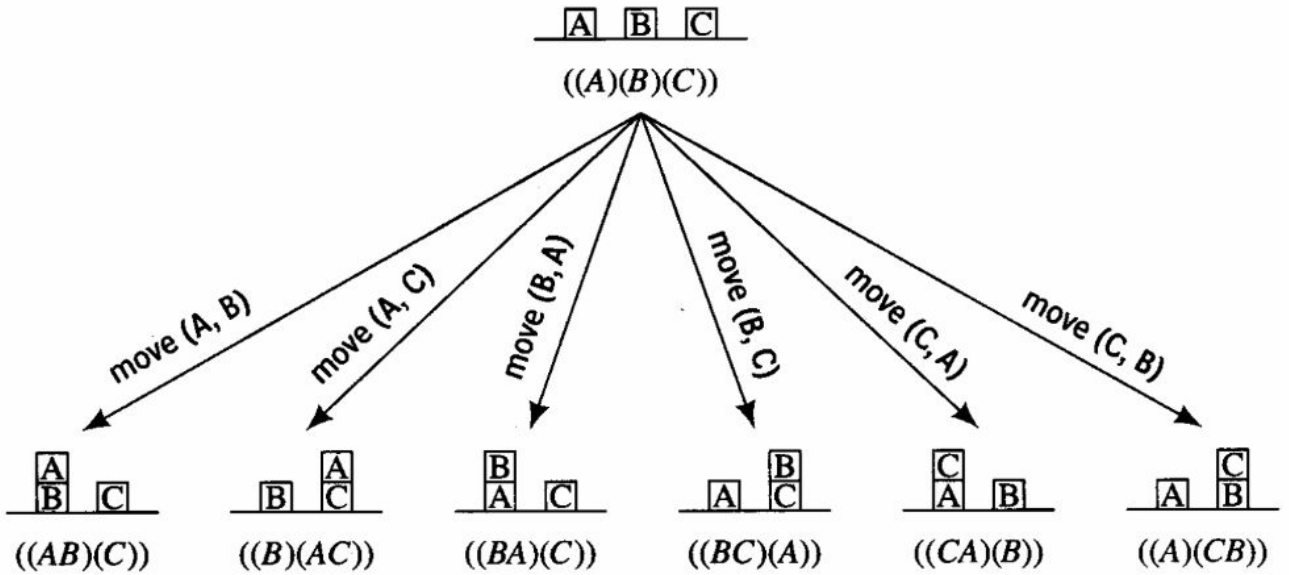
- Given a set of blocks in a certain configuration,
- Move the blocks into a goal configuration.
- Example :
  - (c,b,a) → (b,c,a)



In its basic form, the blocks world problem consists of **cubes in the same size which have all the color black**. A mechanical robot arm has to pick and place the cubes.

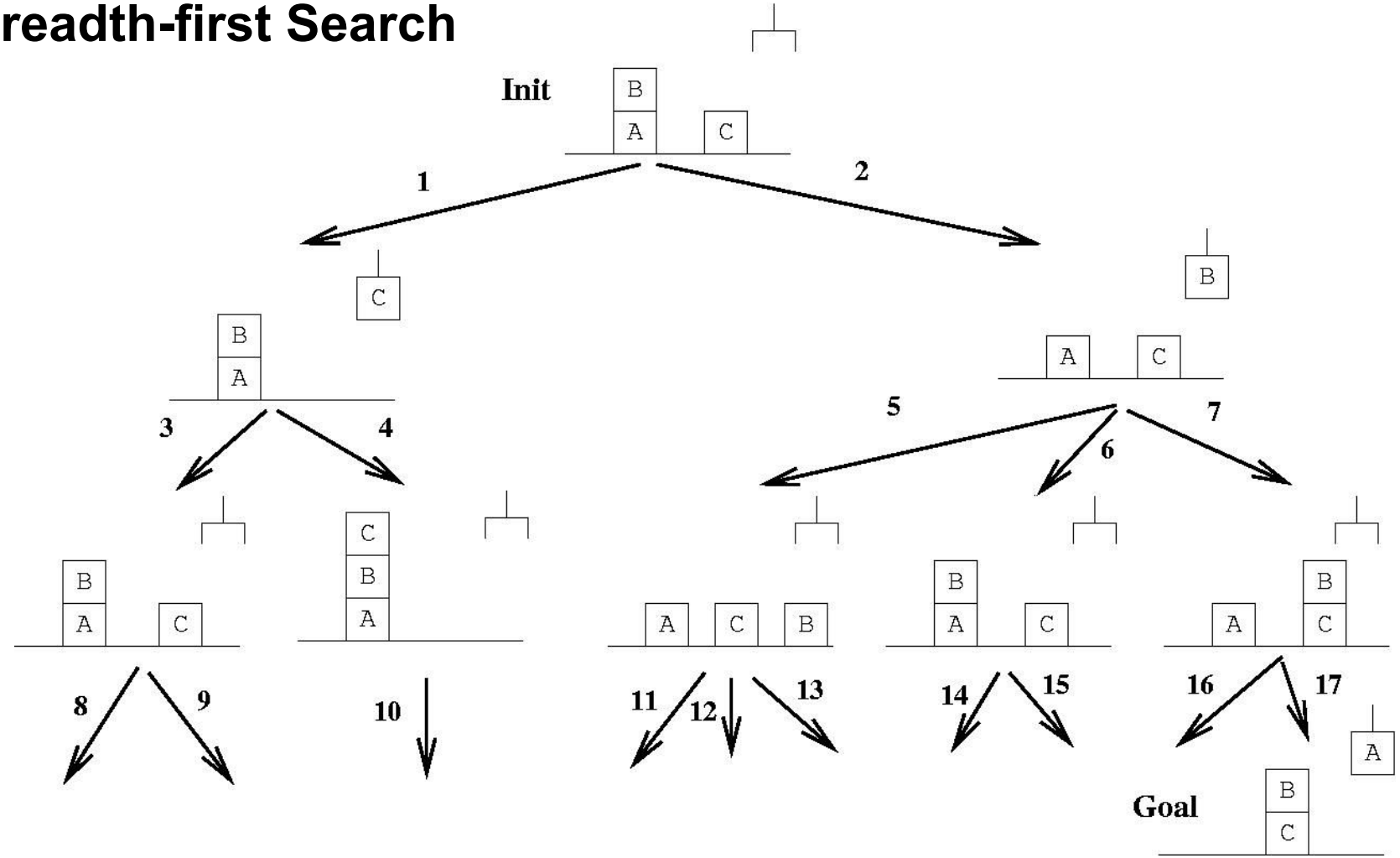
The goal is to build one or more vertical stacks of blocks, turn the initial state into the goal state. Only one block may be moved at a time, it may be placed either on the table or on top of another block.

Operator Description

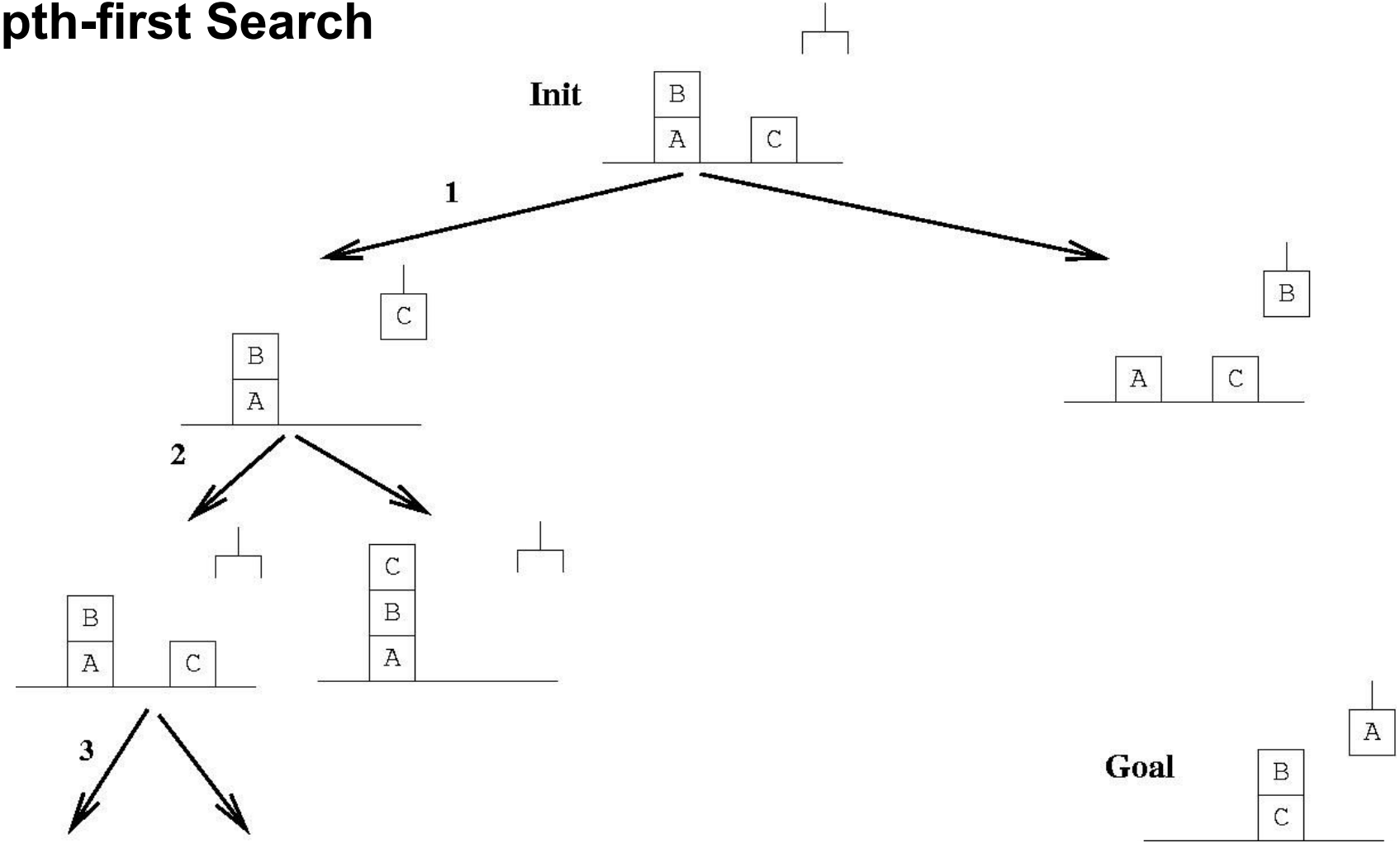


Effects of Moving a Block

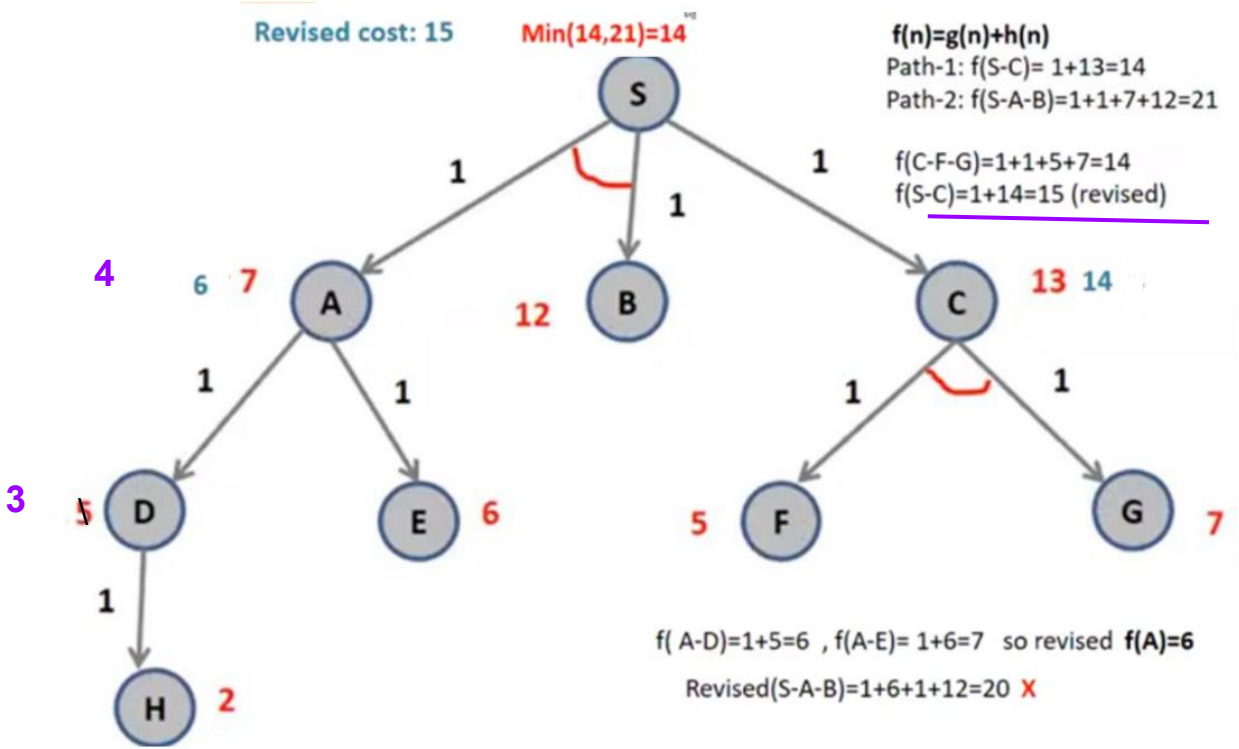
Breadth-first Search



Depth-first Search



Slide 127 correction*



Corrections: f(A-D) revised = 3 +1 = 4
Revised h(A) = min(7,4) = 4
Revised(S-A-B) = 4 + 12 + 1 + 1 = 18

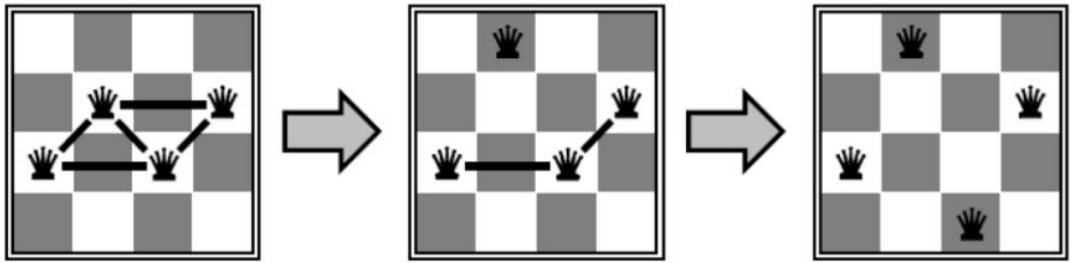
Final revised cost at S = min(15,18) = **15**

Hill Climbing Example

n-queens

- Put n queens on an n × n board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts.

➔ **Objective function: number of conflicts (no conflicts is global minimum)**

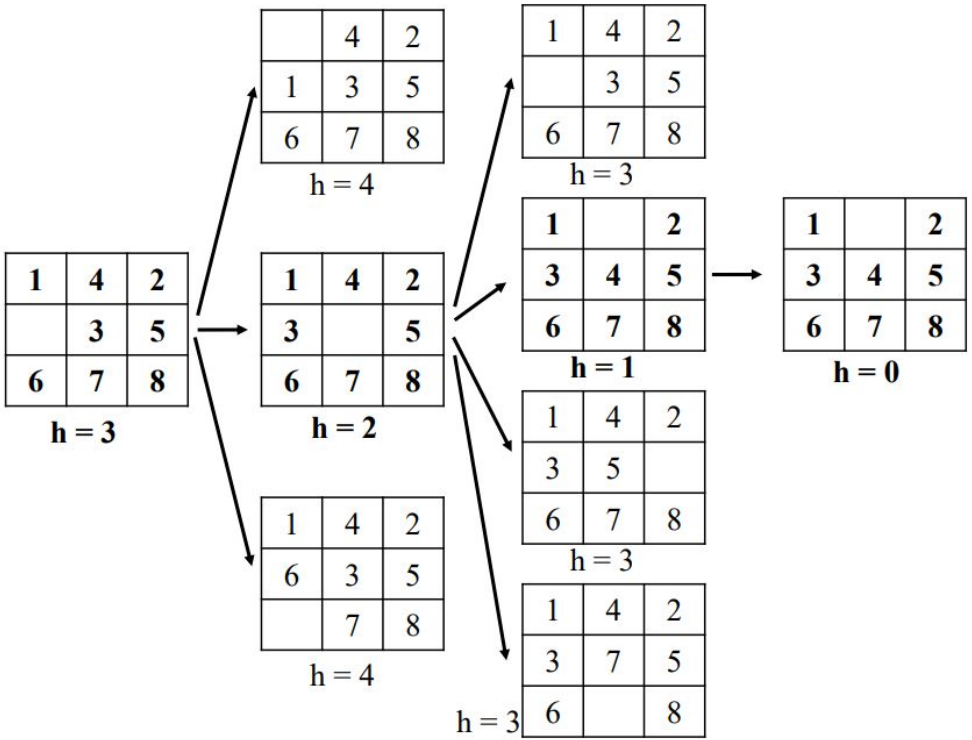


h = 5 h = 2 h = 0

- The successors of a state are all possible states generated by moving a single queen to another square in the same column (so each state has n*(n-1) successors).
- The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly.
- The global minimum of this function is zero, which occurs only at perfect solutions.

Hill Climbing Example

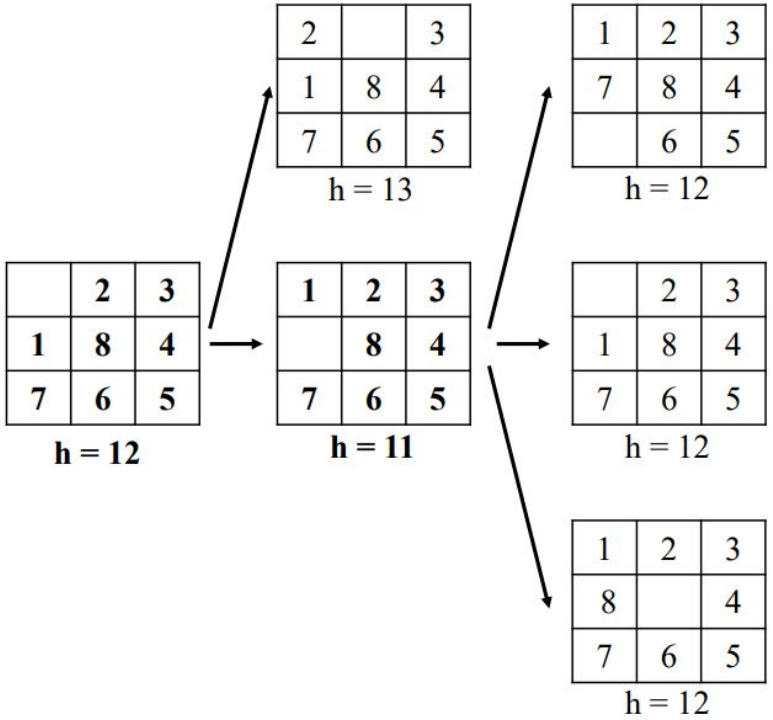
8-puzzle: a solution case



Heuristic function is Manhattan Distance

Hill Climbing Example

8-puzzle: stuck at local maximum



Heuristic function is Manhattan Distance

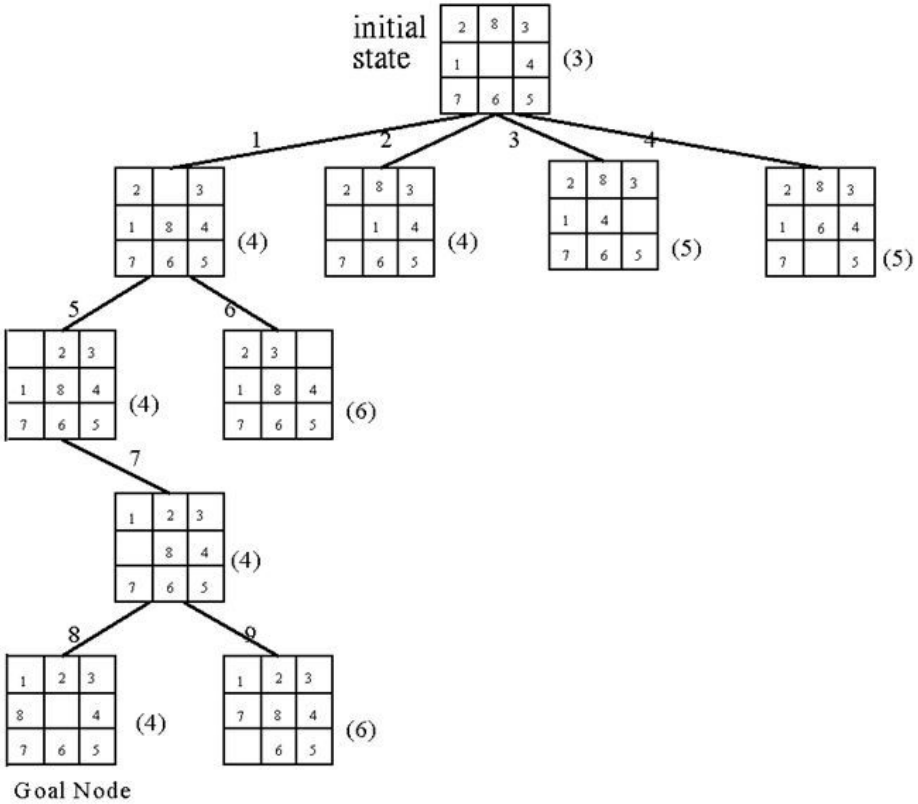
We are stuck with a local maximum.

Hill Climbing

- Hill Climbing is **NOT complete**.
- Hill Climbing is **NOT optimal**.

- **Why use local search?**
  - Low memory requirements – usually constant
  - Effective – Can often find good solutions in extremely large state spaces
  - Randomized variants of hill climbing can solve many of the drawbacks in practice.

- Many variants of hill climbing have been invented.

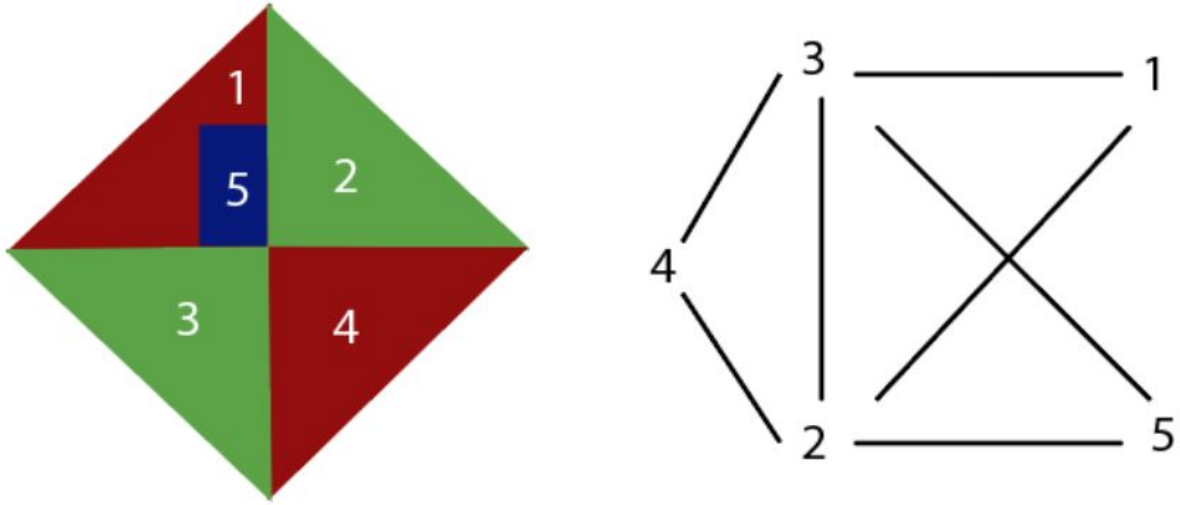


An 8-puzzle problem solved by a hill climbing method.

CSP Problems

Constraint satisfaction includes those problems which contains some constraints while solving the problem. CSP includes the following problems:

- **Graph Coloring:** The problem where the constraint is that no adjacent sides can have the same color.



Graph Coloring

- **Sudoku Playing:** The gameplay where the constraint is that no number from 0-9 can be repeated in the same row or column.

SUDOKU

4							5	9
2	6		5				3	
				9	2			
		2		6			1	
		3	8	1	9	7		
	7			3		5		
			3	4				
	3				6		2	7
5	9							6

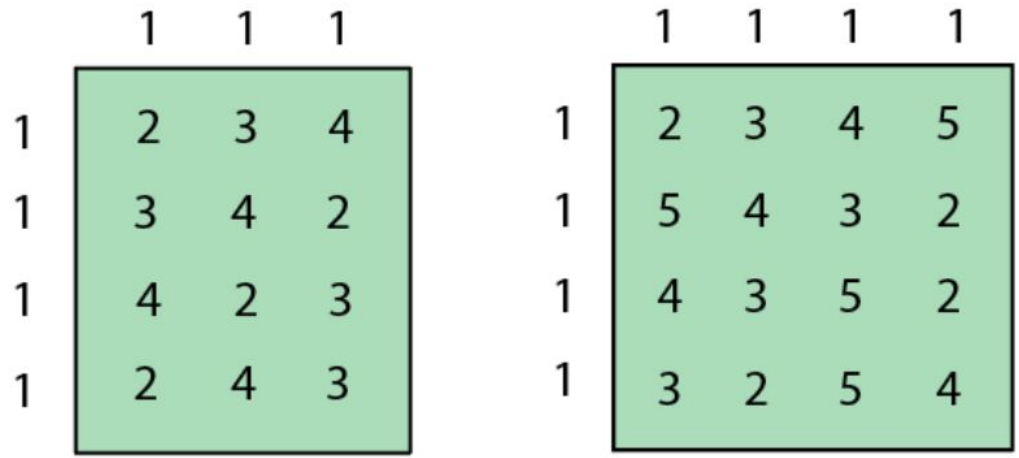
Puzzle

4	1	7	6	8	3	2	5	9
2	6	9	5	7	1	8	3	4
3	8	5	4	9	2	6	7	1
8	4	2	7	6	5	9	1	3
6	5	3	8	1	9	7	4	2
9	7	1	2	3	4	5	6	8
7	2	6	3	4	8	1	9	5
1	3	8	9	5	6	4	2	7
5	9	4	1	2	7	3	8	6

Solution

- **Crossword:** In crossword problem, the constraint is that there should be the correct formation of the words, and it should be meaningful.

- **Latin square Problem:** In this game, the task is to search the pattern which is occurring several times in the game. They may be shuffled but will contain the same digits.



Latin Squence Problem

- **Cryptarithmetic Problem:** This problem has one most important constraint that is, we cannot assign a different digit to the same character. All digits should contain a unique alphabet.

Constraint Satisfaction Problem (Cryptarithmetic Puzzles)

TWO+TWO = FOUR

there are seven solutions to the problem:

938+938=1876
928+928=1856
867+867=1734
846+846=1692
836+836=1672
765+765=1530
734+734=1468

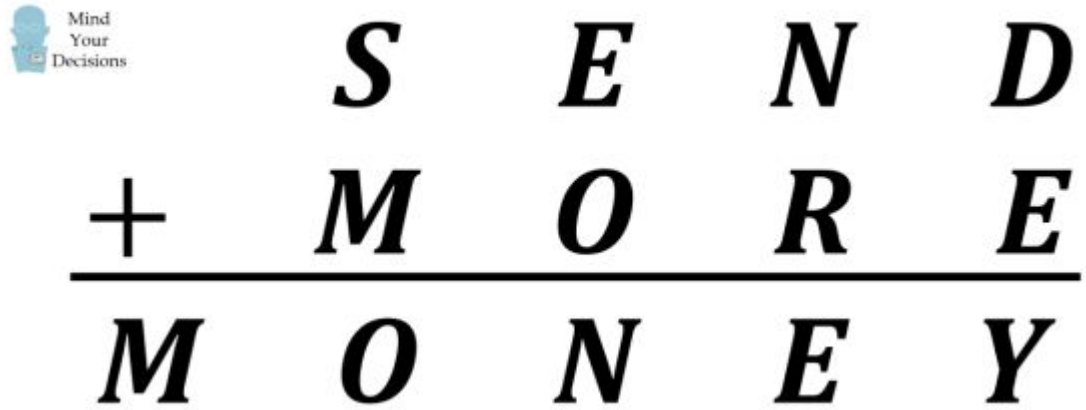
Solutions:

BASE + BALL = GAMES

If we allow G to be zero, the solutions are not unique. Below are all the 3 possible solutions.
{B=2, A=4, S=6, E=1, L=5, G=0, M=9}
{B=2, A=4, S=8, E=3, L=5, G=0, M=9}
{B=7, A=4, S=8, E=3, L=5, G=1, M=9}

```
  2461          2483          7483          BASE
  2455          2455          7455          BALL
---------  +  ----------  +  ----------  +  ---------  +
  04916         04938         14938         GAMES
```

If G has to be non-zero digit, the solution is unique.



SEND + MORE = 9567 + 1085 = 10652

https://mindyourdecisions.com/blog/2018/09/06/send-more-money-a-great-puzzle/

Backtracking Algorithm

Generally, the backtracking algorithm is **recursive**. It maintains an **incomplete variable assignment.** Here, **all variables are initially unassigned**. At each step, a variable is selected, and all possible values are subsequently assigned to it.

After attempting all possible values, the algorithm retraces its steps. **Consistency** in this fundamental backtracking algorithm is defined as the **satisfaction of all constraints whose variables are all assigned.**

The variables are ordered in some fashion, we try to **place first the variables that are more highly constrained or with smaller ranges.** We start assigning values to variables. We **check constraint satisfaction** at the earliest possible time and **extend an assignment if the constraints** involving the currently bound variables **are satisfied.**

Constraint Satisfaction Problems (CSP)

- The idea: represent states as a vector of feature values.
  - k-features (or variables)
  - Each feature takes a value. Each variable has a domain of possible values:
    - height = {short, average, tall},
    - weight = {light, average, heavy}

- In CSPs, the problem is to search for a set of values for the features (variables) so that the values satisfy some conditions (constraints).
  - i.e., a goal state specified as conditions on the vector of feature values.

Example: Sudoku

	2							
			6					3
	7	4		8				
					3			2
	8			4			1	
6			5					
				1		7	8	
5					9			
							4	

1	2	6	4	3	7	9	5	8
8	9	5	6	2	1	4	7	3
3	7	4	9	8	5	1	2	6
4	5	7	1	9	3	8	6	2
9	8	3	2	4	6	5	1	7
6	1	2	5	7	8	3	9	4
2	6	9	3	1	4	7	8	5
5	4	8	7	6	9	2	3	1
7	3	1	8	5	2	6	4	9

Example: Sudoku

- 81 **variables**, each representing the value of a cell.
- **Values**: a fixed value for those cells that are already filled in, the values {1-9} for those cells that are empty.
- **Solution**: a value for each cell satisfying the constraints:
  - No cell in the same column can have the same value.
  - No cell in the same row can have the same value.
  - No cell in the same sub-square can have the same value.

Formalization of a CSP

- More formally, a CSP consists of
  - A set of variables $V_1, \ldots, V_n$
  - For each variable a domain of possible values $Dom[V_i]$.
  - A set of constraints $C_1, \ldots, C_m$.

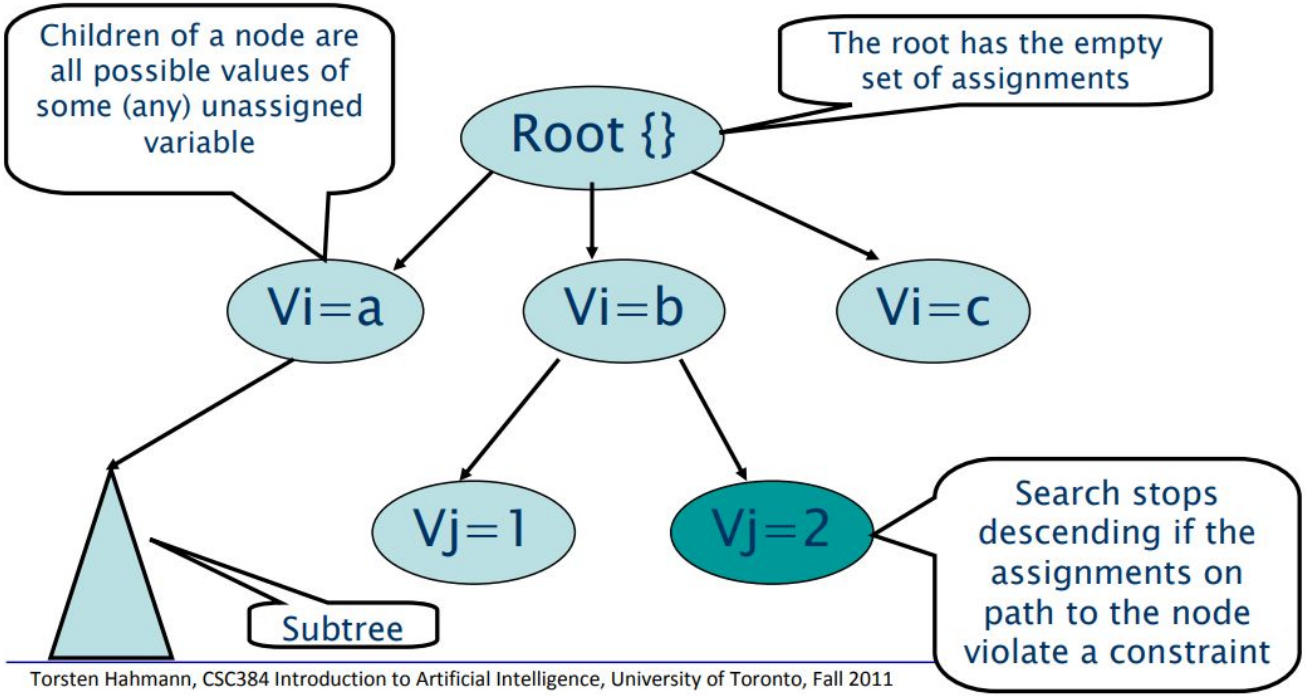
  - A solution to a CSP is an assignment of a value to all of the variables such that *every constraint is satisfied*.
  - A CSP is not satisfiable, if no solution exists.

CSP as a Search Problem

- Initial state: empty assignment
- Successor function: a value is assigned to any unassigned variable, which does not conflict with the currently assigned variables
- Goal test: the assignment is complete
- Path cost: irrelevant

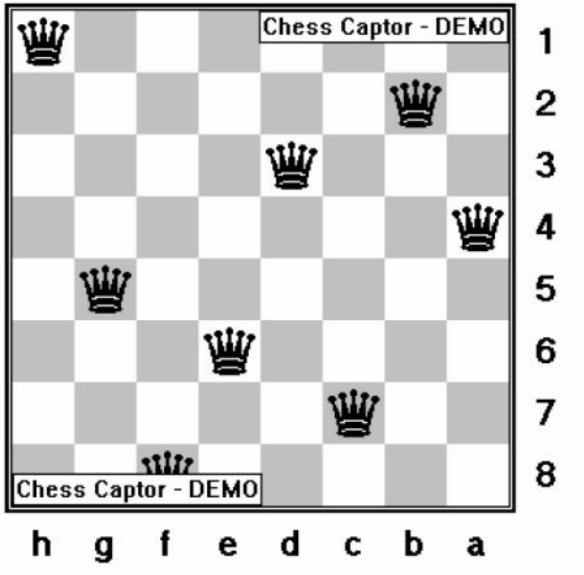
Backtracking Search

- The algorithm searches a tree of partial assignments.



Example: N-Queens

- Place N Queens on an N X N chess board so that no Queen can attack any other Queen.



Example: N-Queens

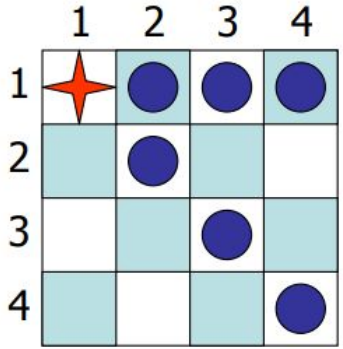
- Constraints:
  - Can't put two Queens in same column

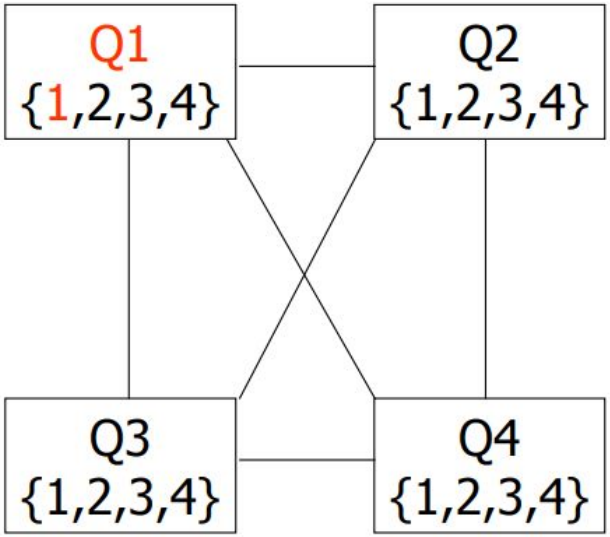
    $Q_i \neq Q_j$ for all $i \neq j$
  - Diagonal constraints

    $|Q_i - Q_j| \neq i - j$
    - i.e., the difference in the values assigned to Qi and Qj can't be equal to the difference between i and j.

4-Queens Problem

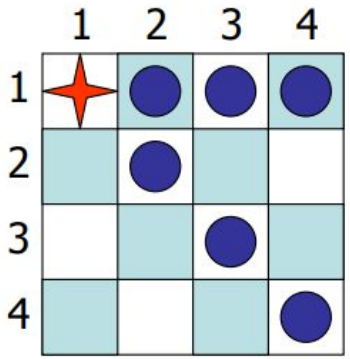
- Encoding with Q1, …, Q4 denoting a queen per column
  - cannot put two queens in same row (instead of same column)

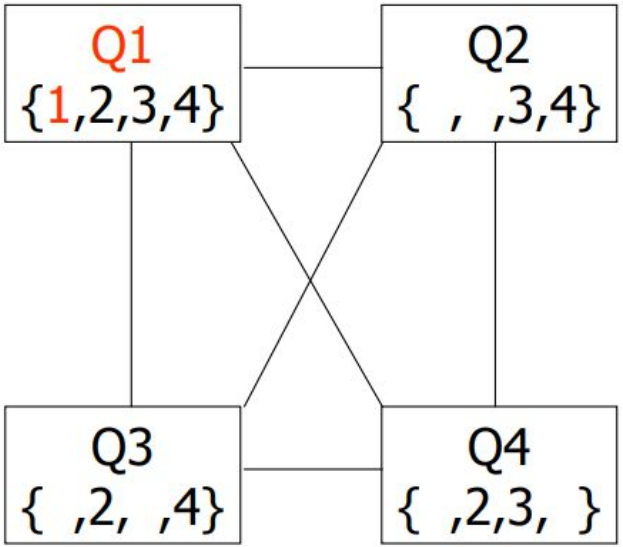




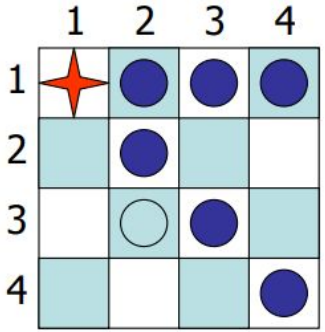
4-Queens Problem

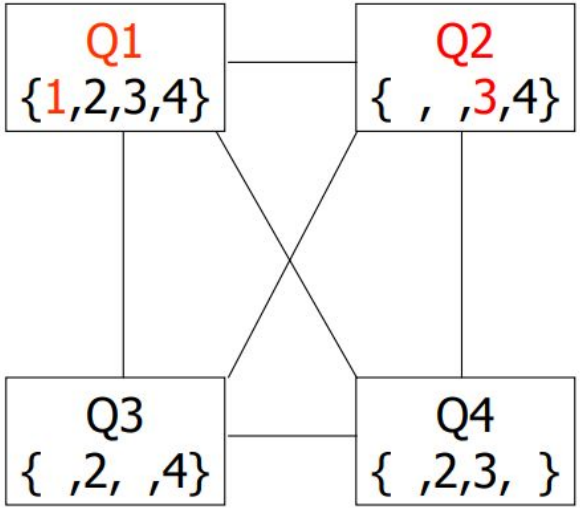
- Forward checking reduced the domains of all variables that are involved in a constraint with one uninstantiated variable:
  - Here all of Q2, Q3, Q4



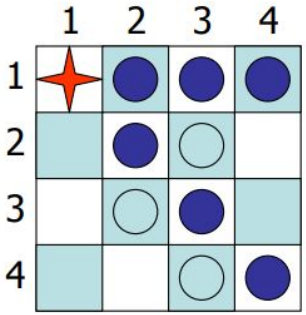


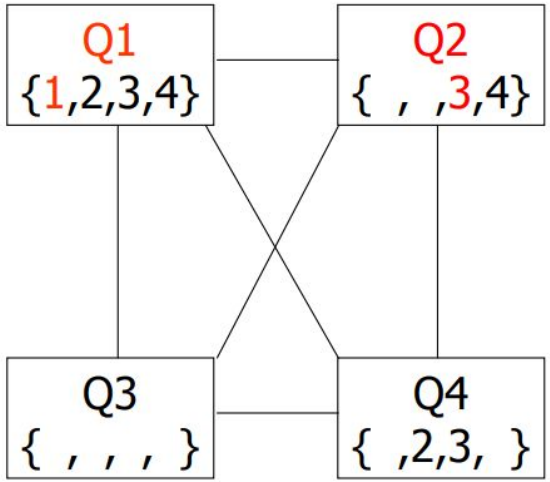
4-Queens Problem





4-Queens Problem

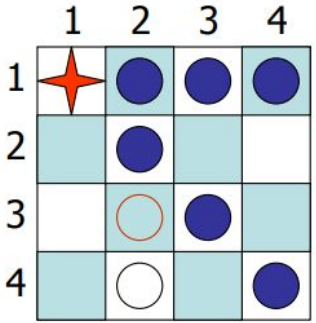


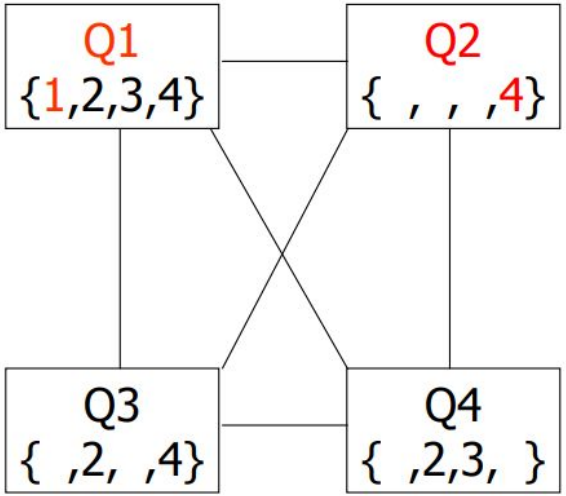


DWO

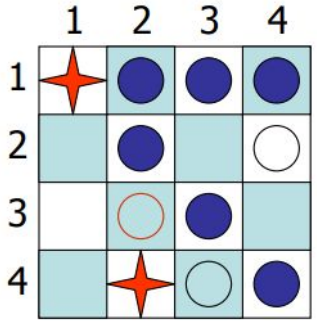
Dead node

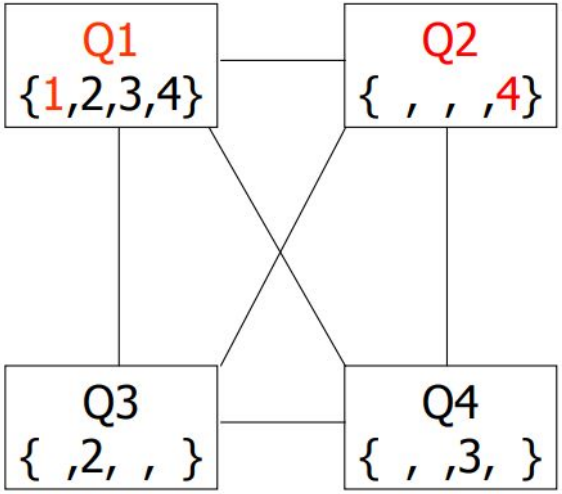
4-Queens Problem



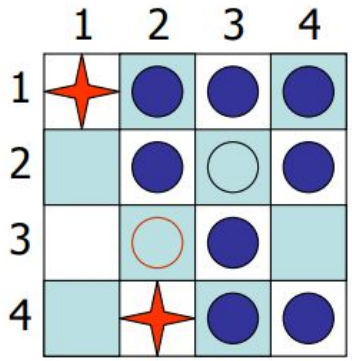


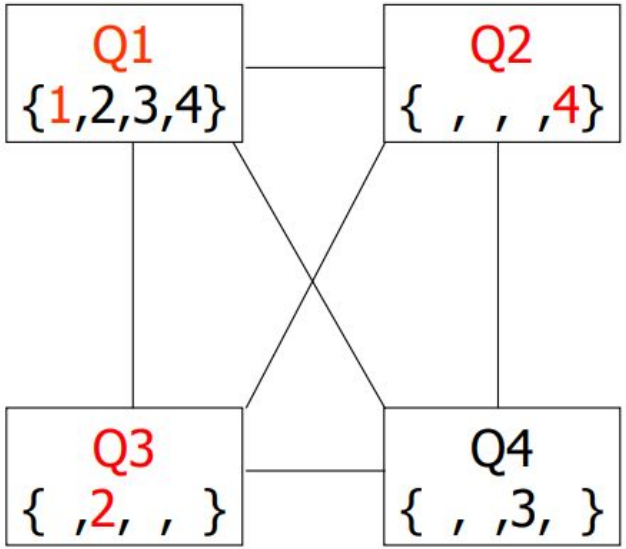
4-Queens Problem



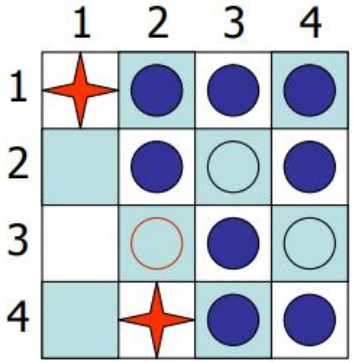


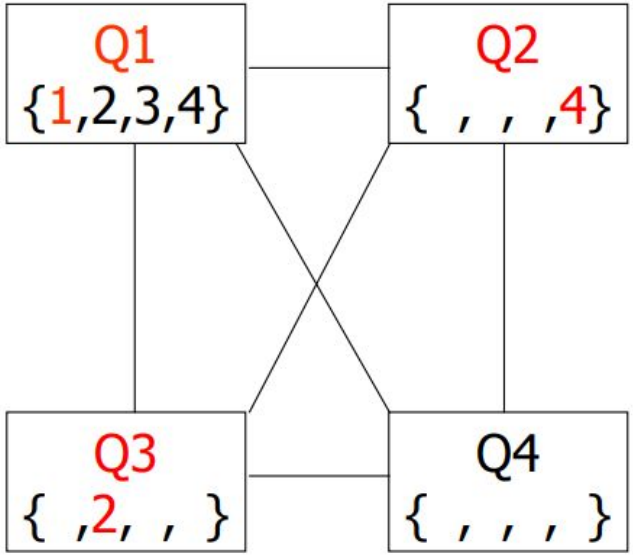
4-Queens Problem





4-Queens Problem

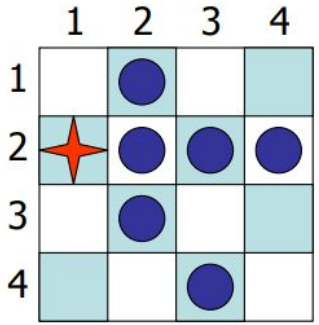


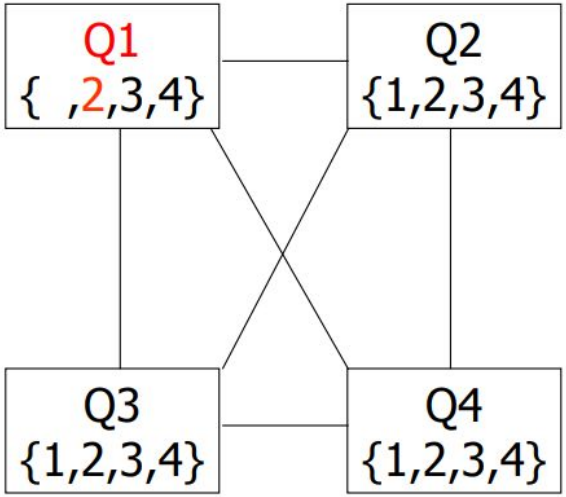


DWO

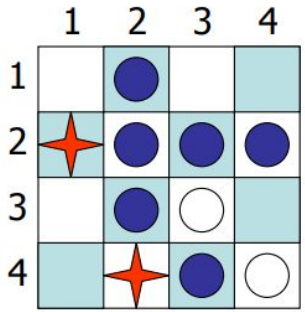
4-Queens Problem

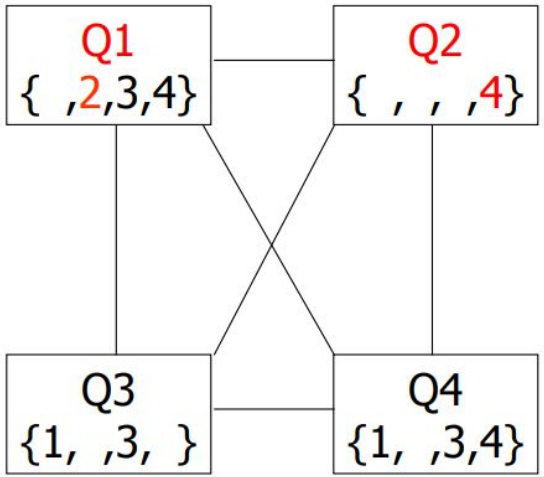
- Exhausted the subtree with Q1=1; try now Q1=2



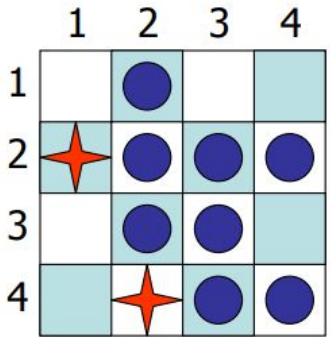


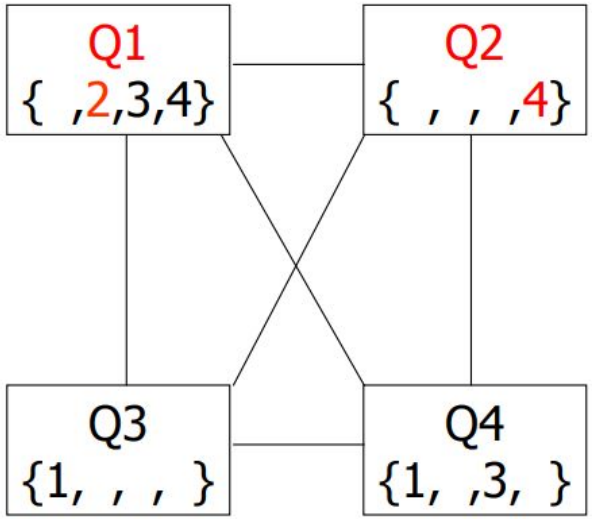
4-Queens Problem



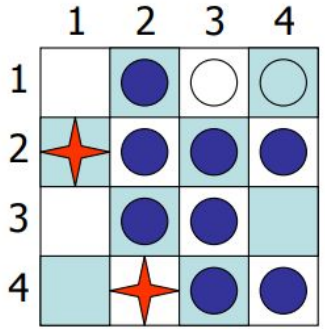


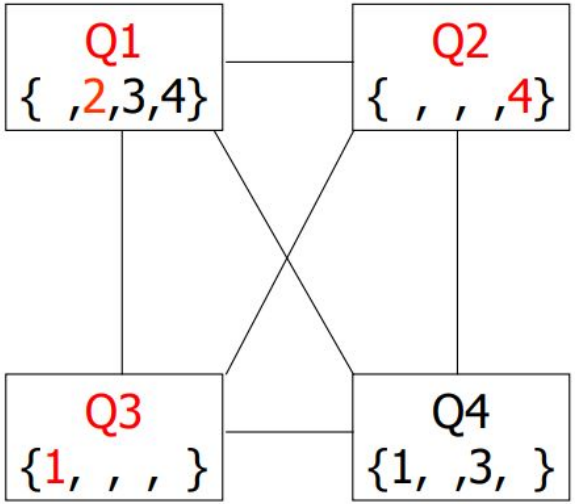
4-Queens Problem



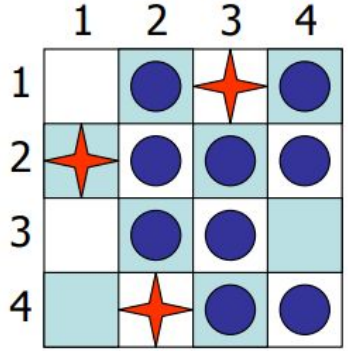


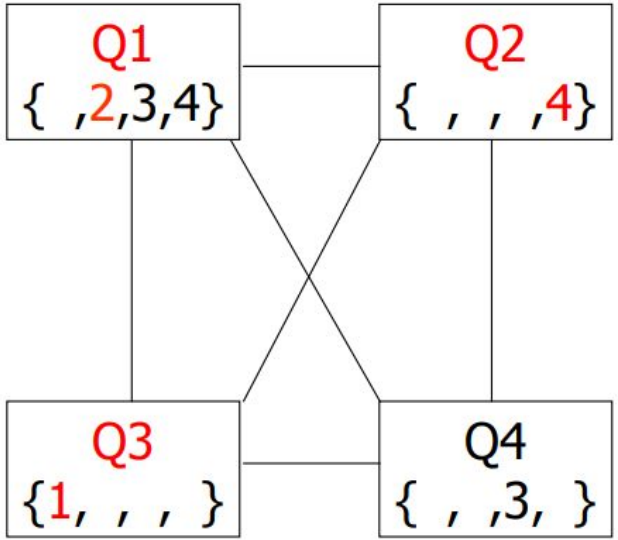
4-Queens Problem





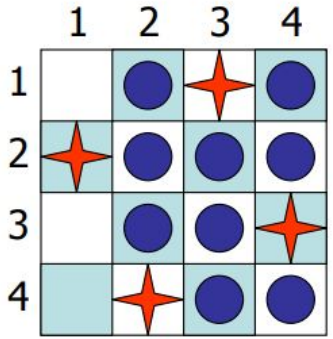
4-Queens Problem

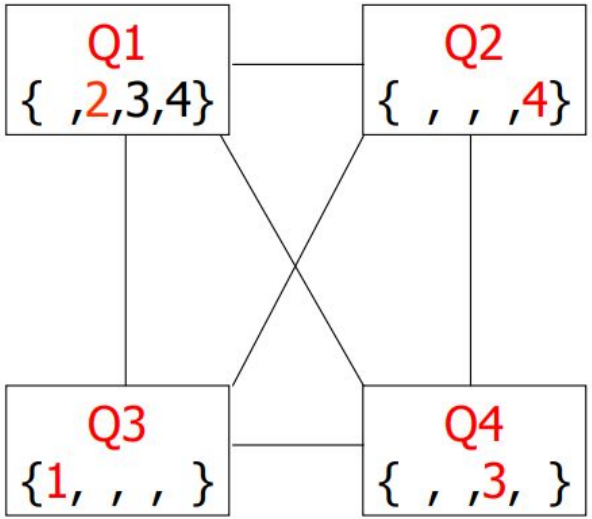




4-Queens Problem

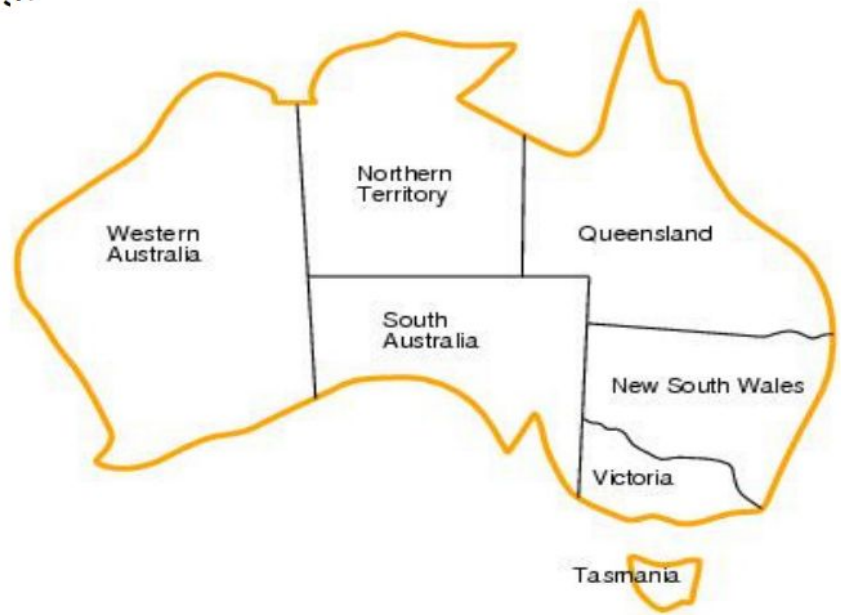
- We have now find a solution: an assignment of all variables to values of their domain so that all constraints are satisfied





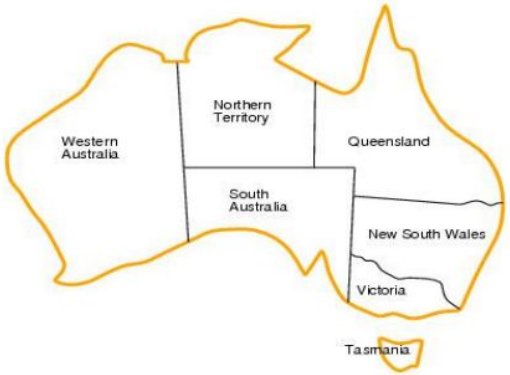
Example – Map Colouring

- Color the following map using *red*, *green*, and *blue* such that adjacent regions have different colors.



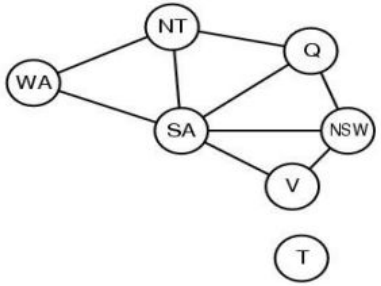
Example – Map Colouring

- **Modeling**
  - Variables: *WA, NT, Q, NSW, V, SA, T*
  - Domains: D_i=*{red, green, blue}*
  - Constraints: adjacent regions must have different colors.
    - E.g. *WA* ≠ *NT*



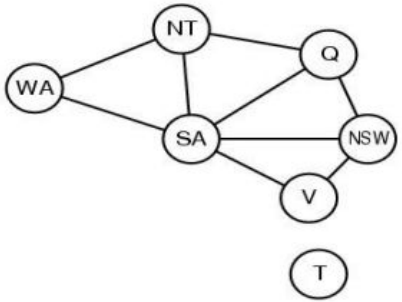
Example – Map Colouring

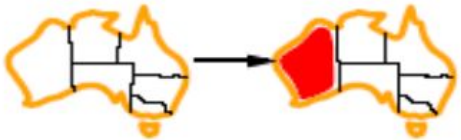
- *Forward checking idea*: keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values.



Example – Map Colouring

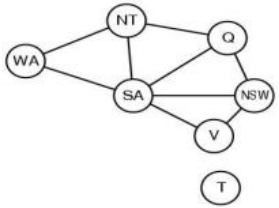
- Assign *{WA=red}*
- Effects on other variables connected by constraints to WA
  - *NT can no longer be red*
  - *SA can no longer be red*





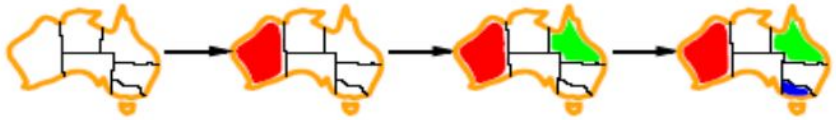
Example – Map Colouring

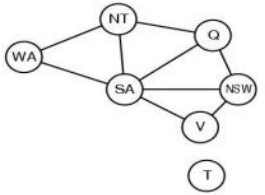
- Assign *{Q=green}*
- Effects on other variables connected by constraints with Q
  - *NT can no longer be green*
  - *NSW can no longer be green*
  - *SA can no longer be green*
- *MRV heuristic* would automatically select NT or SA next



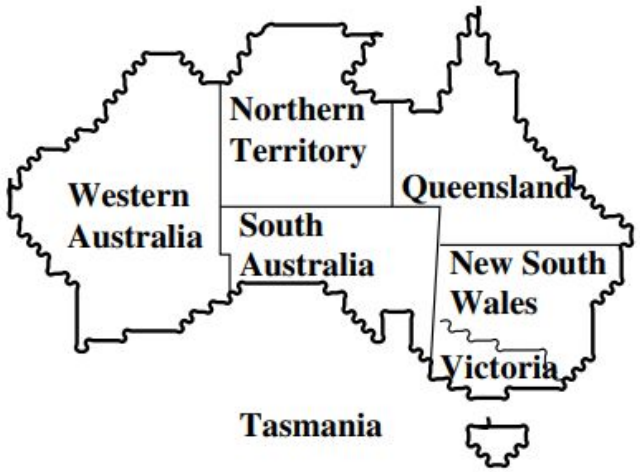
Example – Map Colouring

- Assign *{V=blue}*
- Effects on other variables connected by constraints with V
  - *NSW can no longer be blue*
  - *SA is empty*
- FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.



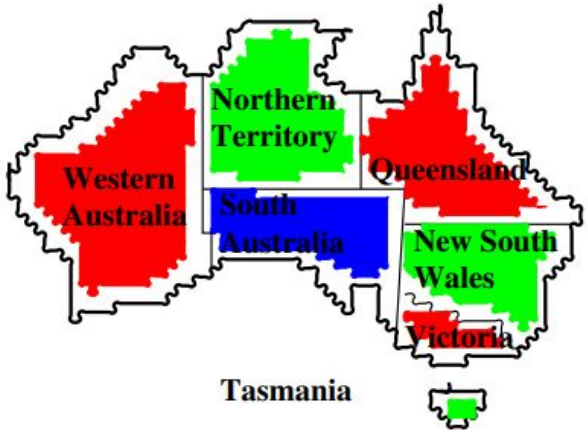


Working example: map-coloring



Variables: WA, NT, Q, NSW, V, SA, T
Domains: $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

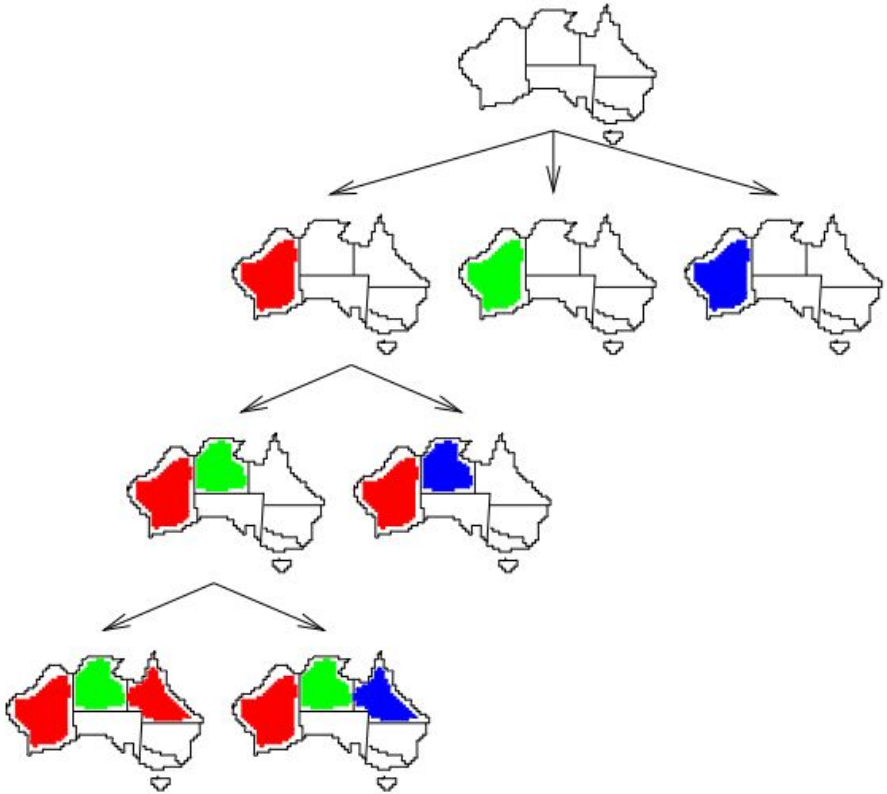
A solution for the map-coloring example



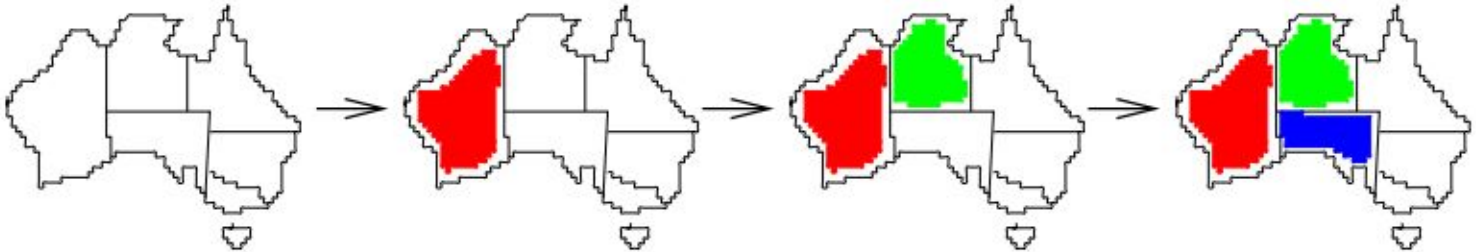
This solution satisfies all the constraints.

$$\{ \quad WA = red, NT = green, Q = red, NSW = green, \\ V = red, SA = blue, T = green\}$$

Backtracking example



Most constrained variable:
choose the variable with the fewest legal values

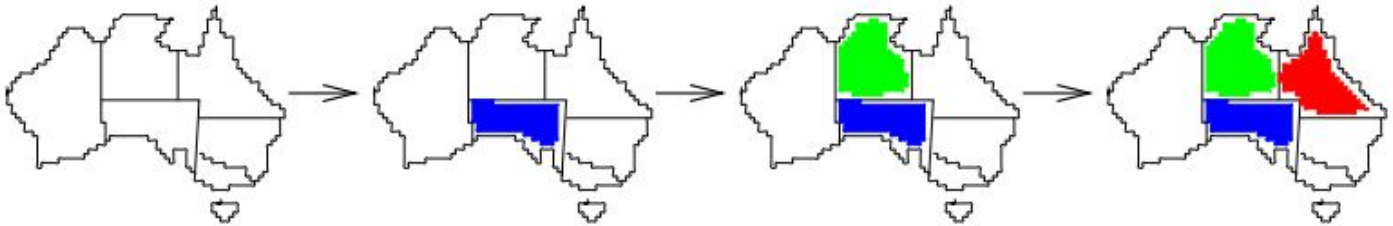


Most constraining variable strategy

Tie-breaker among most constrained variables

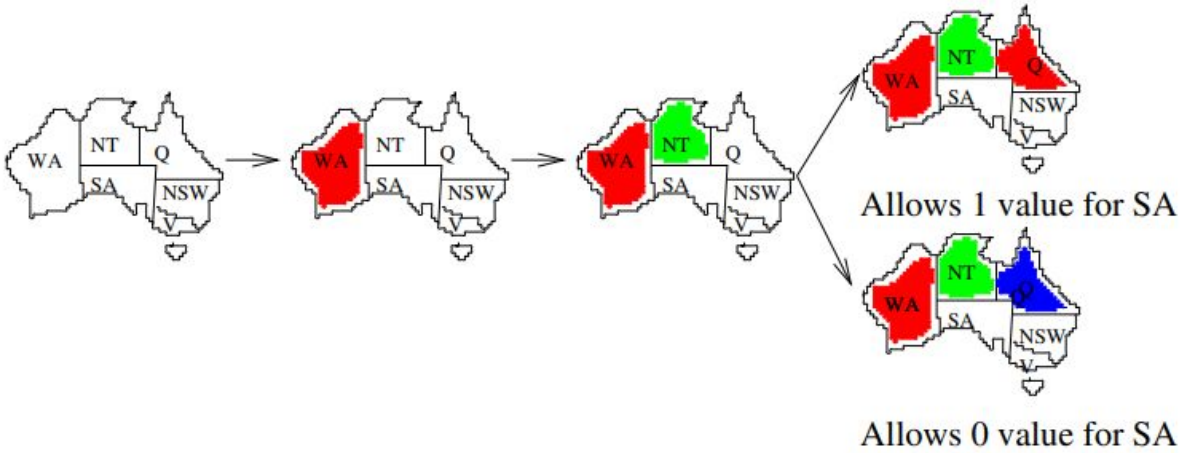
Most constraining variable:

choose the variable with the most constraints on the remaining variables



Least constraining value strategy

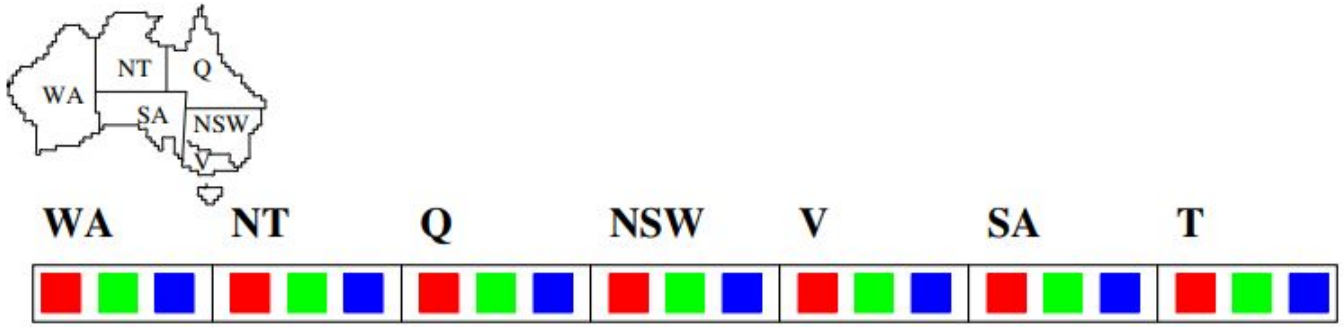
Given a variable, choose the *least constraining value*:
the one that rules out the fewest values in the remaining variables



Combining these heuristics makes 1000 queens feasible

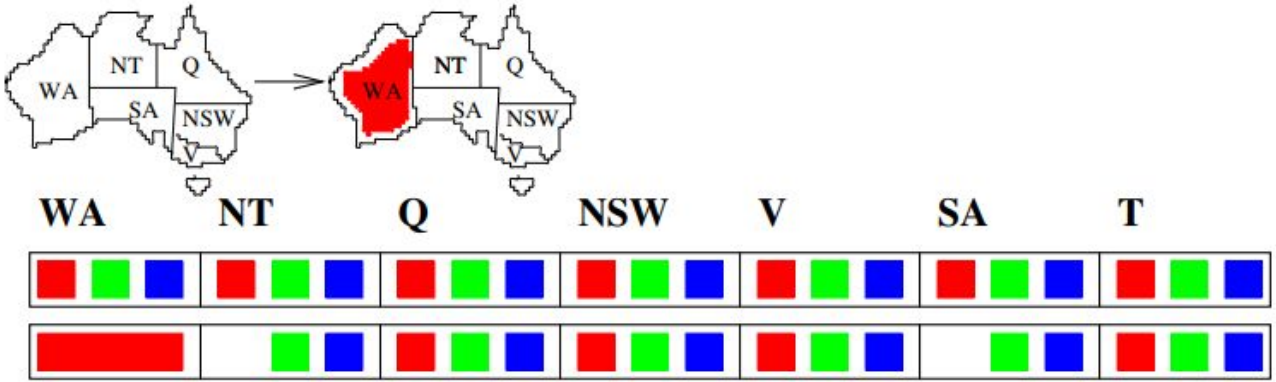
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



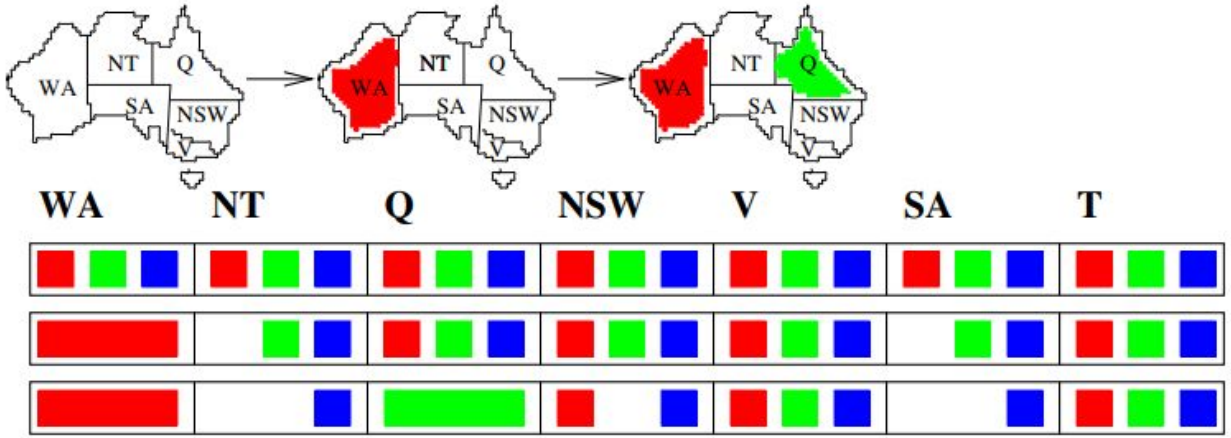
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



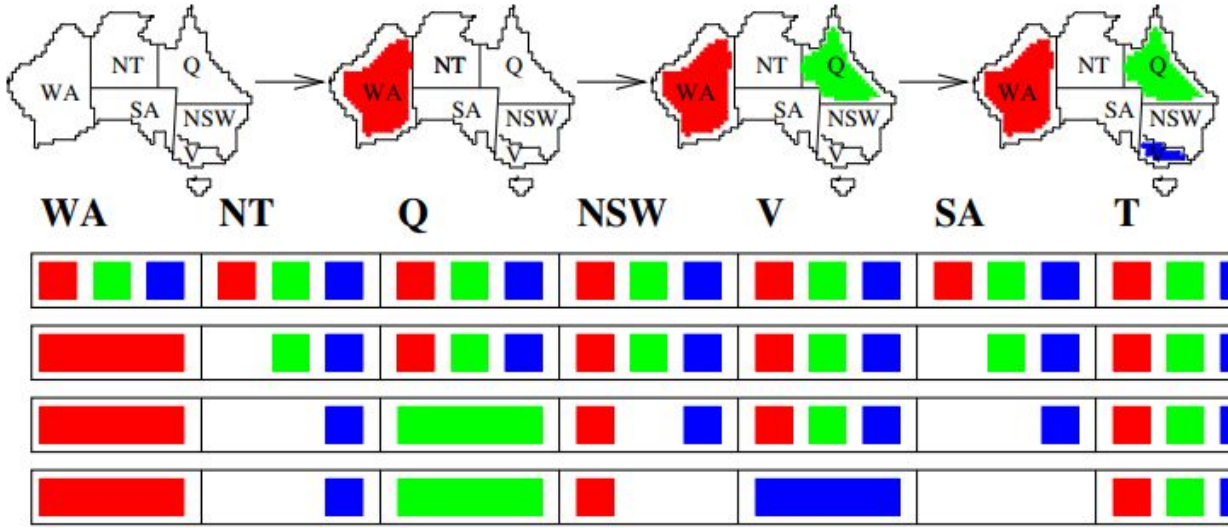
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



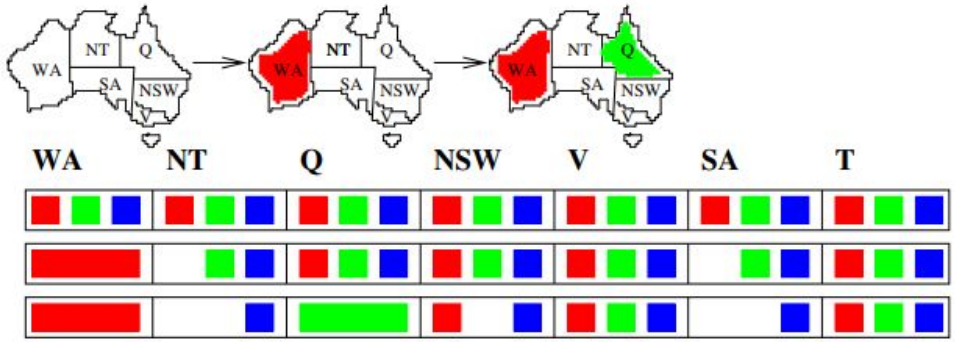
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



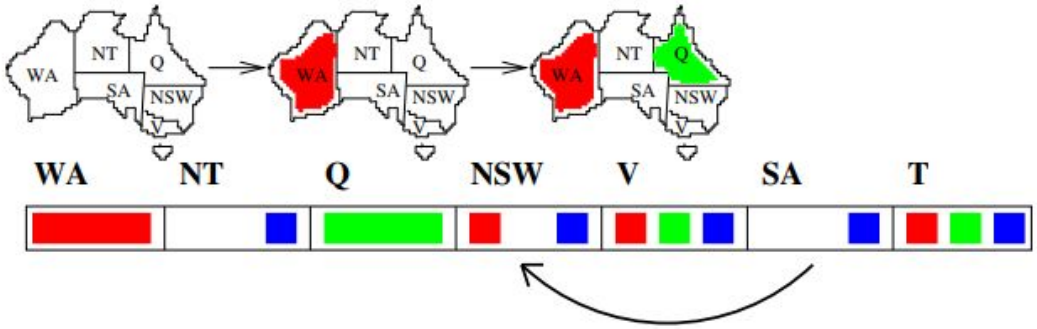
NT and *SA* cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

Arc consistency (1/4)

Simplest form of propagation makes each arc **consistent**

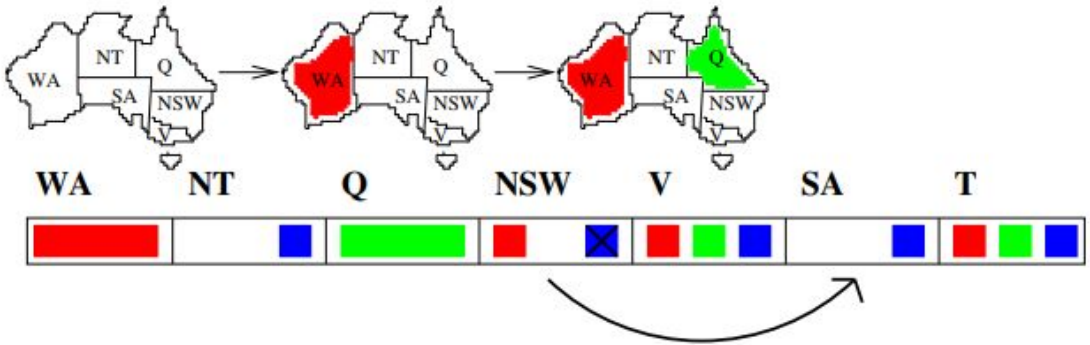
$X \rightarrow Y$ is *consistent* iff
for **every** value x of X there is **some** allowed y from Y



Arc consistency (2/4)

Simplest form of propagation makes each arc **consistent**

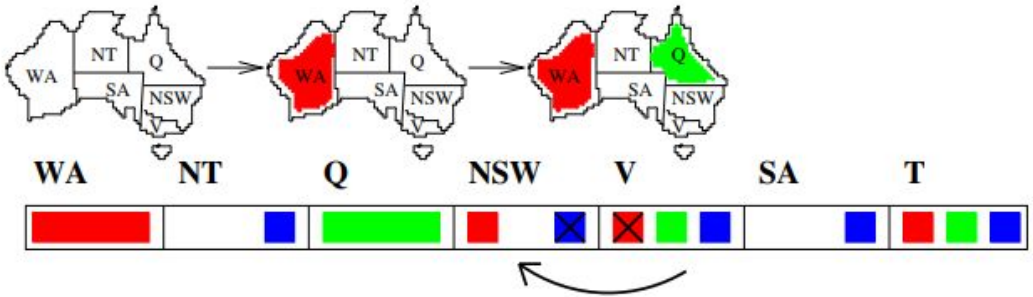
$X \rightarrow Y$ is *consistent* iff
for **every** value x of X there is **some** allowed y from Y



Arc consistency (3/4)

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is *consistent* iff
for **every** value x of X there is **some** allowed y from Y

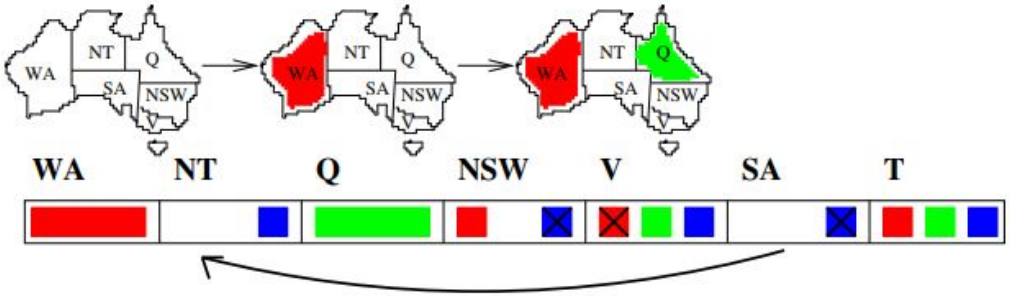


If X loses a value, neighbors of X need to be rechecked

Arc consistency (4/4)

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is *consistent* iff
for **every** value x of X there is **some** allowed y from Y



If X loses a value, neighbors of X need to be rechecked
Arc consistency detects failure earlier than forward checking
Can be run as a preprocessor or after each assignment

Many real-world applications of CSP

- Assignment problems
  - who teaches what class
- Timetabling problems
  - exam schedule
- Transportation scheduling
- Floor planning
- Factory scheduling
- Hardware configuration
  - a set of compatible components

Difference Between Greedy Best-First Search and Hill Climbing Algorithm

Properties	Greedy Best First Search	Hill Climbing Algorithm
Definition	A search algorithm that does not take into account the full search space but instead employs heuristics to choose the best route to a goal node.	An approach to searching that skips the full search space and instead chooses the best path to a goal node using heuristics.
Goal	To always choose the path with the lowest heuristic cost in order to reach the objective node as rapidly as feasible.	to discover the highest point in the search space, even if it is not the global maximum, in order to optimize a solution.

Heuristics	It estimates the cost of getting to the target node using heuristics.	It evaluates nearby solutions using heuristics.
Memory	It doesn't have to keep track of prior nodes.	Only keeps track of the most recent and effective solutions.
Completeness	Not guaranteed to find a solution.	Not always possible to locate the global maximum.
Efficiency	With a suitable heuristic, it is possible to locate a solution quickly in a wide search space.	It can be effective in locating a local maximum, but it can become trapped in a local optimum.
Search space	It uses a breadth-first approach to investigating the search space.	It uses a depth-first approach to investigate the search space.
Backtracking	Does not require backtracking.	It can backtrack steps if a better answer cannot be found.
Examples	It is used in situations involving pathfinding and graph traversal.	It used in scheduling and logistics optimization problems.

Hill-climbing and greedy algorithms are both heuristics that can be used for optimization problems.

In a **hill-climbing** heuristic, you start with an initial solution. Generate one or more neighboring solutions. Pick the best and continue until there are no better neighboring solutions. This will generally yield one solution. In hill-climbing, we need to know how to evaluate a solution, and how to generate a "neighbor."

In a **greedy** heuristic, we need to know something special about the problem at hand. A greedy algorithm uses information to produce a single solution.

In **BFS**, it's about **finding the goal.** So it's about picking the best node (the one which we hope will take us to the goal) among the possible ones. We keep trying to go towards the goal.

But in **hill climbing**, it's about maximizing the target function (i.e., **minimising heuristic cost)**. We pick the node which provides the highest ascent. **So unlike BFS, the 'value' of the parent node is also taken into account.** If we can't go higher, we just give up. In that case we may not even reach the goal. We might be at a local maxima.

Ask the question what is our **aim**... the **final goal state**? or the **Best path to reach the goal state**

> The **Best First Search** is a systematic search algorithm where systematicity is achieved by **moving forward iteratively on the basis of finding out the best heuristic value** for the adjacent nodes for every current node.

Here the **evaluation function (heuristic function) calculates the best possible path to achieve the goal state**. So here we could see the Best First search is concerned about the best PATH to reach to the goal state.

However there are many problems where "**Path to the Goal**" is **not a concern**, the only thing is concerned is **to achieve the final state** in any possible ways or paths. (For eg: **8-queens problem**).

Hence for this **local search algorithms** are used.

> Local search algorithms operate using a **single current node** and **generally move only to neighbor** of that node.

Hill Climbing algorithm is a **local search algorithm**. So here we need to understand the approach to get to the goal state not the best path to reach when thinking about hill climbing.

Hill Climbing algorithm **does not look ahead beyond the immediate neighbors of the current state**. It is only concerned with best neighboring node to expand.

Whereas, Best First Search algorithm **look ahead of the immediate neighbors** to find the best path to the goal(using Heuristic evaluation) and then proceeding with the best one. **So difference lies in approach of local searches and systematic search algorithms.**

ALPHA BETA PRUNING

Example 1: https://www.javatpoint.com/ai-alpha-beta-pruning

Example 2: https://www.codingninjas.com/codestudio/library/the-alpha-beta-pruning-algorithm