# Solving Problems By Searching

# Well-defined Problems

- There are four components for defining a problem:
  - The *initial state* that the agent starts in.
  - A description of the possible *actions* available to the agent.
    - The most common formulation uses a *successor function*. Given a particular state x, SUCCESSOR-FN(x) returns a set of {action, successor} ordered pair. For example, from the state *In(Arad)*, the successor function would return {(*Go(Sibui), In(Sibui)), (Go(Timisoara), In(Timisoara)), Go((Zerind), In(Zerind))*}
    - The *state space* is the set of all states reachable from the initial state.
    - A *path* in the state space is a sequence of states connected by a sequence of action.
  - The *goal test*, which determines whether a given state is a goal state. For example, in chess, the goal is to reach a state called “checkmate”.
  - A *path cost* function that assigns a numeric cost to each path.
    - A path can be described as the sum of the costs of the individual actions along the path.
    - The *step cost* of taking action ***a*** to go from state x to state y is denoted by c(x, a, y),

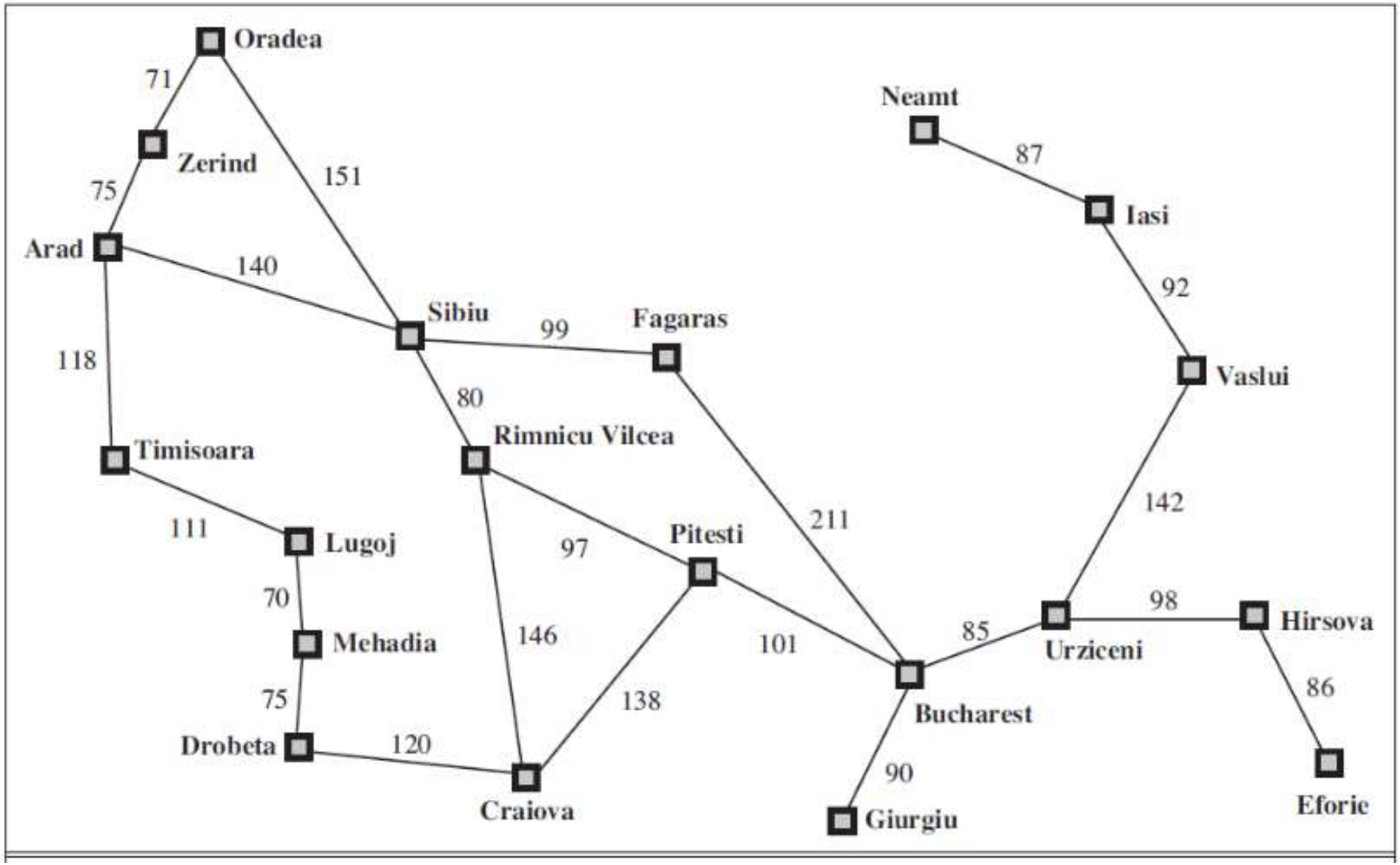
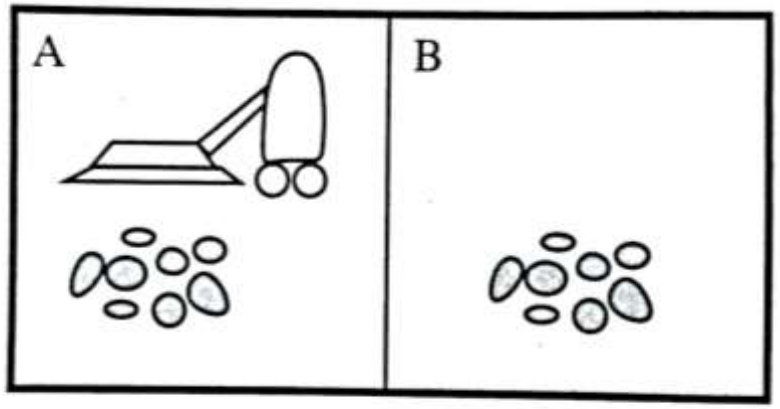



**Fig. A simplified road map of part of Romania**

# Solution

- A ***solution*** to a problem is a path from the initial state to a goal state.
- Solution quality is measured by the ***path cost*** function.
- An ***optimal solution*** has the lowest path cost among all solutions.




A vacuum-cleaner world with just two locations.

| Percept Sequence | Action |
|---|---|
| [A, Dirty] | Suction |
| [A, Clean] | Right |
| [B, Dirty] | Suction |
| [B, Clean] | Left |

Fig. Tabulation Representation

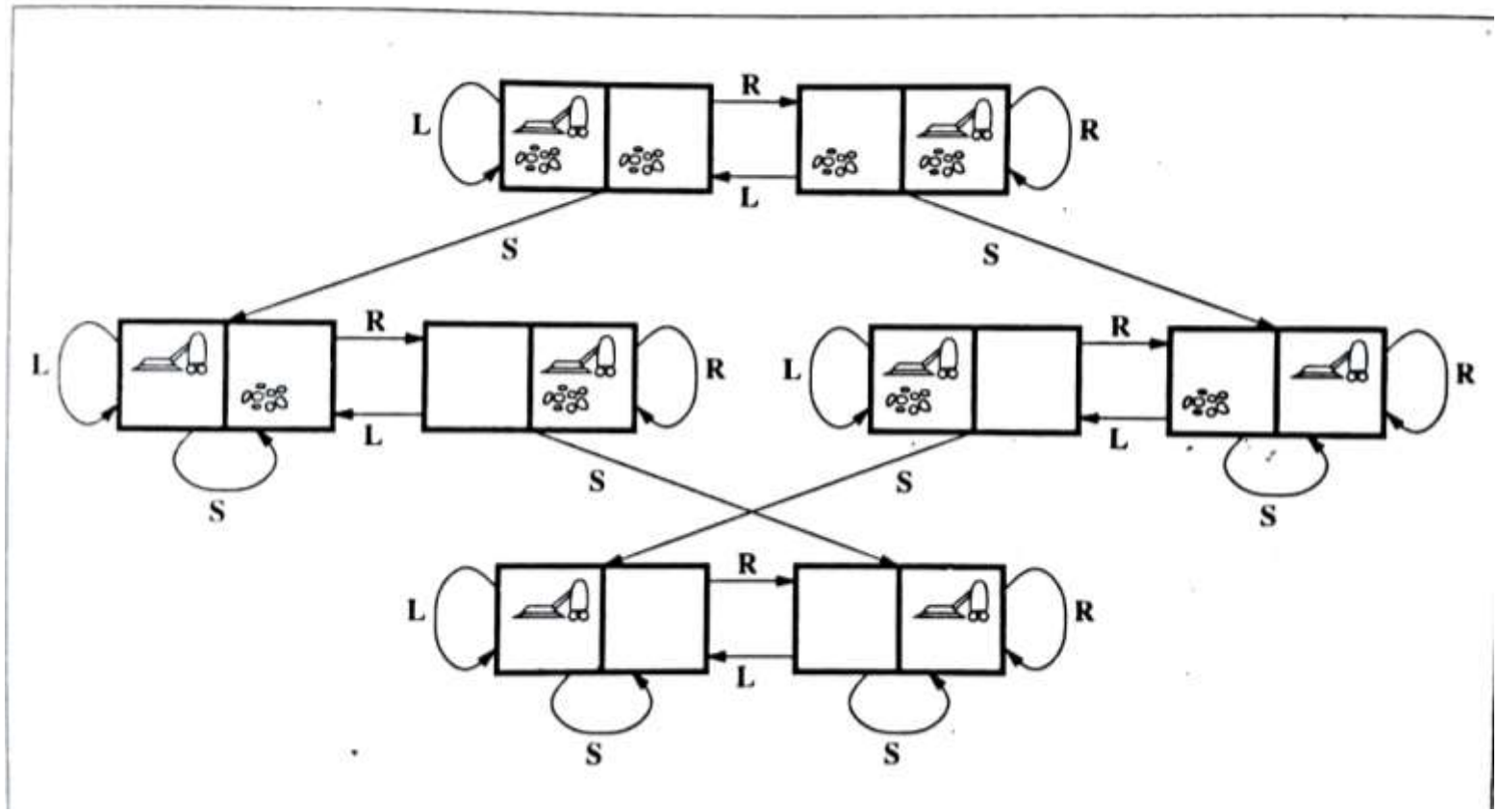



Figure: The state space for the vacuum world. Arcs denote actions:
L = Left, R = Right, S = Suction

# Problem Formulation

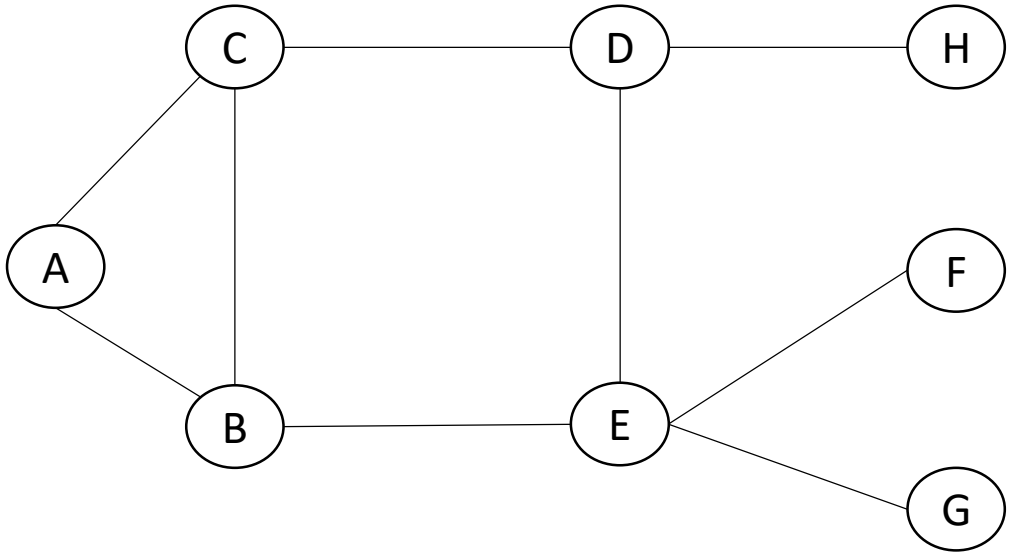
- The vacuum world can be formulated as a problem as follows:
- States
  - The agent is in one of two locations, each of which might or might not contain dirt.
- Initial state
  - Any state can be designated as the initial state
- Successor functions
  - This generates the legal states that result from trying the three actions (Left, Right, Suction)
- Goat test
  - This checks whether all the squares are clean
- Path cost
  - Each step costs 1, so the path cost is the number of steps in the path

# Searching For Solutions

- Search strategies refer to systematic methods and approaches used to explore and find relevant information or solutions within a given search space or dataset. Parameters for ***Evaluating*** Search Technique Performance:
- Completeness:
  - Determines if the search technique guarantees finding a solution if one exists within the search space.
- Time and Space Complexity:
  - Evaluates the efficiency of the search technique in terms of the time and space required to find a solution.
- Optimality:
  - Determines whether the search technique can find the best or optimal solution among all possible solutions.

# Uninformed Search (Brute Force Search / Blind Search / Exhaustive Search)

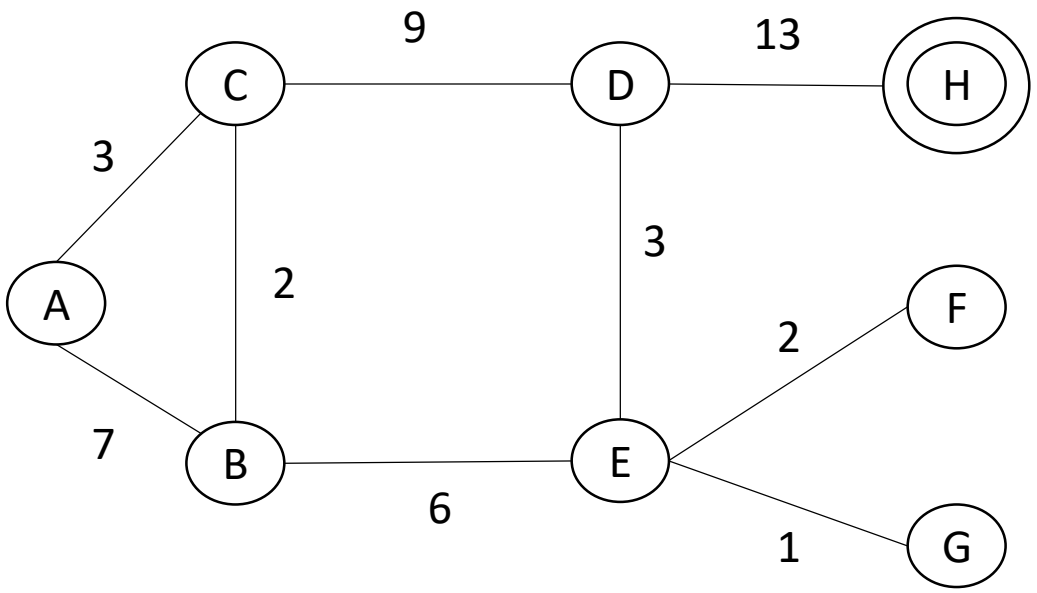
- Uninformed search strategies explore the search space without any specific information or heuristics about the problem.
- Here we proceed in a systematic way by exploring nodes in some predetermined order or simply by selecting nodes at random.
- Advantage:
  - **Simplicity:** Uninformed search strategies are generally easy to implement and understand.
- Disadvantage:
  - **Inefficiency:** Without additional information, uninformed search strategies may require an extensive search, leading to inefficiency in terms of time and space.
- Examples:
  - Breadth First Search
  - Depth First Search
  - Uniform Cost Search




**Apply BFS and DFS of the above Graph**

# Uniform-cost Search (UCS)

- BFS is optimal when all step costs are equal.
- We can find an algorithm that is optimal with any step cost function.
- UCS expands the node n with the *lowest path cost*.
- Features:
  - Reaches to Goal node
  - Backtracking is possible
  - Uses priority queue
  - Node expansion is based on path cost
- Advantages
  - Mostly give optimal solutions
- Disadvantages
  - Can get stuck itself in infinite loop



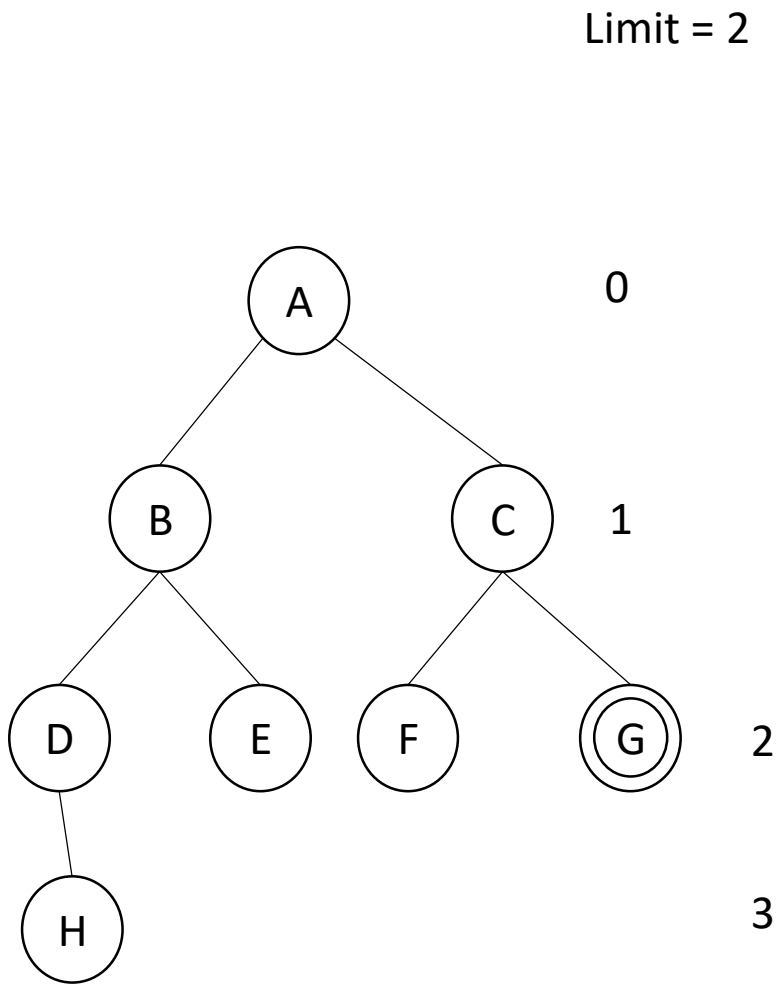

**A -> C -> B -> E -> G -> E -> F ->E -> D -> H**

**3 2 6 1 1 2 2 3 13**

**Path cost: 33**

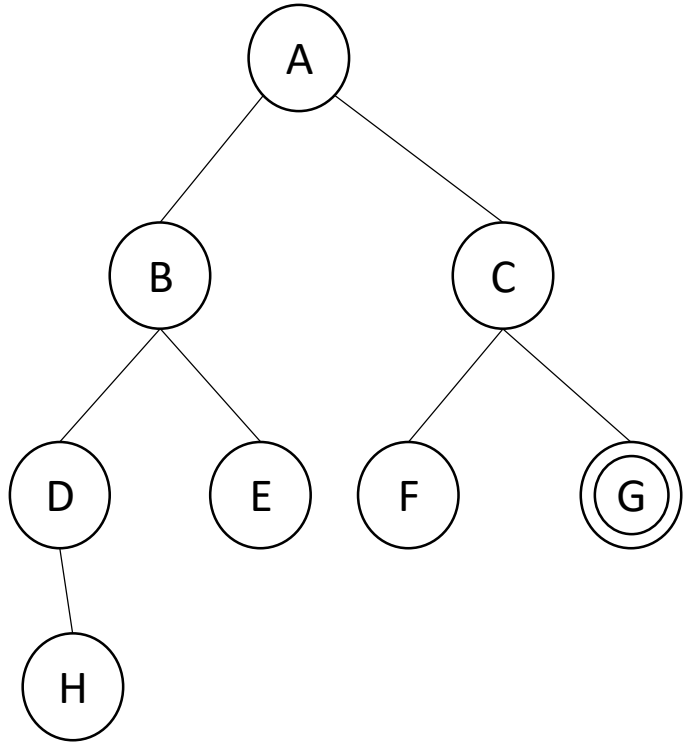
# Depth-limited Search (DLS)

- What is DFS?
- What is DLS?
- DLS terminates if
  - The solution is found
  - If there is no solution within given depth limit
- Advantage
  - Solve problem of infinitely deep path with no solution of DFS
  - Memory efficient
- Disadvantage
  - It can be incomplete if solution is below depth limit
  - Not necessary it will always give you best solution



# Iterative Deepening Depth-first Search

- Iterative deepening search is a general strategy, often used in combination with depth-first search that finds the best depth limit.
- Advantages
  - It inherits advantages of both DFS and DLS
- Disadvantages
  - Keeps repeating process of previous phase



# Bidirectional Search

- To run two simultaneous searches – one forward from the initial state and the other backward from the goal, stopping when the two searches meet in the middle.
- Pros
  - Much efficient and fast
  - Reduces time requirements
- Cons
  - Implementation is difficult
  - Goal state must be known in advance

- **Bidirectional search** typically follows the **Breadth-First Search (BFS)** strategy on both ends, starting from the initial state and the goal state simultaneously.

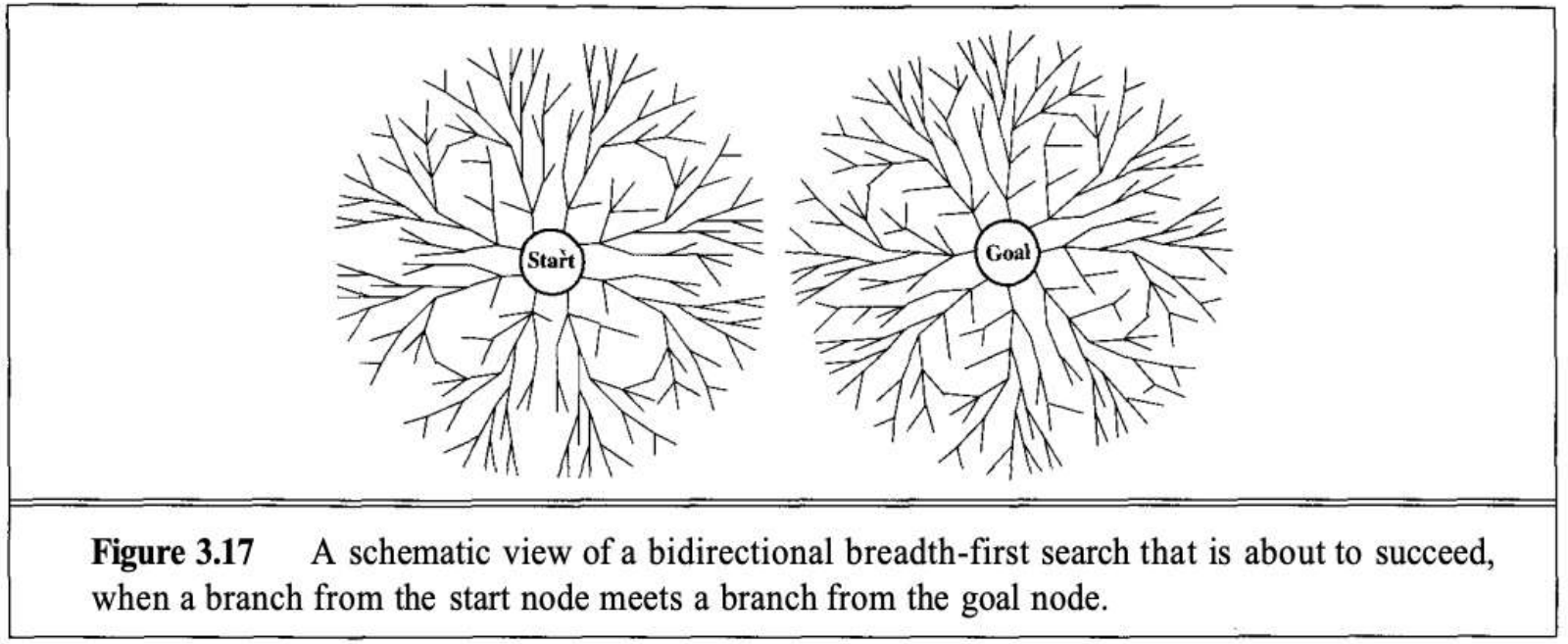



**Figure 3.17** A schematic view of a bidirectional breadth-first search that is about to succeed, when a branch from the start node meets a branch from the goal node.

# Comparing Uninformed search strategies

**Step Cost (ε):** The minimum cost of moving from one node to a neighboring node.

**Maximum Path Cost (C*):** The cost of the optimal solution (i.e., the cost of the least-cost path from the start node to the goal node).

b = branching factor

d = solution depth

l = depth limit

*a* complete if *b* is finite

*b* complete if step *cost* >= ε

for positive ε

- m = max tree depth
- C* = cost of optimal
- ε = min action cost
- c optimal if step cost are identical
- d if both directions use BFS

| Criterion | Breadth-First | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening | Bidirectional (if applicable) |
|---|---|---|---|---|---|---|
| Complete? | Yes[a] | Yes[a,b] | No | No | Yes[a] | Yes[a,d] |
| Time | $O(b^d)$ | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(b^m)$ | $O(b^\ell)$ | $O(b^d)$ | $O(b^{d/2})$ |
| Space | $O(b^d)$ | $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ | $O(bm)$ | $O(b\ell)$ | $O(bd)$ | $O(b^{d/2})$ |
| Optimal? | Yes[c] | Yes | No | No | Yes[c] | Yes[c,d] |

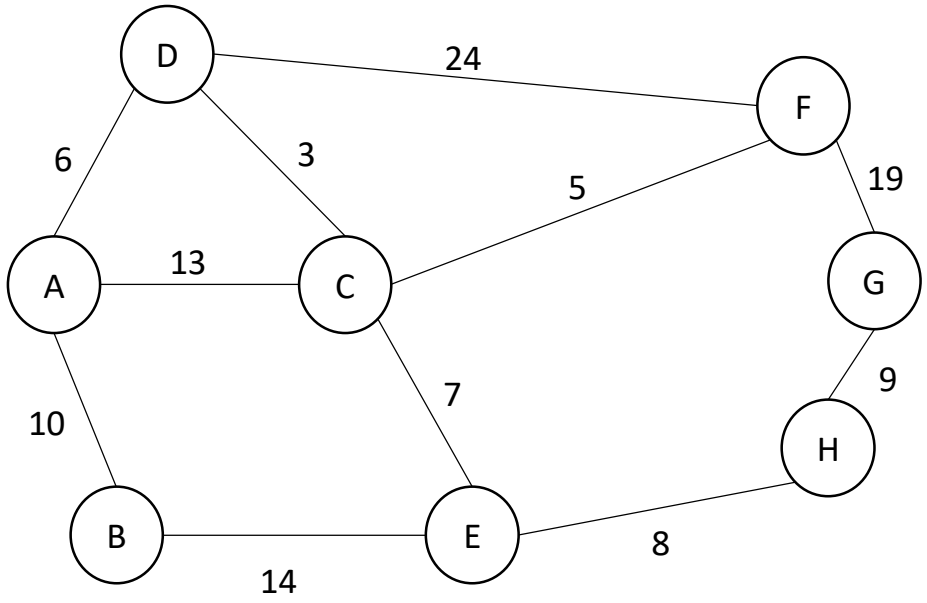
# Informed Search (Heuristic Search)

- Informed search strategies utilize domain-specific information or heuristics to guide the search towards more promising paths.
- Here we have knowledge such as how far we are from the goal, path cost, how to reach to goal node etc.
- This knowledge helps agents to explore less to the search space and find more efficiently the goal node.
- Advantage:
  - Efficiency: By leveraging domain knowledge, informed search strategies can make informed decisions and focus the search on more relevant areas, leading to faster convergence to a solution.
- Disadvantage:
  - Heuristic Accuracy: The effectiveness of informed search strategies heavily relies on the quality and accuracy of the chosen heuristic function. An inaccurate or misleading heuristic can lead to suboptimal or incorrect solutions.
- Example:
  - Hill Climbing
  - Best First Search
  - A* Algorithm

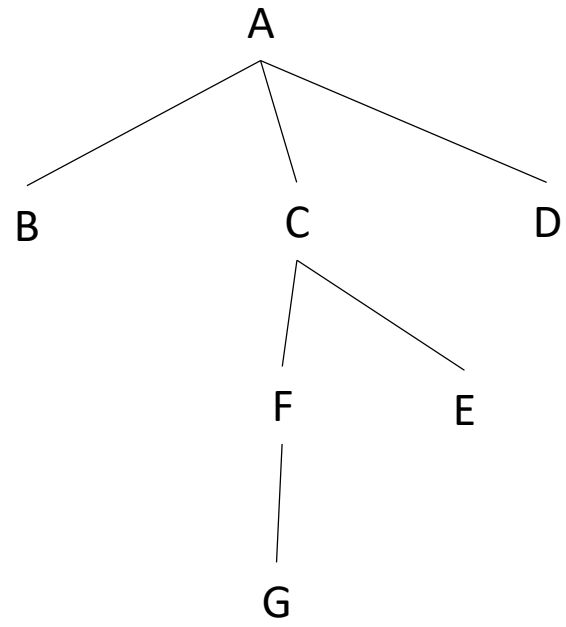
# Best-first Search

- Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.
- It's called "best-first" because it ***greedily*** chooses the node that seems most promising according to the heuristic.
- Best-first search takes the advantage of both BFS and DFS. The evaluation function is construed as a cost estimate, so the node with the lowest evaluation is expanded first.
- h(n) = estimated cost of the cheapest path from the state at node n to a goal state.

# Best-first Search




| Node | h(n) |
|---|---|
| A | 39 |
| B | 31 |
| C | 24 |
| D | 34 |
| E | 18 |
| F | 16 |
| H | 9 |
| G | 0 |



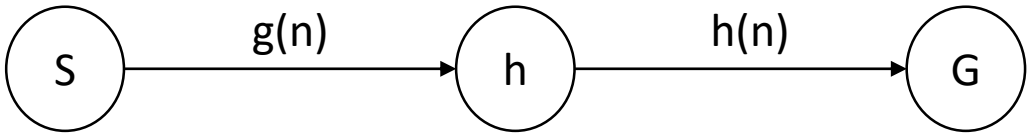
# Best-first Search

- Advantages of Best-First Search
  - It can find a solution without exploring much of the state space.
  - It uses less memory than other informed search methods like A* as it does not store all the generated nodes.
- Disadvantages of Best-First Search
  - It is not complete. In some cases, it may get stuck in an infinite loop.
  - It is not optimal. It does not guarantee the shortest possible path will be found.
  - It heavily depends on the accuracy of the heuristic.

# A* search: Minimizing the total estimated solution cost

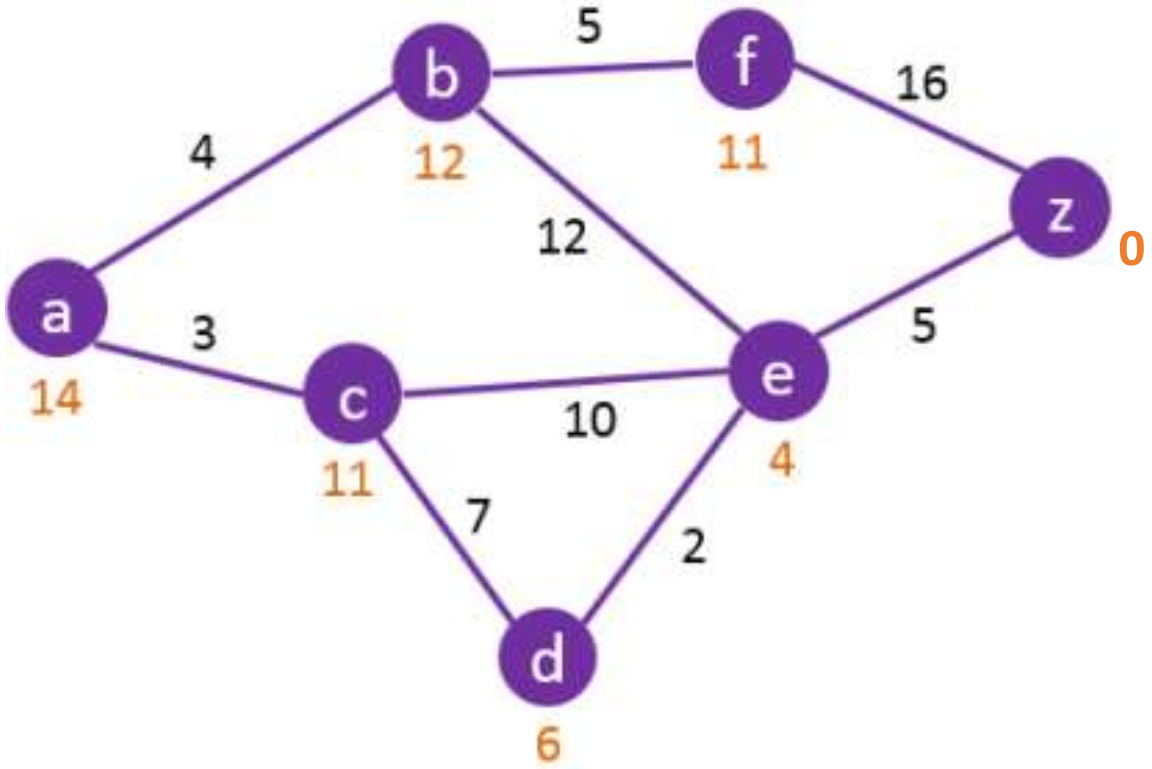
- A* Search is an informed search algorithm that combines the advantages of both Uniform Cost Search and Greedy Best-First Search.
- It evaluates a node based on a combination of the cost of the path from the start node to that node and an estimated heuristic function that estimates the cost to reach the goal form the current node.

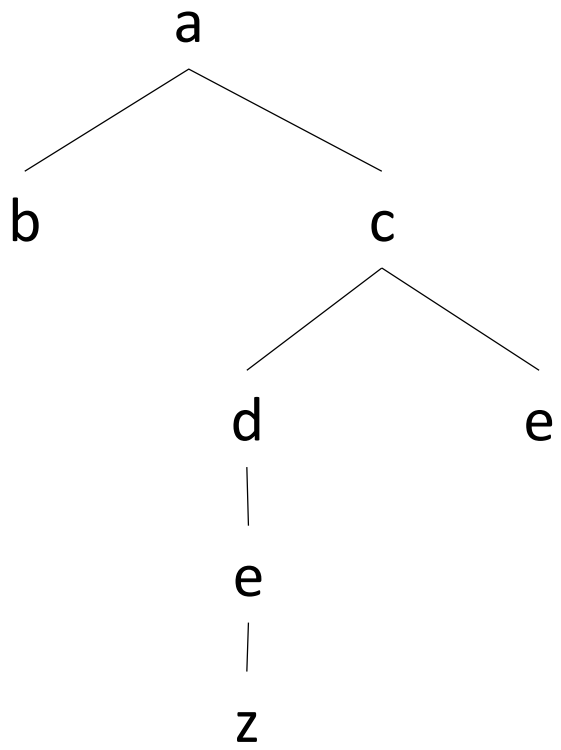
$$f(n) = g(n) + h(n)$$




- A* search evaluates nodes by combining g(n), the cost to reach the node, and h(n), the estimated cost to get from the node to the goal: $f(n) = g(n) + h(n)$

# A* search



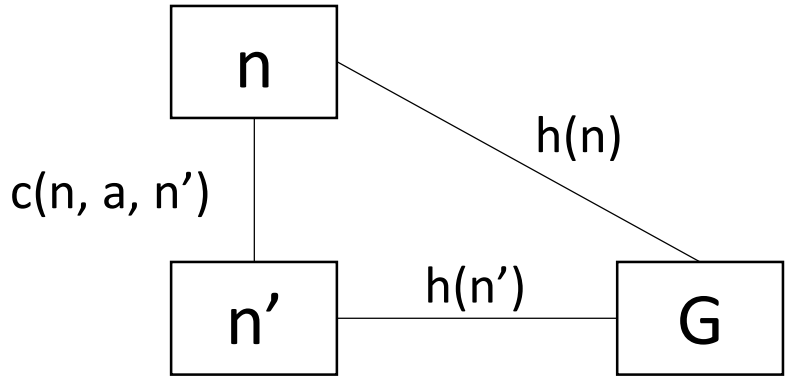



# Conditions for optimality: Admissibility and consistency

- The first condition we require for optimality is that h(n) be an admissible heuristic.
- An admissible heuristic is one that ***never overestimates*** the cost to reach the goal.
- Admissible heuristics are by nature ***optimistic*** because they think the cost of solving the problem is less than it actually is.
- For example, Straight-line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot be an overestimate.

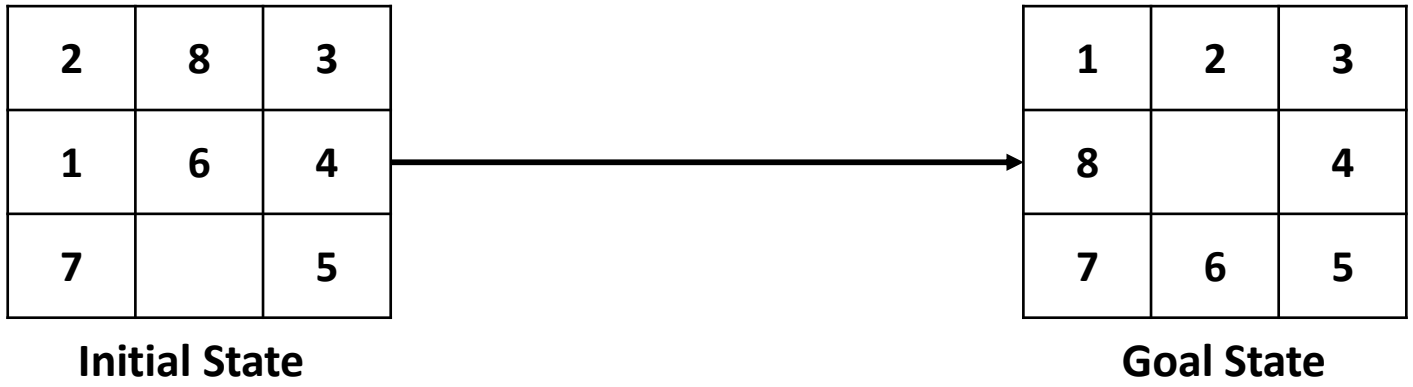
# Conditions for optimality: Admissibility and consistency

- A second, slightly stronger condition called ***consistency*** (or monotonicity) is required only for applications of A* to graph search.
- A heuristic h(n) is consistent if, for every node n and every successor n’ of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n’ plus the estimated cost of reaching the goal from n’:
- $h(n) \leq c(n, a, n') + h(n')$



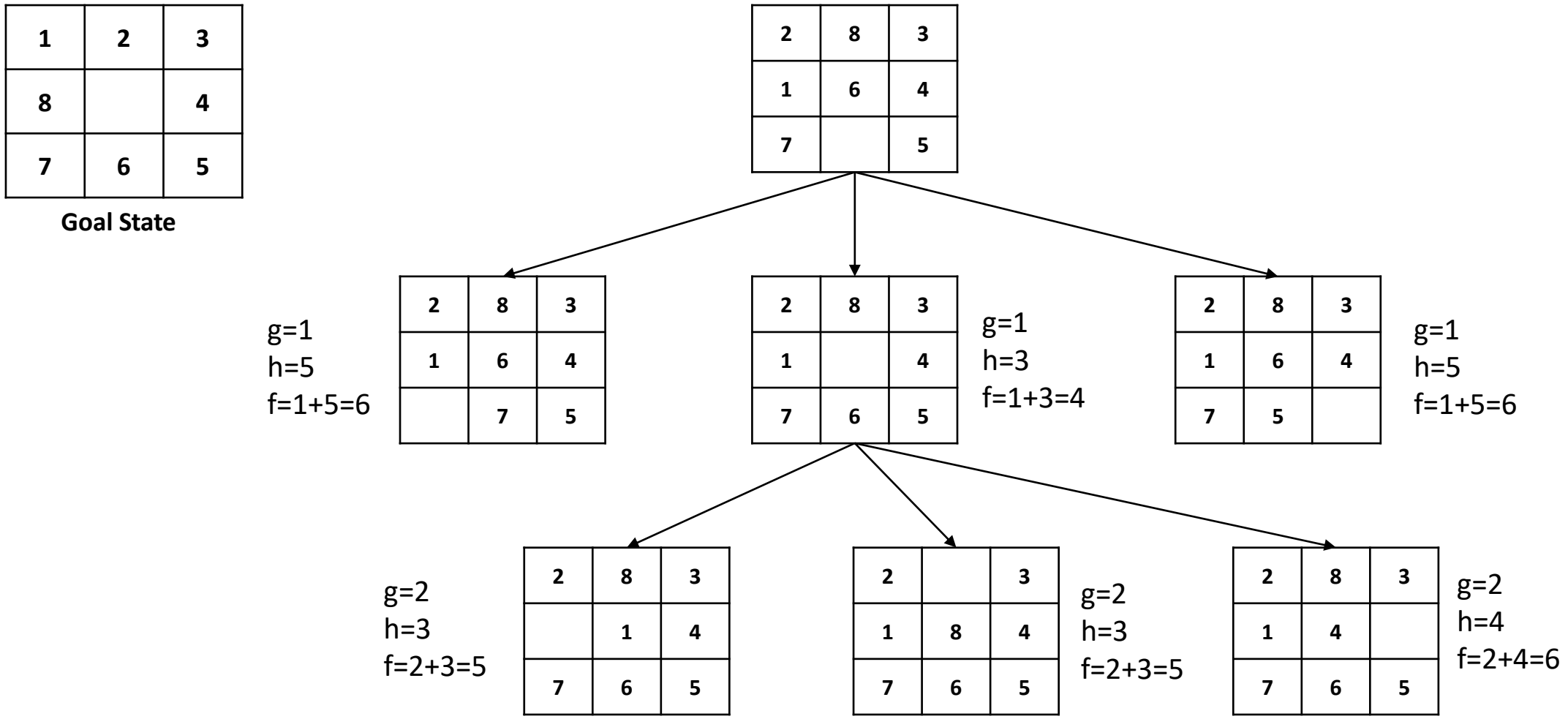
# The 8-puzzle Problem

- The 8-puzzle problem is a classic search problem in the field of AI and computer science.
- It involves a 3x3 grid with 8 numbered tiles and one empty space.
- The goal is to move the tiles around until they are in a specific goal state.
- You can move a tile into the empty space if it is adjacent to it (up, down, left, or right). Diagonal moves are not allowed.

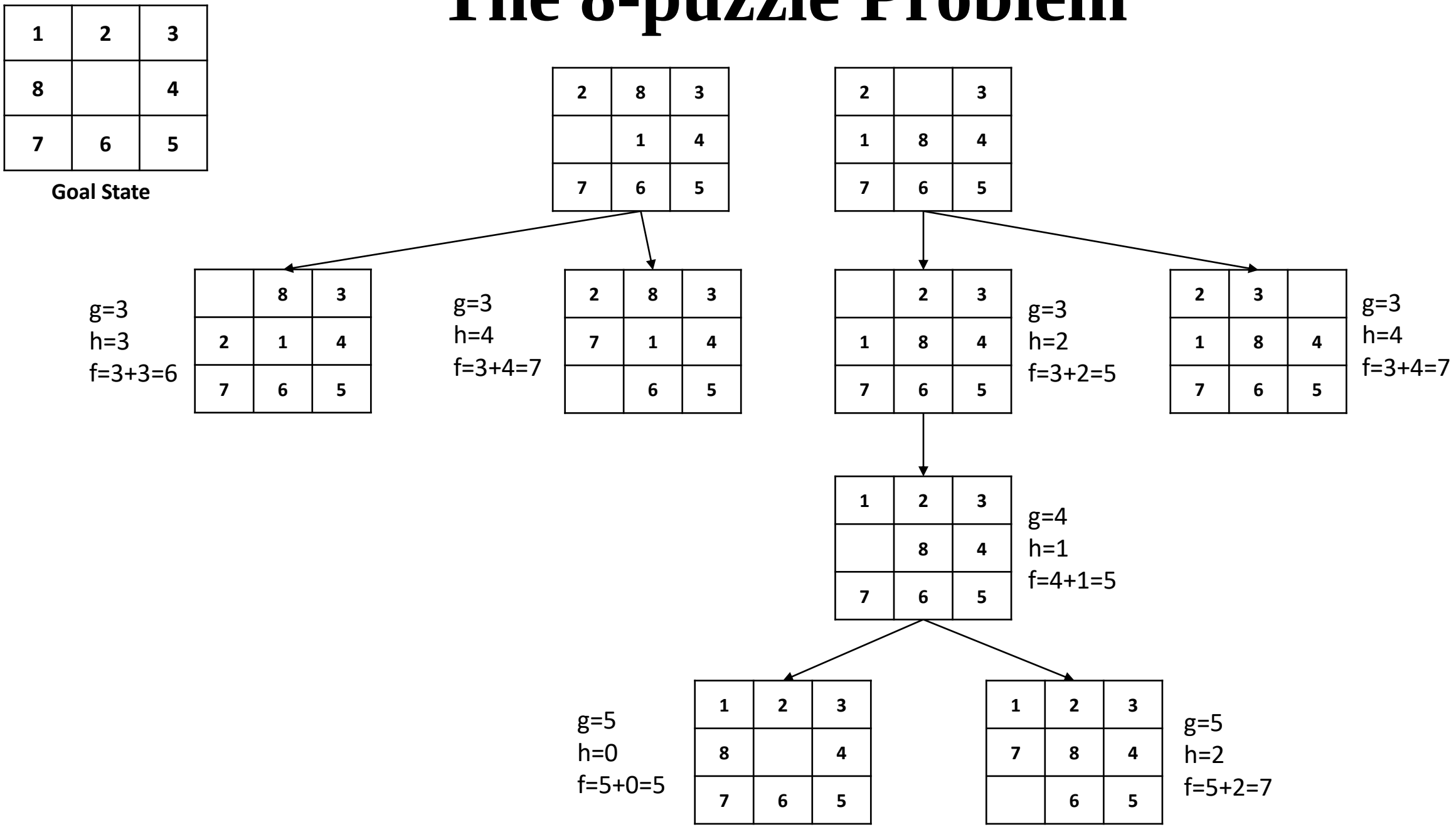



- Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
- f(n) = g(n) + h(n)
- Consider g(n) = Depth of node, and h(n) = Number of misplaced tiles.

# The 8-puzzle Problem

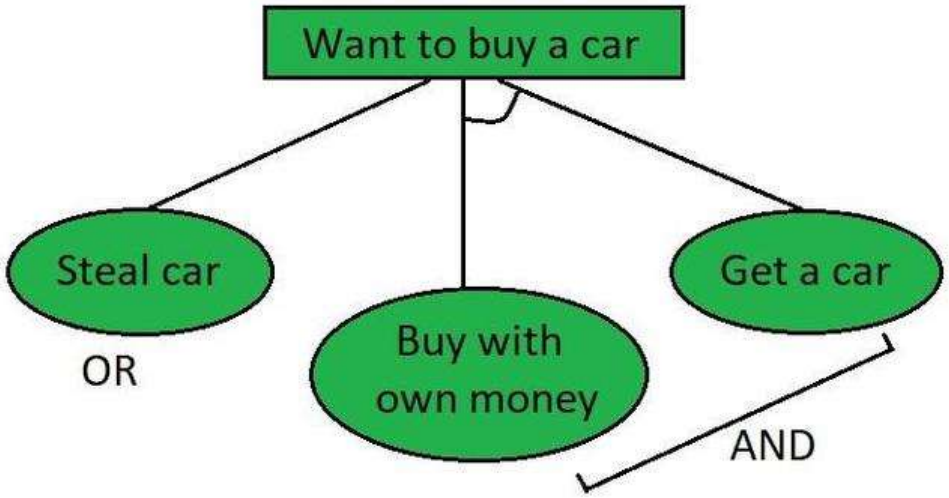


# The 8-puzzle Problem

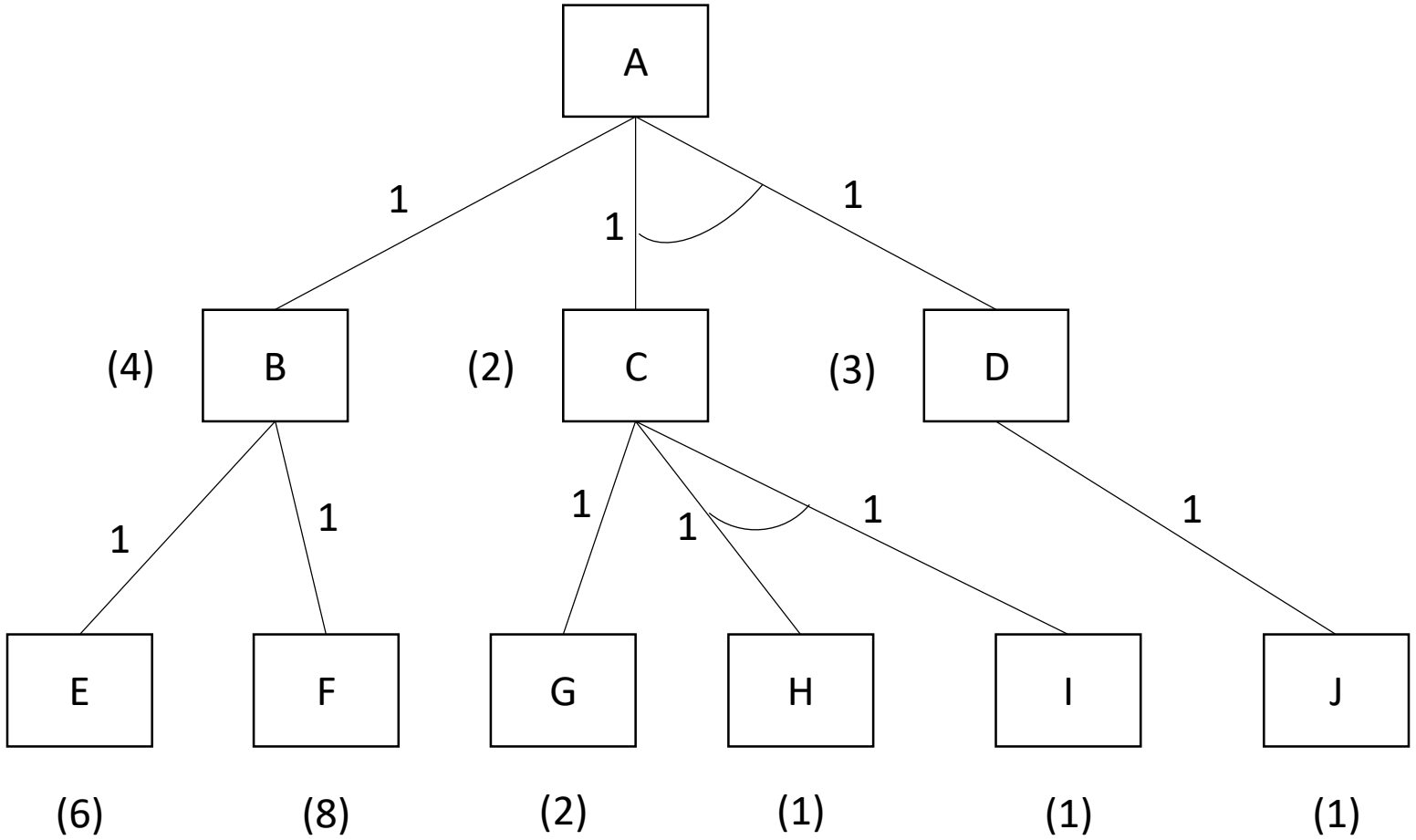


# Problem with A* Algorithm

- So far we have considered search strategies for OR graphs through which we want to find a single path to a goal.
- The AND-OR GRAPH (or tree) is useful for representing the solution of problems that can be solved by decomposing them into a set of *smaller problems*, all of which must then be solved.
- This *decomposition, or reduction*, generates arcs that we call AND arcs.
- One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution.



# AO* Algorithm



# Local Search Algorithms

- Local search algorithms operate using a single current state and generally move only to neighbors of that state.
- Local search algorithms are not systematic.
- Advantages
  - Use very little memory
  - Often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable
- The aim is to find the best state according to an ***objective function***.

# Local Search Algorithms

- To understand local search, we will find it very useful to consider the ***state space land-scape***.
- A landscape has both “location” (defined by the state) and “elevation” (defined by the value of heuristic cost function or objective function).
- If elevation corresponds to cost, then the aim is to find the lowest valley – a ***global minimum***.
- If elevation corresponds to an objective function, then the aim is to find the highest peak – a ***global maximum***.

# Hill Climbing Algorithm

- It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by making a change, If the change produces a better solution, another incremental change is made to the new solution, and so on until no further improvements can be found.
- Hill climbing is sometimes called greedy local search with no backtracking because it grabs a good neighbor state without thinking ahead about where to go next.

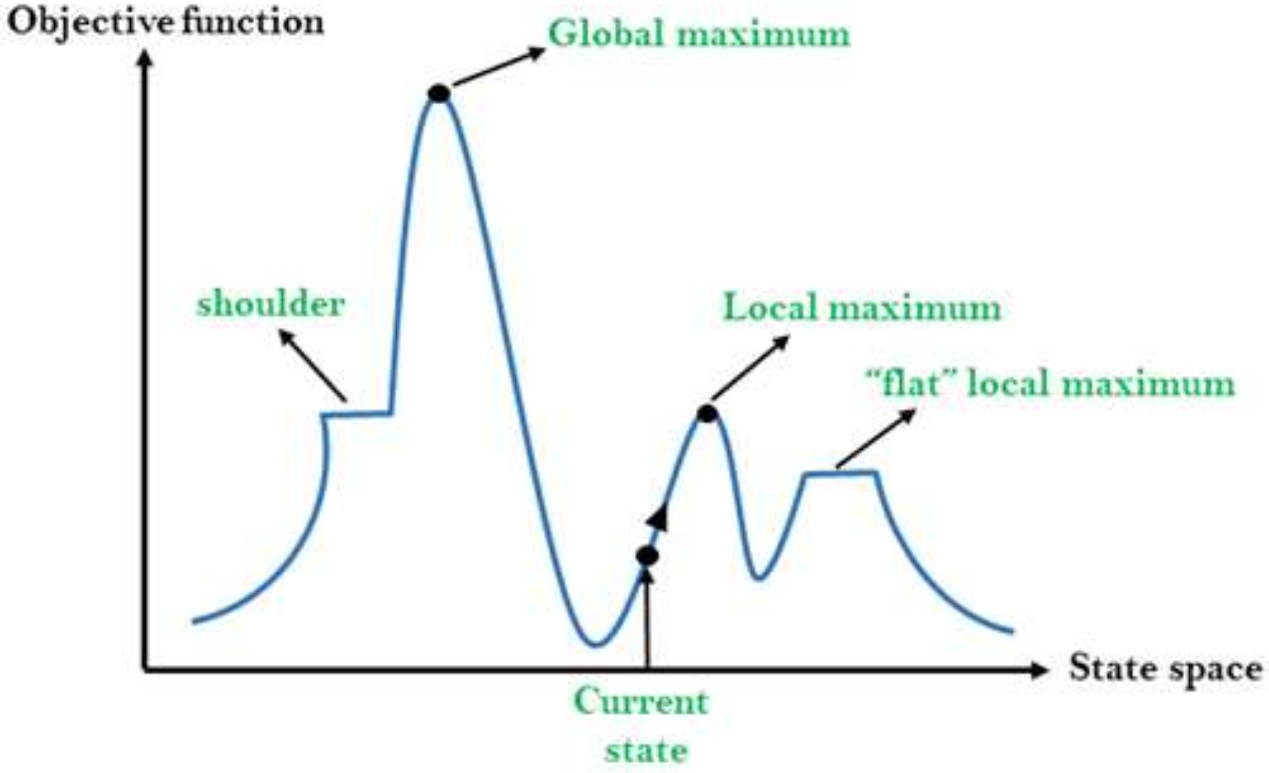
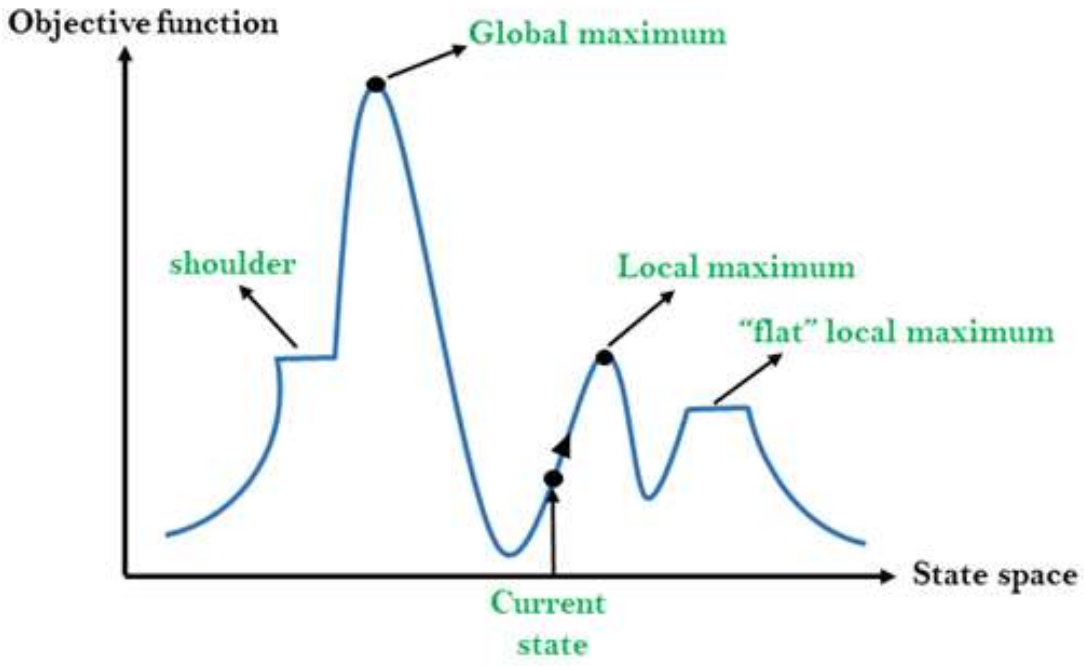



***Figure.*** *A one-dimensional state space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow.*

# Problems with Hill Climbing

- **Local maxima:** A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.
- **Ridges:** Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
- **Plateau:** A plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. A hill-climbing search might get lost on the plateau. In each case, the algorithm reaches a point at which no progress is being made.




# Random-restart hill climbing

- To avoid local optima, you can perform multiple runs of the algorithm from different random initial states. This is called random-restart hill climbing and increases the chance of finding a global optimum.
- **Tabu Search:** To prevent the algorithm from getting stuck in a loop or revisiting states, a list of previously visited states can be maintained, which are then avoided in future steps.
- **Local Beam Search:** To tackle plateau and local optima, this variation of hill climbing keeps track of *k* states rather than just one. It begins with *k* randomly generated states. At each step, all the successors of all k states are generated, and if any one is a goal, it halts. Else, it selects the best *k* successors from the complete list and repeats.

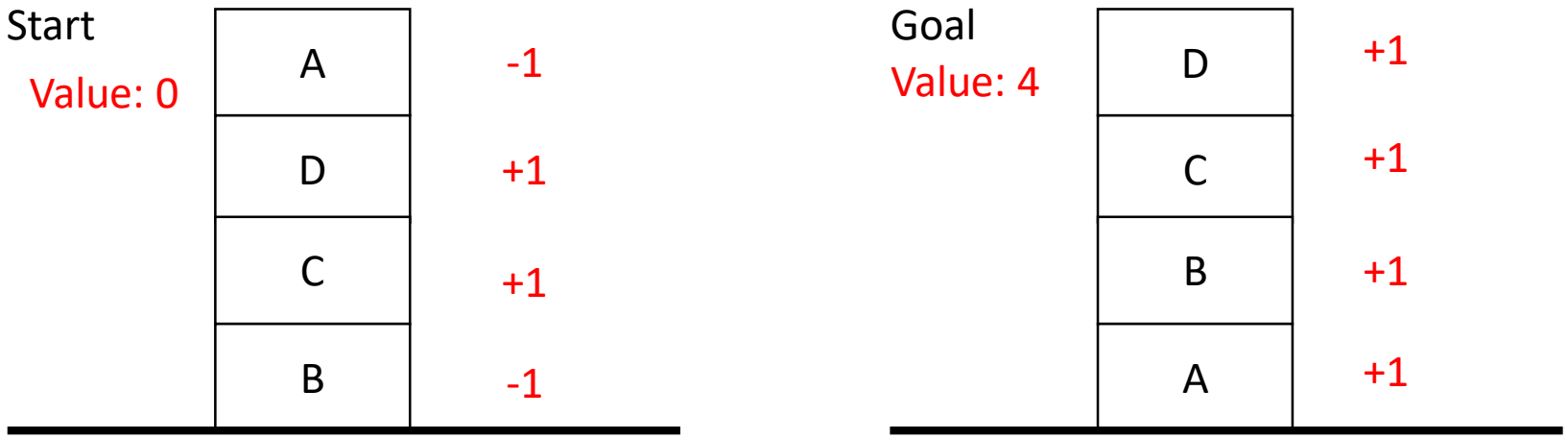
# Steepest-Ascent Hill Climbing

- A useful variation on simple hill climbing considers all the moves from the current state and selects the best one as the next state.

# Simulated Annealing Search

- A hill-climbing algorithm that never makes “downhill” moves is guaranteed to be incomplete, because of getting stuck into local maximum.
- A purely random walk – is complete, but extremely inefficient.
- Simulated annealing combines hill climbing with a random walk in some way that yields both efficiency and completeness.
  - Annealing schedule is maintained
  - Moves to worst states may be accepted
  - Best state found so far is also maintained
- Annealing is a process in metallurgy where metals are slowly cooled to make them reach a state of low energy where they are very strong.

# Hill Climbing: Local Heuristic Function

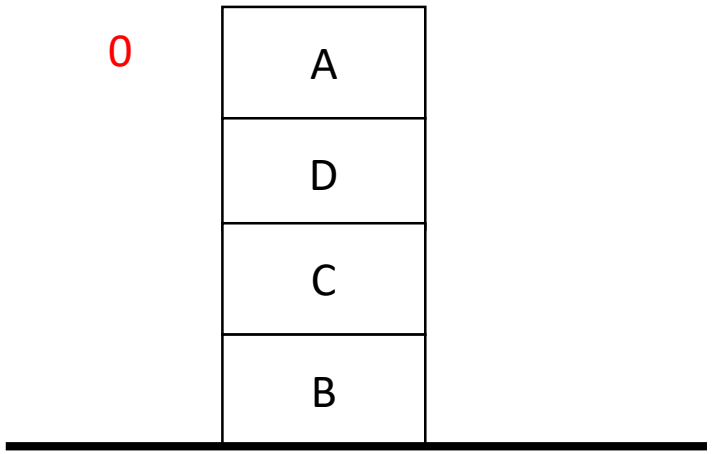


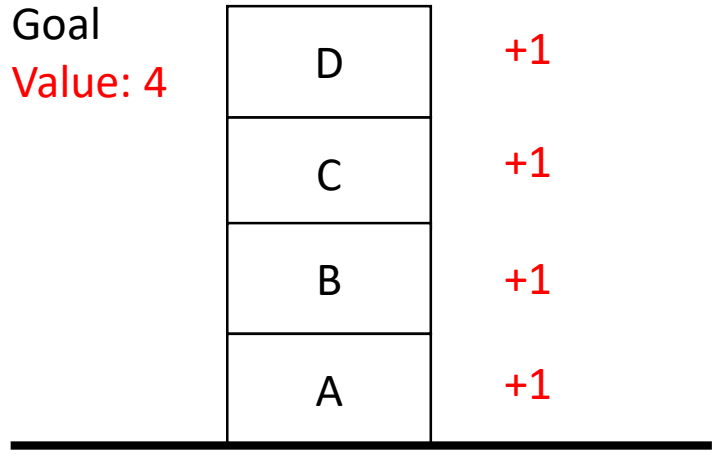

**Blocks World**

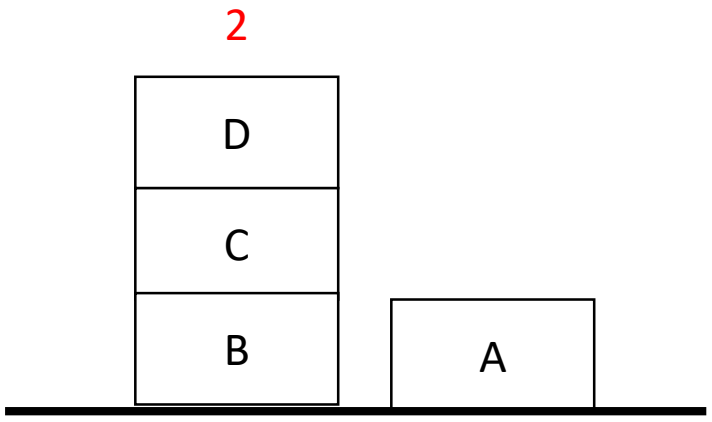
**Local heuristic:**

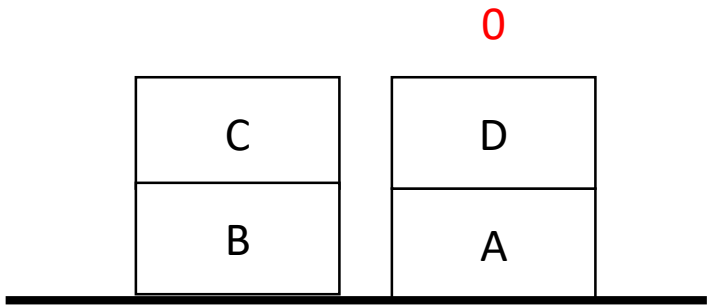
- Consider immediate consequences
- +1 for each block that is resting on the thing it is supposed to be resting on.
- -1 for each block that is resting on a wrong thing.

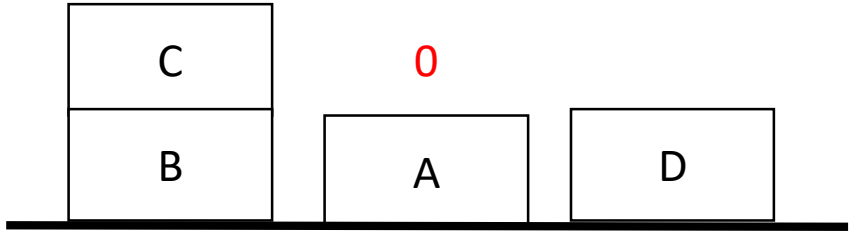
# Hill Climbing: Local Heuristic Function



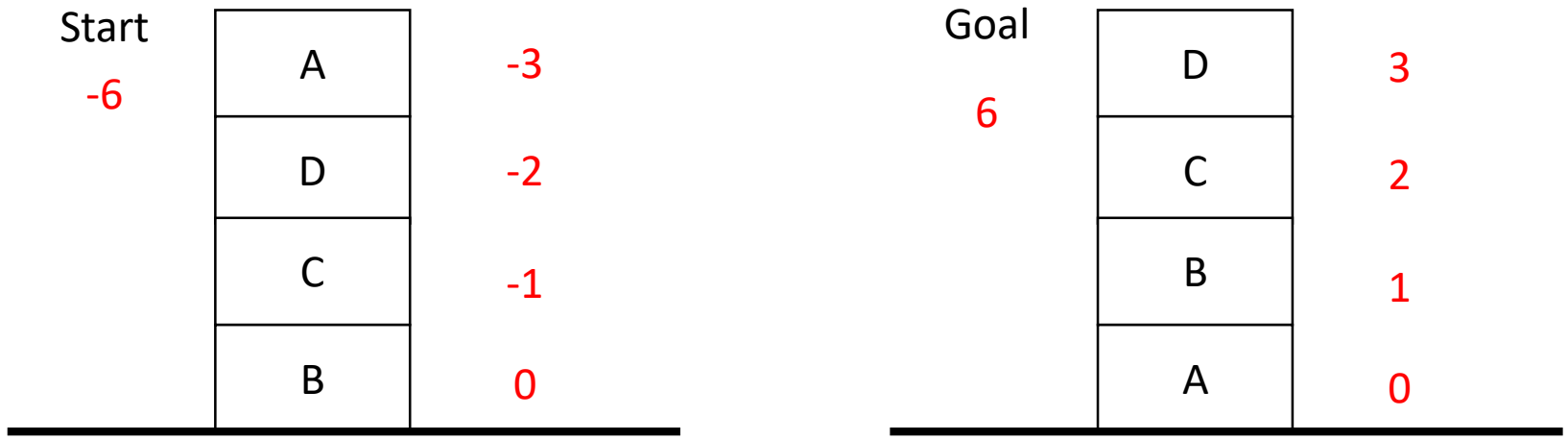



# Hill Climbing: Global Heuristic Function

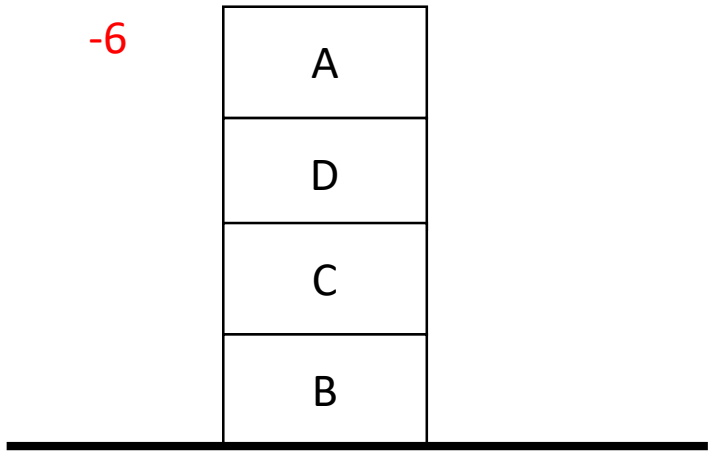


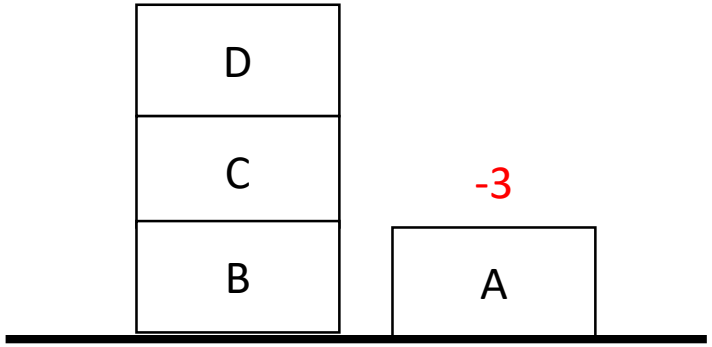

**Blocks World**

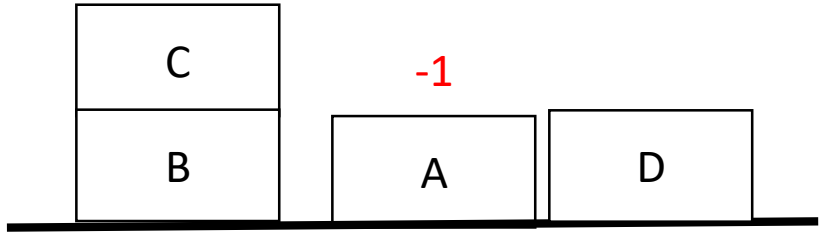
**Global heuristic:**

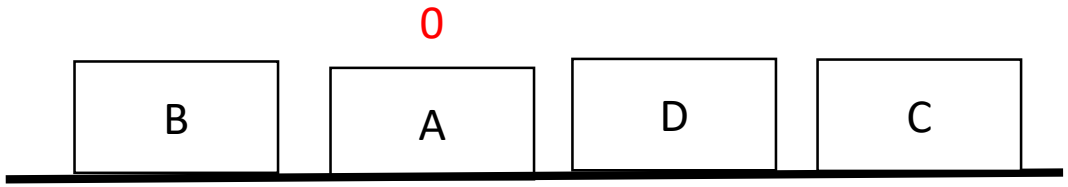
- Consider global information or what will happen in future
- For each block that has the correct support structure: +1 to every block in the support structure.
- For each block that has the wrong support structure: -1 to every block in the support structure.

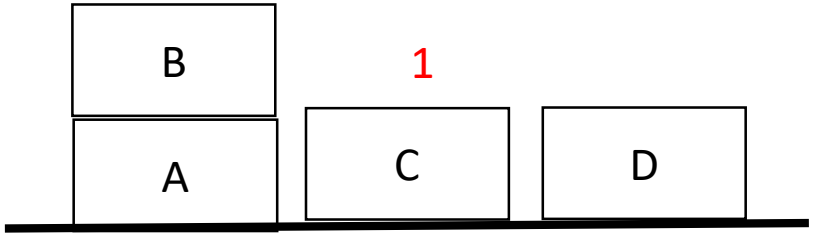
# Hill Climbing: Global Heuristic Function

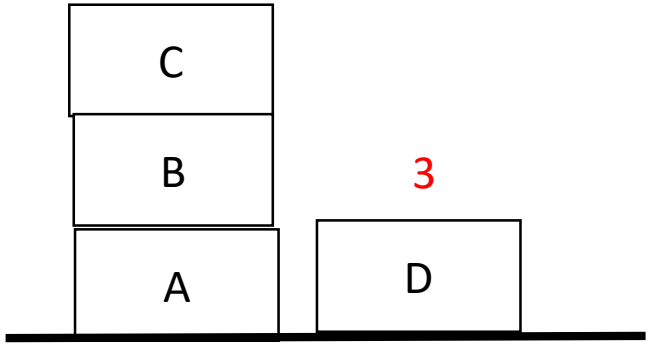


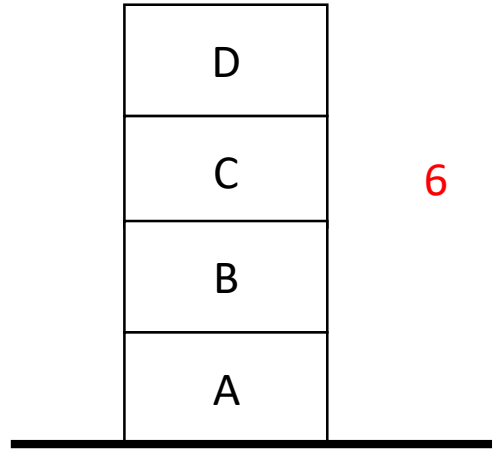



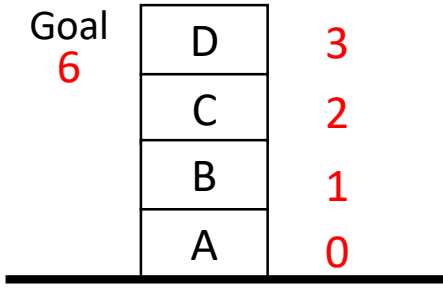



# Hill Climbing: 8-Queens Problem

- Local-search algorithms typically use a complete-state formulation, where each state has 8 queens on the board, one per column.
- The successor function returns all possible states generated by moving a single queen to another square in the same column (so each state has 8 x 7 = 56 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: 8-Queens Problem

- *Fig a* shows a state with $h = 17$.
- The figure also shows the values of all its successors, with the best successors having $h = 12$.
- Hill-climbing algorithms typically choose randomly among the set of best successors, if there is more than one.
- Hill climbing often makes very rapid progress towards a solution, because it is usually quite easy to improve a bad state.
- For example, from the state in *Fig a*, it takes just five steps to reach the state in *Fig b*, which has $h = 1$ and is very nearly a solution.


Fig a

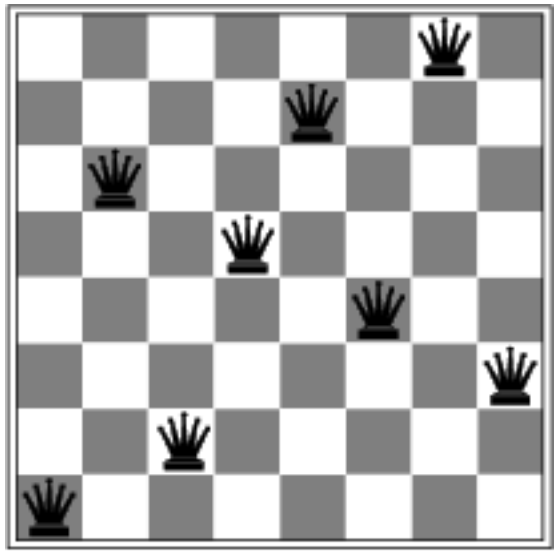


Fig b

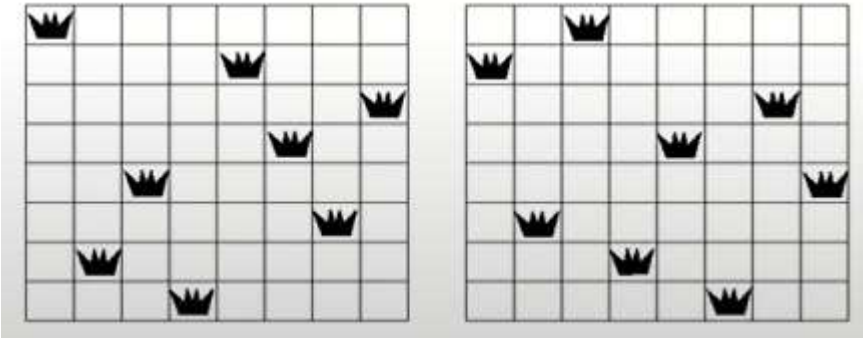
# Genetic Algorithm

- Biology Concepts
  - Population
  - Fitness
  - Selection
  - Crossover
  - Mutation

# Population

- Biology
  - Collection of individuals

- Algorithm
  - Collection of states



# Fitness

- Biology
  - More healthy, less prone to diseases

- Algorithm
  - Closest to the final solution

# Selection

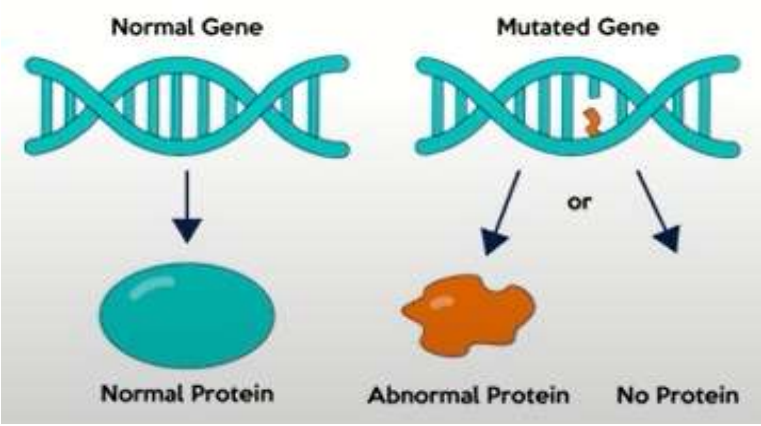
- Biology
  - Selecting species that are the most biologically fit
- Algorithm
  - Selecting states that are closest to the solution (Fittest)

# Crossover

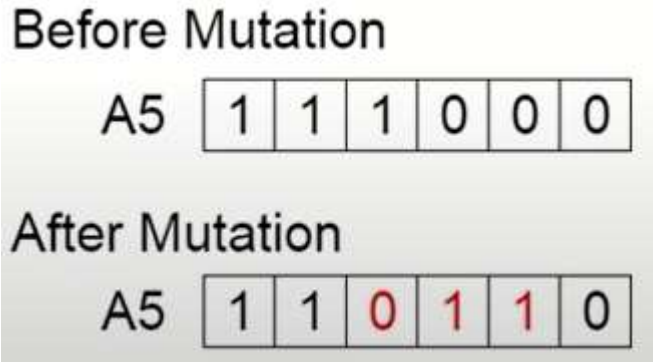
- Biology
  - Mating or Reproducing
- Algorithm
  - Interchanging values between selected states

# Mutation

- Biology
  - Change or variation

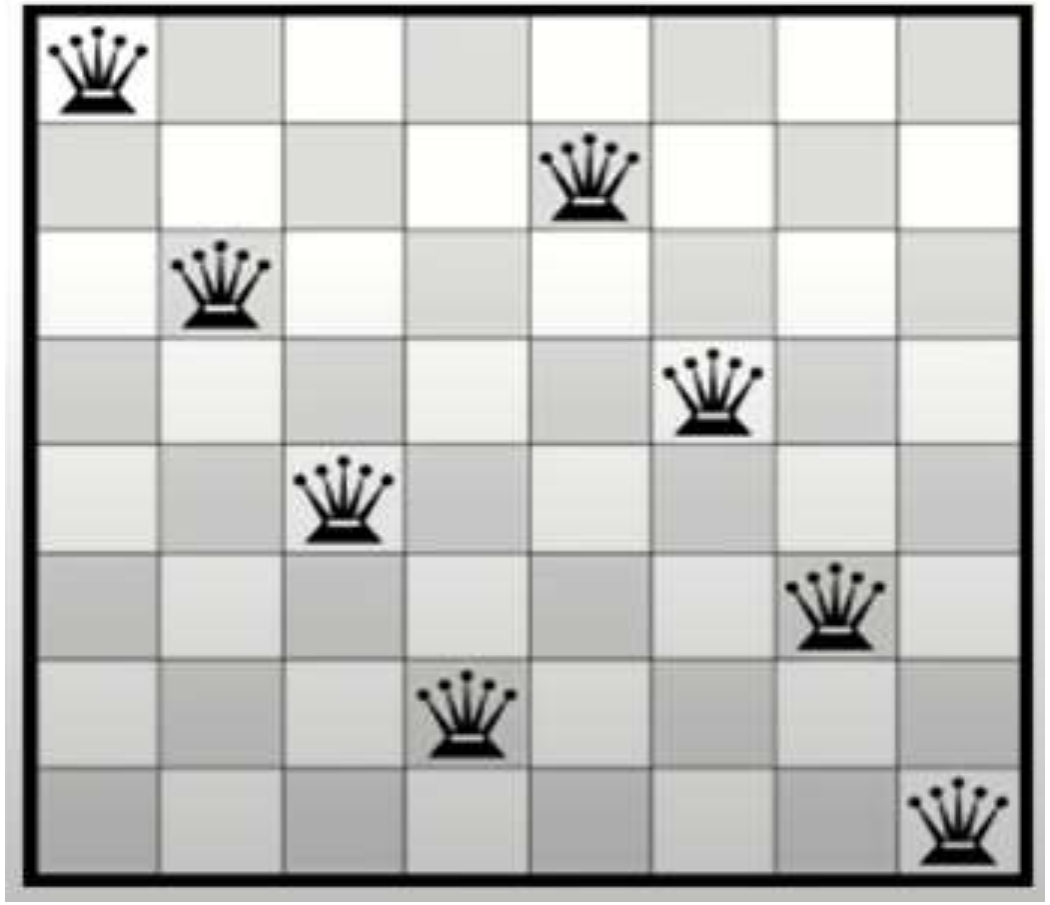



- Algorithm
  - Alteration



# 8 – Queens Problem

- Arrange 8 queens on a standard chess board is such a way that no queen attacks each other.



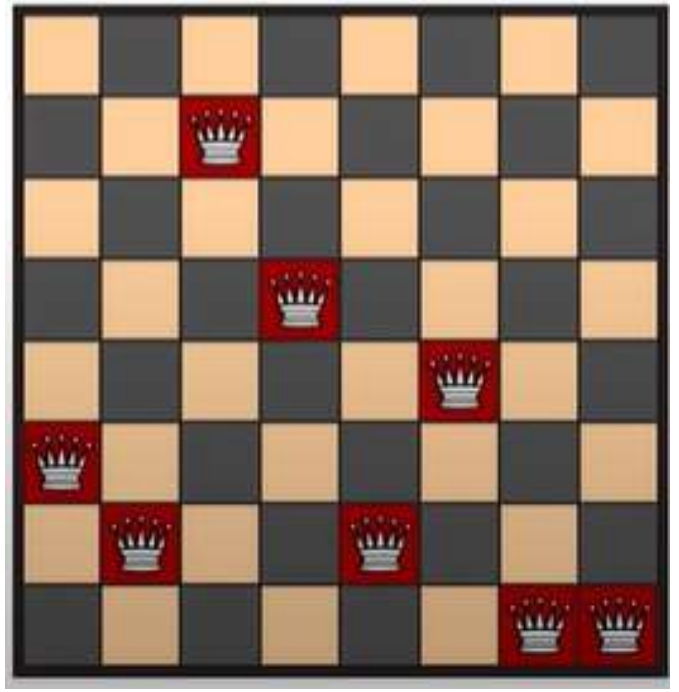
# Solving the 8 – Queen Problem using the Genetic Algorithm

- Step 1: Representing individuals
- Step 2: Generating an initial Population
- Step 3: Applying a Fitness Function
- Step 4: Selecting parents for mating in accordance to their fitness
- Step 5: Crossover of parents to produce new generation
- Step 6: Mutation of new generation to bring diversity
- Step 7: Repeat until solution is reached

# Step 1: Representing individuals

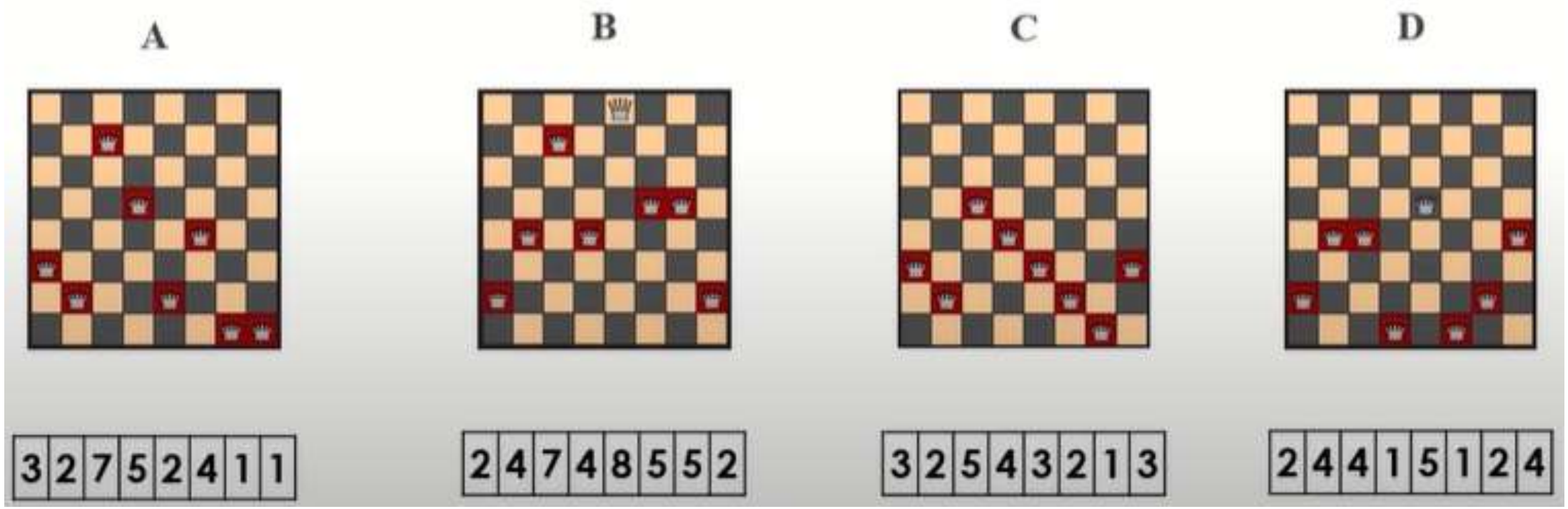
- Formulate an appropriate method to represent individuals of a population
- Array
- Index: Column
- Value : Row

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



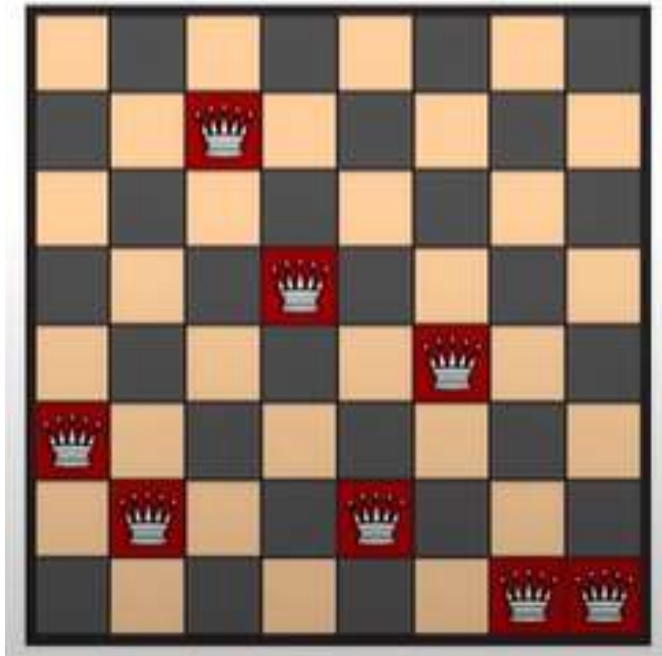
# Step 2: Generate initial Population

- Generate random arrangements of 8 queens on a standard chess board.



# Step 3: Apply Fitness Function

Individual

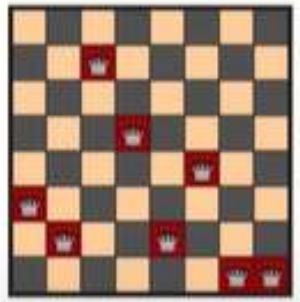
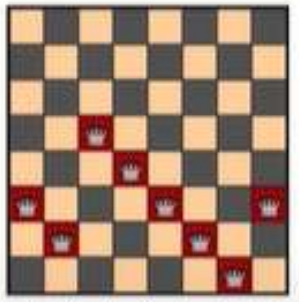
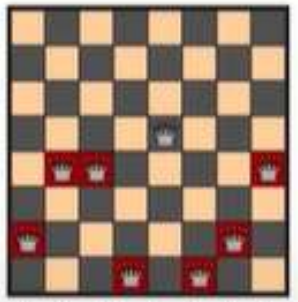


| 3 | 2 | 7 | 5 | 2 | 4 | 1 | 1 |
|---|---|---|---|---|---|---|---|

Fitness = No. of non attacking pairs

- **Queen 1:** 6
- **Queen 2:** 5
- **Queen 3:** 4
- **Queen 4:** 3
- **Queen 5:** 3
- **Queen 6:** 2
- **Queen 7:** 0
- **Queen 8:** 0
- **Total:** 23

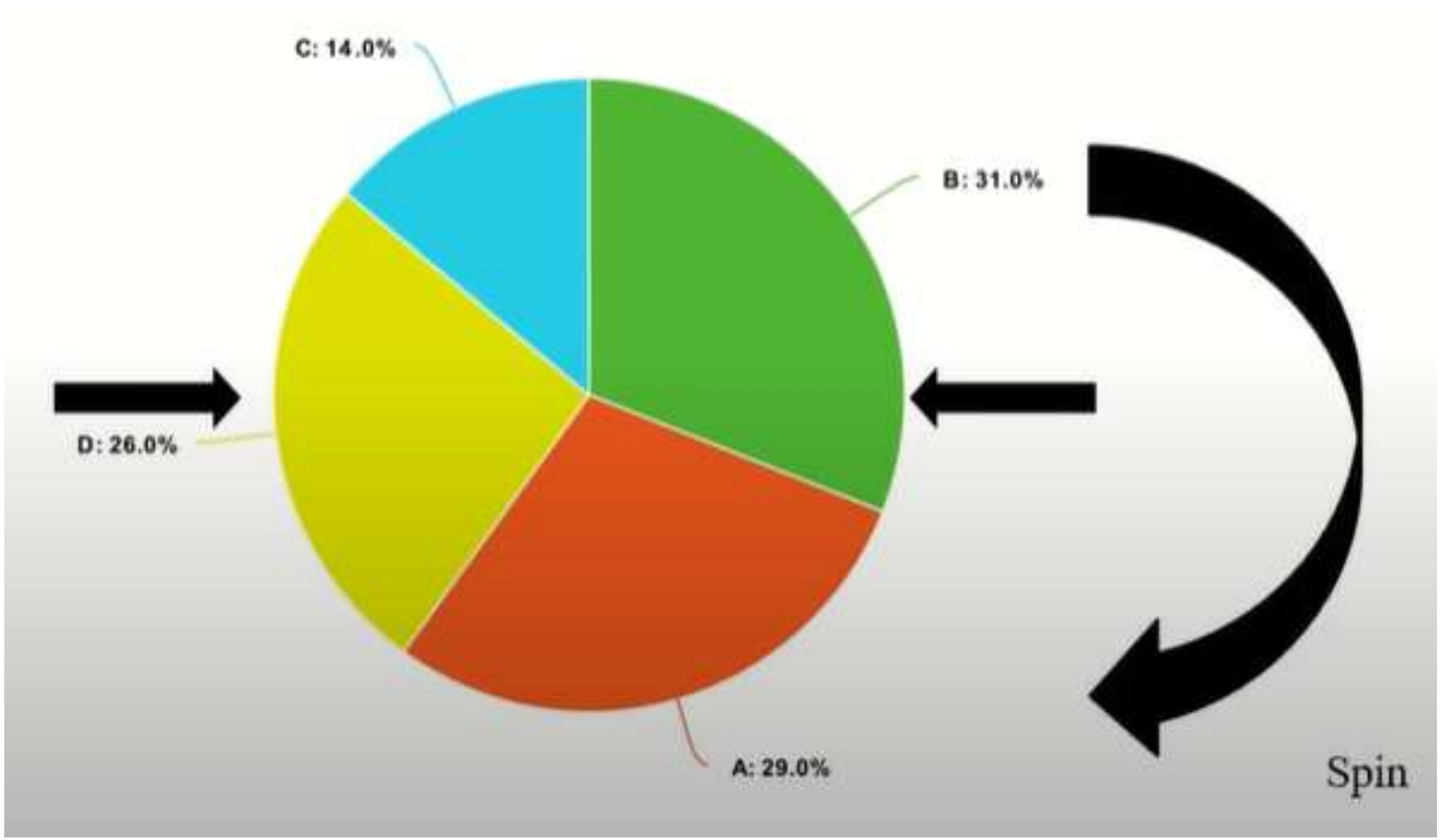
# Step 3: Apply Fitness Function

| | A | B | C | D |
|---|---|---|---|---|
| Individuals |  3 2 7 5 2 4 1 1 |  2 4 7 4 8 5 5 2 |  3 2 5 4 3 2 1 3 |  2 4 4 1 5 1 2 4 |
| Fitness | 23 | 24 | 11 | 20 |
| Fitness % | 29% | 31% | 14% | 26% |

# Step 4: Selection

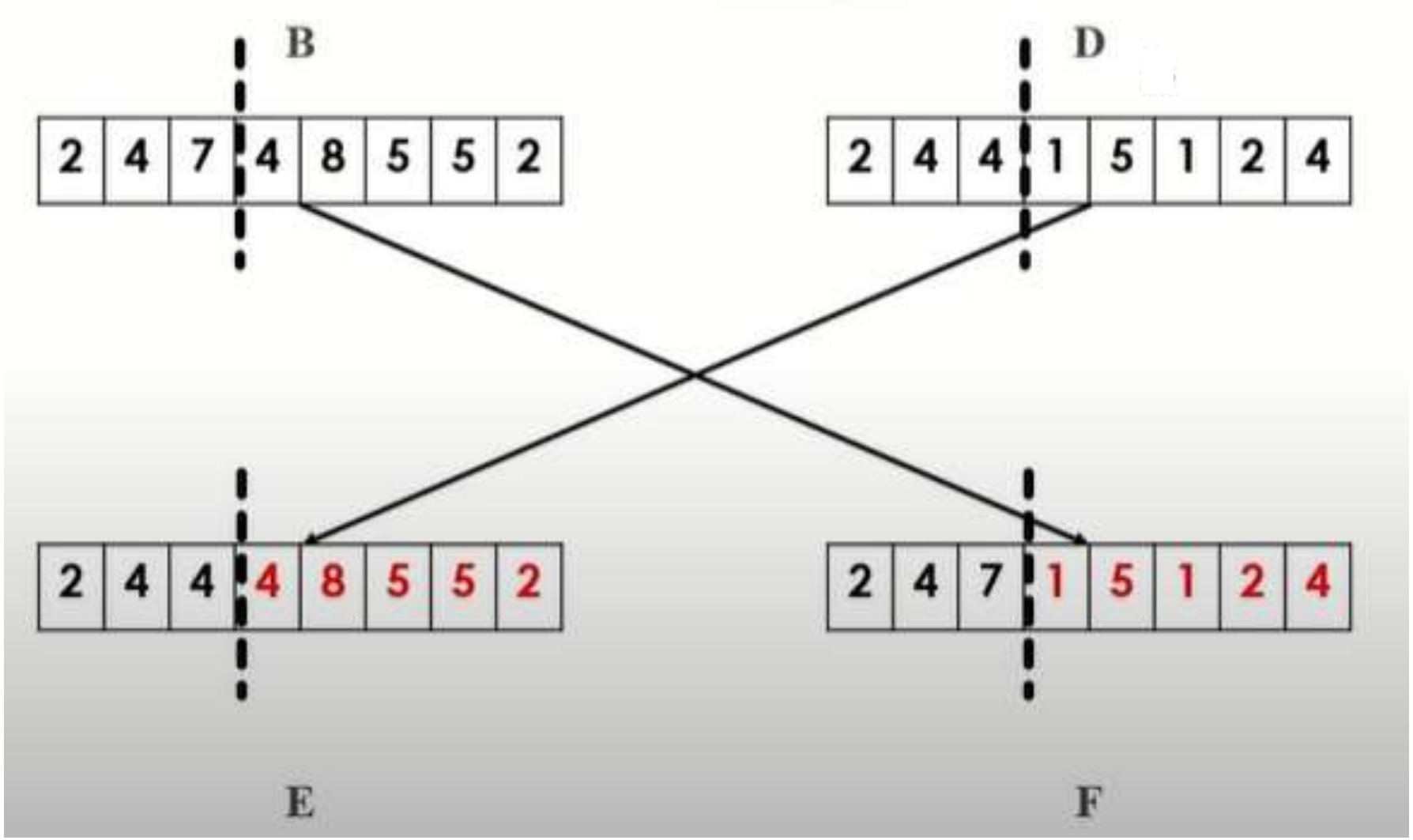
- There are various methods of selection
- Rouletted Wheel, Tournament, Rank, etc.
- Population is divided on a wheel according to their respective percentages of fitness and two fixed points are placed.
- Wheel is spun and those individuals are selected at which the fixed points are pointing when the wheel stops.

# Step 4: Selection (Cont.)

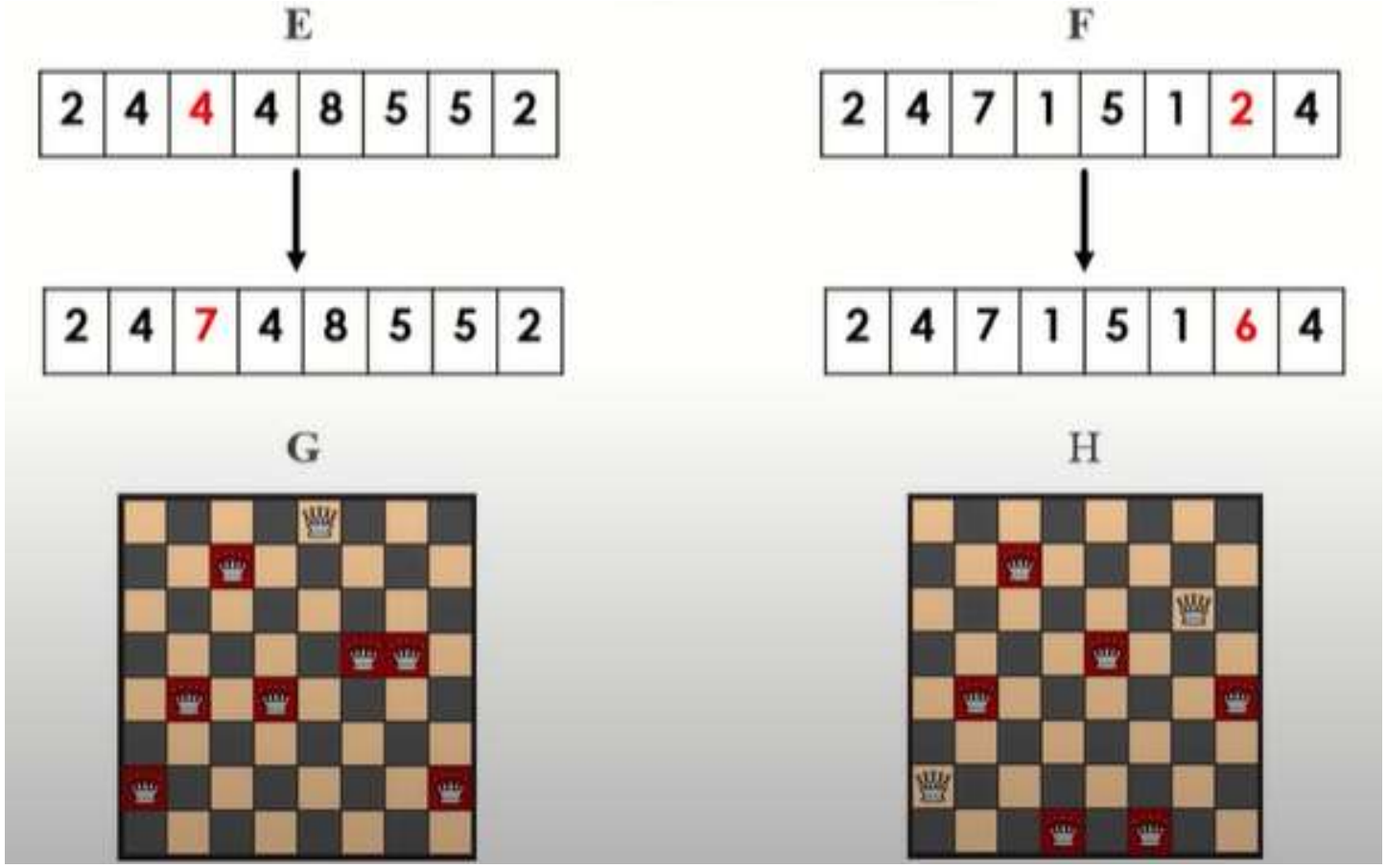



B and D are selected as parents

# Step 5: Crossover



# Step 6: Mutation



# Step 7: Repeat

- All steps are repeated until best solution is reached
- Best solution = Highest fitness score (28 in this case)
- Summary
  - Method of representation is formulated
  - Random initial population is generated
  - Fitness function is applied
  - Selection of parents
  - Crossover of parents to produce next generation
  - Mutation to bring diversity
  - All steps are repeated until solution is reached

# Problem Types due to incompleteness

- Sensorless problems
  - Agent has no sensor at all
  - Each action can lead to several possible initial states to several possible successor states.
- Contingency problems
  - When the environment is such that the agent can obtain new information from its sensors after acting, the agent faces a contingency problems.
  - A problem is called ***adversial*** if the uncertainty is caused by the actions of another agent.
- Exploration problems
  - When the states and actions of the environment are unknown, the agent must act to discover them.

| Aspect | Uninformed Search | Informed Search |
|---|---|---|
| Knowledge | Searches without any prior knowledge. | Uses knowledge, such as heuristics, to guide the search. |
| Time Efficiency | Generally more time consuming. | Finds solutions quicker by prioritizing certain paths. |
| Complexity | Higher complexity due to lack of information, affecting time and space complexity. | Reduced complexity and typically more efficient in both time and space due to informed decisions. |
| Example Techniques | Depth-First Search (DFS), Breadth-First Search (BFS). | A* Search, Heuristic Depth First Search, Best-First Search. |
| Solution Approach | Explores paths blindly. | Explores paths strategically towards the goal. |