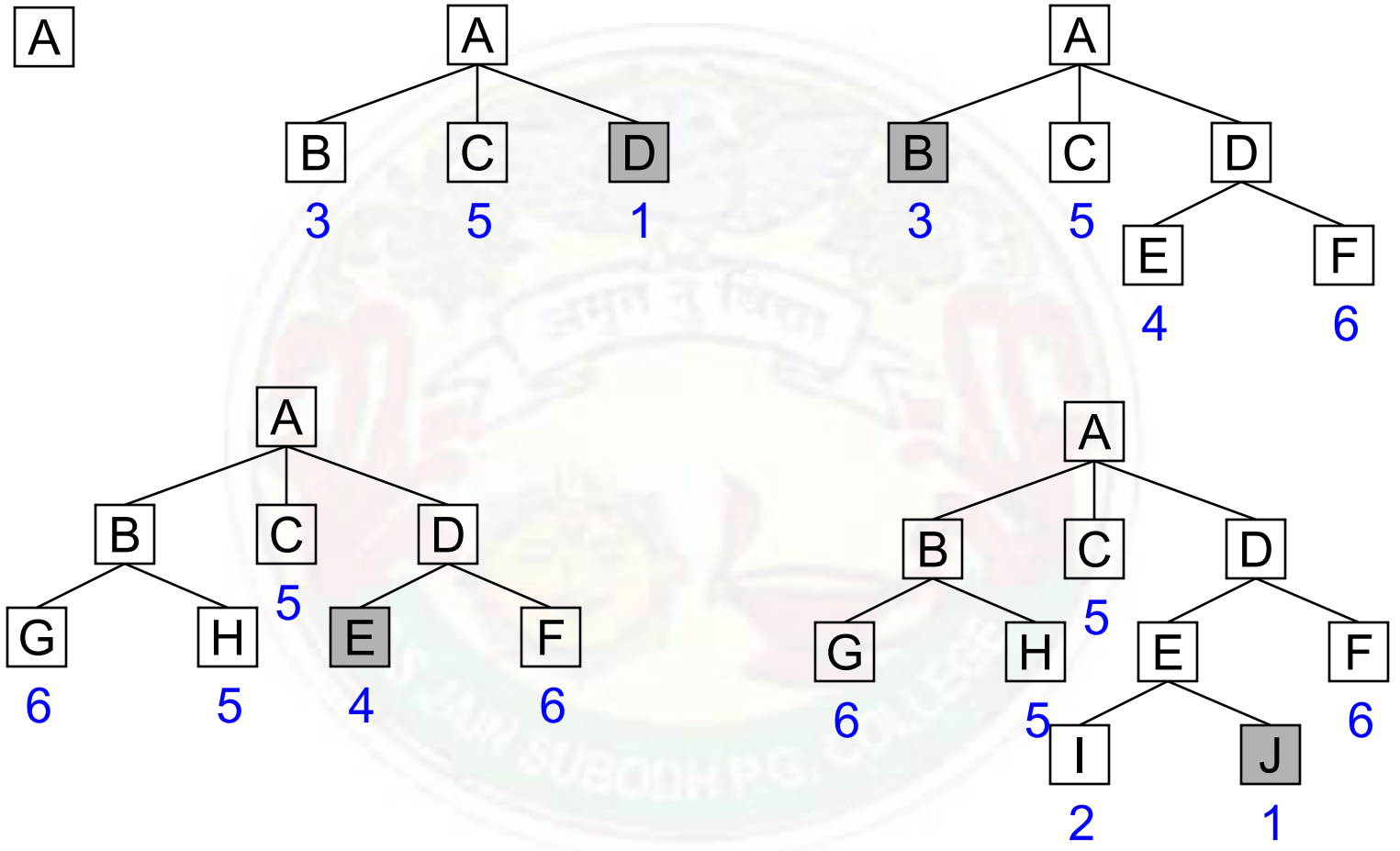
Best-First Search

- Combines the advantages of both Breadth-First search and Depth-First search Techniques.
  - **DFS:** follows a single path, don't need to generate all competing paths.
  - **BFS:** doesn't get caught in loops or dead-end-paths.
- **Best First Search**: Follow a single path at a time, but switch paths whenever some competing path looks more promising then the current one does.
- While goal not reached do the following:-
  1. Generate all potential successor states and add to a list of states.
  2. Pick the best state in the list and go to it.

Best-First Search Example



Best-First Search is very similar to the Steepest-Ascent Hill Climbing with two exceptions:-

- In Hill Climbing one move is selected and all others are rejected and never to be reconsidered. While in Best-First search, one move is selected and others are kept around so they can be revisited later if the selected path becomes less promising.
- The best available state is selected in Best-First Search, even if that state has a value that is lower than the value of the state that was just explored. This is in contrast with Hill Climbing which will stop if there are no successor state with better values than the current state.

Best-First Search

- **OPEN:** nodes that have been generated and have had the heuristic function applied to them but have not yet been examined(i.e. had their successors generated).

  Open is actually a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.

- **CLOSED:** nodes that have already been examined. We need to keep those nodes in memory.

  Whenever a new node is generated, check whether it has been generated before.

Best-First Search Algorithm

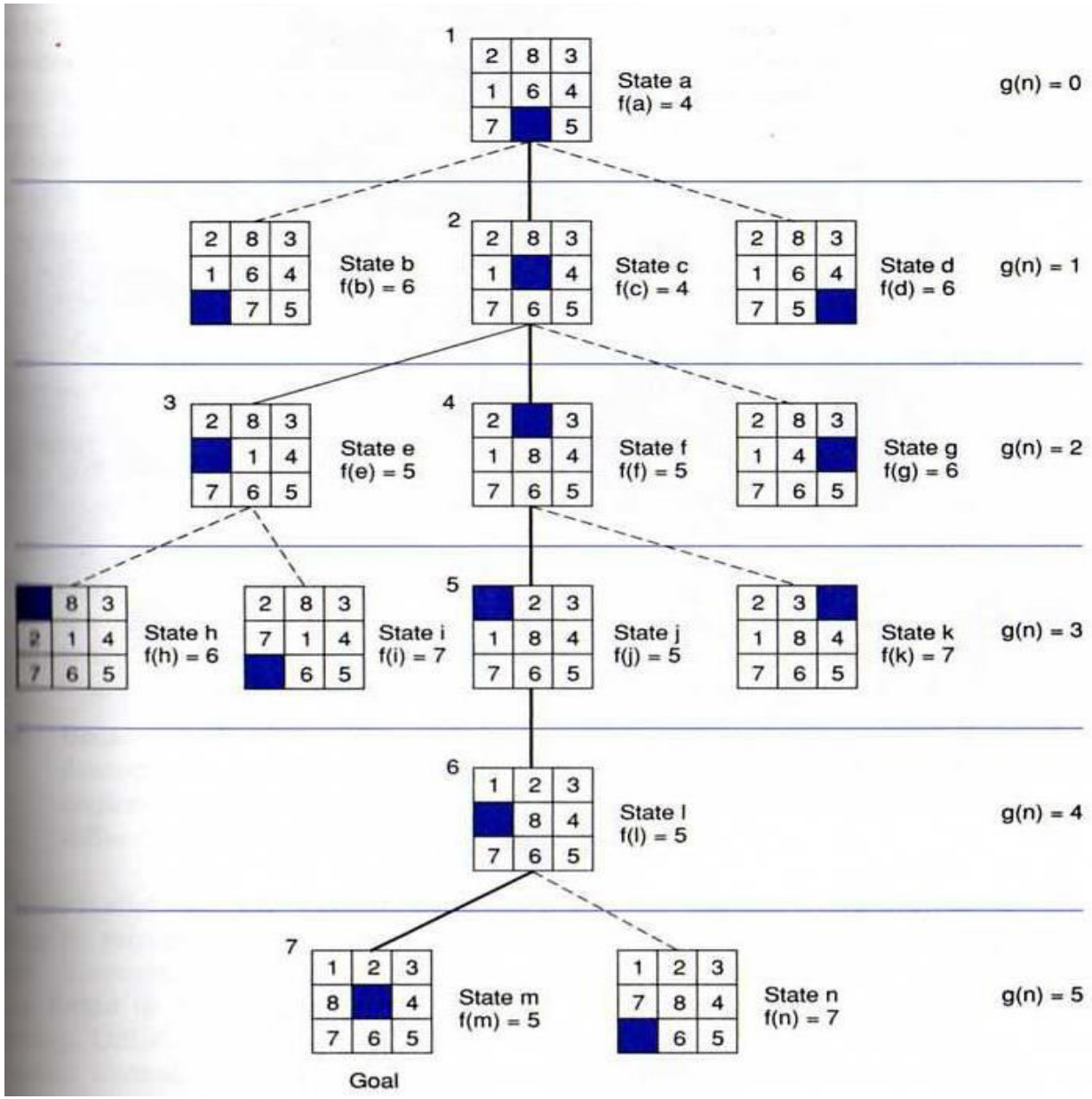
1. OPEN = {initial state}.
2. Loop until a goal is found or there are no nodes left in OPEN:
   a) Pick the best node in OPEN
   b) Generate its successors
   c) For each successor:
      (i) If it has not been generated before, evaluate it, add it to OPEN and record its parent.
      (ii) If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

A* Search

- The most widely- Known form of Best-First search is called A* Search.
- It evaluates nodes by combining g(n) and h'(n).
- **g(n)** :- cost of the cheapest path from the initial state to node n.
- **h'(n)** :- estimated cost of the cheapest path from node n to a goal state.

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

- A* minimizes the total path cost.
- Under the right conditions A* provides the cheapest cost solution in the optimal time!


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

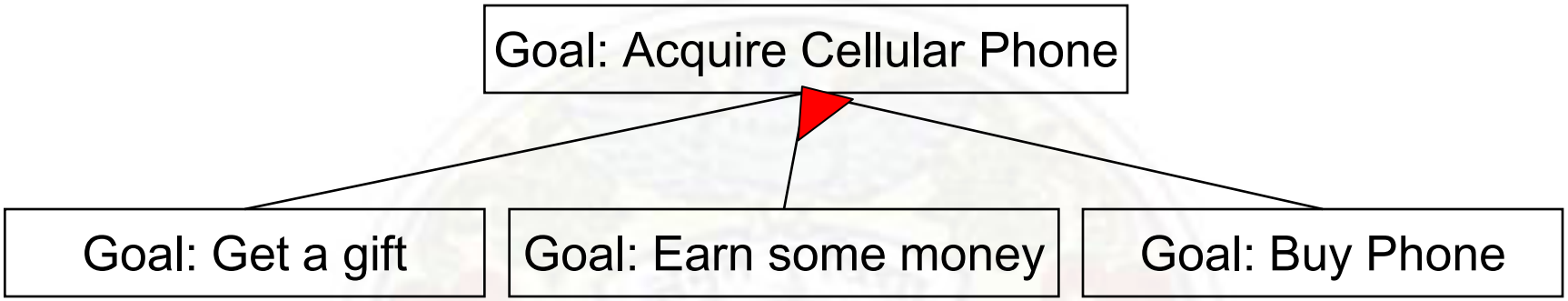
A* Algorithm

1. Create a priority queue of search nodes (initially the start state). Priority is determined by the function f)
2. While queue not empty and goal not found:
   a) Get best state x from the queue.
   b) If x is not goal state:
      (i) Generate all possible children of x and save path information with each node.
      (ii) Apply f to each new node and add to queue.
      (iii)Remove duplicates from queue (using f to pick the best).

Problem Reduction Search

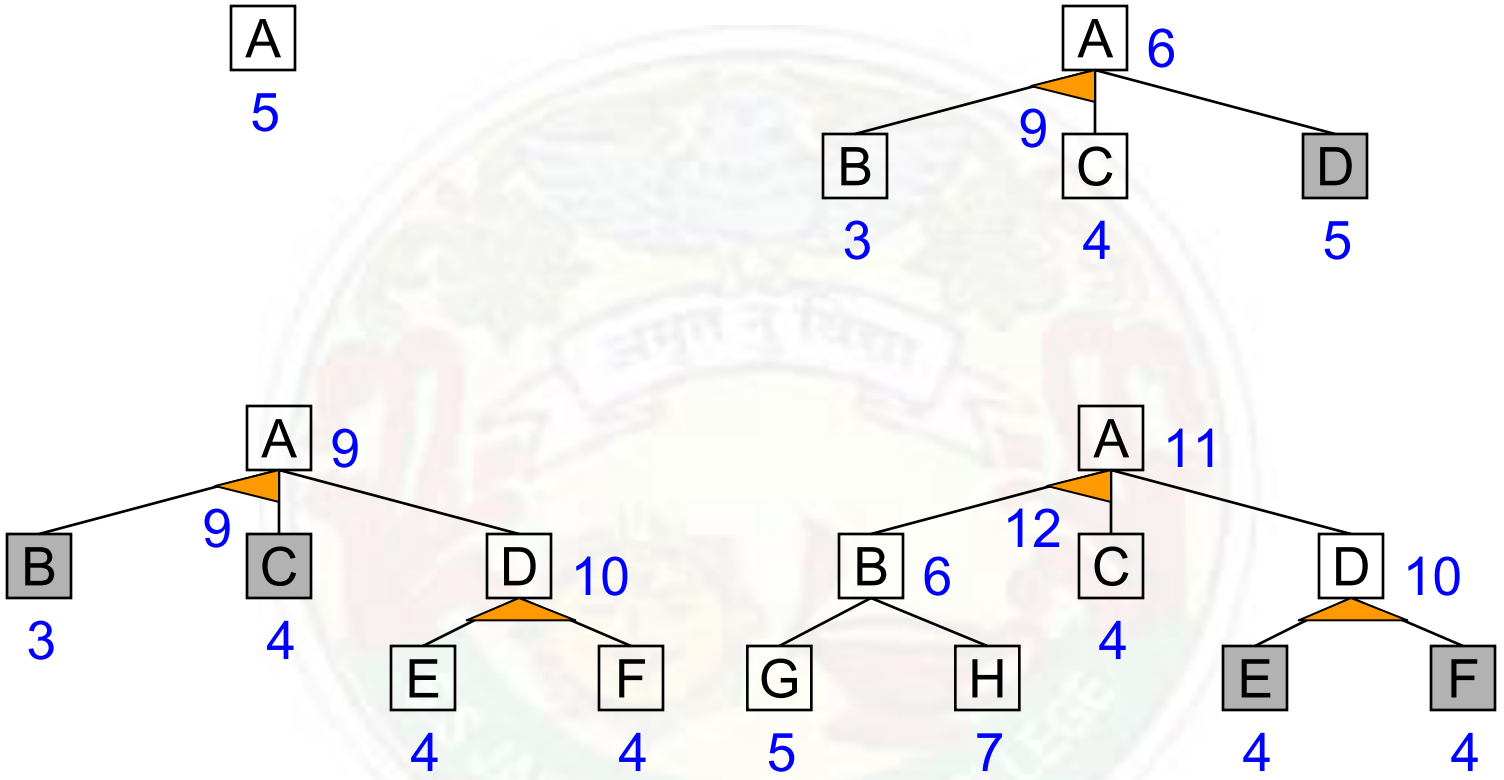
- Planning how best to solve a problem that can be recursively decomposed into sub problems in multiple ways.
- In OR graphs, arcs indicate the number of alternative ways.
- The AND-OR graph is useful for representing the solution of complicated problems.
- AND arc may point to any number of successors, all of which must be solved.
- In AND-OR graph, we need algorithm similar to Best-First Search but with the ability to handle the AND arcs appropriately.

Problem Reduction



AND-OR Graphs

Problem Reduction: AO*



AO* Search Algorithm

1. Place the start node on open.
2. Using the search tree, compute the most promising solution tree TP .
3. Select node n that is both on open and a part of tp, remove n from open and place it on closed.
4. If n is a goal node, label n as solved. If the start node is solved, exit with success where tp is the solution tree, remove all nodes from open with a solved ancestor.
5. If n is not solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. Remove all nodes from open ,with unsolvable ancestors.
6. Otherwise, expand node n generating all of its successor compute the cost of for each newly generated node and place all such nodes on open.
7. Go back to step(2)

Constraint Satisfaction

- Many AI problems can be viewed as problems of constraint satisfaction.
- Cryptarithmetic puzzle Example:

```
  SEND
+ MORE
 MONEY
```

- As compared with a straightforward search procedure, viewing a problem as one of constraint satisfaction can reduce substantially the amount of search.

Constraint Satisfaction

- Operates in a space of constraint sets.
- Initial state contains the original constraints given in the problem.
- A goal state is any state that has been constrained “enough”.

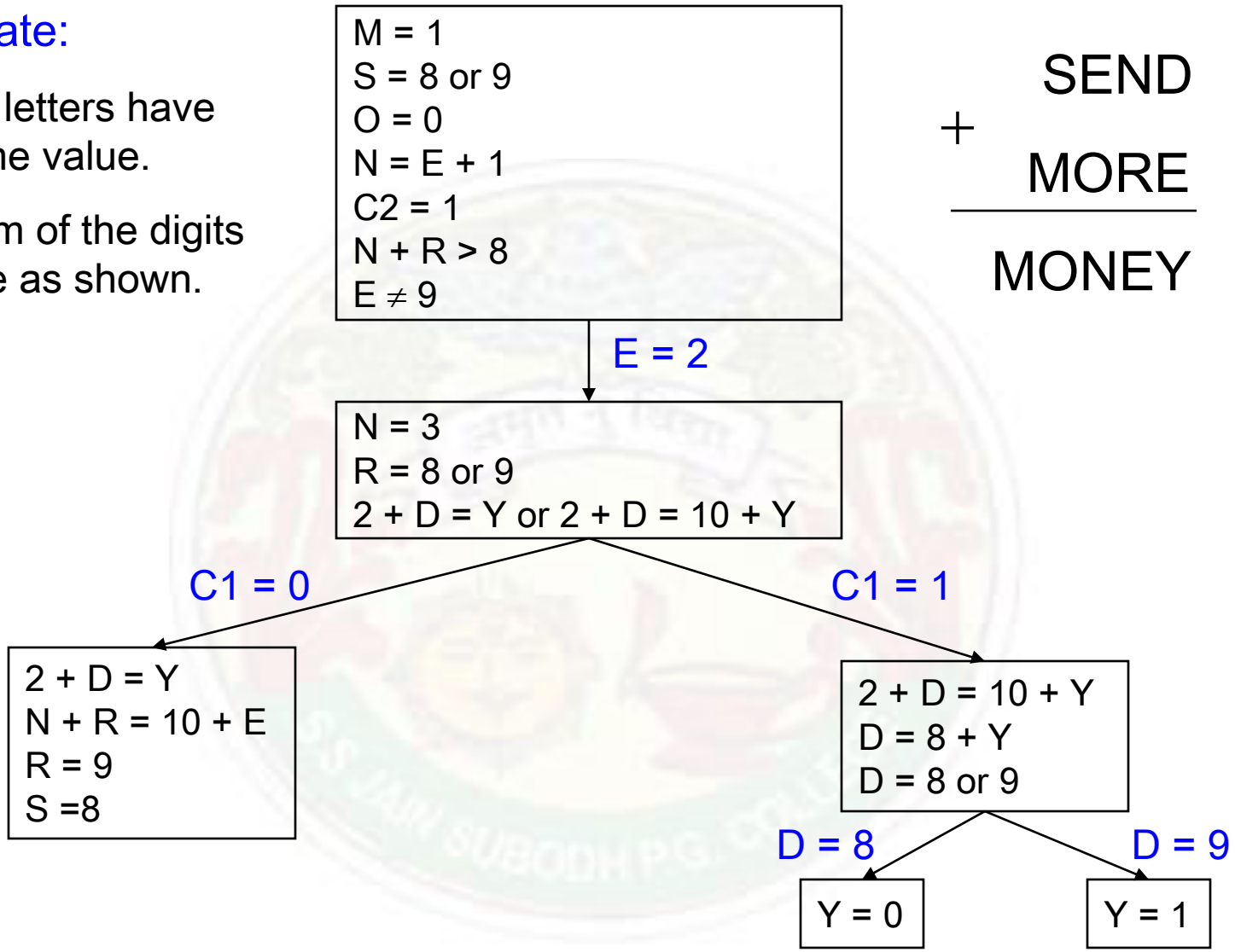
Two-step process:

1. Constraints are discovered and propagated as far as possible.

2. If there is still not a solution, then search begins, adding new constraints.

Initial state:

- No two letters have the same value.
- The sum of the digits must be as shown.



SEND
\+ MORE
MONEY

Constraint Satisfaction

Two kinds of rules:

1. Rules that define valid constraint propagation.

2. Rules that suggest guesses when necessary.

Backtracking

- Backtracking is a general algorithm for finding all (or some) solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons each partial candidate c ("backtracks") as soon as it determines that c cannot possibly be completed to a valid solution
- Backtracking is similar to DFS but uses less space, keeping just one current solution state and updating it.
- Backtracking is most efficient technique for problems, like n-queen problem.

Thank You