P problems (Polynomial time)

Problems that can be solved efficiently in polynomial time ($O(nK)$, $O(n^K)$, ~~O(n~~ where K is a constant).

Example: Finding the shortest path in a graph.

NP problems (Non-deterministic polynomial time)

Problems for which a given solution can be verified in polynomial time, even if finding the solution might take exponential time.

Example: Sodoku puzzle solving.

Difference between NP-Hard and NP Complete

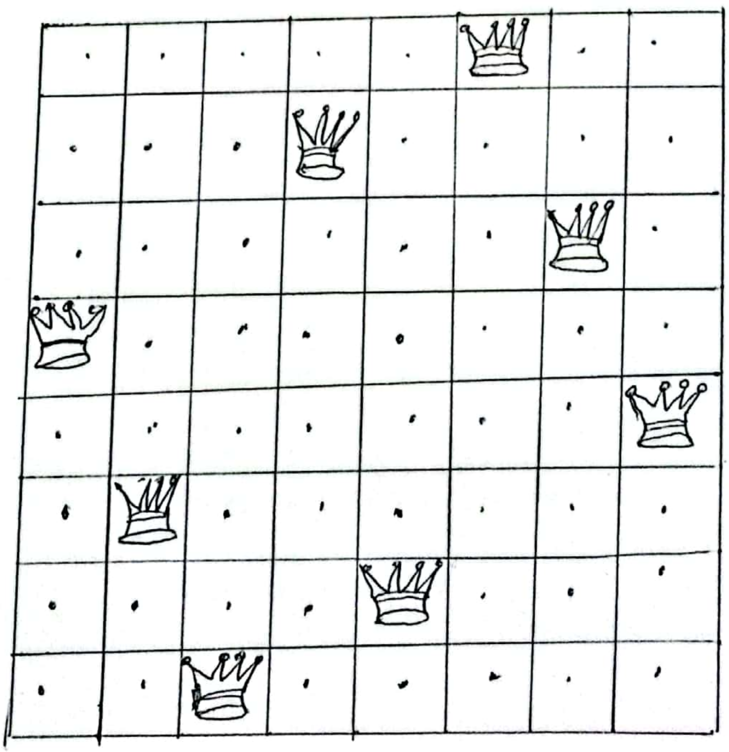
problems:

NP hard problems	NP complete problems
Problems that are at least as hard as NP problems but are not necessarily in NP	The hardest problems in NP. if one NP complete problem is solved in polynomial time, all NP problems can be solved in polynomial time.
NP hard problems may be not in NP.	all NP-complete problems belongs to NP.
May not be verifiable in polynomial time	A given solution can be verifiable in polynomial time
No known polynomial-time solution exists.	No known polynomial-time solution exists, but they are varifiable in polynomial time.

Backtracking method:

Backtracking is a recursive algorithmic technique used to solve problems by trying out different possibilities and undoing (backtracking) incorrect choices when they lead to a dead end. It is commonly used in decision making problems where multiple solutions are possible.

A possible solution space for the 8-Queens problem:

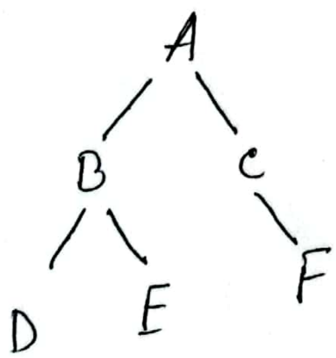


Breath-first Search (BFS)

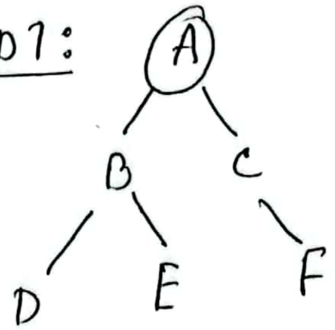
Algorithm:

1. Start from a chosen source node
2. Initialize a queue and add the source node
3. Mark the source node is visited.
4. While the queue is not empty
   - Dequeue a node
   - process it (print it)
   - Enqueue all it unvisited neighbors and mark them as visited
5. Repeat until all reachable nodes are visited.

For example:

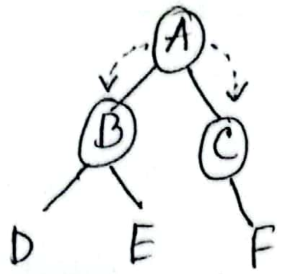


Step 1:



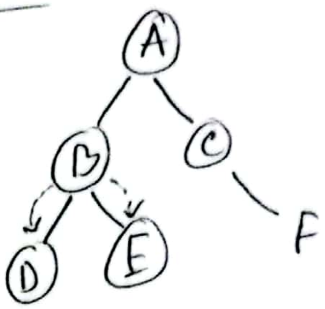
visited queue: A

Step 2:



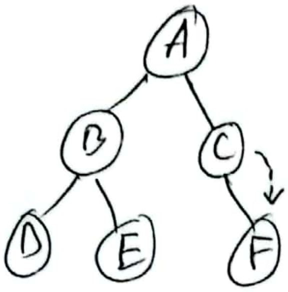
visited queue: A→B→C

Step 3:



visited queue: A→B→C→D→E

Step 4:



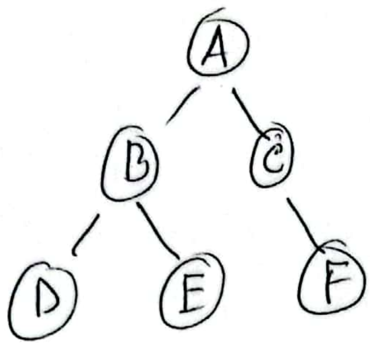
visited queue: A→B→C→D→E→F

Depth First Search (DFS)

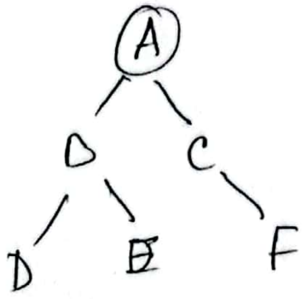
Algorithm:

1. Start from a chosen source node.
2. Mark the node as visited
3. Process the node (print it)
4. Recursively visit all unvisited nodes
5. Backtrack if neened and continue traversal.

For example:

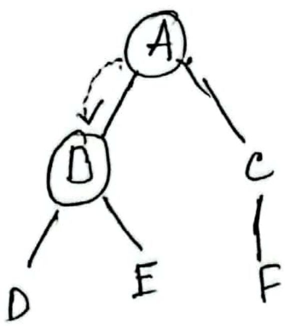


Step 1:



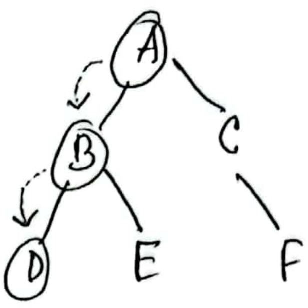
visited node: A ->

Step 2:



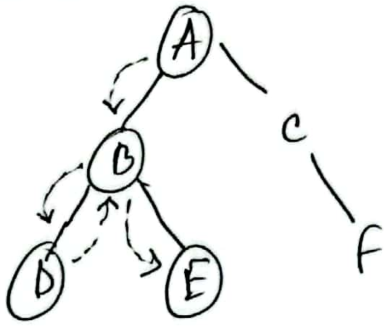
visited nodes: A -> B

Step 3:



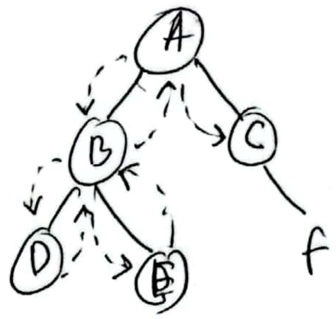
visited nodes: A -> B -> D

Step 4:



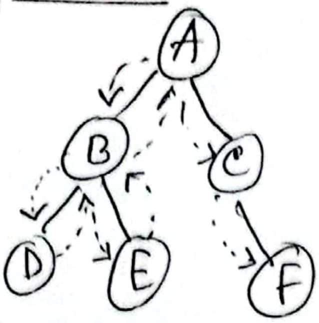
visited nodes: A -> B -> D -> E

Step 5:



visited nodes: A -> B -> D -> E -> C

Step 6:

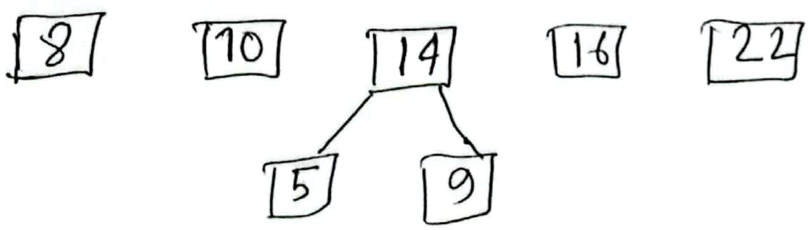


visited nodes: A → B → D → E → C → F

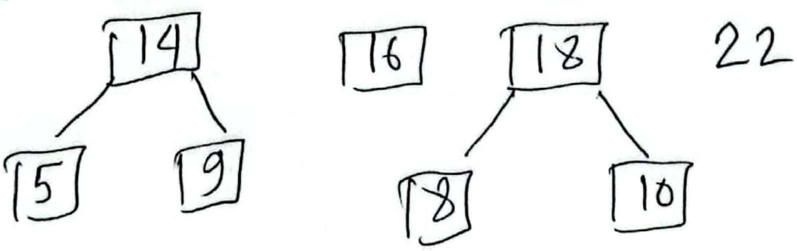
Huffman code Algorithms

A	B	C	D	E	F
22	16	10	8	5	9

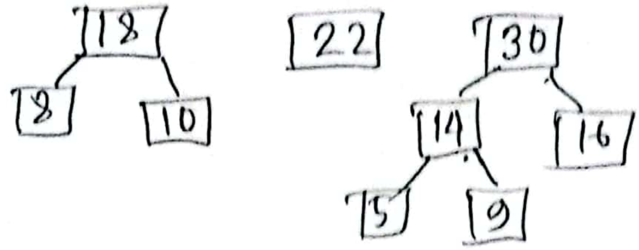
Step 1:



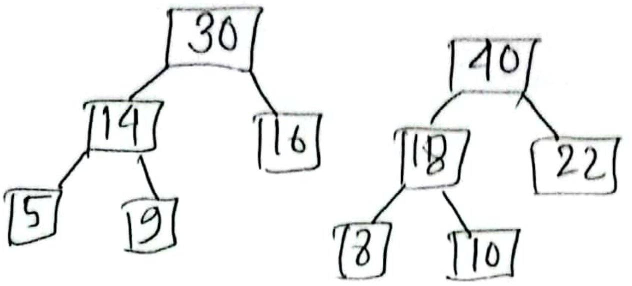
Step 2:



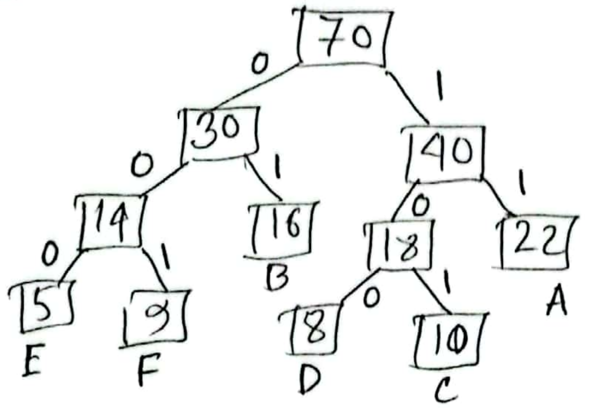
step3:



step4:



step5:



A = 11 = 44 bits

B = 01 = 32 bits

C = 101 = 30 bits

D = 100 = 24 bits

E = 000 = 15 bits

F = 001 = 27 bits

total = ~~535 bits~~ 172 bits

binary bits = ~~16 bits~~ 16 bits

8X6 alphabets = ~~48~~ 48 bits

total = ~~599 bits~~ 236 bits

$\therefore$ compression ratio $= \frac{\cancel{589}\ 236}{70 \times 8}$

$= \frac{236}{560}$

$= 0.42$