DIGITAL ASSIGNMENT - 1

CSE2012 - DESIGN AND ANALYSIS OF ALGORITHM

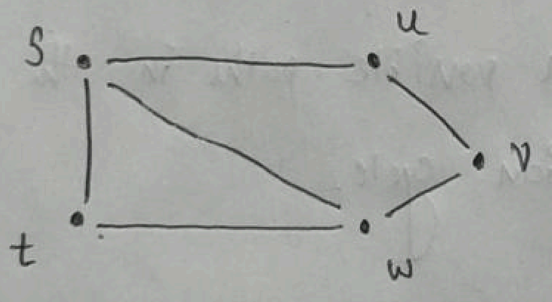
GAYATHRI P

ANIRUDH VADERA - 20BCE2940

SLOT - G1+TG1

QUESTION - 1

Discuss in detail Hamiltonian cycle problem. Differentiate Hamiltonian cycles problem and TSP problem. Consider the graph given below with 'S' as start vertex. Use backtracking approach and find Hamiltonian cycles (atleast 2 cycles) in the given graph.



Solution:

formally: Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges in the graph, does there exist a simple cycle that contains every vertex in V? A simple cycle is a path that starts and ends at the same vertex, and does not visit any vertex more than once, except for the starting and ending vertex.

The problem involves determining wether a graph contains a cycle that visit every vertex exactly once. Such a cycle is called a Hamiltonian cycle or Hamiltonian circuit.

It is an NP-Complete problem and has many applications such as in routing and scheduling problems.

Some ways to solve it:

1) Brute force

↳ Enumerate all possible cycles in graph and check wether each cycle is Hamiltonian or not.

↳ Infeasible for large graphs

2) Backtracking

↳ Symmetrically explores all possible paths in the graph to find a Hamiltonian cycle.

3) Branch and bound

↳ Combines the backtracking approach with a technique of pruning the search space

↳ Explores the graph by branching out to new vertices and keeping track of the minimum number of edges needed to complete a Hamitonian cycle.

Difference b/w Hamiltonian cycle and TSP.

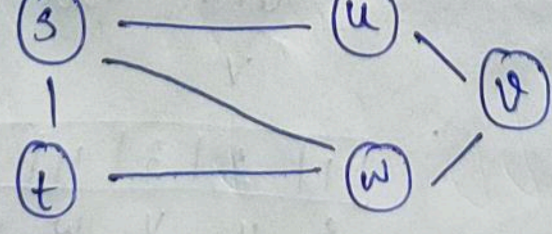
- The main difference b/w TSP and Hamiltonian cycle is that in Hamiltonian cycle we are not sure wether a tour that visits each city exactly once exists or not, and we have to determine it. In TSP, a Hamiltonian cycle always exists because the graph is complete and the problem is to find a Hamiltonian cycle with minimum weight.

- Hamiltonian problem is a decision problem, while TSP is an optimization problem

- Hamiltonian problem is easier to solve than TSP.
  The Hamiltonian problem is a NP-complete, which means that is computationally hard to solve for large graphs, but it is easier to approximate than the TSP.
  The TSP is a NP-hard, which means that it is computationally difficult to solve, and there is no known algorithm that can solve it for all instances.

Solving the problem

Graph :



First answer

Let Adjancency matrix be

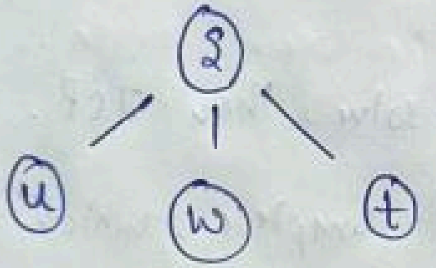
$$G : \begin{array}{c|ccccc|} & s & u & v & w & t \\ \hline s & 0 & 1 & 0 & 1 & 1 \\ u & 1 & 0 & 1 & 0 & 0 \\ v & 0 & 1 & 0 & 1 & 0 \\ w & 1 & 0 & 1 & 0 & 1 \\ t & 1 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

if there exits a edge b/w node a to b then it has a value 1 in the matrix otherwise 0

Let the visited main array be initially

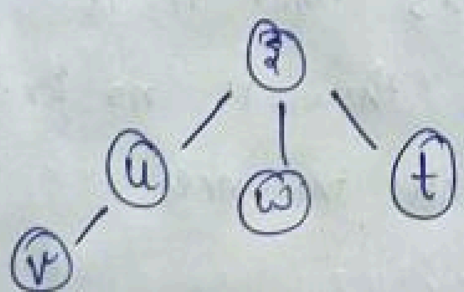
1	1	1	1	1
s	u	v	w	t

↳ as s is starting vertex we start our tree from node s (root).



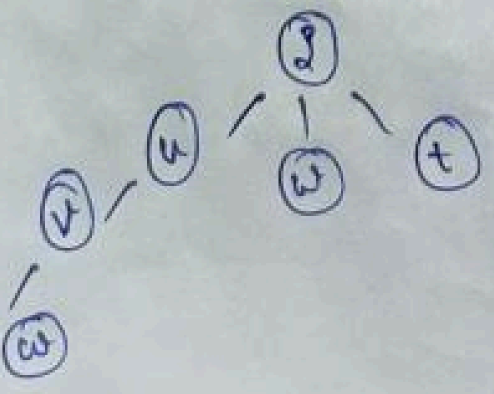
↳ append all children of s but we then check for u first

1	2	1	1	1
s	u	v	w	t



Let's add v adjacent to u

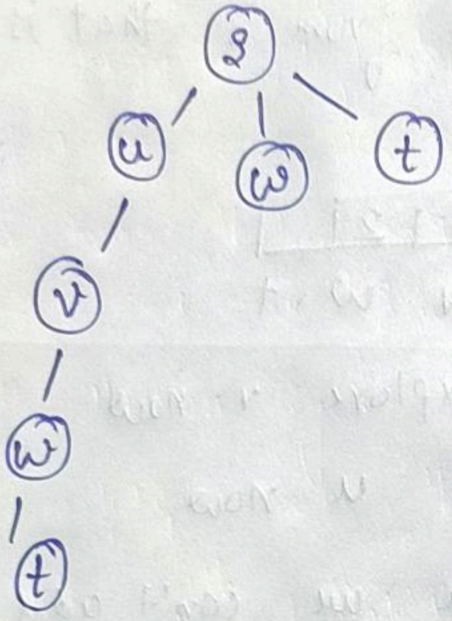
1	2	3	1	1
s	u	v	w	t



Let's add w adjacent to v

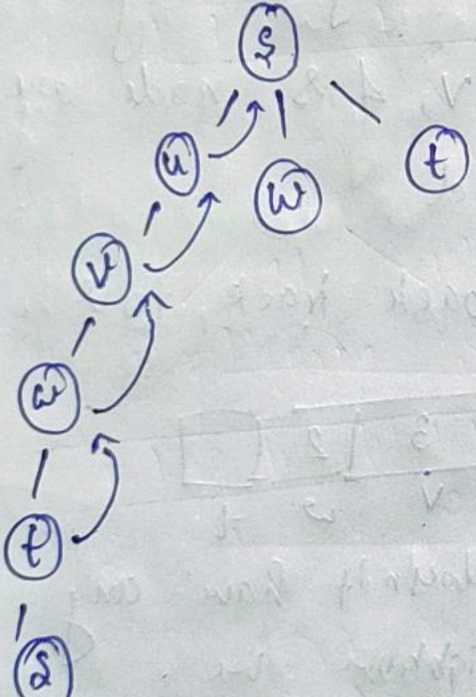
1	2	3	4	
s	u	v	w	t

Next let's add t adjacent to w



1	2	3	4	5
s	u	v	w	t

↳ now let's check if there exist a path back to starting vertex s.
and yes we have an edge



Solution 1

↳ Hence we get one answer as.

$\boxed{s \to u \to v \to w \to t \to s}$

↳ now let's try to find another answer
let's backtrack.

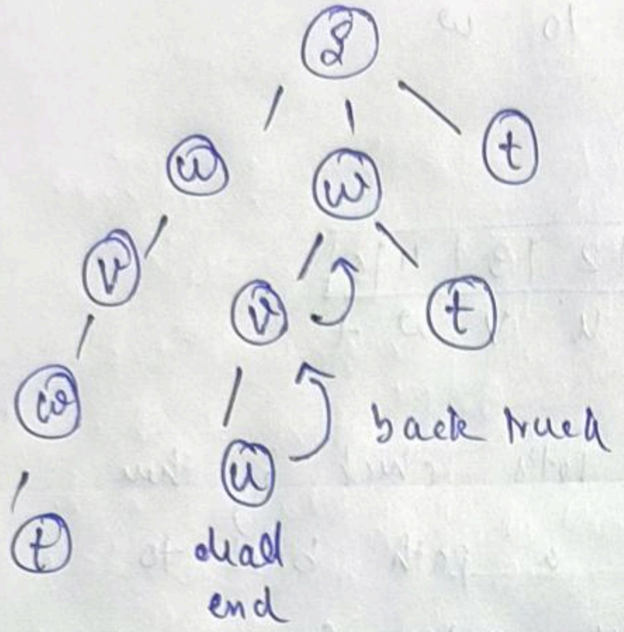
↳ As w has no more child left that has not yet been added so back track again
Hence we back track again for v & u with same reason

↳ As we back track we remove entries from our visited array

Second ans

after backtracking to s

s	u	v	w	t



↳ Let's explore the next unexplored from s that is w.

s	u	v	w	t
1	4	3	2	

↳ Let's explore v now

we add u now

Now for u we can't add any of it's adjacent neighbours as both v, & s node are already added

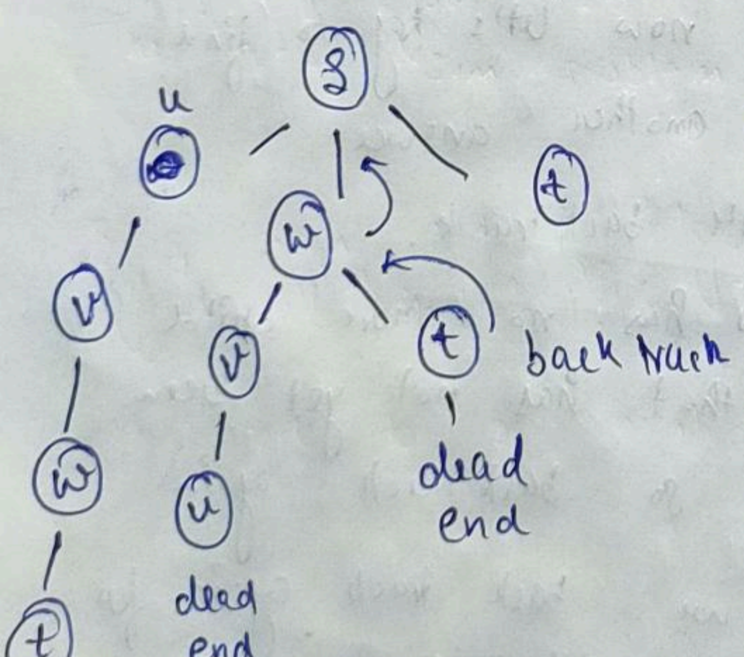
Hence we back track to v

s	u	v	w	t
1	1	3	2	

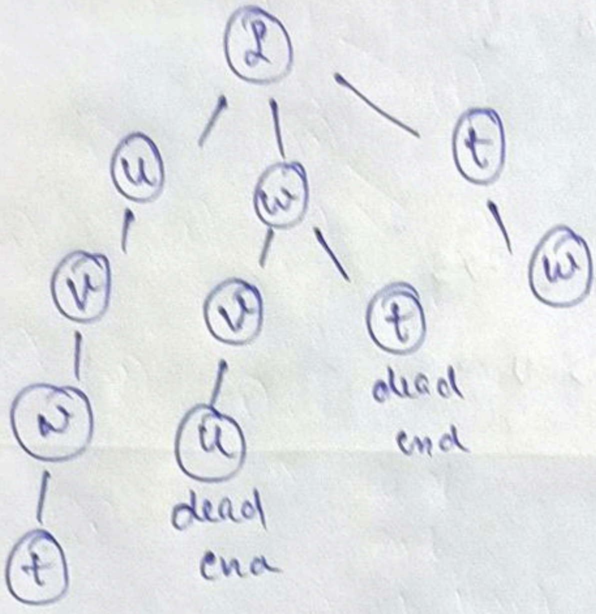
Now v doesn't have any other neighbour we back track again

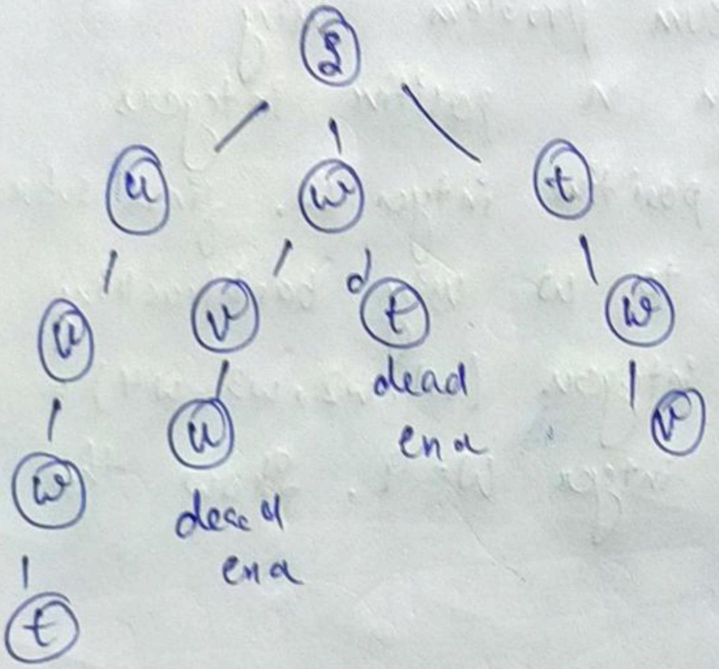
Now we explore next neighbour of w which is t.

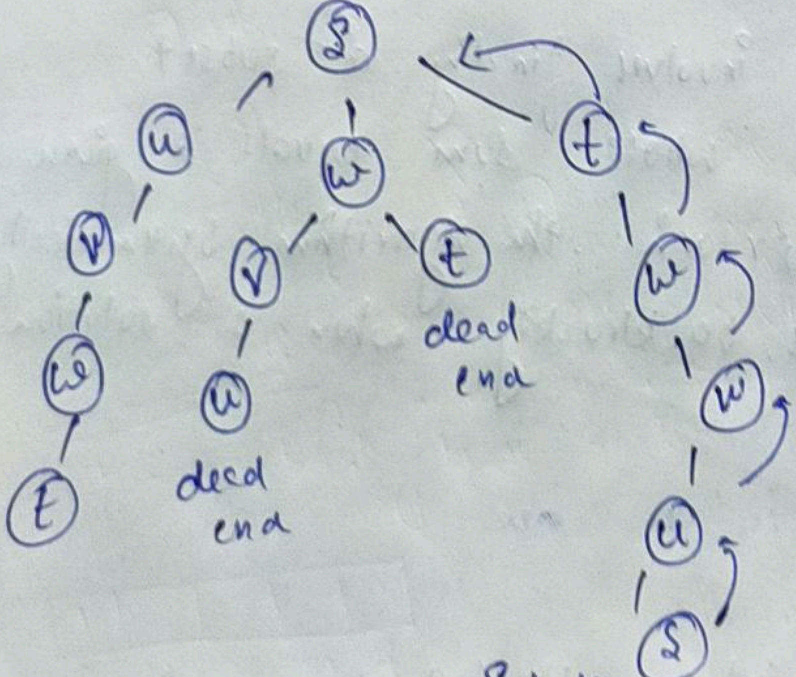
s	u	v	w	t
1	1	1	2	3



Now from t we can't go further hence we backtrack to s.







Solution 2

Now we explore the next unexplored neighbour of s which is t.

s	u	v	w	t
1				2

Let's go to w now and from w to v as s & t are already explored

s	u	v	w	t
1		4	3	2

we can now add last remaining node u.

and as there exist a edge from u to starting vertex s hence we found another solution

s	u	v	w	t
1	5	4	3	2

$\boxed{s \to t \to w \to v \to u \to s}$

Now we backtrack again to s.

Hence we got 2 solutions.

1) s → u → v → w → t → s

4 2) s → t → w → v → u → s

Question 2

Discuss in detail subset sum problem using backtracking approach. Given n positive integers w1, w2, w3, w4 and a positive integer W, find subsets of n integers that sum to W using backtracking approach. Let n = 4, Positive integers (w1, w2, w3, w4) = (2, 3, 4, 5) and a positive integer W = 9. Show the steps of your work.

Solution

The subset sum problem involves finding a subset of a given set of integers whose sum equals a given target value. In this approach, the algorithm systematically explores all possible solutions, backtracking when a solution is found to be invalid.

1) Formate the problem

given target t, set of integers W = (w1, w2 — wn).

2) Define the search space :

Search space is set of all possible subsets of S.

3) Define the solution space :

Set of all subsets of S whose sum equals t.

4) Define search tree :

It is a binary tree that represents all possible decisions that can be made when selecting elements from the set S. Each node in the tree represents a decision to include or exclude an element from the subset being considered.

5) Implement the backtracking algorithm :

Starts at the root node and recursively explore all possible paths through the tree. At each node the algorithm checks wether the current subset sums to the target value. If the subset is invalid i.e sum exceeds the target the algorithm back tracks and look for another path.

Time Complexity : O(2^n).

n → size of set

Pseudo Code:

s → set , t → target

```
function find (s,t).
    n = length(s)
        ↱ index
    function backtrack (sum, i , subset):
        if (sum == t):
            print (subset)
        elif (i < n and sum + s[i] <= t):
            subset.add (s[i])
            backtrack (sum + s[i], i+1, subset)
            subset.remove (s[i])
            backtrack (sum, i+1, subset).
    end function
    backtrack (0, 0, [])
end function
```

Ex. n = 4, given w or set w = (2, 3, 4, 5)

target = 9 = W.

let's first start with sum as 0 and index as 0 as well

calling backtrack (0, 0, [])

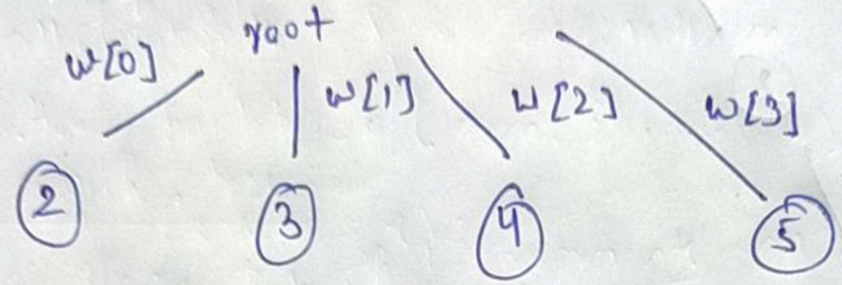
↳ empty subset

↳ index

↳ current sum

as sum is less than target we continue. let's first try including the first element

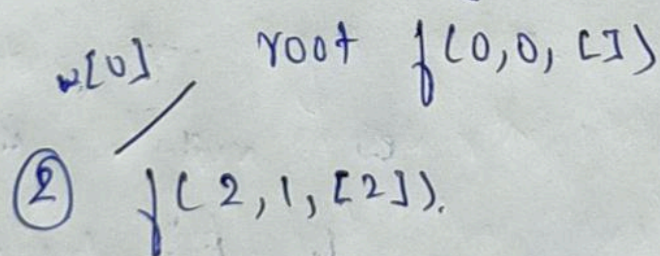
tree : → each node contains up to now sum.



let's first explore 2.

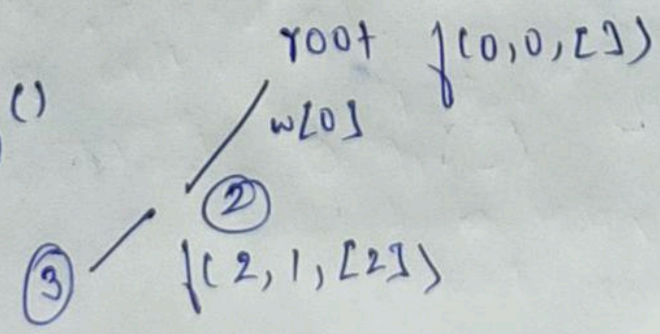
now at each point we can either include a index or exclude it so first level we first included 2 then we ignore 2 & add 3 then we ignore 2 & 3 & add 4 then we only include 5 ignoring all others.

let's first explore only 2



let's now include next index as well

function call ← f()



Lets include next index as well

root $f(0, 0, [])$

$f(2, 1, [2])$

2

$f(5, 2, [2,3])$

5

Lets include next ~~both~~ index as well

root $f(0, 0, [])$

$f(2, 1, [2])$

2

$f(5, 2, [2,3])$

5

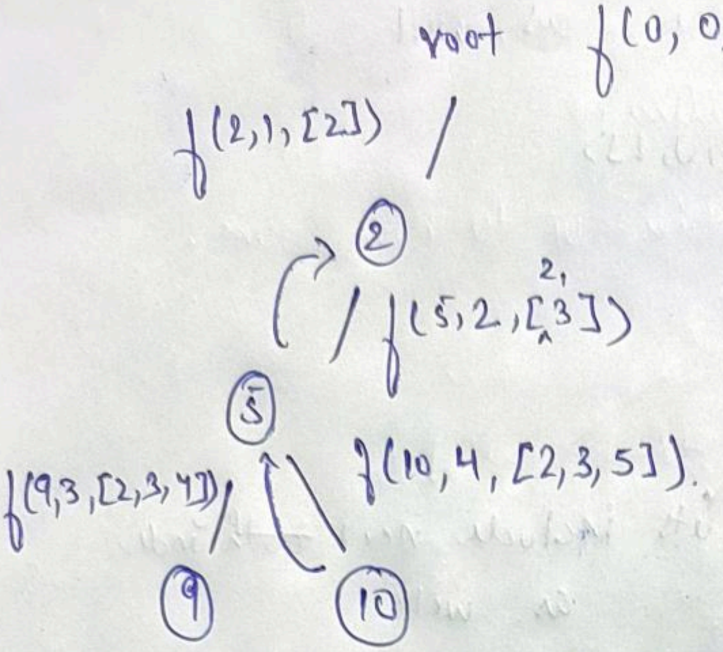
$f(9, 3, [2,3,4])$

9

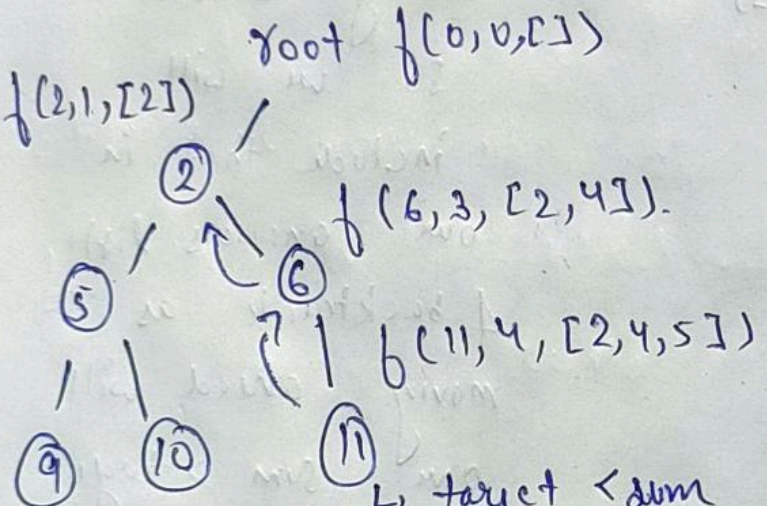
↳ == target we got our answer here hence we can stop here and lets do backtracking now

we got answer so we will include that in our answers list & backtrack as moving ahead will give sum > target which we don't want.

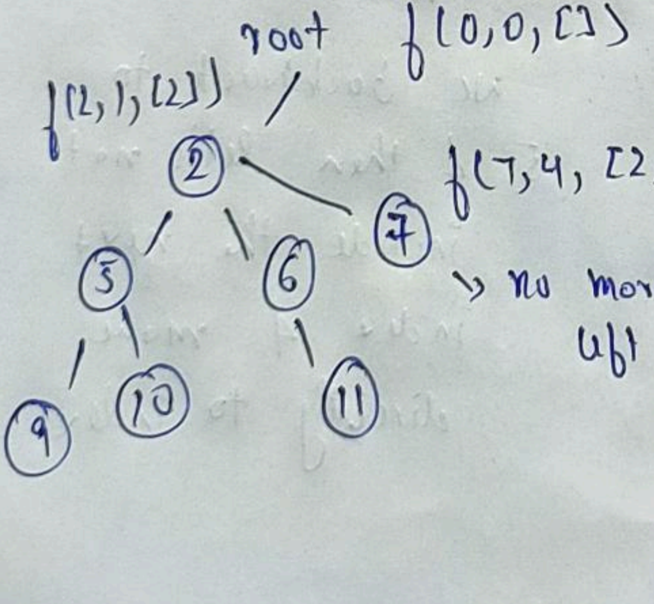
We backtrack to 5 & then let's not include the next index & move directly to index



↳ target > < sum
backtrack



↳ target < sum
backtrack



↳ no more index
left backtrack

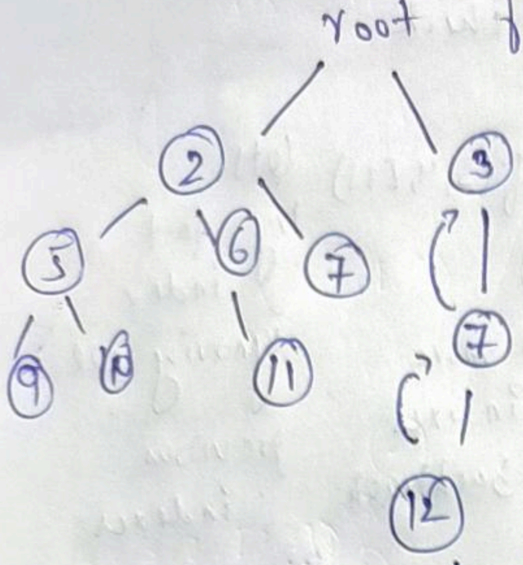
↳ as the target is ~~of~~ less than the sum obtained we backtrack from here at node 5

↳ Now at node 5 we reached end index of our array hence we can't go further hence we backtrack to node 2

↳ We continue the same process until we backtrack back to root.

↳ as no more indexes are left to add into subset we backtrack again to root & let's ignore first index for now.

$f(0,0,[\,])$



$f(3,2,[3])$

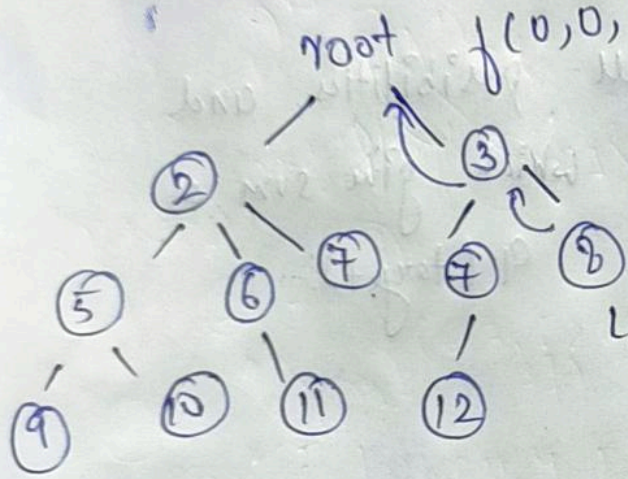
$f(7,3,[3,4])$

$f(12,4,[3,4,5])$

↳ sum > target backtrack

let's try next indexes after index 1.

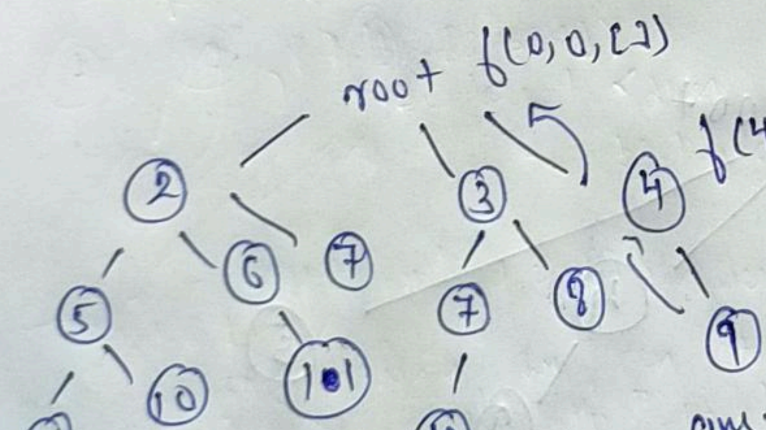
$f(0,0,[\,])$



$f(8,4,[3,5])$

↳ no more index to add back track

$f(0,0,[\,])$



$f(4,3,[4])$

$f(9,4,[4,5])$

ans. (sum == target)

we backtrack now

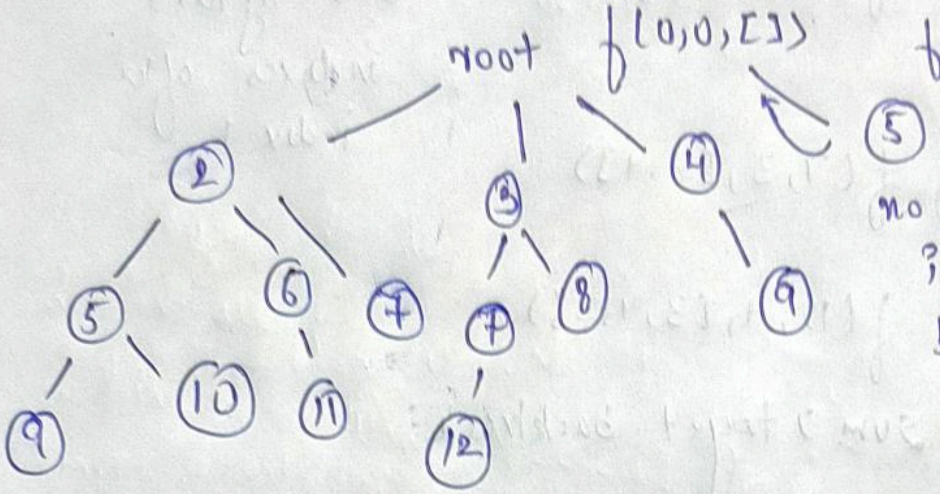
↳ let's now ignore 0 & 1st index and try 2nd index

↳ let's now add last index

we see sum again as target 9

Hence we add it to answer list

Finally we add the last index of check



no more indexes backtrack

lets try last index ignoring all previous indexes

Now we have tried all posibilities and we found 2 answers which give sum as 9 (target).

Sol 1. {2, 3, 4}

Sol 2 {4, 5}

final Tree