Problem-1 Minimum count of numbers required from given array to represent S.

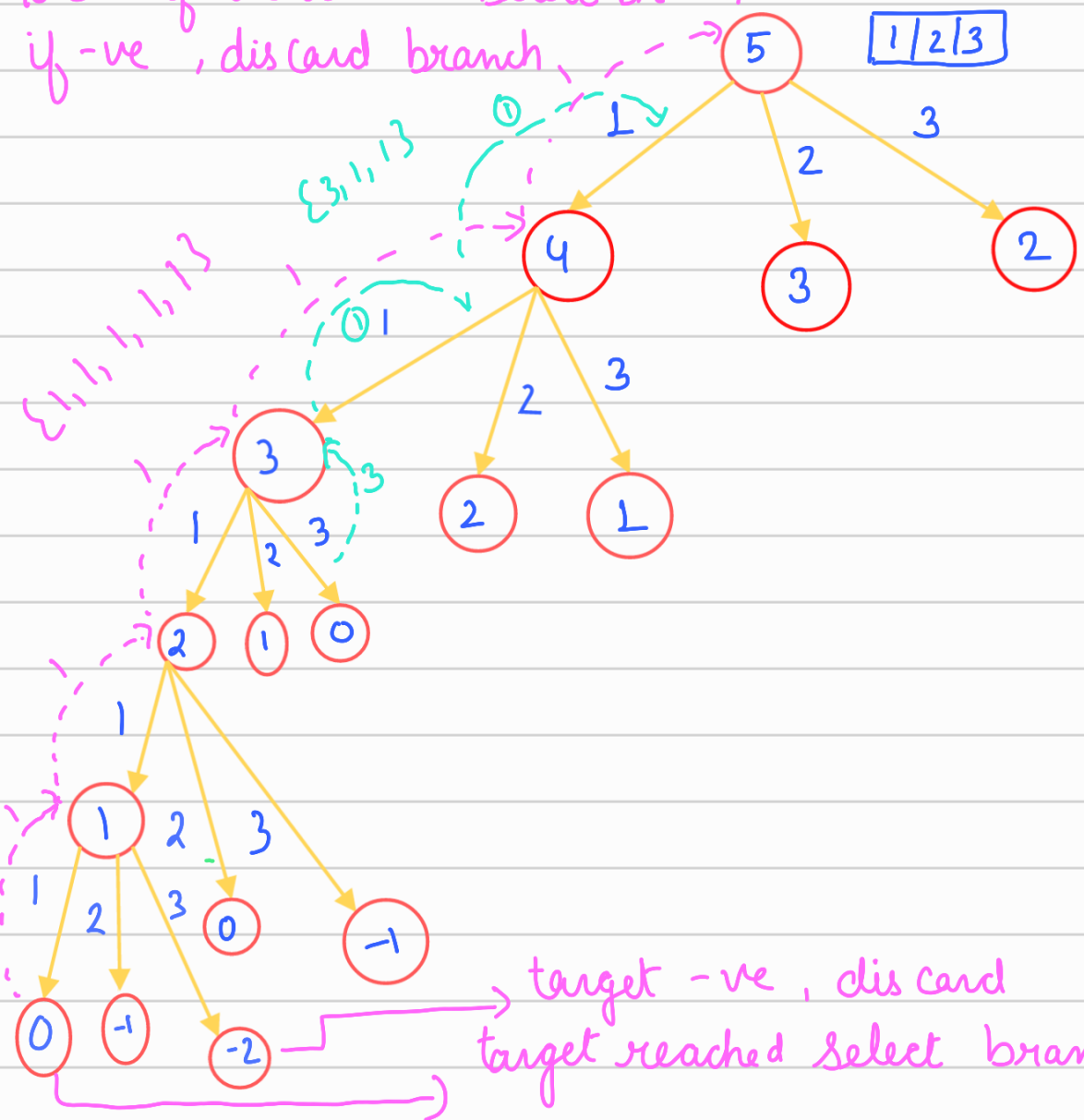
I/p → {1, 2, 3} , target = 5

1) {1, 1, 1, 1, 1} → Count 5 (1+1+1+1+1 = 5)
2) {1, 2, 2} → Count-3
3) {1, 1, 3} → 3
4) {1, 2, 1, 1} → 4
5) {2, 3} → 2

↳ Answer → min. element

Solution → we will start the target 5, & select the element from array. reduce the target by that element. Go until you get '0'.

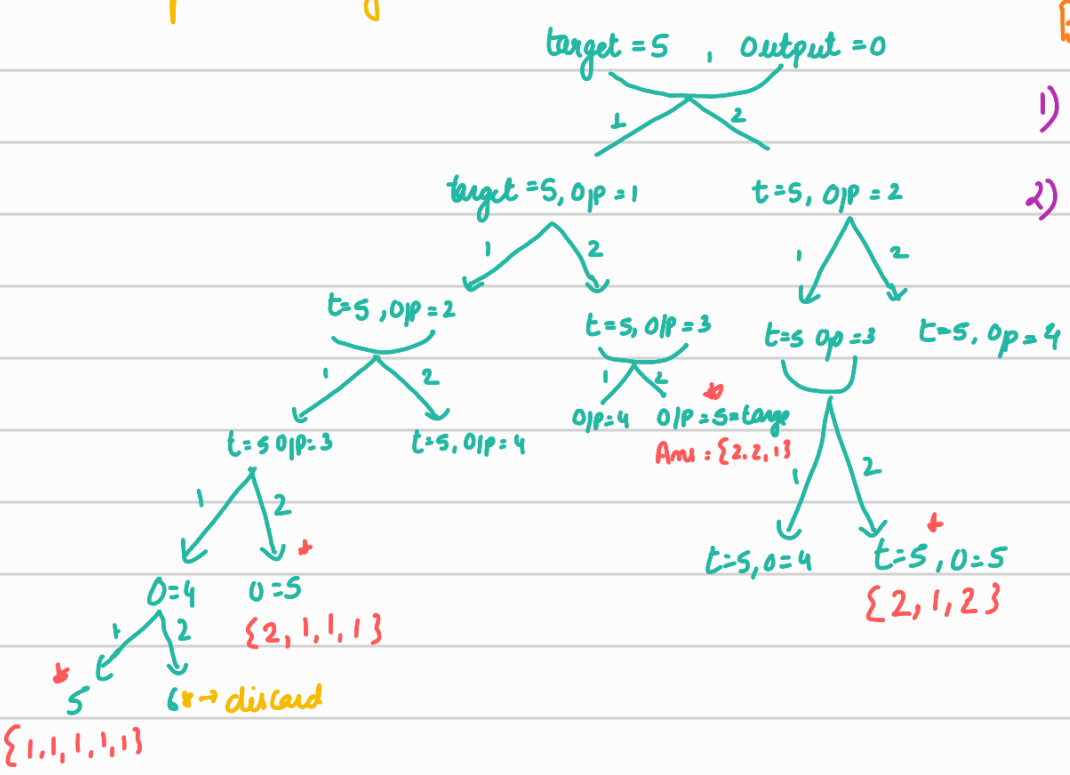
B.C → if we reach '0' select branch
if -ve, discard branch



We will also have to sent 'min' variable , which will keep track of getting min. elements

```cpp
#include <iostream>
#include <limits.h>
#include <vector>
using namespace std;

int solve(vector<int>& arr, int target){
    //base case
    if(target == 0){
        return 0;
    }

    if(target < 0){
        return INT_MAX;
    }

    //let's solve 1 case
    int mini = INT_MAX;
    for(int i=0; i<arr.size(); i++){
        int ans  = solve(arr, target - arr[i]);

        if(ans != INT_MAX){
             mini = min(mini,ans+1);
        }

    }

    return mini;
}

int main()
{
    vector<int> arr{1,2};
    int target = 5;

    int ans = solve(arr, target);
    cout << "Ans is: " << ans <<endl;

    return 0;
}

```

2nd way, suppose target = 5 & arr = {1,2}, then we will start with o/p = 0 & add each array element to it until output = target



Base case :-

1) if (target = = output)

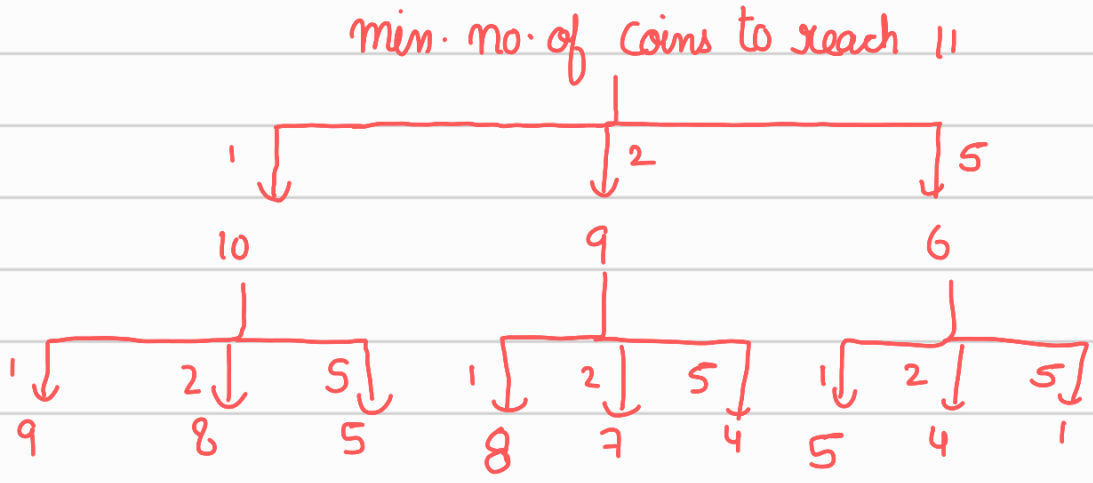
2) if (output > target)

Note → code implementation remaining

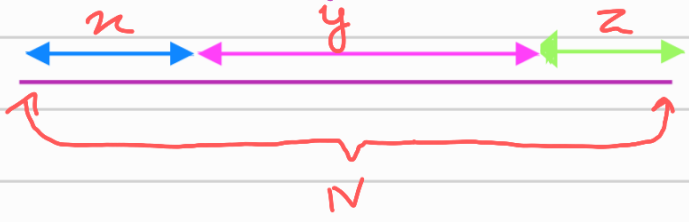
Note → House - Robbery - I

Problem statement → I/p → Infinite supply of coins (1, 2, 5)
target = 11 Rs

Solution

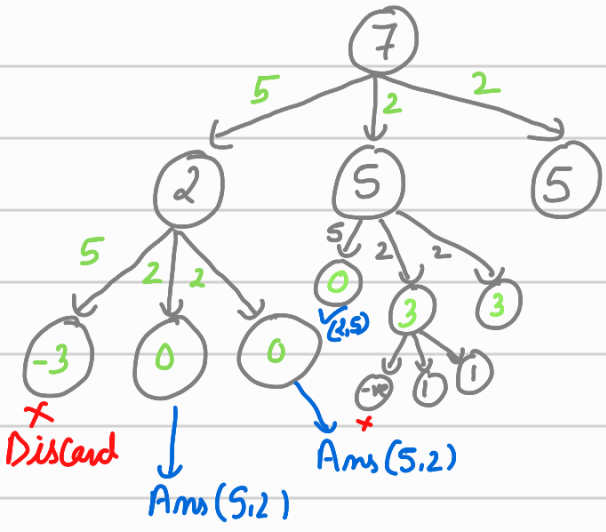


Problem Statement → Cut into segments
I/p → N → rod length



We have to find max no. of segment we can make provided that we can only use segments of length x, y & z.

Solution → Suppose x, y, z is {5, 2, 23 then



```cpp
#include <iostream>
#include<vector>
#include<limits.h>
using namespace std;

int solve(int n, int x, int y, int z ){
  //base case

  if(n == 0){
    return 0;
  }

  if(n < 0){
    return INT_MIN;
  }


  int ans1 = solve(n-x, x,y,z) +1;
  int ans2 = solve(n-y, x,y,z) +1;
  int ans3 = solve(n-z, x,y,z) +1;


  int ans = max(ans1, max(ans2,ans3));
  return ans;


}

int main() {

  int n = 8;
  int x = 3;
  int y = 3;
  int z = 3;

  int ans = solve(n, x,y,x);

  if(ans < 0){
    ans=0;
  }
  cout << "Answer is: " << ans << endl;

  return 0;
}
```

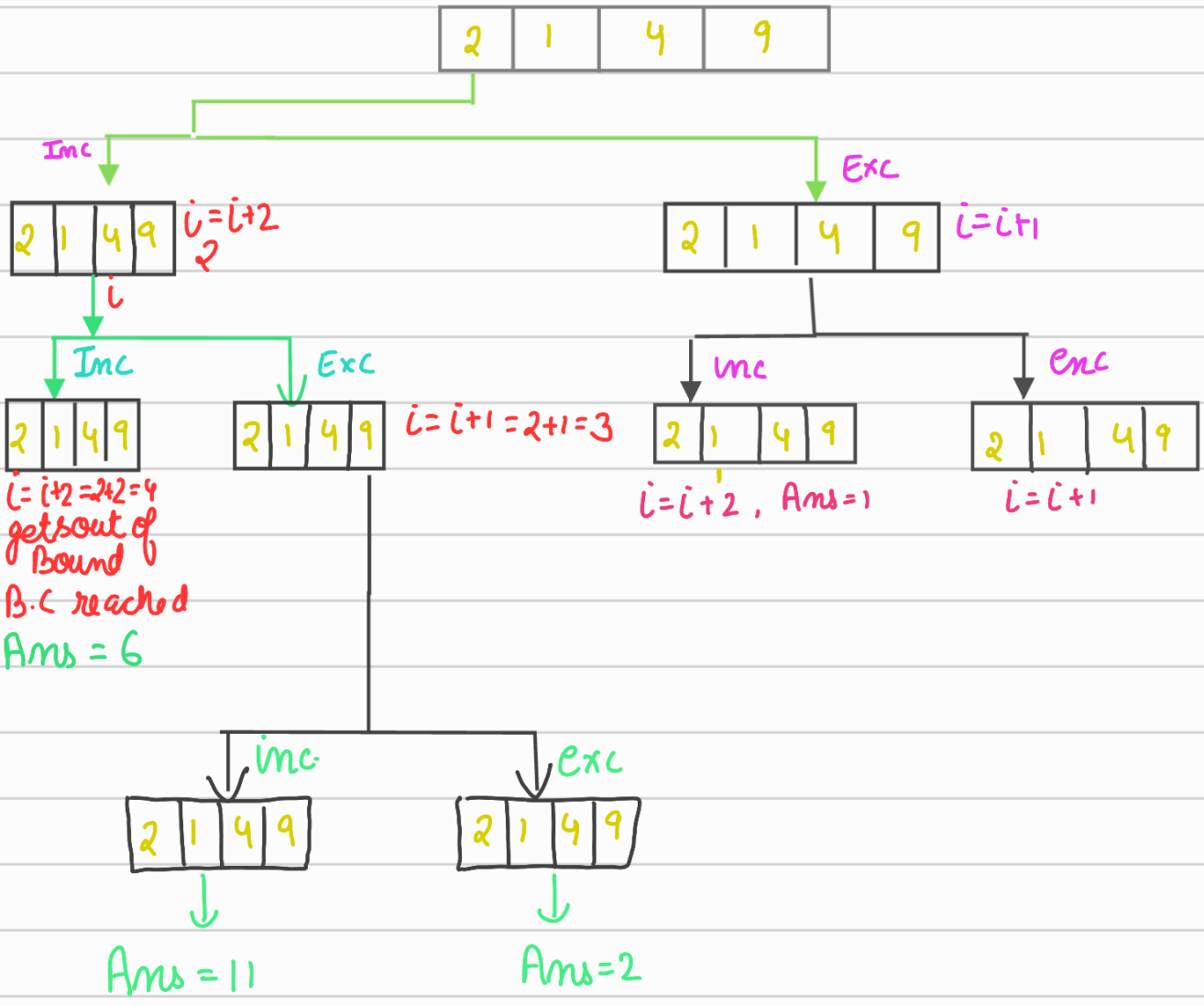
Problem Statement → Max sum of non-adjacent elements

I/p →

Return the max sum of subsequence in which no two elements are adjacent.

O/p → 2+9 = 11

Note → This is an example of inclusion-exclusion pattern problem




```cpp
#include <iostream>
#include<vector>
#include<limits.h>
using namespace std;

int solve(vector<int>& arr, int i, int sum, int &maxi){
  //base case
  if( i >= arr.size()){
    //maxi update
    maxi = max(sum, maxi);
    return 0;
  }
  //include
  solve(arr, i+2, sum+arr[i], maxi);
  //exclude
  solve(arr, i+1, sum, maxi);

}

int main() {
  vector<int> arr{1,2,3,4,5};
  int sum=0;
  int maxi = INT_MIN;
  int i =0;
  solve(arr,i,sum,maxi);

  cout << maxi << endl;

  return 0;
}
```