Combination Sum

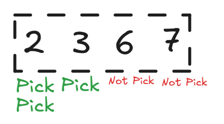
- We are given an array {2, 3, 6, 7} and we need to print all the possible combinations required to reach a target 7 .
- The main catch here is that element can be repeated any number of times, we want all possible combinations.
- answer would be [2,2,3] and [7].

```
#include<bits/stdc++.h>
using namespace std;

void f( int index, vector<int> ds, int target, vector<int> arr){

    // base case

    if( target < 0 ){
        return ;
    }

    if( index == arr.size() ){
        if(target == 0){
            for( auto i : ds){
                cout << i << " ";
            }
            cout << endl;
        }
        return;
    }

    // picking
    ds.push_back(arr[index]);
    f( index, ds, target - arr[index], arr );

    // coming back
    ds.pop_back();
    f(index + 1, ds, target, arr );

}
int main(){

    vector<int> arr = {2,3,6,7};
    int target = 7;
    vector<int> ds;
    f(0, ds, target, arr);

    return 0;

}
```

- The logic is simply that you have to implement pick and not pick .

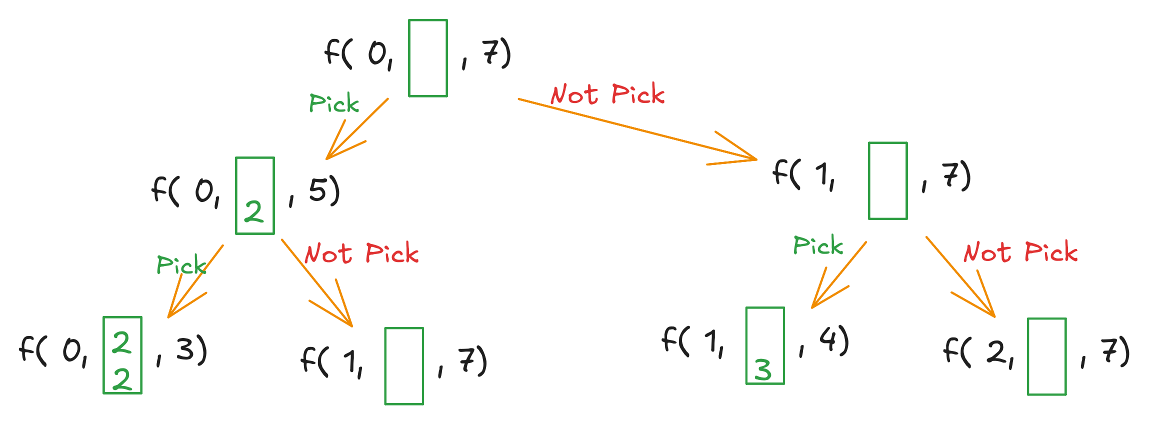
- { 2 , 3 , 6 , 7 }

- For the answer of { 2 , 2, 3 }, we need to do

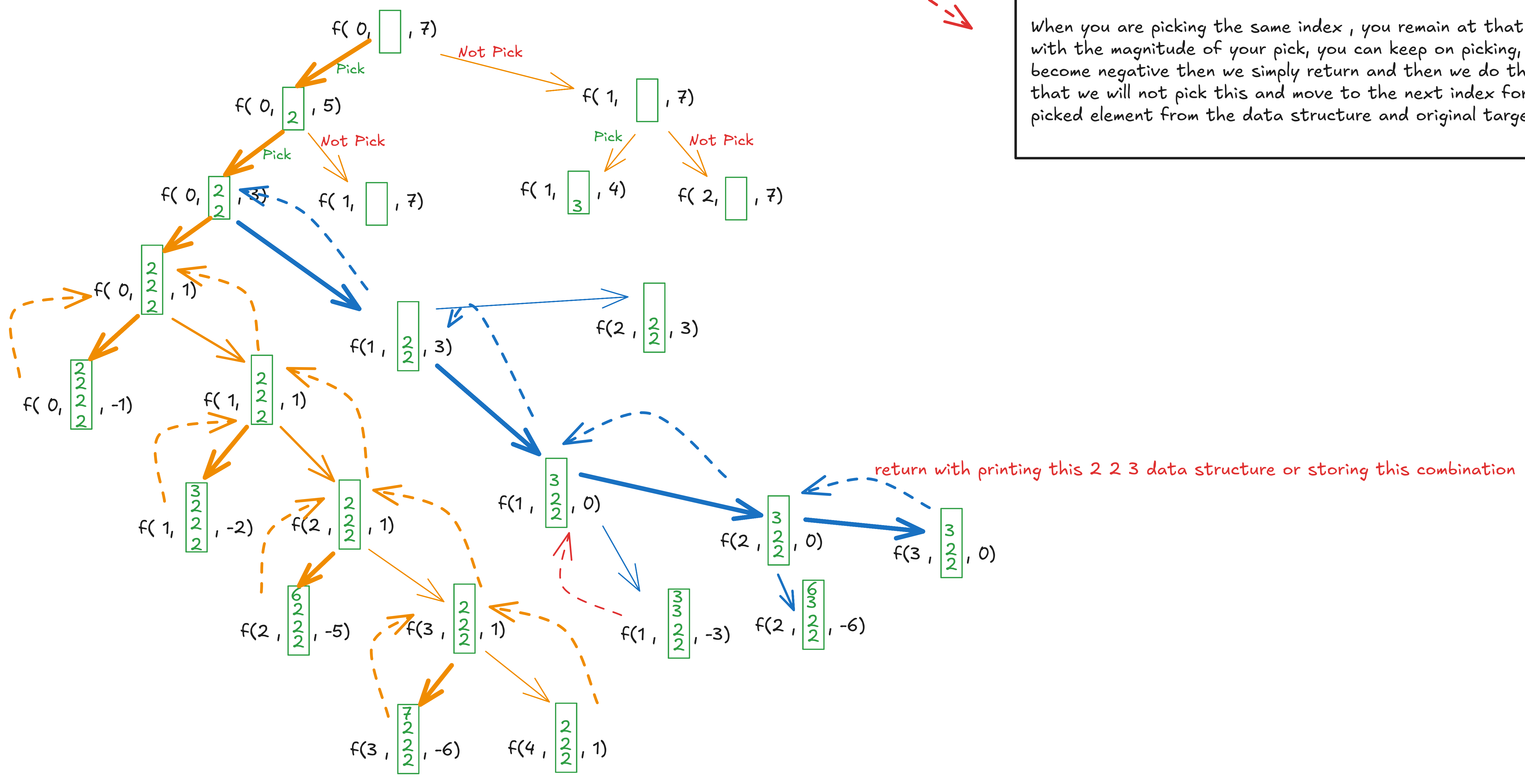


- We need to think of how to incorporate the multiple pick.

f(index, [ ] , target) : This is the basic building block of our logic which has an index, data structure to put in the values and the target which changes after inclusion or exclusion of elements in the data structure which make the sum.



When you are picking the same index , you remain at that index only , you just reduce the target with the magnitude of your pick, you can keep on picking, until target becomes negative, if it has become negative then we simply return and then we do the not pick part, not pick simply means that we will not pick this and move to the next index for which you add the line of removing the picked element from the data structure and original target.



f(0, [ ] , 7)
Pick
Not Pick
f(1, [ ] , 7)
f(0, [2] , 5)
Pick
Not Pick
Pick
Not Pick
f(1, [3] , 4)
f(2, [ ] , 7)
f(0, [2 2] , 3)
f(1, [ ] , 7)
f(0, [2 2 2] , 1)
f(0, [2 2 2 2] , -1)
f(1, [2 2 2] , 1)
f(1 , [2 2] , 3)
f(2 , [2 2] , 3)
f(1, [3 2 2 2] , -2)
f(2 , [2 2 2] , 1)
f(2 , [6 2 2 2] , -5)
f(3 , [2 2 2] , 1)
f(3 , [7 2 2 2] , -6)
f(4 , [2 2 2] , 1)
f(1 , [3 2 2] , 0)
f(1 , [3 3 2 2] , -3)
f(2 , [3 2 2] , 0)
f(2 , [6 3 2 2] , -6)
f(3 , [3 2 2] , 0)
return with printing this 2 2 3 data structure or storing this combination