Permutations (arrangement)

amt : 5 coins -> 2, 3, 5

amt

coins

0	1	2	3	4	5
1	0	1	1	1	1+1+1
.		2.	3.	22.	32. 23. 5.

combinations (selection)

amt : 5 coins -> 2, 3, 5

coins

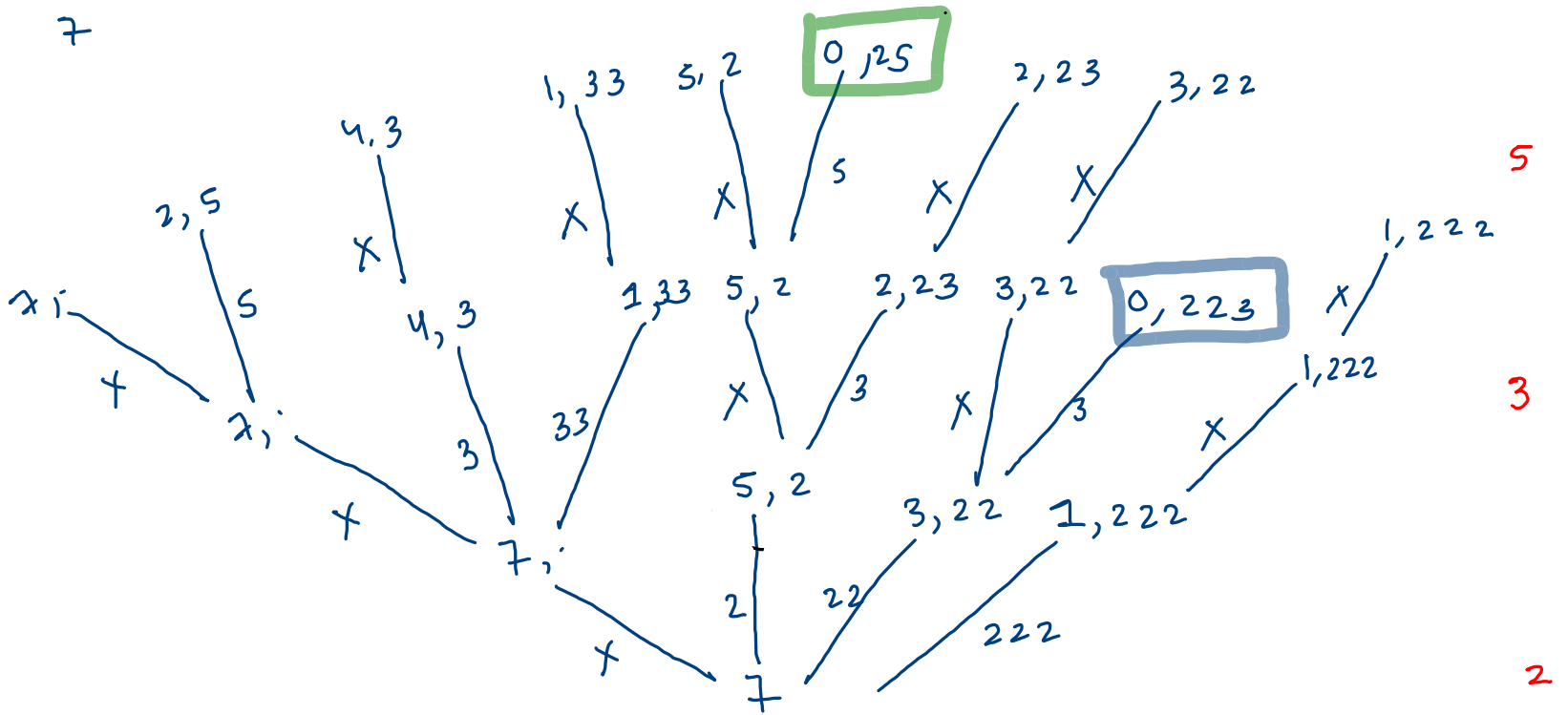
amt

0	1	2	3	4	5
1	0	1	1	1	1
.		2.	3.	22.	23. 5.

coins : 2, 3, 5

amt : 7

coin change comb



memoise → 2d qb

1st dimension → idx

2nd dimension → amt

amt : 7

coins : 3, 2, 5

coin change perm

amt

coins

1	0	1	1	1	3	2	5
0	1	2	3	4	5	6	7
.		2.	3.	22.	23. 32. 5.	33. 222.	223. 232. 322. 52. 25.

amt : 7

coins : 3, 2, 5

coin change comb.

1		1	1	1	2	2	1+1
0	1	2	3	4	5	6	7
		2.	3.	22.	32.	33.	322.
					5.	222.	25.

coins

amt

0 - 1 Knapsack :

1. You are given a number n, representing the count of items.
2. You are given n numbers, representing the values of n items.
3. You are given n numbers, representing the weights of n items.
3. You are given a number "cap", which is the capacity of a bag you've.
4. You are required to calculate and print the maximum value that can be created in the bag without overflowing it's capacity.

Note -> Each item can be taken 0 or 1 number of times. You are not allowed to put the same item again and again.

```
5
15 14 10 45 30
2 5 1 3 4
7
```

2d → 1 dim : items

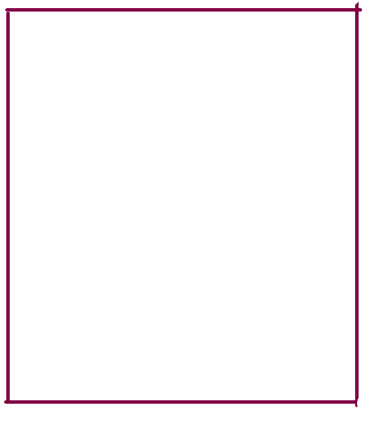
2 dim : target

	0	1	2	3	4
val	15	14	10	45	30
wt	2	5	1	3	4

cap → 7kg

2 - 15	2 - 15	3 - 45
4 - 30	5 - 14	4 - 30
1 - 10		

2^5



(i) non-breakable items

(ii) repeatation of items is not allowed

target sum subset

	0	1	2	3	4
val	15	14	10	45	30
wt	2	5	1	3	4

cap = 7 kg

$(val-wt)_i$	0	1	2	3	4	5	6	7
$(15-2)_0$	0	0	15	15	15	15	15	15
$(14-5)_1$	0	0	15	15	15	15	15	29
$(10-1)_2$	0	10	15	25	25	25	25	29
$(45-3)_3$	0	10	15	45	55	60	70	70
$(30-4)_4$	0	10	15	45	55	60	70	75

max (dp[i-1][j] , dp[i-1][j-wt[i]] + val[i])

exc

inc

dp[i][j]

dp[i][j] → using items [0, i] max profit generated if cap is j.

	0	1	2	3	4
val	15	14	10	45	30
wt	2	5	1	3	4

cap = 7

```java
public static int zero_one_ks(int[]val,int[]wt,int cap) {
    int n = val.length;
    int[][]dp = new int[n][cap+1];

    for(int i=0; i < dp.length;i++) {
        for(int j=0; j < dp[0].length;j++) {
            if(j == 0) {
                //cap is 0
                dp[i][j] = 0;
            }
            else if(i == 0) {
                //only one element
                if(j >= wt[i]) {
                    dp[i][j] = val[i];
                }
            }
            else {
                int exc = dp[i-1][j];
                int inc = 0;

                if(j - wt[i] >= 0) {
                    inc = dp[i-1][j-wt[i]] + val[i];
                }

                dp[i][j] = Math.max(inc,exc);
            }
        }
    }

    return dp[dp.length-1][dp[0].length-1];
}
```

	0	1	2	3	4	5	6	7
$(15-2)_0$	0	0	15	15	15	15	15	15
$(14-5)_1$	0	0	15	15	15	15	15	29
$(10-1)_2$	0	10	15	25	25	25	25	29
$(45-3)_3$	0	10	15	45	55	60	70	70
$(30-4)_4$	0	10	15	45	55	60	70	75

unbounded knapsack :

```
5
15 14 10 45 30
2 5 1 3 4
7
```

ans = 100

cap = 7

(i) item unbreakable

(ii) repeatation of item is allowed.

c c p

c c c

doesn't matter

```
5
15 14 10 45 30
2 5 1 3 4
7
```

	0	1	2	3	4
val	15	14	10	45	30
wt	2	5	1	3	4

items

cap

Coin change comb.

0	10	20	45	55	65	90	100
0	1	2	3	4	5	6	7
	1	~~2~~ 11	~~2~~ ~~111~~ 3	~~22~~ ~~1111~~ 13	~~22~~ ~~11111~~ 113	~~222~~ ~~111111~~ 33	~~222~~ ~~1111111~~ 133

Coin change perm

	0	1	2	3	4
val	15	14	10	45	30
wt	2	5	1	3	4

cap

items

65+15 20+14 90+10 55+45 45+30

2 5 1 3 4

0	10	20	45	55	65	90	100
0	1	2	3	4	5	6	7
	1	11	3.	31	311	33.	331

	0	1	2	3	4
val	15	14	10	45	30
wt	2	5	1	3	4

```java
public static int unbounded_ks(int[]val,int[]wt,int cap) {
    int[]dp = new int[cap+1];

    //dp[i] -> max profit generated if cap is 'i'
    dp[0] = 0;

    for(int i = 1; i < dp.length;i++) {
        int max = 0;
        for(int j=0; j < val.length;j++) {
            int rem_cap = i - wt[j];

            if(rem_cap >= 0) {
                int profit = dp[rem_cap] + val[j];

                if(profit > max) {
                    max = profit;
                }
            }
        }

        dp[i] = max;
    }

    return dp[cap];

}
```

131

0	10	20	45	55	65		
0	1	2	3	4	5	6	7