August 17, 2025

RECURSION DRILL

* Kth Finding

Ques 1. For Tower of Hanoi, find Kth move recursively. (without generating everything)

Number of moves needed to move 'x' disks :

$$f(x) = f(x-1) + 1 + f(x-1)$$

This is Recursive Relation

- $f(x-1)$ → Move x-1 disk to 3rd
- 1 → Move the largest to 2nd
- $f(x-1)$ → Move x-1 disk from 2 → 3

$$\boxed{f(x) = 2f(x-1) + 1}$$

$f(1)$ = 1 move $f(2)$ = 3 moves

$f(0)$ = 0 move $f(3)$ = 7 moves

Closed form formula (Mathematical) to find out no. of moves needed to move 'x' disks :

$$\boxed{f(x) = 2^x - 1}$$

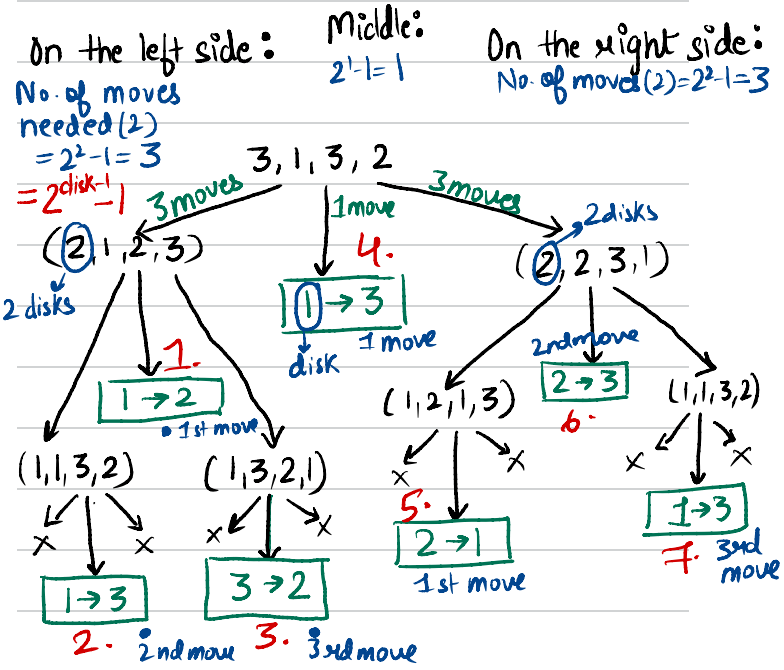
Putting this value in our recursive function

$$f(x) = 2(2^{x-1} - 1) + 1$$

$2^{x-1} = 2^{x-1}$ indicates that $f(x) = 2^x - 1$ is definitely one of the solution.

→ Runthrough using Recursion Tree for previous Tower of Hanoi example: Moving 3 disk from 1 to 3 using 2

move(3, 1, 3, 2)



Looking at the tree, let's say we are finding 5th move. Then where shall our 5th move lie? left (3 moves) + Middle (1 move) = 4

The answer is on the right side of the tree, and the first move on the right side will be our answer (5th move).

This is how we will generate Kth move without whole tree generation.

We will only go to the side of the tree where our K th lies.

```
void movedisk(int disks,int from,int to, int extra,int k){
    if(disks==0)return;
    if(k <= (1<<(disks-1))-1){ //check if exist on left side
        movedisk(disks-1,from,extra,to,k);
    }
    else if(k==(1<<(disks-1))){ //Middle
        cout<<"From "<<from<<" To "<<to<<endl;
    } // 2^(disk-1)-1
    else{ //exist in right
        movedisk(disks-1,extra,to,from,k-(1<<(disks-1)));
    }
}
```

Substract the left & Middle moves = Right

$1 << (disk-1) - 1 = 2^{disk-1} - 1$

Bit manipulation

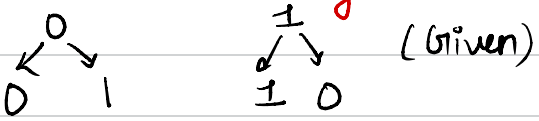
AlgoZenith

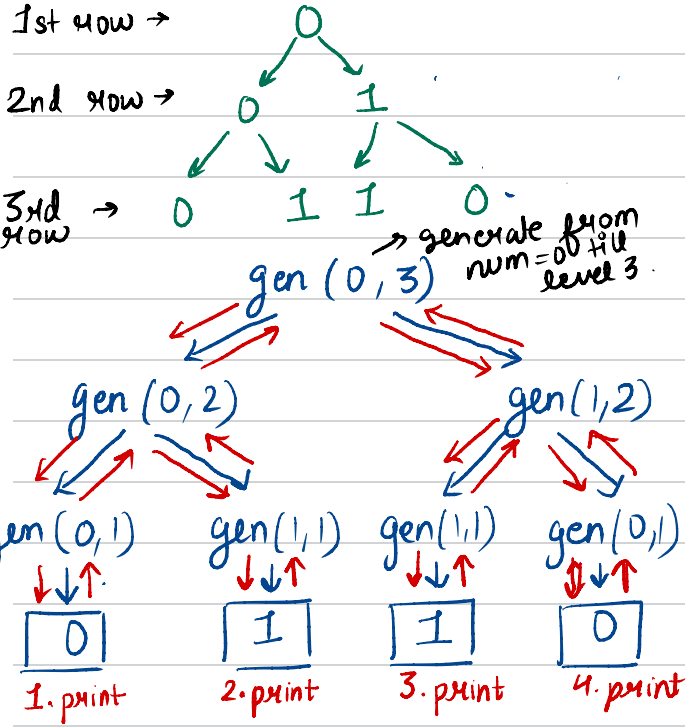
Ques 2. k-th symbol in Grammar

https://leetcode.com/problems/k-th-symbol-in-grammar/description/

> We build a table of `n` rows (**1-indexed**). We start by writing `0` in the `1st` row. Now in every subsequent row, we look at the previous row and replace each occurrence of `0` with `01`, and each occurrence of `1` with `10`.
>
> - For example, for `n = 3`, the `1st` row is `0`, the `2nd` row is `01`, and the `3rd` row is `0110`.
>
> Given two integer `n` and `k`, return the `kth` (**1-indexed**) symbol in the `nth` row of a table of `n` rows.

Firstly, identify Recurrence Relation that can be used to generate Nth row.





⇒ 0110 1st level

- gen (0, level)
  - → gen (0, level-1)
  - → gen (1, level-1)

Now, let us say that we are looking for K=3 (3rd element)

In the tree, there is only right side and left side. If we look at left, gen (0, 2), this tells us that it can expand to 2 elements. Similarly, right can also expand to only 2 elements.

So, for K=3 (3>2) The answer will lie on right side of the tree. Since 2 moves are already done on the left side, our answer will

= (K - leftmove)th move

= 3 - 2

= 1 st move of right side.

Ans = 1

Code for n-th row generation:

```
void generate(int cur,int level){
    if(level==1){
        cout<<cur;
        return;
    }

    generate(cur,level-1);
    generate(1-cur,level-1);
}

void solve(){
    generate(0,3);
}
```

This is what these lines are doing

```
if(cur==0){
    generate(0,level-1);
    generate(1,level-1);
}else{
    generate(1,level-1);
    generate(0,level-1);
}
```

- Code updated to accomodate kth move.

```
int pow2(int x){
    return round(pow(2,x));
}
void generate(int cur,int level,int k){
  // cout<<cur<<" "<<level<<" "<<k<<endl;
    if(level==1){
        cout<<cur;
        return;
    }//go towards left
    if(k <= pow2(level-2))
        generate(cur,level-1,k);
    else // right
        generate(1-cur,level-1,k-pow2(level-2));
}
```

No of moves at (level-1)

$= 2^{x-1} - 1$

$= 2^{lvl-1-1} - 1$

$= 2^{level-2} - 1$

Ques 3. Christmas

https://atcoder.jp/contests/abc115/tasks/abc115_d

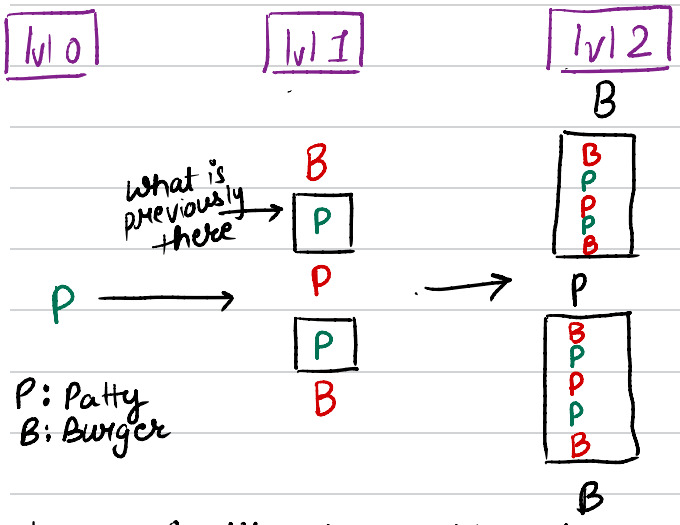
In some other world, today is Christmas.

Mr. Takaha decides to make a multi-dimensional burger in his party. A *level-L burger* (L is an integer greater than or equal to 0) is the following thing:

- A level-0 burger is a patty.
- A level-L burger ($L \geq 1$) is a bun, a level-$(L-1)$ burger, a patty, another level-$(L-1)$ burger and another bun, stacked vertically in this order from the bottom.

For example, a level-1 burger and a level-2 burger look like BPPPB and BBPPPBPBPPPBB (rotated 90 degrees), where B and P stands for a bun and a patty.

The burger Mr. Takaha will make is a level-N burger. Lunlun the Dachshund will eat X layers from the bottom of this burger (a layer is a patty or a bun). How many patties will she eat?



→ Let us simplify the problem by looking at no of P & B at each level

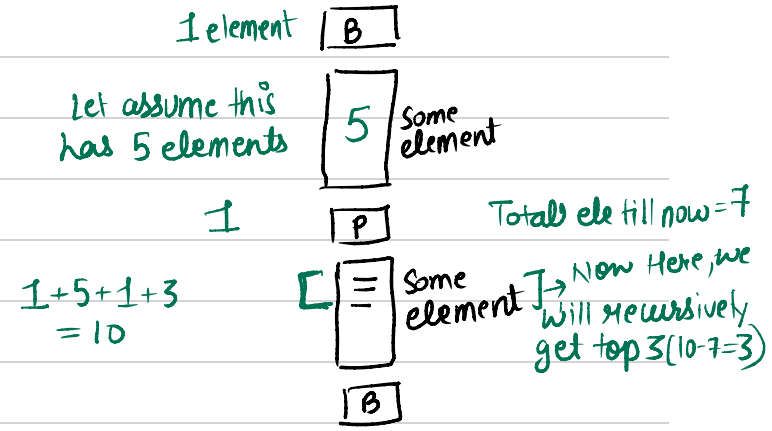
	level 0	level 1	level 2
B	0	2	4 (2×2+2 BPB)
P	1	3	7 (2×3+1 BPB)

Here, previous val × 2 (occuring 2 times) + B + P + B, so this B will be 2 from BPB and Patty 3 as (2×1 + 1(BPB)) = 3.

So, we can pre compute the above values at levels.

→ Now, what if we want Top x at a level

Let us say x = 10



Here, we are not recursing entirely but only partially as above elements count can be easily pre computed like shown.

```
#define int long long
#define F first
#define S second

pair<int,int> fullcount[51];
void pre(){
    fullcount[0] = {0,1};
    for(int i=1;i<=50;i++){
        fullcount[i] = fullcount[i-1];
        fullcount[i].F *= 2;
        fullcount[i].S *= 2;
        fullcount[i].F += 2;
        fullcount[i].S += 1;
        cout<<fullcount[i].F<<" "<<fullcount[i].
        S<<endl;
    }
}
pair<int,int> getcnt(int n,int x){
    if(x==0)return {0,0};
    if(n==0)return {0,1};

    pair<int,int> ans = {0,0};
    // B
    if(x>=1){ans.F+=1;x-=1;}
    else return ans;
    // L-1 Burget
    if(x>=fullcount[n-1].F+fullcount[n-1].S){
        ans.F+=fullcount[n-1].F;
        ans.S+=fullcount[n-1].S;
        x-=fullcount[n-1].F+fullcount[n-1].S;
    }else{
        auto temp = getcnt(n-1,x);
        ans.F+=temp.F;
        ans.S+=temp.S;
        return ans;
    }
    // P
    if(x>=1){ans.S+=1;x-=1;}
    else return ans;
    // L-1 Level Burget
    if(x>=fullcount[n-1].F+fullcount[n-1].S){
        ans.F+=fullcount[n-1].F;
        ans.S+=fullcount[n-1].S;
        x-=fullcount[n-1].F+fullcount[n-1].S;
    }else{
        auto temp = getcnt(n-1,x);
        ans.F+=temp.F;
        ans.S+=temp.S;
        return ans;
    }
    // B
    if(x>=1){ans.F+=1;x-=1;}
    else return ans;

    return ans;
}
```