Digit DP

→ [It's not as scary as it sounds]

- Gaurish Baliga

Let's consider a problem to understand

Find all integers from 0 to 10^{18} that they have the sum of digits = 150

Can you do a brute force here?

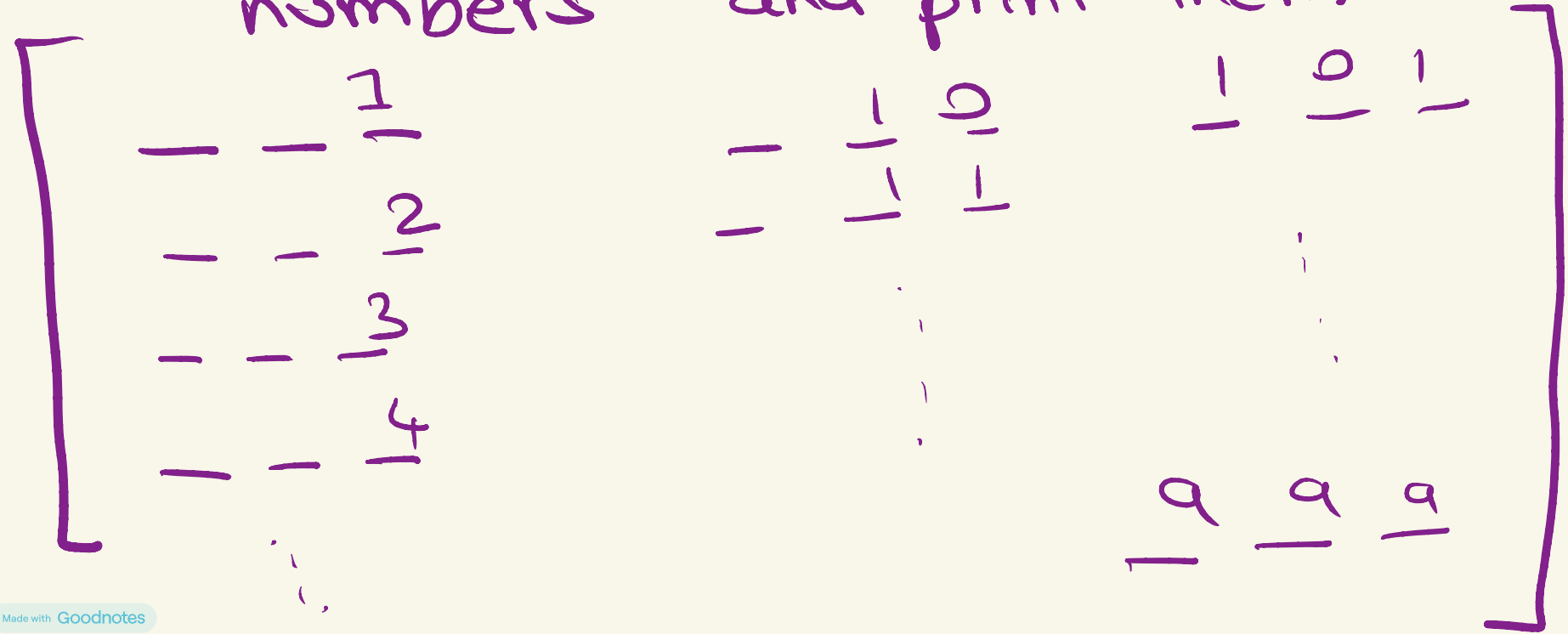
specific property

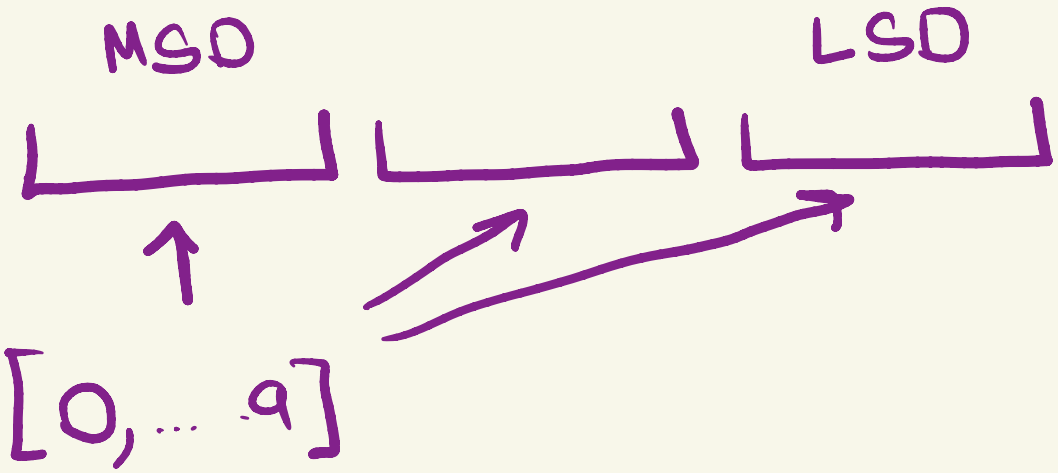
? L → R TLE

→ combinatorial → complicated → DP

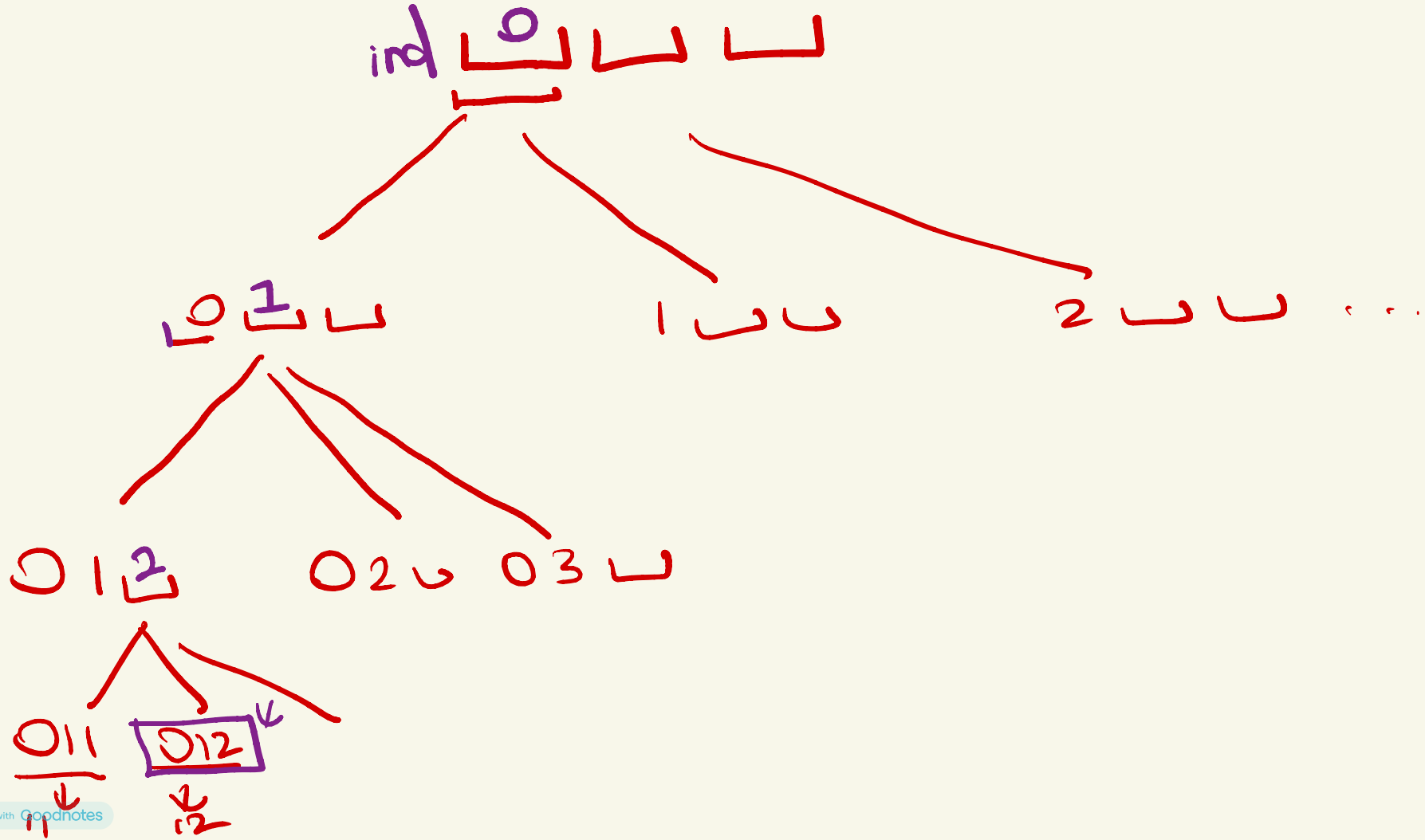
↳ Digit DP

1. Digit by digit, recursively generate all 1, 2 & 3 digit numbers and print them





0 9 8 ⟶ 98



Print the amount of numbers with $\leq d$ digits

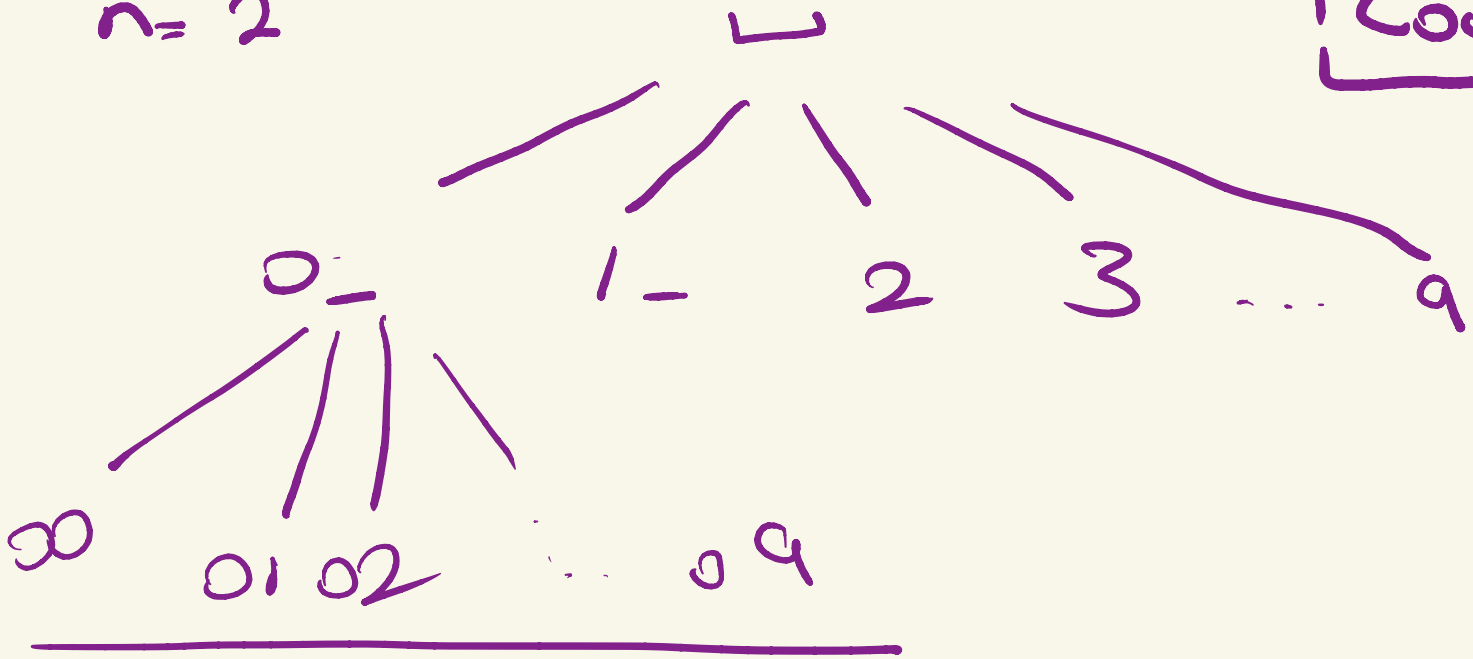
$3 \rightarrow 1000$

$[0, 999]$

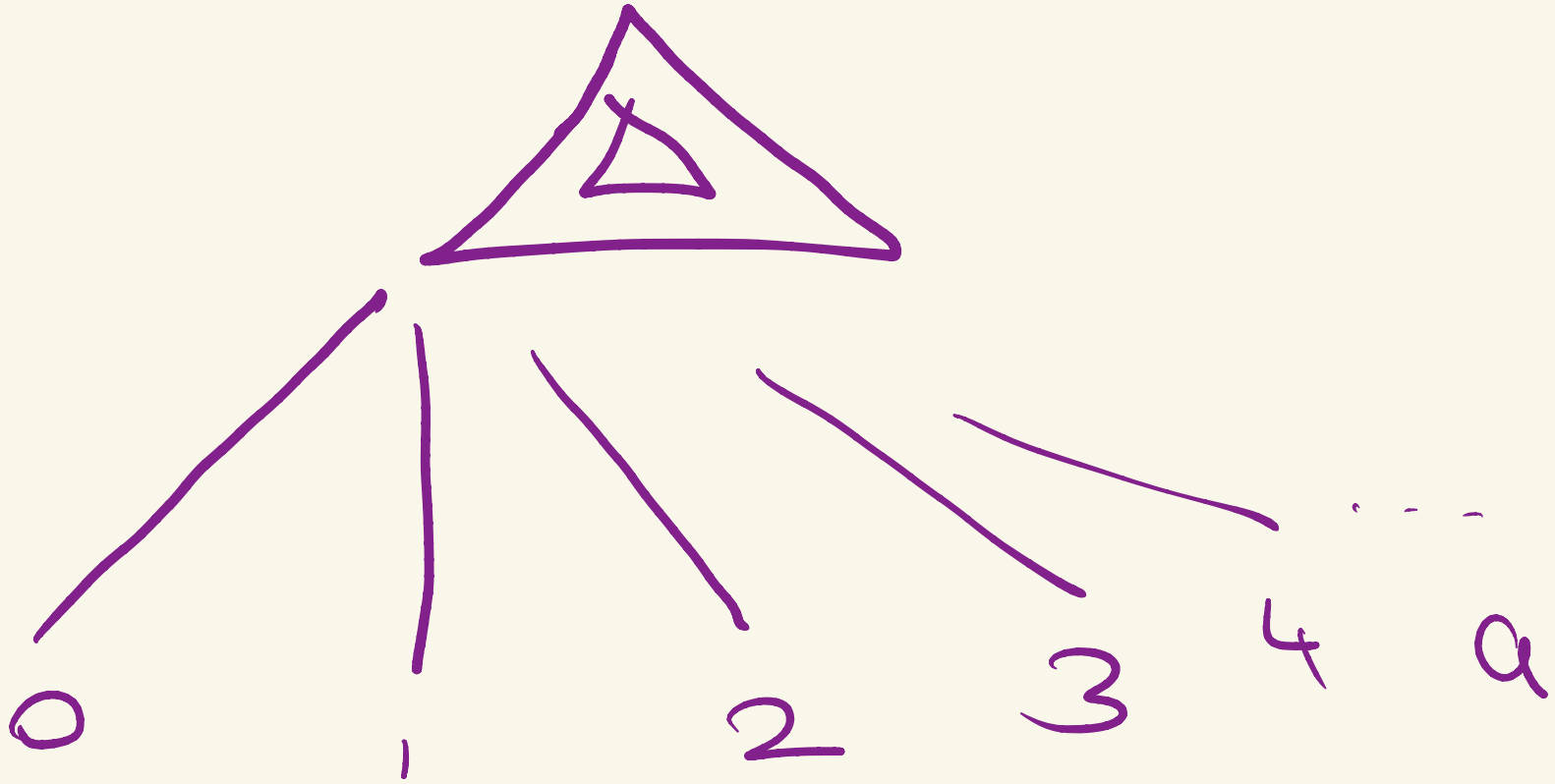
$2 \rightarrow 100$

n = 2

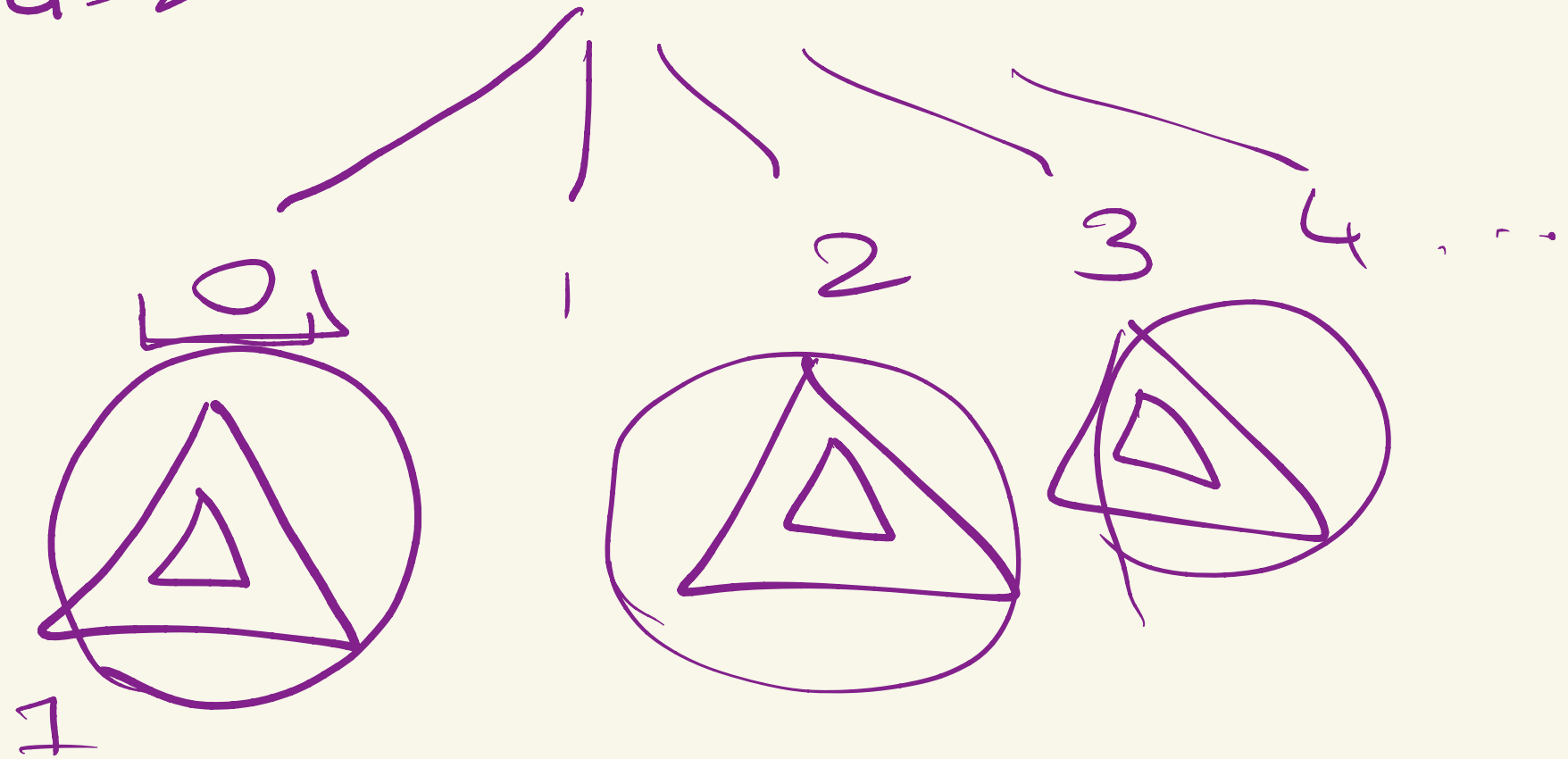
Count



d = 1



d = 2



memoise ?? → Yes

Initial TC → $O(10^9)$

memoised TC → $O(d \cdot b)$

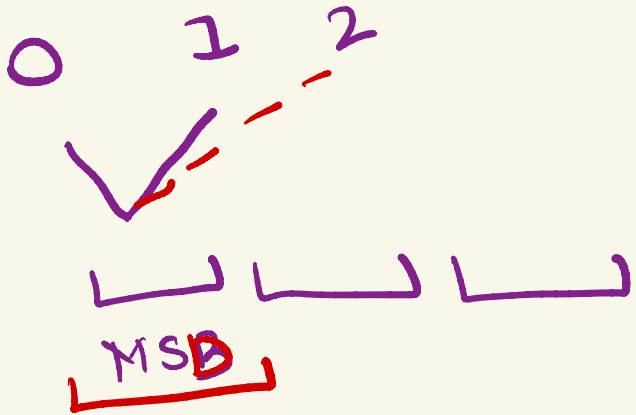
→ ≤ 10,000 digits

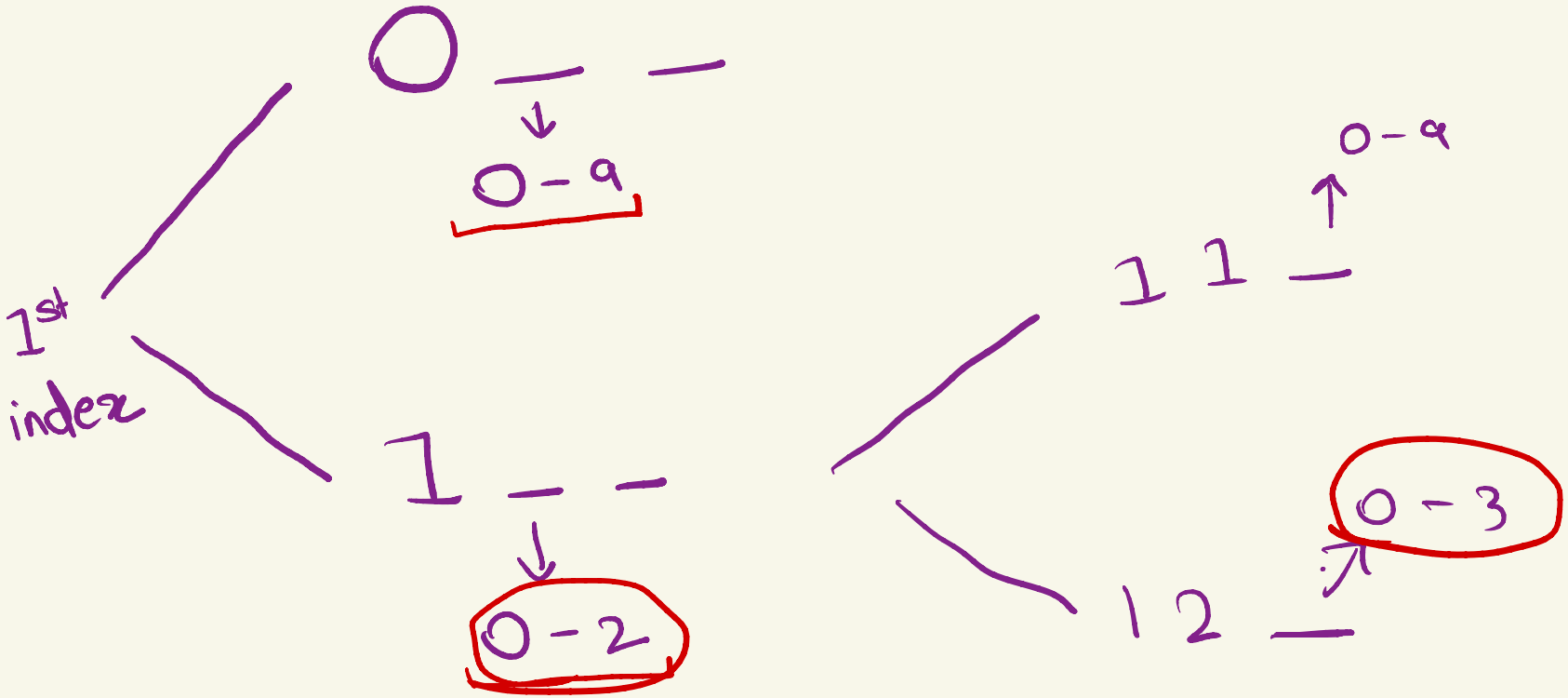
dp[10,000]

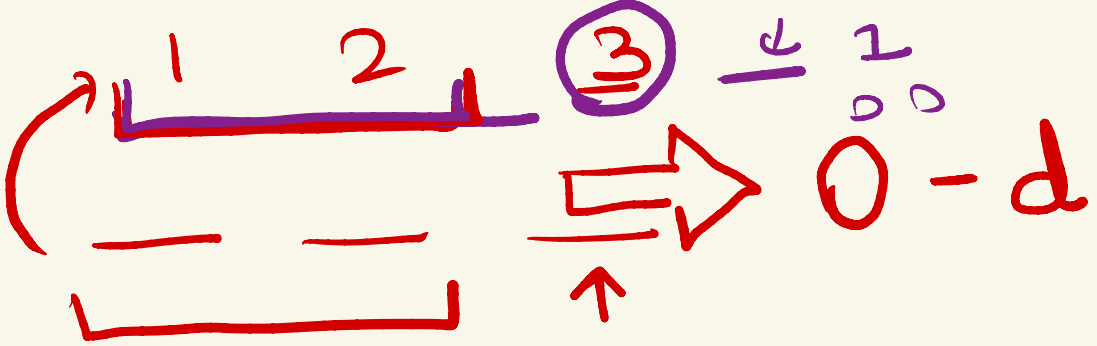
≈ 10,000 × 10
indices 0-9

1, 0 . . . . . .
10,000 zeroes

* Using recursion, print all numbers from 0 - 123

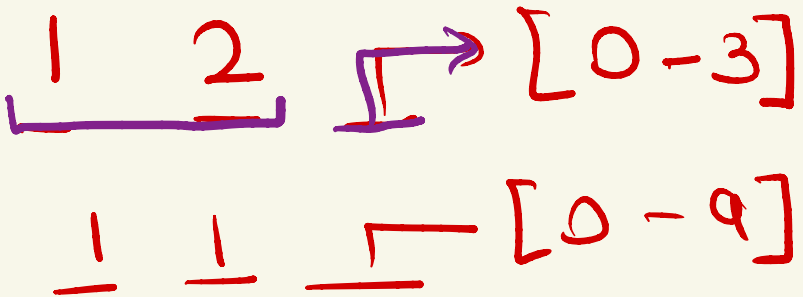






i

tight → true
↘ false

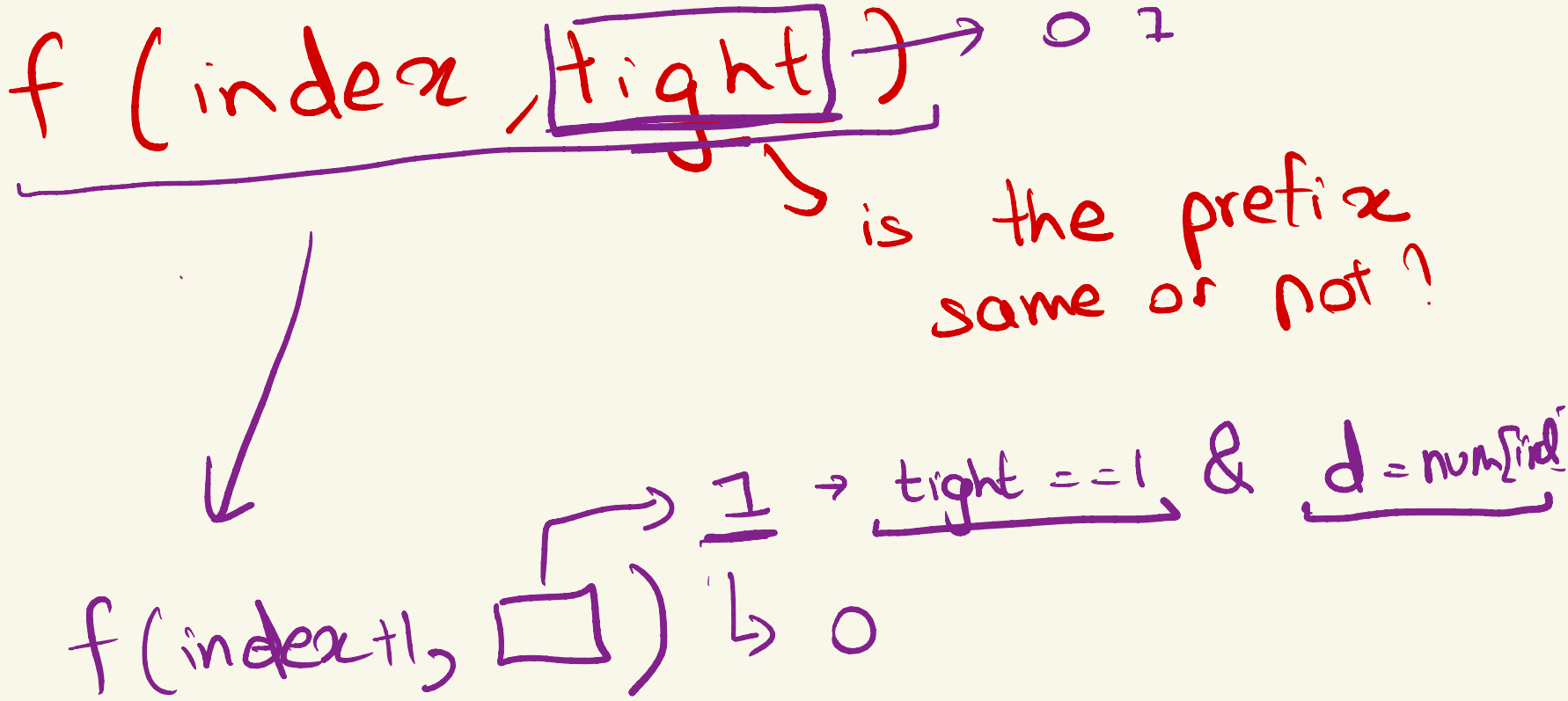


1 2 1 3 4 5 7

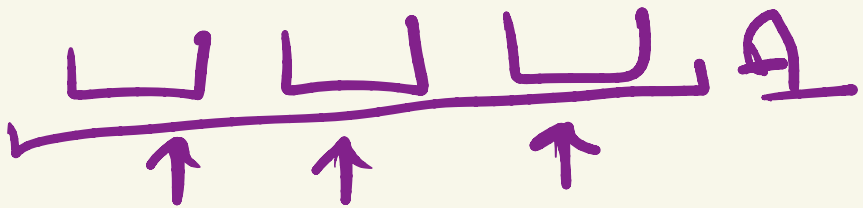
1 0 1 3 4 _

↓

[0-9]



1 2 3



Time Comp. : $O(\text{numdig} \cdot 10 \times 2)$

iterations

tight

$10 \times 10 \times 2 = 200$

memoisation ✓

10^9

$\rightarrow O(\text{numdigits})$

✗ memoise $\rightarrow$ $O(\text{num})$

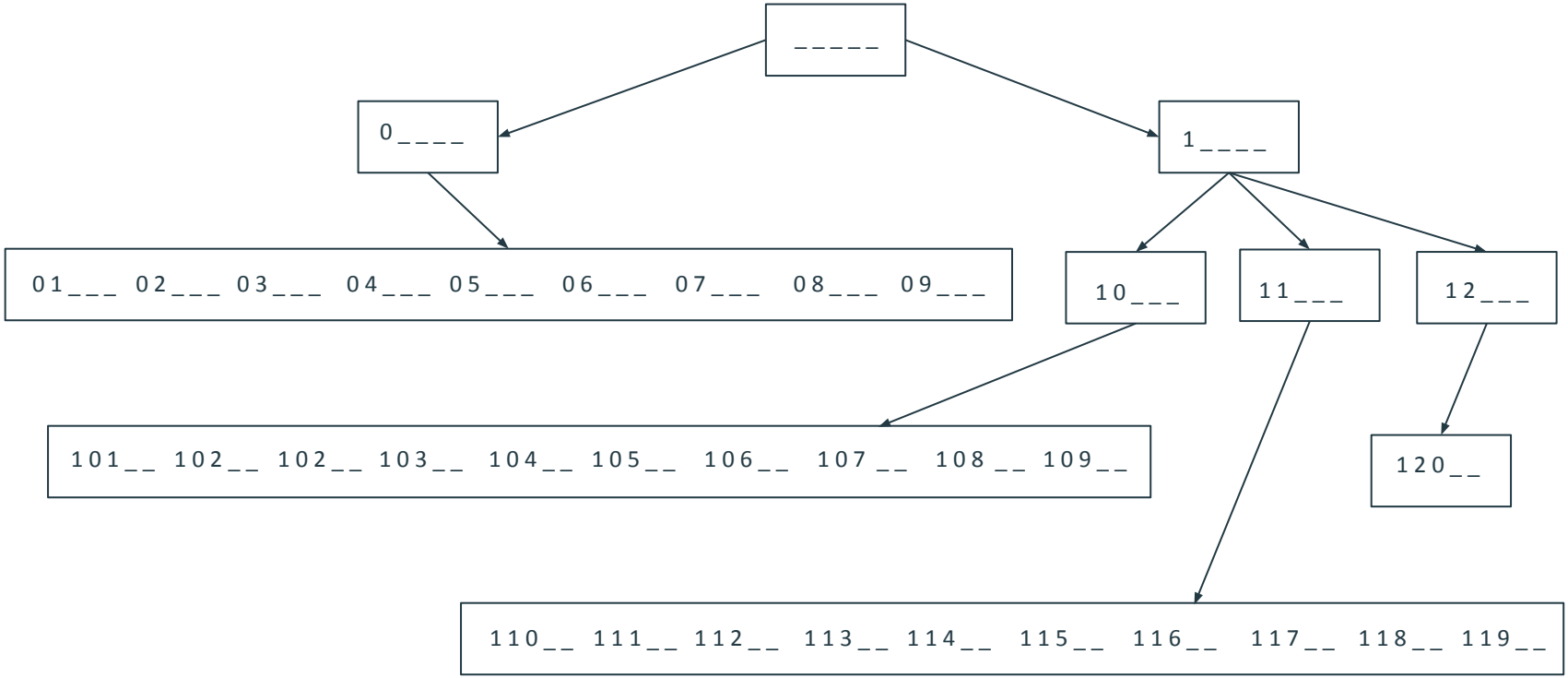
10^9

Use of Digit DP

Digit DP is a used to solve problems that asks you to find the number of integers within a range that satisfies some property based on the digits of the integers.

Typically, the ranges are between large integers (such as between 1 to 10^{18}), so looping through each integer and checking if it satisfies the given property is too slow.

Constructing a number <= 12025 recursively



Some observations

When we are trying to construct a number from the most significant digit to the least significant digit which is <= UpperLimit (UL)

- If UL = 10987 and the number constructed so far is 109_ _, then the 4th digit of the number must be <= the 4th digit in the UL.
- If UL = 10987 and the number constructed so far is 108 _ _, then the 4th digit can be anything from 0 to 9 as it won't make the number exceed UL

Ensuring the i^{th} digit keeps number <= UL

Case 1: The digits entered till $i-1^{th}$ are same as the digits in UL

- i^{th} digit can be from 0 to i^{th} digit in UL

Case 2: The digits entered in $i-1^{th}$ are not same as UL

- I^{th} digit can be anything from 0 to 9

How to track the 2 cases? Storing another variable - **tight**

- tight = 1 => prefix of UL matches created number
- tight = 2 => prefix of UL is bigger than the created number

→ [count numbers $\leq 10^{18}$ with sum of digits = S (iso)]

$[0, 10^{18}]$ → sum of digits = S

$[0, 18 \times 9]$

$[0, 162]$

Solving the same problem

Find all integers from 0 to 10^{18} that they have the sum of digits = 150

Can we recursively generate all possible numbers by keeping track of 3 variables

- **i** = current digit
- **tight** =whether prefix is same or not
- **sum** = current sum of digits

Brute Force

```cpp
int n, k;
int validNumbers(int i, int tight, int sum, string UL){
    if(i == n){ // created a number <= UL
        if(sum == k)
            return 1;
        return 0;
    }

    int highestDigit = tight ? (UL[i] - '0') : 9;
    long long ans = 0;
    for(int currDigit = 0; currDigit <= highestDigit; currDigit++){
        int newSum = sum + currDigit;
        int newTight = (tight && (currDigit == highestDigit)) ? 1 : 0;
        ans += validNumbers(i + 1, newTight, newSum, UL);
    }
    return ans;
}
void solve() {
    long long UL;
    cin >> UL >> k;
    string UL_string = to_string(UL);
    n = UL_string.size();
    cout << validNumbers(0, 1, 0, UL_string) << endl;
}
```

Overlapping Subproblems!!!

Suppose UpperLimit = 66245

091_ _ _ - i = 2, tight = 0, sum = 10

019_ _ _ - i = 2, tight = 0, sum = 10

082_ _ _ - i = 2, tight = 0, sum = 10

505_ _ _ - i = 2, tight = 0, sum = 10

Dynamic Programming?

Digit DP Solution

State: Dp[i][tight][sum]

Number of valid ways to fill the digits from i to n - 1 such that the final number <= UpperLimit and the sum of digits = K

Transition: Dp[i][tight][sum] = Sum(Dp[i + 1][newTight][sum + currD])

currD goes from 0 to highestAllowedD for i^{th} digit

newTight = 1 when tight = 1 && currD = highestAllowedD, 0 otherwise

Digit DP Solution

Base Case:

- Dp[n][anything][K] = 1
- Dp[n][anything][anything but not K] = 0

Final Subproblem: Dp[0][1][0], because we are looking for valid ways to fill digits from 0 to n-1 such that the first digit is constrained to be <= UL[0] and the current sum = 0.

Digit DP code

```cpp
int n, k;
vector<vector<vector<int>>> dp(20, vector<vector<int>>(2, vector<int>(200, -1)));
// 20 because 1e18 has maximum 19 digits, 200 because given K (150) <= 150

int validNumbers(int i, int tight, int sum, string UL){
    if(sum > k)
        return 0;

    if(i == n){ // created a number <= UL
        if(sum == k)
            return 1;
        return 0;
    }

    if(dp[i][tight][sum] != -1)
        return -1;

    int highestDigit = tight ? (UL[i] - '0') : 9;
    long long ans = 0;
    for(int currDigit = 0; currDigit <= highestDigit; currDigit++){
        int newSum = sum + currDigit;
        int newTight = (tight && (currDigit == highestDigit)) ? 1 : 0;
        ans += validNumbers(i + 1, newTight, newSum, UL);
    }
    return dp[i][tight][sum] = ans;
}
```

Things to consider

- Time complexity?
- Space?
- Why is recursive code better than iterative here?

Things to consider

- Time complexity
- Space complexity
- Why is recursive DP better than iterative here

Problem 1

1 - K → d

Digit Sum: https://atcoder.jp/contests/dp/tasks/dp_s

Find how many numbers from 1 to R (1 <= 1, R <= 10^{18}) have the sum of digits divisible by K

dig ≤ 10,000

sum ≤ 9 × 10,000

dig × sum

↳ 9e8

$d \leq 100$

$\Rightarrow$ $\underline{sum}\ \%d = 0$

$\rightarrow (a+b)\ \%\ c = (a\ \%\ c + b\ \%\ c)\ \%\ c$

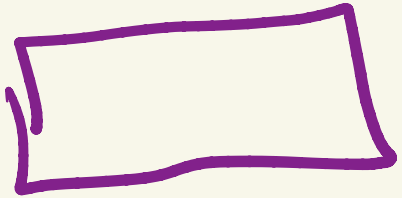
??

$= dp[index][tight][sum\ \%\ d]$

dp[i, tight, sum%d]

i → digit i

sum → sum we have made yet

dp[0, 1, 0] → 

d = 3

1 2 3 4 5 → (1+2+3+4+5) % 3

0

1 → (1) % 3

12 → (1+2) % 3

123 → (0) % 3

1234 → (1) % 3

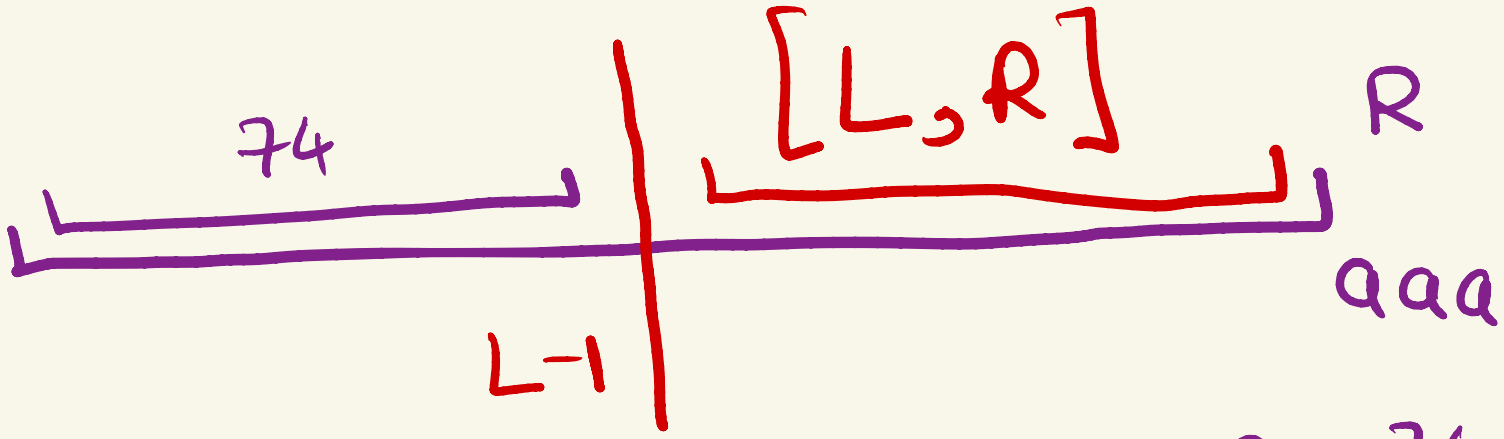
12345 → (1+0) % 3

```
int dp[Ksize][2][D];

int f(int index, int tight, int sum) {
    if(index == s.size()) {
        if(sum == 0) return 1;
        return 0;
    }

    if(dp[index][tight][sum] != -1) return dp[index][tight][sum];

    int bound = (tight ? s[index] - '0' : 9);

    int answer = 0;

    for(int i = 0; i <= bound; i++) {
        answer = (answer + f(index + 1, (tight & (s[index] - '0' == i)), (sum + i) % d)) % MOD;
    }

    return dp[index][tight][sum] = answer;
};
```

1 / 0 - K

L - R

$1 \leq L \leq R \leq 10^{10,000}$



Prefix Sum [L, R]

count(R) - count(L-1)

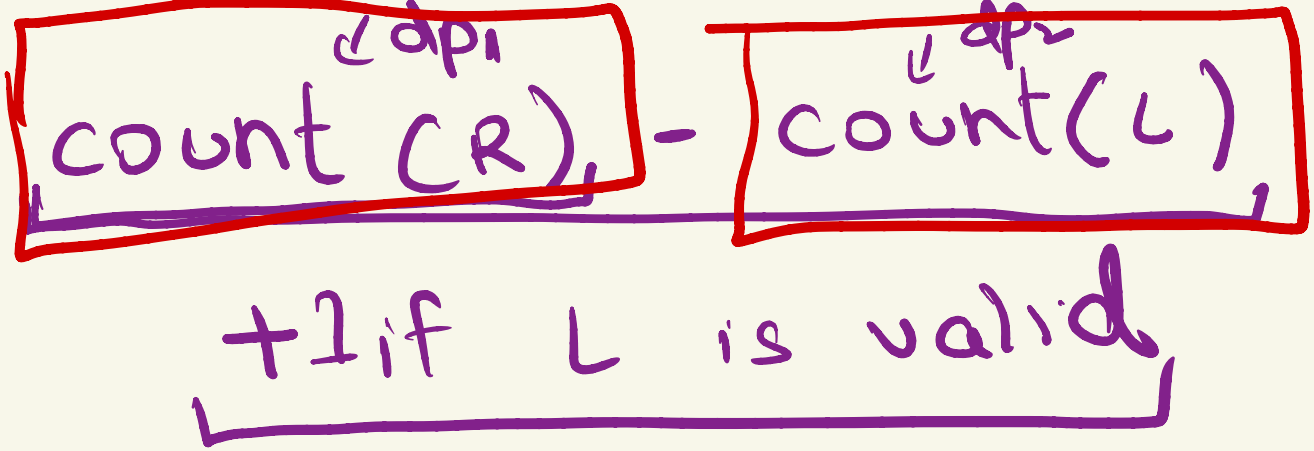
9 7 8 6 4 3 8 9 9 9

9 7 8 6 4 3 9 0 0 0 0

!!

L-1 ??

Hectic



L
1 → L is valid
0 → L isn't valid

* 2 dp vectors → L
→ R

* 1 check function → L

* 2 function calls

[L, R] → c(R) − c(L) + 1 → L is valid

Problem 2

Homework

Next class

Find how many numbers from L to R ($1 <= L, R <= 10^{18}$) have odd digits at odd indices and even digits at even indices