* Find No. of digits :

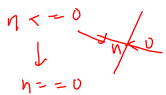
$1 <= n <= 10^9$

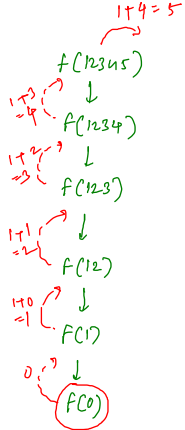
ip: 1 2 3 4 5

op: 5

```
function numDigits(n) {
  if(n == 0) return 0;
  const smallAns = numDigits(n/10);
  return 1 + smallAns;
}
```

* parseInt(n/10)





1. faith

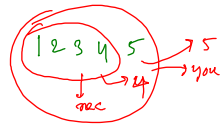
⇒ numDigits(n)

⇒ will give me no. of digits in n.

2. logic

⇒ numDigits(12345) ⇒ 5

⇒ numDigits(1234) ⇒ 4

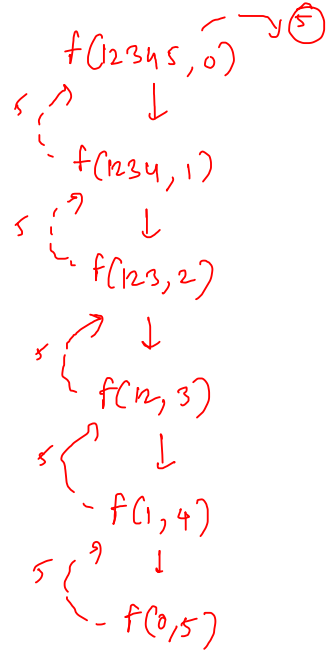


⇒ rec + 1

preorder :

```
function numDigits(n, cnt) {
    if(n==0) return cnt;
    return numDigits(n/10, cnt+1);
}
```

parseInt(n/10)



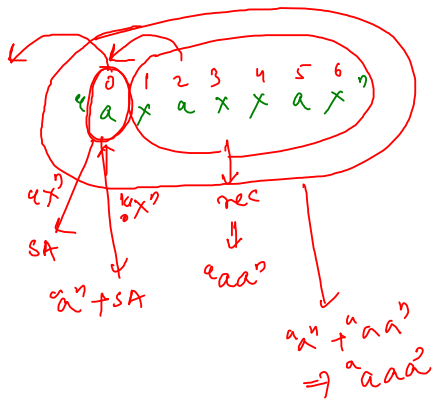
★ remove all 'x's :

"x" — do not consider

"!x" — consider

ip: "axaxxax"

op: "aaa"



1. faith

⇒ removeX(str, idx)

⇒ give me a string without 'x' in idx → n.

2. logic

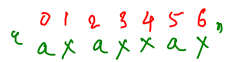
⇒ removeX(str, 0)

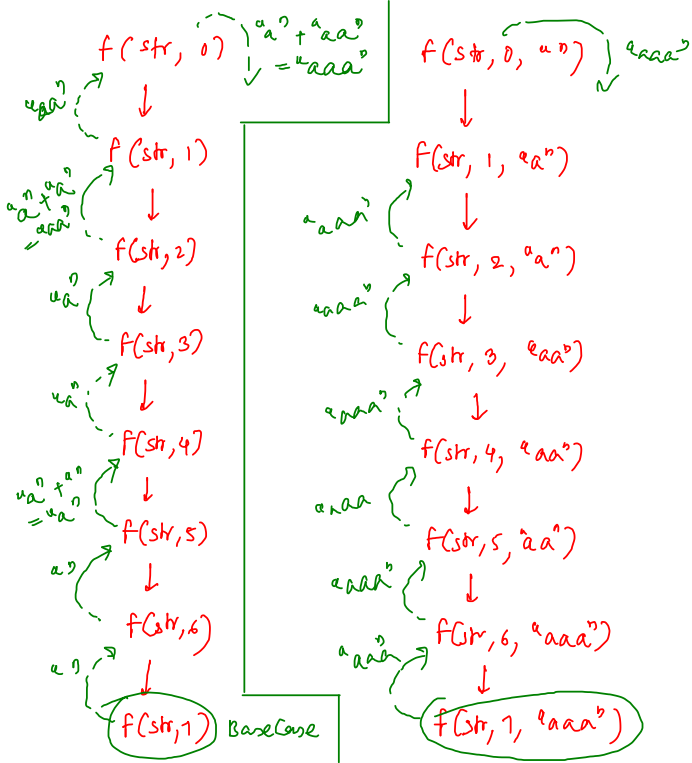
⇒ "aaaa"

⇒ removeX(str, 4)

⇒ "a"

```
// postorder
function removeX(str, idx) {
  if(idx == str.length) return "";

  const smallAns = removeX(str, idx + 1);
  if(str[idx] == 'x') return smallAns;
  return str[idx] + smallAns;
}

// preorder
function removeX(str, idx, ans) {
  if(idx == str.length) return ans;

  if(str[idx] == 'x')
    return removeX(str, idx + 1, ans); // not consider

  return removeX(str, idx + , ans + str[idx]); // consider
}
```





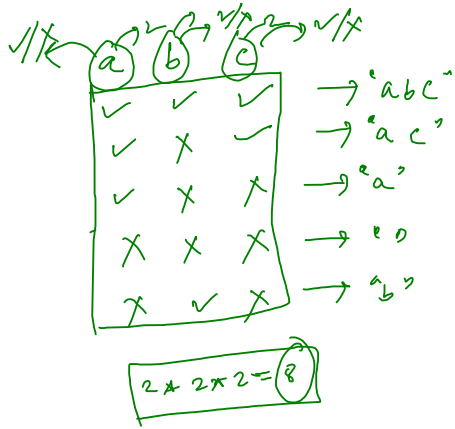
✱ Subsequences of a string :

ip : "abc"

op : "a" "b" "c" "" "ab" "bc" "ac" "abc"

8 - subsequences

You have 3 letters
- combinations



postorder :

Str = "abcd" (0 1 2 3)

"bcd"

1 1 1
1 1 0
1 0 1
1 0 0
0 1 1
0 1 0
0 0 1
0 0 0

1. faith

⇒ getSubseqwences(str, idx)

⇒ it will return/give me all the subseqwences from idx → n

→ getsubseqwences(str, 1)

⇒ ["bcd", "bc", "bd", "b", with "b" ✓

"cd", "c", "d", "", without "b" ✗]

2. logic

⇒ getSubseqwences(str, 2)

⇒ ["cd", → "bcd"
"c", → "bc"
"d", → "bd"
"", → "b"]

* ans → [ ] → with 'A'
sA → [ ] → without 'A'
[...ans, ...sA]

"abcd" → 2^4 = 16

with "A" ✓

⇒ ["abcd", "abc", "abd", "ab", "acd", "ac", "ad", "a",
"bcd", "bc", "bd", "b", "cd", "c", "d", ""]

without "A" (x)

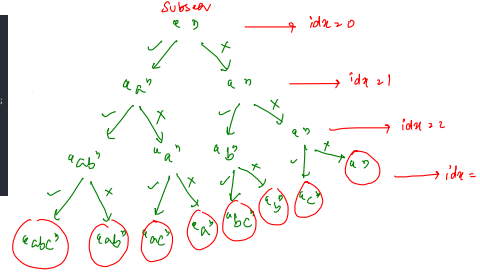
preorder :

✓ ✓ ✓ ✓ → "abcd"

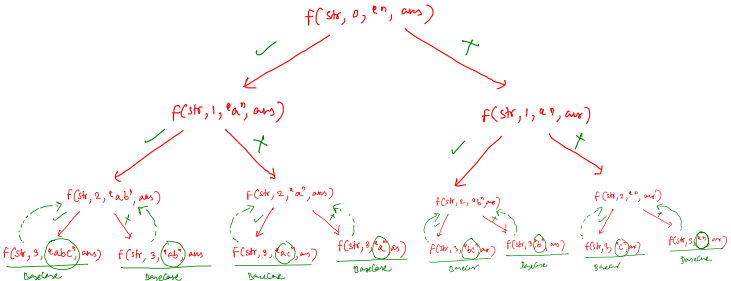
✓ x x ✓ → "ad"

```
function printSubseq ( str, idx, subseq, ans ) {
    if ( idx == str.length ) {
        ans.add (subseq);
        return;
    }
    printSubseq (str, idx+1, subseq + str[idx], ans); // consider
    printSubseq (str, idx+1, subseq, ans); // not consider
}
```

```
// preorder
function printSubSeq(str, idx, subseq, ans) {
  if (idx == str.length) {
    ans.add(subseq);
    return;
  }

  printSubSeq(str, idx + 1, subseq + str[idx], ans);
  printSubSeq(str, idx + 1, subseq, ans);
}

function printSubsequence(str) {
  const ans = [];
  printSubSeq(str, 0, "", ans);
  console.log(...ans);
}
```



Str = "abc"
0 1 2

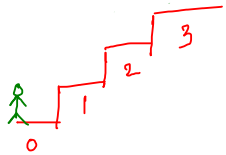


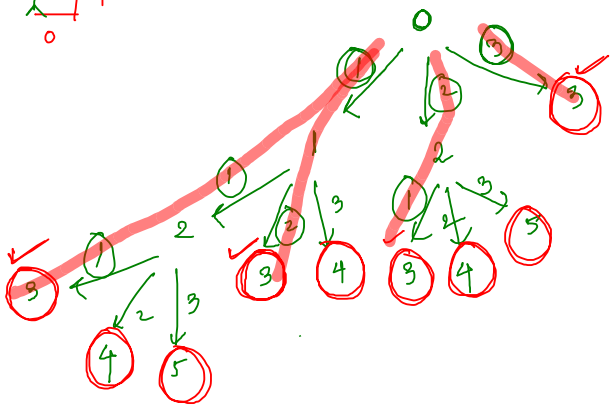
★ print stair paths

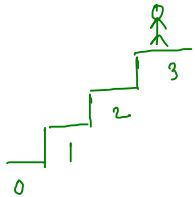
3 choices: 1, 2, 3

op: "111", "21", "12", "3"

n = 3







0 → 3 (climbing)
3 → 0 (getting down)
} both are same