kth Symbol in Grammar

DATE / /

1 0
2 0 1
3 0 1 1 0
4 0 1 1 0 1 0 0 1

complement.

8/4.

if k ≤ mid.
solve (n-1, k)

Since same as prev. row.

if k > mid.
(solve (n-1, k % mid).

(k % mid)

n == 1 && k == 1
return 0;

length = pow(2, n-1).
mid = length/2;

pow

$2^0 = 1$
$2^1 = 2$

MON TUE WED THU FRI SAT SUN

DATE / /

Towers of Hanoi

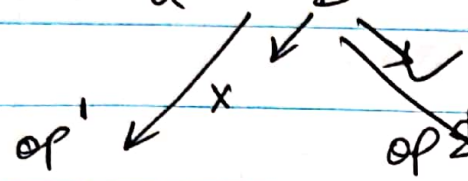
```
void TOH(int n, int from_rod, int to_rod,
                    int aux_rod)

if (n == 1)
    cout << from_rod << " " << to_rod
                                dest = aux_rod for n-1
***
TOH(n-1, from_rod, aux_rod, to_rod)

cout << from << " " << aux_rod
        _rod

***
TOH(n-1, aux_rod, to_rod, from_rod)

return n;
```

Print subsets

An recursive tree; base condition is leaf node

DATE / /

① "a" "b"



"a" " " "ab" " "

↑input

```
if (input.length() == 0) {
    cout << op << endl.
    return;
}
Str op1 = op
Str op2 = op.

op2 += Str[0]

input.erase(input.begin() + 0)

Solve (input, op1)
Solve (input, op2)

return
}
```

Unique Subsets

DATE / /

① Use a vector of strings and push all subsets in the vector

② Use `set` to get unique subsets

Print all subsets → Print all powersets

Print all Subsequences → Same hi kara do.

Print Powerset / subsequence / all subsets } Write same code

If O/P wanted in lexicographical order; put in vector and sort.

MON TUE WED THU FRI SAT SUN

Powerset
= All subsets of a set.

Print all subsets ≡ Print powersets

Substring	→	(continuous)
Subsequence	→	(Not continuous) Order matters
Subset	→	Order also not matters

DATE / /

Permutation with spaces

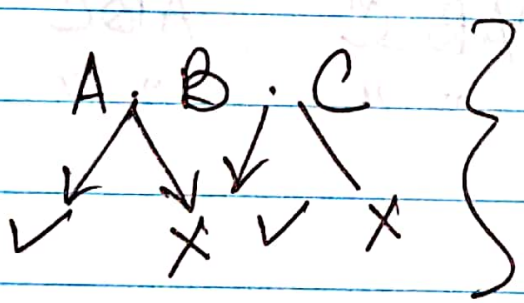
ABC

A _ B _ C

A B . C

A B _ C

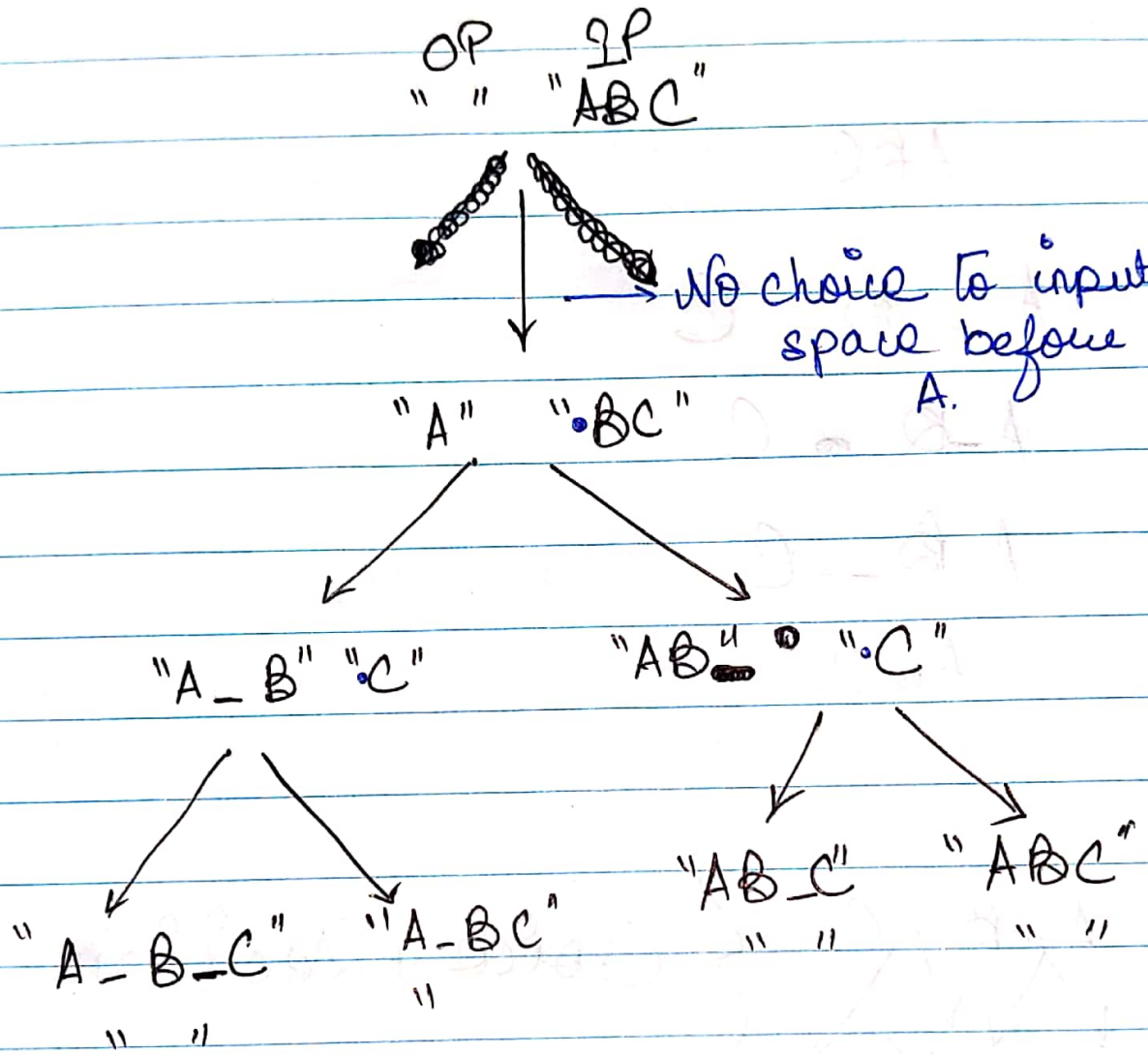
ABC



} Choice + Decision.

Recursive Tree

DATE / /

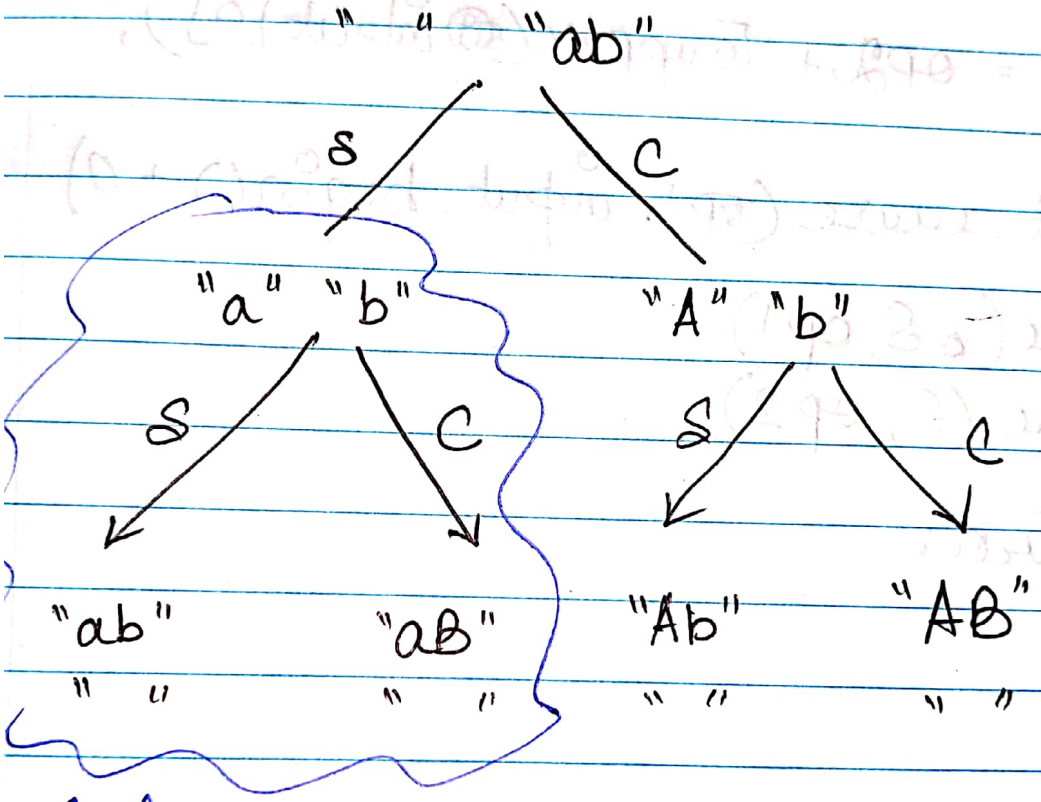


Code for Recursion in Love Babbar 450.

Permutation with case change

IP = ab

OP = ab aB Ab AB.



Code

```
if (input.length() == 0) {
    cout << op << " "
    return;
}
```

DATE / /

```
string op1 = op;
string op2 = op;

op1 += op1 + input[0];
op2 = op2 + toupper(input[0]);

input.erase(input.begin() + 0)

solve(S, op1)
solve(S, op2)

return;

}
```

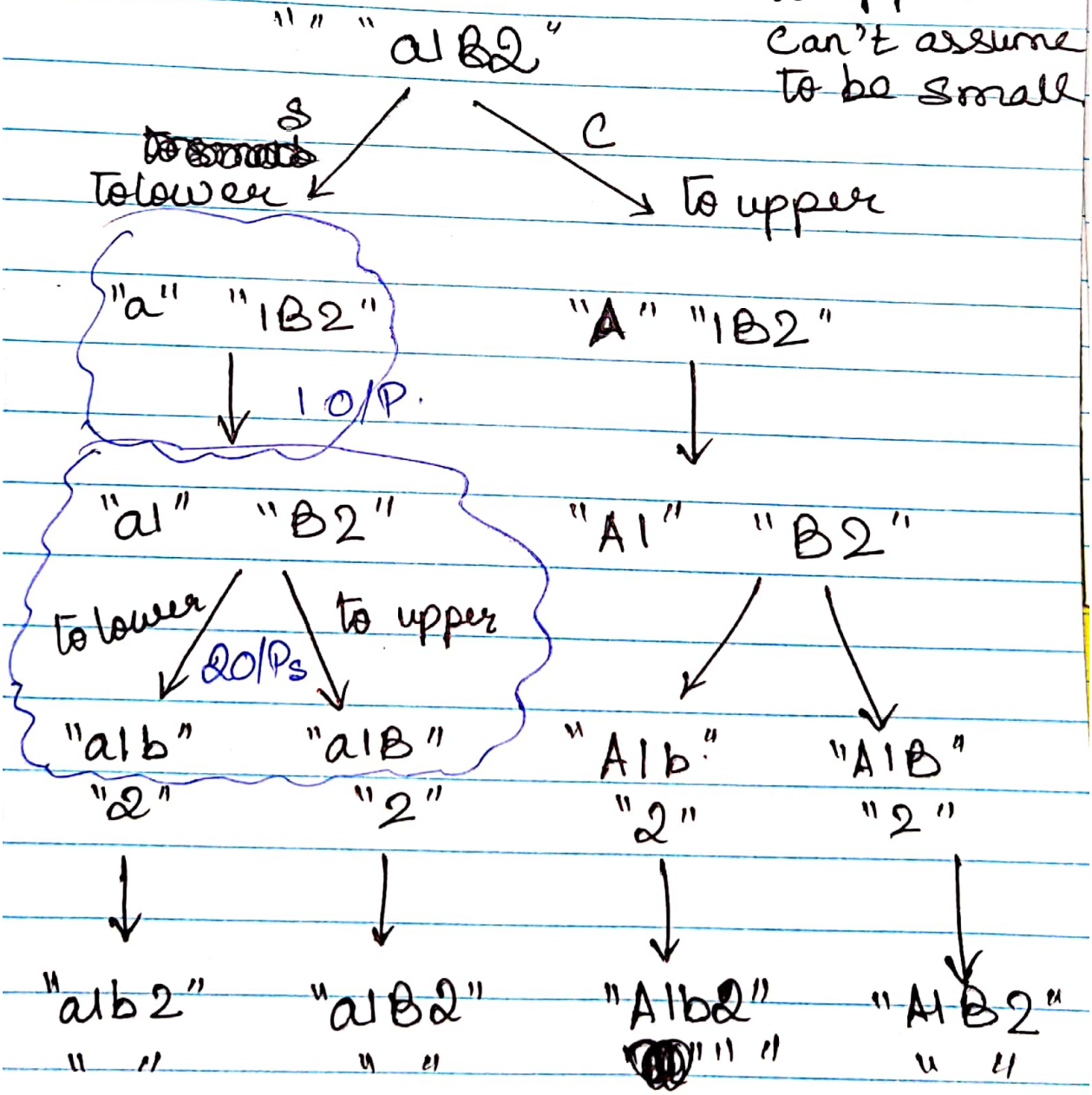
Letter case Permutation (LeetCode)

Only changes from prev.

① Ip input can be small/capital

② digit also included.

(use to lower / to upper
can't assume to be small)



MON TUE WED THU FRI SAT SUN

Generate all Balanced Parenthesis

Given an integer n, representing the number of pairs of parenthesis the task is to generate all combinations of balanced parentheses.

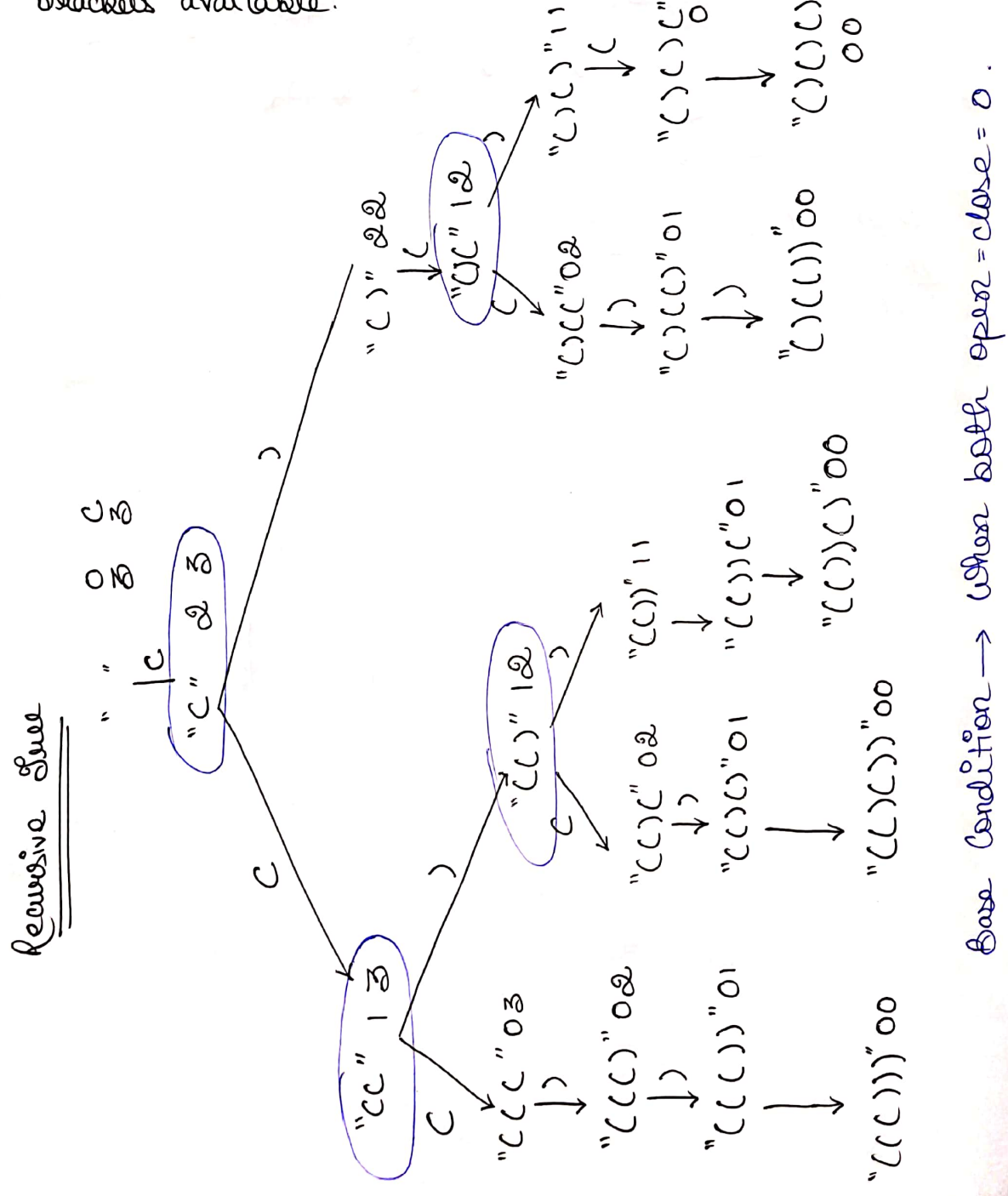
— x —

I/P - 3

O/P - ['((()))', '(()())', '(())()', '()(())'
'()()()']

Create 2 variables that keeps track of no. of open/close brackets available.

Recursive Tree



Choices

① We have choices of "(" always until no. available is 0.

② We have choices of ")" only when no. of close brackets > no. of opening brackets.

Code

```
void util(string op, int open, int close, vector<string>& res){
    if(open == 0 && close == 0){
        res.push_back(op);
        return;
    }
    if (open != 0){
        string op1 = op;
        op1 += "(";
        util(op1, open - 1, close, res);
    }
    if (close > open){
        string op2 = op;
        op2 += ")"
        util (op2, open, close-1, res);
    }
    return;
}

vector<string> AllParenthesis(int n){
    vector<string> res;
    int open = n;
    int close = n
    string op = ""
    util(op, open, close, res);
    return res;
}
```

#Print N-bit binary numbers having more 1's than 0's.

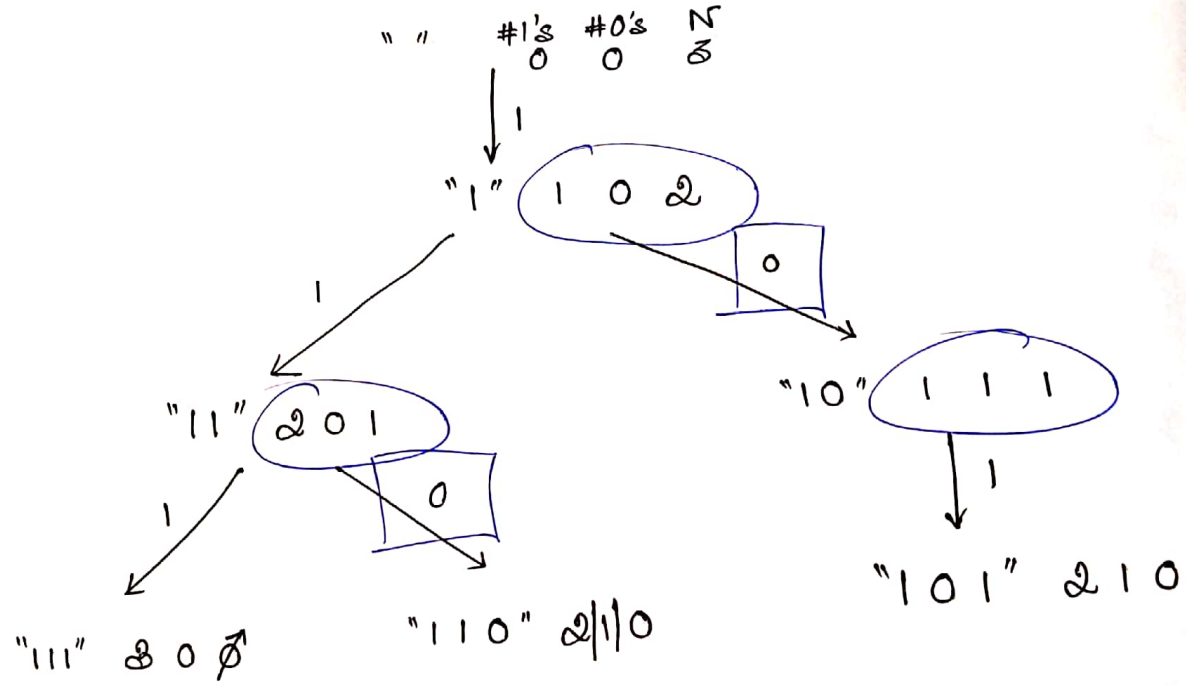
Find all possible N bit binary numbers having more than or equal number of 1's than 0's for any prefix.

—x—

N = 3

O/P => 111 110 101

Recursive Tree



Conclusions

① A "1" is available always.

② We can include a "0" only when

#1s > #0s.

③ Base condition (N=0).

Code

```
void util(int N, int ones, int zeros, string op, vector<string> & res){
    if(N == 0){
        res.push_back(op)
        return;
    }
    string op1 = op;
    op1 += "1";   // Choice of one always available
    util(N-1, ones+1, zeros, op1, res);

    if(ones > zeros){ // Choice of zero only when #1's > #0's
        string op2 = op;
        op2 += "0";
        util(N-1, ones, zeros+1, op2, res);
    }
    return res;
}

vector<string> NBitBinary(int N){
    vector<string> res;
    int ones = 0, zeros = 0;
    string op = "";
    util(N, ones, zeros, op, res);
    return res;
}
```