Backtracking

1

```
//all the subset of the string
//will get 2^n
#include <bits/stdc++.h>
using namespace std;
int substring(string a,string output[]){
    if (a.length()==0)
    {
        output[0] = "";
        return 1;
    }
    int smallerOutputSize=substring(a.substr(1),output);
    for (int i = 0; i < smallerOutputSize; ++i)
    {
        output[i+smallerOutputSize] = a[0]+output[i];
    }
    return 2*smallerOutputSize;


}
int main(){
    string output[1000];
    string a;
    cin >>a;
    //int n = a.length();
    //cout <<a << " " <<n;
    int smallerOutputSize = substring(a,output);
    for (int i = 0; i < smallerOutputSize; ++i)
    {
        cout << output[i] << endl;
    }
}
```

2

```
#include <bits/stdc++.h>
using namespace std;
void substring(string a,int n,int z){
    if (a.length()==n)
    {
        return ;
    }
    cout << a[n];
    for (int i = 0; i < z; ++i)
    {
        if(a[n]==a[i]){
```

```
            continue;
        }
        cout <<a[i];
    }
    cout << endl;
    cout << a[n];
    for (int i = z-1; i >= 0; i--)
    {
        if(a[n]==a[i]){
            continue;
        }
        cout <<a[i];
    }
    cout << endl;
    substring(a,n+1,z);
}
int main(){
    string a;
    cin >>a;
    int z = a.length();
    int n = 0;
    // cout <<a << " " <<n;
    substring(a,n,z);
}
```

3

```
#include <bits/stdc++.h>
using namespace std;
bool solveSudoku(int mat[][9],int n , int j ,int n){
    //base case
    if (i==n)
    {
        //print the solution later
        return true;
    }
    //rec case
    if(j==n){
        return solveSudoku(mat,i+1,0,n);
    }
    //skip the prefilled cell
    if (mat[i][j]!=0)
    {
        return solveSudoku(mat,i,j+1,n);
    }
    //cell to be filled
    //try out all possibilties
    for (int no = 1; no <= n; ++no)
```

```
    {
        //whether it is safe to place the number or not
        if ()
        {
            /* code */
        }
    }
}
//not completed
int main(){
    int n = 9;
    int mat[9][9]={
        {5,3,0,0,7,0,0,0,0},
        {6,0,0,1,9,5,0,0,0},
        {0,9,8,0,0,0,0,6,0},
        {8,0,0,0,6,0,0,0,3},
        {4,0,0,8,0,3,0,0,1},
        {7,0,0,0,2,0,0,0,6},
        {0,6,0,0,0,0,2,8,0},
        {0,0,0,4,1,9,0,0,5},
        {0,0,0,0,8,0,0,7,9}
    };
    if(!solveSudoku(mat,0,0,n)){
        cout << "No solution exists!" << endl;
    }
}
```