BACKTRACKING

Array backtracking :

```
public class arraybacktracking{
    public static void changeArr(int arr[], int idx, int val){
        // base case
        if(idx==arr.length){
            printArr(arr);
            return;
        }
        //recursion
        arr[idx]=val;
        changeArr(arr,idx+1,val+1);// recursion function
        //backtracking a array with -2
        arr[idx]=arr[idx]-2;
    }
     public static void printArr(int arr[]){
        for(int i=0;i<arr.length;i++){
            System.out.print(arr[i]);
        }
    }


    public static void main(String args[]){
        int arr[]=new int[5];
        int val;
       // fcall
       changeArr(arr, 0, 1);
       System.out.println();
       printArr(arr);

    }
}
Output:
12345
-10123
```

Find substrings using backtracking:

```
public class findsubstrings {

    public static void substrings(String str,String ans,int i){
```

```
        // base case
        if(i==str.length()){
            if(ans.length()==1){
                System.out.println("null");
              }else{
                System.out.println(ans);
              }
            return;
              }

              // Yes recursion
              substrings(str, ans+str.charAt(i), i+1);
              //no recursion
              substrings(str, ans, i+1);

    }
    public static void main(String args[]){
        String str= "aaa";
        substrings(str, " ", 0);
    }

}
Output:
 abc
 ab
 ac
 a
 bc
 b
 c
 null
```

permutations using backtracking :

```
public class permutations {

    public static void permutations(String str,String ans){
        // base case
        if(str.length()==0){
            System.out.println(ans);
            return;
```

```
        }
        // recursion using string builders
        for(int i=0;i<str.length();i++){
            char currchar=str.charAt(i);
        String newStr = str.substring(0,i) +str.substring(i+1, str.length());
        permutations(newStr, ans+currchar);
    }
}
    public static void main(String args[]){
        String str="abc";
        permutations(str, " ");


    }

}
Output:
 abc
 acb
 bac
 bca
 cab
 cba
```

N-QUEEN Problem for n=2:

```
public class nQueenss {

    // nQueens function
    public static void nQueens(char board[][], int rows){
        //base case
        if(rows== board.length){
            Printboard(board);
            return;
        }
        // columns tracking
        for(int j=0;j<board.length;j++){
            board[rows][j]='Q';// replacing q with .
            nQueens(board, rows+1);// recursion fcall
            board[rows][j]='X';// back tracking step
```

```
        }

    }
    public static void Printboard(char board[][]){
        for(int i=0;i<board.length;i++){
            for(int j=0;j<board.length;j++){
                System.out.print(board[i][j]+" ");
            }
            System.out.println();

        }
        System.out.println("-----------");
    }

public static void main(String args[]){
    int n=2;

    char board[][]= new char [n][n];
     // initaialize that all rowa and cols are empty so putn (.)

     for(int i=0;i<board.length;i++){
        for(int j=0;j<board.length;j++){
            board[i][j]='X';
        }
     }
     // nqueens function call
     nQueens(board,0);// fcall
}

}

Output:
Q X
Q X
-----------
Q X
X Q
-----------
X Q
Q X
-----------
X Q
X Q
-----------
```

N-QUEEN MODIFICATION:

```java
public class nQueenss {

    static boolean isSafe(char board[][], int row, int col){
        // vertical
        for(int i=row-1;i>=0;i--){
            if(board[i][col]=='Q'){
                return false;
            }
        }
        // diagonal left
        for(int i=row-1,j=col-1;i>=0 && j>=0;i--,j--){
            if(board[i][j]=='Q'){
                return false;
            }
        }
        // diag right
        for(int i=row-1,j=col+1;i>=0&&j<board.length;i--,j++){
            if(board[i][j]=='Q'){
                return false;
            }
        }
    return true;

    }

    public static void nQueens(char board[][], int row){
        //base case
        if(row== board.length){
            Printboard(board);
            return;
        }
        // columns tracking
        for(int j=0;j<board.length;j++){
            if(isSafe(board, row, j)){
                board[row][j]='Q';// replacing q with .
                nQueens(board, row+1);// recursion fcall
                board[row][j]='X';// back tracking step

            }
        }

    }
    public static void Printboard(char board[][]){
```

```
        for(int i=0;i<board.length;i++){
            for(int j=0;j<board.length;j++){
                System.out.print(board[i][j]+" ");
            }
            System.out.println();

        }
        System.out.println("-----------");
    }

public static void main(String args[]){
    int n=4;

    char board[][]= new char [n][n];
     // initaialize that all rowa and cols are empty so putn (.)

     for(int i=0;i<board.length;i++){
        for(int j=0;j<board.length;j++){
            board[i][j]='X';
        }
     }
     // nqueens function call
     nQueens(board,0);// fcall
}

}
Output:
X Q X X
X X X Q
Q X X X
X X Q X
-----------
X X Q X
Q X X X
X X X Q
X Q X X
-----------
```

Count the total no of ways:

```
public class nQueenss {
```

```java
static boolean isSafe(char board[][], int row, int col){
    // vertical
    for(int i=row-1;i>=0;i--){
        if(board[i][col]=='Q'){
            return false;
        }
    }
    // diagonal left
    for(int i=row-1,j=col-1;i>=0 && j>=0;i--,j--){
        if(board[i][j]=='Q'){
            return false;
        }
    }
    // diag right
    for(int i=row-1,j=col+1;i>=0&&j<board.length;i--,j++){
        if(board[i][j]=='Q'){
            return false;
        }
    }
return true;

}

public static void nQueens(char board[][], int row){
    //base case
    if(row== board.length){
        //Printboard(board);
        count++;
        return;
    }
    // columns tracking
    for(int j=0;j<board.length;j++){
        if(isSafe(board, row, j)){
            board[row][j]='Q';// replacing q with .
            nQueens(board, row+1);// recursion fcall
            board[row][j]='X';// back tracking step

        }
    }

}
static int count=0;
public static void Printboard(char board[][]){
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board.length;j++){
```

```
                System.out.print(board[i][j]+" ");
            }
            System.out.println();

        }
        System.out.println("-----------");
    }

public static void main(String args[]){
    int n=5;

    char board[][]= new char [n][n];
     // initaialize that all rowa and cols are empty so putn (.)

     for(int i=0;i<board.length;i++){
        for(int j=0;j<board.length;j++){
            board[i][j]='X';
        }
     }
     // nqueens function call
     nQueens(board,0);// fcall
     System.out.println("total no of ways : "+ count);
}

}
Total no of ways= 10
```

Print only one possible way:

```
public class nQueenss {

    static boolean isSafe(char board[][], int row, int col){
        // vertical
        for(int i=row-1;i>=0;i--){
            if(board[i][col]=='Q'){
                return false;
            }
        }
        // diagonal left
        for(int i=row-1,j=col-1;i>=0 && j>=0;i--,j--){
            if(board[i][j]=='Q'){
                return false;
            }
        }
```

```
        // diag right
        for(int i=row-1,j=col+1;i>=0&&j<board.length;i--,j++){
            if(board[i][j]=='Q'){
                return false;
            }
        }
    return true;

    }

    public static boolean nQueens(char board[][], int row){
        //base case
        if(row== board.length){
            //Printboard(board);
            count++;
            return true;
        }
        // columns tracking
        for(int j=0;j<board.length;j++){
            if(isSafe(board, row, j)){
                board[row][j]='Q';// replacing q with .
                if(nQueens(board, row+1)){// recursion fcall
                    return true;
                }
                board[row][j]='X';// back tracking step

            }
        }
        return false;

    }
    static int count=0;
    public static void Printboard(char board[][]){
        for(int i=0;i<board.length;i++){
            for(int j=0;j<board.length;j++){
                System.out.print(board[i][j]+" ");
            }
            System.out.println();

        }
        System.out.println("-----------");
    }

public static void main(String args[]){
    int n=2;
```

```
    char board[][]= new char [n][n];
     // initaialize that all rowa and cols are empty so putn (.)

     for(int i=0;i<board.length;i++){
        for(int j=0;j<board.length;j++){
            board[i][j]='X';
        }
     }
     // nqueens function call
    if( nQueens(board,0)){// fcall
        System.out.println("possible");
      Printboard(board);}
    else{
        System.out.println(" not possible ");
    }

    //  System.out.println("total no of ways : "+ count);
}

}
Output:
 Not possible
```

Grid ways(for right ang down only):

```
public class gridways {
    public static int gridway(int row, int col,int n,int m){
        //base case
    // if the arrow at n-1 and m-1 return 1
    if(row==n-1 || col==m-1){
        return 1;
    }
    // if arrow goes out of boundary like >n or >m
    else if(row >n || col>n){
        return 0;
    }
    // recursion function
    int way1 =gridway(row+1, col, n, m);// right side
    int way2=gridway(row, col+1, n, m);//down side
    return way1+way2;
```

```
    }

    public static void main(String args[]){
        int n=3;
        int m=3;
     int totways =gridway(0, 0, n, m);
     System.out.println(totways);

    }

}
Output;
3
```

Optimised grid ways:

```
public class gridways2 {
    public static int fact(int n){
        if(n==0){
            return 1;
        }
        return n*fact(n-1);
    }
    public static int gridways(int n,int m){
        int nm1f=fact(n-1);
        int mm1f=fact(m-1);
        int totf= fact(n-1+m-1);
        return totf/(nm1f+mm1f);


    }
    public static void main(String args[]){
        int n=3,m=3;
     System.out.println(gridways(n, m));


    }

}
Output: 6
```

Soduku Solver:

```java
public class SUDOKUSOLVEING {
    public static boolean issafe(int sudoku[][], int row,int col, int digit){
        // column
        for(int i=0;i<9;i++){
            if(sudoku[i][col]== digit){
                return false;
            }
        }
        // row
        for(int j=0;j<9;j++){
            if(sudoku[row][j]==digit){
                return false;
            }
        }
        // grid
        int sr=(row/3)*3;
        int sc=(col/3)*3;
        //3x3 grid
        for(int i=sr;i<sr+3;i++){
            for(int j=sc;j<sc+3;j++){
                if(sudoku[i][j]==digit){
                    return false;
                }

            }
        }
        return true;
    }


    public static boolean sodukusolver(int sudoku[][], int row ,int col ){
        //base case
        if(row==9){
            return true;
        }
        // recursion
    int nextrow=row, nextcol=col+1;
    if(col+1==9){
       nextrow= row+1;
        nextcol=0;
    }

    if(sudoku[row][col]!=0){
```

```
        return  sodukusolver(sudoku, nextrow, nextcol);
    }
        for(int digit=1;digit<=9;digit++){
           if(issafe(sudoku,row,col,digit)){
              sudoku[row][col]=digit;
            if(sodukusolver(sudoku, nextrow, nextcol)){
                return true;
            }
            sudoku[row][col]=0;

           }
        }
        return false;
    }

public static void printSoduku(int sodoku[][]){
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
           System.out.print( sodoku[i][j]+" ");
        }
        System.out.println();
    }
}

    public static void main(String args[]){
        int sudoku[][]={{0, 0, 8, 0, 0, 0, 0, 0, 0},
        {4, 9, 0, 1, 5, 7, 0, 0, 2},
        {0, 0, 3, 0, 0, 4, 1, 9, 0},
        {1, 8, 0, 0, 6, 0, 0, 2, 0},
        {0, 0, 0, 0, 2, 0, 0, 6, 0},
        {9, 6, 0, 4, 0, 5, 3, 0, 0},
        {0, 3, 0, 0, 7, 2, 0, 0, 4},
        {0, 4, 9, 0, 3, 0, 0, 5, 7},
        {8, 2, 7, 0, 0, 9, 0, 1, 3},};


        if(sodukusolver(sudoku,0,0)){
            System.out.println("sollution exists");
            printSoduku(sudoku);
        }
        else{
            System.out.println("solution does not exist");
        }

    }
```

```
}
Output:
sollution exists
2 1 8 3 9 6 7 4 5
4 9 6 1 5 7 8 3 2
7 5 3 2 8 4 1 9 6
1 8 4 7 6 3 5 2 9
3 7 5 9 2 8 4 6 1
9 6 2 4 1 5 3 7 8
5 3 1 6 7 2 9 8 4
6 4 9 8 3 1 2 5 7
8 2 7 5 4 9 6 1 3
```

Maze problem:

```
public class mazeproblem{
    public static boolean solvemaze(int maze[][]){
        int n=maze.length;
        int sol[][]=new int [n][n];
        if(solvemazeutil(maze,0,0,sol)== false){
            System.out.println("soln does not exist");
            return false;
        }
        printSolution(sol);
        return true;
    }
    public static boolean solvemazeutil(int maze[][],int x,int y,int sol[][]){
        if(x==maze.length-1 &&y==maze.length-1&&maze[x][y]==1){
            sol[x][y]=1;
            return true;
        }
        // check if maze [x][y] is valid
        if(issafe(maze,x,y)== true){
            if(sol[x][y]==1)
                return false;
                sol[x][y]=1;
                if(solvemazeutil(maze, x+1, y, sol))
                return true;
                if(solvemazeutil(maze, x, y+1, sol))
```

```
                return true;
                sol[x][y]=0;
                return false;

        }
        return false;
    }

    public static boolean issafe(int maze[][], int x ,int y){
        // if(x,y outside maze)return false
        return(x>=0 && x<maze.length && y>=0 && y<maze.length && maze[x][y]==1);
    }
    public static void printSolution(int sol[][]){
        for(int i=0;i<sol.length;i++){
            for(int j=0;j<sol.length;j++){
                System.out.print(" "+sol[i][j]+" ");
            }
            System.out.println();
        }
    }

    public static void main(String args[]){
        int maze[][]={{1,0,0,0},
                      {1,1,0,1},
                      {0,1,0,0},
                      {1,1,1,1}};

    solvemaze(maze);
    }
}
Output:
1  0  0  0
1  1  0  0
0  1  0  0
0  1  1  1
```

Letter combinations:

```
public class keypadcombinations {
    final static char L[][]={{},{},{'a','b','c'},
                             {'d','e','f'},
                             {'g','h','i'},
                             {'j','k','l'},
                             {'m','n','o'},
```

```
                            {'p','q','r','s'},
                            {'t','u','v'},
                            {'w','x','y','z'}};
public static void lettercombinations(String d){
int len=d.length();
if(len==0){
    System.out.println("");
    return;
}
bfa(0,len,new StringBuilder(),d);
}
public static void bfa(int pos, int len , StringBuilder sb,String d){
    if(pos==len){
        System.out.println(sb.toString());
    }
    else{
        char letters[]=L[Character.getNumericValue(d.charAt(pos))];
        for(int i=0;i<letters.length;i++){
            bfa(pos+1, len, new StringBuilder(sb).append(letters[i]), d);
        }
    }
}
public static void main(String args[]){
    lettercombinations("23");

}
}
Output:
ad
ae
af
bd
be
bf
cd
ce
cf
```

Knight moves

```
public class knightstour {
    static int N=8;
    public static boolean isSafe(int x,int y,int sol[][]){
```

```
        return(x>=0  && x<N && y>=0 && y<N && sol[x][y] == -1);
    }
    public static void printSolution(int sol[][])
     {
        for(int x=0; x<N;x++)
        {
          for(int  y=0;y<N;y++)
           System.out.print(sol[x][y] +" ");
            System.out.println();
        }
    }
public static boolean solveKT()
    {
        int sol[][] =new int[8][8];
          for(int x=0;x<N;x++)
        for(int y=0;y<N;y++)
          sol[x][y] = -1;
          int xMove[] = {2,1, -1, -2, -2, -1,1,2};
          int yMove[] = {1,2,2,1, -1, -2, -2, -1};
        //As the Knight starts from cell(0,0)
           sol[0][0] =0;
        if(!solveKTUtil(0,0,1,sol,xMove,yMove))
        {
            System.out.println("Solution does not exist");
        return false;
       }
        else printSolution(sol);
        return true;}
public static boolean solveKTUtil(int x,int y,int movei,int sol[][],int
xMove[],int yMove[])
        {
         int k,next_x,next_y;
            if(movei== N * N)
            return true;
          for(k=0;k<8;k++)
          {
            next_x = x+xMove[k];
            next_y= y+yMove[k];
            if(isSafe(next_x,next_y,sol))
            {sol[next_x][next_y] =movei;
        if(solveKTUtil(next_x,next_y,movei+1,sol,xMove,yMove))
                return true;
            else sol[next_x][next_y]= -1;
            // backtracking
        }
```

```
    }
        return false;
    }
public static void main(String args[]){

solveKT();}




}
```

```
Output:
0 59 38 33 30 17 8 63
37 34 31 60 9 62 29 16
58 1 36 39 32 27 18 7
35 48 41 26 61 10 15 28
42 57 2 49 40 23 6 19
47 50 45 54 25 20 11 14
56 43 52 3 22 13 24 5
51 46 55 44 53 4 21 12
```