Lecture-63
(Backtracking)

FREEMIND

Date

Page

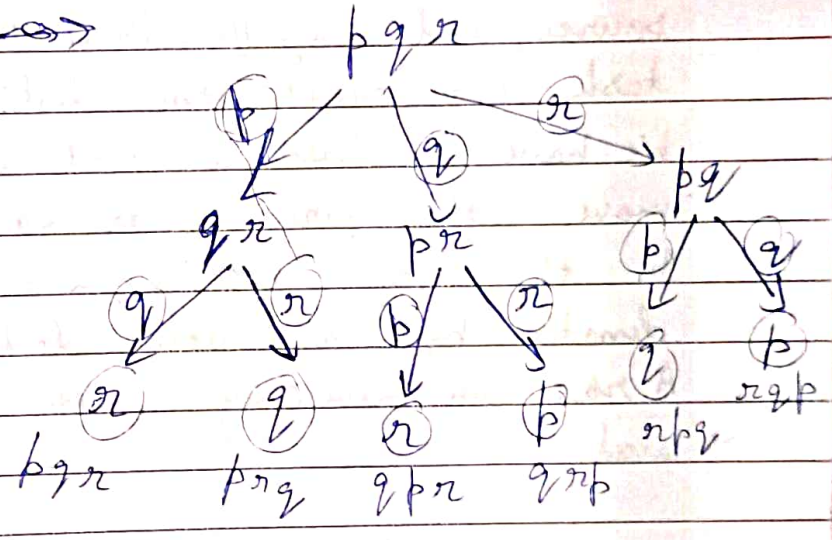
→ Can say :→ Modified form of recursion
→ Agr kisi bhi state me change kiya to usse reset bhi karda.

Pblm 1→ Permutation with back tracking;

Q1 Write a ~~proper~~ program to ~~print~~ print all permutations of the given string s in lexicographically sorted order.
Input: PQR
Output: PQR, PRQ, QPR, QRP, RPQ, RQP

Sol→ f(string s, int i) ⟹

{
this function generates all possible forms of string s from index i to n-1.




```
#include <iostream>
using namespace std;

void permutation(string &str, int i){
    //base case
    if(i == str.size()){
        cout << str << endl;
        return;
    }
    for(int j=i; j<str.size(); j++){
        swap(str[i], str[j]);
        permutation(str, i+1);
        swap(str[i], str[j]);
    }
```

```
int main () {
    string str = "abc";
    permutation (str, 0);
    return 0;
}
```

Problem 2: Rat in a maze

A maze is an N*N binary matrix of blocks where the upper left block is known as the source block, and the lower rightmost block is known as the destination block. If we consider the maze, then maze[0][0] is the source, and maze[N-1][N-1] is the destination. Our main task is to reach the destination from the source. We have considered a rat as a character that can move either forward or downwards.
In the maze matrix, a few dead blocks will be denoted by 0 and active blocks will be denoted by 1. A rat can move only in the active blocks.

→ Calculate total no. of ways to reach bottom right

Ans

```
# include <vector>
using namespace std;

bool canwego( int a, int b, vector<vector<int>> &grid ) {
    return (a < grid.size() and b < grid.size() and a >= 0 and b >= 0
            and grid[a][b] == 1;
}
int f (int i, int j, vector<vector<int>> &grid) {
    int n = grid.size();
    if ( i == n-1 and j == n-1 ) {
        for ( i = 0; i < n; i++ ) {
            for ( j = 0; j < n; j++ ) {
                cout << grid[i][j] << " ";
```

```
        }cout<<"\n";
    }
    cout<<"****\n";
    return 1;
}
int ans = 0;
grid[i][j] = 2;
if(canwego(i, j+1, grid))
    ans += f(i, j+1, grid);

if(canwego(i+1, j, grid))
    ans += f(i+1, j, grid);

if(canwego(i, j-1, grid))
    ans += f(i, j-1, grid);

if(canwego(i-1, j, grid))
    ans += f(i-1, j, grid)

grid[i][j] = 1;
return ans;
}
int main() {

    Vector<Vector<int>> grid = { {1, 1, 1, 1},
                                 {0, 1, 0, 1},
                                 {0, 1, 1, 1},
                                 {0, 1, 1, 1}
                               };

    int res = f(0, 0, grid);
    cout << res << "\n";
    return 0;
}
```

FREEMIND

Date

Page

Problem 3 ⇒ N queens

Q ⇒ Consider an N*N chessboard. N queen problem is to accommodate N queens on the N*N chessboard such that no 2 queens can attack each other.

input n = 4

output :- {0, 1, 0, 0}
{0, 0, 0, 1}
{1, 0, 0, 0}
{0, 0, 1, 0}

Ans ⇒

```
#include<vector>
using namespace std;

bool canPlaceQueen(int row, int col, vector<vector<char>>&grid)
{ // if someone attacking from vertical up.
  for(int i= row-1; i>=0; i--){
      if(grid[i][col]=='Q'){
          return false;
      }
  }
  for(int i= row-1, j=col-1; i>=0 and j>=0; i--, j--){
      if(grid[i][j]=='Q'){
          return false;
      }
  }
  for(int i= row-1, j=col+1; i>=0 and j<grid.size(); i--, j++)
  {
      if(grid[i][j]=='Q'){
          return false;
      }
  }
  return true;
}
```

```
void nQueen(int currRow, int n, vector<vector<char>>& grid){
    if (currRow == n) {
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++) {
                cout << grid[i][j] << " ";
            }
            cout << endl;
        }
        cout << "***\n";
    }
    for (col=0; col<n; col++){
        // we will go to all the cols
        // lets check if we can place a queen at
           currRow and col
        if(canPlaceQueen(currRow, col, grid)) {
            grid[currRow][col] = 'Q';
            nQueen(currRow+1, n, grid);
            grid[currRow][col] = '.';
        }
    }
}
int main() {
    int n=4;
    vector<vector<char>> grid(n, vector<char>(n,'.'));
    nqueen(0, n, grid);
    return 0;
}
```

FREEMIND

Date

Page

Lecture - 70

Backtracking - 2

→ Problem 4 : Sudoko Solver

Q-⊛ Consider a 9*9 2D array grid that is partially filled with no. from 1 to 9. The sudoko solver problem is to fill remaining blocks with no. from 1 to 9. So that every row, column and subgrid (3*3) ~~must~~ contains exactly one instance of digits (1 to 9).

input - (Unfilled cells are denoted as 0)

```
{ { 3, 0, 6, 5, 0, 8, 4, 0, 0},
  { 5, 2, 0, 0, 0, 0, 0, 0, 0}
  .
  .
  .
  . ——— ——— ——— ——— }}
```

output

```
3 1 6 5 7 8 4 9 2
5 2 9 1 3 4 7 6 8
```

Sol→

```
class Solution {
public:
    bool canWePlace(vector<vector<char>>& board, int row, int col, char num)
    {
        for (int i=0; i<9; i++) {
            if (board[i][col] == num) return false;
        }
        for (int j=0; j<9; j++) {
            if (board[row][j] == num) return false;
        }
        int r = (row/3)*3;
        int c = (col/3)*3;
        for (int i=r; i<(r+3); i++) {
            for (int j=c; j<(c+3); j++) {
                if (board[i][j] == num) return false;
            }
        }
        return true;
    }
```

0 1 2
3 4 5
6 7 8

FREEMIND

Date

Page

```
bool Sudoku (Vector<Vector<char>>& board, int row, int col)
    for (int num=1; num<=9; num++) {
        if (canWePlace(board, row, col, '0'+num)) {
        if (col == 9) return Sudoku(board, row+1, 0);
        if (row == 9) return true;
        if (board[row][col] == '.') {
            for (int num=1; num<=9; num++) {
                if (canWePlace(board, row, col, '0'+num)) {
                board[row][col] = '0'+num;
                bool res = Sudoku(board, row, col+1);
                if (res) return true;
                board[row][col] = '.';
            }  }
        }  }
        return false;
        } else {
            return Sudoku(board, row, col+1);
        }
}
Void Solve_Sudoku(Vector<Vector<char>>& grid) {
    Sudoku(board, 0, 0);
};
```

(A) → Problem 5 Place K-knights such that they do not attack each other

Q. Given integers M, N and K, the task is to place K knights on an M*N chessboard such that they don't attack each other. The knights are expected to be placed on different squares on the board. A knight can move two squares vertically and one square horizontally or two squares horizontally and one square vertically.

The knights attack each other if one of them can reach the other in single move. There are multiple ways of placing K knights on an M*N board or sometimes, no way of placing them. We are expected to list out all the possible solutions.

Examples:

Input:- M=3, N=3, k=5

Output:- K A K A K A K A K A K K K A K A Total no. of Solutions: 2

Sol:-

```
#include <iostream>
#include <vector>
using namespace std;

bool canWePlace(vector<vector<char>>& grid; int i, int j) {
    if (i-1>=0 and j-2>=0 and grid[i-1][j-2]=='K')
        return false;
    if (i-2>=0 and j-1>=0 and grid[i-2][j-1]=='K') return false;
    if (i-1>=0 and j+2>=0 and grid[i-1][j+2]=='K') return false;
    if (i-2>=0 and j+1>=0 and grid[i-2][j+1]=='K') return false;
    return true;
}
Void Knights(int m, int n, vector<vector<char>>& grid, int k,
             int i, int j){
    if (k==0) {
        //all knights are Placed
        for(int i=0; i<m; i++){
            for(int j=0; j<n; j++) {
                cout<< grid[i][j] <<" ";
            }
            cout<<"\n";
        } cout<<"****\n";
        return;
    }
```

grid[i][j] → (grid[i][j]!='K'?'_':'K');

```
    if (j == n) {
        return Knights(m, n, grid, k, i+1, 0);
    }
    if (i == m) {
        return;
    }
    for (int row = i; row < m; row++) {
        for (int col = (row == i ? j : 0); col < n; col++) {
            if (canWePlace(grid, row, col)) {
                grid[row][col] = 'k';
                Knights(m, n, grid, k-1, row, col+1);
                grid[row][col] = '0';
            }
        }
    }
}
int main() {
    int n = 3, m = 3, k = 5;
    vector<vector<char>> grid(n, vector<char>(m, '0'));
    Knights(m, n, grid, k, 0, 0);
    return 0;
}
```

(★) Problem 8: Knight's Tour

Q3 Consider an N*N chessboard. The knight's tour problem is to print order when the knight visits that block of the chessboard. Initially the knight will be placed at the first block of chess board. The rule is that the knight visits each block exactly once.

FREEMIND

Date

Page

Sol:-

```
#include<vector>
using namespace std;

bool f(Vector<Vector<int>> &grid, int i, int j, int n, int cnt){
    if(i<0 and j<0 && i>=n && j>=n && grid[i][j]>=0) return false;
    if(cnt = n*n-1){
        grid[i][j] = cnt;
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                cout << grid[i][j] << "\t";
            }
            cout << "\n";
        }
        return true;
    }
    grid[i][j] = cnt;
    if(grid, i-1, j-2, n, cnt+1) return true;
    if(grid, i-2, j-1, n, cnt+1) return true;
    if(grid, i-1, j+2, n, cnt+1)
    if(grid, i-2, j+1, n, cnt+1)
    if(grid, i+1, j-2, n, cnt+1)
    if(grid, i+1, j+2, n, cnt+1)
    if(grid, i+2, j-1, n, cnt+1)
    if(grid, i+2, j+1, n, cnt+1) return true;
    grid[i][j] = -1;
    return false;
}
int main(){
    int n = 95
    Vector<Vector<int>> grid(n, Vector<int>(n,-1));
    f(grid, 0, 0, n, 0);
    return 0;
}
```

FREEMIND

Date

Page

⭐ Combination Set:-

Q:- Given a collection of candidate numbers and a target no. Find all unique combinations in candidates where the candidate numbers sum to target.
Each number in candidates may only be used once in the combination.

Note:- The solution set must not contain duplicate combinations.

Sol:-

```
Class Solution {
public:
    Vector<Vector<int>> result;

    Void f(Vector<int>&C; int idx; int t; Vector<int>& V) {

    if (t==0) {
        V.push_back
        result.push_back(V);
        return;
    }
    if (idx == C.size()) return;
    if (C[idx] <= t) {
            V.push_back(C[idx]);        // We can Pick
f(C, idx, t-C[idx], V) --- f(C, idx+1, t-C[idx], V);
            V.pop_back();
    }
    int j = idx+1
    while (j<C.size() and C[j]==C[j-1]) j++;;
    //not pick
    f(C, j, t, V);
}
Vector<Vector<int>> CombinationSum2(Vector<int>&C,
                                    int target){
```

For next Ques

```
        result.clear();
        sort(c.begin(), c.end());
        vector<int> v;
        f(c, 0, target, v);
        return result;
    }
};
```

Q) Almost Same as Previous Questions ..

→ Distinct Integers given

→ Same no. can be choosen multiple times.

→ The test cases are generated such that the no. of unique combinations that sum up to target is less than 150 combinations for the given output.

Ans → Modified Sol. of Prev. Ques.

Trees is a hierarchical representation of nodes where one node is designated as root node and the other nodes are divided into disjoint Sub Trees.

→ Node with more than one child is Internal node

→ Path → Set of Consecutive edges make a path.

→ Forest → Collection of Trees

→ Depth → length of Path from Root to Node

→ Two trees are said to be similar if they have same Struct. (structure and)

→ " " Same if their content is same

- Full Binary Tree → Either 0 or 2 child
- Complete Binary Tree :- All level completely filled with Nodes except the last level. In last level all nodes are left as possible