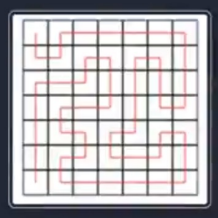
September 25, 2025

DOUBT SESSION

Ques 1. Count Valid Grid Paths

There are 88418 paths in a 7 × 7 grid from the upper-left square to the lower-left square. Each path corresponds to a 48-character description consisting of characters **D** (down), **U** (up), **L** (left), and **R** (right).

For example, the path:

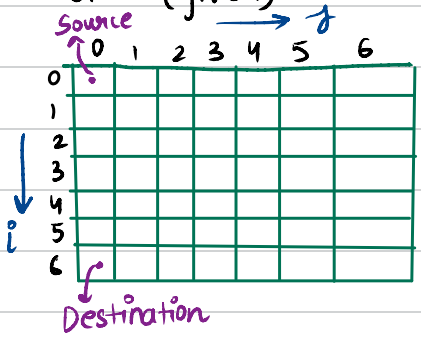


corresponds to the description:

DRURRRRRDDDLUULDDDLDRRURDDLLLLLURULURRUULDLLDDDD.

You are given a description of a path which may also contain characters **?** (any direction). Your task is to calculate the number of paths that match the description.

7X7 Grid (given)



We have to visit from source → destination in such a way that we go through every cell of the grid (only once) to reach our destination

→ Visit all the cell exactly once.

We will get a string as a direction like (DRUL ?], we have to swap '?' with a direction such that we follow all the rules.

Brute force:

Worst case scenario is the string filled with '?'. Best case is no '?' present.

Worst case T.C = $4^{49} \rightarrow 2^{98}$ (TLE)

Best case = O(1)

Recursion → optimise

position in cell, visited

```
rec(i, j, vis, steps)
{
    if (i==6 && j==0) //at destination
    {
        if (steps==49) return 1;
        else return 0;
    }
    if (steps==49) // completed all the steps if dest is reached then return 1 else 0.
    {
        if (i==6 && j==0) return 1;
        return 0;
    }
    if (left && right → visited && up && down → not visited)
        return 0;
```

cell

```
string s;
int vis[9][9];
int helper(int idx, int i, int j)
{
    if ((vis[i][j - 1] && vis[i][j + 1]) && (!vis[i - 1][j] && !vis[i + 1][j]))
    {
        return 0;
    }
    if ((!vis[i][j - 1] && !vis[i][j + 1]) &&  (vis[i - 1][j] && vis[i + 1][j]))
    {
        return 0;
    }
    if (i == 7 and j == 1)
    {
        return (idx == 48);
    }
    if (idx == 48)
    {
        if (i == 7 and j == 1)
        {
            return 1;
        }
        return 0;
    }
    vis[i][j] = 1;
    int cnt = 0;
    if (s[idx] == '?')
    {
        // cout << "rand " << endl;
        int dx[] = {1, -1, 0, 0};
        int dy[] = {0, 0, 1, -1};
        f0(k, 4)
```

```
    {
        int ni = dx[k] + i, ny = dy[k] + j;
        if (ni >= 1 and ny >= 1 and ni <= 7 and ny <= 7 and vis[ni][ny] == 0)
        {
            cnt += helper(idx + 1, ni, ny);
        }
    }
}
else
{
    // cout << "no ";
    if (s[idx] == 'R' and j + 1 <= 7 and vis[i][j + 1] == 0)
    {
        cnt += helper(idx + 1, i, j + 1);
```

Answer = $4*x + 4*y + z$

```
int point[10][10];
int traverse(int currKey, int movesLeft, bool vis[10])
{
    if (movesLeft == 0)
        return 1;

    vis[currKey] = true;

    int numberOfPaths = 0;
    for (int i = 1; i <= 9; i++)
    {
        if (vis[i] == false && (point[currKey][i] == 0 || vis[point[currKey][i]] == true))
        {
            numberOfPaths += traverse(i, movesLeft - 1, vis);
        }
    }

    vis[currKey] = false;

    return numberOfPaths;
}

void solve()
{
    int n;
    cin >> n;
    int ans = 0;
    for (int i = 1; i <= 9; i++)
    {
        for (int j = 1; j <= 9; j++)
        {
            point[i][j] = 0;
        }
    }

    point[1][3] = point[3][1] = 2;
    point[4][6] = point[6][4] = 5;
    point[7][9] = point[9][7] = 8;
    point[1][7] = point[7][1] = 4;
    point[2][8] = point[8][2] = 5;
    point[3][9] = point[9][3] = 6;
    point[1][9] = point[9][1] = 5;
    point[3][7] = point[7][3] = 5;

    bool vis[10];
    for (int i = 1; i <= 9; i++)
    {
        vis[i] = false;
    }
    ans += 4 * traverse(1, n - 1, vis);
    ans += 4 * traverse(2, n - 1, vis);
    ans += traverse(5, n - 1, vis);

    cout << ans;

}
signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
// #ifndef ONLINE_JUDGE
//     freopen("input.txt", "r", stdin);
//     freopen("output.txt", "w", stdout);
// #endif // ONLINE_JUDGE
    int t = 1;
    // cin >> t;
    while (t--)
    {
        solve();
    }
    return 0;
}
```

Ques 2. Android Unlock Pattern

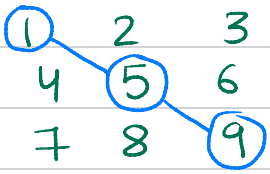
Android mobile unlock pattern is a grid of 3 × 3 cells, where drawing a specific pattern connecting a specific sequence of cells in order will unlock the android mobile.

Given a number n, your task is to find the number of android unlock patterns connecting exactly n cells.

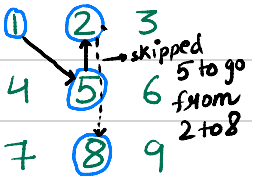
Rules of a valid pattern:

1. Each pattern must connect exactly n cells.
2. All the cells must be distinct.
3. If the line connecting two consecutive cells in the pattern passes through any other cells, the other cells must have previously been selected in the pattern. No jumps through the non-selected cell are allowed.
4. The order of cells used matters.

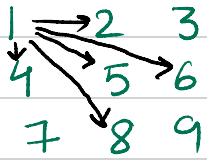
Ex: n = 3



n = 4



Where can we go directly from 1?



How many patterns can we make with n=3?

Let us say, starting from 1 (3,7,9)

$Count_1 = x$

Starting from 2 (will be similar for 4, 6, 8 due to symmetry)

$Count_2 = y$

Starting from 5

$Count_5 = z$

Ques 3 Topological Labelling

You are given a directed acyclic graph with n vertices and m edges. There are no self-loops or multiple edges between any pair of vertices. The graph can be disconnected.

You should assign labels to all vertices in such a way that:

If there exists an edge from vertex v to vertex u then $label_v$ should be smaller than $label_u$.

Labels form a valid permutation of length n — an integer sequence such that each integer from 1 to n appears exactly once in it.

Permutation should be lexicographically smallest among all suitable.

Find such a sequence of labels to satisfy all the conditions.

Example:

Let's say we have three nodes

Node : 3 2 1

Label : 3 1 2

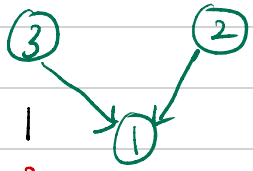
Now, have to write permutation first there will be label of 1, then label of 2 and Label of 3

→ 2 1 3

If there is an edge from $v \to u$

$\Rightarrow$ Label [v] < Label [u]

Example



- Node: 3 2 1

L : 3 1 2

Is 3 < 2, No so this is not valid labelling

label of 3 — label of 1 — edge from 3 to 1

- N = 3 2 1

L = 1 3 2

Edge: 3 → 1 : L[3] < L[1], 1 < 2, Yes

2 → 1 : L[2] < L[1], 3 < 2, No, Wrong label

N = 3 2 1

L = 1 2 3 Correct Labelling

3 → 1 : L[3] < L[1], 1 < 3, YES

2 → 1 : L[2] < L[1], 2 < 3, YES

Since labelling is correct, lets see its permutation:

P_1 → 3 (Node→1) 2 (2) 1 (3)

Can this also be correct labelling

N = 3 2 1

L = 2 1 3 Correct

Permutation, P_2 → 3 1 2

Which among P_1 & P_2 is lexographically smaller?

$\Rightarrow$ P_2 (3 1 2)

→ Smallest priority → write label in increasing order

→ Reverse graph → Biggest → label decreasing