EASY :

1. Lucky number in a matrix

```
class Solution {
public:
   vector<int> luckyNumbers (vector<vector<int>>& matrix)
   {
      vector<int>arr;
      int minnn=0;
      int maxxx=0;
      for(int i=0;i<matrix.size();i++)
      {
          for(int j=0;j<matrix[0].size();j++)
         {
            int val=matrix[i][j];
            minnn=INT_MAX;
            maxxx=INT_MIN;

            for(int k=0;k<matrix.size();k++)
            {
               if(matrix[k][j]>maxxx)
               maxxx=matrix[k][j];
            }

             for(int l=0;l<matrix[0].size();l++)
            {
               if(matrix[i][l]<minnn)
               minnn=matrix[i][l];
            }

            if(val==maxxx&&val==minnn)
            arr.push_back(val);
         }
      }
    return arr;

   }
};
```

2. Transpose of a matrix

```
class Solution {
public:
   vector<vector<int>> transpose(vector<vector<int>>& matrix)
   {
      int n=matrix.size();
```

```
        int m=matrix[0].size();
        vector<vector<int>>arr(m,vector<int>(n,0));
        for(int i=0;i<m;i++)
        {
            for(int j=0;j<n;j++)
            {
                arr[i][j]=matrix[j][i];
            }
        }
        return arr;

    }
};
```

3. Toeplitz matrix

```
class Solution {
public:
    bool isToeplitzMatrix(vector<vector<int>>& matrix)
    {
        int n=matrix.size();
        int m=matrix[0].size();
        if(n==1 || m==1)
        return 1;
        else
        {
            for(int i=1;i<n;i++)
            {
                for(int j=1;j<m;j++)
                if(matrix[i][j]!=matrix[i-1][j-1])
                return 0;
            }
            return 1;

        }
        return 1;

    }
};
```

MEDIUM :

1. Game of life

```
class Solution {
public:
    void gameOfLife(vector<vector<int>>& board)
    {
        int n=board.size();
        int m=board[0].size();
        vector<vector<int>>v(n);

        for(int i=0;i<n;i++)
        v[i].resize(m);

        int c=0;
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                c=0;
                if(j>0 && board[i][j-1]==1)
                c++;
                if(j>0 && i>0 && board[i-1][j-1]==1)
                c++;
                if(i>0 && board[i-1][j]==1)
                c++;
                if(i>0 && j<m-1 && board[i-1][j+1]==1)
                c++;
                if(j<m-1 && board[i][j+1]==1)
                c++;
                if(j<m-1 && i<n-1 && board[i+1][j+1]==1)
                c++;
                if(i<n-1 && board[i+1][j]==1)
                c++;
                if(i<n-1 && j>0 && board[i+1][j-1]==1)
                c++;
                if(c==3 && board[i][j]==0)
                v[i][j]=1;
                else if(c<2 && board[i][j]==1)
                v[i][j]=0;
                else if(c<=3 && c>=2 && board[i][j]==1)
                v[i][j]=1;
                else if(c>3 && board[i][j]==1)
                v[i][j]=0;
                else
                v[i][j]=board[i][j];
            }
        }
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                board[i][j]=v[i][j];
            }
        }
    }
};
```

2. Count square submatrices with all ones

```
class Solution {
public:
    int countSquares(vector<vector<int>>& mat)
    {
        int n=mat.size();
        int m=mat[0].size();
        int dp[n][m];

        for(int i=0;i<n; i++)
        dp[i][0] = mat[i][0];

        for(int j=0;j<m;j++)
        dp[0][j] = mat[0][j];

        int ans=0;

        for(int i=1; i<n; i++)
        {
            for(int j=1; j<m; j++)
            {
                if(mat[i][j] == 1)
                dp[i][j] = min(dp[i][j-1],min( dp[i-1][j],dp[i-1][j-1])) + 1;
                else
                dp[i][j] = 0;
            }
        }
        for(int i=0; i<n; i++)
        for(int j=0; j<m; j++)
        ans+=dp[i][j];

        return ans;
     }
};
```

3. Rotate image

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {

        int n= matrix.size();
        for(int i=0; i< n; i++)
        {
            for(int j=0; j<i; j++)
            {
                swap(matrix[i][j],matrix[j][i]);
            }
        }
        for(int i=0; i<n; i++)
        reverse(matrix[i].begin(), matrix[i].end());
    }
};
```

4. Maximal square

```
class Solution {
public:
    int maximalSquare(vector<vector<char>>& matrix)
    {
         int n=matrix.size();
         int m=matrix[0].size();
         int ans=0;
         vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= m; j++)
            {
                if (matrix[i-1][j-1] == '1')
                {
                    dp[i][j]=min(min(dp[i-1][j], dp[i][j-1]), dp[i-1][j-1])+1;
                    ans=max(ans,dp[i][j]);
                }
            }
        }

        return ans*ans;
    }
};
```

5. Search in a 2D

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target)
    {
        int i=0;
        int j=0;

        for(j=0;j<matrix.size();j++)
        {
            if(matrix[j][i]>target)
            break;
        }
        if(j>=1)
        j--;
        for(int i=0;i<matrix[0].size();i++)
        {
            if(matrix[j][i]==target)
            return true;
        }
        return false;
    }
};
```

6. Spiral Matrix

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix)
    {
          vector<int>ans;
          if(matrix.size()==0)
          return ans;

          int r=matrix.size();
          int c=matrix[0].size();

          int i;
          int k=0;
          int l=0;

          while (k<r && l<c)
          {

               for(i=l;i<c;++i)
               ans.push_back(matrix[k][i]);
```

```
            k++;


            for(i=k;i<r;++i)
            ans.push_back(matrix[i][c-1]);

            c--;


            if (k<r)
            {
               for(i=c-1;i>=l;--i)
               ans.push_back(matrix[r-1][i]);

               r--;
            }


            if (l<c)
            {
               for (i=r-1;i>=k;--i)
               ans.push_back(matrix[i][l]);

               l++;
            }
      }
      return ans;
   }
};
```

7. Snakes and ladders

```
class Solution {
public:
   int snakesAndLadders(vector<vector<int>>& board)
   {
      unordered_map<int, int>mp;
      int  n = board[0].size();
      int dir = 1;
      int index = 1;
      for(int i=n-1;i>=0;i--)
      {
         if(dir==1)
         {
            for(int j = 0; j < n; j++)
            {
                if(board[i][j] != -1)
                mp[index] = board[i][j];
                index++;
            }
        }
        else
        {
            for(int j = n - 1; j >= 0; j--)
            {
                if(board[i][j] != -1)
                mp[index] = board[i][j];
                index++;
            }
        }
        dir=-dir;

    }

              int m=n*n+1;
    vector<int>dist(m, -1);
    queue<int>q;
    q.push(1);
    dist[1] = 0;
    while (!q.empty())
    {
        int v=q.front();
        q.pop();
        if (v== n*n)
        return dist[n*n];
        for (int k=1;k<=6;k++)
        {
            int temp=v+k;
            if (temp>n*n)
            break;
            if (mp.find(temp) != mp.end())
            temp = mp[temp];
            if (dist[temp] == -1)
            {
                dist[temp] = dist[v] + 1;
                q.push(temp);
            }
         }
     }
    return -1;
  }
};
```