Practical 1

Write a program to determine whether the given number is Prime or not.

Code:

```
#include <iostream>
using namespace std;
int main() {
  int i, n;
  bool is_prime = true;
  cout << "Enter a positive integer: ";
  cin >> n;
  if (n == 0 || n == 1) {
    is_prime = false;
  }
  for (i = 2; i <= n/2; ++i) {
    if (n % i == 0) {
      is_prime = false;
      break;
    }
  }
  if (is_prime)
    cout << n << " is a prime number";
  else
    cout << n << " is not a prime number";
  return 0;
}
```

Output:

```
Enter a positive integer: 29
29 is a prime number
```

Practical 2

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int searchInsert(vector<int>& nums, int target) {
  int low = 0, high = nums.size() - 1;
  while (low <= high) {
    int mid = low + (high - low) / 2;
    if (nums[mid] == target) return mid;
    else if (nums[mid] < target) low = mid + 1;
    else high = mid - 1;
  }
  return low;
}
int main() {
  int N;
  cin >> N;
  vector<int> nums(N);
  for(int i = 0; i < N; i++) {
    cin >> nums[i];
  }
  int target;
  cin >> target;
  cout << searchInsert(nums, target);
  return 0;
}
```

Output:

```
5
1 3 5 6 8
5
```

Practical 3

There are N children standing in a line with some rating value. You want to distribute a minimum number of candies to these children such that: Each child must have at least one candy. The children with higher ratings will have more candies than their neighbours. You need to write a program to calculate the minimum candies you must give.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int N;
    cin >> N;
    vector<int> ratings(N);
    for (int i = 0; i < N; i++) {
        cin >> ratings[i];
    }
    vector<int> candies(N, 1);
    for (int i = 1; i < N; i++) {
        if (ratings[i] > ratings[i - 1]) {
            candies[i] = candies[i - 1] + 1;
        }
    }
    for (int i = N - 2; i >= 0; i--) {
        if (ratings[i] > ratings[i + 1]) {
            candies[i] = max(candies[i], candies[i + 1] + 1);
        }
    }
    int total = 0;
    for (int c : candies) total += c;
    cout << total;
    return 0;
}
```

Output:

Input:

5

1 2 2 3 1

Output:

8

Practical 4

There is a new barn with N stalls and C cows. The stalls are located on a straight line at positions x1, xN (0 <= xi <= 1,000,000,000). We want to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

Code:

```
#include <bits/stdc++.h>
using namespace std;

bool canPlace(vector<int> &stalls, int cows, int dist) {
      int count = 1;  // place the first cow at first stall
      int lastPos = stalls[0];
      for (int i = 1; i < stalls.size(); i++) {
        if (stalls[i] - lastPos >= dist) {
          count++;
          lastPos = stalls[i];
          if (count >= cows) return true;
        }
      }
      return false;
}

int main() {
  int N, C;
  cin >> N >> C;
  vector<int> stalls(N);
  for (int i = 0; i < N; i++) cin >> stalls[i];
  sort(stalls.begin(), stalls.end());

  int low = 1;
  int high = stalls.back() - stalls[0];
  int ans = 0;
```

```
  while (low <= high) {
    int mid = low + (high - low) / 2;
    if (canPlace(stalls, C, mid)) {
      ans = mid;
      low = mid + 1;
    }
    else {
      high = mid - 1;
    }
  }
  cout << ans << "\n";
  return 0;
}
```

Input:

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

Output:

```
3
```

Practical 5

Given an undirected graph with V vertices and E edges, check whether it contains any cycle or not

Code:

```
#include <bits/stdc++.h>
using namespace std;

bool dfs(int node, int parent, vector<int> adj[], vector<bool>
&visited) {
  visited[node] = true;

  for (int neighbor : adj[node]) {
    if (!visited[neighbor]) {
      if (dfs(neighbor, node, adj, visited)) return true;
    }
    else if (neighbor != parent) return true;
  }
  return false;
}

string detectCycle(int V, vector<pair<int, int>> &edges) {
  vector<int> adj[V];
  for (auto edge : edges) {
    int u = edge.first, v = edge.second;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  vector<bool> visited(V, false);
  for (int i = 0; i < V; i++) {
    if (!visited[i]) {
      if (dfs(i, -1, adj, visited))
        return "Cycle Detected";
    }
  }
  return "No Cycle Detected";
}
```

```
int main() {
  int V, E;
  cin >> V >> E;

  vector<pair<int, int>> edges;
  for (int i = 0; i < E; i++) {
    int u, v;
    cin >> u >> v;
    edges.push_back({u, v});
  }
  cout << detectCycle(V, edges) << endl;
  return 0;
}
```

Input:

```
4 4
0 1
1 2
2 3
3 0
```

Output:

```
Cycle Detected
```

Practical 6

There are n servers numbered from 0 to n – 1 connected by undirected server-to-server connections forming a network where connections[i] = [ai, bi] represents a connection between servers ai and bi. Any server can reach other servers directly or indirectly through the network. A critical connection is a connection that, if removed, will make some servers unable to reach some other servers. Return all critical connections in the network in any order.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
  int n, m;
  if (!(cin >> n >> m)) return 0;
  vector<vector<int>> adj(n);
  for (int i = 0; i < m; ++i) {
    int u, v; cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }
  vector<int> disc(n, 0), low(n, 0);
  int timer = 0;
  vector<pair<int,int>> bridges;
  function<void(int,int)> dfs = [&](int u, int p) {
    disc[u] = low[u] = ++timer;
    for (int v : adj[u]) {
      if (v == p) continue;
      if (disc[v] == 0) {
        dfs(v, u);
        low[u] = min(low[u], low[v]);
      }
      if (low[v] > disc[u]) bridges.emplace_back(u, v);
      else low[u] = min(low[u], disc[v]);
    }
  };
```

```
  for (int i = 0; i < n; ++i)
       if (disc[i] == 0) dfs(i, -1);
  cout << (bridges.empty() ? "Yes" : "No") << '\n';
  return 0;
}
```

Input:

```
4 5
0 1
1 2
2 3
3 0
0 2
```

Output:

```
Yes
```

Practical 7

Given a grid of size NxM (N is the number of rows and M is the number of columns in the grid) consisting of '0's (Water) and '1's(Land). Find the number of islands

Code:

```
#include <bits/stdc++.h>
using namespace std;

int R, C;
vector<vector<char>> grid;
vector<vector<bool>> visited;
int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};

bool isValid(int x, int y) {
  return (x >= 0 && x < R && y >= 0 && y < C && grid[x][y] == '1'
&& !visited[x][y]);
}

void dfs(int x, int y) {
  visited[x][y] = true;
  for (int d = 0; d < 4; d++) {
    int nx = x + dx[d];
    int ny = y + dy[d];
    if (isValid(nx, ny)) dfs(nx, ny);
  }
}

int main() {
  cin >> R >> C;
  grid.resize(R, vector<char>(C));
  visited.resize(R, vector<bool>(C, false));
  for(int i = 0; i < R; i++) {
    for (int j = 0; j < C; j++) {
      cin >> grid[i][j];
    }
  }
```

```
  int islands = 0;
  for (int i = 0; i < R; i++) {
    for (int j = 0; j < C; j++) {
      if (grid[i][j] == '1' && !visited[i][j]) {
        dfs(i, j);
        islands++;
      }
    }
  }
  cout << islands << endl;
  return 0;
}
```

Input:

```
4 5
1 1 0 0 0
1 1 0 0 1
0 0 1 0 1
0 0 0 1 1
```

Output:

```
4
```

Practical 8

Given a grid of dimension N x M where each cell in the grid can have values 0, 1, or 2 which has e following meaning: 0: Empty cell 1: Cells have fres 2. Cells have oranrotten oranges We have to determine what is the minimum time required to rot all oranges. A rotten orange at index [i,j] can rot other fresh oranges at indexes [i-1,j], [i+1,j], [i,j-1], [i,j+1] (up, down, left and right) in unit time'

Code:

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;
class Cell {
public:
  int x;
  int y;
  int time;

  Cell(int x, int y, int time) {
    this->x = x;
    this->y = y;
    this->time = time;
  }
};

bool isValid(int x, int y, int N, int M, const vector<vector<int>>&
grid) {
      return x >= 0 && x < N && y >= 0 && y < M && grid[x][y] == 1;
}

int orangesRotting(vector<vector<int>>& grid) {
  int N = grid.size();
  if (N == 0) return 0;
  int M = grid[0].size();
  queue<Cell> q;
  int freshOranges = 0;
```

```
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < M; ++j) {
      if (grid[i][j] == 2) q.push(Cell(i, j, 0));
      else if (grid[i][j] == 1) freshOranges++;
    }
  }
  int time = 0;
  int dx[] = {0, 0, 1, -1};
  int dy[] = {1, -1, 0, 0};
  while (!q.empty()) {
    Cell current = q.front();
    q.pop();
    time = max(time, current.time);
    for (int i = 0; i < 4; ++i) {
      int newX = current.x + dx[i];
      int newY = current.y + dy[i];
      if (isValid(newX, newY, N, M, grid)) {
        grid[newX][newY] = 2;
        freshOranges--;
        q.push(Cell(newX, newY, current.time + 1));
      }
    }
  }
  if (freshOranges > 0) {
    return -1;
  }
  return time;
}
int main() {
  int N, M;
  cin >> N >> M;
  vector<vector<int>> grid(N, vector<int>(M));
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < M; ++j)
      cin >> grid[i][j];
  }
  cout << orangesRotting(grid) << endl;
  return 0;
}
```

Input:

```
3 3
2 1 1
1 1 0
0 1 1
```

Output:

```
4
```

Practical 9

Given two strings str1 and str2 and below operations that can be performed on str1. Find minimum number of edits (operations) required to convert 'str1' into 'str2'. Insert Remove Replace, All of the above operations are of equal cost.

Code:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
int minEditDistance(const string &str1, const string &str2) {
  int m = str1.size();
  int n = str2.size();
  vector<vector<int>> dp(m+1, vector<int>(n+1, 0));
  for(int i = 0; i <= m; ++i) {
    for (int j = 0; j <= n; ++j) {
      if (i == 0) dp[i][j] = j;
      else if (j == 0) dp[i][j] = i;
      else if (str1[i-1] == str2[j-1]) dp[i][j] = dp[i-1][j-1];
      else dp[i][j]=1+min(min(dp[i-1][j],dp[i][j-1]),dp[i-1][j-1]);
    }
  }
  return dp[m][n];
}
int main() {
  string str1, str2;
  cin >> str1 >> str2;
  cout << minEditDistance(str1, str2) << endl;
  return 0;
}
```

Input:

```
kitten
sitting
```

Output:

```
3
```

Practical 10

Minimum Path Sum" says that given a n x m grid consisting of non-negative integers and we need to find a path from top left to bottom right, which minimizes the sum of all numbers along the path.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
  int n, m;
  cin >> n >> m;
  vector<vector<int>> grid(n, vector<int>(m));
  for (int i = 0; i < n; ++i){
    for(int j = 0; j < m; ++j)
      cin >> grid[i][j];
  }
  vector<vector<int>> dp(n, vector<int>(m));
  dp = grid;
  for (int j = 1; j < m; ++j)
    dp[0][j] = dp[0][j-1] + grid[0][j];
  for (int i = 1; i < n; ++i)
    dp[i][0] = dp[i-1][0] + grid[i][0];
  for(int i = 1; i < n; ++i) {
    for (int j = 1; j < m; ++j)
      dp[i][j] = grid[i][j] + min(dp[i-1][j], dp[i][j-1]);
  }
  cout << dp[n-1][m-1] << endl;
  return 0;
}
```

Input:

```
3 3
1 3 1
1 5 1
4 2 1
```

Output:

```
7
```

Practical 11

Given string num representing a non-negative integer num, and an integer k, return the smallest possible integer after removing k digits from num.

Code:

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

string removeKdigits(string num, int k) {
  vector<char> stack;
  for (char digit : num) {
    while (!stack.empty() && k > 0 && stack.back() > digit) {
      stack.pop_back();
      k--;
    }
    stack.push_back(digit);
  }
  while (k > 0) {
    stack.pop_back();
    k--;
  }

  string result = "";
  for (char digit : stack)
    result += digit;

  size_t firstDigit = result.find_first_not_of('0');
  if (string::npos != firstDigit) result =
result.substr(firstDigit);
  else return "0";
  return result.empty() ? "0" : result;
}
```

```
int main() {
  string a;
  int b;
  cin>>a>>b;
  cout << removeKdigits(a, b) << endl;
  return 0;
}
```

Input:

```
10200 1
```

Output:

```
200
```

Practical 12

There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]).The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corn

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int uniquePaths(int m, int n) {
  vector<vector<int>> dp(m, vector<int>(n, 1));
  for (int i = 1; i < m; ++i) {
    for (int j = 1; j < n; ++j)
      dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
  }
  return dp[m - 1][n - 1];
}
int main() {
  int m, n;
  cin>>m>>n;
  cout << uniquePaths(m, n) << endl;
  return 0;
}
```

Input:

```
3 7
```

Output:

```
28
```