**1.Rat in a Maze problem**  
  
Solution:  
class Solution {

public:

vector<string> findPath(vector<vector<int>> &mat) {

int n = mat.size();

vector<string> res;

if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) {

return res;

}

dfs(mat, 0, 0, n, "", res);

sort(res.begin(), res.end());

return res;

}

void dfs(vector<vector<int>>& mat, int i, int j, int n, string path, vector<string>& res) {

if (i == n - 1 && j == n - 1) {

res.push\_back(path);

return;

}

mat[i][j] = 0;

if (i + 1 < n && mat[i + 1][j] == 1) {

dfs(mat, i + 1, j, n, path + 'D', res);

}

if (i - 1 >= 0 && mat[i - 1][j] == 1) {

dfs(mat, i - 1, j, n, path + 'U', res);

}

if (j + 1 < n && mat[i][j + 1] == 1) {

dfs(mat, i, j + 1, n, path + 'R', res);

}

if (j - 1 >= 0 && mat[i][j - 1] == 1) {

dfs(mat, i, j - 1, n, path + 'L', res)

}

mat[i][j] = 1;

}

};

**2.Check if two Strings are Anagrams of each other**  
  
class Solution {

public:

bool areAnagrams(string& s1, string& s2) {

const int MAX\_CHAR=26;

vector<int>freq(26,0);

for(char ch:s1){

freq[ch-'a']++;

}

for(char ch:s2){

freq[ch-'a']--;

}

for(int count:freq){

if(count!=0){

return false;

}

}

return true;

}

};  
  
**3.Find the row with maximum number of 1s**  
  
class Solution {

public:

int rowWithMax1s(vector<vector<int> > &arr) {

int n = arr.size();

int m = arr[0].size();

int maxRowInd = -1;

int j = m - 1;

for (int i = 0; i < n; i++) {

while (j >= 0 && arr[i][j] == 1) {

j--;

maxRowInd = i;

}

}

return maxRowInd;

}

};  
  
**4.Longest Consecutive Subsequence**

class Solution {

public:

int findLongestConseqSubseq(vector<int>& arr) {

int n=arr.size();

int ans=0;

unordered\_set<int>s;

for(int i=0;i<n;i++)

s.insert(arr[i]);

for(int i=0;i<n;i++){

if(s.find(arr[i]-1)==s.end()){

int j=arr[i];

while(s.find(j)!=s.end())

j++;

ans=max(ans,j-arr[i]);

}

}

return ans;

}

};  
  
**5.Longest Palindromic Substring**  
class Solution{

public:

string longestPalindrome(string S){

int n=S.length();

if(n==0) return "";

int start=0,maxlen=1;

for(int i=0;i<n;i++){

for (int j = 0; j <= 1; j++) {

int low=i;

int high=i+j;

while(low>=0 && high<n && S[low]==S[high]){

int currentlen=high-low+1;

if(currentlen >maxlen){

start=low;

maxlen=currentlen;

}

low--;

high++;

}

}

}

return S.substr(start,maxlen);

}

};