**// template**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const char nl= '\n';

#define all(x) x.begin(),x.end()

#define minimum(x) \*min\_element(x.begin(),x.end())

#define maximum(x) \*max\_element(x.begin(),x.end())

int32\_t main() {

    ios\_base::sync\_with\_stdio(false);

    cin.tie(nullptr);

    return 0;

}

**// normal sieve**

bool isPrime(int n) {

  if (n == 2 || n == 3) {

    return true;

  }

  if (n <= 1 || n % 2 == 0 || n % 3 == 0) {

    return false;

  }

  for (int i = 5; i \* i <= n; i += 6) {

    if (n % i == 0 || n % (i + 2) == 0) {

      return false;

    }

  }

  return true;

}

**// bitwise sieve**

bool Check(int N, int pos) { return (bool)(N & (1 << pos)); }

int Set(int N, int pos) { return N = N | (1 << pos); }

const int mx = 100000000;

int N = mx;

int status[(mx / 32) + 2];

vector<int> primes;

void sieve() {

    int i, j, sqrtN, cnt = 1;

    sqrtN = int(sqrt(N));

    for (i = 3; i <= sqrtN; i += 2) {

        if (Check(status[i >> 5], i & 31) == 0) {

            cnt++;

            for (j = i \* i; j <= N; j += (i << 1)) {

                status[j >> 5] = Set(status[j >> 5], j & 31);

            }

        }

    }

    primes.push\_back(2);

    for (i = 3; i <= N; i += 2) {

        if (Check(status[i >> 5], i & 31) == 0) {

            primes.push\_back(i);

        }

    }

}

**// bitset prime**

const int N=1e5+9;

bitset<N> is\_prime;

void sieve(){

is\_prime.set();

is\_prime[1]=0;

for(int i=2; i\*i<N; i++){

if(is\_prime[i]) {

for(int j=i\*i; j<N; j+=i) {

is\_prime[j]=0;

}

}

}

}

// main

vector<int>prime;

prime.pb(0);

prime.pb(2);

for(int i=3; i<N; i+=2) if(is\_prime[i]) prime.pb(i);

**// power section**

bool isPowerof(long long num, long long base) {

  if (num <= 0) return false;

  if (num % base == 0) {

    return isPowerof(num / base, base);

  }

  if (num == 1) return true;

  return false;

}

long long powmod(long long a, long long b, long long MOD) {

  long long res = 1;

  a %= MOD;

  assert(b >= 0);

  for (; b; b >>= 1) {

    if (b & 1) res = res \* a % MOD;

    a = a \* a % MOD;

  }

  return res;

}

long long binpow(long long a, long long b) {

  if (b == 0) return 1;

  long long res = binpow(a, b / 2);

  if (b % 2) return res \* res \* a;

  else return res \* res;

}

**// long multi,div,rem section**

int remainder(string str, int n) {

  int len = str.length();

  int num, rem = 0;

  for (int i = 0; i < len; i++) {

    num = rem \* 10 + (str[i] - '0');

    rem = num % n;

  }

  return rem;

}

string longDivision(string number, int divisor) {

  string ans;

  int idx = 0;

  int temp = number[idx] - '0';

  while (temp < divisor)

    temp = temp \* 10 + (number[++idx] - '0');

while (number.size() > idx) {

    ans += (temp / divisor) + '0';

    temp = (temp % divisor) \* 10 + number[++idx] - '0';

  }

  if (ans.length() == 0)

    return "0";

  return ans;

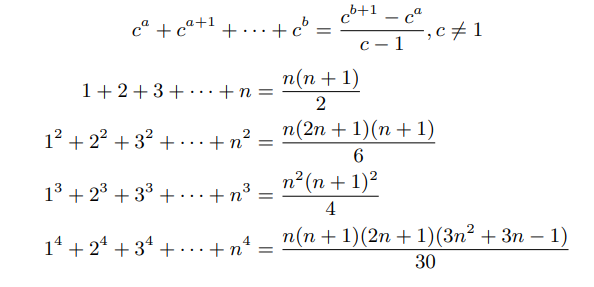
}

string multiply(string num1, string num2) {

    vector<int> vec1(num1.size());

    for (int i = 0; i < num1.size(); i++) {

        vec1[i] = num1[num1.size() - i - 1] - '0';

    }

    vector<int> vec2(num2.size());

    for (int i = 0; i < num2.size(); i++) {

        vec2[i] = num2[num2.size() - i - 1] - '0';

    }

    vector<int> result(vec1.size() + vec2.size());

    for (int i = 0; i < vec2.size(); i++) {

        int carry = 0;

        for (int j = 0; j < vec1.size(); j++) {

            int product = vec1[j] \* vec2[i] + carry + result[i + j];

            carry = product / 10;

            result[i + j] = product % 10;

        }

        result[i + vec1.size()] = carry;

    }

    while (result.size() > 1 && result.back() == 0) {

        result.pop\_back();

    }

    string str(result.size(), '0');

    for (int i = 0; i < result.size(); i++) {

        str[result.size() - i - 1] = result[i] + '0';

    }

    return str;

}

**// divisor**

vector<int> divs;

for (int i = 1; i \* i <= n; i++) {

  if (n % i == 0) {

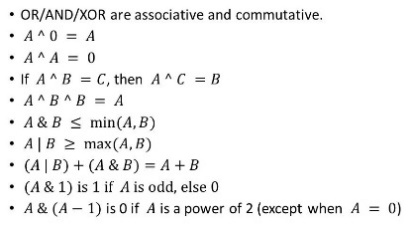
    divs.push\_back(i);

    if (i != n / i) divs.push\_back(n / i);

  }

}

**// math section**



**// hitting the bits**

n&(1<<k) -> Kth bit on or off

(n>>k)&1 -> kth bit on or off // needs modification

n|(a<<k) -> kth bit on

n&((1<<30)-1-(1<<k)) -> kth bit off

n&((1<<k)-1) -> last k bits on

\_\_builtin\_popcount(x)

\_\_builtin\_clz(x)

\_\_builtin\_ctz(x)

bool cmp(pair<int,int>a,pair<int,int>b){

if(a.first>b.first) return true;

else if(a.first==b.first){

if(a.second<=b.second) return true;

}

 return false;

}

int convert(long long n) {

  int dec = 0, i = 0, rem;

  while (n!=0) {

    rem = n % 10;

    n /= 10;

    dec += rem \* pow(2, i);

    ++i;

  }

  return dec;

}

string DecimalToBinary(int num) {

    string str;

      while(num){

      if(num & 1) // 1

        str+='1';

      else // 0

        str+='0';

      num>>=1; // Right Shift by 1

    }

      return str;

}

void reverse(string str) {

  for(int i=str.size()-1 ; i>=0 ; i--)

   cout<< str[i];

}

  int decimal, binary = 0, remainder, product = 1;

  cin >> decimal;

  while (decimal != 0) {

    remainder = decimal % 2;

    binary = binary + (remainder \* product);

    decimal = decimal / 2;

    product \*= 10;

  }

int binary\_search\_algo(int ar[],int size, int x) {

    int left=0, right= size-1, mid;

    while(left<=right) {

        mid=(left+right)/2;

        if(ar[mid]==x) return mid;

        if(ar[mid]<x) left=mid+1;

           else right=mid-1;

    }

    return -1;

}

bool isPowerOfTwo(ll n) {

    if (n == 0) return false;

    return (ceil(log2(n)) == floor(log2(n)));

}  // num&(num-1)

**// Graph**

**// dfs height and depth**

const int N = 1e5+5;

vector<int> g[N];

bool visited[N];

int depth[N];

int height[N];

void dfs(int source) {

visited[source] = true;

//cout << "Visiting node: " << u << endl;

for(int child: g[source]) {

if(visited[child] == true) continue;

depth[child] = depth[source] + 1;

dfs(child);

height[source] = max(height[source], height[child]+1);

}

}

**// bfs**

**bfs including shortest path printing**

const int N = 1e5+5;

vector<int>g[N];

bool visited[N];

int dis[N];

int par[N];

void bfs(int s) {

queue<int>q;

q.push(s);

dis[s] = 0;

par[s] = -1;

visited[s] = true;

while(!q.empty()) {

int parent = q.front();

q.pop();

for(int child : g[parent]) {

if(!visited[child]) {

q.push(child);

visited[child] = true;

par[child] = parent;

dis[child] = dis[parent] + 1;

}

}

}

}

int32\_t main() {

int n, e;

cin >> n >> e;

while (e--) {

int u, v;

cin >> u >> v;

g[u].push\_back(v);

g[v].push\_back(u); // undirected

}

bfs(1);

// for any specific node source

int source;

cin >> source;

if(visited[source]) {

vector<int>path;

int search = source;

while(search != -1) {

path.push\_back(search);

search = par[search];

}

reverse(path.begin(), path.end());

for(int val : path) cout<< val <<" ";

cout<<'\n';

}

else cout<<"NO PATH\n";

return 0;

}

**// bfs 2D**

using pii = pair<int,int>;

const int N = 1005;

bool vis[N][N];

int dis[N][N];

int n, m;

vector<pii> direction = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

bool isValid(int cI, int cJ) {

if (cI >= 0 && cI < n && cJ >= 0 && cJ < m) return true;

else return false;

}

void bfs(int si, int sj) {

queue<pii> q;

q.push({si, sj});

dis[si][sj] = 0;

vis[si][sj] = true;

while (!q.empty()) {

pii parent = q.front();

int pI = parent.first;

int pJ = parent.second;

q.pop();

for(pii p : direction) {

int cI = pI + p.first;

int cJ = pJ + p.second;

if (isValid(cI, cJ) && !vis[cI][cJ]) {

vis[cI][cJ] = true;

q.push({cI, cJ});

dis[cI][cJ] = dis[pI][pJ] + 1;

}

}

}

}

int32\_t main() {

cin >> n >> m;

char a[n][m];

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

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

cin >> a[i][j];

}

}

int si, sj;

cin >> si >> sj;

bfs(si, sj);

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

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

cout << dis[i][j] << " ";

}

cout << endl;

}

return 0;

}

// Dijkstra

using pii = pair<int,int> ;

const int N = 1e3+7;

const int INF = 1e9+7;

vector<pii> g[N];

vector<int> dist(N, INF);

vector<bool> visited(N);

vector<int>par(N,-1);

vector<int>path;

void dijkstra(int source) {

priority\_queue<pii, vector<pii>, greater<pii>> pq;

dist[source] = 0;

pq.push({0,source});

while(!pq.empty()) {

int parent = pq.top().second;

pq.pop();

visited[parent] = true;

for(pii child\_pair: g[parent]) {

int child\_node = child\_pair.first;

int child\_cost = child\_pair.second;

if(!visited[child\_node] and dist[child\_node] > dist[parent]+child\_cost) {

dist[child\_node] = dist[parent]+child\_cost;

pq.push({dist[child\_node],child\_node});

par[child\_node] = parent;

}

}

}

}

int32\_t main() {

int n,m;

cin >> n >> m;

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

int u,v,w;

cin >> u >> v >> w;

g[u].push\_back({v,w});

g[v].push\_back({u,w});

}

int source = 1;

dijkstra(source);

// all sources distance

for(int i=1;i<=n;i++) {

cout << "Distance of node " << i;

cout << " : " << dist[i] << endl;

}

// printing path from source to end

if(dist[n]==INF) {

cout<<"-1"<<'\n';

return 0;

}

else {

for(int parent=n; parent!=-1;parent=par[parent]) {

path.push\_back(parent);

}

reverse(path.begin(), path.end());

for(auto x : path) cout<<x<<" ";

}

return 0;

}

// bellman ford

int n, e;

class Edge {

public:

int u;

int v;

int w;

Edge(int u, int v, int w) {

this->u = u;

this->v = v;

this->w = w;

}

};

void bellman\_ford(int source,vector<Edge> &g) {

int dis[n + 1];

for (int i = 1; i <= n; i++) {

dis[i] = INT\_MAX;

}

dis[source] = 0;

for (int i = 1; i <= n - 1; i++) {

for (int j = 0; j < g.size(); j++) {

Edge ed = g[j];

int a = ed.u;

int b = ed.v;

int w = ed.w;

if (dis[a] + w < dis[b]) {

dis[b] = dis[a] + w;

}

}

}

for (int i = 1; i <= n; i++) {

cout << "Node " << i << ": " << dis[i] << endl;

}

}

**// negative cycle detection**

const int N = 1e5+7;

const ll INF = 1e18+7;

vector<pii> g[N];

vector<pair<pii, ll>> list\_of\_edges;

ll d[N];

int n,m;

bool cycle = false;

void bellman\_ford(int source) {

for(int i=1;i<=n;i++) d[i] = INF;

d[source] = 0;

for(int i=1;i<n;i++) {

for(auto edge: list\_of\_edges) {

ll u = edge.first.first;

ll v = edge.first.second;

ll w = edge.second;

if(d[u]!=INF && d[v] > d[u]+w)

d[v] = d[u] + w;

}

}

for(auto edge: list\_of\_edges) {

ll u = edge.first.first;

ll v = edge.first.second;

ll w = edge.second;

if(d[u]!=INF && d[v] > d[u]+w) {

cycle = true;

break;

}

}

}

// Floyd warshall

const int INF = 1e7;

int32\_t main() {

int n, e;

cin >> n >> e;

int dis[n + 1][n + 1];

for (int i = 1; i <= n; i++) {

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

dis[i][j] = INF;

if (i == j) dis[i][j] = 0;

}

}

while (e--) {

int a, b, w;

cin >> a >> b >> w;

dis[a][b] = w;

}

for (int k = 1; k <= n; k++) {

for (int i = 1; i <= n; i++) {

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

if (dis[i][k] + dis[k][j] < dis[i][j]) {

dis[i][j] = dis[i][k] + dis[k][j];

}

}

}

}

return 0;

}

**// cycle detect using dfs**

vector<int> g[N];

bool visited[N];

bool dfs(int source, int p=-1) {

bool cycleExists = false;

visited[source] = true;

for(int child: g[source]) {

if(child == p) continue;

if(visited[child]) return true;

cycleExists = cycleExists | dfs(child,source);

}

return cycleExists;

}

for(int i=1;i<=n;i++) {

if(visited[i]) continue;

isCycle |= dfs(i);

}

// MST prims

using pi = pair<int,int>;

const int N = 1e5 + 5;

vector<pi> g[N];

bool vis[N];

class Edge {

public:

int a, b, w;

Edge(int a, int b, int w) {

this->a = a;

this->b = b;

this->w = w;

}

};

class cmp {

public:

bool operator()(Edge a, Edge b) {

return a.w > b.w;

}

};

void prims(int s) {

priority\_queue<Edge, vector<Edge>, cmp> pq;

vector<Edge> edgeList;

pq.push(Edge(s, s, 0));

while (!pq.empty()) {

Edge parent = pq.top();

pq.pop();

int a = parent.a;

int b = parent.b;

int w = parent.w;

if(!vis[b]) {

vis[b] = true;

edgeList.push\_back(parent);

for (int i = 0; i < g[b].size(); i++) {

pi child = g[b][i];

if (!vis[child.first]) {

pq.push(Edge(b, child.first, child.second));

}

}

}

}

edgeList.erase(edgeList.begin());

for (Edge val : edgeList) {

cout << val.a << " " << val.b << " " << val.w << endl;

}

}

// undirected cycle detection using dsu

#include <bits/stdc++.h>

using namespace std;

const int N = 1e5+5;

int parent[N];

int parentSize[N]; // size based dsu

int parentLevel[N]; // rank based dsu

void dsu\_set(int n) {

for (int i = 1; i <= n; i++) {

parent[i] = -1;

parentSize[i] = 1;

parentLevel[i] = 0;

}

}

int dsu\_find(int node) {

while (parent[node] != -1) {

node = parent[node];

}

return node;

}

void dsu\_union\_using\_size(int a, int b) {

int leaderA = dsu\_find(a);

int leaderB = dsu\_find(b);

if (leaderA != leaderB) {

if (parentSize[leaderA] > parentSize[leaderB]) {

// A leader

parent[leaderB] = leaderA;

parentSize[leaderA] += parentSize[leaderB];

}

else {

parent[leaderA] = leaderB;

parentSize[leaderB] += parentSize[leaderA];

}

}

}

void dsu\_union\_using\_rank(int a, int b) {

int leaderA = dsu\_find(a);

int leaderB = dsu\_find(b);

if (leaderA != leaderB) {

if (parentLevel[leaderA] > parentLevel[leaderB]) {

parent[leaderB] = leaderA;

}

else if (parentLevel[leaderB] > parentLevel[leaderA]) {

parent[leaderA] = leaderB;

}

else {

parent[leaderB] = leaderA;

parentLevel[leaderA]++;

}

}

}

int32\_t main() {

int n, e,cnt=0;

cin >> n >> e;

dsu\_set(n);

while (e--) {

int a, b;

cin >> a >> b;

int leaderA = dsu\_find(a);

int leaderB = dsu\_find(b);

if (leaderA == leaderB) cnt++; // cycle detection and counting

else dsu\_union\_using\_size(a, b);

// else dsu\_union\_using\_rank(a,b);

}

return 0;

}

**// DP**

**// fibo**

ll save[N];

ll fibo(ll n) {

if (n == 0 || n == 1) return 1;

if (save[n] != -1) return save[n];

return save[n] = fibo(n - 1) + fibo(n - 2);

}

**// 0-1 Knapsack (Bounded)**

int dp[n + 1][s + 1];

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

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

if (i == 0 || j == 0) dp[i][j] = 0;

}

}

for (int i = 1; i <= n; i++) {

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

if (w[i - 1] <= j) dp[i][j] = max(v[i - 1] + dp[i - 1][j - w[i - 1]], dp[i - 1][j]);

else dp[i][j] = dp[i - 1][j];

}

}

**// unbounded** dp[i][j] = max(val[i - 1] + dp[i][j - w[i - 1]], dp[i - 1][j]);

**// subset sum**

bool dp[n + 1][s + 1];

dp[0][0] = true;

for (int i = 1; i <= s; i++) dp[0][i] = false;

for (int i = 1; i <= n; i++) {

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

if (a[i - 1] <= j) {

dp[i][j] = dp[i - 1][j - a[i - 1]] || dp[i - 1][j]; // for counting +

}

else dp[i][j] = dp[i - 1][j];

}

}

//

// minimum subset sum difference

int n;

cin >> n;

int a[n], s = 0;

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

cin >> a[i];

s += a[i];

}

bool dp[n + 1][s + 1];

dp[0][0] = true;

for (int i = 1; i <= s; i++) dp[0][i] = false;

for (int i = 1; i <= n; i++) {

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

if (a[i - 1] <= j) {

dp[i][j] = dp[i - 1][j - a[i - 1]] || dp[i - 1][j];

}

else dp[i][j] = dp[i - 1][j];

}

}

vector<int> v;

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

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

if (dp[i][j] == 1) v.push\_back(j);

}

}

int ans = INT\_MAX;

for (int val : v) {

int s1 = val;

int s2 = s - s1;

ans = min(ans, abs(s1 - s2));

}

cout << ans << endl;

return 0;

}

// Count Subset Sum with Given Difference

/\*

s1 = (dif+sum)/2;

s1%2==0 true

just count subset sum of s1

same is target sum

equal some same total array sum /2🡪 find the subset sum

\*/

**coin change 1**

int dp[n + 1][s + 1];

dp[0][0] = 1;

for (int i = 1; i <= s; i++) dp[0][i] = 0;

for (int i = 1; i <= n; i++) {

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

if (w[i - 1] <= j) dp[i][j] = dp[i][j - w[i - 1]] + dp[i - 1][j];

else dp[i][j] = dp[i - 1][j];

}

}

**// coin change 2**

int dp[n + 1][s + 1];

dp[0][0] = 0;

for (int i = 1; i <= s; i++) dp[0][i] = INT\_MAX - 1;

for (int i = 1; i <= n; i++) {

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

if (w[i - 1] <= j) dp[i][j] = min(1 + dp[i][j - w[i - 1]], dp[i - 1][j]);

else dp[i][j] = dp[i - 1][j];

}

}

if (dp[n][s] == INT\_MAX - 1) cout << "Not Possible" << endl;

else cout << dp[n][s] << endl;

**// LCS**

string a, b;

cin >> a >> b;

int n = a.size(), m = b.size();

int dp[n + 1][m + 1];

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

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

if (i == 0 || j == 0) dp[i][j] = 0;

}

}

for (int i = 1; i <= n; i++) {

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

if (a[i - 1] == b[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

}

else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

}

}

cout << dp[n][m] << endl;

// minimum insertion and deletion to make b from a --> delete = a.size() - dp[n][m] insert = b.size() - dp[n][m]

// print the lcs

int i = n, j = m;

string ans;

while (i != 0 && j != 0) {

if (a[i - 1] == b[j - 1]) {

ans += a[i - 1];

i--;

j--;

}

else {

if (dp[i][j - 1] > dp[i - 1][j]) j--;

else i--;

}

}

reverse(ans.begin(), ans.end());

cout << ans << endl;

}

**// LPS**

string a;

cin >> a;

int n = a.size();

string b = a;

reverse(b.begin(), b.end());

int m = b.size();

int dp[n + 1][m + 1];

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

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

if (i == 0 || j == 0) dp[i][j] = 0;

}

}

for (int i = 1; i <= n; i++) {

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

if (a[i - 1] == b[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

}

else {

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

// minimum deletion and insertion to make palindrome--> delete = b - dp[n][m] insert = b - dp[n][m]

int i = n, j = m;

string ans;

while (i != 0 && j != 0) {

if (a[i - 1] == b[j - 1]) {

ans += a[i - 1];

i--;

j--;

}

else {

if (dp[i][j - 1] > dp[i - 1][j]) j--;

else i--;

}

}

reverse(ans.begin(), ans.end());

cout << ans << endl;

}

**// shortest common supersequence**

string a, b;

cin >> a >> b;

int n = a.size();

int m = b.size();

int dp[n + 1][m + 1];

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

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

if (i == 0 || j == 0) dp[i][j] = 0;

}

}

for (int i = 1; i <= n; i++) {

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

if (a[i - 1] == b[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

}

else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]); }

}

cout << n + m - dp[n][m] << endl; // scs size

// print section

int i = n, j = m;

string ans;

while (i != 0 && j != 0) {

if (a[i - 1] == b[j - 1]) {

ans += a[i - 1];

i--;

j--;

}

else if (dp[i - 1][j] > dp[i][j - 1]) {

ans += a[i - 1];

i--;

}

else {

ans += b[j - 1];

j--;

}

}

while (i != 0) {

ans += a[i - 1];

i--;

}

while (j != 0) {

ans += b[j - 1];

j--;

}

reverse(ans.begin(), ans.end());

cout << ans << endl;

}

**// longest common substring**

string a, b;

cin >> a >> b;

int n = a.size(), m = b.size();

int dp[n + 1][m + 1];

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

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

if (i == 0 || j == 0) dp[i][j] = 0;

}

}

for (int i = 1; i <= n; i++) {

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

if (a[i - 1] == b[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

}

else {

dp[i][j] = 0;

}

}

}

int mx = 0, ci, cj;

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

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

if (dp[i][j] > mx){

mx = dp[i][j];

ci = i;

cj = j;

}

}

}

string ans;

while (ci != 0 && cj != 0) {

if (a[ci - 1] == b[cj - 1]) {

ans += a[ci - 1];

ci--;

cj--;

}

else break; }

reverse(ans.begin(), ans.end());

cout << ans << endl;

}

**// DEBUG**

divisible by 0

array bound

constraints

pre calculation

generate max and min cases

check TLE

check overflow

anything uninitialized

vector might fail

check consecutive case

check output format whitespace

Check all cases provided in problem

Check for overflow (long long case)

TLE:

Check constraint

how many times are you using loop? or nested loop?

If it is a math problem and you are getting TLE then loop won't work. Check for a equation

Runtime error:

Check if you are doing division by zero

array out of bound (al)

Check if you are running loop outside your array limit or string limit

Wrong Answer:

Check for constraint. Are you thinking for the highest constraint?

Does it need long long?

Are you checking for the lowest constraint?

Any corner case you are not thinking?

Facing overflow? check if needs long long

Garbage? Any uninitialized varible you are using?

Presentation error:

Use newline after output