**Sintaxis básica de Python**

# Es importante tener en cuenta que python es aproximadamente 40 veces mas lento que C++

import array as arr # Para trabajar con arrays

import numpy as np

def main ():

    # Leer una linea completa, separar los elementos por los espacios y almacenarlos en una lista

    read = input()

    numbers = read.split()

    numbers = list(map(int, numbers)) # 'map' si permite repeticiones, en cambio 'set' no

    #Leer un entero

    n = int(input())

    # Leer un número flotante

    d = float(input())

    # Leer una cadena

    s = input()

    # Tamaño de una cadena

    x = len(s)

    # Agregar un caracter a una cadena

    s2 += s1[i]

    s2 += 'a'

    # Negar un bool

    cond = True

    not cond # Esto seria False

    # Potencia de un numero

    ans = base \*\* exp

    # Ciclo for

    for i in range (0, 5): # Esto iria hasta el 4

    for auto in autos: # autos seria una lista

    # Condicionales

    if(cond1):

    elif(cond2):

    else:

    # Minimo y maximo

    mina = min(mina, number)

    maxa = max(maxa, number)

    # Convertir un numero binario (guardado como cadena) en un entero base 10

    n10 = int(n2, 2)

    # Pasar un numero de base 10 a base 2 (se convierte a cadena)

    ans2 = bin(ans10)[2:]

    # Evaluar una string que es una expresion matematica (tambien puede tener parentesis)

    result = eval(expresion)

    # Metodos de listas (list)

    # Resaltar que una sola lista puede contener varios tipos de datos, incluida otra lista

    lista = [1, "dos", [3, "cuatro"], 5.0, True] # Declarar

    lista.append(6) # Agregar un elemento

    lista.remove('dos') # Eliminar un elemento

    lista.sort() # Ordenar

    print(lista) # Imprimir toda la lista

    lista.extend([7, 8])  # Agrega 7 y 8 a la lista

    lista.insert(1, 'a') # Inserto 'a' en la posicion 1, el resto de elementos corren un espacio a la derecha

    n = lista.pop() # Elimina el ultimo elemento de la lista y se lo asigna a 'n'

    n = lista.pop(3) # Elimina el ultimo en la posicion 3 de la lista y se lo asigna a 'n'

    lista.clear() # Elimina todos los elementos la lista, quedando: lista = []

    lista.reverse # Invierte los elementos de la lista

    indice = lista.index("cuatro") # Devuelve el indice de la primera aparicion de "cuatro"

    indice2 = lista.index(x, start) # Comienza a buscar desde el indice start

    indice3 = lista.index(x, start, end) # Devuelve la primera aparicion de x en el rango [start, end)

    conteo = lista.count(x) # Devuelve el numero de veces que aparece 'x' en la lista

    # Metodos de sets

    # No tiene elementos repetidos, no esta ordenado

    s = {1, 2, 3} # Declarar

    s.add(4) # Agregar un elemento

    s.remove(4) # Eliminar un elemento. Si el elemento no esta lanza un error

    s.discard(4) # Tambien elimina un elemento pero si no esta no genera error

    x = s.pop() # Elimina y retorna un elemento arbitrario

    s.clear() # Vacía el conjunto

    s1 = {1, 2}

    s2 = {2, 3}

    s3 = {4, 5, 6}

    # Metodos de operaciones de conjuntos

    # Union de conjuntos

    s4 = s1.union(s2, s3)

    s4 = s1 | s2 | s3

    # Interseccion de conjuntos

    s4 = s1.instersection(s2, s3)

    s4 = s1 & s2 & s3

    # Retorna lo que esta en el primer conjunto pero no en los demas

    s4 = s1.difference(s2, s3)

    s4 = s1 - s2 - s3

    # Los elementos que solo estan en un conjunto

    # Se recomienda usarla solo para dos conjuntos

    s4 = s1.symmetric\_difference(s2).symmetric\_difference(s3)

    s4 = (s1 ^ s2) ^ s3

    # Metodos para relaciones de conjuntos

    s1.issubset(s2) # Devuelve True si s1 es subconjunto de s2

    s1.issuperset(s2) # Devuelve True si s1 es superconjunto de s2

    s1.isdisjoint(s2) # Devuelve True si no tienen elementos en comun

    # Metodos de actualizacion

    # Actualiza s1 con la union entre s1 y s2

    s1.update(s2)

    s1 |= s2

    # Actualiza s1 con la interseccion entre s1 y s2

    s1.intersection\_update(s2)

    s1 &= s2

    # Actualiza s1 eliminando los elementos que estan en s2

    s1.difference\_update(s2)

    s1 -= s2

    # Actualiza s1 con la diferencia simetrica entre s1 y s2

    s1.symmetric\_difference\_update(s2)

    s1 ^= s2

    # SortedSet puede hacer todo eso, pero esta ordenado y tambien tiene:

    # pip install sortedcontainers -> Instalacion

    from sortedcontainers import SortedSet

    s = SortedSet([1, 2, 3, 4]) # Declarar

    mini = s.pop() # Retorna el elemento menor

    cond = 2 in s # Retorna True si s esta en s

    i = s.index(2) # Retorna la posicion de 2 (0-index)

    s.update([4, 5, 7]) # Agrega elementos

    s.bisect\_left(6) # Retorna el indice en que se insertaria el 6

    s.bisect\_right(3) # Lo mismo que bisect\_left pero si el elemento ya

                      # esta en s entonces devuelve su posicion + 1

main()

**Caras de un grafo**

// Estructura Point con campos "x" y "y" y operaciones entre puntos

struct Point {

    ll x, y;

    Point(ll x\_, ll y\_): x(x\_), y(y\_) {}

    bool operator < (const Point& p) const {

        return tie(x, y) < tie(p.x, p.y);

    }

    Point operator - (const Point& p) const {

        return Point(x - p.x, y - p.y);

    }

    ll cross (const Point & p) const {

        return x \* p.y - y \* p.x;

    }

    ll cross (const Point & p, const Point & q) const {

        return (p - \*this).cross(q - \*this);

    }

    ll half () const {

        return ll(y < 0 || (y == 0 && x < 0));

    }

};

//Hallar las caras de un grafo plano

vector<vector<ll>> find\_faces(vector<Point> vertices, vector<vector<ll>> adj) {

    ll n = vertices.size();

    vector<vector<char>> used(n);

    for (ll i = 0; i < n; i++) {

        used[i].resize(adj[i].size());

        used[i].assign(adj[i].size(), 0);

        auto compare = [&](ll l, ll r) {

            Point pl = vertices[l] - vertices[i];

            Point pr = vertices[r] - vertices[i];

            if (pl.half() != pr.half())

                return pl.half() < pr.half();

            return pl.cross(pr) > 0;

        };

        sort(adj[i].begin(), adj[i].end(), compare);

    }

    vector<vector<ll>> faces;

    for (ll i = 0; i < n; i++) {

        for (ll edge\_id = 0; edge\_id < adj[i].size(); edge\_id++) {

            if (used[i][edge\_id]) {

                continue;

            }

            vector<ll> face;

            ll v = i;

            ll e = edge\_id;

            while (!used[v][e]) {

                used[v][e] = true;

                face.push\_back(v);

                ll u = adj[v][e];

                ll e1 = lower\_bound(adj[u].begin(), adj[u].end(), v, [&](ll l, ll r) {

                    Point pl = vertices[l] - vertices[u];

                    Point pr = vertices[r] - vertices[u];

                    if (pl.half() != pr.half())

                        return pl.half() < pr.half();

                    return pl.cross(pr) > 0;

                }) - adj[u].begin() + 1;

                if (e1 == adj[u].size()) {

                    e1 = 0;

                }

                v = u;

                e = e1;

            }

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

            ll sign = 0;

            for (ll j = 0; j < face.size(); j++) {

                ll j1 = (j + 1) % face.size();

                ll j2 = (j + 2) % face.size();

                ll val = vertices[face[j]].cross(vertices[face[j1]], vertices[face[j2]]);

                if (val > 0) {

                    sign = 1;

                    break;

                } else if (val < 0) {

                    sign = -1;

                    break;

                }

            }

            if (sign <= 0) {

                faces.insert(faces.begin(), face);

            } else {

                faces.emplace\_back(face);

            }

        }

    }

    return faces;

}

int main(){

    ios\_base::sync\_with\_stdio(0);cin.tie(NULL);

    // Nos van a dar n aristas

    ll n; cin>>n;

    // Almacenamiento de vertices y aristas para llamar el algoritmo de findFaces

    vector<Point> vertices;

    vector<vector<ll>> adj;

    map<Point, ll> appear;

    for(ll i = 0; i < n; i++){

        ll x1, y1, x2, y2;

        cin>>x1>>y1>>x2>>y2;

        Point save1(x1, y1), save2(x2, y2);

        ll index1, index2;

        map<Point, ll>::iterator it1, it2;

        it1 = appear.find(save1);

        it2 = appear.find(save2);

        if(it1 == appear.end()) {

            index1 = appear[save1] = vertices.size();

            vertices.pb(save1);

            adj.emplace\_back();

        }

        else

            index1 = it1->second;

        if(it2 == appear.end()) {

            index2 = appear[save2] = vertices.size();

            vertices.pb(save2);

            adj.emplace\_back();

        }

        else

            index2 = it2->second;

        adj[index1].pb(index2);

        adj[index2].pb(index1);

    }

    vector<vector<ll>> faces = find\_faces(vertices, adj);

    return 0;

}

**Tarjan**

/\* Complexity: O(E + V)

Tarjan’s algorithm for finding strongly connected

components.

\*d[i] = Discovery time of node i. (Initialize to -1)

\*low[i] = Lowest discovery time reachable from node

i. (Doesn’t need to be initialized)

\*scc[i] = Strongly connected component of node i. (Doesn’t

need to be initialized) (0-index)

\*s = Stack used by the algorithm (Initialize to an empty

stack)

\*stacked[i] = True if i was pushed into s. (Initialize to

false)

\*ticks = Clock used for discovery times (Initialize to 0)

\*current\_scc = ID of the current\_scc being discovered

(Initialize to 0)

\*/

const int MAXN = 1e5 + 1;

const ll mod = 1e9 + 7;

vector<int> g[MAXN];

vector<int> d(MAXN, -1), low(MAXN), scc(MAXN), reps(MAXN, 0);

vector<bool> stacked(MAXN);

stack<int> s;

int ticks = 0, current\_scc = 0;

void tarjan(int u){

    d[u] = low[u] = ticks++;

    s.push(u);

    stacked[u] = true;

    const vector<int> &out = g[u];

    for (int k=0, m=out.size(); k<m; ++k){

        const int &v = out[k];

        if (d[v] == -1){

            tarjan(v);

            low[u] = min(low[u], low[v]);

        }

        else if (stacked[v])

            low[u] = min(low[u], low[v]);

    }

    if (d[u] == low[u]){

        int v;

        do{

            v = s.top();

            s.pop();

            stacked[v] = false;

            scc[v] = current\_scc;

        }while (u != v);

        current\_scc++;

    }

}

void solver(){

    int n, m; cin>>n>>m;

    for(int i = 1; i <= m; i++){

        int u, v;cin>>u>>v;

        g[u].pb(v);

    }

    // Hallar los SCC

    for(int i = 1; i <= n; i++){

        if(d[i] == -1)

            tarjan(i);

    }

    cout<<current\_scc<<endl;

}

**DSU**

const int maxn = 200005;

int components;

vector<int> graph[maxn];

vector<int> leader(maxn);

vector<int> sets[maxn];

void initDSU(int n){

    components = n;

    for(int i = 1; i <= n; i++){

        leader[i] = i;

        sets[i].push\_back(i);

    }

}

void join(int u, int v){

    int leaderU = leader[u], leaderV = leader[v];

    if(leaderU != leaderV){

        if(sets[leaderV].size() > sets[leaderU].size())

            swap(leaderU, leaderV);

        for(int i = 0; i < sets[leaderV].size(); i++){

            int v = sets[leaderV][i];

            leader[v] = leaderU;

            sets[leaderU].push\_back(v);

        }

        sets[leaderV].clear();

        components--;

    }

}

int main(){

    ios\_base::sync\_with\_stdio(0);cin.tie(NULL);

    int n, m;cin>>n>>m;

    initDSU(n); //Inicializar DSU

    for(int i = 0; i < m; i++){

        int u, v;cin>>u>>v;

        graph[u].push\_back(v);

        graph[v].push\_back(u);

        join(u, v); //Unir vertice u con v

    }

    cout<<components<<endl;

    return 0;

}

**DSU Compresión de caminos**

void initDSU(int n){

    components = n;

    for(int i = 1; i <= n; i++){

        leader[i] = i;

        sizen[i] = 1;

    }

}

int find(int u){

    if(leader[u] != u)

        leader[u] = find(leader[u]);

    return leader[u];

}

void join(int u, int v){

    int leaderU = find(u), leaderV = find(v);

    if(leaderU != leaderV){

        if(sizen[leaderV] > sizen[leaderU])

            swap(leaderU, leaderV);

        leader[leaderV] = leaderU;

        sizen[leaderU] += sizen[leaderV];

        components--;

    }

}

void updateAllLeaders(int n){

    for(int i = 1; i <= n; i++)

        find(i);

}

**Read Maze**

const int maxn = 1e3+5;

string maze[maxn];

void solver(){

    int n, m; cin>>n>>m; // n rows, m columns

    /\* En este caso, queremos que el laberinto sea 0-index

    por lo tanto, llenaremos los bordes con "-" \*/

    string add;

    for(int i = 0; i <= m + 1; i++)add += '-';

    maze[0] = add;

    for(int j = 1; j <= n; j++) {

        string aux; cin>>aux;

        maze[j] = "-" + aux + "-";

    }

    maze[n+1] = add;

    /\* En este caso el laberinto sera 0-index y tendra una source identificada como '@' \*/

    for(int j = 0; j < n; j++) {

        string aux; cin>>aux;

        for(int i = 0; i < m; i++){

            if(aux[i] == '@'){

                sx = i;

                sy = j;

                aux[i] = '.';

            }

        }

        maze[j] = aux;

    }

}

**BFS Mazes**

typedef pair<int, int> pii;

const int maxn = 1e3+5, inf = 2e9;

void BFS(int sx, int sy, int n, int m, string maze[], int d[][maxn], bool visited[][maxn]){

    d[sy][sx] = 0;

    visited[sy][sx] = 1;

    queue<pii> q;

    q.push({sy, sx});

    while(!q.empty()){

        pii u = q.front(); q.pop();

        int j = u.first, i = u.second;

        if(i - 1 >= 0 && !visited[j][i-1] && maze[j][i-1] != '#'){

            visited[j][i-1] = 1;

            d[j][i-1] = d[j][i] + 1;

            q.push({j, i-1});

        }

        if(i + 1 < m && !visited[j][i+1] && maze[j][i+1] != '#'){

            visited[j][i+1] = 1;

            d[j][i+1] = d[j][i] + 1;

            q.push({j, i+1});

        }

        if(j - 1 >= 0 && !visited[j-1][i] && maze[j-1][i] != '#'){

            visited[j-1][i] = 1;

            d[j-1][i] = d[j][i] + 1;

            q.push({j-1, i});

        }

        if(j + 1 < n && !visited[j+1][i] && maze[j+1][i] != '#'){

            visited[j+1][i] = 1;

            d[j+1][i] = d[j][i] + 1;

            q.push({j+1, i});

        }

    }

}

void solver(){

    string maze[maxn];

    int d[maxn][maxn];

    bool visited[maxn][maxn];

    int n, m, sx, sy; cin>>n>>m;

    for(int j = 0; j < n; j++) {

        string aux; cin>>aux;

        for(int i = 0; i < m; i++){

            if(aux[i] == '@'){

                sx = i;

                sy = j;

                aux[i] = '.';

            }

        }

        maze[j] = aux;

    }

    for(int j = 0; j <= n + 1; j++){

        for(int i = 0; i <= m + 1; i++){

            d[j][i] = inf;

            visited[j][i] = 0;

        }

    }

    BFS(sx, sy, n, m, maze, d, visited);

    for(int j = 0; j < n; j++){

        for(int i = 0; i < m; i++)

            cout<<d[j][i]<<'\t';

        cout<<endl;

    }

    for(int j = 0; j < n; j++){

        for(int i = 0; i < m; i++)

            cout<<visited[j][i]<<'\t';

        cout<<endl;

    }

}

**DFS**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define MAXH 100

#define MAXW 100

#define WHITE 2

#define GRAY 1

#define BLACK 0

#define myInfinite 2147483647

#define NIL -1

struct cell

{

    int coord\_x;

    int coord\_y;

};

struct cell readMaze (char maze[][MAXW + 1], int W, int H) {

    char line [MAXW + 1];

    int idRow, idColumn;

    struct cell source;

    for (idRow = 1; idRow <= H; idRow++) {

        scanf ("%s", line);

        for (idColumn = 1; idColumn <= W; idColumn++) {

            maze[idRow][idColumn] = line[idColumn - 1];

            if (line[idColumn - 1] == '@') {

                source.coord\_x = idColumn;

                source.coord\_y = idRow;

                maze[idRow][idColumn] = '.';

            }

        }

    }

    return source;

}

void printMaze (char maze[][MAXW + 1], int W, int H) {

    int idRow, idColumn;

    printf ("\nThe original maze without source position:\n\n");

    for (idRow = 1; idRow <= H; idRow++) {

        for (idColumn = 1; idColumn <= W; idColumn++)

            printf ("%c", maze[idRow][idColumn]);

        printf ("\n");

    }

    printf ("\n");

}

void initializerMovements (struct cell movements[]) {

    movements[0].coord\_x = 0;

    movements[0].coord\_y = 0;

    movements[1].coord\_x = 0;

    movements[1].coord\_y = -1;

    movements[2].coord\_x = 0;

    movements[2].coord\_y = 1;

    movements[3].coord\_x = 1;

    movements[3].coord\_y = 0;

    movements[4].coord\_x = -1;

    movements[4].coord\_y = 0;

}

void DFS\_Visit (char maze[][MAXW + 1], int W, int H, struct cell u, int \*time, int d[][MAXW + 1],

                int f[][MAXW + 1], int color[][MAXW + 1], struct cell pi[][MAXW + 1])

{

    int idMovement;

    struct cell movements[5], v;

    initializerMovements (movements);

    color[u.coord\_y][u.coord\_x] = GRAY;

    (\*time)++;

    d[u.coord\_y][u.coord\_x] = \*time;

    for (idMovement = 1; idMovement <= 4; idMovement++) {

        v.coord\_x = u.coord\_x + movements[idMovement].coord\_x;

        v.coord\_y = u.coord\_y + movements[idMovement].coord\_y;

        if ((v.coord\_x >= 1 && v.coord\_x <= W) && (v.coord\_y >= 1 && v.coord\_y <= H) && (maze[v.coord\_y][v.coord\_x] == '.' && color[v.coord\_y][v.coord\_x] == WHITE)) {

            pi[v.coord\_y][v.coord\_x] = u;

            DFS\_Visit (maze, W, H, v, &(\*time), d, f, color, pi);

        }

    }

    color[u.coord\_y][u.coord\_x] = BLACK;

    (\*time)++;

    f[u.coord\_y][u.coord\_x] = \*time;

}

void DFS (char maze[][MAXW + 1], int W, int H, int d[][MAXW + 1],

            int f[][MAXW + 1], int color[][MAXW + 1], struct cell pi[][MAXW + 1])

{

    int time = 0, idRow, idColumn;

    struct cell NILFather, u;

    NILFather.coord\_x = NIL;

    NILFather.coord\_y = NIL;

    for (idRow = 1; idRow <= H; idRow++) {

        for (idColumn = 1; idColumn <= W; idColumn++) {

            color[idRow][idColumn] = WHITE;

            pi[idRow][idColumn] = NILFather;

        }

    }

    for (idRow = 1; idRow <= H; idRow++) {

        for (idColumn = 1; idColumn <= W; idColumn++) {

            if (maze[idRow][idColumn] == '.' && color[idRow][idColumn] == WHITE) {

                u.coord\_x = idColumn;

                u.coord\_y = idRow;

                DFS\_Visit (maze, W, H, u, &time, d, f, color, pi);

            }

        }

    }

}

void solver (char maze[][MAXW + 1], int W, int H) {

    int color[MAXH + 1][MAXW + 1], d[MAXH + 1][MAXW + 1], f[MAXH + 1][MAXW + 1], idRow, idColumn;

    struct cell pi[MAXH + 1][MAXW + 1];

    DFS (maze, W, H, d, f, color, pi);

    printf("Matrix of colors:\n\n");

    for(idRow = 1; idRow <= H; idRow++)

    {

        for(idColumn=1; idColumn <= W; idColumn++)

        {

            if(color[idRow][idColumn] == WHITE)

                printf(" W");

            if(color[idRow][idColumn] == GRAY)

                printf(" G");

            if(color[idRow][idColumn] == BLACK)

                printf(" B");

        }

        printf("\n");

    }

    printf("\n");

    printf("Matrix of time:\n\n");

    for(idRow = 1; idRow <= H; idRow++)

    {

        for(idColumn=1; idColumn <= W; idColumn++)

            printf ("%d %d\t\t", d[idRow][idColumn], f[idRow][idColumn]);

        printf("\n");

    }

    printf("\n");

    printf("Matrix of fathers:\n\n");

    for(idRow=1; idRow<=H; idRow++)

    {

        for(idColumn=1; idColumn<=W; idColumn++)

        {

            if(pi[idRow][idColumn].coord\_x == NIL)

                printf(" [ -1, -1]");

            else

            {

                if(pi[idRow][idColumn].coord\_x < 10)

                    printf(" [  %d,", pi[idRow][idColumn].coord\_x);

                else

                    printf(" [ %d,", pi[idRow][idColumn].coord\_x);

                if(pi[idRow][idColumn].coord\_y < 10)

                    printf("  %d]", pi[idRow][idColumn].coord\_y);

                else

                    printf(" %d]", pi[idRow][idColumn].coord\_y);

            }

        }

        printf("\n");

    }

    printf("\n");

}

int main () {

    char maze[MAXH + 1][MAXW + 1];

    int T, W, H, idCase;

    struct cell source;

    scanf ("%d", &T);

    for (idCase = 1; idCase <= T; idCase++) {

        scanf ("%d %d", &W, &H);

        source = readMaze (maze, W, H);

        printMaze (maze, W, H);

        solver (maze, W, H);

    }

    return 0;

}

**Polygon Area**

// Estructura Point con campos "x" y "y" y operaciones entre puntos

struct Point {

ll x, y;

Point(ll x\_, ll y\_): x(x\_), y(y\_) {}

bool operator < (const Point& p) const {

return tie(x, y) < tie(p.x, p.y);

}

Point operator - (const Point& p) const {

return Point(x - p.x, y - p.y);

}

ll cross (const Point & p) const {

return x \* p.y - y \* p.x;

}

ll cross (const Point & p, const Point & q) const {

return (p - \*this).cross(q - \*this);

}

ll half () const {

return ll(y < 0 || (y == 0 && x < 0));

}

};

//P es un polígono ordenado anticlockwise.

//Si es clockwise, retorna el area negativa.

//P[0] != P[n-1]

ld PolygonArea(const vector<Point> &p){

ld r = 0.0;

for (ll i=0; i<p.size(); ++i){

ll j = (i+1) % p.size();

r += p[i].x\*p[j].y - p[j].x\*p[i].y;

}

return r/2.0;

}