Tiempo

Dicen que en invierno hace frío y en verano calor. En el Primer Observatorio de la Temperatura Español (POTE) decimos que un día es estable si la diferencia entre la temperatura máxima y mínima es de solo 5 grados o menos. POTE considera que un día frío es un día estable con mínima menor de 7 grados. Finalmente POTE define día caluroso como día estable con temperatura máxima mayor de 28 grados.
POTE ofrece 3 datos por día correspondiendo a la temperatura de la mañana, la de la tarde y la de la noche. Dada la información de 7 días queremos un resumen de cómo ha sido la semana según POTE.

Estrategia de la solución

En este problema, tenemos que buscar el mínimo y máximo de tres números y analizar qué pasa con su diferencia. Este proceso lo tendremos que repetir 7 veces y contar que parte de los datos cumplen las condiciones del problema.
La primera condición es que el máximo - mínimo sea menor o igual a 5.
Una vez sepamos que es un día estable miraremos si el mínimo es menor que 7 o el máximo mayor que 28.
Por cada día que cumpla estos requisitos, sumamos uno a una variable (por cada tipo de día), y una vez finalizado los cálculos podemos imprimir los resultados.

Solución en Python

```
ncases = int(input())

for case in range(ncases):
    dcal = 0
    dfrio = 0
    dest = 0

    for day in range(7):
        ma = int(input())
        ta = int(input())
        no = int(input())

        mx = max(ma, ta, no)
        mn = min(ma, ta, no)

        if (mx - mn <= 5):
                dest = dest + 1
                if (mn < 7): dfrio = dfrio + 1
                if (mx > 28): dcal = dcal + 1

    print(dest, dfrio, dcal)
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>

using namespace std;

int main() {

    int num;
    cin>>num;
    for (int caso=0; caso < num; caso++) {

        int frio=0, caliente=0, estable=0;
        for (int i=0; i<7; ++i) {
            int read;
            int max, min;
            cin>>read;
            min=read;
            max=read;
            for (int j=1; j<3; ++j) {
                cin >>read;
                if (max<read) max=read;
                if (min>read) min=read;
            }

            if (max-min<=5) {
                estable++;
                if (min<7) frio++;
                if (max>28) caliente++;
            }

        }
        cout <<estable <<" "<<frio<<" "<<caliente<<endl;
    }
}
```

Autor de la solución: Joan Alemany (Miembro del comité organizador)

Patinete eléctrico

Lindon se ha comprado un patinete eléctrico nuevo para ir desde su casa hasta su trabajo. La ciudad donde vive está formada por *n* cruces y *m* calles, cada calle va de un cruce a otro distinto. Los cruces están numerados del *0* al *n-1*. Lindon vive en el cruce *0* y trabaja en el cruce *1*.
Cada calle tiene un valor entero b_i que son las unidades de batería que el patinete gasta al pasar por esta calle.
Dadas todas las calles con sus valores b_i, ¿cuál es el mínimo número de unidades de batería que Lindon necesita para ir desde su casa a su trabajo?
Se garantiza que siempre existe una manera de ir desde el cruce *0* al cruce *1* pasando por las calles.

Estrategia de la solución

Tenemos un grafo dirigido con pesos en las aristas y tenemos que encontrar el camino de peso mínimo entre dos vértices. Para ello usamos el algoritmo de camino mínimo de Dijkstra.

Solución en Python

```
from queue import PriorityQueue

ncases = int(input())
for case in range(ncases):
    n, m = input().split(' ')
    n, m = int(n), int(m)
    G = [[] for i in range(n)]

    for i in range(m):
        a, b, w = input().split(' ')
        a, b, w = int(a), int(b), int(w)
        G[a].append([b, w])
        G[b].append([a, w])

    Q = PriorityQueue()
    COST = [1e9 for i in range(n)]
    Q.put([0, 0])
    COST[0] = 0

    while (not Q.empty()):
        no = Q.get()
        if (COST[no[1]] == no[0]):
                for x in G[no[1]]:
                        if (COST[x[0]] > no[0] + x[1]):
                                COST[x[0]] = no[0] + x[1]
                                Q.put([no[0] + x[1], x[0]])
    print(COST[1])
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

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

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int, int> pi;
typedef vector<pi> vpi;
typedef vector<vpi> vvpi;

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n, m;
        cin >> n >> m;
        vvpi G(n);
        for (int i = 0; i < m; ++i) {
            int x, y, b;
            cin >> x >> y >> b;
            G[x].push_back(pi(y,b));
            G[y].push_back(pi(x,b));
        }
        vi D(n, -1);
        D[0] = 0;
        priority_queue<pi> Q;
        Q.push(pi(0,0));
        while (!Q.empty()) {
            int x = Q.top().second;
            int d = -Q.top().first;
            Q.pop();
            if (D[x] != d) continue;
            for (auto it : G[x]) {
                int y = it.first;
                int c = it.second;
                if (D[y] == -1 or D[y] > D[x] + c) {
                    D[y] = D[x] + c;
                    Q.push(pi(-D[y],y));
                }
            }
        }
        cout << D[1] << endl;
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)