Soluciones OIFem 2020
Entrenamiento 1

Hola, mundo

Solución

Para conseguir los 100 puntos, basta con una solución que imprima ese mensaje al usuario.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    cout << "Hola, mundo"; // imprimimos el mensaje
    return 0;
}
```

Python

```python
print("Hola, mundo") # imprimimos el mensaje
```

Hola, Usuario

Solución

Le pedimos el nombre al usuario, lo guardamos en una variable de tipo string e imprimimos un mensaje personalizado.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    string nombreUsuario; // declaramos una variable que guarde el nombre
    cin >> nombreUsuario; // le pedimos el nombre al usuario
    cout << "Hola, " << nombreUsuario; // imprimimos un mensaje personalizado
    return 0;
}
```

Python

```python
nombreUsuario = input() # le pedimos el nombre al usuario y lo guardamos
print("Hola, " + nombreUsuario) # imprimimos un mensaje personalizado
```

La suma

Solución

Le pedimos los dos números al usuario, los guardamos en variables de tipo int e imprimimos la suma.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    string nombreUsuario; // declaramos una variable que guarde el nombre
    cin >> nombreUsuario; // le pedimos el nombre al usuario
    cout << "Hola, " << nombreUsuario; // imprimimos un mensaje personalizado
    return 0;
}
```

Python

```python
a, b = map(int, input().split()) # partimos la línea de entrada por el espacio y convertimos a int ambos números
print(a+b) # imprimimos su suma
```

El área

Solución

Le pedimos al usuario la base y la altura e imprimimos el área del triángulo, calculada como $\frac{base*altura}{2}$.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <iomanip> // biblioteca que contiene las instrucciones para que podamos imprimir decimales
using namespace std;


int main() {
    float base, altura; // declaramos dos variables para guardar la base y la altura
    cin >> base >> altura; // le pedimos estos valores al usuario
    cout << fixed << setprecision(1) << base*altura/2; // imprimimos el área del triángulo, redondeada a la décima
    return 0;
}
```

Python

```python
a, b = map(float, input().split()) # procesamos el input como en el problema anterior
print("{:.1f}".format(a*b/2)) # imprimimos el resultado redondeado a la décima
```

Caliente o frío

Solución

Le preguntamos al usuario cuántas veces va a medir la temperatura. Sabiendo esto, vamos pidiendo medida a medida lo que lee el sensor y reaccionamos según las instrucciones.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    int casos; // declaramos una variable para guardar el número de temperaturas
    cin >> casos; // le preguntamos al usuario cuántas temperaturas va a medir
    float temperatura; // declaramos una variable donde ir guardando las temperaturas
    for (int contador = 1; contador <= casos; contador++) {
        // iteramos una vez por cada temperatura
        cin >> temperatura; // pedimos la temperatura
        if (temperatura < 10) {
            // es inferior a 10 grados -> encendemos
            cout << "Enciende\n";
        } else if (temperatura <= 15) {
            // está entre 10 y 15 grados -> la dejamos estar
            // no hace falta volver a revisar si es inferior o no a 10, ya que se habría ejecutado el primer bloque 'if' de ser ese el caso
            cout << "Nada\n";
        } else {
            // es superior a 15 grados -> apagamos
            cout << "Apaga\n";
        }
    }
    return 0;
}
```

Python

```python
t = int(input()) # le preguntamos al usuario cuántas temperaturas va a leer
for i in range(t):
    temp = float(input()) # pedimos la temperatura
    if temp < 10:
        print("Enciende") # si es inferior a 10 grados, encendemos
    elif temp > 15:
        print("Apaga") # si es superior a 15 grados, apagamos
    else:
        print("Nada") # si no, la dejamos estar
```

La tabla del n

Solución

Le pedimos al usuario el valor de n y contamos del 1 al 10, imprimiendo el resultado de cada multiplicación.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    int n; // declaramos una variable para guardar n
    cin >> n; // le pedimos al usuario el valor de n
    for (int i = 1; i <= 10; i++) {
        // para cada número del 1 al 10, imprimimos ese número multiplicado por n
        cout << i << " x " << n << " = " << i*n << "\n";
    }
    return 0;
}
```

Python

```python
n = int(input()) # le pedimos al usuario el valor de n
for i in range(1, 11):
    print(i, "x", n, "=", i*n) # imprimimos el valor de cada multiplicación
```

Comprando chuches

Solución

Vamos chuche a chuche preguntando el valor al usuario, pero solo nos quedamos las que tienen un valor positivo, ya que son las únicas que aumentarán nuestra felicidad. Por lo tanto, guardamos e imprimimos la suma de los valores positivos.

Código

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;


int main() {
    int numChuches, felicidadMaxima = 0, valorChuche;
    cin >> numChuches; // le preguntamos al usuario el número de chuches que hay en la tienda
    for (int contador = 1; contador <= numChuches; contador++) {
        cin >> valorChuche; // le pedimos al usuario el valor de cada chuche
        if (valorChuche > 0) {
            // solo nos interesa la chuche si va a aportarnos felicidad
            felicidadMaxima += valorChuche; // es lo mismo que decir felicidadMaxima = felicidadMaxima + valorChuche;
        }
    }
    cout << felicidadMaxima;
    return 0;
}
```

Python

```python
numChuches = int(input()) # le preguntamos al usuario el número de chuches que hay en la tienda
felicidadMaxima = 0
for i in range(numChuches):
    valorChuche = int(input()) # le pedimos el valor de cada chuche
    if valorChuche > 0:
        felicidadMaxima += valorChuche # lo sumamos al total si es positivo
print(felicidadMaxima)
```

Fibonacci a la inversa

Teoría

- Relaciones de recurrencia

Solución

Usando la fórmula de Fibonacci, vamos calculando cada valor en la secuencia basándonos en los dos anteriores. Una vez hayamos llegado al n'ésimo, repetimos el proceso hacia detrás, imprimiendo por el camino los números que nos encontramos hasta llegar a 0.

Código

C++

```cpp
#include <cstdio>
#include <iostream>
using namespace std;

int main() {
    long long int anterior = 1, preAnterior = 0, actual;
    /* en el turno i:
    preAnterior -> guarda el (i-2)'ésimo número de Fibonacci
    anterior -> guarda el (i-1)'ésimo número de Fibonacci
    actual -> guarda el i'ésimo número de Fibonacci */
    int turnos = 2, n; // turnos llevará la cuenta de en qué turno estamos
    cin >> n;
    if (n == 1) {
        // si solo queremos el primer número, no hace falta mirar más
        cout << "0\n";
    } else {
        while(turnos < n) {
            // hasta llegar al n'ésimo turno, actualizamos los valores de actual,
            ↪  anterior y preAnterior según la fórmula de Fibonacci
            actual = anterior + preAnterior;
            preAnterior = anterior;
            anterior = actual;
            turnos++;
        }
        while(turnos > 0) {
            // repetimos el mismo proceso hacia detrás, imprimiendo los números de
            ↪  la secuencia
            actual = anterior;
            anterior = preAnterior;
            preAnterior = actual - anterior;
            cout << actual << '\n';
            turnos--;
        }
    }
    return 0;
}
```

Python

```python
n = int(input())
if n == 1:
    # si solo queremos el primer número, no hace falta mirar más
    print(0)
else:
    """
    en el turno i:
    preAnterior -> guarda el (i-2)'ésimo número de Fibonacci
    anterior -> guarda el (i-1)'ésimo número de Fibonacci
    actual -> guarda el i'ésimo número de Fibonacci
    """
    turnos = 2 # turnos llevará la cuenta de en qué turno estamos
    anterior = 1
    preAnterior = 0

    while turnos < n:
        # hasta llegar al n'ésimo turno, actualizamos los valores de actual,
        ↪ anterior y preAnterior según la fórmula de Fibonacci
        actual = anterior + preAnterior
        preAnterior = anterior
        anterior = actual
        turnos += 1

    while turnos > 0:
        # repetimos el mismo proceso hacia detrás, imprimiendo los números de la
        ↪ secuencia
        actual = anterior
        anterior = preAnterior
        preAnterior = actual - anterior
        print(actual)
        turnos -= 1
```

La Pachanga

Teoría

- Búsqueda binaria
- Grafos bipartitos

Solución

Es fundamental enfocar este problema como uno de búsqueda y no de ordenamiento. Sabemos que el valor de k estará entre 0 e ∞ (vale con coger un valor superior al máximo odio), por lo que podemos ir probando y descartando intervalos.

Como queremos el valor mínimo de k, si probamos un valor de k y con este se puede hacer una división, podemos descartar todos los valores superiores al de prueba. Si no se puede, podemos descartar todos los valores inferiores al k de prueba. Así, vamos descartando la mitad del intervalo de búsqueda con cada iteración, hasta quedarnos con un solo valor de k. Esta técnica se llama búsqueda binaria.

Para comprobar si es posible partir a los jugadores en dos equipos cuyo odio máximo entre las parejas de un equipo no supere el k de prueba, veremos si es posible construir un grafo bipartito, tomando como aristas las parejas con un odio $> k$ de prueba.

Vamos coloreando el grafo componente conexa a componente conexa. Al encontrarnos una componente conexa que aún no hemos visitado, asignamos equipo a un jugador de la componente conexa (da igual el equipo en este caso). Entonces, visitaremos a los jugadores conectados con este (jugadores cuyo odio a este jugador sea superior a k de prueba) y los asignaremos al equipo contrario. Así lo haremos recursivamente con todos los jugadores de la componente conexa.

Si logramos finalizar este proceso para un valor de k de prueba sin que esto suponga una contradicción, este valor es válido. Si, por el contrario, observamos una contradicción, podremos concluir que este valor de k de prueba es demasiado pequeño y probar números mayores hasta conseguir una configuración válida.

Código

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

vector<vector<pair<int, int>>> conexiones; // conexiones[jugador1] = [(jugador2, odio2), (jugador3, odio3)...]
vector<int> equipo;
/* equipo[jugador] = -1 -> equipo no asignado
equipo[jugador] = 0 -> jugador está en el primer equipo
equipo[jugador] = 1 -> el jugador está en el segundo equipo */

int N; // número de jugadores
bool esPosible; // usaremos esta variable como símbolo para cortar rápido la prueba de un valor de k si ya sabemos que no es posible


void designaEquipo(int jugador1, int kDePrueba) {
    // designaEquipo revisa las conexiones de jugador1 y pone en el equipo contrario a todo jugador al que odie con una intensidad superior a kDePrueba
    if (!esPosible) return; // paramos si ya sabemos que kDePrueba es demasiado pequeño
    int jugador2, odio;
    for (auto pareja: conexiones[jugador1]) {
        jugador2 = pareja.first;
        odio = pareja.second;
        if (odio > kDePrueba) {
            // jugador1 y jugador2 tienen que estar en diferentes equipos
            if (equipo[jugador2] == -1) {
                // jugador2 no tiene equipo asignado
                equipo[jugador2] = 1-equipo[jugador1]; // lo asignamos al equipo contrario
                designaEquipo(jugador2, kDePrueba); // repetimos el proceso para los jugadores conectados con jugador2
            } else if (equipo[jugador2] == equipo[jugador1]) {
                // jugador1 y jugador2 están en el mismo equipo -> kDePrueba es demasiado pequeño
                esPosible = false;
                return;
            }
        }
    }
}
```

```cpp
bool divisionPosible(int kDePrueba) {
    // devuelve si es posible hacer una división con k <= kDePrueba
    esPosible = true;
    equipo.assign(N, -1); // empezamos la prueba con todos los jugadores por asignar
    for (int jugador = 0; jugador < N; jugador++) {
        // vamos jugador a jugador, viendo si ya tiene equipo asignado
        if (equipo[jugador] == -1) {
            // si está sin asignar, ningún jugador de su componente conexa (grupo de jugadores conectados entre sí por odios superiores a kDePrueba) ha sido asignado
            equipo[jugador] = 1; // por lo tanto, lo asignamos al primer equipo (sería igual asignarlo al segundo)
            designaEquipo(jugador, kDePrueba); // miramos a los jugadores que odia más de kDePrueba y los ponemos en el equipo contrario
            if (!esPosible) // no es posible dividir a la componente conexa de jugador
                return false;
        }
    }
    return true; // dividimos con éxito todas las componentes conexas
}

int main() {
    int T, M, U, V, H, k, bajo, medio, alto;
    cin >> T;
    while (T--) {
        cin >> N >> M;
        conexiones.assign(N, vector<pair<int, int>>());
        // creamos el grafo de conexiones como "adjacency list" con pesos
        while (M--) {
            cin >> U >> V >> H;
            conexiones[U-1].push_back({V-1, H});
            conexiones[V-1].push_back({U-1, H});
        }
        // para encontrar el valor de k más bajo, hacemos una búsqueda binaria entre todas las posibilidades
        k = 0;
        bajo = 0;
        alto = 1e9+10;
        while(bajo <= alto) {
            medio = bajo + (alto - bajo)/2;
            if (divisionPosible(medio)) {
                // se puede partir a los jugadores en dos equipos cuyo odio máximo no supera medio
                k = medio; // actualizamos k
                alto = medio-1;
            } else {
                // es imposible partir a los jugadores en dos equipos cuyo odio máximo no supere medio
                bajo = medio+1;
            }
        }
        cout << k << '\n';
    }
    return 0;
}
```

Python

```python
conexiones = [] # conexiones[jugador1] = [(jugador2, odio2), (jugador3, odio3)...]
equipo = []

"""
equipo[jugador] = -1 -> equipo no asignado
equipo[jugador] = 0 -> jugador está en el primer equipo
equipo[jugador] = 1 -> el jugador está en el segundo equipo
"""

N = 0 # número de jugadores
esPosible = True # usaremos esta variable como símbolo para cortar rápido la prueba de un valor de k si ya sabemos que no es posible

def designaEquipo(jugador1, kDePrueba):
    # designaEquipo revisa las conexiones de jugador1 y pone en el equipo contrario a todo jugador al que odie con una intensidad superior a kDePrueba
    global esPosible, equipo
    if not esPosible:
        # paramos si ya sabemos que kDePrueba es demasiado pequeño
        return
    for (jugador2, odio) in conexiones[jugador1]:
        if odio > kDePrueba:
        # jugador1 y jugador2 tienen que estar en diferentes equipos
            if equipo[jugador2] == -1:
                # jugador2 no tiene equipo asignado
                equipo[jugador2] = 1 - equipo[jugador1] # lo asignamos al equipo contrario
                designaEquipo(jugador2, kDePrueba) # repetimos el proceso para los jugadores conectados con jugador2
            elif equipo[jugador2] == equipo[jugador1]:
                # jugador1 y jugador2 están en el mismo equipo -> kDePrueba es demasiado pequeño
                esPosible = False
                return
```

```python
def divisionPosible(kDePrueba):
    # devuelve si es posible hacer una división con k <= kDePrueba
    global esPosible, equipo
    esPosible = True
    equipo = [-1 for i in range(N)] # empezamos la prueba con todos los jugadores por asignar
    for jugador in range(N):
        # vamos jugador a jugador, viendo si ya tiene equipo asignado
        if equipo[jugador] == -1:
            # si está sin asignar, ningún jugador de su componente conexa (grupo de jugadores conectados entre sí por odios superiores a kDePrueba) ha sido asignado
            equipo[jugador] = 1 # por lo tanto, lo asignamos al primer equipo (sería igual asignarlo al segundo)
            designaEquipo(jugador, kDePrueba) # miramos a los jugadores que odia más de kDePrueba y los ponemos en el equipo contrario
            if not esPosible:
                # no es posible dividir a la componente conexa de jugador
                return False
    # dividimos con éxito todas las componentes conexas
    return True

t = int(input())
for i in range(t):
    N, M = map(int, input().split())
    # creamos el grafo de conexiones como "adjacency list" con pesos
    conexiones = [[] for i in range (N)]
    for e in range(M):
        u, v, h = map(int, input().split())
        conexiones[u-1].append((v-1, h))
        conexiones[v-1].append((u-1, h))
    # para encontrar el valor de k más bajo, hacemos una búsqueda binaria entre todas las posibilidades
    k = 0
    bajo = 0
    alto = 1e9 + 10
    while (bajo <= alto):
        medio = (bajo+alto)//2
        if divisionPosible(medio):
            # se puede partir a los jugadores en dos equipos cuyo odio máximo no supera medio
            k = medio # actualizamos k
            alto = medio-1
        else:
            # es imposible partir a los jugadores en dos equipos cuyo odio máximo no supere medio
            bajo = medio+1
    print(int(k))
```