## OMI 2022 - Solución para *Serpientes y escaleras*

### SUBTAREA 1: $M = 0$. No puedes agregar ningún portal.

Como menciona la descripción del problema, dado un arreglo de vectores el juego esta totalmente definido. Para resolver esta subtarea basta con simular y contar el número de saltos.

```
    for(int i = 1; i <= n; ++i){
        std::cin >> l >> r;
        // PARA CADA POSICION CONFIGURA COMO SU PAREJA LA POSICION
        // DONDE ESTA EL OTRO EXTREMO DEL PORTAL
        pareja[l] = r;
        pareja[r] = l;
    }

    // SIMULA EL RECORRIDO
    pos = 1;
    while(pos <= 2 * n){
        pos = pareja[pos];
        ++pos;
        ++res;
    }

    std::cout << res << "\n";
```

La complejidad de la solución es $O(N)$.

### SUBTAREA 2: $r_i = l_i + 1$. Todos los portales están separados.

**Nota:** En la versión original que se usó durante la OMI, esta subtarea aparecía en el tercer lugar mientras que la que aquí se denominará SUBTAREA 3 aparecía en segundo lugar.

Por la estructura inicial de los portales, en el tablero original la ficha pasa por todos ellos. ¿Qué se puede hacer con un nuevo portal? Obviamente, siempre se puede agregar un nuevo portal al final de todos los existentes y asegurar que la ficha va a pasar por ahí, de modo que como mínimo, por cada portal que puedas agregar, puedes lograr que el puntaje resultante aumente el $\mathbf{1}$. ¿Es posible lograr que un portal incremente el puntaje en más de $\mathbf{1}$?

**OBSERVACIÓN 1:** Sin importar la configuración de portales, la ficha siempre llega al extremo derecho.

La primera observación esencial es darse cuenta de que la ficha no puede quedarse dando vueltas en un ciclo. Asume que la ficha pasa por un portal del punto $A$ al punto $B$ y que más adelante en su camino encuentra un portal que va del punto $C$ al punto $D$ donde $D$ es el primer extremo con portal a la izquierda de $A$. Esto ocasionaría que la ficha vuelva a tomar el portal de $A$ a $B$ repita el camino, vuelva a llegar a $C$, de ahí salte a $D$ y se quede dando vueltas en un ciclo.

Observa que para que este ciclo sea posible, la ficha tuvo que llegar inicialmente a $A$, para llegar a $A$ tiene que llegar por el lado izquierdo, pero dado que $D$ es el primer extremo a la izquierda de $A$ quiere decir que la ficha empezó entre $D$ y $A$ (lo cual es inválido) o que la ficha llegó a $D$ saltando desde $C$ (de otra forma al llegar a $D$ por la izquierda saltaría de ahí a $C$ y tomaría otro camino). Pero si llegó saltando desde $C$ tuvo que haber pasado primero por $B$ (ya que $B$ forma parte del camino para llegar a $C$, recuerda que el camino de la ficha es fijo dada una configuración de portales) y a su vez, para llegar a $B$ tiene que venir de $A$. Pero estás tratando de llegar a $A$, de modo que no puedes venir de $A$. Por lo tanto, concluyes que es imposible lograr dicha configuración.

**OBSERVACIÓN 2:** No se puede pasar por un portal en un sentido dado dos veces. Esto es una consecuencia de la **observación 1**, pasar dos veces por un portal en el mismo sentido implicaría la creación de un ciclo (ya que el camino está definido), de modo que cada portal se puede utilizar como máximo 2 veces.

Con las dos observaciones anteriores puedes saber ¿cuál es el máximo de puntaje que puede incrementar un nuevo portal? * El nuevo portal se puede utilizar como máximo 2 veces. * El nuevo portal puede hacer que un portal que se usa sólo una vez se use por segunda vez. * El nuevo portal puede hacer que un portal que no se usa, se use hasta un máximo de dos veces.

Experimentando un poco puedes ver que la configuración de dos portales: $A, B$ y $C, D$ tales que los extremos están ordenados $[A, C, B, D]$ utiliza ambos portales dos veces (su máximo). De modo que para resolver esta subtarea debes hacer lo siguiente con cada nuevo portal...

- Si hay un portal que no se está usando dos veces, usa el nuevo portal en la configuración anterior. Esto incrementará el puntaje en $\mathbf{3}$, el segundo salto del portal que ya estaba y los dos saltos del nuevo portal.
- Si ya no hay portales iniciales sub utilizados entonces agrega el nuevo portal como un portal de un uso al final de la línea.

El siguiente código resuelve esta subtarea:

```
// PASA POR TODOS LOS PORTALES ORIGINALES
res = n;

// CON CADA PORTAL NUEVO SE PUEDE AUMENTAR 3
// SI HAY UN PORTAL ORIGINAL SIN USAR
res += 3 * std::min(n, m);

// CON CADA PAREJA DE PORTALES EXTRA SE PUEDE SUBIR 4
// SI QUEDA PORTAL EXTRA SIN PAREJA SE AGREGA 1
d = m - n;
if (d > 0){
    res += 4 * (d / 2);
    res += d % 2;
}
```

La complejidad es $O(1)$.

**SUBTAREAS 3 a 6.**

Se agrupan todas las subtareas ya que dependen de una misma observación y la diferencia es únicamente la estructura de datos que se utilice para su implementación.

**OBSERVACIÓN 3:** En un tablero dado, los portales que no se utilizan (ninguna vez) siempre forman ciclos, es decir, si colocas la ficha en un extremo de portal que no se use y simulas su recorrido la ficha va a volver, después de una serie de saltos, al mismo punto, repitiendo ese ciclo eternamente.

La demostración de esta observación es similar a la **observación 1**. Si existe un portal $A, B$ que no se usa es porque hay un camino de portales que pasa por un punto $C$ accesible desde el inicio y que está a la izquierda de $A$ que eventualmente llega a un punto $D$ a la derecha de $B$ cuyo recorrido ya nunca regresa, causando que el recorrido se *salte* el portal $A, B$. Si se empieza el recorrido desde $A$ eventualmente se llegará al punto $D$ que te hará retroceder a $C$ y volver a llegar de nuevo al punto $A$.

**OBSERVACIÓN 4:** Si se representa cada sentido de un portal como un nodo en un grafo, el grafo resultado es una línea (el camino que va del origen al final) y un conjunto de ciclos disjuntos. Esta observación es una consecuencia de las tres observaciones previas.

**OBSERVACIÓN 5:** Con un nuevo portal es posible agregar todos los nodos de un ciclo al camino original.

Observa que si pones un portal entre dos nodos del camino original (la línea del grafo) y dos nodos de un ciclo el nuevo camino será como sigue:

- La ficha sigue camino original hasta encontrar el extremo $A$ del nuevo portal.

- La ficha *entra* al ciclo a través del extremo $B$ del nuevo portal.
- La ficha recorre el ciclo hasta volver a llegar a $B$.
- La ficha regresa al camino original en $A$ y continúa con el recorrido previo.

De modo que el nuevo portal agregó al camino original todos los nodos del ciclo y además los dos nodos del nuevo portal, ya que lo recorrió en ambos sentidos. El puntaje por lo tanto se incrementa en el largo del camino más dos del nuevo portal.

En base a las observaciones previas, el algoritmo para resolver el problema es:

- Construir una variante del modelo previo (línea más ciclos).
- Ordenar los ciclos por tamaño.
- Mientras tengas nuevos portales ve uniendo los ciclos al camino original del mayor al menor.
- Si tras haber unido todos los ciclos quedan portales nuevos, únelos con el algoritmo de la subtarea 2.

```cpp
int main() {

    std::cin >> n >> m;
    std::iota(pareja, pareja + MAXP + 1, 1);
    for(int i = 1; i <= n; ++i){
        std::cin >> l >> r;

        // PARA CADA POSICION CONFIGURA COMO SU PAREJA LA POSICION
        // DONDE ESTA EL OTRO EXTREMO DEL PORTAL
        pareja[l - 1] = r;
        pareja[r - 1] = l;
    }

    // VALIDA LOS CICLOS CON UNA ESTRUCTURA TIPO UNION FIND.
    // INICIALIZA LA ESTRUCTURA
    std::fill(tam, tam + MAXP + 1, 0);
    std::iota(ciclo, ciclo + MAXP + 1, 0);

    for(int i = 0; i <= MAXP; ++i){
        if (ciclo[i] == i){
            c = i;
            pos = pareja[i];
            ciclo[pos] = c;

            if (pos != c + 1) ++tam[c];
            while (pos < MAXP && pos != c){
                if (pareja[pos] == pos + 1){
                    pos = pos + 1;
                    ciclo[pos] = c;
                }
                else{
                    pos = pareja[pos];
                    ciclo[pos] = c;
                    ++tam[c];
                }
            }

            // EL C ES UN CAMINO, NO UN CICLO, NO HAY QUE AGREGARLO
            if (c) ciclosPorTamano.push_back(c);
        }
    }

    // ORDENA LOS CICLOS POR TAMANO
    std::sort(ciclosPorTamano.begin(), ciclosPorTamano.end(), [&](int a, int b){return t

    res = tam[0];
    d = m - ciclosPorTamano.size();
    if (d > 0){
        res += (d >> 1) << 2;
        res += d & 1;
    }
    for(int i = 0; i < std::min(m, (int)ciclosPorTamano.size()); ++i) res += tam[ciclosP

    std::cout << res << "\n";

    return 0;
}
```

La complejidad de la solución es $O(NlogN)$