OMI 2022 - Solución para *La vida de las personas*

SUBTAREA: $N, Q = 100$, $a_i = b_i \leq 100$. Cada persona muere al final del año en el que nace.

Dado que todas las personas mueren el año en el que nacen, todas mueren antes de cumplir un año. En todas las preguntas la respuesta a la edad de la persona más joven y la persona más anciana, será 0 años. Por lo tanto, basta con encontrar una forma de contestar la cantidad de personas vivas en un año específico.

El rango posible de años en que nace o muere gente es de $[1, 100]$. Puedes tener un arreglo *vivos* tal que *vivos[i]* contenga el número de personas que nacieron en el año i, de esa forma puedes contestar de forma eficiente la cantidad de personas vivas en un año específico.

```
for (int i = 1; i <= n; i++) {
  cin >> nace >> muere;
  vivos[nace]++;
}
```

SUBTAREA: $N, Q, a_i, b_i \leq 1000$.

Esta subtarea tiene límites que permiten revisar a todas las personas en cada pregunta para ver si estaban vivas en ese año.

Por cada persona debes validar: * Si estaba viva en el año de la pregunta. En caso de estar, aumentar la cuenta de *vivos*. * Si está viva en el año en cuestión, validar si es la persona más joven o más anciana.

El siguiente código resuelve esta subtarea:

```
for (int i = 1; i <= q; i++) {
  cin >> anio;
  int vivos = 0, joven = 1005, anciano = 0;

  // VERIFICA DA PERSONA
  for (int j = 1; j <= n; j++) {
    // REVISA SI ESTA VIVA
    if (persona[j].nacio <= anio && persona[j].murio >= anio) {
      vivos++;
      joven = min(joven, anio - persona[j].nacio);
      anciano = max(anciano, anio - persona[j].nacio);
    }
  }

  if (vivos) cout << vivos << " " << joven << " " << anciano << "\n";
  else cout << "0 0 0\n";
}
```

Para cada pregunta se tiene que validar a todas las personas, la complejidad de la solución es $O(NQ)$.

SUBTAREA: $N \leq 1000$, $a_i, b_i \leq 50000$.

El número de preguntas ya no está limitado, de modo que tenemos que poder contestar cada pregunta de forma eficiente. La cantidad de personas, así como el rango de años aún está limitado, por lo tanto, puedes precalcular

el resultado en cada uno de los años, memorizarlo y usarlo para contesar las preguntas.

Manten tres arreglos ***vivos***, ***joven*** y ***anciano*** tales que en su índice i guarden cada uno el número de personas vivas, la edad de la persona viva más joven y la de la más anciana. Dado que el rango de años es pequeño, para cada persona podemos actualizar uno por uno todos los años en los que estuvo viva.

El siguiente código muestra una forma de mantener los arreglos mencionados.

```
for (int i = 1; i <= 50000; ++i) joven[i] = 50001;
for (int i = 1; i <= n; ++i) {
  cin >> nacio >> murio;
  for (int a = 0; a <= murio - nacio; ++a){ // RECORRE SUS AÑOS DE VIDA
      anio = nacio + a;
      vivos[anio]++;
      joven[anio] = min(joven[anio], a);
      anciano[anio] = max(anciano[anio], a);
  }
}
```

El precálculo se hace recorriendo los años de cada persona. Para contesar la pregunta basta con escribir los valores en los arreglos. La complejidad de la solución es $O(N * max(b_i) + Q)$

SUBTAREA: $a_i, b_i \leq 10^6.

La cantidad de personas y el tamaño del rango de años no permite hacer el precálculo anterior, sin embargo, el rango de años todavía es tal que si encuentras una forma eficiente de hacer el precálculo puedes guardar en memoria el resultado para cada año.

Es posible precalcular cada uno de los tres valores que te piden, pero para cada uno debes usar técnicas distintas que se describen a continuación:

Conteo de personas vivas: Para contar la cantidad de personas vivas puedes marcar el inicio y el fin de su rango de vida en un arreglo, es decir, en cada año que nace una persona agregas un $+1$ y en el primer año que ya no está viva (el año siguiente al que muere) actualizas con un -1. Una vez marcado el inicio y fin de la vida de cada persona haz un *acumulado de prefijos* (ve la solución de *5-5 feliz encuentro* de esta misma OMI) para saber el número de personas vivas en cada año. El siguiente código muestra una manera de implementar esta técnica:

```
    for (int i = 1; i <= n; ++i){
        cin >> nacio >> murio;
        ++vivos[nacio]; // AGREGA UNO AL INICIO DE SU VIDA
        --vivos[murio + 1]; // QUITA UNO CUANDO YA NO ESTA VIVO
    }
    for (int i = 1; i <= MAX_FECHA; ++i) vivos[i] += vivos[i - 1];
```

Persona más joven viva: Para contestar la edad de la persona viva más joven requieres saber quién de los que están vivos fue el **último** que nació. Existe una estructura de datos llamada `stack` (pila en español) que es especialmente útil para contestar cuál es el último dato que se ingreso. Puedes usar un `stack` para recorrer en orden los años e ir ingresando cada que una persona nace, de esa forma el `stack` tendrá siempre en la parte de *arriba* el dato de la persona más joven. Cada año, además, debe validarse si dicha persona ya murió en cuyo caso debe eliminarse del `stack`. El siguiente código muestra una forma de hacerlo:

```cpp
// STACK PARA LLEVAR LA PERSONA MAS JOVEN VIVA
stack<pair<int, int> > joven;
for (int i = 1; i <= n; ++i){
    cin >> nacio >> murio;
    // EL ARREGLO tmp VA A GUARDAR, EN SU POSICION i, LA FECHA DE MUERTE MAS
    // LEJANA ENTRE TODAS LAS PERSONAS QUE NACIERON ESE AÑO
    tmp[nacio] = max(tmp[nacio], murio);
}
for (int i = 1; i <= MAX_FECHA; ++i){
    // SI ALGUNA PERSONA NACIO EL AÑO i, AGREGALA AL STACK, SERA LA MAS JOVEN
    if(tmp[i]) joven.push({i, tmp[i]});

    // SACA DEL STACK CUALQUIER PERSONA QUE ESTE HASTA ARRIBA Y YA HAYA MUERTO
    while(!joven.empty() && joven.top().second < i) joven.pop();

    // ALMACENA LA EDAD DE LA PERSONA MAS JOVEN EN EL AÑO i
    if(!joven.empty()) edad_joven[i] = i - joven.top().first;
    else edad_joven[i] = 0;
}
```

Persona más anciana viva: Para contestar la edad de la persona viva más anciana requieres saber quién de los que están vivos fue el **primero** que nació. Al igual que para el caso del joven, existe una estructura de datos llamada `queue`(cola en español) que es especialmente adecuada para conocer el primer dato que ingreso. Puedes usar un `queue` para recorrer en orden los años e ir ingresando a cada persona que nace, de esa forma, el `queue` tendrá siempre al inicio el dato de la persona más anciana. El siguiente código muestra cómo hacerlo:

📋

```cpp
// QUEUE PARA LLEVAR LA PERSONA MAS ANCIANA VIVA
queue<pair<int, int> > anciano;
for (int i = 1; i <= n; ++i){
    cin >> nacio >> murio;
    // EL ARREGLO tmp VA A GUARDAR, EN SU POSICION i, LA FECHA DE MUERTE MAS
    // LEJANA ENTRE TODAS LAS PERSONAS QUE NACIERON ESE AÑO
    tmp[nacio] = max(tmp[nacio], murio);
}
for (int i = 1; i <= MAX_FECHA; ++i){
    // SI ALGUNA PERSONA NACIO EL AÑO i, AGREGALA AL QUEUE
    if(tmp[i]) anciano.push({i, tmp[i]});

    // SACA DEL STACK CUALQUIER PERSONA QUE ESTE HASTA ARRIBA Y YA HAYA MUERTO
    while(!anciano.empty() && anciano.front().second < i) anciano.pop();

    // ALMACENA LA EDAD DE LA PERSONA MAS ANCIANA EN EL AÑO i
    if(!anciano.empty()) edad_anciano[i] = i - anciano.front().first;
    else edad_anciano[i] = 0;
}
```

Juntando los tres procesos se puede resolver el problema en complejidad $O(N + Q + max(b_i))$

SOLUCIÓN OFICIAL: ninguna restricción.

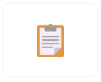
Para la solución completa no podemos utilizar las técnicas de la subtarea anterior ya que no alcanzan en memoria ni en tiempo. Sin embargo, observa que en la subtarea anterior, aunque se recorre todo el rango de fechas posibles, en realidad solo se realizan acciones cuando alguien nace o alguien muere, de modo que si de alguna forma se puede procesar únicamente esos momentos, podemos seguir utilizando las técnicas de la subtarea anterior.

Para esta subtarea puedes utilizar la técnica que se conoce como *line sweep* que consiste básicamente en ir *barriendo* una línea a través de un espacio de *puntos* ordenados deteniéndose únicamente en aquellos puntos que son de interés para hacer las actualizaciones correspondientes. En el caso de este problema el *espacio* en el que mueves la línea es el rango de fechas y los *puntos* que nos interesan son los años en que alguien nace, alguien muere o tenemos que contesar una pregunta.

Además del *line sweep* se utiliza otra técnica que se conoce como *solución offline,* esta técnica consiste en contestar las preguntas que te hacen no en el orden en el que te las hacen sino en el orden en el que te resulta más conveniente. En este caso, debes contestar las preguntas ordenadas por el año, para poder ir contestando mientras mueves la *línea* del *line sweep*.

Los pasos del algoritmo son los siguientes: * Agrega todos los *puntos de interés* (nacimientos, muertes y preguntas) a un arreglo y ordénalo por fecha. * Procesa uno por uno los elementos del arreglo (barrido de la línea) y en cada *punto*, dependiendo de su tipo, realiza las acciones correspondientes. * Cuando alguien nazca: Actualiza los vivos, el `stack` de persona más joven y el `queue` de persona más anciana. * Cuando alguien muera: Actualiza los vivos. * Cuando te hagan una pregunta: Extrae los datos inválidos (personas que ya murieron) del `stack` y del `queue` y memoriza la respuesta para esa pregunta. * Una vez procesados todos los puntos, contesta las preguntas en el orden en el que las pide el problema.

En la solución oficial, en vez de usar un `stack` y un `queue` se utiliza una estructura llamada `deque` que es justo la unión de un `stack` con un `queue` (justo lo que necesitas :)).



```cpp
#define nacimiento 0
#define pregunta 1
#define muerte 2

#define fecha first.first
#define tipo first.second
#define id second

std::vector<pair<pair<int, int>, int> > eventos;
int n, q, a, b;
int resultados[3][MAX + 2], vivo[MAX + 2];

int main() {
  // LEE LA ENTRADA Y GENERA LOS EVENTOS.
  // LOS EVENTOS DEBEN IR ORDENADOS POR AÑO Y LUEGO TIPO,
  // EN EL TIPO LA PREFERENCIA DEBE SER: NACIMIENTO, PREGUNTA, MUERTE
  std::cin >> n >> q;
  for (int i = 1; i <= n; ++i) {
    std::cin >> a >> b;
    eventos.push_back({{a, nacimiento}, i});
    eventos.push_back({{b, muerte}, i});
  }
  for (int i = 1; i <= q; ++i) {
    std::cin >> a;
    eventos.push_back({{a, pregunta}, i});
  }

  // ORDENA LOS EVENTOS
  std::sort(eventos.begin(), eventos.end());

  // PROCESA LOS EVENTOS
  int personasVivas = 0;
  std::deque<pair<int, int> > vivos;
  for (auto ev : eventos) {
    if (ev.tipo == nacimiento) {
      // SI ES UN NACIMIENTO, REGISTRA QUE HAY UNA PERSONA VIVA MAS
      // Y GUARDA A DICHA PERSONA EN LA LISTA DE PERSONAS VIVAS PARA CONTESTAR
      // PREGUNTAS
      ++personasVivas;
      vivo[ev.id] = 1;
      vivos.push_back({ev.fecha, ev.id});
    } else if (ev.tipo == pregunta) {
      // PARA LAS PREGUNTAS DEBEMOS DE CONTESAR LOS SIGUIENTE DATOS:
      // PERSONAS VIVAS, ESE VALOR ESTAN EN LA VARIABLE personas vivas
      resultados[0][ev.id] = personasVivas;

      // PARA LA PERSONA MAS JOVEN Y MAS ANCIANA SE USA EL DEQUE DE vivos
      // ES IMPORTANTE IR ELMINANDO A LAS PERSONAS QUE YA NO ESTAN VIVAS DE LOS
      // EXTREMOS DEL DEQUE QUITA LOS NO-VIVOS DEL INICIO
      while (!vivos.empty() && !vivo[vivos.front().id]) vivos.pop_front();

      // QUITA LOS NO-VIVOS DEL FINAL
      while (!vivos.empty() && !vivo[vivos.back().id]) vivos.pop_back();

      if (vivos.empty())
        resultados[1][ev.id] = resultados[2][ev.id] =
            0;  // SI NO HAY NADIE VIVO DEVUELVE CEROS
      else {
        resultados[1][ev.id] = ev.fecha - vivos.back().first;
        resultados[2][ev.id] = ev.fecha - vivos.front().first;
```

```
      }

    } else {
      // CUANDO UNA PERSONA MUERE :( HAY QUE DISMINUIR LA CUENTA DE VIVOS
      // Y MARCARLA COMO NO VIVA PARA CUANDO SEA NECESARIO SACARLA DEL DEQUE
      --personasVivas;
      vivo[ev.id] = 0;
    }
  }

  for (int i = 1; i <= q; ++i) {
    for (int j = 0; j < 3; ++j) {
      std::cout << resultados[j][i] << " ";
    }
    std::cout << "\n";
  }

  return 0;
}
```