Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2021 - 1

Tarea 0

Fecha de entrega código e informe: Jueves 8 de abril

Objetivos

- Familiarizar al estudiante con el lenguaje de programación `C` y compararlo con el lenguaje `Python`.
- Entender e implementar estructuras de datos que combinan arreglos, listas ligadas y árboles.
- Realizar operaciones sobre nodos de árboles.

Introducción

Kevin es un contador que trabaja en `Dunder Mifflin`, una empresa distribuidora de papel. Amante de las galletas y experto en la cocina, Kevin decide preparar un ***chilli* con carne** para sus compañeros de trabajo.

Llegado el día en que Kevin lleva su aclamada receta a La Oficina, ocurre un **accidente** de gran envergadura: Se le cae al suelo todo el *chilli* con carne. Desesperado, Kevin empieza a limpiar la alfombra con papeles y otros elementos de oficina. Este video muestra cómo sucedieron los hechos.

Momentos antes del desastre

Lo que Kevin nunca pudo prever fue que su intento de limpieza gatilló un **desastre bioquímico**: La contaminación del *chilli* con carne en la alfombra y la incorporación de papel de `Dunder Mifflin`, creó un ambiente propicio para el origen de un **nuevo germen altamente contagioso**.

Desgraciadamente, los camiones que distribuyen el papel de `Dunder Mifflin` portaron el germen y en cuestión de horas, **distintas regiones del mundo registraron pacientes ceros contagiados**. La `OMS` diseñó un `Test` con la ayuda del jefe de aseguramiento de calidad de `Dunder Mifflin` para confirmar si una persona está contagiada, pero necesitan de tu ayuda para llevar un registro mundial de los contagios.

Problema

Tu misión será modelar a todas las `PERSONAS` que estén contagiadas o sean casos sospechosos de todas las regiones de cada país a través de **un bosque de árboles**, en donde un árbol corresponde a los casos de una sola región de un país.[1]

Modelado del Bosque

El bosque se estructura como un arreglo de arreglos. Cada elemento del primer arreglo hace referencia a un país y por cada país hay otro arreglo que se asocia con sus regiones. Por ejemplo, la coordenada `(0, 3)` hace referencia a la región 3 del país 0. La cantidad de regiones por país puede variar, por lo que el arreglo de arreglos **no es una matriz regular**.

La celda `(i, j)` del arreglo de arreglos guarda al puntero correspondiente al **paciente cero** de la región `j` del país `i`. Dicho caso cero es la raíz del árbol de contagiados y de sospechosos de esa localidad.

La lógica de esta estructura de datos ya está implementada en `Python` bajo el nombre `World` y deberás traspasarlo al lenguaje `C`.

Aspectos relevantes de un Árbol de contagios

A continuación, se detallan aspectos relevantes del árbol de contagios de una región de un país:

- Un árbol se construye a partir de personas, en donde cada persona se representa como una estructura de datos llamada `Persona`.
- Una `Persona` puede tener 5 estados, cada uno con un número que lo identifica a lo largo del problema:
  - `Sospechoso(0)`: Es contacto estrecho de algún contagiado o de otro sospechoso.
  - `Sospechoso en espera (1)`: Es un sospechoso que realiza un `Test` y espera el resultado.
  - `Contagiado (2)`: Tiene un `Test` positivo.
  - `Recuperado (3)`: Es un contagiado que se recuperó.
  - `No contagiado (4)`: Tiene un `Test` negativo
- Además de un estado, cada persona posee **un identificador único (`ID`) numérico**. Los `IDs` parten por convención en 0, el cual le pertenece al paciente cero de la región.
- Cada árbol inicia con una `Persona contagiada` (2), llamada paciente cero.
- Cada `Persona` se puede contagiar una única vez.
- A excepción del paciente cero, toda `Persona` es contagiada por otro contagiado.
- Cada `Persona` puede tener un número indefinido de contactos estrechos. Para facilitar el problema, un contacto estrecho solo puede aparecer una vez a lo largo del problema.
- Cuando una `Persona` es declarada como `contagiada`, todos sus contactos estrechos pasan al estado `sospechoso en espera`, por lo que se deben realizar un `Test`:
  - Si el `Test` de un contacto estrecho es positivo, se le declara como `contagiado`, y consecuentemente sus contactos estrechos pasan al estado `sospechoso en espera` para someterse al `Test`.
  - Si el `Test` es negativo, el `sospechoso` se declara como `no contagiado` y se debe eliminar del árbol.[2]

[1] Por simplicidad, se asume que una persona solamente puede ser contagiada por alguien de su misma región.
[2] Un árbol solo contendrá `Personas` con estados `0, 1, 2 y 3`.

Modelado de un Árbol de contagios

A continuación se visualiza la lógica de la estructura de datos `Persona` que es la unidad constituyente del árbol:

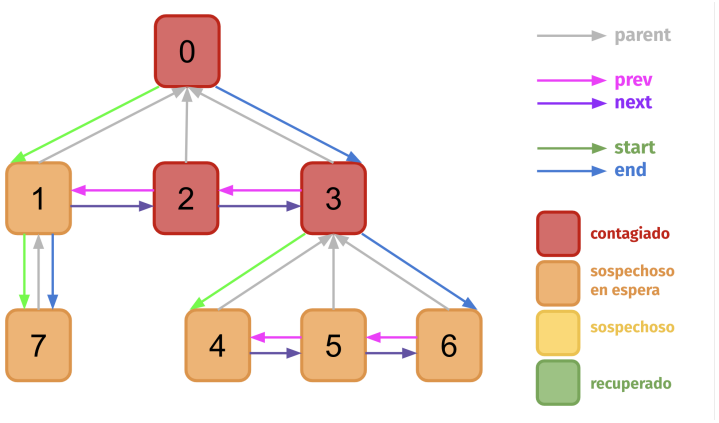

Figura 1: Conexiones entre `Personas` en un árbol

La estructura de datos `Persona` es bastante peculiar, pues **es simultáneamente el nodo de un árbol y un elemento de una lista ligada**:

- Por un lado y al igual que un nodo, tiene un atributo `parent`, que indica a la `Persona` que reportó al nodo como contacto estrecho. Además, los atributos `start` y `end` conectan al nodo con sus hijos, específicamente, a las `Personas` ubicadas al inicio y al fin de la lista ligada de los contactos estrechos.
- Por otro lado, los atributos `prev` y `next` conectan a los contactos estrechos de una `Persona` como una lista ligada, en donde cada elemento de la lista ligada es un contacto estrecho del padre. El puntero `end` del padre ayuda también a agregar nuevos contactos estrechos al final de la lista ligada.

Cabe aclarar que **estas conexiones son punteros** de forma tal que la `Persona` del comienzo de la flecha guarda como atributo al puntero de la `Persona` del final de la flecha. Por ejemplo, en la Figura 1, la `Persona` con `ID 0` guarda en `start` el puntero de la `Persona 1` y en `end` el puntero de la `Persona 3`. En este caso, los otros 3 atributos `parent`, `prev` y `next` son todos `NULL`.

Eventos

Un evento representa una acción que ocurre sobre algún árbol de contagios del bosque. Para explicar los eventos se utilizarán los siguientes conceptos:

- `country`(int): Número entero. Representa el índice del país donde ocurrirá el evento.
- `region`(int): Número entero. Representa el índice de la región donde ocurrirá el evento.
- `person_route`: Indica el camino a recorrer para llegar a una cierta `Persona`. Se representa como `N` x_1 x_2 ... x_N. Donde `N` es la profundidad donde se ubica la `Persona` (desde la raíz con `ID 0`), y x_1 x_2... x_{N-1} son los ids por los cuales bajar para llegar a dicha `Persona`. El último elemento x_N corresponde al `ID` de la `Persona` a la cual se quiere llegar.
- `close_contacts`(int): Indica la cantidad de estrechos que se le agregan a una `Person`.

A continuación se observa un ejemplo de `person_route` (2 3 5), que comienza desde la raíz y que llega hasta la `Persona 5`. El 2 en azul representa a la profundidad `N` donde se ubica la `Persona` objetivo, por lo que esta tendrá 2 `IDs` subsecuentes para llegar al nodo deseado. Siguiendo las flechas grises y escogiendo el `ID 3`, llegaremos a la `Persona` objetivo de `ID 5`. Si `person_route es 0`, nos referimos a la raíz. En las próximas secciones detallaremos los eventos que se realizarán sobre cada árbol.

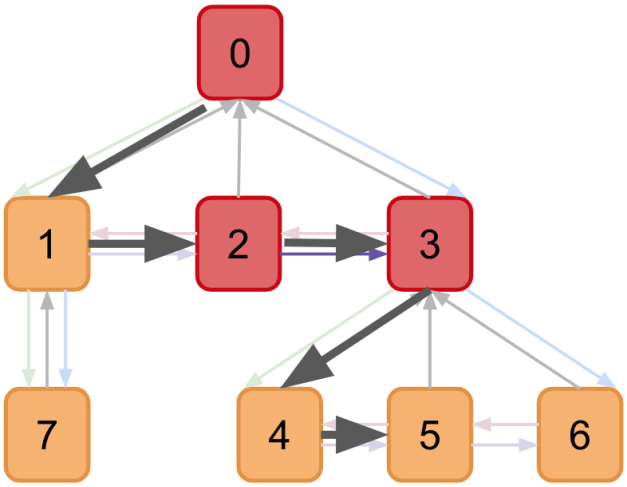

Figura 2: Ruta 2 3 5

0.1 `ADD_CONTACTS country(int) region(int) contact_route close_contacts(int)`

Este evento inserta en el árbol a las `Personas` nuevas que fueron contactos estrechos con la `Persona` encontrada en la ruta, que pasa a ser el padre de estas nuevas entradas. Los `IDs` de estas nuevas `Personas` se asignarán de acuerdo a un contador que inicia en `1` (pues el nodo raíz tiene `ID 0`) y aumenta en `1` por cada nuevo contacto estrecho agregado. El padre puede o no tener registrado casos estrechos anteriores. En caso de tenerlos, los nuevos contactos estrechos se deben agregar al final de la lista ligada.
A continuación, se aplica este evento a la `Persona 2` del país 1 de la región 3:

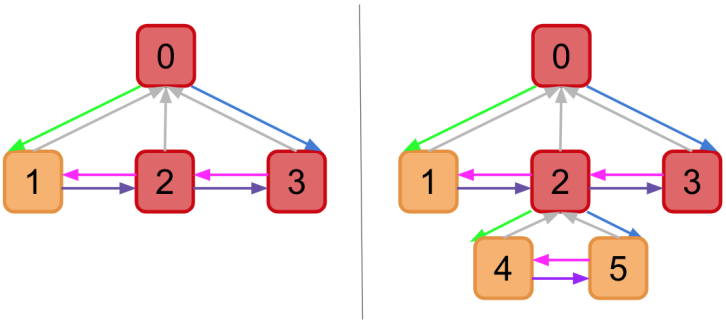

Figura 3: Ejemplo evento `ADD_CONTACTS 1 3 1 2 2`

- Si el estado del padre es `contagiado`, los contactos estrechos se inicializan con estado `sospechoso en espera`.
- Si el estado del padre es distinto a `contagiado`, los contactos estrechos se inicializan con estado `sospechoso`.

0.2 `RECOVERED country(int) region(int) person_route`

Este evento registra que una persona que estaba `contagiada` se recuperó de la enfermedad. Por consecuencia su estado cambia a `recuperado`. Ninguna acción se realiza sobre sus hijos.

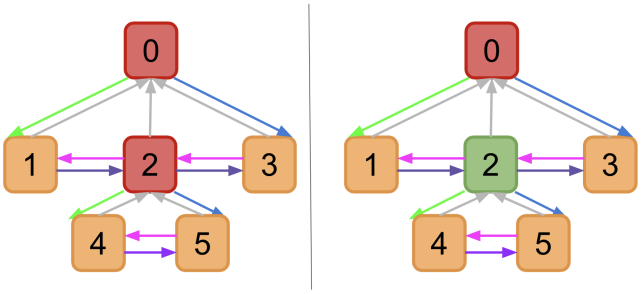

Figura 4: Ejemplo evento `RECOVERED 1 3 1 2`

0.3 `POSITIVE country(int) region(int) person_route`

Este evento registra que una cierta `Persona` que era `sospechosa en espera` recibe un `Test` positivo, por lo que su estado cambia a `contagiado`. En caso de tener contactos estrechos, cambia el estado de cada uno a `sospechoso en espera`. El cambio de estado a sospechoso en espera solo se realiza a los hijos directos del caso positivo.
A continuación, se aplica este evento a la `Persona 2` del país 1 de la región 3:

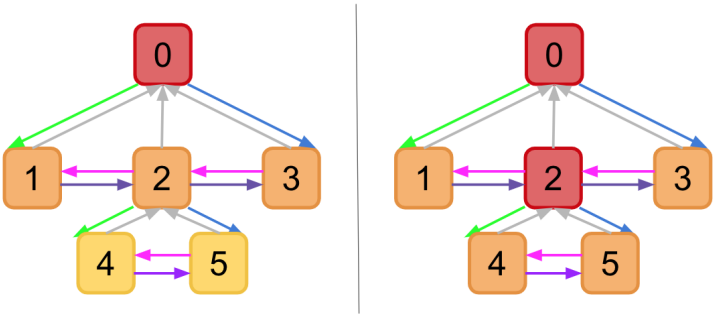

Figura 5: Ejemplo evento `POSITIVE 1 3 1 2`

0.4 `NEGATIVE country(int) region(int) person_route`

Este evento registra que una `Persona` que se realizó un `Test` no está contagiada. Por consecuencia, se debe llegar hasta la `Persona` y eliminarla del árbol. También se deberán eliminar a sus hijos, los hijos de sus hijos, y así sucesivamente.

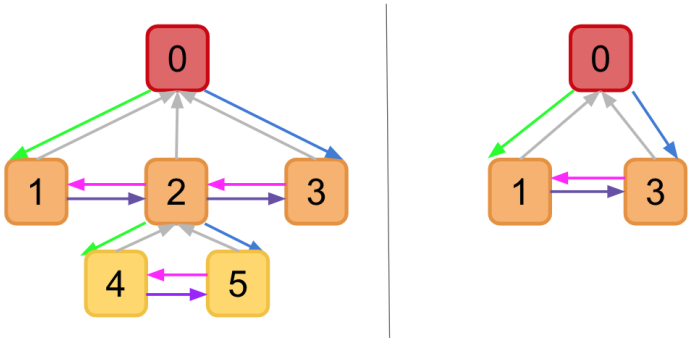

Figura 6: Ejemplo evento `NEGATIVE 1 3 1 2`

0.5 `CORRECT country(int) region(int) person_route_1 person_route_2`

Este evento invierte los contactos estrechos (y descendientes) de 2 `Personas` debido a un error del sistema. En caso de que el nuevo padre esté contagiado, sus nuevos hijos que estén es estado `sospechoso` pasarán a `sospechoso en espera`. En cualquier otro caso, los estados de los hijos se mantendrán a cómo estaban antes de aplicar este evento. Se asume que **NO** hay intercambios entre `Personas` donde **una desciende de otra**.

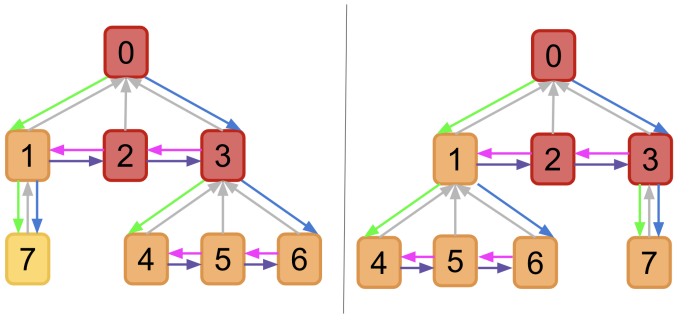

Figura 7: Ejemplo evento `CORRECT 1 3 1 1 1 3`

0.6 INFORM country(int) region(int)

A diferencia de los otros eventos, `INFORM` retorna una representación del árbol que se guarda en el archivo `output`. Tomemos de ejemplo este árbol:

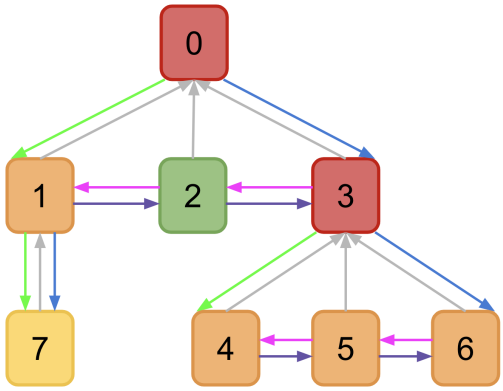

Figura 8: Ejemplo evento `INFORM 1 3`

Como vemos a continuación, el evento `INFORM` muestra el `ID` y el estado de cada `Persona` del árbol:

```
INFORM 1 3
0:2
    1:1         // Primer hijo del caso cero
        7:0     // El primer hijo del nodo 1, con ID 7 y estado 0
    2:3
    3:2
        4:1
        5:1
        6:1
```

Al imprimir los contactos estrechos, se debe seguir el orden de la lista ligada, imprimiendo siempre al `ID` con su estado y después a sus nodos hijos. Se indenta `4` espacios.

0.7 STATISTICS country(int) region(int)

Por último, el evento `STATISTICS` retorna la cantidad de `Personas` según su estado en el árbol. Utilizando de nuevo la Figura 8, este evento debe escribir `estado:cantidad` en el archivo `output`:

```
STATISTICS 1 3
0:1
1:4
2:2
3:1
```

Siempre se debe seguir el mismo orden `sospechoso - sospechoso en espera - contagiado - recuperado` y se deben reportar a pesar de que no existan `Personas` con algún estado. Reportar los `no contagiados` no es de interés, puesto que se eliminan del árbol.

Archivos iniciales

Para ayudarte a empezar a programar, tus ayudantes han preparado una base en *Python* y tu misión es pasarlo al lenguaje *C*. Tu primera tarea es convertir las estructuras `World` y `Person` y los eventos `ADD_CONTACTS` e `INFORM`. El resto de los eventos tendrás que programarlos por tu propia cuenta. Además te entregamos un código en `C` que lee el archivo de **input**. Todo esto lo encontrarás en el repositorio inicial de la tarea.

Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un ejecutable de nombre `kevin-21` que se ejecuta con el siguiente comando:

```
./kevin-21 <input> <output>
```

Donde `input` será un archivo con los eventos a simular y `output` el archivo a guardar los resultados.

Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada uno de estas ejecuciones que tenga un `output` igual al esperado. A continuación detallaremos los archivos de input y output.

Input

La información del archivo de `input` viene entregada en el siguiente formato:

- La primera línea inicia un número `N` que indica la cantidad de países que habrán. Luego le siguen `N` números que indican la cantidad de regiones de cada país.
- La siguiente línea tiene otro número único `M` que indica la cantidad de eventos que se realizarán sobre el bosque.
- Por último, cada una de las siguientes `M` líneas contiene un evento que se realiza sobre un árbol. Lo que contiene cada evento depende de los atributos que se explicaron en cada caso.

Un ejemplo del archivo *input* es el siguiente:

```
1
2  5 2 6 2 4 2     // 5 países con 2, 6, 1, 4, 2 regiones respectivamente
3  8               // 8 eventos
4  ADD_CONTACTS 2 1 0 3
5  ADD_CONTACTS 2 1 1 1 3
6  ADD_CONTACTS 0 0 0 2
7  ADD_CONTACTS 3 0 0 5
8  ADD_CONTACTS 4 1 0 2
9  ADD_CONTACTS 1 2 0 1
10 INFORM 2 1
11 INFORM 0 0
```

Puedes asumir que los eventos detallados en `input` siempre serán válidos y ejecutables para el estado que debería tener el programa.
Los archivos de Test se encontrarán subidos en SIDING, en la carpeta “Material Tareas”.

Output

El `output` esperado corresponde a cada ejecución de los eventos `INFORM` y `STADISTICS` anteriormente descritos. Para el ejemplo de `input` anterior, el `output` correspondiente es el siguiente:

```
INFORM 2 1
0:2
   1:1
      4:0
      5:0
      6:0
    2:1
    3:1
INFORM 0 0
0:2
   1:1
   2:1
```

Recomendamos ver los ejemplos subidos para que tus archivos de `output` finales sean los iguales a los esperados.

Uso de memoria

Parte de los objetivos de esta tarea es que trabajen solicitando y liberando memoria manualmente. Para evaluar esto, usaremos *valgrind*. Se recomienda fuertemente ver los videos de este repositorio.

Análisis

Deberás escribir un informe de análisis donde menciones los siguientes puntos:

- Calcula y justifica la complejidad en notación $\mathcal{O}$ para cada uno de los eventos en función de la cantidad de `Personas` de un `Árbol`. Además deberás entregar la complejidad de búsqueda de un paciente cero.
- Imagina una nueva estructuración de un árbol, en donde los contactos estrechos de una `Persona` se organizan como un arreglo, en lugar de una lista ligada, y además se conoce índice de cada `ID`. ¿Cómo cambia la complejidad de la búsqueda, de inserción y de eliminación de un elemento?
- Compara las velocidades de ejecución de ambos programas escritos en los diferentes lenguajes de programación, `C` y `Python` usando el comando `time` de `bash`. Te entregaremos tests especiales que contienen exclusivamente los eventos `ADD_CONTACTS` e `INFORM` para que los puedas comparar.

Evaluación

La nota de tu tarea se descompone como se detalla a continuación:

- 70% a la nota de tu código, dividido en:
  - 40% a que el output de tu programa sea correcto. (Tests easy y medium)
  - 40% a que el output de tu programa sea eficiente. (Tests hard)
  - 10% a que *valgrind* indique que tu programa no tiene errores de memoria.
  - 10% a que *valgrind* indique que tu programa no tiene *memory leaks*.

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos. De lo contrario, recibirás 0 puntos en ese test.

- 30% a la nota del informe, dividido en:
  - 50% por calcular las complejidades teóricas y justificarlas para cada uno de los eventos.
  - 30% por comentar las ventajas y desventajas en el nuevo modelado de `Persona`.
  - 20% por contrastarlo con los tiempos de Python con `C`.

Entrega

Código: GIT - Rama master del repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Informe: SIDING - En el cuestionario correspondiente, en formato PDF. Sigue las instrucciones del cuestionario. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.