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

IIC2133 — Estructuras de Datos y Algoritmos
2020 - 2

Tarea 0

Fecha de entrega código e informe: Sábado 29 de Agosto

Objetivos

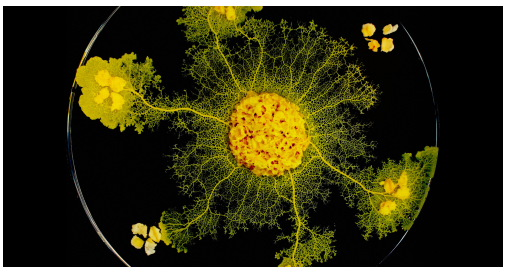
- Familiarizar al estudiante con el lenguaje de programación `C` y compararlo con el lenguaje `Python`
- Entender e implementar una estructura de datos básica
- Simular eventos secuenciales mediante operaciones en dicha estructura de datos

Introducción

Cansado del dulce confort de la vida eucarionte, Guido Van Slimossum ha decidido evolucionar al siguiente nivel de complejidad organizacional. Dado que la vida biológica ha pasado de moda en los últimos años, y las tristes limitantes de la carne son cosa del pasado, Guido ha decidido apuntar a la nueva frontera, la vida artificial.

Su principal inspiración para esto es lo que se conoce como *Slime Mould*, una colonia de células eucariontes independientes que actúan como un solo organismo. Para eso contrató los servicios de *Pyrell Corporation*, los cuales prepararon la lógica de su nueva vida simulada como un *Slime Mould*. Desgraciadamente, dado que esta compañía trabaja exclusivamente con Python, Guido se encontró a si mismo tardándose días en hacer algo tan simple como replicarse, reorganizarse y recordar sus sueños de ovejas eléctricas.

Es por esto que Guido te ha contactado a ti, experto en C, para que lo ayudes a pasar el programa a dicho lenguaje y así abandonar de una vez por todas su existencia terrenal. Afortunadamente, gracias al contrato con *Pyrell Corporation*, tienes a tu disposición la especificación completa del programa en Python 3.6.



Slime Mould buscando comida en una placa petri

Problema

Si bien toda la lógica del problema está escrita en *Python* y tu misión es pasarlo al lenguaje *C*, a continuación se describirá el problema para su mayor entendimiento.

Modelación del Problema

El `SLIME MOULD` se compone de una red de `CELLS` (`C`). La unión de estas `(CELLS)` formarán un conjunto, el cual será el cuerpo del `SLIME MOULD`.

Las conexiones entre las `CELLS` siguen las siguientes reglas biológicas:

- Cada `CELL` tiene una sola `CELL` madre.
- Cada `CELL` puede tener múltiples `CELLS` hijas (hasta 10, enumeradas con índices del 0 al 9).
- La `CELL` germinal, llamada raíz, no tiene madre.

Cada `CELL` es única y se reconocen mediante **un identificador único (`ID`) numérico**. Los `IDs` parten por convención en 0 (el cual le pertenece a la raíz) y se incrementan automáticamente a medida que se van agregando más células. Por ejemplo, para la raíz se tiene la siguiente figura:

Figura 1: CELL que representa a la raíz.

Para recrear el crecimiento de nuestro `SLIME MOULD`, este deberá seguir los eventos descritos en un archivo de `input`, los que representarán exactamente los pasos llevados a cabo para la proliferación de él. Un ejemplo de `SLIME MOULD` se presenta a continuación. En azul se observa el índice de cada hijo para su madre. Por ejemplo, la célula 2 tiene como hijas a las células 4 y 5, que se encuentran en los índices 0 y 9 respectivamente. En rojo podemos visualizar la profundidad a la que está cada célula. Por ejemplo, las células 6, 4 y 5 se encuentran a profundidad 2. La profundidad nos permite determinar el largo del camino a recorrer para encontrar una `CELL` objetivo.

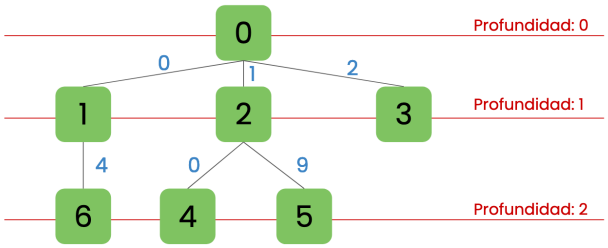


Figura 2: Estructura de ejemplo. En rojo está la profundidad y en azul el `INDEX` de la `CELL` madre

Eventos

Un evento representa una acción ocurre sobre el `SLIME MOULD`. Cada línea del archivo de *input* de la simulación contiene uno de estos eventos, los cuales se ejecutarán en el mismo orden en que se encuentran en el archivo. Para explicar los eventos, se utilizarán los siguientes conceptos:

- `index` (int): Número entero. representa un índice de los hijos de una célula. Sus valores pueden ser del 0 al 9.
- `cell_route`: Indica un camino a recorrer. se representa como `N` x_1 x_2 ... x_N, Donde `N` el número de células a recorrer (generalmente desde la raíz), y x_1 x_2... x_N son los índices por los cuales bajar para llegar a dicha célula.

A continuación se observa un ejemplo de `cell_route`, que comienza desde la raíz. El 3 en azul representa la profundidad `N` de la `CELL` objetivo, por lo que esta tendrá 3 índices subsecuentes para determinar su posición en la estructura. Siguiendo el camino rojo y escogiendo los índices 0, 3 y 2, llegaremos a la `CELL` objetivo de id 8. Si `N = 0`, nos referimos a la raíz.

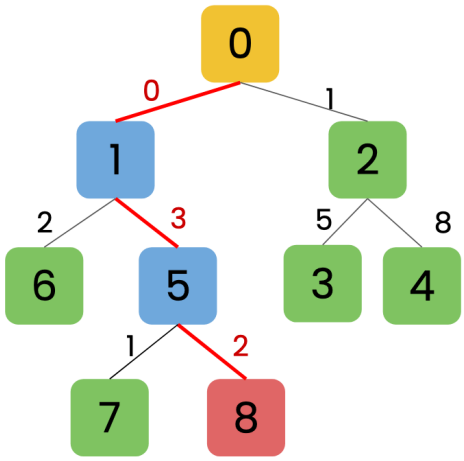


Figura 4: Ruta 3 0 3 2

Con esta información, definimos los siguientes eventos.

`GROW cell_route index`

Una nueva `CELL` es agregada a `SLIME MOULD`. Esta será hija de la última `CELL` de `cell_route`, y dentro de ella ocupará la posición `index`. Puedes asumir que en las instrucciones no habrá ningún `index` inexistente, inválido u ocupado.

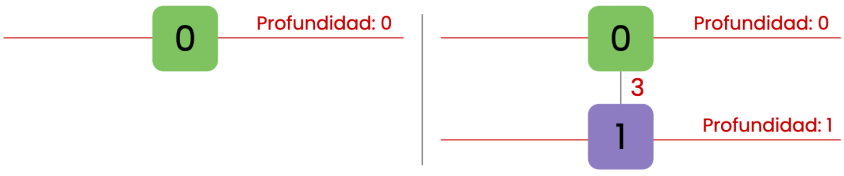


Figura 4: Ejemplo del evento `GROW` 0 3

`BUD cell_route index`

Se elimina la `CELL` en el índice `index` de la célula referenciada por `cell_route`. Además, se eliminan todas las hijas de dicha `CELL`.

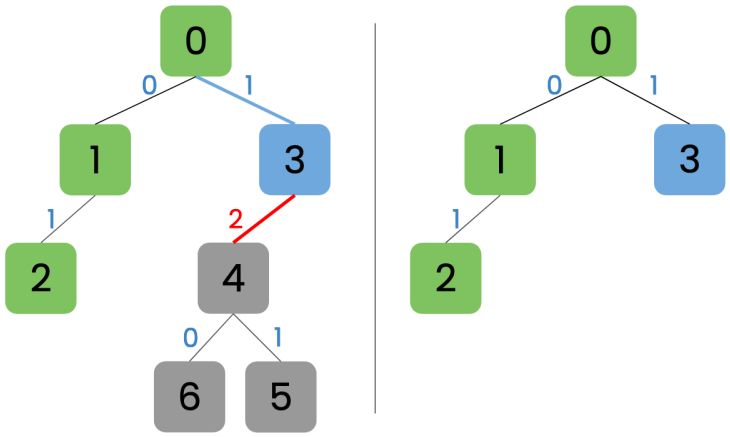


Figura 5: Ejemplo `BUD` 1 1 2

`CLONE cell_route_1 cell_route_2 index`

Se duplica un conjunto de `CELLS` de `SLIME MOULD`. Se copia todo el conjunto de `CELLS` desde la última `CELL` de `cell_route_1` hacia abajo, y se inserta en el *mismo* orden que tenían las `CELLS` originales, como hija de la última `CELL` de `cell_route_2` en la posición `index`.

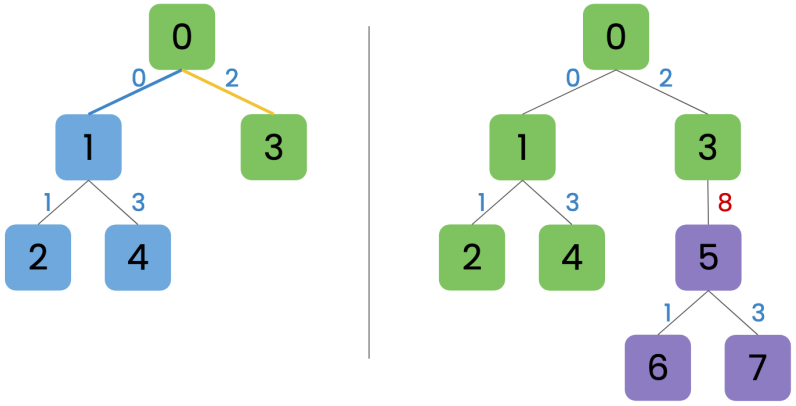


Figura 6: Ejemplo `CLONE` 1 0 1 2 8

`CROSSOVER cell_route_1 index_1 cell_route_2 index_2`

Sean $\texttt{CELL}_1$ y $\texttt{CELL}_2$ las últimas `CELL` de `cell_route_1` y `cell_route_2` respectivamente. Se permuta el conjunto de `CELLS` de la posición `index_1` del $\texttt{CELL}_1$ con el conjunto de `CELLS` de la posición `index_2` del $\texttt{CELL}_2$. Así, ahora en la posición `index_1` de la $\texttt{CELL}_1$ pasa a estar el conjunto de `CELLS` que se encontraba en la posición `index_2` de la $\texttt{CELL}_2$, y viceversa. Los hijos de cada `CELL` afectado se mantienen *iguales*.

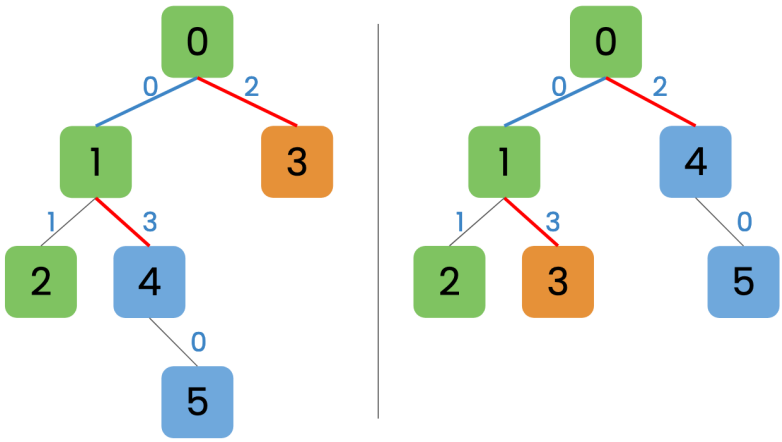


Figura 7: Ejemplo `CROSSOVER 1 0 3 0 2`

`ABSORB cell_route_1 index cell_route_2`

Se elimina un tramo de `SLIME MOULD`. Para llevar esto a cabo, conecta la $\texttt{CELL}_1$ encontrada a través de la `cell_route_1` con la $\texttt{CELL}_2$ encontrado a través de la `cell_route_2`. El índice `index` indica la posición de la hija de la $\texttt{CELL}_1$ desde la cual se debe recorrer la `cell_route_2` para encontrar la $\texttt{CELL}_2$. Así $\texttt{CELL}_2$ siempre será hija de $\texttt{CELL}_1$. Todas las `CELLS` que se encuentran entre la $\texttt{CELL}_1$ y la $\texttt{CELL}_2$ y sus respectivas hijas son eliminadas.

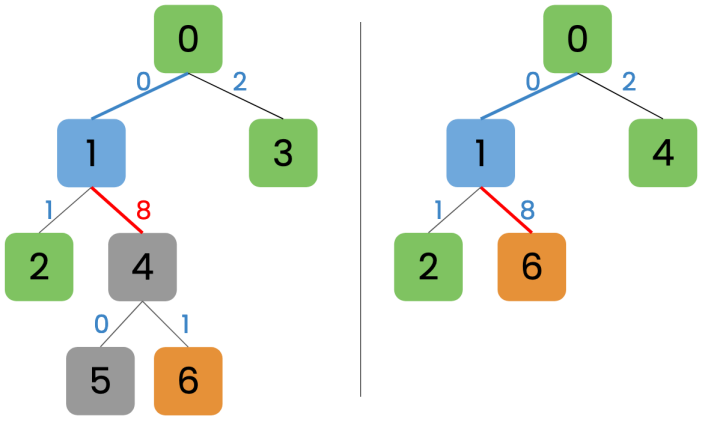


Figura 8: Ejemplo `ABSORB 1 0 8 1 1`

`OBSERVE cell_route`

Para saber la situación actual del `SLIME MOULD` durante la simulación y saber cómo va su evolución, será necesario imprimir el estado actual de su configuración de la `CELL` determinada por `cell_route`. Esta configuración tendrá la siguiente representación impresa en el archivo `output`. cada célula se representará con el formato `index:id`, y para el caso de la célula germinal, su formato será `r:0`. Los `index` deben ser impresos de menor a mayor, y en caso de no haber una `CELL` hija en ese `index`, no imprimir esa línea.

```
STATE
index_0:id_A
   index_0:id_B // index perteneciente a CELL A
      index_1:id_F // index perteneciente a CELL B
      .
      .
   index_1: id_C
      index_0: id_D
      .
      .
   index_k:id_E
   .
   .
```

A continuación se mostrarán dos ejemplos de output de la función `OBSERVE` para la siguiente estructura de `SLIME MOULD`. Se destaca que por cada profundidad nueva se agregan tres espacios.

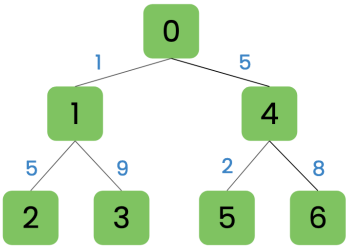


Figura 9: Ejemplo estructura para `OBSERVE`

Ejemplo 1: `OBSERVE 0`

```
STATE
r:0
   1:1
      5:2
      9:3
   5:4
      2:5
      8:6
```

Ejemplo 2: `OBSERVE 1 1`

```
STATE
1:1
   5:2
   9:3
```

Ejecución

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

```
./simulate <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 tiene un único número `N` que indica la cantidad de eventos que habrán, es decir, la cantidad de líneas futuras.
- Luego, cada una de las siguientes `N` líneas contiene un evento de los anteriormente descritos. Los eventos pueden repetirse.

Un ejemplo del archivo *input* es el siguiente:

```
11
GROW 0 0
GROW 0 1
GROW 1 1 0
GROW 2 1 0 0
CLONE 1 1 0 2
GROW 1 1 2
OBSERVE 1 1
CROSSOVER 1 1 0 1 2 0
BUD 0 1
ABSORB 0 2 2 0 0
OBSERVE 0
```

Puedes asumir que los eventos detallados en `input` siempre serán válidos y ejecutables para el estado que debería tener el programa. En particular:

- Las rutas e índices de todos los eventos siempre serán válidos.
- No se realizará el evento `BUD` sobre la `CELL` germinal.
- No se realizará el evento `ABSORB` sobre una `CELL` que no sea descendiente de otra.
- No se realizará un evento `CLONE` de dos `CELLS` en donde una sea descendiente de otra.

Cabe destacar que **todo lo relacionado a la lectura del archivo ya viene implementado** en sus respectivos repositorios.

Los archivos de Test se encuentran subidos en SIDING, en la carpeta "Material Tareas".

Output

El `output` esperado corresponde a cada ejecución del evento `OBSERVE` anteriormente descrito. Para el ejemplo de `input` anterior, el `output` correspondiente es el siguiente:

```
STATE
1:2
   0:3
      0:4
   2:8
STATE
r:0
   0:1
   2:4
```

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 leer el código disponible en el repositorio principal dentro de la carpeta `Ayudantías/Ayudantía 0 - Intro a C/Aprende C`. El archivo `src/6_memoria/main.c` incluye en sus comentarios como interpretar el output de *valgrind*. El viernes 14 veremos en clases como usar esta herramienta.

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 `CELLS` de `SLIME MOULD`. Debes simplificar la expresión lo más posible.
- Compares las velocidades de ejecución de ambos programas escritos en los diferentes lenguajes de programación, `C` y `Python` usando el comando `time` de `bash`. Además, deberás hacer una pequeña investigación y describir, en no más de media plana, por qué ocurren estas diferencias de rendimiento.

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*.
- 30% a la nota del informe, dividido en:
  - 70% por calcular las complejidades teóricas y justificarlas para cada uno de los eventos.
  - 30% por contrastarlo con los tiempos de Python y la investigación.

Entrega

Código: GIT - 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.