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

IIC2133 — Estructuras de Datos y Algoritmos
Segundo semestre 2017

Tarea 0

Fecha de entrega: Jueves 24 de Agosto

Objetivos

- Familiarizar al alumno con el lenguaje de programación C
- Implementar y analizar un algoritmo básico.

Introducción

Los investigadores del centro de estudios biológicos INCUBACT desean simular el comportamiento de las bacterias que estudian arduamente. Por lo anterior, te han contactado para desarrollar el algoritmo encargado de esta tarea y te han provisto de una interfaz gráfica con la que te tendrás que comunicar para mostrar la evolución del ecosistema. Esperan que a partir de un estado inicial de la comunidad bacteriana se puedan predecir el estado de las siguientes generaciones, utilizando las características del ambiente e interacciones entre bacterias. Esta información se detalla a continuación.

Mundo

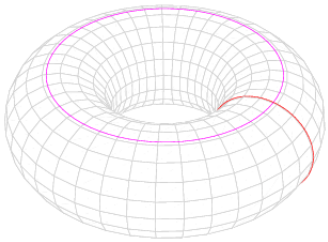


Figura 1: Toro

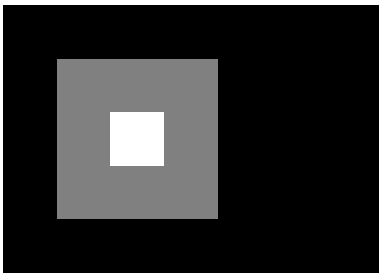
El ambiente en que se desarrollarán las bacterias corresponde a un toro (un ejemplo de esta superficie se muestra en la Figura 1), el cual visualmente se representará a través de una cuadrícula de dimensiones M x N de tal forma que la siguiente columna luego de la última es la primera columna y la siguiente fila luego de la última fila es la primera fila.

Cada celda de la cuadrícula tiene 2 estados posibles:

- *Poblada*: Existe un bacteria viva en esa posición de la cuadrícula, representada con el color blanco.
- *Despoblada*: No existen una bacterias en esta celda, representada con el color negro.

Interacciones entre las bacterias

La aparición o desaparición de bacterias en una celda depende de las celdas vecinas. Se considera que una bacteria está alrededor de otra si se encuentra en una celda adyacente o diagonal.



(a) Vecindad, en gris, de una bacteria

(b) Misma vecindad, en otra posicion

Las reglas que modelan el comportamiento de las bacterias en la cuadrícula se detallan a continuación:

1. Si una bacteria está rodeada por menos de 2 bacterias, muere y la celda pasa a estar despoblada, debido al aislamiento.
2. Si una bacteria está rodeada por 2 o 3 bacterias sobrevive y pasa a la siguiente generación.
3. Si una bacteria está rodeada por más de 3 bacterias muere por sobrepoblación.
4. Si una celda despoblada está rodeada de 3 bacterias se genera una bacteria en ella y la celda pasa a estar poblada.

Interfaz de la GUI

Como ya te hemos mencionado anteriormente los investigadores ya cuentan con una GUI implementada para poder visualizar la evolución del ecosistema de bacterias. Los detalles de los métodos disponibles se dan a continuacion:

- `watcher_open(height, width)`: Abre una interfaz con una grilla que contiene `height` filas y `width` columnas.
- `watcher_set_cell(row, col, alive)`: Se cambia el estado de la celda en la fila `row` y columna `col` según el `bool alive`. Si `alive` es verdadero, significa que la celda está poblada y pasará a ser de color blanco. En el caso contrario será de color negro.
- `watcher_refresh()`: Actualiza la imagen de la interfaz. Se recomienda usar este método después de calcular el nuevo estado de todas las celdas.
- `watcher_close()`: Cierra la interfaz usada.

Análisis

Deberás entregar un informe en el que menciones todos los detalles que creas necesarios de tu implementación y además debes realizar un análisis del algoritmo en cuanto a complejidad, tiempo de ejecución y uso de memoria.

Input

Tu tarea esta debe recibir los siguientes inputs:

1. La ruta del archivo `.txt` que representa la generación inicial de bacterias.
2. La cantidad de generaciones.

El archivo .txt el cual posee la siguiente estructura:

- La primera línea contiene los números `M` y `N` que indican la cantidad de filas y columnas que tendrá la grilla, respectivamente.
- Cada una de las siguientes `M` líneas corresponderán a `N` números, 0 ó 1 separados por espacios que denotan los estados de las celdas. Será 1 si la celda está poblada y 0 si está despoblada.

Las posteriores generaciones se encuentran aplicando las reglas antes descritas de manera simultánea.

Output

El único *output* de tu programa debe ser la visualización del mundo en el tablero, por lo cual debes asegurarte que tanto los resultados intermedios como el resultado final se muestren correctamente. Debes utilizar la funcion `watcher_refresh()` solo una vez por generación.

Evaluación

Tu código será evaluado con tests aleatorios (ver Anexo) y se evaluará cada generación según lo que hayas enviado al watcher.

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

- 50% que las asignaciones de los resultados intermedios y final sean correctas
- 20% a la calidad del código
  - 10% a que *valgrind* indique que no contiene **errores**
  - 10% a que valgrind indique que no tiene **memory leaks**
- 30% a tu análisis de complejidad. Específicamente,
  - 20% a que sea correcto.
  - 10% a que la complejidad dea la esperada.

Entrega

- **Lugar:** GIT - Repositorio asignado (asegúrate de seguir la estructura inicial de éste).
  - En la carpeta *Programa* debe encontrarse el código.
  - En la carpeta *Informe* debe encontrarse un archivo `Informe.pdf` con el análisis de la tarea.
  - Se recogerá el estado en la rama *master*.
- **Hora:** 1 minuto pasadas las 23:59 del día de la entrega.
- Se espera que el código compile con `make` dentro de la carpeta *Programa* y genere un ejecutable de nombre `simulate` en esa misma carpeta.
- No se permitirán entregas atrasadas.

Bonus

- **Buen uso de espacio y del formato (+5% a la nota del *Informe*)** :
  La nota de tu informe aumentará un 5% si tu informe está bien presentado y usa el espacio y formato de manera de transmitir mejor la información. A juicio del corrector.

Anexo: Generador

Además de los tests que vienen en tu repositorio, puedes usar el generador para probar tu programa con tests aleatorios.

Este recibe 4 parámetros:

1. La ruta al archivo donde se guardará el test generado
2. La cantidad de filas que deberá tener la cuadrícula
3. La cantidad de columnas que deberá tener la cuadrícula
4. La semilla aleatoria, para poder replicar resultados