ESCUELA DE CIENCIAS DE LA
COMPUTACIÓN E INFORMÁTICA
UNIVERSIDAD DE COSTA RICA

CI-0113 Programción 2
II semestre 2018
Prf.: Alan Calderón Castro

Tercer laboratorio guiado de programación

Objetivos

Al completar este laboratorio el estudiante debería ser capaz de:

1. Explicar los conceptos básicos sobre plantillas.
2. Explicar principios básicos para el diseño de plantillas.
3. Programar y depurar plantillas.

Descripción del problema

Se deberá elaborar un programa que realice una simulación de la dispersión de un virus en una red de computadoras similar a la que se muestra en: http://netlogoweb.org/launch#http://netlogoweb.org/assets/modelslib/Sample%20Models/Networks/Virus%20on%20a%20Network.nlogo. La resolución del problema se deberá basar en el diseño provisto por el docente, lo que implica que los archivos de encabezados de las clases provistas por el docente NO PODRÁN SER MODIFICADOS A MENOS QUE ASÍ LO AUTORICE EXPLÍCITAMENTE.

De acuerdo con lo anterior usted deberá programar las siguientes clases:

Nombre de la clase	Función que cumple
GrafoGnr< T >	Plantilla de clase que representa un grafo con vértices tipo T.
Grafo	Representa la red de computadoras en la que se dispersará el virus (GrafoGnr< Nodo >).
Nodo	Representa el estado de una computadora o nodo de la red.
Simulador	Ejecuta el proceso de simulación de la dispersión de virus.

Los parámetros de entrada de la simulación son:
1. Cantidad de nodos en la red (“number-of-nodes” → N).
2. Cantidad de nodos contagiados inicialmente (“initial-outbreak-size” → ios).
3. Probabilidad de contagio (“virus-spread-chance” → vsc).
4. Frecuencia de chequeo anti-virus (“virus-check-frequency” → vcf).
5. Probabilidad de eliminación de infección en un nodo (“recovery-chance” → rc).
6. Probabilidad de generar resistencia (“gain-resistance-chance” → grc).

Las salidas de la simulación son:
1. Por tic: promedio, porcentaje y cantidad actual de nodos susceptibles.
2. Por tic: promedio, porcentaje y cantidad actual de nodos infectados.
3. Por tic: promedio, porcentaje y cantidad actual de nodos resistentes.
4. Promedio, porcentaje y cantidades FINALES de nodos en S y en R.
5. Cantidad de tics que se necesitaron para que se estabilizara la red. La red se estabiliza cuando no queda ningún nodo en I, todos están en R o S.

La topología de la red de nodos o computadoras se derivan por uno de tres métodos o constructores:
1. Red de nodos aleatoria.
2. Red de nodos basada en un archivo de datos. El formato del archivo de datos es el siguiente:

```
100 #representa la cantidad de nodos
12,25,43,45,67, # representa los nodos adyacentes al nodo 0
0,1,4,44,67,89,90,93 # representa los nodos adyacentes al nodo 1
… # deberá aparecer una línea por cada uno de los 100 nodos
```

3. Red de nodos basada en la topología de “pequeños mundos”, ver: https://es.wikipedia.org/wiki/Red_de_mundo_peque%C3%B1o. La topología de “pequeño mundo” en una red se logra mediante el algoritmo de Watts y Strogratz descrito en https://en.wikipedia.org/wiki/Watts%E2%80%93Strogatz_model. Este algoritmo se deberá implementar para crear la red de computadoras al azar. Los parámetros de entrada del algoritmo de Watts-Strogatz son:

1. El promedio de grado de los nodos (K).
2. Probabilidad de re-enlace (ß).

La salida del algoritmo de Watts-Strogatz es una red (o grafo) que muestra el efecto de “pequeño mundo”, característica topológica distintiva de muchas redes, en particular la Internet.

Las reglas para la simulación del proceso de dispersión del virus son las siguientes:

1. Las computadoras pueden estar en uno de tres estados: susceptible (S), infectada (I), resistente (R). Una computadora en S puede ser contagiada por alguna de sus vecinas (computadoras conectadas o adyacentes). Una computadora en I puede contagiar a sus vecinas o pasar a estado R. Una computadora en R no puede ser contagiada.
2. Dado un nodo N en S con K vecinos (o nodos adyacentes), de los cuales K’ <= K están en I, puede ser contagiado por cualquiera de los K’ vecinos en I con la “probabilidad de contagio”. Cada posibilidad de contagio se deberá simular con un evento de generación de número aleatorio independiente.
3. Dado un nodo N en I, puede pasar a S si en el tic actual toca aplicar el “chequeo anti-virus” con “probabilidad de eliminación de infección”.
4. Dado un nodo N en I que ha logrado eliminar la infección en el mismo tic, puede pasar a R con “probabilidad de generar resistencia”. El estado instantáneo de S no se representa porque todo sucede en el mismo tic, de manera que pasa de I → S → R en el mismo tic, y sólo se representa finalmente el estado R. Este sub-proceso debe simularse en dos eventos independientes: “eliminación de infección” y “generación de resistencia”.
5. Un nodo N en R, NO puede pasar a S ni a I.

El funcionamiento del programa “main()” consiste en:
1. leer un archivo de experimentos como el que se muestra a continuación,
2. validar los datos del archivo de experimentos y generar mensajes adecuados si no son correctos,
3. controlar la simulación y detenerla cuando ya no haya nodos infectados,
4. generar la salida por consola y el archivo.

El archivo de experimentos tendrá el siguiente formato:

```
10 0.6 3 0.3 0.2 10 3500 2 6 0.9
10 0.6 3 0.3 0.2 10 500 2 6 0.9
10 0.6 3 0.3 0.2 10 150 2 6 0.9
10 0.6 3 0.3 0.2 10 500 1 0.3
10 0.6 3 0.3 0.2 10 150 1 0.3
10 0.6 3 0.3 0.2 10 3 red_peq.txt
```

Donde por ejemplo la primera línea indica:

10 = porcentaje de nodos infectados inicialmente (ios),
0.6 = probabilidad de contagio (vsc),
3 = frecuencia de chequeo antivirus (VCF),
0.3 = probabilidad de recuperación (rc),
0.2 = probabilidad de generar resistencia (grc),
10 = cantidad de veces que se deberá repetir el experimento,
3500 = total de nodos en la red (N),
2 = la red se genera con base en algoritmo de “pequeños mundos”,
6 = promedio de adyacencias por nodo,
0.9 = beta.

Las líneas #4 y #5 comparten la función de los primeros 7 valores, los demás se deben interpretar así:
1 = la red de nodos se genera aleatoriamente con,
p = 0.3 la probabilidad de que exista una adyacencia entre cualesquiera dos nodos.

La última línea comparte la función de los primeros 6 valores, los demás se deben interpretar así:
3 = la red de nodos se genera con base en el archivo de nombre “red_peq.txt”.

Cronograma IDEAL: (el orden de trabajo es obligatorio)

Del 12/noviembre al 19/noviembre: 1 semana
+ Programar la plantilla GrafoGnr< T >.
Del 19/noviembre al 26/noviembre: 1 semana
+ Re-programar la clase Grafo especializando GrafoGnr< T > con T == Nodo.
+ Aplicar las pruebas y depurar la clase Grafo.
Del 26/noviembre al 2/diciembre: 1 semana
+ Re-programar al clase Simulador.
+ Usar el archivo de experimetos para depurar el “main()” y “Simulador”.

Entrega: domingo 2 de diciembre a las 23:55pm.
SÓLO DEBERÁ SUBIR LOS ARCHIVOS DE CÓDIGO FUENTE (*.h y *.cpp) Y LOS DATOS.

Procedimiento

Este es un laboratorio guiado y por tanto la resolución del problema planteado deberá realizarse en el siguiente orden:

1. Programar la plantilla de clase GrafoGnr< T >. No se programarán pruebas directas a esta plantilla, en su lugar se aplicarán las mismas pruebas desarrolladas para la clase Grafo.
2. Adaptar el código de la clase Grafo usando la especialización GrafoGnr< Nodo >.
3. Aplicar el programa de pruebas elaborado en el primer laboratorio guiado para depurar la nueva clase Grafo adaptada con GrafoGnr< Nodo >. Consecuentemente se estará depurando la plantilla GrafoGnr< T >.
4. Realizar pequeñas adaptaciones al código de la clase “Simulador” y al “main()”.
5. Depurar “Simulador” y el “main()” con base en el archivo de experimentos.

NOTA: para depurar el Simulador se recomienda hacer pruebas paso a paso, es decir, mediante “simular()”.

Criterios de evaluación

La nota final de su trabajo dependerá de si en su programa se:

1. Respetan las reglas de estilo del código: márgenes, nombres de objetos empiezan en minúsculas, nombres de clases empiezan en mayúsculas, nombres de métodos también empiezan en minúscula pero se usan mayúsculas para concatenar palabras, comentarios para los atributos y variables de métodos. Formato automático dado por NetBeans.
2. El código es lógicamente lo más simple posible.
3. Implementan los métodos de manera eficiente.
4. Hace uso óptimo de la memoria usando las estructuras de datos discutidas en clase para cada una de las clases.
5. Respeta la división de responsabilidades entre el controlador-modelo de manera que sólo el main() se ocupa de la entrada de datos, generar mensajes de error, el despliegue de ciertos resultados por la "Consola", así como invocar a los demás objetos para que realicen todos los procesamientos necesarios.

Fecha de entrega: domingo 2 de diciembre a las 23:55pm por medio del enlace en el sitio del curso. SÓLO DEBERÁ SUBIR LOS ARCHIVOS DE CÓDIGO FUENTE (*.h y *.cpp) Y LOS DATOS.

Evaluación

Criterio	%Prc
Reglas de estilo y división de responsabilidades controlador-modelo	5
Simplicidad lógica del del main	5
Eficacia según pruebas, eficiencia y simplicidad de Grafo	50
Eficacia según pruebas, eficiencia y simplicidad de Simulador y main()	40
Hasta 10/100 puntos extra por graficación con freeglut	10
Hasta 10/100 puntos extra por un buen reporte de errores cuando **no** funcione	

Notas importantes:
1. Este proyecto deberá realizarse idealmente y a lo más en parejas. NO SE ACEPTARÁ NINGÚN TRABAJO ELABORADO POR MÁS DE DOS PERSONAS.
2. Cada hora de atraso en la entrega se penalizará con -1/100, lo que se aplicará a la nota obtenida.
3. A TODOS LOS ESTUDIANTES INVOLUCRADOS EN UN FRAUDE SE LES APLICARÁ EL ARTÍCULO #5 INCISO C DEL "Reglamento de Orden y Disciplina de los Estudiantes de la Universidad de Costa Rica".
4. NO SUBA ningún otro archivo que no sea de código fuente (*.h y *.cpp) o de datos para evitar la transmisión de virus.