Práctica 3. Divide y vencerás

21 de noviembre de 2017

Esta práctica consiste en la evaluación de diferentes estrategias para la implementación de la función de selección para el caso de un algoritmo voraz similar al que diseñó en la primera práctica. Antes de comenzar el desarrollo de esta tercera práctica debe situarse en el directorio *BASE/p1* y ejecutar las siguientes órdenes para anular la estrategia para la colocación de las defensas definida en la primera práctica.

```
rm libDefenseStrategy.so
cp libDefenseStrategy.null.so libDefenseStrategy.so
```

La forma de generar la biblioteca dinámica es idéntica a la utilizada en la primera práctica, con la salvedad de que en este caso la función en la que debe implementar su algoritmo se denomina *placeDefenses3*. Copie el algoritmo que implementó en el fichero *BASE/p1/DefenseStrategy.cpp* y sustituya la llamada a la función *cellValue* por una llamada a la función *defaultCellValue*, cuya implementación puede encontrar en el fichero *BASE/p3/DefenseStrategy.example.cpp*. Extraiga las celdas en orden decreciente de valor (primero las de mayor puntuación). Añada todas las instrucciones necesarias para resolver los siguientes ejercicios. Utilice la clase *cronometro*, definida en el fichero *cronometro.h*, para realizar las mediciones. En el fichero *DefenseStrategy.example.cpp* tiene un ejemplo de su uso.

Evalúe la eficiencia del algoritmo desarrollado en la práctica 1 en función del número de defensas a colocar y celdas en las que se divide el planeta, dadas las siguientes circunstancias:

1. Sin pre-ordenación: no se ordenan las celdas del terreno de batalla de acuerdo a su valor. En su lugar, han de recorrerse todas las celdas del terreno de batalla cada vez que se busca la celda más prometedora para colocar una defensa.

2. Pre-ordenación usando el algoritmo de ordenación por fusión: deberá implementar su propia versión de dicho algoritmo.

3. Pre-ordenación usando el algoritmo de ordenación rápida: deberá implementar su propia versión de dicho algoritmo.

4. Uso de un montículo: se utiliza una estructura de montículo para almacenar las celdas del terreno de batalla. Puede utilizar la implementación de la estructura de montículo disponible en la biblioteca estándar.

Repita el algoritmo dentro de la función *placeDefenses3* tantas veces estime oportuno, de forma que en cada ejecución del simulador se obtenga el tiempo empleado en todos los casos, sin necesidad de volver a compilar la biblioteca. Recuerde que en cada ejecución del simulador se asalta un único planeta, por lo que para generar la gráfica serán necesarias varias ejecuciones del mismo. Este proceso puede ser automatizado ejecutando la siguiente orden.

```
make data
```

Configure su biblioteca de tal forma que el resultado obtenido por pantalla coincida con el formato del contenido del fichero *data.txt*. Puede generar una gráfica conjunta a partir de estos valores usando la herramienta *gnuplot*, ejecutando la siguiente orden.

```
make plot
```

Tras analizar el resultado, modifique su algoritmo y/o seleccione la estrategia para la función de selección que mejor rendimiento ofrece en general. Puede implementar un algoritmo que use una u otra estrategia en función de las características del planeta. No olvide realizar antes una copia del código fuente para poder contestar al ejercicio 6.

En la competición asociada a esta práctica se tomará en cuenta el tiempo requerido para realizar una llamada completa a la función *placeDefenses3*, por lo que debe tratar de evitar el uso de instrucciones innecesarias en ella. El simulador incorpora varias opciones para facilitar esta medición[1]. Sitúese en el directorio *BASE/p3* y use la siguiente orden para obtener el tiempo empleado por la función *placeDefenses3*. El simulador utilizará un esquema indirecto y adaptativo de medida en el que se considerará un error absoluto de 0.01 y uno relativo de 0.1.

```
make time
```

1. Ejercicios

1. Describa las estructuras de datos utilizados en cada caso para la representación del terreno de batalla.

2. Implemente su propia versión del algoritmo de ordenación por fusión. Muestre a continuación el código fuente relevante.

3. Implemente su propia versión del algoritmo de ordenación rápida. Muestre a continuación el código fuente relevante.

4. Realice pruebas de caja negra para asegurar el correcto funcionamiento de los algoritmos de ordenación implementados en los ejercicios anteriores. Detalle a continuación el código relevante.

5. Analice de forma teórica la complejidad de las diferentes versiones del algoritmo de colocación de defensas en función de la estructura de representación del terreno de batalla elegida. Comente a continuación los resultados. Suponga un terreno de batalla cuadrado en todos los casos.

6. Incluya a continuación una gráfica con los resultados obtenidos. Utilice un esquema indirecto de medida (considere un error absoluto de valor 0.01 y un error relativo de valor 0.001). Es recomendable que diseñe y utilice su propio código para la medición de tiempos en lugar de usar la opción *-time-placeDefenses3* del simulador. Considere en su análisis los planetas con códigos 1500, 2500, 3500,..., 10500, al menos. Puede incluir en su análisis otros planetas que considere oportunos para justificar los resultados. Muestre a continuación el código relevante utilizado para la toma de tiempos y la realización de la gráfica.

Todos los ejercicios tienen la misma puntuación. No es necesario que explique el código fuente en los ejercicios en los que se le solicita que lo incluya. Puede incrustar en el código los comentarios que considere oportunos.

[1] A partir de la versión 1.7.