Práctica 3. Divide y vencerás

Nombre Apellido1 Apellido2
correo@servidor.com
Teléfono: xxxxxxxx
NIF: xxxxxxxxm

13 de diciembre de 2022

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

   En este caso, se ha utilizado un vector de flotantes. Este es de tamaño nCellsHeight * nCellsWidth, ya que de esto modo podemos tener todos los elemetos de todas las celdas, como en una matriz pero en forma de vector, de modo que si por ejemplo queremos acceder a la 3 fila y la 4 columna, teniendo 5 columnas en esta matriz, esa celda es la posicion 3 * 5 + 4 del cector, es decir, 19.

   Des esta forma accedemos facilmemte en cualquier momento a cualquier celda, y si lo que queremos es de la posicion por ejemplo 21 del caso anterior, la fila de daca haciendo 21 / 5 para la fila y 21 % 5 para columna.

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

   void merge(int* vec, int tam, int izquierda, int medio, int derecha) int h = izquierda ,i = izquierda ,j = medio + 1; int res[tam];

   while((h ¡= medio) && (j ¡= derecha)) if(vec[h] ¿= vec[j]) res[i] = vec[h]; h++; else res[i] = vec[j]; j++; i++; if(h ¿ medio) for(int k = j; k ¡= derecha; k++) res[i] = vec[k]; i++; else for(int k = h; k¡=medio; k++) res[i] = vec[k]; i++; for(int k = izquierda; k¡=derecha; k++) vec[k] = res[k];

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

   void quicksort(int* vec,int pri,int ult) int med = (pri+ult)/2, i = pri, j = ult; double pivote = vec[med];

   do while(vec[i] ¡ pivote) i++; while(vec[j] ¿ pivote) j–; if(i¿=j) int aux = vec[i]; vec[i] = vec[j]; vec[j] = aux; i++; j–; while(i ¡= j); if(pri ¡ j) quicksort(vec, pri, j); /*mismo proceso con sublista izquierda*/ if(i¡ ult) quicksort(vec, i, ult); /*mismo proceso con sublista derecha*/

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.

   Escriba aquí su respuesta al ejercicio 4.

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.

   Escriba aquí su respuesta al ejercicio 5.

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.

   Escriba aquí su respuesta al ejercicio 6.

Todo el material incluido en esta memoria y en los ficheros asociados es de mi autoría o ha sido facilitado por los profesores de la asignatura. Haciendo entrega de este documento confirmo que he leído la normativa de la asignatura, incluido el punto que respecta al uso de material no original.