Práctica 3. Divide y vencerás

Jesús Rodríguez Heras
jesus.rodriguezheras@alum.uca.es
Teléfono: 628576107
NIF: 32088516C

16 de enero de 2019

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

   - **Sin ordenación:** En el caso en el que no tenemos ordenación, he usado una matriz de tipo `float` del número de celdas de ancho (de anchura) por el número de celdas de alto (de altura).
   - **Con ordenación:** En los casos en los que sí tenemos ordenación, he creado una clase llamada `posicionConValor`, la cual guarda tanto la posición (dos enteros) como el valor (`float`) de una celda.

```
class posicionConValor{
public:
    int i, j;
    float valor;
    posicionConValor(){i=j=0;valor=0.0;};

    bool operator <(posicionConValor p){
        return this->valor < p.valor;
    }

    bool operator >(posicionConValor p){
        return this->valor > p.valor;
    }

    bool operator ==(posicionConValor p){
        return this->valor == p.valor;
    }

    bool operator >=(posicionConValor p){
        return this->valor >= p.valor;
    }

    bool operator <=(posicionConValor p){
        return this->valor <= p.valor;
    }
};
```

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

```
void fusion(posicionConValor* mapaOrdenado, int i, int k, int j){
    int n = j-i+1;
    int p = i;
    int q = k+1;
    posicionConValor w[n];

    for (int l = 0; l < n; ++l) {
        if ((p <= k) && (q > j || mapaOrdenado[p] > mapaOrdenado[q])) {
            w[l] = mapaOrdenado[p];
            ++p;
        }else{
            w[l] = mapaOrdenado[q];
            ++q;
        }
```

```
    }

    for (int l = 0; l < n; ++l) {
        mapaOrdenado[i+l] = w[l];
    }
}

void ordenacionFusion(posicionConValor* mapaOrdenado, int i, int j){
    int n = j-i+1;
    if (n > 1) {
        int k = i-1+(n/2);
        ordenacionFusion(mapaOrdenado, i, k);
        ordenacionFusion(mapaOrdenado, k+1, j);
        fusion(mapaOrdenado, i, k, j);
    }
}
```

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

```
int pivote(posicionConValor* mapaOrdenado, int i, int j){
    int p = i;
    posicionConValor x = mapaOrdenado[i];
    posicionConValor aux;
    for (int k = i+1; k <= j; ++k) {
        if (mapaOrdenado[k] > x) {
            ++p;
            aux = mapaOrdenado[k];
            mapaOrdenado[k] = mapaOrdenado[p];
            mapaOrdenado[p] = aux;
        }
    }
    mapaOrdenado[i] = mapaOrdenado[p];
    mapaOrdenado[p] = x;
    return p;
}

void ordenacionRapido(posicionConValor* mapaOrdenado, int i, int j){
    int n = j-i+1;
    if (n > 1) {
        int p = pivote(mapaOrdenado, i, j);
        ordenacionRapido(mapaOrdenado, i, p-1);
        ordenacionRapido(mapaOrdenado, p+1, j);
    }
}
```

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.

   - **Ordenación por fusión:**

```
//Codigo anterior
ordenacionFusion(mapaOrdenado, 0, nCellsWidth*nCellsHeight);

for (int i = 0; i < (nCellsHeight*nCellsWidth - 1); ++i) {
    if (mapaOrdenado[i].valor < mapaOrdenado[i+1].valor) {
        std::cout << "Fallo en el algoritmo de fusion" << '\n';
        cajaNegraFusion = true; // Se pone a true cuando falla el algoritmo
    }
}
```

   - **Ordenación rápida:**

```
//Codigo anterior
ordenacionRapido(mapaOrdenado, 0, nCellsWidth*nCellsHeight);

for (int i = 0; i < (nCellsHeight*nCellsWidth - 1); ++i) {
    if (mapaOrdenado[i].valor < mapaOrdenado[i+1].valor) {
        std::cout << "Fallo en el algoritmo de ordenacion rapida" << '\n';
```

```
            cajaNegraRapido = true; // Se pone a true cuando falla el algoritmo
        }
    }
```

- **Ordenación por montículo:**

```
std::make_heap(vectorOrdenado.begin(), vectorOrdenado.end());
std::sort_heap(vectorOrdenado.begin(), vectorOrdenado.end());

for (int i = 0; i < (nCellsHeight*nCellsWidth - 1); ++i) {
    if (vectorOrdenado[i].valor > vectorOrdenado[i+1].valor) {
        std::cout << "Fallo en el algoritmo de monticulo" << '\n';
        cajaNegraMonticulo = true; // Se pone a true cuando falla el algoritmo
    }
}
```

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.

   - **Sin ordenación:** Para este caso, el código para colocar las defensas es de orden $\theta(n^2)$.
   - **Con ordenación:**
     - **Montículo:** Para este caso, el algoritmo de colocación de defensas es de orden $\theta(n \cdot log_2(n))$.
     - **Fusión y ordenación rápida:** En estos casos tenemos tiempos en el orden de $O(n \cdot log_2(n))$.

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.

   El algoritmo usado para la toma de tiempos es el siguiente:

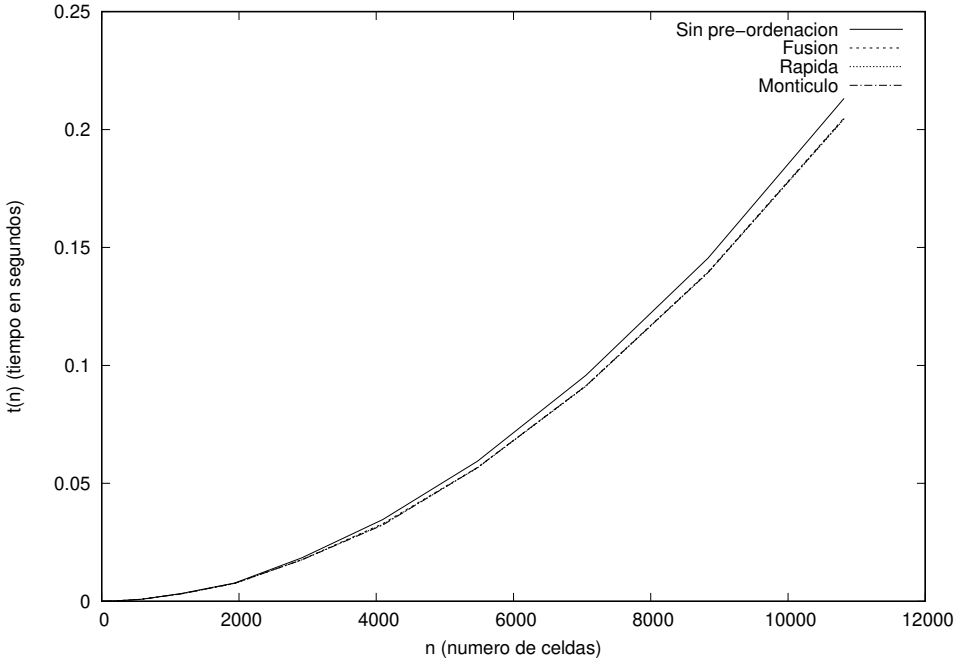
```
void DEF_LIB_EXPORTED placeDefenses3(bool** freeCells, int nCellsWidth, int nCellsHeight,
    float mapWidth, float mapHeight, List<Object*> obstacles, List<Defense*> defenses) {

    float cellWidth = mapWidth / nCellsWidth;
    float cellHeight = mapHeight / nCellsHeight;

    cronometro c1;
    long int r1 = 0;
    c1.activar();
    do {
        placeDefensesSinOrdenacion(freeCells, nCellsWidth, nCellsHeight,
            mapWidth, mapHeight, obstacles, defenses);
        ++r1;
    } while(c1.tiempo() < 1.0);
    c1.parar();

    cronometro c2;
    long int r2 = 0;
    c2.activar();
    do {
        placeDefensesFusion(freeCells, nCellsWidth, nCellsHeight, mapWidth,
            mapHeight, obstacles, defenses);
        ++r2;
    } while(c2.tiempo() < 1.0);
    c2.parar();

    cronometro c3;
    long int r3 = 0;
    c3.activar();
    do {
        placeDefensesRapido(freeCells, nCellsWidth, nCellsHeight, mapWidth,
            mapHeight, obstacles, defenses);
```

```
        ++r3;
    } while(c3.tiempo() < 1.0);
    c3.parar();

    cronometro c4;
    long int r4 = 0;
    c4.activar();
    do {
        placeDefensesMonticulo(freeCells, nCellsWidth, nCellsHeight,
            mapWidth, mapHeight, obstacles, defenses);
        ++r4;
    } while(c4.tiempo() < 1.0);
    c4.parar();


    std::cout << (nCellsWidth * nCellsHeight) << '\t' << c1.tiempo() / r1 <<
    '\t' << c2.tiempo() / r2 << '\t' << c3.tiempo() / r3 << '\t' <<
    c4.tiempo() / r4 << std::endl;
}
```

Para la realización de la gráfica usamos el código que hay en el `makefile` de la práctica y en el archivo `graphic.plot`:

```
# Codificacion ISO Latin-1 y terminal EPS.

set encoding iso_8859_1
set terminal postscript eps

# Titulo de cada eje.

set xlabel "n (numero de celdas)"
set ylabel "t(n) (tiempo en segundos)"

# Estilo de presentacion (puntos interpolados linealmente).

#set data style linespoints

# Creacion de los ficheros EPS.

set output "graphic.eps"
plot 'data.txt' using 1:2 title "Sin pre-ordenacion" with lines, \
     '' using 1:3 title "Fusion" with lines, \
         '' using 1:4 title "Rapida" with lines, \
         '' using 1:5 title "Monticulo" with lines
```

El resultado obtenido de las mediciones es el siguiente:



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.