Práctica 3. Divide y vencerás

Francisco Jesus Diaz Zajara
francisco.diazza@alum.uca.es
Teléfono: 673540730
NIF: 44062267V

4 de enero de 2018

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

   Para la representación del terreno de batalla he utilizado una matriz de celdas para el caso del algoritmo sin preordenacion y un vector para el resto de casos (ordenación por fusión, ordenación rápida y montículo).

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

```
void fusion(std::vector<Celda>& v, int i, int k, int j){

        int n = j - i + 1;
        int p = i;
        int q = k + 1;

        std::vector<Celda> w;

        for(int l=0; l<n; l++){

                if(p<=k && (q>j || v[p].valor > v[q].valor)){
                        w.push_back(v[p]);
                        p = p+1;
                }
                else{
                        w.push_back(v[q]);
                        q = q+1;
                }
        }

        for(int l=0; l<n; l++)
                v[i + l] = w[l];

}

void ordenacionFusion(std::vector<Celda>& v, int i, int j){

        int n = j - i + 1;

        if(n <= 2){
                if(v[i].valor < v[j].valor){
                        Celda aux = v[j];
                        v[j] = v[i];
                        v[i] = aux;
                }
        }
        else{
                int k = i - 1 + n/2;
                ordenacionFusion(v, i, k);
                ordenacionFusion(v, k+1, j);
                fusion(v, i, k, j);
        }
}
```

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

```
int pivote(std::vector<Celda>& v, int i, int j){

        int p = i;
        int x = v[i].valor;

        for(int k=i+1; k<=j; k++){

                if(v[k].valor > x){
                        p = p+1;
                        Celda aux = v[k];
                        v[k] = v[p];
                        v[p] = aux;
                }
        }

        Celda aux_2 = v[i];
        v[i] = v[p];

        v[p].pos.x = aux_2.pos.x;
        v[p].pos.y = aux_2.pos.y;
        v[p].valor = x;

        return p;
}

void ordenacionRapida(std::vector<Celda>& v, int i, int j){

        int n = j - i + 1;

        if(n <= 2){
                if(v[i].valor < v[j].valor){
                        Celda aux = v[j];
                        v[j] = v[i];
                        v[i] = aux;
                }
        }
        else{
                int p = pivote(v, i, j);
                ordenacionRapida(v, i, p);
                ordenacionRapida(v, 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.

```
bool estaOrdenado(std::vector<Celda> v){

    bool ordenado = true;
    int comprobado = 0;

    for(int i=0; i<v.size() && (comprobado==0); i++){
        if(v[i].valor < v[i+1].valor){
                comprobado++;
            ordenado = false;
        }
        }

    return ordenado;
}

void cajaNegra(std::vector<Celda> v){

        ordenacionFusion(v, 0, v.size()-1);
        if(estaOrdenado)
        std::cout << "EL algoritmo por fusion funciona correctamente." << std::endl ;
    else
```

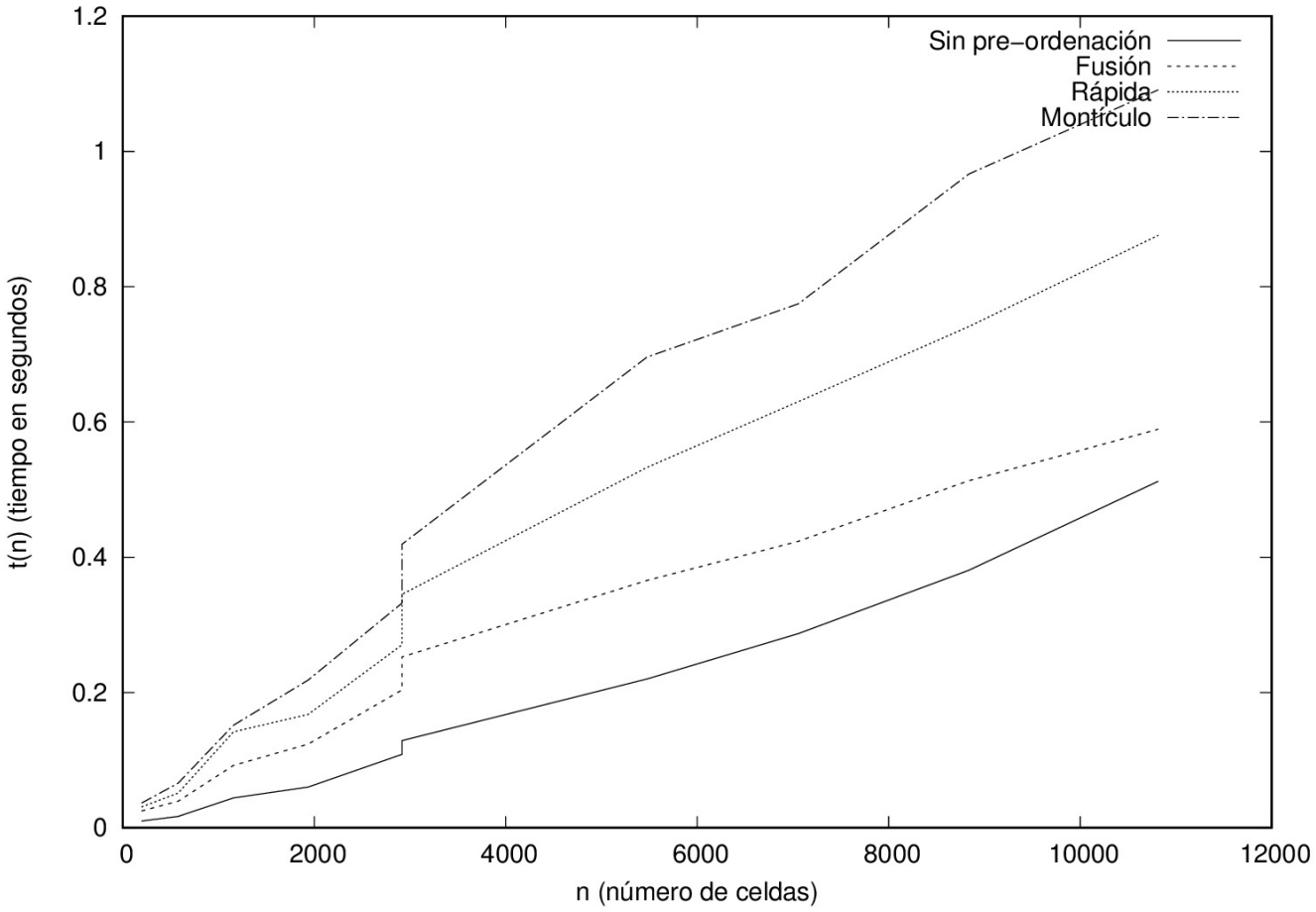
```
        std::cout << "Error, fusion no ordena correctamente." << std::endl;

        ordenacionRapida(v, 0, v.size()-1);
        if(estaOrdenado)
        std::cout << "El algoritmo por ordenacion rapida funciona correctamente." << std::
            endl ;
    else
        std::cout << "Error, ordenacion rapida no ordena correctamente." << std::endl;
}
```

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.

   Como veremos en la gráfica más adelante, el algoritmo sin preordenacion es el más eficiente en todos los casos para la representación del terreno de batalla escogida por mi, seguido del algoritmo de ordenación por fusión, ordenación rápida y montículo en orden de menor a mayor eficiencia.

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.




```
    long int r1 = 0;

    c.activar();
    do {
                List<Celda>::iterator currentCell = listaCeldas.begin();
```

```
            List<Defense*>::iterator currentDefense = defenses.begin();

            while(currentDefense != defenses.end() && currentCell != listaCeldas.end()) {

                    sinPreOrdenacion(matrizCeldas, nCellsWidth, (*currentCell));

                    if(esFactible(obstacles, defensasColocadas, mapWidth, mapHeight, (*
                        currentDefense), (*currentCell))){

                            (*currentDefense)->position.x = (*currentCell).pos.x;
                            (*currentDefense)->position.y = (*currentCell).pos.y;
                            (*currentDefense)->position.z = 0;
                        defensasColocadas.push_back((*currentDefense));
                            currentDefense++;
                    }
                    currentCell++;
            }
            ++r1;

            for(int i=0; i<nCellsWidth; i++)
                    for(int j=0; j<nCellsWidth; j++){
                            matrizCeldas[i][j].valor = (int)defaultCellValue(i, j,
                                freeCells, nCellsWidth, nCellsHeight
                            , mapWidth, mapHeight, obstacles, defenses);    //Damos
                                valores de nuevo a nuestra matriz de celdas.
                    }

} while(c.tiempo() < e_abs/e_rel + e_abs);
c.parar();

double tiempo_1 = c.tiempo();

long int r2 = 0;

c.activar();
do {
        std::vector<Celda> vectorCeldas;

    for(int i=0; i<nCellsWidth; i++)
            for(int j=0; j<nCellsWidth; j++)
                    vectorCeldas.push_back(matrizCeldas[i][j]);

            ordenacionFusion(vectorCeldas, 0, vectorCeldas.size()-1);

            List<Defense*>::iterator currentDefense = defenses.begin();

            while(currentDefense != defenses.end()) {

                    Celda currentCell = seleccion(vectorCeldas);

                    if(esFactible(obstacles, defensasColocadas, mapWidth, mapHeight, (*
                        currentDefense), currentCell)){
                            (*currentDefense)->position.x = currentCell.pos.x;
                            (*currentDefense)->position.y = currentCell.pos.y;
                            (*currentDefense)->position.z = 0;
                        defensasColocadas.push_back((*currentDefense));
                            currentDefense++;
                    }
            }
            ++r2;
} while(c.tiempo() < e_abs/e_rel + e_abs);
c.parar();

double tiempo_2 = c.tiempo();

long int r3 = 0;

c.activar();
do {
    std::vector<Celda> vectorCeldas;
```

```
                for(int i=0; i<nCellsWidth; i++)
                        for(int j=0; j<nCellsWidth; j++)
                        vectorCeldas.push_back(matrizCeldas[i][j]);

        ordenacionRapida(vectorCeldas, 0, vectorCeldas.size() - 1);

                List<Defense*>::iterator currentDefense = defenses.begin();

                while(currentDefense != defenses.end()) {

                        Celda currentCell = seleccion(vectorCeldas);

                        if(esFactible(obstacles, defensasColocadas, mapWidth, mapHeight, (*
                            currentDefense), currentCell)){
                                (*currentDefense)->position.x = currentCell.pos.x;
                                (*currentDefense)->position.y = currentCell.pos.y;
                                (*currentDefense)->position.z = 0;
                            defensasColocadas.push_back((*currentDefense));
                                currentDefense++;
                        }
                }
                ++r3;
    } while(c.tiempo() < e_abs/e_rel + e_abs);
    c.parar();

    float tiempo_3 = c.tiempo();

    long int r4 = 0;

    c.activar();
    do {
                std::vector<Celda> vectorCeldas;

                for(int i=0; i<nCellsWidth; i++)
                        for(int j=0; j<nCellsWidth; j++)
                                vectorCeldas.push_back(matrizCeldas[i][j]);

                monticulo(vectorCeldas);

                List<Defense*>::iterator currentDefense = defenses.begin();

                while(currentDefense != defenses.end()) {

                        Celda currentCell = seleccion(vectorCeldas);

                        if(esFactible(obstacles, defensasColocadas, mapWidth, mapHeight, (*
                            currentDefense), currentCell)){
                                (*currentDefense)->position.x = currentCell.pos.x;
                                (*currentDefense)->position.y = currentCell.pos.y;
                                (*currentDefense)->position.z = 0;
                            defensasColocadas.push_back((*currentDefense));
                                currentDefense++;
                        }
                }
                ++r4;
    } while(c.tiempo() < e_abs/e_rel + e_abs);
    c.parar();

    float tiempo_4 = c.tiempo();

    std::cout << (nCellsWidth * nCellsHeight) << '\t' << tiempo_1/r1 << '\t' << tiempo_2/r2
        << '\t' << tiempo_3/r3 << '\t' << tiempo_4/r4 << std::endl;
```

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.