Práctica 3. Divide y vencerás

Miguel Angel Chaves Perez
miguelangel.chavesperez@alum.uca.es
Teléfono: 675287145
NIF: 49071750p

January 7, 2017

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

   Para el primer caso en donde no hay ordenamiento, que es la función conMatriz, hemos utilizado una matriz para representar el terreno de batalla en tanta filas y columnas como tiene la misma y el valor calculado para colocar el centro de extracción. Mientras que con los demás casos (con preordenación) utilizamos un vector de estructuras de tipo celda, esta estructura tipo celda se compone de:

   struct celda int x; int y; float punt; struct celda operator =(const struct celda c2) x = c2.x; y = c2.y; punt = c2.punt; return *this;

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

```
green!40!blackvoid blackinsercion(blacksCelda* blackcells , green!40!blackint blacki, green!40!blackint
     blackj)
{
  green!40!blackfor(green!40!blackint blackk = blacki + 1; blackk <= blackj; blackk++)
  {
    blacksCelda blacktmp = blackcells[blackk];
    green!40!blackint blackl = blackk;
    green!40!blackwhile((blackl > blacki) && (blackcells[blackl] < blackcells[blackl - 1]))
    {
      blackcells[blackl] = blackcells[blackl - 1];
      blackl--;
    }
    blackcells[blackl] = blacktmp;
  }
}

green!40!blackvoid blackfusion(blacksCelda* blackv, green!40!blackint blacki, green!40!blackint blackk,
    green!40!blackint blackj)
{
  green!40!blackint blackn = blackj - blacki + 1;
  green!40!blackint blackp = blacki;
  green!40!blackint blackq = blackk + 1;

  blacksCelda blackw[blackn];

  green!40!blackfor(green!40!blackint blackl = 0; blackl <= blackn; blackl++)
  {
    green!40!blackif(blackp <= blackk && (blackq > blackj || blackv[blackp].blackpunt <= blackv[blackq
        ].blackpunt))
    {
      blackw[blackl] = blackv[blackp];
      blackp++;
    }
    green!40!blackelse
    {
      blackw[blackl] = blackv[blackq];
      blackq++;
    }
  }
```

```
  green!40!blackfor(green!40!blackint blackl = 0; blackl < blackn; blackl++)
  {
    blackv[blacki + blackl] = blackw[blackl];
  }
}
green!40!blackvoid blackordfusion(blacksCelda* blackv, green!40!blackint blacki , green!40!blackint
    blackj)
{
  green!40!blackint blackn = blackj - blacki + 1;
  green!40!blackif(blackn <= 2)
  {
    blackinsercion(blackv,blacki,blackj);
  }
  green!40!blackelse
  {
    green!40!blackint blackk = blacki - 1 + blackn/2;
    blackordfusion(blackv,blacki,blackk);
    blackordfusion(blackv,blackk+1,blackj);
    blackfusion(blackv,blacki,blackk,blackj);

  }

}
```

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

```
green!40!blackvoid blackinsercion(blacksCelda* blackcells , green!40!blackint blacki, green!40!blackint
     blackj)
{
  green!40!blackfor(green!40!blackint blackk = blacki + 1; blackk <= blackj; blackk++)
  {
    blacksCelda blacktmp = blackcells[blackk];
    green!40!blackint blackl = blackk;
    green!40!blackwhile((blackl > blacki) && (blackcells[blackl] < blackcells[blackl - 1]))
    {
      blackcells[blackl] = blackcells[blackl - 1];
      blackl--;
    }
    blackcells[blackl] = blacktmp;
  }
}

green!40!blackint blackpivote(blacksCelda* blackv, green!40!blackint blacki , green!40!blackint blackj)
{
  green!40!blackint blackp = blacki;
  blacksCelda blacktmp = blackv[blacki];
  blacksCelda blackaux;
  green!40!blackfor(green!40!blackint blackk = blacki + 1; blackk <= blackj; blackk++)
  {
    green!40!blackif(blackv[blackk].blackpunt <= blacktmp.blackpunt)
    {
      blackp++;
      blackaux = blackv[blackp];
      blackv[blackp] = blackv[blackk];
      blackv[blackk] = blackaux;
    }
  }
  blackv[blacki] = blackv[blackp];
  blackv[blackp] = blacktmp;
  green!40!blackreturn blackp;
}

green!40!blackvoid blackordrapida(blacksCelda* blackv, green!40!blackint blacki, green!40!blackint
    blackj)
{
  green!40!blackint blackn = blackj - blacki + 1;
  green!40!blackif(blackn <= 2)
  {
```

```
    blackinsercion(blackv,blacki,blackj);
  }
  green!40!blackelse
  {
    green!40!blackint blackp = blackpivote(blackv,blacki,blackj);
    blackordrapida(blackv,blacki,blackp-1);
    blackordrapida(blackv,blackp+1,blackj);
  }
}
```

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.

```
green!40!blackvoid blackDEF_LIB_EXPORTED blackplaceDefenses3(green!40!blackbool** blackfreeCells,
    green!40!blackint blacknCellsWidth, green!40!blackint blacknCellsHeight, green!40!blackfloat
    blackmapWidth, green!40!blackfloat blackmapHeight
               , blackList<blackObject*> blackobstacles, blackList<blackDefense*> blackdefenses)
{

  green!40!blackfloat blackcellWidth = blackmapWidth / blacknCellsWidth;
  green!40!blackfloat blackcellHeight = blackmapHeight / blacknCellsHeight;


green!40!black#green!40!blackifdef PRUEBAS_CAJA_NEGRA
    blacksCelda blackceldas[10];
    blacksCelda blackw[10];
    green!40!blackfor(green!40!blackint blacki=0;blacki<10;++blacki) {
        blackceldas[blacki].blackpunt = blacki;
        blackceldas[blacki].blackx = 0;
        blackceldas[blacki].blacky = 0;
    }

    green!40!blackwhile ( blackstd::blacknext_permutation(blackceldas, blackceldas+10)) {

      green!40!blackfor(green!40!blackint blacki = 0; blacki < 10; blacki++)
      {
        blackw[blacki] = blackceldas[blacki];
      }

        blackordfusion(blackw,0, 10-1);
        green!40!blackif(!blackordenado(blackw))
            blackstd::blackcout << orange"orangeErrororange" << blackstd::blackendl ;
    }

    green!40!blackfor(green!40!blackint blacki=0 ; blacki < 10 ; ++blacki) {
        blackceldas[blacki].blackpunt = blacki;
        blackceldas[blacki].blackx = 0;
        blackceldas[blacki].blacky = 0;
    }

    green!40!blackwhile ( blackstd::blacknext_permutation(blackceldas, blackceldas+10)) {

        green!40!blackfor(green!40!blackint blacki = 0; blacki < 10; blacki++)
      {
        blackw[blacki] = blackceldas[blacki];
      }

        blackordrapida(blackw,0,10-1);
        green!40!blackif(!blackordenado(blackw))
            blackstd::blackcout << orange"orangeErrororange" << blackstd::blackendl ;
    }
}

green!40!blackbool blackordenado(blacksCelda blackw[])
{
  green!40!blackbool blackcheck = green!40!blacktrue;

  green!40!blackfor(green!40!blackint blacki = 0; (blacki < 10 - 1) && (blackcheck ==
      green!40!blacktrue); blacki++)
  {
```

```
    green!40!blackif(blackw[blacki].blackpunt > blackw[blacki+1].blackpunt) {blackcheck =
        green!40!blackfalse;}
  }
  green!40!blackreturn blackcheck;
}
```

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.

   Lo esperado es que todos los metodos de ordenaciones deberian ser más eficientes que utilizar la matriz sola, ya que, debemos reccorer la matriz entera cada vez que queramos encontrar un candidato nuevo. Si utilizamos los algoritmos de ordenación sabemos que se perdera un tiempo en ordenar,pero no tendremos que recorrer todos los elementos para encontrar el mejor cuando seleccionamos, bastará con cogerlos secuencialmente.

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). Considere en su análisis los planetas con códigos 1500, 2500, 3500,..., 10500. Incluya en el análisis los planetas que considere oportunos para mostrar información relevante.

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.

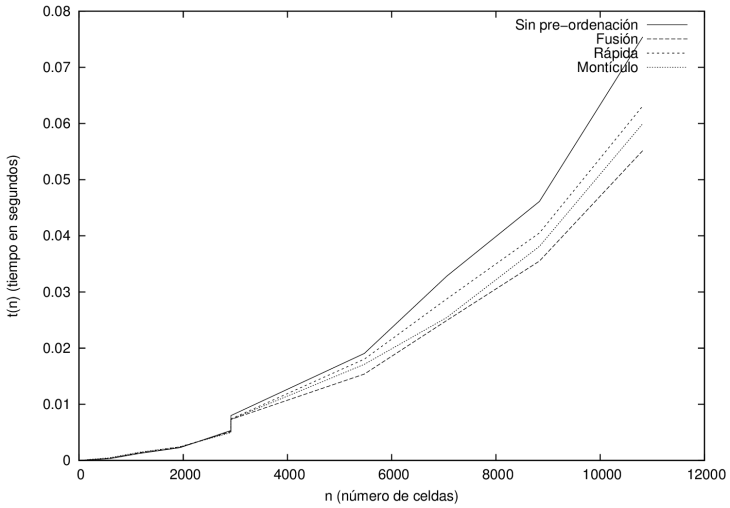

Figure 1: Gráfica