Grado en Ingeniería Informática

Estructura de Datos y Algoritmos, Grupo 80M, 2015/2016

14 de Marzo de 2016

Nombre y Apellidos:

….....

PROBLEMA 1 (1 punto)

Se quiere implementar un método que, sobre un TAD lineal que almacene caracteres (clase Character en JAVA), elimine todas las repeticiones consecutivas que encuentre. Algunos ejemplos del efecto que dicho método debe tener son los siguientes:

- P-E-R-R-O debe producir P-E-R-O
- P-E-R-r-O debe producir P-E-R-r-O (sensible a mayúsculas y minúsculas)
- P-A-P-P-A-A-D-A debe producir P-A-P-A-D-A

Se pide:

a) Elegir un tipo de TAD lineal sobre el que implementar la solución, justificando la elección en términos de eficiencia, simplicidad, etc.

b) Implementar SÓLO el método pedido asumiendo que se dispone de las clases que definen el nodo y el interfaz del TAD elegido.

c) Razona sobre el caso peor y mejor del algoritmo. ¿Cuál es la complejidad (en orden, no la función de tiempo) del método en el caso peor?

En la solución no puedes usar arrays.

Solución:

a) Lo más sencillo es usar una lista simplemente enlazada, porque aunque necesitemos más referencias para borrar un nodo, las operaciones al hacerlo son menos numerosas. La memoria usada es un poco menor. El número de operaciones hasta llegar al nodo de interés es algo mayor que en una lista doble, aunque una vez alcanzado dicho nodo, son algo menores para borrarlo. El caso mejor no cambia (lista vacía), y en cuanto al peor, habrá que recorrer todos los nodos del TAD, da igual si están repetidos o no. Se propone usar una SList.

Hay justificaciones válidas para la DList, como no necesitar tantas referencias para recorrer la lista.

b)

```
public void removeRepes() {
  SNode aux = firstNode;                          //1
  if(firstNode==null) return;                     //2
  for (SNode nodeIt = firstNode.next; nodeIt != null; nodeIt = nodeIt.next) {
                                                  //2+2n
       Character charPrev=aux.elem;               //n
       Character charAct=nodeIt.elem;             //n
       if (charAct.equals(charPrev)) {            //n
             aux.next = nodeIt.next;              //n-k
       }
       else aux = nodeIt;                         //k
  }//Esa k significa las k veces que se no se ha cumplido la condición del if.
 }  //Como ambas partes (if y else) tienen una línea, nos da igual cuál se haya
cumplido para el caso peor. La complejidad no cambia, pues la suma de ambas sigue
siendo n
```

c) El caso mejor supone que la lista esté vacía, y obtenemos entonces un orden contante O(1). En el caso peor habrá que recorrer la lista hasta el final y borrar todos los nodos menos uno (si todos eran iguales). La complejidad del método será lineal (O(n)).

PROBLEMA 2 (1 punto)

Desarrolla un **método RECURSIVO** que, pudiendo acceder a una matriz cuadrada de NxN casillas de booleanos a *false*, coloque N reinas (marcándolas como *true*) de forma que no se puedan comer y modifique la matriz con dichas N casillas a *true*. Declara antes del método cualesquiera variables que necesites (la del tablero la damos por declarada e inicializada al valor N). Asume que puedes usar un método llamado `public boolean conflict ()` que devuelve *true* si hay conflicto de reinas (y *false* si no lo hay) tal y como en ese momento se hallen colocadas en el tablero (por tanto, no necesita parámetros).

Nota: Las soluciones iterativas (basadas en bucles) no serán evaluadas.

Solución:

```
boolean solved = false;

private void solve(int column) {
            iter++;
            if (column == size)
                  solved = true;
            else {
                  int i;
                  for (i = 0; i < size; i++) {
                        if (!solved) {
                              if(i>0)
                                    table[i-1][column] = false;
                              table[i][column] = true;
                              if (conflict()){
                                    table[i][column] = false;
                              }
                              else{
                                    solve(column + 1);
                              }
                        }
```

```
                }
            }
        }
```