Respuestas del examen. 6 de febrero de 2.007

1. (2 puntos)

NOMBRE TablaCerrada[C, V]
CONJUNTOS
T – conjunto de tablas de dispersión cerrada
C – conjunto de claves de la tabla
V – conjunto de valores almacenables en la tabla
B – conjunto de los valores booleanos
N – conjunto de números naturales
M – conjunto de mensajes de error {"Error: el elemento no está en la tabla"}
SINTAXIS
crear: N → T
añadir: T x C x V → T
esVacía: T → B
está: T x C → B
recuperar: T x C → V ∪ M
borrar: T x C → T
tamaño: T → N
numCubetas: T → N /* Esta operación la añadimos nosotros */
estáLlena: T → B
SEMÁNTICA, ∀ t ∈ T, ∀ c, c1, c2 ∈ C, ∀ v ∈ V, ∀ n ∈ N
esVacía(crear(n)) = true
esVacía(añadir(t, c, v)) = false
está(crear(n), c) = false
está(añadir(t, c1, v), c2) = esIgual(c1, c2) OR está(t, c2)
recuperar(crear(n), c) = "Error: el elemento no está en la tabla"
recuperar(añadir(t, c1, v), c2) = si esIgual(c1, c2) → v | recuperar(t, c2)
borrar(crear(n), c) = crear(n)
borrar(añadir(t, c1, v), c2) = si esIgual(c1, c2) → borrar(t, c2) | añadir(borrar(t, c2), c1, v)
tamaño(crear(n)) = cero
tamaño(añadir(t, c, v)) = suma(tamaño(t), si está(t, c) → cero | sucesor(cero))
numCubetas(crear(n)) = n
numCubetas(añadir(t, c, v)) = numCubetas(t)
estáLlena(t) = esIgual(numCubetas(t), tamaño(t))

Si no se hace la suposición de que la tabla no se puede llenar, la especificación se hace más complicada, porque al aplicar la operación *añadir* puede que el elemento no se coloque realmente en la tabla. En concreto, si se aplica *añadir* sobre una tabla llena, entonces si la clave ya se encontraba se modifica el valor, y en otro caso no se inserta la asociación. En consecuencia, la operación *añadir* debería dejar de ser un constructor del TAD.

2. (2,6 puntos)

Suponemos que existe una operación primeros(p, i), que dada una cadena p, devuelve la subcadena de los i primeros caracteres de p. La operación se puede implementar usando recursividad. En cada instante, debemos garantizar que el nodo actual por el que nos movemos en el trie está entre **p1** y **p2**. La implementación podría ser como la siguiente; la llamada inicial sería: **Listar(raizTrie, p1, p2, 1, “”)**.

```
operación Listar (t: trie; p1, p2: cadena; i: entero; act: cadena)
    para cada carácter c hijo del nodo t hacer
        act2:= act+c
        si (act2≥primeros(p1, i) AND act2≤p2) entonces
            si (c == $) AND (act2≥p1) entonces
                Mostrar act
            sino
                Listar (Consulta(t, c), p1, p2, i+1, act2)
            finsi
        finsi
    finpara
```

3. (3 puntos)

El ejercicio es una variación del problema de caminos mínimos, que se puede resolver con el algoritmo de Dijkstra con varias modificaciones: (1) queremos calcular los caminos mínimos entre varios orígenes y todos los nodos de destino, por lo que suponemos un origen ficticio con coste 0 para los nodos invadidos inicialmente; (2) en lugar de calcular los nodos de paso en el array **P**, calculamos el origen de cada camino mínimo en el array **A**; (3) en caso de empate para un nodo en el camino mínimo y con distinto origen (distinto valor de **A**) se marca el nodo como destruido; si el mínimo seleccionado en **v** (dentro del proceso del algoritmo de Dijkstra) está marcado, entonces no se aplica el paso de actualización del algoritmo. La implementación sería como la siguiente:

```
operación Galaxias (M: array [1..n, 1..n] de real; G: array [1..n] de entero)
var
    D: array [1..n] de real          // Distancia de los caminos mínimos a cualquier bando
    A: array [1..n] de entero        // Bando (+1: rebelde; -1: imperio; 0: neutral; 2: destruido)
    S: array [1..n] de booleano      // Nodos escogidos o no

    // 1. Inicialización. La distancia es 0 para todos los nodos que no sean neutrales
    para v:= 1, ..., n hacer
        S[v]:= FALSE
        A[v]:= G[v]
        D[v]:= (G[v] ≠ 0) ? 0 : +∞       // Suponemos el operador condicional de  C
    finpara

    // 2. Cuerpo del algoritmo. En este caso se repite n veces (no n-1)
    para i:= 1, ..., n hacer
        v:= nodo con S[v]==FALSE y mínimo D[v]
        S[v]:= TRUE
        si A[v] ≠ 2 entonces    // En otro caso, el nodo v está destruido definitivamente
            para cada nodo w adyacente a v hacer
                si (NOT S[w]) AND (D[v]+M[v,w] < D[w]) entonces
                    D[w]:= D[v] + M[v, w]
                    A[w]:= A[v]      // w es del mismo bando que v
                sino si (D[v]+M[v,w] == D[w]) AND (A[w] ≠ A[v]) entonces
                    A[w]:= 2         // w es destruido, a menos que haya otro
                finsi                // camino mínimo menor
            finpara
        finsi
    finpara
```

El array **A** indica al final del algoritmo el bando al que pertenece cada sistema planetario: +1 = rebelde; -1 = imperio; 0 = neutral; 2 = destruido. El array **D** indica el instante en que cada sistema se convierte a uno u otro bando.

4. (2,4 puntos)

(1) a, c (2) a, d (3) c (4) b (5) c (6) a, c