**MONTÍCULOS**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Función *CreaMontículoVacío*** | | La función devuelve un montículo vacío con el contador iniciado a 0 elementos. | | **fun** CreaMonticulo(m: montículo)  m.T ← *null*  m.c ← 0  *m.*Max← *n*  **ffun** |  |  | | --- | | **Función *Primero*** | | Esta función devuelve el valor que hay en la cima del montículo sin eliminarla. *O*(1) | | **fun** Primero(m: montículo): elemento  **si** m.c =0 **entonces dev error**  **sino**  dev m.T[1]  **fsi**  **ffun** |  |  | | --- | | **Función *ObtenerCima*** | |  | | **fun** ObtenerCima(m: montículo): elemento  **var**  e:elemento  **fvar**  **si** m.c *≠* 0 **entonces**  e ←m.T[1]  m.T[1] ←m.T[m.c]  m.c ←m.c - 1  Hundir(m.T,1)  dev e  **fsi**  **ffun** | | |  | | --- | | **Función *MontículoVacío?*** | |  | | **fun** MonticuloVacio?(m: montículo) : bool  **si** (m.c =0) **entonces dev** *cierto* **sino dev** *falso*  **ffun** |  |  | | --- | | **Función *Hundir*** | | Reubica el elemento *i-ésimo* del vector *T* del montículo *m* en caso de que este sea menor que alguno de sus hijos. | | **fun** Hundir(T:vector, i:natural)  **var**  hi,hd,p:natural  **fvar**  **repetir**  hi ← 2·i  hd ← 2·i+l  p ← i  **si** (hd ≤ m.c ) ∧ (T[hd] > T[i]) **entonces**  i ←hd  **fsi**  **si** (hi ≤ m.c) ∧ (T[hi] > T[i]) **entonces**  i ← hi  **fsi**  intercambiar (T[p],T[i])  **hasta** p=i;  **ffun** |  |  | | --- | | **Función *Borrado*** | | *O*(*log n*) | |  | | |  | | --- | | **Función *Flotar*** | | Reubica el elemento *i-esimo* del vector *T* del montículo en caso de que éste sea mayor que el padre, hasta que esté correctamente situado en el montículo y se haya restablecido la *propiedad de montículo.* | | **fun** Flotar(T: vector, i:natural)  *//* el nodo padre de i es i div 2  **mientras** (i>1) A (T[i div 2] < T[i]) **hacer**  intercambiar(T[i], T[i div 2])  i ← i div 2  **fmientras**  **ffun** |  |  | | --- | | **Función *CreaMontículo*** | | Crear un montículo a partir de los elementos de un vector mediante el procedimiento *Hundir:*. *O*(n) | | fun CreaMonticuloLineal(T:vector[ l ..n]): montículo  var  m:montículo  fvar  m ←CreaMonticuloVacio();  **para** i←*n/2* **hasta** l **paso** -l **hacer**  Hundir(T,i);  **fpara**  m.T ←T  m.c ← n  **dev**(m)  **ffun** | | |  | | --- | | **Función *Insertar*** | |  | | **fun** Insertar(e:elemento; m: montículo): montículo  **si** m.c = m.MAX **entonces**  **error**(MonticuloLleno)  **sino**  m.c ← m.c + 1  m.T[m.c] ← e  Flotar (m.T, m.c)  **fsi**  **ffun** |  |  | | --- | | **Función *Heapsort*** | | Ordenación basada en montículos *O*(*n log n*) | | **fun** Heapsort(T:**vector**[l..n] de entero): **vector**[l..n] de entero  **var**  e:entero  M: montículo  S: **vector**[l..n]  **fvar**  M ← CreaMonticuloLineai(T);  **para** i←1**hasta** l **paso** -l **hacer**  e ← ObtenerCima(M)  S[i] ←e  **fpara**  **dev** S  **ffun** | |

|  |
| --- |
| **17SR-**   * Cuál de las siguientes cuestiones es **cierta**:   1. La creación de un montículo a partir de un vector de valores, puede tener coste lineal.   2. Los montículos de máximos son equivalentes a un árbol binario de búsqueda, con los elementos menores que la raíz en el subárbol izquierdo y los mayores en el derecho.   3. Un algoritmo voraz siempre encuentra la solución óptima en tiempo lineal, es decir en un tiempo proporcional al tamaño del conjunto de candidatos.   4. Ninguna de las otras respuestas es correcta.  1. **Pág. 40** 2. El vector T[*1..n]* implementa un montículo cuando:    * + El nodo raíz es el elemento *T* [1] del vector y contiene el mayor (o menor) de los elementos del montículo.      + Los nodos hijos del elemento T[*i*] que son respectivamente T[2i] y *T[2i +1*] cumple que T[i] ≥ T[2i] y T[i] ≥ T[*2i + 1*] para el caso de los montículos de máximos Un algoritmo voraz siempre encuentra la solución óptima en tiempo lineal, es decir en un tiempo proporcional al tamaño del conjunto de candidatos. 3. – 4. Ninguna de las otras respuestas es correcta. |
| **A16SO-**   * Indique cuál de las siguientes afirmaciones es **falsa**:   1. ---   2. ---   3. La operación “mínimo2 en un montículo tiene un coste O(1)   4. El orden de complejidad de la operación de borrado de un elemento en el montículo es O(log n)  1. Verdadera 2. Verdadera 3. **-** 4. Verdadera |
| **15SR-**  En relación a los montículos, ¿cuál de las siguientes afirmaciones es **falsa**?   1. Por razones de eficiencia, los montículos se implementan como vectores 2. El montículo es un tipo particular de árbol balanceado y completo. 3. La ordenación mediante el algoritmo Heapsort tiene un coste *O*(log n) 4. El vector m=[6,5,4,4,1,3,2] es un montículo de máximos 5. Verdadera 6. Verdadera 7. **Falsa. O(n log n)** 8. Verdadera |
| **A18F1-**  Respecto a la estructura de datos montículos de mínimos, cuál de las siguientes afirmaciones es **falsa:**   1. Con un montículo de mínimos disponemos de una estructura de datos en la que encontrar el mínimo es una operación de coste constante. 2. El montículo sirve de apoyo a la creación de un algoritmo de ordenación eficiente conocido como Heapsort. 3. El montículo es una árbol binario que puede estar o no balanceado 4. Cuando se extrae el primer elemento de un montículo, restaurar la propiedad de montículo tiene coste *O*(log n) 5. Verdadera 6. Verdadera 7. **Falsa. Debe estar balanceado** 8. Verdadera |