lista

Ejercicio 1 Escribir una función `void creciente(queue<int> &Q)` que elimina elementos de `Q` de tal manera de que los elementos que quedan estén ordenados en forma creciente. [Tomado en el 1er parcial 27-APR-2004] Resuelto en el archivo `creciente.cpp`

Ejercicio 2 Escriba procedimientos para intercalar (*merge*): (i) dos listas ordenadas `L1` y `L2` en una nueva lista `L`; (ii) un vector `VL` de `n` listas ordenadas como nueva lista `L`. Notar que *intercalar* (*merge*) implica en ambos casos que la nueva lista `L` debe resultar también *ordenada*. Resuelto en el archivo `intercala.cpp`

Ejercicio 3 Escribir una función `void junta (list <int> &L, int n)` que, dada una lista `L`, agrupa de a `n` elementos dejando su suma IN PLACE. Por ejemplo, si la lista `L` contiene L=(1,3,2,4,5,2,2,3,5,7,4,3,2,2), entonces depués de `junta (L,3)` debe quedar L=(6,11,10,14,4). Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución `O (m)`, donde `m` es el número de elementos en la lista original. [Tomado en el examen final del 1/8/2002] Resuelto en el archivo `junta.cpp`

Ejercicio 4 Escribir una función `void ordenag (list <int> &l, int m)` que, dada una lista `l`, va ordenando sus elementos de a grupos de `m` elementos. Por ejemplo si m=5, entonces `ordenag` ordena los primeros 5 elementos entre si, despues los siguientes 5 elementos, y asi siguiendo. Si la longitud `n` de la lista no es un múltiplo de `m`, entonces los últimos `n mod m` elementos también deben ser ordenados entre si. Por ejemplo, si l = (10 1 15 7 2 19 15 16 11 15 9 13 3 7 6 12 1), entonces después de `ordenag (5)` debemos tener l = (1 2 7 10 15 11 15 15 16 19 3 6 7 9 13 1 12). [Tomado en el examen final del 5-Dic-2002]. Resuelto en el archivo `ordenag.cpp`

Ejercicio 5 Usando las operaciones del TAD lista, escribir una función `void particiona (list<int> &L, int a)` la cual, dada una lista de enteros `L`, reemplace aquellos que son mayores que `a` por una sucesión de elementos menores o iguales que `a` pero manteniendo la suma total constante. [Ejercicio tomado en el Exámen Final del 05/07/01] Resuelto en el archivo `particiona.cpp`

Ejercicio 6 Usando las operaciones del TAD lista, escribir una función `void random_shuffle (list <int> &L)` que, dada una lista de enteros `L`, reordena sus elementos en forma aleatoria. Se sugiere el siguiente algoritmo: usando una lista auxiliar `Q` se van generando números enteros desde 0 a `length (L) - 1`. Se extrae el elemento j-ésimo de `l` y se inserta en `Q`. Finalmente, se vuelven a pasar todos los elementos de la cola `Q` a la lista `L`. [Ejercicio tomado en el Exámen Final del 05/07/01] Resuelto en el archivo `random_shuffle.cpp`

arbol orientado

Ejercicio 1 Listado de árboles orientados en diferentes ordenes. Orden previo, posterior y simétrico. Resuelto en el archivo `listarbo.cpp`

Ejercicio 2 El listado en orden de nivel de los nodos de un árbol lista primero la raiz, luego todos los nodos de profundidad 1, después todos los de profundidad 2, y asi sucesivamente. Los nodos que estén en la misma profundidad se listan en orden de izquierda a derecha. Escribir una función `void orden_de_nivel (tree <int> &t)` para listar los nodos de un árbol en orden de nivel. Resuelto en el archivo `orden_nivel.cpp`