

Algoritmos y Estructuras de Datos. 1er Parcial. [2011-09-15]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 40 % en programación (Ej 2), y un 60 % sobre las preguntas de teoría (Ej 3).

[Ej. 1] [clases (30pt)]

- a) [lista (20pt)] Escribir la implementación en C++ del TAD lista (clase `list`) implementado por celdas (simple o doblemente) enlazadas por punteros ó cursores. (**Indicar claramente qué implementación elige.**) Los métodos a implementar son `insert(p,x)`, `erase(p)`, `iterator::operator++(int)` (postfijo), `iterator::operator++()` (prefijo).
- b) [pila-cola (10pt)] Escribir la implementación en C++ de los métodos `push`, `pop`, `front`, y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

[Ej. 2] [Programación (total = 50pt)] Recordar que en los ejercicios de programación **deben usar la interfaz STL.**

- a) [expand (20pt)] **Consigna:** Escribir una función `void expand(list<int> &L, int m)`; que transforma los elementos de una lista `L` de tal forma que todos los elementos de `L` resulten ser menores o igual que `m`, pero de tal forma que su suma se mantenga inalterada. Para esto, si un elemento `x` es mayor que `m` entonces, lo divide en tantas partes como haga falta para satisfacer la condición; por ejemplo si `m=3` podemos dividir a 10 en 3,3,3,1. Es decir si `L=(7,2,3,1,4,5)`, entonces después de hacer `expand(L,2)` debe quedar `L=(2,2,2,1,2,2,1,1,2,2,2,2,1)`.
- b) [rota (10pt)] Escribir una función `void rota(stack<int> &S, queue<int> &Q)`; que realiza las siguientes operaciones (sea `m` el número de elementos en la pila):
- 1) Pone todos los elementos de la pila `S` en la cola `Q`;
 - 2) Saca `m` elementos de `Q` y los pone en `S`.
- Así por ejemplo si `S=(1,3,2,4)` y `Q=(6,2,5)` entonces después de llamar `rota(S,Q)` debe quedar `S=(1,5,2,6)` y `Q=(3,2,4)`.
- c) [is-rotation (20pt)] Determinar si una correspondencia `map<int,int> M` es una “rotación” es decir, una permutación tal que cada elemento del conjunto de las claves es asignado al siguiente elemento, en cierto orden. Por ejemplo `M={1->3,2->8,3->5,4->2,5->4,8->1}` corresponde a una rotación en orden (1,3,5,4,2,8).
- Consigna:** escribir una función `bool is_rotation(map<int,int> &M, list<int> &order)`; que determine si `M` es una rotación y en ese caso devuelve en `L` la lista de los elementos de `M` en el orden dado por la correspondencia.
- Ayuda:** Generar la lista `L` aplicando sucesivamente $x_{i+1} = M(x_i)$ a partir de cualquier clave inicial x_0 . La correspondencia es una rotación si
- El valor del contradominio x_{i+1} siempre es una clave.
 - Los elementos de la secuencia son todos distintos, salvo para el último elemento, cuando $x_n = x_0$.
- (Nota: Utilizar una correspondencia auxiliar para saber cuales claves ya fueron visitadas.)

[Ej. 3] [Preguntas (total = 20pt, 4pt por pregunta)]

- a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones T_1, \dots, T_5 determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = 10 + 2 \log_3 n^{1.2} + 5^4 + 3.14159 \log_2 n$$

$$T_2 = n^3 + 2 \cdot 2^n + 3^5$$

$$T_3 = 2 \cdot 5^n + \sqrt{5} \cdot n + \log_3 n$$

$$T_4 = 2 \cdot 3^n + 4n^5 + 6n!$$

$$T_5 = 5 \log_2 n + \sqrt{n} + 6n^4 + 5n^2$$

- b) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase `lista<>` implementada con **celdas simplemente enlazadas** (por punteros o cursores) en el caso promedio?

Métodos: `insert(p,x)`, `*p`, `erase(p)`, `clear()`, `begin()`, `end()`.

- c) ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si en realidad es siempre verdad que $(n+1)^2 > n^2$?

- d) ¿Qué ocurre si ejecutamos el siguiente código? `S` es una pila de enteros que puede estar vacía o no.

```
S.push(3); S.push(5); S.top(); S.top(); x=S.top();
```

¿Puede dar un error porque `S` se queda vacía? ¿Que valor toma `x`?

- e) Discuta ventajas y desventajas de usar contenedores lineales **ordenados** o **desordenados** para representar correspondencias.

[Ej. 4] [lab (bool)] Dado un `map<int, list<bool> > M`, verificar que para todas las claves pares, la lista correspondiente tenga todos sus elementos en `true` o bien que sea vacía. Si el map no tiene elementos, la salida debe ser `true`.

Ejemplos:

```
M = {2 -> [true, true], 4 -> []} -> returns true
M = {1 -> [false]} -> returns true
M = {2 -> [true, true, true], 3 -> [true, false]} -> returns true
M = {} -> returns true
M = {6 -> [false, true], 4 -> [true, true]} -> returns false
```

La función debe tener la siguiente signatura `bool mostrar_map(map<int, list<bool> >&);`

Instrucciones para la autoverificación: Junto se provee un archivo llamado `evaluar.hpp` que contiene una clase llamada `Evaluar` dentro de un namespace llamado `aed`. Esta clase la deben utilizar para comprobar si lo que hicieron es correcto. Lo único que tienen que hacer es incluir el archivo en su programa, y en el main crear un objeto de esta clase. Como parámetro al constructor deben pasarle un puntero a la función que resuelve el ejercicio (simplemente escriben el nombre de la función). Por ejemplo: `aed::Evaluar obj(mostrar_map);` en el caso de que su función se llame `mostrar_map`. Si el programa les muestra **todo BIEN**, es porque está bien. Si en algún caso en particular hay algún error, se les indicará cual era la respuesta esperada (la correcta) y cual es la que ustedes obtuvieron.