Trabajo práctico 2: Diseño

Lollapatuza

Normativa

Límite de entrega: Viernes 2 de junio *hasta las 23:59 hs.* Ver "Instructivo para entregas" en el sitio Web de la materia.

Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia. (`http://campus.exactas.uba.ar`)

1. Enunciado

Este TP consiste en diseñar módulos para implementar el sistema del *Lollapatuza*, conforme a la especificación que estará disponible en el sitio de la materia a partir del Lunes 15 de Mayo.

Se deben proveer las siguientes operaciones, con las complejidades temporales en **peor caso** indicadas. Usamos las siguientes variables:

- A — cantidad de personas.
- I — cantidad total de ítems en todo el festival.
- P — cantidad de puestos.

1.1. Lollapatuza

1. Iniciar un nuevo sistema. No se pone restricción a la complejidad.
2. Registrar la compra de una cantidad de un ítem particular, realizada por una persona en un puesto. $O(log(A) + log(I) + log(P) + log(cant))$. Donde *cant* corresponde a la cantidad adquirida del ítem que compró la persona en esta transacción.
3. Hackear un ítem consumido por una persona. Se hackea el puesto de menor ID en el que la persona haya consumido ese ítem sin promoción. $O(log(A) + log(I))$. En caso de que el puesto correspondiente deje de ser hackeable para esa persona e ítem luego de esta operación, se permitirá una complejidad de $O(log(A) + log(I) + log(P))$.
4. Obtener el gasto total de una persona. $O(log(A))$.
5. Obtener la persona que más dinero gastó. Si hay más de una persona que gastó el monto máximo, desempata por ID de la persona. $O(1)$.
6. Obtener el ID del puesto de menor stock para un ítem dado. Si hay más de un puesto que tiene stock mínimo, dar el de menor ID. No se pone restricción a la complejidad.
7. Conocer la información general del sistema:
   - Obtener las personas. $O(1)$.
   - Obtener los puestos de comidas, con sus IDs. $O(1)$.
8. Otras operaciones que crean conveniente agregar al módulo. No se pone restricción a la complejidad.

1.2. Puesto de Comida

1. Inicializar un nuevo puesto. No se pone restricción a la complejidad.
2. Obtener el stock de un ítem. $O(log(I))$.
3. Obtener el descuento de un ítem dada la cantidad del mismo. $O(log(I) + log(cant))$. Donde *cant* es la cantidad del ítem.
4. Obtener el gasto realizado por una persona en el puesto. $O(log(A))$.
5. Otras operaciones que crean conveniente agregar al módulo. No se pone restricción a la complejidad.

2. Documentación a entregar

Se deben diseñar los módulo principales (`Lollapatuza` y `Puesto De Comidas`) y todos los módulos auxiliares. La única excepción son los módulos disponibles en el Apunte de Módulos Básicos, que se pueden utilizar sin diseñarlos: lista enlazada, pila, cola, vector, diccionario lineal, conjunto lineal y conjunto acotado de naturales, además del diccionario logarítmico cuya interfaz se encuentra en el anexo de este mismo enunciado.

Todos los módulos diseñados deben contar con las siguientes partes.

1. **Interfaz.**
   - 1.1. *Tipo abstracto ("se explica con ...").* Género (TAD) que sirve para explicar las instancias del módulo, escrito en el lenguaje de especificación formal de la materia. Pueden utilizar la especificación que se incluye en el apéndice.
   - 1.2. *Signatura.* Listado de todas las funciones públicas que provee el módulo.
   - 1.3. *Contrato.* Precondición y postcondición de todas las funciones públicas. Las precondiciones de las funciones de la interfaz deben estar expresadas formalmente en lógica de primer orden.[1]
   - 1.4. *Complejidades.* Complejidades de todas las funciones públicas, cuando corresponda.
   - 1.5. *Aspectos de* aliasing. De ser necesario, aclarar cuáles son los parámetros y resultados de los métodos que se pasan por copia y cuáles por referencia, y si hay *aliasing* entre algunas de las estructuras.
2. **Implementación.**
   - 2.1. *Representación ("se representa con ...").* Módulo con el que se representan las instancias del módulo actual.
   - 2.2. *Invariante de representación.* Debe estar expresado formalmente en lógica de primer orden.
   - 2.3. *Función de abstracción.* Debe estar expresada formalmente en lógica de primer orden.
   - 2.4. *Algoritmos.* Deben estar expresados en pseudocódigo, usando la notación del lenguaje de módulos de la materia o notación tipo C++. Las pre y postcondiciones de las funciones auxiliares pueden estar expresadas en lenguaje natural (no es necesario que sean formales). Indicar de qué manera los algoritmos cumplen con el contrato declarado en la interfaz y con las complejidades pedidas. No se espera una demostración formal, pero sí una justificación adecuada.
3. **Servicios usados.** Módulos que se utilizan, detallando las complejidades, *aliasing* y otros aspectos que dichos módulos deben proveer para que el módulo actual pueda cumplir con su interfaz.

Sobre el uso de lenguaje natural y formal.

Las precondiciones y poscondiciones de las funciones auxiliares, pueden estar expresados en lenguaje natural. Se recomienda aplicar el sentido común para priorizar la **claridad** y **legibilidad**. Por ejemplo:

Más claro		**Menos claro**
"Cada clave del diccionario D debe ser una lista sin elementos repetidos."	✓	"No puede haber repetidos." (¿En qué estructura?).
"sinRepetidos?(claves(D))"	✓	
"Ordenar la lista A usando mergesort."	✓	"Ordenar los elementos." (¿Qué elementos? ¿Cómo se ordenan?).
"A.mergesort()"	✓	

[1] Si la implementación requiere usar funciones auxiliares, sus pre y postcondiciones pueden estar escritas en lenguaje natural, pero esto no forma parte de la interfaz.

Anexo - Interfaz Diccionario Logarítmico

El módulo Diccionario Logarítmico provee un diccionario en el que se puede definir, borrar, y testear si una clave está definida en tiempo logarítmico, aprovechando el uso de árboles balanceados (como el AVL). Notar que es similar al Diccionario Lineal, pero con mejores complejidades.

En cuanto al recorrido de los elementos, se provee un iterador bidireccional que permite recorrer y eliminar los elementos de d como si fuera una secuencia de pares κ, σ.

Para describir la complejidad de las operaciones, vamos a llamar $copy(k)$ al costo de copiar el elemento $k \in \kappa \cup \sigma$ y $equal(k_1, k_2)$ al costo de evaluar si dos elementos $k_1, k_2 \in \kappa$ son iguales (i.e., *copy* y *equal* son funciones de $\kappa \cup \sigma$ y $\kappa \times \kappa$ en $\mathbb{N}$, respectivamente).[2]

Interfaz

parámetros formales
géneros κ, σ
función $\bullet = \bullet$(**in** k_1: κ, **in** k_2: κ) $\to res$: `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} (k_1 = k_2)\}$
Complejidad: $\Theta(equal(k_1, k_2))$
Descripción: función de igualdad de κ's

función COPIAR(**in** k: κ) $\to res$: κ
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} k\}$
Complejidad: $\Theta(copy(k))$
Descripción: función de copia de κ's

función COPIAR(**in** s: σ) $\to res$: σ
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} s\}$
Complejidad: $\Theta(copy(s))$
Descripción: función de copia de σ's

se explica con: DICCIONARIO(κ, σ), ITERADOR BIDIRECCIONAL(TUPLA(κ, σ)).

géneros: `dicc`(κ, σ), `itDicc`(κ, σ).

Operaciones básicas de diccionario

VACÍO() $\to res$: `dicc`(κ, σ)
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \text{vacio}\}$
Complejidad: $\Theta(1)$
Descripción: genera un diccionario vacío.

DEFINIR(**in/out** d: `dicc`(κ, σ), **in** k: κ, **in** s: σ) $\to res$: `itDicc`(κ, σ)
Pre $\equiv$ $\{d =_{\text{obs}} d_0\}$
Post $\equiv$ $\{d =_{\text{obs}} \text{definir}(d, k, s) \land \text{haySiguiente}(res) \land_L \text{Siguiente}(res) = \langle k, s \rangle \land \text{alias}(\text{esPermutación}(\text{SecuSuby}(res), d))\}$
Complejidad: $\Theta\left(log(\sum_{k' \in K} equal(k, k')) + copy(k) + copy(s)\right)$, donde $K = \text{claves}(d)$
Descripción: define la clave k con el significado s en el diccionario. Retorna un iterador al elemento recién agregado.
Aliasing: los elementos k y s se definen por copia. El iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función ELIMINARSIGUIENTE. Además, anteriores(res) y siguientes(res) podrían cambiar completamente ante cualquier operación que modifique el d sin utilizar las funciones del iterador.

DEFINIDO?(**in** d: `dicc`(κ, σ), **in** k: κ) $\to res$: `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \text{def?}(d, k)\}$
Complejidad: $\Theta(log(\sum_{k' \in K} equal(k, k')))$, donde $K = \text{claves}(d)$
Descripción: devuelve `true` si y sólo k está definido en el diccionario.

SIGNIFICADO(**in** d: `dicc`(κ, σ), **in** k: κ) $\to res$: σ
Pre $\equiv$ {def?(d, k)}

[2] Nótese que este es un abuso de notación, ya que no estamos describiendo *copy* y *equal* en función del tamaño de k. A la hora de usarlo, habrá que realizar la traducción.

Post $\equiv$ {alias($res =_{\text{obs}}$ Significado(d, k))}
Complejidad: $\Theta(log(\sum_{k' \in K} equal(k, k')))$, donde K = claves(d)
Descripción: devuelve el significado de la clave k en d.
Aliasing: res es modificable si y sólo si d es modificable.

BORRAR(**in/out** d: `dicc(`κ, σ`)`, **in** k: κ)
Pre $\equiv$ {$d = d_0 \wedge$ def?(d, k)}
Post $\equiv$ {$d =_{\text{obs}}$ borrar(d_0, k)}
Complejidad: $\Theta(log(\sum_{k' \in K} equal(k, k')))$, donde K = claves(d)
Descripción: elimina la clave k y su significado de d.

#CLAVES(**in** d: `dicc(`κ, σ`)`) $\rightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ #claves(d)}
Complejidad: $\Theta(1)$
Descripción: devuelve la cantidad de claves del diccionario.

COPIAR(**in** d: `dicc(`κ, σ`)`) $\rightarrow$ res : `dicc(`κ, σ`)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} d$}
Complejidad: $\Theta\left(\sum_{k \in K} (copy(k) + copy(\text{significado}(k, d)))\right)$, donde K = claves(d)
Descripción: genera una copia nueva del diccionario.

$\bullet = \bullet$(**in** d_1: `dicc(`κ, σ`)`, **in** d_2: `dicc(`κ, σ`)`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} c_1 = c_2$}
Complejidad: $O\left(\sum_{\substack{k_1 \in K_1 \\ k_2 \in K_2}} equal(\langle k_1, s_1 \rangle, \langle k_2, s_2 \rangle)\right)$, donde K_i = claves(d_i) y s_i = significado(d_i, k_i), $i \in \{1, 2\}$.
Descripción: compara d_1 y d_2 por igualdad, cuando σ posee operación de igualdad.
Requiere: $\bullet = \bullet$(**in** $s_1 : \sigma$, **in** $s_2 : \sigma$) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} (s_1 = s_2)$}
Complejidad: $\Theta(equal(s_1, s_2))$
Descripción: función de igualdad de σ's

Operaciones del iterador

El Iterador de Diccionario Logarítmico cuenta con las mismas operaciones del Iterador de Diccionario Lineal. Para más detalles, revisar el Apunte de Módulos Básicos.