DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Guia 3

Ejercicio obligatorio de la práctica

22 de junio de 2020 — Algoritmos y Estructuras de Datos II

Integrante	LU	Correo electrónico
Rodriguez, Miguel	57/19	mmiguerodriguez@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
https://exactas.uba.ar

1. Módulo Cumpleaños

Interfaz

se explica con: CUMPLEAÑOS

géneros: `cumple`

Operaciones básicas

1. PUBLICARLISTA(**in**/**out** c: `cumple`, **in** n: `negocio`, **in** ℓ: `dicc(regalo, nat)`)
   Complejidad: $\mathcal{O}(L + log(N))$. L = #claves(ℓ) y N = #negocios(c).
2. REGALOS(**in** c: `cumple`, **in** n: `negocio`) $\rightarrow$ res : `conj(regalo)`
   Complejidad: $\mathcal{O}(log(N))$. N = #negocios(c).
3. NEGOCIOSCONREGALOS(**in** c: `cumple`) $\rightarrow$ res : `conj(negocio)`
   Complejidad: $\mathcal{O}(1)$.
4. REGALOMÁSBARATO(**in** c: `cumple`) $\rightarrow$ res : `regalo`
   Complejidad: $\mathcal{O}(1)$.
5. COMPRARREGALOMÁSBARATO(**in**/**out** c: `cumple`, **in** n: `negocio`, **in** l: `dicc(regalo, nat)`)
   Complejidad: $\mathcal{O}(N + log(R))$. N = #negocios(c) y R = total regalos del sistema.

Representación

`negocio` **es** `nat`
`regalo` **es** `nat`
`precio` **es** `nat`

`cumple` **se representa con** `estr`

donde `estr` es `tupla`(*negocios*: `diccAVL<negocio, minHeap⟨precio, itConj⟩>` ,
regalosPorNegocio: `diccAVL<negocio, conjLineal(regalo)>` ,
negociosConRegalos: `conjAVL(negocio)` ,
regaloMásBarato: `tupla<negocio, precio, itConj>`)

Algoritmos

1. PUBLICARLISTA(**in**/**out** c: `cumple`, **in** n: `negocio`, **in** ℓ: `dicc(regalo, nat)`)

   a) Itero sobre el diccionario de regalos y me armo dos conjuntos que no van a tener elementos repetidos, por esto, usamos `AgregarRapido()`. Uno de los conjuntos va a ser usado para guardar los regalos por negocio y otro va a ser usado para lo mismo, pero de forma ordenada por precio en un heap. Por cada elemento que inserto en el primer conjunto, guardo su iterador en el segundo conjunto y además, comparo el precio de cada regalo con el que tengamos en `c.regaloMásBarato`. En el que caso haya un nuevo regalo más barato, lo reemplazamos (modificamos `c.regaloMásBarato` y le insertamos el nuevo negocio, precio e iterador al regalo). Costo: $\mathcal{O}(L)$. Iterar un conjunto. Generar dos conjuntos nuevos, sabiendo que no contienen repetidos.

   b) A partir del conjunto generado anteriormente que contenga por cada regalo su <precio, itConj>, uso BuildMinHeap para hacerme un heap que ordene a partir del precio del regalo y guarde su iterador para poder eliminar de `c.regalosPorNegocio` a futuro. Costo: $\mathcal{O}(L)$. Algoritmo de Floyd para generar un minHeap usando BUILD-MIN-HEAP y MIN-HEAPIFY[1].

   c) Inserto en `c.negociosConRegalos` el identificador del negocio. Costo: $\mathcal{O}(log(N))$. Inserción en conjAVL.

[1] Cormen, *Introduction To Algorithms (Third Edition)*, 156-159

d) Inserto en `c.regalosPorNegocio` del negocio correspondiente el conjunto de regalos previamente generado. Costo: $\mathcal{O}(log(N))$. Definir en diccAVL.

iPublicarLista(**in**/**out** c: `cumple`, **in** n: `negocio`, **in** ℓ: `dicc(regalo, nat)`)

```
1:  regalosPorNegocio ← Vacio()
2:  preciosIterador ← Vacio()
3:  itRegalos ← CrearIt(ℓ)
4:  while HaySiguiente?(itRegalos) do                                  ▷ O(L)
5:      regalo ← SiguienteClave(itRegalos)
6:      precio ← SiguienteSignificado(itRegalos)
7:      itRegalo ← AgregarRapido(regalosPorNegocio, regalo)            ▷ O(1)
8:      heapElem ← ⟨precio, itRegalo⟩
9:      AgregarRapido(preciosIterador, heapElem)                       ▷ O(1)
10:     if precio < π2(c.regaloMasBarato) ∨ π2(c.regaloMasBarato) = 0 then
11:         c.regaloMasBarato ← ⟨n, precio, itRegalo⟩                  ▷ O(1)
12:     Avanzar(itRegalos)
13: minHeap ← BuildMinHeap(preciosIterador)                            ▷ O(L)
14: Definir(c.negocios, n, minHeap)                                    ▷ O(log(N))
15: Definir(c.regalosPorNegocio, n, regalosPorNegocio)                 ▷ O(log(N))
16: Agregar(c.negociosConRegalos, n)                                   ▷ O(log(N))
```

2. REGALOS(**in** c: `cumple`, **in** n: `negocio`) $\rightarrow$ res : `conj(regalo)`

   a) Devolver el valor que se encuentra al `obtener(n, c.regalosPorNegocio)`. Costo: $\mathcal{O}(log(N))$. Búsqueda en un diccAVL.

3. NEGOCIOSCONREGALOS(**in** c: `cumple`) $\rightarrow$ res : `conj(negocio)`

   a) Devolver el valor que se encuentra en `c.negociosConRegalos`. Costo: $\mathcal{O}(1)$. Búsqueda en memoria.

4. REGALOMÁSBARATO(**in** c: `cumple`) $\rightarrow$ res : `regalo`

   a) Devolver el valor que se encuentra en $Siguiente(\pi_3($`c.regaloMásBarato`$))$. Costo: $\mathcal{O}(1)$. Búsqueda en memoria.

5. COMPRARREGALOMÁSBARATO(**in**/**out** c: `cumple`)

   a) Buscar en `c.negocios` el minHeap en donde está el regalo más barato y extraerlo. Costo: $\mathcal{O}(log(N) + log(R))$. Búsqueda en diccABL + extraer el primer elemento y mantener ordenado un minHeap. El peor caso para extraer del minHeap con regalos es $log(R)$ y ocurre cuando un negocio contiene todos los regalos. El caso exacto es $log(regalos(negocio))$.

   b) En el caso que el negocio no tenga mas regalos, eliminar de la lista `c.negociosConRegalos` el negocio donde estaba el regalo más barato. Costo: $\mathcal{O}(log(N))$. Eliminar en conjAVL.

   c) Eliminar de `c.regalosPorNegocio` el regalo del negocio sobre el que acabamos de comprar el más barato. Costo: $\mathcal{O}(1)$. Eliminar un elemento de un `conjLineal` a partir de su iterador.

   d) Iterar por todos los negocios en `c.negocios` y a partir del primer elemento de cada minHeap, calcular el nuevo `c.regaloMásBarato`. Costo: $\mathcal{O}(N)$. Iterar un diccAVL. En cada paso, acceder al regalo más barato contenido en el minHeap es $\mathcal{O}(1)$.

   e) Costo final: $\mathcal{O}(N + log(R))$.

iComprarRegaloMásBarato(in/out c: `cumple`**)**

```
1:  negocioRegaloMasBarato ← π1(c.regaloMasBarato)
2:  iteradorRegaloMasBarato ← π3(c.regaloMasBarato)
3:  EliminarSiguiente(iteradorRegaloMasBarato)                          ▷ O(1)
4:  minHeap ← Significado(c.negocios, negocioRegaloMasBarato)           ▷ Supongo aliasing. O(log(N))
5:  Extraer(minHeap)                                                    ▷ O(log(R))
6:  if Vacio?(minHeap) then
7:      Eliminar(c.negociosConRegalos, negocioRegaloMasBarato)          ▷ O(log(N))
8:  if ¬Vacio?(c.negociosConRegalos) then
9:      minPrecio ← NULL                                                ▷ O(1)
10:     itNegocios ← CrearIt(c.negocios)
11:     while HaySiguiente?(itNegocios) do                              ▷ O(N)
12:         heapRegalos ← SiguienteSignificado(itNegocios)
13:         masBarato ← Primero(heapRegalos)          ▷ Regalo más barato del negocio actual
14:         if masBarato.precio ≤ minPrecio ∨ minPrecio = NULL then
15:             minPrecio ← masBarato.precio
16:             c.regaloMasBarato ← ⟨n, masBarato.precio, masBarato.iterador⟩
17:         Avanzar(itNegocios)
18: else
19:     c.regaloMasBarato ← ⟨0, 0, NULL⟩
```

Notas y aclaraciones

(a) Suponemos que existe una función `BuildMinHeap` que a partir de un conjunto de tuplas $\langle precio, itConj \rangle$ genera un MinHeap ordenado según el precio en tiempo lineal[1]. Ademas, el heap no solo tiene como clave para guardar el precio sino que tambiíen guarda el itConj y podemos acceder a cada uno de los valores al preguntar por el primer elemento del heap `.precio` o `.iterador`.

(b) El diccionario recibido como parámetro en PUBLICARLISTA se puede iterar de forma lineal con `itDicc`.

(c) La especificación no pide eliminar un negocio, una vez que se publican los regalos de un negocio, estos solamente pueden ser comprados pero no se pueden agregar más al mismo.

(d) Comprar el regalo más barato actualiza `c.regaloMásBarato`, el caso de no haber más negocios con regalos disponibles, esa tupla se vuelve $\langle 0, 0, NULL \rangle$. Suponemos que no pueden haber regalos con precio = 0.