Algortimos y Estructuras de Datos

2do Cuatrimestre 2024

Complejidades de Estructuras

Rep/Abs

Elección de Estructuras

COMPLEJIDADES

CONSTRUÍR UNA TABLA COMPARATIVA CON LAS COMPLEJIDADES EN PEOR CASO DE LAS OPERACIONES COMUNES PARA CONJUNTOS SOBRE LAS SIGUIENTES ESTRUCTURAS

	Pertenece	Insertar	Borrar	Buscar Minimo	Borrar Minimo
Array					
Lista Enlazada					
ABB					
AVL					
Heap					
Trie					

COMPLEJIDADES

	Pertenece	Insertar	Borrar	Buscar Minimo	Borrar Minimo
Array	O(n)	O(1)	O(1)	O(n)	O(n)
Lista Enlazada	O(n)	O(1)	O(n)	O(n)	O(n)
ABB	O(n)	O(n)	O(n)	O(n)	O(n)
AVL	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
Heap	O(n)	O(log n)	O(n)	O(1)	O(log n)
Trie	O(|k|)	O(|k|)	O(|k|)	O(|k|)*	O(|k|)*

ACLARACIONES

Trie

|k| es la longitud de la palabra mas larga. Se supone que el alfabeto usado es finito, ya que, de otra forma, el costo sería O(|k|*|a|), donde |a| es el tamaño del alfabeto.

Heap

Si lo que se busca es el mínimo/máximo para un min/maxHeap el costo de buscar es O(1), mientras que si es cualquier otro elemento es O(n).
La eliminación está acotada por O(log n) siempre y cuando se sepa la ubicación del elemento a borrar, ya sea que es el mínimo/máximo si es min/maxHeap, o porque primero se realizó una búsqueda del mismo (que puede tomar O(n)).

Array

Los costos de las operaciones indicadas implican cambiar o borrar un valor de una posición del array, si se sabe el índice de la posición a modificar. Si hubiera que mover el resto de los elementos, se agrega un costo O(n) en el peor caso.

EJERCICIO 2 GUIA 8

CONSIDERE LA SIGUIENTE ESPECIFICACION DE UNA RELACION UNO/MUCHOS ENTRE ALARMAS Y SENSORES DE UNA PLANTA INDUSTRIAL: UN SENSOR PUEDE ESTAR ASOCIADO A MUCHAS ALARMAS, Y UNA ALARMA PUEDE TENER MUCHOS SENSORES ASOCIADOS

```
TAD Planta {
obs alarmas: conj⟨Alarma⟩
obs sensores: conj⟨⟨Sensor, Alarma⟩⟩
proc nuevaPlanta () : Planta {
    asegura {res.alarmas = {}}
    asegura {res.sensores = {}}
}
```

```
proc agregarAlarma (inout p : Planta, in a : Alarma) {
    requiere {p = P_0}
    requiere {a ∉ p.alarmas}
    asegura {p.alarmas = P_0.alarmas ∪ {a}}
    asegura {p.sensores = P_0.sensores}
}

    proc agregarSensor (inout p : Planta, in a : Alarma, in s : Sensor) {
        requiere {p = P_0}
        requiere {a ∈ p.alarmas}
        requiere {⟨s, a⟩ ∉ p.sensores}
        asegura {p.alarmas = P_0.alarmas}
        asegura {p.sensores = P_0.sensores + {⟨s, a⟩}}
    }
}
```

SE DECIDIO UTILIZAR LA SIGUIENTE ESTRUCTURA COMO REPRESENTACION, QUE PERMITE CONSULTAR FACILMENTE TANTO EN UNA DIRECCION (SENSORES DE UNA ALARMA) COMO EN LA CONTRARIA (ALARMAS DE UN SENSOR).

```
Módulo PlantaImpl implementa Planta <
    var alarmas: Diccionario<Alarma, Conjunto<Sensor>>
    var sensores: Diccionario<Sensor, Conjunto<Alarma>>
>
```

SE PIDE:

- **ESCRIBIR FORMALMENTE Y EN CASTELLANO EL INVARIANTE DE REPRESENTACION.**
- **ESCRIBIR LA FUNCION DE ABSTRACCION.**

EJERCICIO 2 GUIA 8

REP

A- Para cada clave del diccionario alarmas, todos los elementos del conjunto que corresponde son claves del diccionario sensores

B- Para cada clave del diccionario sensores, todos los elementos del conjunto que corresponde son claves del diccionario alarmas

C- Si un sensor está en el conjunto que corresponde a una alarma en el diccionario alarmas, entonces esa alarma tiene que estar en el conjunto correspondiente del sensor en sensores

D- Si una alarma está en el conjunto que corresponde a un sensor en el diccionario sensores, entonces ese sensor tiene que estar en el conjunto correspondiente de la alarma en alarmas

E-No hay sensores sin alarmas asociadas

```
pred Rep (p: Planta) {
    alarmasValidas(p) ∧ sensoresValidos(p)
}
pred alarmasValidas (p: Planta) {
    (∀a : Alarma)(a ∈ p.alarmas →L (∀s : Sensor)(s ∈ p.alarmas[a] →L s ∈ p.sensores ∧ a ∈ p.sensores[s]))
}
pred sensoresValidos (p: Planta) {
    (∀s : Sensor)(s ∈ p.sensores →L (∀a : Alarma)(a ∈ p.sensores[s] →L a ∈ p.alarmas ∧ s ∈ p.alarmas[a]) ∧ |p.sensores[s]| > 0)
}
```

EJERCICIO 2 GUIA 8

ABS

A- Toda key de var Alarmas esta en Alarmas y viceversa

B-Para toda tupla en sensores , el primer componente de la tupla esta en las claves de sensores y el segundo componete esta en el conjuto Alarma

C-Toda clave del var Sensores esta como tupla en el obs Sensores con todas las alarmas del significado

pred Abs (p': PlantaImpl, p: Planta) {

$$\overbrace{alarmasConsistentes(p', p)}^{A} \land \underbrace{sensoresValidosEnTad(p', p)}_{B} \land \overbrace{sensoresValidosEnImple(p', p)}^{C}$$

}

pred alarmasConsistentes (p': PlantaImpl, p: Planta) {

$$(\forall a : Alarma)(a \in p'.alarmas \leftrightarrow a \in p.alarmas)$$

}

pred sensoresValidosEnTad (p': PlantaImpl, p: Planta) {

$$(\forall t : Tupla\langle Sensor, Alarma\rangle)(t \in p.sensores \rightarrow_L t_0 \in p'.sensores \land t_1 \in P'.sensores[t_0])$$

}

pred sensoresValidosEnImple (p': PlantaImpl, p: Planta) {

$$(\forall s : Sensor)\big(s \in p'.sensores \rightarrow_L (\forall a : Alarma)(a \in p'.sensores[s] \rightarrow \langle s, a\rangle \in p.sensores)\big)$$

}

EJERCICIO 9 GUIA 8

LA MADERERA SAN BLAS VENDE, ENTRE OTRAS COSAS, LISTONES DE MADERA.LOS COMPRA EN ASERRADEROS DE LA ZONA, LOS CEPILLA Y ACONDICIONA, Y LOS VENDE POR MENOR DEL LARGO QUE EL CLIENTE NECESITE.
TIENEN UN SISTEMA UN POCO PARTICULAR Y CIERTAMENTE NO MUY EFICIENTE: CUANDO INGRESA UN PEDIDO, BUSCAN EL LISTON MAS LARGO QUE TIENE EN EL DEPOSITO, REALIZAN EL CORTE DEL TAMAÑO QUE EL CLIENTE PIDIO, Y DEVUELVEN EL TROZO QUE QUEDA AL DEPOSITO.
POR OTRA PARTE, IDENTIFICAN A CADA CLIENTE CON UN CODIGO ALFANUMERICO DE 10 DIGITOS Y CUENTAN CON UN FICHERO EN EL QUE REGISTRAN TODAS LAS COMPRAS QUE HIZO CADA CLIENTE (CON LA FECHA DE LA COMPRA Y EL TAMAÑO DEL LISTON VENDIDO). ESTE SERIA EL TAD SIMPLIFICADO DEL SISTEMA:

```
Cliente es string
TAD Maderera {

proc comprarUnListon (inout m: Maderera, in tamaño: ℤ) {
    // comprar en el aserradero un listón de un determinado tamaño
}

proc venderUnListon (inout m: Maderera, in tamaño: ℤ, in cli: Cliente, in f: Fecha) {
    // vender un listón de un determinado tamaño a un cliente particular en una fecha determinada
}

proc ventasACliente (in m: Maderera, in cli: Cliente) {
    // devolver el conjunto de todas las ventas que se le hicieron a un cliente
    // (para cada venta, se quiere saber la fecha y el tamaño del listón)
}
}
```

SE PIDE:
ESCRIBA UNA ESTRUCTURA QUE PERMITA REALIZAR LAS OPERACIONES INDICADAS CON LAS SIGUIENTES COMPLEJIDADES:

- comprarUnListon en O(log(m)) - venderUnListon en O(log(m)) - ventasACliente en O(1)

DONDE M ES LA CANTIDAD DE PEDAZOS DE LISTON QUE HAY EN EL DEPOSITO

ESCRIBA EL ALGORITMO PARA LA OPERACION **venderUnListon**

EJERCICIO 9 GUIA 8

Eleccion de Estructura

VEAMOS QUE ESTRUCTURA NECESITAMOS PARA CUMPLIR CADA UNA DE LAS COMPLEJIDADES REQUERIDAS.

ventasACliente en O(1)

Para poder conseguir todas las ventas que se le hiceron a un mismo cliente en O(1), Para esto tendriamos que usar un DiccTrie, donde la clave sea el cliente y el significado un conjunto con todas las ventas realizadas a ese cliente. La complejidad es 0(1) debido a que la palabra mas larga esta acotada

comprarUnListon en O(log(m))

Para este caso como para venderUnListon dependemos de m, la cantidad de listones disponibles. Para esto podemos utilizar una cola de prioridad ordenada por el Máximo, para mantener el próximo listón a cortar en O(log m)

venderUnListon en O(log(m))

En el caso de la venta también necesitamos registrarla en el diccionario del cliente. Para que esto no nos afecte las complejidades, el conjunto será lineal para poder hacer agregarRapido en O(1) (ya que nunca vamos a tener ventas con la misma fecha)

EJERCICIO 9 GUIA 8

Eleccion de Estructura

ESCRIBA EL ALGORITMO PARA LA OPERACION **venderUnListon**

```
Liston es int
Cliente es string
Módulo MadereraImpl implementa Maderera <
      var listones: ColaPrioridadMax<Liston>
      var clientes: DiccTrie<Cliente, ConjLista<Fecha, Liston>>
>
```

```
function VENDERUNLISTON(inout M: Maderera, in c: Cliente, in tamaño: int, in f: fecha)
    int tamañoMax := M.listones.desencolarMax()                          O(log m)
    int resto := tamañoMax - tamaño
    if resto > 0 then
      M.listones.encolar(resto)                                          O(log m)
    end if
    ConjLista<Fecha, Liston> ventasCliente := M.clientes.obtener(c)      O(1)
    ventasCliente.agregarRapido(<f, tamaño>)                             O(1)
end function
```

Terminamos!!

Momento de Consultas