Algoritmos y Estructuras de Datos
Segundo recuperatorio – Martes 12 de diciembre de 2023

#Orden	Libreta	Apellido y Nombre	E1	E2	E3	Nota Final
168	190/22	PACHECO Tomás	40	26	27	93 (P)

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen, además de los apuntes de la cátedra.
- Cada ejercicio debe entregarse en **hojas separadas**.
- Incluir en cada hoja el número de orden asignado, número de libreta, número de hoja, apellido y nombre.
- El parcial se aprueba con 60 puntos. Para promocionar es necesario tener al menos 70 y ningún ejercicio con 0 puntos (en ambos parciales).

E1. Elección de estructuras (40 pts)

Se quiere implementar el `TAD Agenda` que modela una agenda semanal donde se registran actividades. Cada actividad tiene un identificador, un horario de inicio y uno de finalización. No puede haber dos actividades con el mismo identificador. Para simplificar, las actividades sólo pueden comenzar y terminar en horarios en punto (por ejemplo, 21:00 hs) y terminan en un horario posterior a su inicio. Tampoco pueden empezar y terminar en días diferentes. Para contar la cantidad de actividades que transcurren en un determinado horario no se debe tener en cuenta aquellas que finalizan en ese momento. En cada actividad se pueden agregar tags que permiten agrupar las distintas actividades por temáticas. Los tags tienen cómo máximo 20 caracteres.

Dadas las siguientes operaciones y de acuerdo a las complejidades temporales de peor caso indicadas, donde d es la cantidad de días registrados hasta el momento y a la cantidad total de actividades, respectivamente:

- `proc RegistrarActividad(inout ag: Agenda, in act: IdActividad, in día: Día, in inicio: Hora, in fin: Hora)`
  Requiere: la hora de fin sea mayor que la de inicio y la actividad no está actualmente registrada.
  Descripción: se agrega la actividad a la agenda, marcando en el día indicado la hora de inicio y finalización. 0
  Complejidad: $O(\log(a) + \log(d))$
- `proc VerActividad(in ag: Agenda, in act: IdActividad): struct<día: Día, inicio: Hora, fin: Hora>`
  Requiere: la actividad debe estar registrada en la agenda.
  Descripción: se devuelven el día y horario en que se realiza la actividad.
  Complejidad: $O(\log(a))$.
- `proc AgregarTag(inout ag: Agenda, in act: IdActividad, in t: Tag)`
  Requiere: la actividad está registrada en la agenda y aún no tiene registrado ese tag.
  Descripción: se agrega el tag a la actividad indicada.
  Complejidad: $O(1)$.
- `proc HoraMásOcupada(in ag: Agenda, in d: Día): Hora`
  Requiere: el día indicado tiene al menos una actividad registrada.
  Descripción: se devuelve la hora que tiene más actividades registradas.
  Complejidad: $O(\log(d))$.
- `proc ActividadesPorTag(in ag: Agenda, in t: Tag): Conjunto<IdActividad>`
  Requiere: true.
  Descripción: se devuelven las actividades que tienen registrado el tag t.
  Complejidad: $O(1)$.

Se pide:

a) Definir la estructura de representación del módulo `AgendaImpl`, que provea las operaciones solicitadas. Se debe explicar detalladamente qué información se guarda en cada parte y las relaciones entre ellas.
b) Explicar cómo se cumplen las complejidades pedidas por cada operación, haciendo aclaraciones sobre aliasing. Indicar todas las suposiciones tomadas sobre la implementación de las estructuras mencionadas.
c) Escribir el algoritmo de `RegistrarActividad` justificando **detalladamente** que se cumple la cota de de complejidad requerida.

E2. Invariante de representación y función de abstracción (30 pts)

Se tiene el TAD `Materia` que modela las calificaciones obtenidas por los alumnos de las clases prácticas una vez concluida la cursada (con todas las notas ya cargadas).

```
TAD Materia {
    obs calificaciones: dict<Alumno, struct<primero: Nota, segundo: Nota>>
    obs aprobados: conj<Alumno>
    obs reprobados: conj<Alumno>
}
Alumno es int
Nota es int
```

`calificaciones` relaciona a cada alumno con la nota que obtuvo en cada examen. La materia cuenta con dos exámenes, la primera posición del struct corresponde a la nota del primer examen y la segunda, a la del segundo. Las calificaciones toman valores entre 0 y 10. Si un alumno no se presenta a un parcial, su nota en ese parcial será cero. `aprobados` es el conjunto de los alumnos que aprobaron la materia. Un alumno aprueba si tiene más de 6 en ambos exámenes. `reprobados` es el conjunto de alumnos que reprobaron la materia.

Se utiliza la siguiente estructura para implementar el TAD descrito:

```
Módulo MateriaImpl implementa Materia {
    var primerParcial: Diccionario<Alumno, Nota>
    var segundoParcial: Diccionario<Alumno, Nota>
    var alumnos: Conjunto<Alumno>
    var aprobados: Conjunto<Alumno>
}
```

`primerParcial` y `segundoParcial` asocian a cada alumno con la nota que obtuvieron en cada examen. `alumnos` son todos los alumnos que tiene la materia. `aprobados` son aquellos alumnos que aprobaron la materia, es decir, que tienen más de 6 en ambos exámenes.

Se pide:

a) Escribir en forma coloquial y detallada el invariante de representación y la función de abstracción.
b) Escribir ambos en el lenguaje de especificación.

E3. Sorting (30 pts)

Se cuenta con un sistema de seguimiento de contaminación ambiental que cubre toda la ciudad de Buenos Aires.

Así, regularmente se registra información de diferentes sensores en diferentes momentos y se quiere saber en qué zonas se registra mayor contaminación, ya que cada sensor está asociado a una zona particular de la urbe.

De esta forma, dados ciertos registros realizados, que constan del identificador del sensor, que es un valor alfanumérico de máximo 64 caracteres, el momento en que se tomó el registro y el valor medido, donde ambos son naturales no acotados, se quiere obtener el acumulado de cada sensor para las últimas k mediciones (se puede considerar cualquier valor de k, en tanto se tiene una gran cantidad de registros por cada sensor). Con esta información se desea hacer un ranking, donde aparezcan primero los sensores con mayor contaminación registrada acumulada. En caso de empate, se mostrarán primero los sensores con registros más recientes.

Se quiere implementar la función MÁSCONTAMINADOS, que recibe un arreglo que contiene mediciones de los sensores y la cantidad de mediciones que se desean considerar para obtener un acumulado, y se desea obtener un *ranking*, donde primero aparecen los sensores que tienen mayor valor total:

```
MásContaminados(in a: Array<struct<sensor: string, in t:int, in v:int>>, in k: int): Array<string>
```

La función debe tener cota de peor caso $O(n.\log n)$, siendo n la cantidad de mediciones registradas entre todos los sensores.

Por ejemplo, dados el A indicado abajo y k igual a 2, `MásContaminados(A, 2)` retornará `[48A, 1AB, 1C]`, ya que tanto el sensor `1AB` como el `48A` registran un acumulado de 120 en las últimas 2 mediciones, pero el `48A` tiene un registro más reciente, por lo que queda primero, y luego queda el `1C`, que sólo acumula 55.

```
A = [ (1AB, 8, 100), (48A, 10, 100), (1AB, 9, 25), (48A, 9, 25),
    (1AB, 14, 95), (48A, 15, 20), (1C, 12, 5), (1C, 17, 50) ]
```

a) Se pide escribir el algoritmo de `MásContaminados`, justificando **detalladamente** la complejidad.
b) ¿Cuál sería el *mejor caso* para este algoritmo? ¿Cuál sería la cota de complejidad más ajustada?

PACHECO THOMAS LU: 190/22 | HOJA 1

1)

a)

VAR ACTIVIDADES : DICLOG <IDACTIVIDAD, STRUCT <DÍA, INICIO, FIN>>
VAR TEMÁTICAS : DICTRIE <TAG, CONJLIN <IDACTIVIDAD>>
VAR MÁXIMAHORA : DICLOG <DÍA, DICLOG <HORA, NAT>>
VAR MAX : DICLOG <DÍA, HORA>

ALMACENO CADA ID ACTIVIDAD EN ACTIVIDADES CON SU DÍA, INICIO Y FIN. A SU VEZ EN TEMÁTICAS CATALOGO SEGÚN EL TAG, EL CONJUNTO DE ACTIVIDADES (DE LAS QUE PERTENECEN EN ACTIVIDADES). LUEGO MAXIMAHORA, ALMACENO EN CADA DÍA EL HORARIO CON MÁS ACTIVIDADES. ESTÁ ÍNTIMAMENTE RELACIONADO CON MAX YA QUE VOY GUARDANDO LA HORA CON EL QUE TIENE LA MAYOR # DE ACTIVIDADES DE ESE DÍA. SABIENDO QUE HORA ESTÁ ACOTADA AL IGUAL QUE TAG

b)

PROC REGISTRARACTIVIDAD (INOUT AG: AGENDA, IN ACT: IDACTIVIDAD, IN DIA: DÍA, IN INICIO: HORA, IN FIN HORA)

- AÑADO LA ACTIVIDAD A ACTIVIDADES CON SU STRUCT... ESO ES $\Theta(\log(a))$

- DESPUÉS EN MÁXIMAHORA AÑADO EL DÍA Y A SU VEZ AÑADO [INICIO, FIN] LE SUMO 1 COMO HORA ESTÁ ACOTADA? Y COMPARO MI MAX SI ALGUNO ES MAYOR (TIENE MÁS ACTIVIDAD) LO MODIFICO. ANTES TAMBIÉN DEFINO MAX CON EL DÍA DADO Y 0. NO HAY ALIASING

COMPLEJIDAD: $O(\log(a) + \log(D) + \log(D) + O(1)) = O(\log(a) + \log(D))$ (DICTRIE)

PROC VERACTIVIDAD (IN AG: AGENDA, IN ACT: IDACTIVIDAD) : STRUCT <DIA, INICIO, FIN>

DEVUELVO POR ALIASING (POR REFERENCIA) POR LO TANTO
ES ACCEDER A ACTIVIDADES Y DEVOLVER

COMPLEJIDAD: O(LOG (a))

PROC AGREGARTAG (INOUT AG: AGENDA, IN ACT: IDACTIVIDAD, IN T: TAG)

DEFINO EN TEMATICAS TAG, AL ESTAR ACOTADO ES O(1)
Y DESPUES A MI CONJ LINEAL, LO AGREGO RAPIDO O(1) LA ACTIVIDAD
NO HAY ALIASING
COMPLEJIDAD: O(1)

PROC HORAMASOCUPADA (IN AG: AGENDA, IN D: DIA) : HORA

DEVUELVO POR ALIASING EN DE MAXIMASHORAS, OBTENGO EL
DIA Y LO DEVUELVO EL VALOR
COMPLEJIDAD O(LOG d) Notá que nada cambia si la buscabas en maximahora pq son a la suma 24 = cte

PROC ACTIVIDADESPORTAG (IN AG: AGENDA, IN T: TAG) : CONJUNTO <IDACTIVIDAD>

DEVUELVO POR ALIASING EL VALOR, COMO TEMATICAS ES
UN DICTRIA CON TAG ACOTADO, ES O(1)
OBTENER

COMPLEJIDAD: O(1)

Asamblea

e)

REGISTRARACTIVIDAD {

AG.ACTIVIDADES.DEFINIR(IDACTIVIDAD, STRUCT<DIA,INICIO,FIN>) //θ(log d)

IF AG.MAXIMAHORA.ESTA(DIA) THEN //θ(log d)

FOR(INT i = INICIO, i <= FIN i++){

IF !AG.MAX.ESTA(DIA) THEN

AG.MAX.DEFINIR(DIA, 0)

FI

~~AG.ESTA(AG, INICIO, FIN)~~

IF AG.MAXIMAHORA.ESTA(DIA) THEN //θ(log d)

FOR(INT i = INICIO; i <= FIN; i++) { θ(1)

AG.MAX.OBTENER(DIA).OBTENER(i)++ //θ(log d + log 1 + 1) ← LE SUMO

}

FOR(INT i = INICIO; i <= FIN; i++) { //θ(1)

IF AG.MAX.OBTENER(DIA) < i THEN //θ(log d)

AG.MAX[DIA] = i //θ(1)

FI

}

ESTA(A

FOR(INT i = INICIO; i <= FIN; i++) { //θ(1)

IF !AG.MAXIMAHORA.[DIA].ESTA(i) THEN //θ(log d)

AG.MAXIMAHORA.DEFINIR(i, 0) //θ(log d)

FI

}

PARA SALVARLO EN CASO DE QUE EN ESA FRANJA NO HAYA DEFINIDOS

EN RESUMEN ORDENO POR ESTA DEFINIDO Y DESPUES HAGO COMPARACIONES EN UN INTERVALO ACOTADO POR LO TANTO ES A LO SUMO LOGARITMICO DEFINIR, OBTENER PERO O(1) RECORRER...

POR LO TANTO LA COMPLEJIDAD FINAL ES $O(\log d + \log e)$

b)

No es lo que quiero decir

POS ≡ A

PROC INVERS(m': MATERIAIMPL) {

FORALL : A: ~~ESTUDIANTE~~ ALUMNO :: A IN m'.PRIMERPARCIAL.DATA &&
A IN m'.SEGUNDOPARCIAL.DATA <=> A IN m'.ALUMNOS.ELEMS
&&
FORALL : A:ALUMNO :: A IN m'.ALUMNOS => (0 <= m'.PRIMERPARCIAL[A].NOTA <=
&& 0 <= m'.SEGUNDOPARCIAL[A] <= 10)
&&
FORALL : A:ALUMNO :: A IN m'.APROBADOS.ELEMS <=>
(m'.PRIMERPARCIAL[A].NOTA >= 6 && m'.SEGUNDOPARCIAL[A].NOTA >= 6)
}

AUX FUNCIONS(m': MATERIAIMPL) : MATERIA {

m: MATERIA

FORALL A:ALUMNO :: A IN m.CALIFICACIONES.DATA <=> A IN m'.ALUMNOS.ELEMS
&&
FORALL A:ALUMNO :: A IN m.CALIFICACIONES.DATA <=> (m.CALIFICACIONES[A].
PRIMERO == m'.PRIMERPARCIAL.DATA && m.CALIFICACIONES[A].SEGUNDO
== m'.SEGUNDOPARCIAL[A].DATA)
&&
FORALL A:ALUMNO :: A IN m.APROBADOS.ELEMS <=> A IN m'.APROBADOS.ELEMS
&&
~~EXISTS X: CONJUNTO<ALUMNO> ::~~
FORALL A:ALUMNO :: A IN m.DESAPROBADOS.ELEMS.DATA <=>
(A IN m'.ALUMNOS.ELEMS && ! A IN m'.APROBADOS.ELEMS)

PACHECO THOMÁS ; LU: 190/22 | HOJA 3

2)

a-

INVERP:

TANTO PRIMER PARCIAL COMO SEGUNDO PARCIAL / LAS NOTAS (PARA CADA ALUMNO) TIENEN QUE VALER ENTRE 0 Y 10 ([0,10]). ADEMÁS TODOS LOS ALUMNOS QUE ESTÁN EN PRIMERPARCIAL Y SEGUNDOPARCIAL DEBEN PERTENECER A ALUMNOS.
POR OTRO LADO, PARA TODO ALUMNO CUYA NOTA EN AMBOS (O SEA EN AMBOS) PARCIALES ES MAYOR ~~o igual~~ A 6 DEBEN PERTENECER A APROBADOS

FUNCABS:
TOMANDO AL INVERP VÁLIDO, MI FUNCABS DEBE SER:
PARA TODO ALUMNO QUE PERTENECE A CALIFICACIONES TIENE QUE ESTAR SI O SI EN ALUMNOS (MÓDULO) Y ADEMÁS LAS ~~CALIFICACIONES~~ NOTAS TIENEN QUE ~~SER~~ SALIR DEL PRIMER PARCIAL Y SEGUNDO PARCIAL ~~EXPLICANDO~~ QUE SI O SI DEBEN ESTAR EN CALIFICACIONES DE ESE ALUMNO, O SEA EN EL STREET. POR OTRO LADO LOS APROBADOS DEL MODULO DEBEN ESTAR SI O SI EN APROBADOS (ABS). POR ÚLTIMO ~~LOS DESAPROBADOS~~ PARA TODOS LOS ALUMNOS QUE PERTENECEN A PRIMER Y SEGUNDO PARCIAL (MÓDULO) CUYA CALIFICACIÓN (SUMA DE) NO SUPERE 12 ENTONCES DEBEN ESTAR SI O SI EN REPROBADOS (DES)

↳ El criterio es que ambos sean ≥ 6
Además ese alu tiene que estar en el Rep. Podes usar
desaprobados = alumnos − aprobados

PACHECO THOMAS LU: 190/22 | HOJA 4

3)

a)

MÁSCONTAMINADO (in A: ARRAY < STRUCT: STRING, in t: INT, in v: INT >, in K: INT): ARRAY<STRING>

(segunda componente)

MERGESORT(A) EN BASE A LA ÚLTIMA MEDICIÓN // $\Theta(m \log m)$
DE MAYOR A MENOR

d ← DICDIG < SENSOR, TUPLA < K, 0, 0 > > // $\Theta(1)$ → VACÍO
(NAT → ACTUALIZACIÓN (t))

¿?

RECORRO A Y VOY DEFINIENDO POR SENSOR COMO CLAVE E INICIALIZO COMO VALOR LA TUPLA < K, 0, 0 > PARA PODER HACER OPERACIONES ESO TIENE COMPLEJIDAD $\Theta(m)$. ACCEDER AL DICDIG Y DEFINIR ES O(1) ~~PERO~~

* ACLARACIÓN:
COMO ORDENÉ EN FUNCIÓN DE LA ÚLTIMA MEDICIÓN, PUEDO OBTENER LAS K MEDICIONES MAYORES DE CADA SENSOR...

FOR(INT i = 0; i < A.LENGTH; i++){ // $\Theta(m)$

IF d[A[i][0]][0] > 0 THEN // $\Theta(1)$

d[A[i][0]][0]-- // $\Theta(1)$
d[A[i][0]][1] += A[i][2] // $\Theta(1)$
d[A[i][0]][2] += A[i][1] // $\Theta(1)$

FI

↳ pusiste los valores [c] significado cruzados respecto a A?

}

RES ← ARREGLO TAMAÑO M // $\Theta(m)$
DE TUPLA < SENSOR, t, v >

En v está bien acumular, en t necesitabas el mayor valor (el 1° en tupla)

No entendí que quisiste hacer

CREO UN ITERADOR PARA EL DICCIONARIO Y VOY RECORRIENDO Y PONIENDO EN RES LOS VALORES PEDIDOS DE LA TUPLA. ~~EL~~ LA COMPLEJIDAD SERÍA $\Theta(m)$. ADEMÁS ACCEDER A CADA VALUE ES O(1)

UNA VEZ LISTO, APLICO MERGESORT EN RES EN BASE AL VALOR (V) Y COMO DESEMPATE t (ÚLTIMA MEDICIÓN) // $\Theta(m \log m)$

POR ÚLTIMO CREO UN ARREGLO RES2 (PERSÚ POR EL NOMBRE "n") DE TAMAÑO M Y TRANSFIERO SOLO LOS STRING DE RES COMO APARECEN Y LISTO. ESO ES $\Theta(m)$

RETURN RES2.

LA COMPLEJIDAD FINAL ES: $\Theta(m+m \dots + m \log m)$ PERO COMO $O(m) \subset O(m \log m) \Rightarrow O(m \log m)$.

EL ALGORITMO ES ESTABLE. APLICAR MERGESORT NO ME AFECTA LA ESTABILIDAD, DESPUÉS UTILIZAR DICC Y ARRAYS AUXILIARES TAMPOCO

b) EL MEJOR CASO ES QUE SOLO HUBIERA UN TIPO DE SENSOR PERO AÚN ASÍ LA COMPLEJIDAD SERÍA $\Theta(m \log m)$ PERO PODRÍAMOS MEJORARLA... → COTA AJUSTADA

ANTES DE APLICAR EL PRIMER MERGE PODRÍA RECORRER EL ARREGLO, VERIFICAR SI ES UN TIPO SOLO Y PODRÍAMOS LOGRAR $\Theta(m)$ PERO PARA UN CASO PUNTUAL... "n"