DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Algoritmos y Estructuras de Datos (ex Algo II)

Simulacro de parcial

23 de noviembre de 2023 @valnrms

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
http://www.exactas.uba.ar

1. Ejercicio 1: Complejidad

Enunciado:

```
AlgoritmoExótico(A: arreglo de int, R: arreglo de tupla<val: int, reps: int>) {
  n := long(A)
  R[0] := <A[0],1>
  r := 0
  i := 1
  res := 0

  while (i < n) {
        if A[i] == R[r].val {
            R[r] := <R[r].val, R[r].reps + 1>
        } else {
            j := 1

            while (j <= R[r].reps) {
                res := res + R[r].val * j
                j := j + 1
            }

            r := r + 1
            R[r] := <A[i], 1>
        }
        i := i + 1
  }

  return res
}

a) Complejidad del peor caso
b) Complejidad del mejor caso
```

Resolución (Ej. 1):
Mi **recomendación** en la vida es: primero entendé, después te vas a las trompadas (así te vas con motivo).
Acá se traduciría en: **primero entendé el algoritmo, después resolvés los enunciado** (así resolvés correctamente).

El **input** del algoritmo es un A de tipo $Array < int >$ y un B de tipo $Array < Tupla < val : int, reps : int > >$
Los nombre **val** *(valores)* y **reps** *(repeticiones)*, nos ayuda a inferir a que refieren las tuplas.

Las primeras 5 línas son triviales, aunque es interesante **notar que pisamos la posición 0 del arreglo R con una tupla <A[0], 1>**. Esto nos puede ayudar a inferir el comportamiento del algoritmo.
Después me voy a preocupar de *res*. No intentemos entender todo de una.

Entramos al while.
Entramos en la rama *True*, si y solo si $A[i] == R[r].val$. **Esto nos infiere que i se usa para indexar A.** Idem r.

Entonces, observación: r es un dedo en el arreglo R, y i es un dedo en el arreglo A.

En caso de entrar en la rama *True*, se suma una repetición a al valor en R.
Veamos un ejemplo para entenderlo mejor:

```
A := [1,1,1]
R := [...]
R_2 := [<8,2>,<2,1>,<4,1>,<5,4>,<9,1>]

R_2 es un ejemplo más concreto de R.
Con esta entrada nos quedaría (según lo que vimos hasta ahora):

A := [1,1,1]
R := [<1,3>, ...]
R_2 := [<1,3>,<2,1>,<4,1>,<5,4>,<9,1>]
```

Podemos ir haciéndonos la idea temporal de que vamos a ir pisando los valores de R, metiendo los elementos que aparecen en A junto a sus repeticiones.

Vamos al caso *False* ahora.
Tenemos **otro while que depende de un valor que se incrementa afuera**, depende de las repeticiones. Todo lo de adentro es $O(1)$;).

Volvamos a un ejemplo para explicarlo mejor:

```
A := [1,1,3,5]
R := [...]
R_2 := [<8,2>,<2,1>,<4,1>,<5,4>,<9,1>]

Con esta entrada nos quedaría:

A := [1,1,3,5]
R := [<1,2>, <3,1>, <5,1>, ...]
R_2 := [<1,2>,<3,1>,<5,1>,<5,4>,<9,1>]

# Nótese que estoy ignorando res. No me piden "qué hace el algoritmo", me piden realizar en análisis asintótico dado del
    peor y el mejor caso. Y ya vimos antes que las operaciones que involucran a res son O(1).
```

Entonces..., vamos pisando el arreglo R, poniendo los valores de A junto a sus repeticiones. Esto nos invita a plantearnos tres casos.

- Todos los **elementos** del arreglo A son **iguales**.
- Todos los **elementos** del arreglo A son **distintos**.
- Todos los **elementos** del arreglo A son **iguales salvo el último elemento**.

Nótese que para entrer en el segundo while es necesario tener al menos un elemento distinto.

Ya sabemos entonces como es el algoritmo, y de ahí pudimos determinar los casos de interés.
Pensemos qué pasa con la complejidad si son todos iguales.

```
## Caso, todos los elementos son iguales.

AlgoritmoExótico(A: arreglo de int, R: arreglo de tupla<val: int, reps: int>) {
  n := long(A)                                  // O(1)
  R[0] := <A[0],1>                              // O(1)
  r := 0                                        // O(1)
  i := 1                                        // O(1)
  res := 0                                      // O(1)

  while (i < n) {                               // O(n)
        if A[i] == R[r].val {                   // O(1)
            R[r] := <R[r].val, R[r].reps + 1>   // O(1)
        } else {                                // -
            j := 1                              // -

            while (j <= R[r].reps) {            // -
                res := res + R[r].val * j       // -
                j := j + 1                      // -
            }

            r := r + 1                          // -
            R[r] := <A[i], 1>                   // -
        }
        i := i + 1                              // O(1)
  }

  return res                                    // O(1)
}
```

Lo que nos da $O(n)$.
Pues la rama del *True* se ejecuta n veces, mientras que la rama del *False* se ejecuta 0 veces.
Al no entrar nunca a la rama del *False*, nunca se ejecuta en segundo *while*.

```
## Caso, todos los elementos son distintos.

AlgoritmoExótico(A: arreglo de int, R: arreglo de tupla<val: int, reps: int>) {
  n := long(A)                                  // O(1)
  R[0] := <A[0],1>                              // O(1)
  r := 0                                        // O(1)
  i := 1                                        // O(1)
  res := 0                                      // O(1)

  while (i < n) {                               // O(n)
        if A[i] == R[r].val {                   // -
            R[r] := <R[r].val, R[r].reps + 1>   // -
        } else {                                // O(1)
            j := 1                              // O(1)

            while (j <= R[r].reps) {            // O(1)
# while (j <= R[r].reps) es O(1) pues al ser todos distintos, no hay repeticiones; lo que permite acotar las repticiones
    por "1 (una) repetición".
                res := res + R[r].val * j       // O(1)
                j := j + 1                      // O(1)
            }

            r := r + 1                          // O(1)
            R[r] := <A[i], 1>                   // O(1)
        }
        i := i + 1                              // O(1)
  }

  return res                                    // O(1)
```

```
}
```

Lo que nos da $O(n)$.
Pues la rama del *True* se ejecuta 0 veces, mientras que la rama del *False* se ejecuta n veces.
Y el *while* se ejecuta sola vez por iteración.

```
## Caso, todos los elementos son iguales menos el último elemento.

AlgoritmoExótico(A: arreglo de int, R: arreglo de tupla<val: int, reps: int>) {
  n := long(A)                                  // O(1)
  R[0] := <A[0],1>                              // O(1)
  r := 0                                        // O(1)
  i := 1                                        // O(1)
  res := 0                                      // O(1)

  while (i < n) {                               // O(n)
        if A[i] == R[r].val {                   // O(1)
            R[r] := <R[r].val, R[r].reps + 1>   // O(1)
        } else {                                // O(1)
            j := 1                              // O(1)

            while (j <= R[r].reps) {            // O(n-1)
                res := res + R[r].val * j       // O(1)
                j := j + 1                      // O(1)
            }

            r := r + 1                          // O(1)
            R[r] := <A[i], 1>                   // O(1)
        }
        i := i + 1                              // O(1)
  }

  return res                                    // O(1)
}
```

Uno acá estaría tentado a decir $O(n) * O(n-1) = O(n^2 - n) = O(n^2)$. **¡Pero eso está mal!**
La complejidad acá también es $O(n)$. Y voy a convencerte por qué.

Hagamos un análisis más profundo de **toda la ejecución del programa**.
¿**Cuántas veces entramos** a la rama del ***True***? –> $n-1$ veces.
¿**Cuántas veces entramos** a la rama del ***False***? –> 1 vez.
¿**Cuántas veces se ejecuta** el segundo ***while***? –> n veces.

Vamos despacio porque **esto es complicado**.

Entonces, entro al primer *while* y hago $n-1$ iteraciones de la rama $True$ del if.
Luego, en la n-ésima iteración *(la última)*, voy a la rama $False$; entonces,
Tengo las $n-1$ operaciones elementales que ejecuté antes, **más** $(+)$ las n operaciones elementales del segundo *while*.

(Recordatorio, un *while* $i < n$ con $i := 0$ que realiza $i := i + 1$ al final de cada iteración, da $O(n)$ porque hace $O(1 + ... + 1)$ donde el 1 se repite n veces. Es decir, hay n operaciones elementales. **¡Y estas se suman!**)

Retomando, tengo: $O(((n-1) * 1) + (1 * n))$ que es igual a $O((n-1) + n) = O(2n - 1) = O(2n) = O(n)$.

Ahora, **hay algo que nos debería hacer ruido**. ¿Cuál es el mejor y cuál es el peor caso?, ¿dan lo mismo? Planteamos nuestros escenarios, tres escenarios, y en todo dio O(n).

Acá el truco está en darse cuenta que tanto el **mejor**, como el **peor caso coinciden**. Podríamos determinar cual es el peor/mejor caso de la forma más robusta y cavernícola que hay. Contar una por una las operaciones elementales.

2. Ejercicio 2: InvRep y función de abstracción

Enunciado:

```
Considerar la siguiente especificación del show de TV de lucha libre llamado "99%_Lucha". Como en todos estos shows, hay
    luchadores del bando de los "buenos" y de los "malos". Los luchadores se pueden sumar en cualquier momento al show,
    pero no se van nunca. Así, se registran todas las luchas uno contra uno que ocurren a lo largo del programa.

TAD 99%Lucha {
    obs buenos:   conj(int)
    obs malos:  conj(int)
    obs cantLuchas: dict(tupla(b: int, m: int), nat)
}

Módulo

Modulo 99%LuchaImpl implementa 99%Lucha {
  var buenos:     Conjunto(nat)
  var malos:      Conjunto(nat)
  var contrincantes:    Diccionario(nat, Conjunto(nat))
  var históricoLuchas:  Vector(Tupla< b: nat, m: nat >)
}

Donde buenos y malos representan los conjuntos de identificadores de los luchadores buenos y malos, respectivamente,
    contrincantes asocia a cada luchador (tanto bueno como malo) con el conjunto de todos los contrincantes que tuvo al
    menos una vez y, por último, históricoLuchas tiene la secuencia de contrincantes (bueno, malo) que se enfrentaron, en
    el orden en que las luchas ocurrieron en el show.

a) Escribir coloquialmente el invariante de representación
b) Escribir formalmente el invariante de representación
c) Escribir formalmente la función de abstracción
```

Resolución (Ej. 2):