Análisis de Algoritmos I

José de Jesús Lavalle Martínez

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Maestría en Ciencias de la Computación
Análisis y Diseño de Algoritmos
MCOM 20300

Otoño 2020

Contenido

Llamados recursivos I

- El análisis de algoritmos recursivos usualmente es directo, al menos hasta cierto punto.

Llamados recursivos I

- Una simple inspección al algoritmo frecuentemente da lugar a una **ecuación de recurrencia** que imita el flujo de control en el algoritmo.

Llamados recursivos I

- Una vez que ha sido obtenida la ecuación de recurrencia, se pueden aplicar técnicas generales para resolver dichas ecuaciones y transformarlas a la notación asintótica no recursiva, la cual es más simple.

Llamados recursivos II

Como un ejemplo, considere de nuevo el problema de calcular la secuencia de Fibonacci, pero esta vez con el algoritmo recursivo $\textsc{Fibrec}(n)$.

function $\textsc{Fibrec}(n)$
1 **if** $n < 2$
2 **then return** n
3 **else return** $\textsc{Fibrec}(n-1) + \textsc{Fibrec}(n-2)$

Llamados recursivos II

Como un ejemplo, considere de nuevo el problema de calcular la secuencia de Fibonacci, pero esta vez con el algoritmo recursivo $\textsc{Fibrec}(n)$.

function $\textsc{Fibrec}(n)$
1 **if** $n < 2$
2 **then return** n
3 **else return** $\textsc{Fibrec}(n-1) + \textsc{Fibrec}(n-2)$

- Sea $T(n)$ el tiempo consumido por un llamado a $\textsc{Fibrec}(n)$. Si $n < 2$, el algoritmo simplemente regresa n, lo cual consume tiempo constante a.

Llamados recursivos II

Como un ejemplo, considere de nuevo el problema de calcular la secuencia de Fibonacci, pero esta vez con el algoritmo recursivo $\textsc{Fibrec}(n)$.

```
function FIBREC(n)
1   if n < 2
2      then return n
3      else  return FIBREC(n − 1) + FIBREC(n − 2)
```

- Sea $T(n)$ el tiempo consumido por un llamado a $\textsc{Fibrec}(n)$. Si $n < 2$, el algoritmo simplemente regresa n, lo cual consume tiempo constante a.
- De otra manera, la mayor parte del trabajo se gasta en los dos llamados recursivos, los cuales consumen respectivamente tiempo $T(n-1)$ y $T(n-2)$.

Llamados recursivos III

- Más aún, debe ser ejecutada una adición que involucra a f_{n-1} y a f_{n-2} (los cuales son los valores regresados por los llamados recursivos), así como, el control de la recursión y la prueba "**if** $n < 2$".

Llamados recursivos III

- Sea $h(n)$ que denota el trabajo involucrado en esta adición y control, esto es, el tiempo requerido por un llamado a $\textsc{Fibrec}(n)$ ignorando el tiempo gastado dentro de los dos llamados recursivos.

Llamados recursivos III

- Por definición de $T(n)$ y $h(n)$ obtenemos la siguiente recurrencia.

$$T(n) = \begin{cases} a & \text{si } n = 0 \text{ o } n = 1, \\ T(n-1) + T(n-2) + h(n) & \text{en otro caso} \end{cases} \tag{1}$$

Llamados recursivos IV

function FIBREC(n)

```
1   if n < 2
2       then return n
3       else  return FIBREC(n − 1) + FIBREC(n − 2)
```

$$T(n) = \begin{cases} a & \text{si } n = 0 \text{ o } n = 1, \\ T(n-1) + T(n-2) + h(n) & \text{en otro caso} \end{cases}$$

Llamados recursivos V

- Si contamos las adiciones de costo unitario, $h(n)$ está acotada por una constante. Al aplicarle a (1) técnicas para solución de recurrencias encontramos que $T(n) \in O(f_n)$.

Llamados recursivos V

- Un razonamiento similar muestra que $T(n) \in \Omega(f_n)$ y por tanto $T(n) \in \Theta(f_n)$.

Llamados recursivos V

- Usando la fórmula de de Moivre (esta fórmula nos dice que f_n el $n-$ésimo término en la secuencia, es aproximadamente igual a $\phi^n/\sqrt{5}$ donde $\phi = (1+\sqrt{5})/2$ es la **proporción dorada**), concluimos que $\textsc{Fibrec}(n)$ consume tiempo exponencial en n.

Llamados recursivos VI

- Si no contamos la adición de costo unitario, $h(n)$ ya no estará acotada por una constante.

Llamados recursivos VI

- En su lugar $h(n)$ está dominada por el tiempo requerido para la adición de f_{n-1} y f_{n-2} para n suficientemente grande.

Llamados recursivos VI

- Ya hemos visto que esta adición consume un tiempo en el orden exacto de n. Por tanto $h(n) \in \Theta(n)$.

Llamados recursivos VI

- Sorprendentemente, al aplicarle a (1) técnicas para solución de recurrencias, el resultado es el mismo sin importar cuando $h(n)$ es constante o lineal, es decir, se sigue cumpliendo que $T(n) \in \Theta(f_n)$.

Llamados recursivos VI

- En conclusión, ¡$\textsc{Fibrec}(n)$ consume tiempo exponencial en n sin importar que consideremos o no a las adiciones como de costo unitario!

Ciclos “while” y “repeat” I

- Los ciclos **while** y **repeat** usualmente son más difíciles de analizar que los ciclos **for** ya que no existe una manera a priori obvia para saber cuántas veces se entrará al ciclo.

Ciclos “while” y “repeat” I

- La técnica estándar para analizar estos ciclos es encontrar una función de las variables involucradas cuyo valor decrezca conforme se entre al ciclo.

Ciclos “while” y “repeat” I

- Para concluir que el ciclo eventualmente termina es suficiente con demostrar que este valor debe ser un entero positivo, ya que no se puede decrementar indefinidamente un entero positivo.

Ciclos “while” y “repeat” I

- Para determinar cuántas veces se repite el ciclo, necesitamos entender bien cómo decrece el valor de dicha función.

Ciclos “while” y “repeat” I

- Un enfoque alternativo para el análisis de los ciclos **while** consiste en tratarlos como algoritmos recursivos. Ilustraremos ambas técnicas con el mismo ejemplo.

Ciclos “while” y “repeat” II

- Usaremos el algoritmo de BinarySearch para ilustrar el análisis de ciclos **while**.

Ciclos “while” y “repeat” II

- El propósito de la búsqueda binaria es encontrar un elemento x en un arreglo $T[1 \ldots n]$ que está en **orden no decreciente**.

Ciclos “while” y “repeat” II

- Asuma por simplicidad que se garantiza que x aparezca al menos una vez en T. Se requiere encontrar un entero i tal que $1 \leq i \leq n$ y $T[i] = x$.

Ciclos “while” y “repeat” II

- Asuma por simplicidad que se garantiza que x aparezca al menos una vez en T. Se requiere encontrar un entero i tal que $1 \leq i \leq n$ y $T[i] = x$.
- La idea básica detrás de la búsqueda binaria es comparar x con el elmento y que está en la mitad de T.

Ciclos “while” y “repeat” II

- Asuma por simplicidad que se garantiza que x aparezca al menos una vez en T. Se requiere encontrar un entero i tal que $1 \leq i \leq n$ y $T[i] = x$.
- La idea básica detrás de la búsqueda binaria es comparar x con el elmento y que está en la mitad de T.
- La búsqueda termina si $x = y$, si $x > y$ se busca en la mitad superior del arreglo, si no es el caso se busca en la mitad inferior del arreglo, tenemos así el siguiente algoritmo.

Ciclos “while” y “repeat” III

function BINARYSEARCH$(T[1 \ldots n], x)$

```
1  i ← 1
2  j ← n
3  while i < j
4  do k ← (i + j) ÷ 2
5      switch
6          case x < T[k] : j ← k − 1
7          case x = T[k] : i, j ← k
8          case x > T[k] : i ← k + 1
9  return i
```

Ciclos “while” y “repeat” IV

- Recuerde que para analizar el tiempo de ejecución de un ciclo **while** debemos encontrar una función de las variables involucradas cuyo valor decrezca cada vez que se entra al ciclo. En este caso, es natural considerar a $j - i + 1$ al que llamaremos d.

Ciclos “while” y “repeat” IV

- Así, d representa el número de elementos que todavía se tienen que considerar, inicialmente $d = n$. El ciclo termina cuando $i \geq j$, lo cual es equivalente a que $d \leq 1$.

Ciclos “while” y “repeat” IV

- En cada iteración existen tres posibilidades: j toma el valor $k-1$, i toma el valor $k+1$ o tanto i como j toman el valor k.

Ciclos “while” y “repeat” IV

- Sean d y d' respectivamente el valor de $j - i + 1$ antes y después de la iteración bajo consideración, usaremos i, j, i' y j' de manera similar.

Ciclos “while” y “repeat” V

Si $x < T[k]$, la instrucción “$j \leftarrow k - 1$” se ejecuta y así

$$i' = i \text{ y } j' = [(i + j) \div 2] - 1.$$

Ciclos “while” y “repeat” V

Si $x < T[k]$, la instrucción “$j \leftarrow k - 1$” se ejecuta y así

$$i' = i \text{ y } j' = [(i + j) \div 2] - 1.$$

Por lo tanto,

$$d' = j' - i' + 1 = [(i + j) \div 2] - 1 - i + 1 =$$

Ciclos “while” y “repeat” V

Si $x < T[k]$, la instrucción “$j \leftarrow k - 1$” se ejecuta y así

$$i' = i \text{ y } j' = [(i + j) \div 2] - 1.$$

Por lo tanto,

$$d' = j' - i' + 1 = [(i + j) \div 2] - 1 - i + 1 =$$
$$(i + j) \div 2 - i \leq (i + j)/2 - i =$$

Ciclos “while” y “repeat” V

Si $x < T[k]$, la instrucción “$j \leftarrow k - 1$” se ejecuta y así

$$i' = i \text{ y } j' = [(i + j) \div 2] - 1.$$

Por lo tanto,

$$\begin{aligned} d' = j' - i' + 1 = [(i + j) \div 2] - 1 - i + 1 = \\ (i + j) \div 2 - i \leq (i + j)/2 - i = \\ (i + j - 2i)/2 = (j - i)/2 < \end{aligned}$$

Ciclos “while” y “repeat” V

Si $x < T[k]$, la instrucción “$j \leftarrow k - 1$” se ejecuta y así

$$i' = i \text{ y } j' = [(i + j) \div 2] - 1.$$

Por lo tanto,

$$\begin{aligned} d' = j' - i' + 1 = [(i + j) \div 2] - 1 - i + 1 = \\ (i + j) \div 2 - i \leq (i + j)/2 - i = \\ (i + j - 2i)/2 = (j - i)/2 < \\ (j - i)/2 + 1/2 = (j - i + 1)/2 = d/2. \end{aligned}$$

Ciclos “while” y “repeat” VI

Similarmente, si $x > T[k]$, la instrucción “$i \leftarrow k + 1$” se ejecuta y así

$$i' = [(i + j) \div 2] + 1 \text{ y } j' = j.$$

Ciclos “while” y “repeat” VI

Similarmente, si $x > T[k]$, la instrucción “$i \leftarrow k + 1$” se ejecuta y así

$$i' = [(i + j) \div 2] + 1 \text{ y } j' = j.$$

Por lo tanto,

$$d' = j' - i' + 1 = j - [((i + j) \div 2) + 1] + 1 =$$

Ciclos “while” y “repeat” VI

Similarmente, si $x > T[k]$, la instrucción “$i \leftarrow k + 1$” se ejecuta y así

$$i' = [(i + j) \div 2] + 1 \text{ y } j' = j.$$

Por lo tanto,

$$d' = j' - i' + 1 = j - [((i + j) \div 2) + 1] + 1 =$$
$$j - (i + j) \div 2 - 1 + 1 \leq j - (i + j)/2 =$$

Ciclos “while” y “repeat” VI

Similarmente, si $x > T[k]$, la instrucción “$i \leftarrow k + 1$” se ejecuta y así

$$i' = [(i + j) \div 2] + 1 \text{ y } j' = j.$$

Por lo tanto,

$$d' = j' - i' + 1 = j - [((i + j) \div 2) + 1] + 1 =$$
$$j - (i + j) \div 2 - 1 + 1 \leq j - (i + j)/2 =$$
$$(2j - (i + j))/2 = (j - i)/2 <$$

Ciclos “while” y “repeat” VI

Similarmente, si $x > T[k]$, la instrucción “$i \leftarrow k + 1$” se ejecuta y así

$$i' = [(i + j) \div 2] + 1 \text{ y } j' = j.$$

Por lo tanto,

$$d' = j' - i' + 1 = j - [((i + j) \div 2) + 1] + 1 =$$
$$j - (i + j) \div 2 - 1 + 1 \leq j - (i + j)/2 =$$
$$(2j - (i + j))/2 = (j - i)/2 <$$
$$(j - i)/2 + 1/2 = (j - i + 1)/2 = d/2.$$

Ciclos “while” y “repeat” VII

- Finalmente, is $x = T[k]$ entonces i y j toman el mismo valor por tanto $d' = 1$; pero d fue al menos 2 ya que de otro modo no se habría entrado al ciclo (la condición del ciclo es que $i < j$ y $d = j - i + 1$).

Ciclos “while” y “repeat” VII

- Concluimos que $d' = d/2$ en cualquier caso, lo que significa que el valor de d se reduce al menos a la mitad en cada iteración.

Ciclos “while” y “repeat” VII

- Ya que paramos cuando $d \leq 1$, el proceso eventualmente termina, pero ¿cuánto tiempo toma?

Ciclos “while” y “repeat” VIII

- Para determinar una cota superior sobre el tiempo de ejecución de la búsqueda binaria, sea d_l que denota el valor $j - i + 1$ al final de la $l-$ésima iteración para $l \geq 1$ y sea $d_0 = n$.

Ciclos “while” y “repeat” VIII

- Ya que d_{l-1} es el valor de $j - i + 1$ **antes** de iniciar la $l-$ésima iteración y como hemos demostrado que $d_l \leq d_{l-1}/2$ para todo $l \leq 1$, se sigue inmediatamente por inducción matemática que $d_l \leq n/2^l$.

Ciclos “while” y “repeat” VIII

- Pero el ciclo termina cuando $d \leq 1$, lo cual sucede a lo más cuando $l \leq \lceil \lg n \rceil$(recuerde que en los preliminares matemáticos dijimos que $\lg x$ es una abreviación para $\log_2 x$).

Ciclos “while” y “repeat” IX

- Concluimos que se entra al ciclo a lo más $\lceil \lg n \rceil$ veces. Ya que cada iteración tarda un tiempo constante, la búsqueda binaria tarda un tiempo en $O(\log n)$.

Ciclos “while” y “repeat” IX

- Un razonamiento similar nos da como resultado una cota inferior en $\Omega(\log n)$, por tanto la búsqueda binaria tarda un tiempo en $\Theta(\log n)$.

Ciclos “while” y “repeat” X

- El enfoque alternativo para analizar el tiempo de ejecución de la búsqueda binaria empieza muy parecido.

Ciclos “while” y “repeat” X

- La idea es pensar en el ciclo **while** como si estuviera pensado recursivamente en lugar de iterativamente.

Ciclos “while” y “repeat” X

- En cada iteración, reducimos el rango de posibles posiciones en el arreglo para x.

Ciclos “while” y “repeat” XI

- Sea $t(d)$ que denota el tiempo máximo necesario para terminar el ciclo **while** cuando $j - i + 1 \leq d$, esto es cuando existen a lo más d elementos por considerar. Hemos visto que el valor de $j - i + 1$ al menos decrece la mitad en cada iteración.

Ciclos “while” y “repeat” XI

- En términos recursivos esto significa que $t(d)$ es a lo más el tiempo constante b necesario para realizar una vez la iteración, más el tiempo $t(d \div 2)$ suficiente para que la iteración se termine.

Ciclos “while” y “repeat” XI

- Ya que determinar si el ciclo ha terminado, cuando $d = 1$, tarda un tiempo constante c, obtenemos la siguiente recurrencia.

Ciclos “while” y “repeat” XII

$$t(d) = \begin{cases} c & \text{si } d = 1 \\ b + t(d \div 2) & \text{en otro caso} \end{cases}$$

Ciclos “while” y “repeat” XII

$$t(d) = \begin{cases} c & \text{si } d = 1 \\ b + t(d \div 2) & \text{en otro caso} \end{cases}$$

Las técnicas que veremos para resolver recurrencias se aplican para que fácilmente concluyamos que $t(n) \in O(\log n)$.

Uso de un barómetro I

- El análisis de muchos algoritmos se simplifica significativamente cuando una instrucción, o una prueba, se puede identificar como *barómetro*.

Uso de un barómetro I

- Una instrucción barómetro es una que se ejecuta al menos tan frecuentemente como cualquier otra instrucción en el algoritmo.

Uso de un barómetro I

- No hay problema si algunas instrucciones se ejecutan hasta un número constante de veces más a menudo que el barómetro, ya que su contribución se absorbe en la notación asintótica.

Uso de un barómetro I

- Siempre que el tiempo que tarda cada instrucción está acotado por una constante, el tiempo que tarda todo el algoritmo está en el orden exacto del número de veces que la instrucción barómetro se ejecuta.

Uso de un barómetro II

- Esto es útil porque nos permite despreciar los tiempos exactos que tarda cada instrucción.

Uso de un barómetro II

- En particular, evita la necesidad de introducir constantes tales como aquellas que acotan el tiempo que tardan varias operaciones elementales, las cuales no son importantes ya que dependen de la implementación y que son descartadas cuando el resultado final se expresa en términos de la notación asintótica.

Uso de un barómetro III

- Por ejemplo, considere el análisis de FIBITER cuando contamos todas las operaciones aritméticas como de costo unitario.

Uso de un barómetro III

- Vimos que el algoritmo tarda un tiempo acotado superiormente por cn para alguna constante c despreciable, y por lo tanto tarda un tiempo en $\Theta(n)$.

Uso de un barómetro III

- Hubiera sido más simple decir que la instrucción $j \leftarrow i + j$ se puede tomar como barómetro, que obviamente esta instrucción se ejecuta n veces y por tanto el algoritmo tarda un tiempo en $\Theta(n)$.

Uso de un barómetro IV

- Cuando un algoritmo involucra varios ciclos anidados, cualquier instrucción del ciclo más interno usualmente puede tomarse como barómetro.

Uso de un barómetro IV

- No obstante, esto se debe hacer con cuidado porque hay casos donde es necesario tomar en cuenta el control implícito del ciclo.

Uso de un barómetro IV

- Esto sucede tipicamente cuando algunos de los ciclos se ejecutan cero veces, ya que tales ciclos consumen tiempo aunque no se llegue a ejecutar la instrucción barómetro.

Uso de un barómetro IV

- Si esto sucede con frecuencia, el número de veces que la instrucción barómetro se ejecuta puede ser minimizado por el número de veces que se ejecuta el control de los ciclos vacíos y por tanto sería un error considerar tal instrucción más interna como barómetro.

Uso de un barómetro V

- Considere por ejemplo ordenamiento por casillas.

Uso de un barómetro V

- Aquí generalizamos el algoritmo para manejar el caso donde los elementos a ordenar son enteros que se sabe están entre 1 y s en lugar de entre 1 y 10000.

Uso de un barómetro V

- Recuerde que $T[1 \ldots n]$ es el arreglo a ordenar y $U[1 \ldots s]$ es un arreglo construido tal que $U[k]$ contiene el número de veces que el entero k aparece en T.

Uso de un barómetro V

- La fase final del algoritmo reconstruye T en order no decreciente a partir de la información que hay en U.

Uso de un barómetro VI

```
procedure ExtractoDeCasillas()
1  i ← 0
2  for k ← 1 to s
3  do while U[k] ≠ 0
4       do i ← i + 1
5          T[i] ← k
6          U[k] ← U[k] − 1
```

Uso de un barómetro VI

```
procedure ExtractoDeCasillas()
1  i ← 0
2  for k ← 1 to s
3  do while U[k] ≠ 0
4       do i ← i + 1
5          T[i] ← k
6          U[k] ← U[k] − 1
```

- Para analizar el tiempo requerido para este procedimiento, usaremos "$U[k]$" para denotar el valor almacenado *originalmente* en $U[k]$ ya que todos estos valores son definidos a 0 durante el proceso.

Uso de un barómetro VI

```
procedure ExtractoDeCasillas()
1  i ← 0
2  for k ← 1 to s
3  do while U[k] ≠ 0
4       do i ← i + 1
5          T[i] ← k
6          U[k] ← U[k] − 1
```

- Para analizar el tiempo requerido para este procedimiento, usaremos "$U[k]$" para denotar el valor almacenado *originalmente* en $U[k]$ ya que todos estos valores son definidos a 0 durante el proceso.
- Es tentador elegir a cualquiera de las instrucciones en el ciclo más interno como barómetro. Para cada valor de k, estas instrucciones son ejecutadas $U[k]$ veces.

Uso de un barómetro VII

- Por tanto el número total de veces que son ejecutadas es $\sum_{k=1}^{s} U[k]$. Pero esta suma es igual a n, el número de enteros a ordenar, ya que la suma del número de veces que cada elemento aparece da el número total de elementos.

Uso de un barómetro VII

- Si en verdad estas instrucciones sirvieran como barómetro, concluiríamos que este procedimiento tarda un tiempo en el orden exacto de n.

Uso de un barómetro VII

- Un ejemplo simple es suficiente para convencernos que éste no es necesariamente el caso.

Uso de un barómetro VIII

- Suponga que $U[k] = 1$ cuando k es un cuadrado perfecto y $U[k] = 0$ en otro caso. Esto correspondería a ordenar un arreglo T que contiene exactamente una vez cada cuadrado perfecto entre 1 y n^2, usando $s = n^2$ casillas.

Uso de un barómetro VIII

- En este caso, el procedimiento tarda claramente un tiempo en $\Omega(n^2)$ ya que el ciclo más externo se ejecuta s veces.

Uso de un barómetro VIII

- Por lo tanto no puede ser que el tiempo que tarda esté en $\Theta(n)$.

Uso de un barómetro VIII

- Esto demuestra que haber elegido una de las instrucciones en el ciclo más interno como barómetro fue incorrecto.

Uso de un barómetro VIII

- El problema surge ya que sólo podemos despreciar el tiempo utilizado en iniciar y controlar los ciclos siempre y cuando estemos seguros de haber incluido algo que tome en cuenta si el ciclo se ejecuta cero veces.

Uso de un barómetro IX

- El análisis correcto y detallado del procedimiento es como sigue.

Uso de un barómetro IX

- Sea a el tiempo necesario para la prueba $U[k] \neq 0$ cada vez que se pretende ejecutar el ciclo más interno y sea b el tiempo que tarda en una ejecución de las instrucciones en el ciclo más interno, incluyendo la operación de secuenciación para regresar al inicio del ciclo para volver a probar su condición.

Uso de un barómetro IX

- Ejecutar completamente el ciclo más interno para un valor dado de k tarda un tiempo $t_k = (1 + U[k])a + U[k]b$, donde sumamos 1 a $U[k]$ antes de multiplicarlo por a para tomar en cuenta el hecho de que la prueba se realiza cada vez que va a empezar otra iteración y una más para determinar que el ciclo se ha completado.

Uso de un barómetro X

- El asunto crucial es que este tiempo no es cero aún cuando $U[k] = 0$.

Uso de un barómetro X

- El procedimiento completo tarda un tiempo $c + \sum_{k=1}^{s}(d + t_k)$, donde c y d son constantes nuevas que toman en cuenta el tiempo necesario para inciar y controlar el ciclo más externo, respectivamente.

Uso de un barómetro X

- Cuando se simplifica, esta expresión nos da $c + (a + d)s + (a + b)n$.. Concluimos que el procedimiento tarda un tiempo en $\Theta(n + s)$.

Uso de un barómetro X

- Así, el tiempo depende de dos parámetros independientes n y s; no se puede expresar como una función de sólo uno de ellos.

Uso de un barómetro XI

- Es fácil ver que la fase de iniciación de ordenamiento por casillas también tarda un tiempo en $\Theta(n + s)$.

Uso de un barómetro XI

- Esta técnica de ordenamiento tarda un tiempo total en $\Theta(n+s)$ para ordenar n enteros entre 1 y s. Si prefiere, se puede invocar a la regla del máximo para establecer que este tiempo está en $\Theta(\max\{n, s\})$.

Uso de un barómetro XI

- Así, ordenamiento por casillas vale la pena si demostramos que s es suficientemente pequeño comparado con n.

Uso de un barómetro XI

- Por ejemplo, si estamos interesados en el tiempo requerido como una función que dependa solamente del número de elementos a ordenar, esta técnica tiene éxito en un tiempo lineal asombroso si $s \in O(n)$ pero alcanza tiempo cuadrático si $s \in \Theta(n^2)$.

Uso de un barómetro XII

- A pesar de lo anterior, el uso de un barómetro es apropiado para analizar ordenamiento por casillas. El problema es que no se eligió el barómetro adecuado.

Uso de un barómetro XII

- En vez de las instrucciones *dentro* del ciclo interno, se debe elegir como barómetro la *prueba* $U[k] \neq 0$ del ciclo más interno. En verdad, ninguna instrucción en el procedimiento se ejecuta más veces que dicha prueba, lo cual es la definición de un barómetro.

Uso de un barómetro XII

- Es fácil mostrar que $U[k] \neq 0$ se ejecuta exactamente $n + s$ veces, por lo tanto la conclusión correcta sobre el tiempo de ejecución del procedimiento se sigue inmediatamente sin necesidad de introducir constantes sin sentido.

Uso de un barómetro XII

- En conclusión, el uso de un barómetro es una herramienta útil para simplificar el análisis de muchos algoritmos, pero esta técnica debe usarse con cuidado.

Uso de un barómetro XIII

Como otro ejemplo del uso de la técnica del barómetro y del análisis de ciclos anidados, veamos el algoritmo de ordenamiento por selección visto anteriormente.

Uso de un barómetro XIII

```
procedure SELECT(T[1...n])
 1   for i ← 1 to n − 1
 2   do minj ← i
 3        minx ← T[i]
 4        for j ← i + 1 to n
 5        do if T[j] < minx
 6                 then minj ← j
 7                      minx ← T[j]
 8        T[minj] ← T[i]
 9        T[i] ← minx
```

Uso de un barómetro XIV

- No obstante el tiempo que tarda cada iteración del ciclo más interno no es constante, toma más tiempo cuando $T[j] < minx$, este tiempo está acotado superiormente por alguna constante c (que toma en cuenta el control del ciclo más interno).

Uso de un barómetro XIV

- Para cada valor de i, las instrucciones en el ciclo más interno se ejecutan $n - (i + 1) + 1 = n - i$ veces, por lo tanto el tiempo que tarda el ciclo más interno es $t(i) \leq (n - i)c$.

Uso de un barómetro XIV

- El tiempo que tarda la $i-$ésima iteración del ciclo más externo está acotado superiormente por $b + t(i)$ para una constante apropiada b que toma en cuenta las operaciones elementales antes y después del ciclo más interno y el control del ciclo más externo.

Uso de un barómetro XV

Por lo tanto, el tiempo total que tarda el algoritmo está acotado superiormente por:

Uso de un barómetro XV

Por lo tanto, el tiempo total que tarda el algoritmo está acotado superiormente por:

$$\sum_{i=1}^{n-1} b + (n-i)c = \sum_{i=1}^{n-1} (b + cn) - c \sum_{i=1}^{n-1} i$$

Uso de un barómetro XV

Por lo tanto, el tiempo total que tarda el algoritmo está acotado superiormente por:

$$\begin{aligned}\sum_{i=1}^{n-1} b + (n-i)c &= \sum_{i=1}^{n-1}(b+cn) - c\sum_{i=1}^{n-1} i \\ &= (n-1)(b+cn) - cn(n-1)/2\end{aligned}$$

Uso de un barómetro XV

Por lo tanto, el tiempo total que tarda el algoritmo está acotado superiormente por:

$$\begin{aligned}\sum_{i=1}^{n-1} b + (n-i)c &= \sum_{i=1}^{n-1}(b+cn) - c\sum_{i=1}^{n-1} i \\ &= (n-1)(b+cn) - cn(n-1)/2 \\ &= \frac{1}{2}cn^2 + \left(b - \frac{1}{2}c\right)n - b,\end{aligned}$$

Uso de un barómetro XV

Por lo tanto, el tiempo total que tarda el algoritmo está acotado superiormente por:

$$\begin{aligned}\sum_{i=1}^{n-1} b + (n-i)c &= \sum_{i=1}^{n-1}(b+cn) - c\sum_{i=1}^{n-1} i \\ &= (n-1)(b+cn) - cn(n-1)/2 \\ &= \frac{1}{2}cn^2 + \left(b - \frac{1}{2}c\right)n - b,\end{aligned}$$

el cual está en $O(n^2)$. Un razonamiento similar muestra que este tiempo esta también en $\Omega(n^2)$ en todos los casos, y por tanto ordenamiento por selección tarda un tiempo en $\Theta(n^2)$ para ordenar n elementos.

Uso de un barómetro XVI

- El argumento anterior se puede simplificar, obviando la necesidad de introducir constantes explícitas tales como b y c, cuando nos sentimos cómodos con la noción de una instrucción barómetro.

Uso de un barómetro XVI

- Aquí, es natural tomar la prueba del ciclo más interno $T[j] < minx$ como barómetro y contar el número exacto de veces que se ejecuta.

Uso de un barómetro XVI

- Esto es una buena medida del tiempo total del algoritmo ya que ninguno de los ciclos se puede ejecutar cero veces (en cuyo caso el control del ciclo podría consumir más tiempo que el barómetro).

Uso de un barómetro XVII

Se ve fácilmente que el número de veces que la prueba es ejecutada es:

Uso de un barómetro XVII

Se ve fácilmente que el número de veces que la prueba es ejecutada es:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} (n-i)$$

Uso de un barómetro XVII

Se ve fácilmente que el número de veces que la prueba es ejecutada es:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} (n-i)$$

$$= \sum_{k=1}^{n-1} k$$

Uso de un barómetro XVII

Se ve fácilmente que el número de veces que la prueba es ejecutada es:

$$\begin{aligned}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1 &= \sum_{i=1}^{n-1}(n-i) \\ &= \sum_{k=1}^{n-1} k \\ &= n(n-1)/2.\end{aligned}$$

Uso de un barómetro XVII

Se ve fácilmente que el número de veces que la prueba es ejecutada es:

$$\begin{aligned}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1 &= \sum_{i=1}^{n-1}(n-i)\\ &= \sum_{k=1}^{n-1} k\\ &= n(n-1)/2.\end{aligned}$$

Así el número de veces que la instrucción barómetro se ejecuta está en $\Theta(n^2)$, lo cual da automáticamente el tiempo de ejecución de todo el algoritmo.

Uso de un barómetro XVIII

Note que la sumatoria

$$\sum_{i=1}^{n-1}(n-i)$$

Uso de un barómetro XVIII

Note que la sumatoria

$$\sum_{i=1}^{n-1}(n-i)$$

se simplificó haciendo el cambio de variable $k = (n-i)$, para obtener:

$$\sum k$$

Uso de un barómetro XVIII

Note que la sumatoria

$$\sum_{i=1}^{n-1}(n-i)$$

se simplificó haciendo el cambio de variable $k = (n-i)$, para obtener:

$$\sum k$$

falta calcular los límites de esta última sumatoria, para ello es suficiente sustituir los límites de la sumatoria original, así cuando $i = 1$ el límite superior es $k = n-1$ y cuando $i = n-1$ el límite inferior es $k = n-(n-1) = 1$, por ello se obtuvo:

$$\sum_{k=1}^{n-1} k = n(n-1)/2.$$

Uso de un barómetro XIX

Lo cual evita hacer el siguiente desarrollo:

Uso de un barómetro XIX

Lo cual evita hacer el siguiente desarrollo:

$$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$$

.

Uso de un barómetro XIX

Lo cual evita hacer el siguiente desarrollo:

$$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$$

$$= n\sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1} i$$

.

Uso de un barómetro XIX

Lo cual evita hacer el siguiente desarrollo:

$$\begin{aligned}\sum_{i=1}^{n-1}(n-i) &= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \\ &= n\sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1} i \\ &= n(n-1) - n(n-1)/2\end{aligned}$$

.

Uso de un barómetro XIX

Lo cual evita hacer el siguiente desarrollo:

$$\begin{aligned}\sum_{i=1}^{n-1}(n-i) &= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \\ &= n\sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1} i \\ &= n(n-1) - n(n-1)/2 \\ &= n(n-1)/2.\end{aligned}$$

Ejercicios I

1. Obtenga una ecuación de recurrencia para el siguiente algoritmo.

```
procedure HANOI(m, i, j)
1  if m > 0
2     then HANOI(m − 1, i, 6 − i − j)
3          write i "→" j
4          HANOI(m − 1, 6 − i − j, j)
```

2. Obtenga una ecuación de recurrencia para el siguiente algoritmo.

```
procedure DC(n)
1  if n ≤ 1
2     then return n
3  for i ← 1 to 8
4  do DC(n/2)
5  for i ← 1 to n^3
6  do dummy ← 0
```

Ejercicios II

3. Diga el tiempo exacto que toma el siguiente algoritmo, sin utilizar la técnica del barómetro y utilizando la técnica del barómetro.

procedure INSERT($T[1 \ldots n]$)

```
1  for i ← 2 to n
2  do x ← T[i]
3      j ← i − 1
4      while j > 0   and   x < T[j]
5      do T[j + 1] ← T[j]
6          j ← j − 1
7      T[j + 1] ← x
```