Recursividad y Ordenamiento QuickSort

Héctor Ferrada
académico

Instituto de Informática
Universidad Austral de Chile
INFO088 - Estructuras de Datos y Algoritmos

2do Semestre, 2019

Recursividad

- La recursividad es una de las técnicas más usadas para trabajar con estructuras de punteros, como árboles y grafos. Esta consiste en que en el cuerpo de una función o procedimiento $\mathcal{F}$ se efectúa una llamada a $\mathcal{F}$ misma.
- Este paradigma de programación puede entenderse tal como el clásico. Esto es, si consideramos que hay varias *instancias* de una misma función, la recursiva, que se ejecutan una dentro de otra y respetando el orden secuencial dado por sus llamadas.

Recursividad

- La recursividad es una de las técnicas más usadas para trabajar con estructuras de punteros, como árboles y grafos. Esta consiste en que en el cuerpo de una función o procedimiento $\mathcal{F}$ se efectúa una llamada a $\mathcal{F}$ misma.
- Este paradigma de programación puede entenderse tal como el clásico. Esto es, si consideramos que hay varias *instancias* de una misma función, la recursiva, que se ejecutan una dentro de otra y respetando el orden secuencial dado por sus llamadas.
- Ejemplo: factorial de $n \geq 0$, $n! = n(n-1)!$, con $0! = 1! = 1$.

  Invocamos a $f_n =$ **factorial**(n); la cual se define como:
  factorial$(n)\{$
  if $(n < 2)$ return 1;
  return $n \cdot$**factorial**$(n-1)$;
  $\}$

Codifique en C++, pida el ingreso de n y calcule su factorial, repita mientras $n > 0$.

Números de Fibonacci

La serie de Fibonacci se define como: 1,1,2,3,5,8,13,21,...
Tal que: $f_1 = f_2 = 1$; y para $n > 2$, $f_n = f_{n-2} + f_{n-1}$.

Codifique la función: int **fibonacci**(int n) que retorne el n-ésimo número de la serie.

Números de Fibonacci

La serie de Fibonacci se define como: 1,1,2,3,5,8,13,21,...
Tal que: $f_1 = f_2 = 1$; y para $n > 2$, $f_n = f_{n-2} + f_{n-1}$.

Codifique la función: int **fibonacci**(int n) que retorne el n-ésimo número de la serie.
El pseudocódigo es:

fibonacci(n){
  if ($n \leq 2$)
    return 1;
  return fibonacci($n-2$) + fibonacci($n-1$);
}

Note que siempre tiene que haber una **condición de término** de la recursividad, en este caso es $n \leq 2$, en el caso del factorial fue $n < 2$.

Palíndrome

- Un string $S[1 \ldots n]$, o secuencia de símbolos, se dice palindrome si al imprimir S de izquierda a derecha, símbolo a símbolo, se obtiene lo mismo que al hacerlo de derecha a izquierda.
- Ejemplos: "*reconocer*", "*solos*", "*level*", "*amor a roma*".

Diseñe un pseudocódigo, recursivo, para validar si una secuencia de símbolos es palíndrome o no.

Palíndrome

- Un string $S[1 \ldots n]$, o secuencia de símbolos, se dice palindrome si al imprimir S de izquierda a derecha, símbolo a símbolo, se obtiene lo mismo que al hacerlo de derecha a izquierda.
- Ejemplos: "*reconocer*", "*solos*", "*level*", "*amor a roma*".

Diseñe un pseudocódigo, recursivo, para validar si una secuencia de símbolos es palíndrome o no.
Input: un arreglo de caracteres $C[l..r]$, con $l = 1$, $r = n$ al inicio
Output: **True** si C es palindrome, **False** en otro caso.

```
palindrome(C, l, r){
  if (l ≥ r)
    return True;
  if (C[l]! = C[r])
    return False;
  return palindrome(C, l + 1, r − 1);
}
```

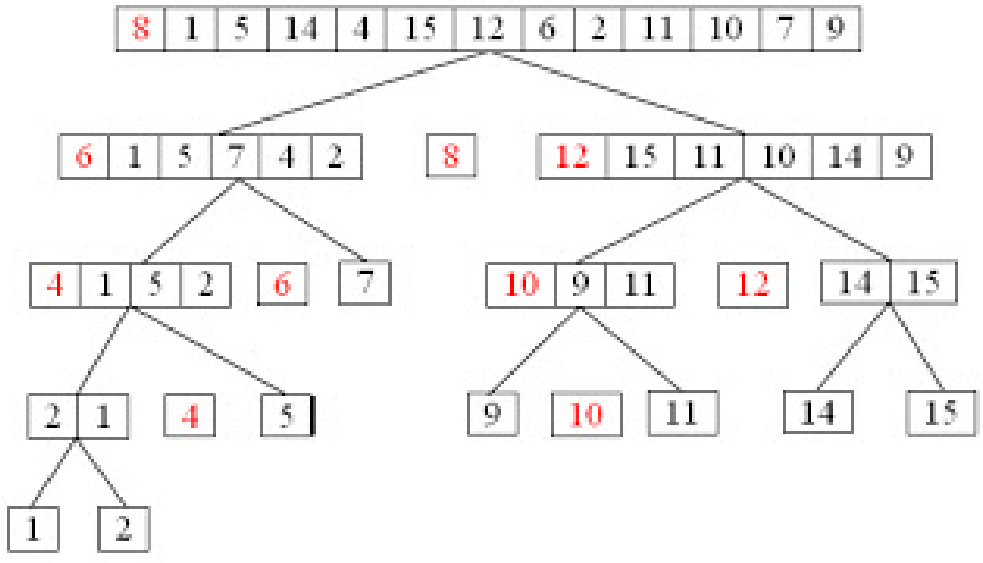
Implemente en C++.

QuickSort

Consigue ordenar $A[1 \ldots n]$ mediante particiones recursivas del arreglo, de tal forma que, luego de cada partición, todos los elementos de la partición de la izquierda son menores que todos los elementos de la partición de la derecha.
El detalle del proceso es el siguiente:

1. Se selecciona un elemento como pivote (pv). En la versión más básica de quicksort $pv = A[l]$
2. Particiona el arreglo en 2: la partición izquierda con elementos menores a pv y la partición de la derecha con los mayores o iguales a pv.
3. Recursivamente se ordenan las dos particiones hasta cubrir todo el intervalo $A[l \ldots r]$.

QuickSort - Ejemplo



Juntando los elementos, el arreglo quedaría ordenado

1	2	4	5	6	7	8	9	10	11	12	14	15

Quicksort - Pseudocódigo

Input: un arreglo $A[l..r]$, con $l = 1$, $r = n$ al inicio
Output: A ordenado ascendentemente

quicksort$(A,\ l,\ r)\{$
  if $l < r\{$
    $p =$partition(A, l, r)
    quicksort$(A, l, p-1)$
    quicksort$(A, p+1, r)$
  $\}$
$\}$

partition$(A,\ l,\ r)\{$
  $p = l$
  $pv = A[p]$
  for$(i = l+1$ to $r)$ do $\{$
    if $(A[i] < pv)$ then $\{$
      swap$(A[i], A[p+1])$
      $p = p+1$
    $\}$
  $\}$
  swap$(A[l], A[p])$
  return$p$
$\}$

Implemente esta versión básica de quickSort.

Ordenando con *qsort* desde la STL

qsort es heredado del C. Ordena la colección utilizando el algoritmo quickSort.
Su complejidad es $O(n \log n)$ en el caso promedio y puede llegar a $O(n^2)$ en el peor caso. Formato:

void *qsort*(**void** ***base**, **size_t** *n*, **size_t** *len*,
int (*compare)(const void*, const void*)); donde:

base: un puntero a la colección a ordenar
n: número de elementos
len: tamaño en bytes de cada elemento
compare: es la función de comparación que devuelve: un entero negativo si el primer argumento es menor que el segundo, un entero positivo si el primer argumento es mayor que el segundo, o 0 si son iguales.
Esta función es llamada repetidamente por qsort para comparar dos elementos; debe obedecer el siguiente prototipo:

int compare(const void *a, const void *b);

Ordenando con *sort* desde la STL

sort es de uso propio del C++. Ordena la colección utilizando un algoritmo híbrido optimizado que mezcla los algoritmos: quickSort, heapSrot e insertionSort.
Su complejidad es $O(n \log n)$ en el peor caso, su formato por defecto es:
template <class RandomAccessIterator> void sort (RandomAccessIterator first, RandomAccessIterator last);
Los iteradores corresponden a las posiciones initial y final de la secuencia a ordenar.

Compile y ejecute el fuente sortArrSTL.cpp.
Codifique un programa que compare los tiempos de su implementación de quickSort con los algoritmos de la STL: qsort y sort.

Ejercicios de Recursividad

1- **Reverso**. Escriba un pseudocódigo de un procedimiento recursivo, que opere in place y sobreescriba el estring en forma reversa. Ej. el reverso de $S =$"*palabra*" es $S^{-1} =$"*arbalap*" Implemente su función recursiva en C++.

Ejercicios de Recursividad

1- **Reverso**. Escriba un pseudocódigo de un procedimiento recursivo, que opere in place y sobreescriba el estring en forma reversa. Ej. el reverso de $S =$"*palabra*" es $S^{-1} =$"*arbalap*" Implemente su función recursiva en C++.

Solución. Recibimos el string como un arreglo $S[1..\ell]$:

reverso$(S, \ell)\{$
    if $(\ell > 1)\{$
        $aux = S[0]$
        $S[0] = S[\ell - 1]$
        $S[\ell - 1] = aux$
        reverso$(S + 1, \ell - 2)$
    $\}$
$\}$

Ejercicios de Recursividad

2- **Reemplazar**. Escriba un pseudocódigo de un procedimiento recursivo, que dado un string $S[1 \ldots n]$ y dos caracteres a, b; sobreescriba S reemplazando todos los caracteres a por b y viceversa.
Ej. reemplazar para $S =$"*xz yyzwy zzxyx yw*", $a =$'*x*', $b =$'*y*'; es
$S' =$"*yz xxzwx zzyxy xw*" Implemente su función recursiva en C++.

Ejercicios de Recursividad

2- **Reemplazar**. Escriba un pseudocódigo de un procedimiento recursivo, que dado un string $S[1 \ldots n]$ y dos caracteres a, b; sobreescriba S reemplazando todos los caracteres a por b y viceversa.
Ej. reemplazar para $S =$"*xz yyzwy zzxyx yw*", $a =$'*x*', $b =$'*y*'; es
$S' =$"*yz xxzwx zzyxy xw*" Implemente su función recursiva en C++.

Solución. Recibimos el string como un arreglo $S[1..n]$ más los dos caracteres a, b, se invoca con con $k = 1$:

```
reemplazar(S, n, a, b, k){
    if (k ≤ n){
        if (S[k] = a)
            S[k] = b
        else if (S[k] = b)
            S[k] = a
        reemplazar(S, n, a, b, k + 1)
    }
}
```

Ejercicios de Recursividad

2- **Reemplazar**. Escriba un pseudocódigo de un procedimiento recursivo, que dado un string $S[1 \ldots n]$ y dos caracteres a, b; sobreescriba S reemplazando todos los caracteres a por b y viceversa.
Ej. reemplazar para $S =$"*xz yyzwy zzxyx yw*", $a =$'x', $b =$'y'; es
$S' =$"*yz xxzwx zzyxy xw*" Implemente su función recursiva en C++.

Solución. Recibimos el string como un arreglo $S[1..n]$ más los dos caracteres a, b, se invoca con con $k = 1$:

reemplazar(S, n, a, b, k){
  if ($k \leq n$){
    if ($S[k] = a$)
      $S[k] = b$
    else if ($S[k] = b$)
      $S[k] = a$
    reemplazar(S, n, a, b, $k + 1$)
  }
}

¿Como podría resolverse este mismo ejercicio si además le piden eliminar todos los espacios en blanco.?

Ejercicios de Recursividad

3- **Suma de Dígitos**. Escriba un pseudocódigo para obtener la suma de todos los dígitos d_i de un entero $num = d_1 d_2 \ldots d_n$ que cumplen con $(d_i \% k) \leq (i \% k)$; para un dígito k. Ej. sumaDigitos($num = 3984592137$, $k = 5$); $\Rightarrow$ retorna sum$= 8 + 4 + 5 + 2 + 1 + 3 = 23$. Implemente en C++.

Ejercicios de Recursividad

3- **Suma de Dígitos**. Escriba un pseudocódigo para obtener la suma de todos los dígitos d_i de un entero $num = d_1 d_2 \ldots d_n$ que cumplen con $(d_i \% k) \leq (i \% k)$; para un dígito k. Ej. sumaDigitos($num = 3984592137$, $k = 5$); $\Rightarrow$ retorna sum$= 8 + 4 + 5 + 2 + 1 + 3 = 23$. Implemente en C++.
Indicación: Conviene empezar desde el último dígito de *num* hasta el primero, incluyendo un parámetro i que indica la posición del dígito que se esta evaluando.

Ejercicios de Recursividad

3- **Suma de Dígitos**. Escriba un pseudocódigo para obtener la suma de todos los dígitos d_i de un entero $num = d_1 d_2 \ldots d_n$ que cumplen con $(d_i \% k) \leq (i \% k)$; para un dígito k. Ej. sumaDigitos($num = 3984592137$, $k = 5$); $\Rightarrow$ retorna sum$= 8 + 4 + 5 + 2 + 1 + 3 = 23$. Implemente en C++. **Indicación**: Conviene empezar desde el último dígito de *num* hasta el primero, incluyendo un parámetro i que indica la posición del dígito que se esta evaluando.

Solución. Recibe el entero *num* y el dígito k, se invoca con con $i = n$:

```
sumaDigitos(num, k, i){
    if (i < 1)
        return 0
    d = num%10
    if ((d%k) ≤ (i%k)){
        return d+sumaDigitos(num/10, k, i − 1)
    return sumaDigitos(num/10, k, i − 1)
}
```

Ejercicios Propuestos

1- Implemente quickSort seleccionando como pivote a un item aleatorio del sub-arreglo no ordenado

2- Proponga un algoritmo eficiente que usando quicksort permita encontrar los k-menores números de un arreglo de enteros creado aleatoriamente. Desde luego, la idea es evitar ordenar todo el arreglo. Describa su algortimo con palabras e implemente en C++.
Note además que no se piden los k menores en orden. Ejemplo sea $A = [33, 64, 21, 78, 11, 19, 43, 75]$, si $k = 3$, entonces los k-menores son: $[33, 21, 11]$

3- Modifique quickSort para que seleccione como pivote a la mediana del subarreglo a ordenar en cada paso, cree una función que le entregue la mediana.

... Ejercicios Propuestos

4- Implemente quickSort seleccionando como pivote a un item aleatorio del sub-arreglo no ordenado

5- Proponga un algoritmo eficiente que usando quicksort permita encontrar los k-menores números de un arreglo de enteros creado aleatoriamente. Desde luego, la idea es evitar ordenar todo el arreglo. Describa su algortimo con palabras e implemente en C++.
Note además que no se piden los k.menores en orden. Ejemplo sea $A = [33, 64, 21, 78, 11, 19, 43, 75]$, si $k = 3$, entonces los k-mnores son: $[33, 21, 11]$

6- Modifique quickSort para que seleccione como pivote a la mediana del subarreglo a ordenar en cada paso, cree una función que le entregue la mediana para un subarreglo.

Quick-sort with Hungarian

- Implemente quickSort tal como se aprecia en el siguiente baile húngaro:
  https://www.youtube.com/watch?v=ywWBy6J5gz8