Taller 4

Carlos Barón - Andrés Cocunubo

14/02/2017

1. Problema 1

   (a) Descripcion del problema

   Escribir un algoritmo para encontrar la subsecencia contigua parcialmente ordenada más larga de una secuencia dada.

   (b) Formalización

   Dada una secuencia S de n elementos que poseen una relacion de orden parcial, encontrar un subsecencia S', donde $|S'|$ sea la mayor.

   - Entradas: Una secuencia S de n elementos S= $<a_n \in T>$ que posea una relacion de orden parcial $\leq$ definida sobre una subsecencia T.

     S= $<a_1,a_2,...,a_n>$
   - Salidas: Un índice i que indica la posición inicial y un j que indica la posición final dentro de una subsecuencia T.
     $i, j, k \in \mathbb{Z} \,|a_i < a_{i+k} < a_j \in S$

   (c) Algoritmos

   - Solución "evidente" o "fuerza bruta"

Algorithm 1

```
procedure LargestOrderedSecuenceEvident(S)
    LargestOrderedSecuenceEvident_Helper(S,1,|S|)
procedure LargestOrderedSecuenceEvident_Helper(S,b,f)
    tam←0 ∧ tam_aux←0
    for i ←b to f do
        tam_aux←1 ∧ j←i+1
        while j<f ∧ S[j-1]≤S[j] do
            tam_aux←tam_aux+1∧ j←j+1
        if tam<tam_aux then
            tam=tam_aux ∧ind←i∧ ind←j-1
    return ind
```

   - Solución "eficiente"

Algorithm 2

```
procedure LongestOrderedSecuence(S)
    LongestOrderedSecuence_Helper(S,1,|S|)
procedure LongestOrderedSecuence_Center(S,i,f,q)
    aux←q
    while q-1≥i ∧ S[q-1]<S[q] do
        q←q-1
    indices←q ∧q←aux
    while q+1≤f ∧S[q+1]∧S[q] do
        q←q+1
    indices←q
    return indices
procedure LongestOrderedSecuence_Helper(S,i,f)
    if i==f then
        indices←i ∧indices←f
        return indices
    else
        q←(f+i)/2
        left←LongestOrderedSecuence_Helper(S,i,q)
        right←LongestOrderedSecuence_Helper(S,q+1,f)
        center←LongestOrderedSecuence_Center(S,i,f,q)
        if right[2]-right[1]≤left[2]-left[1]∧center[2]-center[1]≤left[2]-left[1] then
            return left
        else if left[1]-left[0]≤right[1]-right[0]∧
                center[1]-center[0]≤right[1]-right[0] then
            return right
        else
            return center
```

(d) Invariante

- Solución Evidente:
  Los elementos que se encuentran dentro de las posiciones j y $j-1$ están ordenados.
- Solución Eficiente:
  La subsecuencia entre una posición anterior al pivote $(q-x)$ y una posición posterior $(q+y)$, donde $x \wedge y \in \mathbb{Z}$ está ordenada.

(e) Complejidad

- Solución Evidente:
  Por inspección de código como posee dos ciclo su complejidad es: $O(n^2)$
- Solución Eficiente:
  Según el caso 2 del teorema maestro
  con $a = 2, b = 2, f(n) = O(n)$
  se tiene $O(n) = O(nlog_2(2)log_2^0(n))$ por lo tanto, $T(n) \in \Theta(nlog_2(n))$

(f) Manual de uso

Para ejecutar el programa se debe ejecutar Punto1.py con python3 tanto para la solucion divide y venceras como fuerza bruta
Ejemplo ejecución en ubuntu:

1. Abrir una terminal en linux en el directorio del programa
2. Ejecutar en la terminal "python3 Punto1.py".
3. Se obtendrá la salida del algoritmo eficiente y despueś el algoritmo evidente. En cada salida se puede ver el elemento donde comienza la subsecuencia seguido del elemento donde termina la subsecuencia y el tiempo de ejecución.

2. Problema 2

(a) Descripción del Problema:

Encontrar los elementos minimo y maximo de un arreglo ordenado y rotado.

(b) Formalización:

Dada una secuencia S de n elementos que poseen una relacion de orden parcial, encontrar el elemento a_i que sea $\leq$ a todos los elementos de S y el elemento a_j que sea $>$ a todos los elementos de S

- Entradas: Una secuencia S de n elementos $S = <a_i \in T>$ que posea una relacion de orden parcial $\leq$ definida sobre T y que tenga 1 o r rotaciones.
  $S = <a_1, a_2, ..., a_n> \mid a_1 \leq a_2 \leq a_n \wedge \exists(a_i \nleq a_j)$
- Salidas: Un indice que indica la posición en T del menor y un indice que indica la posición en T del mayor.
  $i, j \in \mathbb{Z} \mid min = a_i \wedge max\ a_j \iff min \in S \wedge max \in S.$

(c) Algoritmos:

- Solución Evidente

Algorithm 3

```
prodecure FindMinMaxElementEvident(A)
    FindMinMaxElementEvidentHelper(A,1,|A|)

prodecure FindMinMaxElementEvidentHelper(A,l,h)
    if A[h] > A[l] then
        R← l
        R← h
        return R
    else
        i← 0
        while i<h ∧ A[i] < A[i+1] do
            i← i+1
        R← i+1
        R← i
        return R
```

- Solución Eficiente

Algorithm 4

```
prodecure FindMinMaxElement(A)
    FindMinMaxElementHelper(A,1,|A|)

procedure FindMinMaxElementHelper(A,l,h)
    m← (h+l)/2
    if A[h] > A[l] then
        R← l
        R← h
        return R
    else if A[m] > A[m+1] then
        R← m+1
        R← m
        return R
    else if A[m-1] > A[m] then
        R← m
        R← m-1
        return R
    else if A[l] > A[m] then
        return FindMinMaxElementHelper(A,l,m)
    else
        return FindMinMaxElementHelper(A,m+1,h)
```

(d) Invariante:

- Solución Evidente:
  La subsecuencia entre 0 e i está ordenanda según el operador $\leq$.
- Solución Eficiente:
  La subsecuencia entre el pivote (m) y el indice correspondiente (l o h) está ordenada, según el operador $\leq$, y rotada.

(e) Complejidad:

- Solución Evidente:
  Por inspección de código, como solo posee un ciclo su complejidad es: $O(n)$
- Solución Eficiente:
  Según el caso 2 del teorema maestro
  con $a = 1$, $b = 2$, $f(n) = O(n)$
  se tiene: $O(1) \equiv \Theta(n^{log_2 1} log_2^0 n)$
  por lo tanto: $T(n) \in \Theta(log_2 n)$

(f) Manual de uso:

Para ejecutar el programa se debe ejecutar Punto2.py con python3

Ejemplo ejecución en ubuntu:

1. Abrir una terminal en linux en el directorio del programa

2. Ejecutar en la terminal "python3 Punto2.py"

3. Se obtendrá la salida del algoritmo eficiente y después del algoritmo evidente. En cada salida se puede ver el menor elemento seguido del mayor elemento y su tiempo de ejecución.