ANÁLISIS Y DISEÑO DE ALGORITMOS PROYECTO

Usted debe hacer un analizador de complejidades para algoritmos recursivos e iterativos, el cual dado un algoritmo en pseudocódigo debe dar como resultado la complejidad en el peor caso, en el mejor caso y en el caso promedio para dicho algoritmo, dicho de otra manera, debe proporcionar el orden de complejidad usando la notación O, Ω y Θ acompañada de cotas fuertes.

Nosotros usaremos las siguientes convenciones:

- Las construcciones cíclicas WHILE, FOR y REPEAT y las construcciones condicionales IF, THEN, ELSE tienen interpretación similar a la que estas tienen en el lenguaje Pascal. Hay una diferencia, en Pascal, el valor de la variable contadora en el loop FOR es indefinido al salir del ciclo, pero aquí, la variable retiene su valor después de salir del FOR. Es decir, inmediatamente después de un loop FOR, la variable de conteo del loop es el valor que primero excedió el límite del loop FOR.
- La sentencia FOR tiene la siguiente sintaxis:

```
for variableContadora  valorInicial to limite do
 begin
       accion 1
       accion 2
       …..
       accion k
 end
```

- La sentencia WHILE tiene la siguiente sintaxis:

```
while (condicion) do
  begin
       accion 1
       accion 2
       …..
       accion k
  end
```

- La sentencia REPEAT tiene la siguiente sintaxis:

```
repeat
       accion 1
       accion 2
       …..
       accion k
until (condicion)
```

- La sentencia IF tiene la siguiente sintaxis:

```
If (condicion) then
   begin
        accion 1
        accion 2
        …..
        accion k
   end
else
   begin
        accion 1
        accion 2
        …..
        accion m
   end
```

- El símbolo "►" indica que el resto de la linea es un comentario .
- La asignación se indica mediante el símbolo "□".
- No se permiten asignaciones múltiples.
- Las variables (tales como i, j etc) son locales a un procedimiento dado. No se usarán variables globales.
- Los elementos de un arreglo son accesados especificando el nombre del arreglo seguido por un indice en corchetes cuadrados. Por ejemplo, A[i] indica el i-ésimo elemento del arreglo A. La notación ".." es usada para indicar un rango de valores dentro de un arreglo ( A[1..j] indica el subarreglo de A consistente de los elementos A[1], A[2], A[3],….,A[j]).
- En cuanto a vectores locales los pueden declarar al inicio del algoritmo, inmediatamente después del begin, y de la misma forma que los definidos como parámetros (nombreVector[tamaño])
- Para especificar el número de elementos en un arreglo A, escribimos length(A) .
- Defina su propia estructura para el manejo de cadenas (gramática,sintaxis etc).
- Defina su propia estructura para el manejo de grafos (gramática,sintaxis etc).
- Las clases se definen antes del algoritmo, especificando el nombre de la clase seguido por llaves en medio de los cuales se detalla la lista de atributos.( Casa {Area color propietario}). Cabe anotar que ningún caracter de puntuación o separado puede hacer parte de ningún nombre de variable, constante o subrutina
- Los datos compuestos son típicamente organizados en objetos. Los objetos deben ser declarados al principio del algoritmo y de la siguiente forma: Clase nombre_del_objeto, los cuales están compuestos de atributos o campos. Un campo en particular se accesa usando el nombre del objeto seguido por "." y luego el nombre del campo.
- Una variable que representa un arreglo o un objeto es tratada como un puntero a los datos representando el arreglo u objeto. Para todos los campos *f* de un objeto *x*, escribiendo *y* □ *x* , causa que *x.f* = *y.f*. En otras palabras *x* y *y* apuntan al mismo objeto después de la asignación *y* □ *x* .

- Algunas veces un puntero no se referirá a objeto alguno. En este caso, le damos el valor especial NULL.
- Los parámetros son pasados a los procedimientos por valor. El procedimiento llamado recibe su propia copia de los parámetros, y si él asigna un valor a un parámetro el cambio no es visto por el procedimiento que llama. Cuando los objetos son pasados, el apuntador a los datos representando al objeto es copiado, pero los campos del objeto no. Por ejemplo, si *x* es un parámetro de un procedimiento llamado la asignación *y* □ *x* dentro del procedimiento llamado no es visible al procedimiento que llama. La asignación *x.f* □ *3*, sin embargo es visible.
- La definición de los parámetros en una subrutina se hace de la siguiente forma :

```
nombre_subrutina(parámetro 1, parámetro 2,….parametro k)
    begin
        accion 1
        accion 2
        …..
        accion k
    end
```

  Si un parámetro es un arreglo se define de la siguiente forma: nombre_arreglo[n]..[m] (los valores dentro de los corchetes son opcionales). Se definen tantos corchetes como dimensiones tenga el arreglo.
  Si un parámetro es un objeto se define de la siguiente forma: Clase nombre_objeto. Cualquier otra clase de parámetro solo se define con el nombre.
  El llamado a una subrutina se hace con CALL seguido por el nombre de la subrutina y entre paréntesis , el nombre de los parámetros.

- Los operadores booleanos son :"and", "or", "not". "and" y "or" son short circuiting. Los operadores short circuiting nos permiten escribir expresiones booleanas tales como *" x <> NULL and x.f = y"* sin preocuparnos acerca de lo que sucede cuando tratamos de evaluar *x.f* cuando *x* es NULL.
- Los valores booleanos son T (true) y F (false).
- Los operadores relacionales son $<, >, \leq, \geq, =, \neq$.
- Los operadores matemáticos son: + (suma), * (multiplicación), / (division real), - (resta),

  mod(residuo), div (división entera), $\lceil\ \rceil$ (techo), $\lfloor\ \rfloor$ (piso).

Entregables:

Debe presentarse el análisis de complejidad en las notaciones O, Ω y Θ del algoritmo "Analizador de Complejidades".

El código fuente debe estar perfectamente documentado.

Consideraciones sobre el informe final sera publicadas posteriormente.