Algoritmos y estructuras de datos III

Trabajo Práctico Nº1

Carla Livorno 424/08 carlalivorno@hotmail.com
Completen!!!

Abril 2010

Índice

Introducción

blah

1. Problema 1

Enunciado del problema...

1.1. Explicación

Poner algo aca...

1.1.1. Análisis de complejidad

Y aca..

1.2. Detalles de la implementación

1.3. Pruebas y Resultados

2. Problema 2

2.1. Explicación

2.1.1. Análisis de complejidad

2.2. Detalles de la implementación

2.3. Pruebas y Resultados

3. Ejercicio 3

3.1. Explicación

Para cada instancia tenemos una lista que contiene para cada programador su horario de ingreso a la empresa y otra con su horario de egreso. Ademas tenemos guardada a cada momento la cantidad maxima de programadores en simultaneo.

Dado que ambas listas se encuentran ordenadas nuestro algoritmo las va recorriendo decidiendo a cada momento si se produce un ingreso o un egreso, es decir, si el horario que sigue en la lista de ingresos es anterior a la de egresos entonces hubo un ingreso, en caso contrario un egreso.

Cuando una persona ingresa a la empresa incrementamos el contador que nos dice la cantidad de programadores que estan simultaneamente en ese horario. Asi cuando se produce un egreso se compara si la cantidad de programadores dentro de la empresa previamente a dicho egreso es mayor a la maxima cantidad de programadores en simultaneo hasta el momento, si es asi actualizamos el maximo. Luego se descuenta el recientemente egresado.

Este procedimiento se repite hasta haber visto todos los ingresos, lo que nos garantiza tener el maximo correspondiente ya que a partir de ese momento solo se produciran egresos.

3.1.1. Análisis de complejidad

Elegimos el modelo uniforme para analizar la complejidad de este algoritmo porque el tamaño de los elementos es acotado y por lo tanto todas las operaciones elementales son de tiempo constante.

Sea n la cantidad de programadores.

```
programadores_en_simultaneo(ingresos, egresos)
    max, tmp, j, k ← 0
    while (j < n)                                          O(n)
        if (ingresos[j] ≤ egresos[k])
            tmp ← tmp + 1
            j ← j + 1
        else
            if (tmp > max)                                 Cada pro-
                max ← tmp
            tmp ← tmp − 1
            k ← k + 1
    if (tmp > max)
        max ← tmp
    return max
```

gramador tiene un ingreso y un egreso, por lo tanto la lista de ingresos y la lista de egresos tienen longitud n. El algoritmo recorre completamente la lista de ingresos. El peor caso es cuando el último ingreso y el último egreso corresponden al mismo programador, ya que para registrar éste último ingreso, la lista de egresos tuvo que ser recorrida hasta encontrarse con el último. Por este motivo, podemos inferir que a lo sumo se realizan $2n - 1$ iteraciones. En cada una de estas iteraciones tenemos un costo constante de operaciones, que no modifican la complejidad en el análisis asintótico. La complejidad algorítmica entonces, es $O(n)$.

3.2. Detalles de la implementación

3.3. Pruebas y Resultados

4. Compilación y ejecución de los programas

5. Conclusiones