Tema 6: Análisis de coste y trazas.

Contenido

1.- Introducción

2.- Análisis de coste

3.- Trazas

1 Introducción

Hasta ahora hemos aprendido a cómo dar solución a un problema con un algoritmo. Dado un problema existen numerosos algoritmos que lo solucionan.

En este tema veremos cómo medir si un algoritmo es mejor que otro analizando el **coste temporal** de ambos (medida de tiempo).

Además veremos cómo evaluar la eficacia de un algoritmo sin ejecutarlo, realizando una **traza** del mismo.

2 Análisis de coste

Coste temporal: medida del tiempo de ejecución de un algoritmo independientemente de la máquina en la que se ejecute. El coste se mide en **pasos**.

Un **paso** es la unidad mínima de tiempo a nivel de programación.

Cada instrucción o estructura del algoritmo tendrá asociado un coste temporal.

El coste temporal asociado al algoritmo completo será la suma de los costes de cada uno de los elementos que lo componen.

2 Análisis de coste

Para calcular el coste seguimos las siguientes reglas:

1) **Estructuras secuenciales**

Asignaciones y operaciones simples: **1 *paso***.

Secuencia de instrucciones: suma de los pesos asociados a cada instrucción por separado

2) **Estructuras de decisión**

Sumamos el coste de la condición mas el coste en el peor de los casos:

Eval. condición + max(bloqueSi,bloqueNo) *pasos*.

2 Análisis de coste

Reglas de cálculo del coste (continuación):

3) **Estructuras de repetición**

Con condición final:

Nºvueltas (cuerpo + condicion) ***pasos***.

Con condición inicial (una evaluación de condición mas):

Nºvueltas (cuerpo + condicion) + condicion ***pasos***.

Con contador: Es necesario tener en cuenta la inicialización y el incremento del contador.

Inicialización + Nºvueltas (cuerpo + condicion + incremento) + condicion ***pasos*****.**

2 Análisis de coste

Ejemplo1:

```
void Funcion1(void) {
    int i,salida;  //La definición de variables no cuenta
    printf("Funcion sencilla\n");   // 1p
    acum=0;                         // 1p
    for(i=-10;i<10;i++){            // 1+20*(1+8+2)+1=222 p
      if( i>=0  && i<100){                          // 3+Max(5,4)=8p
                printf("Positivo\n");               // 1p
                salida=i*2+1;                       // 3 p
                printf("Numero: %d\n",salida);      // 1p
      }else{
                printf("Negativo\n");               // 1p
                salida=i*3;                         // 2p
                printf("Numero: %d\n",salida);      // 1p
      }
}
```

2 Análisis de coste

Ejemplo1:(continuación)

```
    printf("Cuenta atrás:\n");          // 1p
    i=10;                               // 1p
    while(i>=0){                        // 1+11*(3+1)=45 p
        printf("Numero:%d\n",i);        //1p
        i=i-1;                          //2p
    }
    printf("Cuenta alante:\n");         // 1p
    i=0;                                // 1p
    do{                                 // 10*(3+1)=40 p
        printf("Numero:%d\n",i);        //1p
        i=i+1;                          //2p
    }while(i<10);
    return;
}
```

Coste total de la función: 1+1+222+1+1+45+1+1+40=313 pasos.

2 Análisis de coste

Cuando el valor del coste es un número que no depende de ninguna variable se denomina **constante**.

En la mayoría de las ocasiones el coste depende del algún factor externo. En estos casos se calcula en función de las variables implicadas.

Ejemplo: Función factorial

```
int Factorial(int n) {
    int salida,int i;
    salida=1;              // 1p
    for(i=n;i>0;i--){      // 1+ 1+ n*(2+1+2)= 2+5n p
      salida=salida*i;     // 2p
    }
    return(salida);
}
```

Coste total: 3+5n pasos

2 Análisis de coste

Complejidad: Orden de magnitud de la expresión de coste. Puede ser:

Constante: Polinomios de orden 0. Ej: 200p, 3000p, etc...

Lineal: Polinomios de orden 1. Ej: 3n p, 2n+1 p, etc..

Cuadrática: Polinomios de orden 2. Ej: n^2, $5n^2$, $3n^2+n+1$, etc...

Cúbica: Polinomios de orden 3. Ej: $3n^3+2n+2$

….

Para comparar dos algoritmos igualmente efectivos:

- El mejor de ellos será el de menor complejidad
- A igual complejidad, nos quedaremos con el de menor coste.

2 Análisis de coste

Ejemplo: Complejidad del algoritmo de ordenación por intercambio:

```
Void OrdenacionBurbuja(Tipo_base vector[], int tam) {
    Tipo_base aux;
    int i,j;
    for(i=0;i<tam;i++)    //1+tam(1+16tam-14+2)+1 p
      for(j=1;j<tam;j++) // 1+(tam-1)(1+13+2)+1 = 16tam-14p
                if(vector[j-1]>vector[j]){       // 4+9 = 13p
                          aux=vector[j-1];              // 3p
                          vector[j-1]=vector[j];        // 4p
                          vector[j]=aux;                // 2p
                }
    return;
}
```

Coste total de la función: $16tam^2 - 11tam + 2$ pasos

3 Trazas

Traza: Seguimiento paso a paso de la evolución de las variables de un algoritmo para un determinado conjunto de datos de entrada.

Las trazas sirven para evaluar el funcionamiento del algoritmo sin necesidad de ejecutarlo.

Utilizaremos una tabla en la que cada columna representa una variable y cada fila una instrucción.

Cada casilla contiene el valor de cada variable después de ejecutar la instrucción.

Introduciremos únicamente las variables que cambian de valor (**variables significativas**) y las instrucciones que producen cambios (**instrucciones significativas**).

Antes de realizar la traza leeremos el algoritmo y numeraremos las instrucciones significativas del mismo.

3 Trazas

Ejemplo: Traza de la función factorial con n=4:

```
-   int Factorial(int n) {
-     int salida, i;
1     salida=1;
2     for(i=n;i>0;i--){
3                 salida=salida*i;
-     }
-     return(salida);
-   }
```

	i	salida
1	?	1
2	4	1
3	4	4
2	3	4
3	3	12
2	2	12
3	2	24
2	1	24
3	1	24
2	0	24

3 Trazas

Ejemplo: Traza de la siguiente función con a=2 y b=3

```
-   void FuncionPrueba(int a, int b) {
-     int i,j,valor;
1     for(i=0;i<a;i++){
-                 printf("Vuelta %d\n",i);
2                 for(j=0;j<b;j++){
3                           valor=(i+j)*2+1;
-                           printf("para i=%d, j=%d vale %d\n",i,j,valor);
-                 }
-     }
-     return;
-   }
```

3 Trazas

Ejemplo (continuación): Traza de la función con a=2 y b=3

	i	j	valor
1	0	?	?
2	0	0	?
3	0	0	1
2	0	1	1
3	0	1	3
2	0	2	3
3	0	2	5
2	0	3	5

	i	j	valor
1	1	3	5
2	1	0	5
3	1	0	3
2	1	1	3
3	1	1	5
2	1	2	3
3	1	2	7
2	1	3	7
1	2	3	7

Como vemos, para poder realizar una traza correctamente, es necesario conocer de forma exacta el comportamiento de cada estructura.