Universidad autónoma de baja california

Ingeniería en computación

Algoritmos y estructuras de datos

Practica 1

Maestro: Thelma Violeta Ocegueda Miramontes

Erik Garcia Chávez 01275863

Miércoles 28 agosto del 2024

Complejidad del algoritmo que se realizó para hacer la intersección, debemos de tomar en cuenta las declaraciones, las asignaciones de valores a variables, asi como lo que más puede pesar en la complejidad del algoritmo en cuestión del tiempo que son los ciclos.

Para esa función tenemso la siguiente función que nos ayuda a realizar esa operación de toeria de conjuntos, que nos dice que se debe de mostrar solo aquellos elementos de 2 arreglos que esten en ambos arreglos.

Como se va a calcular la complejidad de 1 algoritmo, para eso usamos unas condiciones en las cuales tenemos que:

La declaración de variables es igual a 1

La inicalizcion o asignación de varibales es igual a 1

Un ciclo por ejemplo el ciclo for por lo general puede tender a ser 2n+1. Esto pasa en un ciclo como el de nustro algoritmo normal, en el **cual se inicializa la vairbale de control, que es 1**, se relaiza las **compraciones que es n+1** porque se vana a realizar n compraciones +1 por la lógica de la programación que al terminal de hacer las compraciones por lo genral se realiza otra mas que es la que nos va a sacar del ciclo y **n que son los incrementos** que va a tener nuestro ciclo

Con esta idea iremos relaizando las asignaciones de las acciones de nustro código, el cosigo que este al mismo nivel se suma y los que esten tabulados, como las instrucciones que están dentro de un ciclo se multiplican.

Código:

```
void interseccion(int vec1[], int vec2[],int n, int m){

    int i,j,valVec,contVec,*final; ——→ 1

    final=(int*)malloc(sizeof(int)); ——→ 1

      contVec=0; ——→ 1

    for(i=0;i<n;i++){ ——→ 1 + n+2 + 1 = 2n+1

        valVec=vec1[i]; ——→ 1

        for(j=0;j<m;j++){ ——→ 1 + m+2 + m = 2m+1
```

```
            if(valVec==vec2[j]){   ⟶ 1 (en esta linea solo se esta
comparando por lo que es 1)

                for(int k=0;k<=contVec;k++){  ⟶ 1 + k+1 + k = 2k+1

                    if(contVec==0){ ⟶ 1
                        *(final+contVec)=valVec;  ⟶ 1
                        contVec++;  ⟶ 1
                    }

                    else if(valVec != *(final+k)){⟶  1
                        *(final+contVec)=valVec;  ⟶ 1
                        printf("%d - ", valVec); ⟶ 1
                    }

                }

            }
        }

    }
}
```

Este algoritmo se puede ver algo enredoso, pero el nivel de complejidad no es tan grande. Porque asiganmos nuevas variables a cada iteración, porque cada una responde a varibales de control diferentes, si dependindnan de una, se estableceria, pero en este caso no, por lo que el nuvel de complejidad es mucho mas simple. Por lo que tenemos como función lo siguiente:

1+1+1+(2n+1(1+(2m+1 (1+(2k+1(1(1+1) + 1(1+1)))))) = 9+ 2n + 2m + 2k

Para este tipo de análisis ignoramos las contantes por lo que al final nos quedaría

T(nmk) que nos dice que será de complejidad lineal, será constante, para n cantidad de números en nuestro vector, el tiempo será similar para todos.

Algoritmos de diferencia de conjuntos tanto comparando el arreglo 1 con el arreglo 2 como viceversa, la diferencia nos dice que se mostraran los números o elementos dentro de los conjuntos solo aquellos que no se encuentren en el otro conjunto. este algoritmo tendrá una complejidad parecida al anterior, dado que son muy similares en la construcción.

Código:

```
void diferencia_conjuntos(int vec1[],int vec2[],int n,int m){


    int i,j, valVec, validacion; ⟶ 1
    int contador=0; ⟶ 1
    for(i=0;i<n;i++){  ⟶ 1 +n+1+n= 2n+1
        validacion=0; ⟶ 1
        valVec=vec1[i]; ⟶ 1
        for(j=0;j<m;j++){  ⟶  2m+1

                if(valVec==vec2[j]){⟶  1
                    validacion=0;  ⟶ 1 esta asignación se multiplica con
repecto a la compracion del if, dado que se encuentra anidado.
                    break;

                }
                else{
                    validacion=1;
                }
        }

        if(validacion){
            printf("%d -",valVec );
            contador++;
        }

    }
    if(contador==0){
        puts("todos los elementos de los 2 arreglos son iguales.");
    }
}
```

El algoritmo anterior tiene una complejidad de:

T(n)= 1+ 1+(2n+1(1+1+(2m+1(1(1+1) +1) +1(1+1))) + 1(1)) = 11 + 2n+2m

Ignorando las contantes tenemos que T(nm) lo cual nos deja de la misma manera en una complejidad lineal, por lo que por tantos elementos entres el tiempo será contante,