	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación
salas A y B

Profesor: Adrian Ulises Mercado

Asignatura: Estructura de datos y Algoritmos I

Grupo: 13

No de Práctica(s): ANALISIS DE RECURSIVIDAD

Integrante(s): Martínez Jacques Ricardo

No. de Equipo de cómputo empleado:

No. de Lista o Brigada: Brigada 5

Semestre: 2020-2

Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____

ANALISIS DE RECURSIVIDAD.

INTRODUCCION

Veremos un análisis de algoritmos recursivos, asi mismo la definición de recursividad, algunos ejemplos de su utilización, y una solución de problemas con recursividad.

DESARROLLO

¿Qué es la recursividad?
Es el hecho que un algoritmo hace cuando se autollama, o se llama a si mismo, existen dos tipos de recursividad en programación.

1. Directa o también llamada simple, multiple:

Es la más común, se da cuando una función se llama a sí misma una o varias veces.

2. Indirecta:

Se da cuando una función es llamada de manera indirecta, es decir, por medio de otra función.

¿Qué características tiene una función recursiva?

1. Tener el caso base o condición de donde comenzara el algortimo
2. El paso recursivo.
3. Cada ciclo que se repita, debe ir acercándose más a la solución buscada, haciendo del problema pequeños problemas.

Ejemplos:

- Serie de Fibonacci
- Factorial de un numero
- Recorrido de un árbol
- Potencia de n numero a n numero

¿Tiene ventajas y desventajas la recursividad?

Como ventajas se proponen:

- Se da una solución natural y aplicable
- Soluciones iterativas que son mas difíciles de implementar
- Resuelve problemas que resultan complejos, ya que subdivide el problema

Como desventajas se porponen:

- Puede que la recursividad conlleve a un ciclo infinito y no resolver el problema.
- Es difícil de programar, depende del problema a solucionar

Conocer la Ecuacion Recursiva.

- ✓ Asociar una fucion o método F a un tiempo de corrida T(n) que desconocemos
- ✓ Establecer la ecuación de recurrencia que es la que representa cual es la complejidad de un algoritmo recursivo.
- ✓ El cálculo de la T(n) puede depender de :
  El caso base o paso base , si es valor entero o el valor inicial n es muy pequeño
  El paso recursivo; que puede ir decrementando el valor de n en cada llamada de T, la n es para todos los demás valores enteros positivos superiores al del caso base.
- ✓ Ejemplo del factorial

Ejemplos.
Torres de hanoi

```c
#include <stdio.h>
#include <stdlib.h>

void hanoi(int n, char origen, char destino, char aux){
    if(n ==1){
        printf("mover disco %d desde %c hasta %c\n", n, origen, destino);
    }else{
        hanoi(n-1, origen, aux, destino);//movern-1 discos al auxiliar
        printf("Mover disco %d de %c hasta %c\n", n, origen, destino);
        hanoi(n-1, aux, destino, origen);
    }
}

int main (int argc, char **argv){
    if(argc < 2){
        print("Falta el numero discos\n");
        return -1;
    }
    int n = atoi(argv[1]);
    hanoi (n, '1', '3', '2');
    reuturn 0;
}
```

Analisis de la ecuación
T(n) es 1 para n=1, 2 T(n-1) +1 para n>1

Probando para algunos valores de n.

- ✓ T(2) =2T(1)+1= 2(1) +1= 3
- ✓ T(3) =2T(2)+1= 2(3) +1= 7
- ✓ T(4) =2T(3)+1= 2(7) +1= 15
- ✓ T(5) =2T(4)+1= 2(15) +1= 31

Resolviendo las ecuaciones de recursividad
T(n)= 2T(n-1) +1
T(n)= 2(2T(n-2)+1)= 4 T(n-2)+2+1
T(n)= 4(2T(n-3)+1)= 8T(n-3)+2+1
T(n)= 8(2T(n-4)+1)= 16T(n-4)+2+1

Haciendolo k veces poemos hacer la ecuación de una sumatoria, quedando de esta forma

$$T(n) = 2^{n-1}T(1) + \sum_{i=0}^{n-2} 2^i = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1$$

Para concluir Resumiendo

- La recursividad puede resolver problemas complejos de manera simple
- Para solucionar ecuaciones recursivas hay que descomponer la recursión para encontrar patrones y poder obtener su orden de complejidad
- Tiene una alta complejidad(MALO).