**“PROGRAMACION II – RECURSIVIDAD”**

ALUMNO: Ronny Alexander Fernández Aguilar

REGISTRO: 200921819

CARRERA: INGENIERIA EN SISTEMAS

DOCENTE: Ing. Mary Dunnia López Negretty

1. **Explique el algoritmo del Backtrack ¿Qué significan los niveles de recursividad?**  
   R.- El algoritmo del Backtrack divide la solución en pasos, en cada uno de los cuales hay una serie de opciones que ha que probar de forma sistemática. En cada paso se busca una posibilidad o solución o solución aceptable.  
     
   - *Si se encuentra se pasa a decidir el paso siguiente.*- *Si no se encuentra una solución aceptable, se retrocede hasta la última solución aceptable encontrada y se elige una opción distinta a la anterior. La recursividad se utiliza para poder retroceder hasta encontrar una solución aceptable.*  
   - Ejemplos: *Juegos de tablero, laberintos, etc.*  
   La Recursividad es una técnica de programación para resolver problemas en una idea de ‘atajo’, dicho atajo está basado en un algoritmo en términos de una llamada a sí mismo. Donde la llamada a sí mismo, se conoce como llamada recursiva o *recurrente*.  
     
   - Directa: Se habla de recursión directa cuando la función se llama a sí misma.

* Indirecta: Cuando se definen una serie de subprogramas usándose unos a otros.
* Ahora bien hay que decir que la eficiencia depende de:
  + El número de nodos del árbol de búsqueda que se visitan para conseguir la solución →v(n).
  + El trabajo realizado en cada nodo, esto es, el coste de la función de solución completa o ver si la solución es aceptable hasta el momento. Este coste lo podemos expresar como →p(n), ya que generalmente será polinómico.
  + El coste en general será: O(p(n)v(n)), este coste será exponencial en el peor caso.
* Para conseguir mejoras en los costes se suele recurrir a la poda del árbol de búsqueda, lo cuál se hace marcando los caminos que ya se han estudiado y los no prometedores como cerrados con lo cuál el algoritmo no perderá tiempo con los nodos que estén dentro de estos caminos. También se podrían crear predicados acotadores reduzcan mucho el número de nodos generados, si el predicado acotador sólo dejara un nodo el algoritmo se convertiría en voraz y su coste bajaría a O(p(n)n). Aunque normalmente los costes más bajos que se pueden conseguir son el orden de O(p(n)2n).
* En conclusión podemos decir que debido al coste creado en tiempo y memoria (por la pila recursiva) los algoritmos de vuelta atrás no son todo lo eficientes que deberían, y debemos dejarlos para resolver parte de otros problemas o problemas reducidos. Aún así la gran ventaja que tienen es que si hay solución la encontrarán.

**//Construya una función recursiva que calcule el valor de la combinatoria de 2 números enteros.**

**#include <conio.h>**

**#include <stdio.h>**

**#include <iostream.h>**

**#include <math.h>**

**#pragma argsused**

**//---combinar dos numeros------**

int combi (int m,int n ){

if (n==0) {

return 1;

}else{

int c =combi(m-1,n-1);

c=(c\*m)/n;

return c;

}

}

**/\*Dado un número entero positivo o nulo (en base decimal), construya un algoritmo recursivo que tenga por resultado otro número entero cuyo valor sea la representación en binario (en base 2) del valor dado.  
//----convertir a binario-------\*/**

int convertir (int n,int b ){

if (n<b) {

return n;

}else {

int x =convertir( n / b,b );

x=x \*10+(n%b) ;

return x;

} }

**/\* Escriba un algoritmo recursivo. Dado N muestre la serie y realice el sumatorio : (1 + 1/4 + 1/9 + 1/16 …)  
//generar la serie :=1+ 1/4 + 1/9 + 1/16\*/**

float generar\_serie(int n, int pos)

{

float sum;

if (n==0) {

cout<<" ";

} else {

if (n>1 ) {

if (pos==1 ) {

cout<<"1"<<" + ";

} else {

cout<<"1/"<<pos\*pos<<" + ";

}

} else {

cout<<"1/"<<pos\*pos;

}

sum=(n\*n);

sum=1/sum;

sum=sum+generar\_serie(n-1,pos+1);

} return sum;

}

**/\*Escriba un algoritmo recursivo para generar la siguiente matriz de dimensiones NxM**

Donde N : Es Impar , M: Es par y es siempre ((N/2)+1)

0 0 0 0 0 0 0

1 1 1 1 1

2 2 2

3

\*/

void recursivo(int mat[][10],int nfil,int ncol,int f,int c,int k,int x ) {

if (c>ncol) {

if (c>ncol-x) {

printf("\n");

printf("\n");

f++;

x=x+1;

c=0;

k++;

} }

if (c<ncol-x) {

if (f<nfil) {

if (c<x) {

printf ("%d ",mat[f][c]=0);

recursivo (mat,nfil,ncol,f,c+1,k,x);

} else {

Printf ("%d ",mat[f][c]=k);

recursivo (mat,nfil,ncol,f,c+1,k,x);

} }

} else {

if (c!=ncol)

printf ("%d ",mat[f][c]=0);

recursivo (mat,nfil,ncol,f,c+1,k,x);

} }

**// generar la serie "1^n+2^(n-1)+3^(n-2)+n^1"**

int genser(int n, double pos)

{

double sum;

if (n==0) {

return 0;

} else {

sum=pow(pos,n);

if (n>1) {

cout<<pos<<"^"<<n<<" + ";

} else {

cout<<pos<<"^"<<n<<"\n\n";

}

sum= sum + genser(n-1,pos+1);

}

return sum ; }

**//LLAMADAS PARA LAS FUNCIONES**

**int \_tmain()** {

int m,n;

//mostrar la combinatoria de dos numeros

cout<<"introducir dato1 de combi:";

cin>>m;

cout<<"introducir dato2 de combi:";

cin>>n;

cout<<"el resultado de combi es:";

cout<<combi(m,n)<< endl;

printf("\n");

// mostrar numero en binario

int b;

printf("escriba un numero para mostrar en binario:");

scanf("%d",&n);

printf("escriba la base :");

scanf("%d",&b);

if (b<10) {

n=convertir(n,b);

printf(" el numero en binario es:%d",n);

}

//mostrar la serie 1 + 1/4 + 1/9 + 1/16

int pos=0;

printf("\n");

printf("\n");

cout<<" introduzca el valor de n entre 1 y 10 para la serie:";

cin>>n;

printf("\n");

cout<<"\n\n la sumatoria de la serie es es:"<<generar\_serie(n,pos+1);

//mostrar la Matriz

printf("\n");

printf("\n");

printf("\n");

pos=0;

int mat[10][10];

int k=0;

int f=0;

int c=0;

int x=0;

int ncol=19;

cout<<" introduzca un valor impar de columnas para la matriz:";

cin>>ncol;

int nfil=(ncol/2)+1 ;

printf("\n");

printf("el valor de la matriz es :");

printf("\n");

printf("\n");

recursivo(mat,nfil,ncol,f,c,k,x);

printf("\n");

printf("\n");

cout<<"introduzca el valor de n entre 1 y 10 :";

cin>>n;

int s=genser(n,pos+1);

cout<<"la sumatoria es: "<<s;

printf("\n");

**getch();**

}