Estructuras de Datos y Algoritmos 1

Solución - Práctico 2

Tema: Recursividad

Ejercicios:

Recursión Básica

4) Diseñar una rutina listar1An(n) que liste los números del 1 al n. ¿Podría lograr que los liste en orden inverso? .Calcule su orden. ¿Cómo compara esta implementación con la equivalente y más obvia en base a una estructura de control repetitiva?

```
//Resuelve el Ejercicio 4 del Practico 2

void listar1aN(int);
void listarNa1(int);

void ejercicio4(){

                listar1aN(10);
                cout<<"\n----------------------------------------\n ";
                listarNa1(10);
}

void listar1aN(int n){
                if (n > 0){
                 listar1aN(n - 1);
                 cout<< n <<" - ";
                }
}

void listarNa1(int n){
                if (n > 0){
                 cout<< n <<" - ";
                 listar1aN(n - 1);
                }
}
```

7)

a) Defina una función recursiva int busco(VEC,izq,der N,dato) que retorna la posición ocupada por dato sobre el array VEC entre las posiciones izq y der de vec o –1 si no se encuentra el dato.

```
//Solución lineal
int bLineal(int v[], int e, int inf, int sup){
      if (inf <= sup){
            if (v[inf] == e) return inf;
```

```
            else return bLineal(v, e, inf + 1, sup);
      }
      return -1;
}
//Solución que aplica dividir para conquistar
int bDivConq (int v[], int e, int inf, int sup){
      int mid, esta1, esta2;
      if (inf <= sup){
            mid = (inf + sup)/2;
            if (v[mid] == e) return mid;
            else{
                        esta1 = bLineal(v, e, inf , mid - 1);
                        esta2 = bLineal(v, e, mid + 1, sup);
                        if (esta1 != -1) return esta1;
                        if (esta2 != -1) return esta2;
                        return -1;
            }
      }
      return -1;
}
```

b) }Calcule el orden de la implementación de la parte a) Una forma de resolver el problema podría ser partir el problema en un subproblema izquierdo y uno derecho y buscar en la parte derecha y en la parte izquierda. Resuelva el problema de esta manera y calcule su orden.

$$\left.\begin{array}{l} T(0) = 0 \\ T(n) = T(n-1) + k \end{array}\right\} \quad \underline{O(n)}$$

c) ¿Ayudaría que los elementos de VEC se encontraran ordenados?

En la búsqueda que implementa la estrategia dividir para conquistar se puede desechar en cada comparación la mitad de los elementos a comparar.

d) Implemente una versión mejorada en el supuesto de que los elementos de VEC se encuentran ordenados y calcule su orden.

```
int bBiparticion(int v[], int e, int inf, int sup){
      int mid;

      if (inf > sup) return -1;
      mid = (inf + sup)/2;
      if (v[mid] == e) return mid;
      if (v[mid] > e) return bBiparticion(v, e, inf, mid-1);
      else
            return bBiparticion(v, e, mid+1, sup);
}
```

```
T(1) = 1
T(n) = T(n/2) + k      } O(lg n)
```

9) Idem a 8 (Desarrollar una función que reciba un array y qe retorne su mínimo elemento) pero que retorne la posición del mínimo.

```
int posMinimo (int v[], int inf, int sup){
      int min;

      if (inf == sup) return inf;
      if((v[min = posMinimo(v, inf+1, sup)]) < v[inf])
            return min;
      else
            return inf;
}
```

10) Usando 9 implemente una función recursiva que ordene un array.

```
int posMinimo (int [], int, int);
int intercambio(int*, int*);
void ordenar(int v[], int inf, int sup){
      int posMin;
      if(inf < sup){
            posMin = posMinimo(v, inf, sup);
```

```
            intercambio(&v[inf], &v[posMin]);
            ordenar(v, inf+1, sup);
      }
}

void intercambio(int *a, int *b){
      int aux;
      aux = *a; *a = *b; *b = aux;
}
```

Recursión Avanzada

12) Implementar recursivamente el cálculo de a^b con a real y b entero. ¿Puede mejorar la cantidad de multiplicaciones necesarias? Realizar el diagrama de llamadas para pot(a,b).

```
float potDC(float b, int e){
      float res;

      if (e == 0) return 1;
      if (e == 1) return b;
      if (e < 0) return potDC(1/b, -1*e);
      if (e%2 == 0){
            res = potDC(b, e/2);
            return res * res;
      }
      return b * potDC(b, e - 1);
}
```

13) Eratóstenes construyó una criba para determinar los números primos del 2 al 100 de la siguiente manera:

a) Sobre una madera escribió los números.

b) Recorría la misma secuencialmente buscando el primer número no perforado. Al encontrarlo, perforaba todos sus múltiplos y repetía el paso anterior a partir del siguiente, hasta terminar.

Implemente una versión estrictamente recursiva (sin usar estructuras de control repetitivas) de este algoritmo.

```
void criba(int [], int);
void ejercicio13(){
	int tabla[M], i;
	char a;

	for(i = 0; i < M; tabla[i++] = NO_MARCADO);
	criba(tabla, 2);

	cout<<"------------- Los primos entre 1 y "<<M<<" -------------
\n\n";
	for(i = 1; i < M; i++){
		if(tabla[i] == NO_MARCADO)
			cout<< i <<" - ";
	}
	cout<<"\nIngrese un caracter para continuar..."; cin>>a;
	cout<<"\n\n";
}

void marcar (int [], int, int);

void criba(int t[], int i){

	if (i>=M) return;
	if (i> (int)sqrt(M)) return;
	if(t[i] == NO_MARCADO)
		marcar(t, i, 2);
	criba(t, i + 1);
}
void marcar (int t[], int i, int mult){
	/*for (int j = mult; i * j < M;j++)
		t[i*j] = MARCADO;*/

	if (i*mult>M) return;
	else{
		t[i*mult] = MARCADO;
		marcar(t, i, mult + 1);
	}
```

```
}
```

15) Una forma de determinar si un número n es múltiplo de 9 se basa en sumar sus cifras. Por ejemplo

N = 12345678901234567890123456756789432 1
1+2+3+4+5+6+7+8+9+0+1+2+3+4+5+6+7+8+9+4+3+2+1=100
1+0+0 = 1 no es un múltiplo de 9.

Programar una función booleana bool mult9 (int n) que indique si n es o no un múltiplo de 9.

```
long sumaCif(long);


int esMult9(long n){


      if(n<9) return 0;
      if(n ==9) return 1;
      return esMult9(sumaCif(n));
}


long sumaCif (long n){
      if (n>= 0 && n <= 9) return n;
      else
            return (n%10) + sumaCif(n/10);
}
```

17) Desarrollar una función int ISLAS (MAPA,M,N) que recibe un mapa representado por una matriz de MxN donde hay un 0 en los lugares en que hay agua y un uno donde hay tierra y contesta la cantidad de ISLAS que contiene. Una isla es un conjunto de unos adyacentes (ortogonal u oblicuamente).

0	0	0	**1**	**1**	0	0	0	**1**	0
0	0	**1**	0	**1**	0	0	0	**1**	**1**
0	**1**	0	0	**1**	**1**	**1**	0	0	0
0	**1**	0	0	0	0	0	**1**	0	0
0	0	**1**	0	**1**	0	**1**	0	0	**1**
0	0	**1**	0	0	0	**1**	0	0	**1**
0	0	**1**	0	0	0	**1**	0	0	**1**
0	0	0	**1**	**1**	**1**	0	0	**1**	0
0	0	0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	**1**	0	0	0
0	0	**1**	**1**	0	0	**1**	0	**1**	0
0	0	0	0	0	0	0	0	0	**1**

Por ejemplo en el "mapa" anterior la respuesta sería 6.

```
#define H 20
#define V 20
#define TIERRA 1
#define AGUA 0
#define VISITADO 2
int mapa[H][V], vf[8], vc[8];

void muestoMapa();
void inicioM(int [][V]);
void inicioV(int [], int []);
int cuentoIslas();

void ejercicio17(){
      int cant = 0;
      char a;

      inicioM(mapa);
      inicioV(vf, vc);
      cout<<"\nIngrese un caracter para continuar..."; cin>>a;
      cout<<"\n\n";
      cant = cuentoIslas();
      cout<<"La cantidad de islas encontradas es: "<<cant<<"\n";
      muestoMapa();
      cout<<"\nIngrese un caracter para continuar..."; cin>>a;
      cout<<"\n\n";
}

void borroIsla(int, int);

int cuentoIslas(){
      int c = 0, i, j;

      for (i = 0; i< H; i++)
            for (j = 0; j< H; j++)
                  if (mapa[i][j] == TIERRA){
                        mapa[i][j] = VISITADO;
```

```
                        c++;
                        borroIsla(i, j);
                  }
      return c;
}

int dentro(int, int);
void borroIsla(int i, int j){
      int u, v, k;

      for (k = 0; k<8; k++){
            u = vf[k] + i;
            v = vc[k] + j;
            if (dentro(u, v) && mapa[u][v] == TIERRA){
                  mapa[u][v] = VISITADO;
                  borroIsla(u, v);
            }
      }
}
int dentro(int h, int v){
      return (h>= 0 && h < H && v >= 0 && v < V);
}
void inicioV(int f[], int c[]){
      f[0] = -1;  c[0] = -1;
      f[1] = -1;  c[1] = 0;
      f[2] = -1;  c[2] = 1;
      f[3] = 0;   c[3] = 1;
      f[4] = 1;   c[4] = 1;
      f[5] = 1;   c[5] = 0;
      f[6] = 1;   c[6] = -1;
      f[7] = 0;   c[7] = -1;
}
void inicioM(int mapa[][V]){
      int i, j;
      for (i = 0; i< H; i++)
            for (j = 0; j< V; j++)
                  mapa[i][j] = AGUA;
```

```
mapa[3][0] = TIERRA;mapa[4][0] = TIERRA;mapa[8][0] = TIERRA;
mapa[2][1] = TIERRA;mapa[4][1] = TIERRA;mapa[8][1] = TIERRA;
mapa[9][1] = TIERRA;
mapa[1][2] = TIERRA;mapa[4][2] = TIERRA;mapa[5][2] = TIERRA;
mapa[6][2] = TIERRA;
mapa[1][3] = TIERRA;mapa[7][3] = TIERRA;
mapa[2][4] = TIERRA;mapa[4][4] = TIERRA;mapa[6][4] = TIERRA;
mapa[9][4] = TIERRA;
mapa[2][5] = TIERRA;mapa[6][5] = TIERRA;mapa[9][5] = TIERRA;
mapa[2][6] = TIERRA;mapa[6][6] = TIERRA;mapa[9][6] = TIERRA;
mapa[3][7] = TIERRA;mapa[4][7] = TIERRA;mapa[5][7] = TIERRA;
mapa[8][7] = TIERRA;
mapa[7][8] = TIERRA;
mapa[6][9] = TIERRA;
mapa[2][10] = TIERRA;mapa[3][10] = TIERRA;mapa[6][10] = TIERRA;
mapa[8][10] = TIERRA;
mapa[9][11] = TIERRA;
muestoMapa();
}
void muestoMapa(){
      int i, j;
      for (i = 0; i< H; i++){
            for (j = 0; j< V; j++)
                  cout<<mapa[i][j];
            cout<<"\n";
      }
}
```

18) Una antigua leyenda cuenta que en un monasterio de Hanoi, los monjes disponen de 64 discos perforados de diferente tamaño, que superpuestos en orden descendente de tamaño, conforman una torre. Tienen, además tres ejes uno de origen, otro de destino y un tercer eje que puede ser utilizado como auxiliar.
El problema que los monjes pretenden resolver es encontrar la secuencia de movimiento de discos que permita trasladar la torre del eje origen al destino con la condición de que en ningún momento se apoye un disco sobre uno de menor tamaño. Los monjes realizan un movimiento por día y esperan que cuando concluyan se acabe el mundo. ¿Debemos preocuparnos ante la eventualidad de que tengan razón?

```
void hanoi(int, int, int, int, int*);

void ejercicio21(){
      int origen = 0, destino = 1, auxiliar = 2, mov;
      char a;

      mov = 0;
      hanoi(3, origen, destino, auxiliar, &mov);
      cout<<"\nCantidad de movimientos con 3 discos es: "<<mov;
      cout<<"\n\nIngrese un caracter para continuar..."; cin>>a;
      cout<<"\n\n";

      mov = 0;
      hanoi(7, origen, destino, auxiliar, &mov);
      cout<<"\nCantidad de movimientos con 7 discos es: "<<mov;
      cout<<"\n\nIngrese un caracter para continuar..."; cin>>a;
      cout<<"\n\n";

      mov = 0;
      hanoi(10, origen, destino, auxiliar, &mov);
      cout<<"\nCantidad de movimientos con 10 discos es: "<<mov;
      cout<<"\n\nIngrese un caracter para continuar..."; cin>>a;
      cout<<"\n\n";

      mov = 0;
      hanoi(14, origen, destino, auxiliar, &mov);
      cout<<"\nCantidad de movimientos con 14 discos es: "<<mov;
      cout<<"\n\nIngrese un caracter para continuar..."; cin>>a;
      cout<<"\n\n";

      mov = 0;
      hanoi(15, origen, destino, auxiliar, &mov);
      cout<<"\nCantidad de movimientos con 15 discos es: "<<mov;
      cout<<"\n\nIngrese un caracter para continuar..."; cin>>a;
      cout<<"\n\n";
```

```
 }

void mover (int, int, int);

void hanoi(int n, int o, int d, int a, int *mov){
      if (n > 0){
            hanoi(n-1, o, a, d, mov);
            mover (n, o, d);
            (*mov)++;
            hanoi(n-1, a, d, o, mov);
      }
}

void mover (int disco, int origen, int destino){
      char *pal[3];
      static cantPorLinea = 0;
      pal[0] = "ORIGEN";
      pal[1] = "DESTINO";
      pal[2] = "AUXILIAR";

      if (cantPorLinea == 2){
            cantPorLinea = 0;
            cout<< "\n";
      }
      cout<< "Disco "<<disco<<" : "<<pal[origen]<<" -> "<<pal[destino]<<"
";
      cantPorLinea ++;
}
```