Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Sistemas Computacionales

Análisis y diseño de algoritmos

Práctica 1

Multiplicación utilizando divide y vencerás. Estrategia de Gauss

Alumno: Sandoval Hernández Eduardo

Docente: Juárez Gambino Joel Omar

Grupo: 3CM1

Fecha de entrega: 28 de mayo de 2021

Introducción

La multiplicación es el resultado de sumar un número tantas veces como lo indique otro. La estrategia divide y vencerás consiste en resolver problemas difíciles dividiéndolos en problemas más simples tantas veces como sea necesario hasta que se tiene la resolución del problema general.

El presente reporte contiene la breve descripción del código generado en lenguaje C para resolver la multiplicación de dos números utilizando divide y vencerás y la estrategia de Gauss. Dicho código se apoyará en la estructura de datos Pila o Stack.

El código recibirá los dos números ya sea en base binaria, decimal o hexadecimal, la cual debe ser especificada, y posteriormente procederá a hacer las operaciones necesarias para obtener el resultado de la multiplicación entre ellos y mostrarlo al usuario tanto en la base especificada como en la base decimal.

Para la implementación de este código de utilizó la estructura de datos de la Pila, cuyo código de implementación en lenguaje C se incluye en la librería personal **Pila.h**

El código de implementación de la pila se explica de manera breve:

```
typedef struct Nodo{
    char dato;
    struct Nodo *next;
    } *Stack;
```

Se define la estructura Stack la cual es la base para la pila e incluye un dato tipo char el cual será donde se almacenen los números por separado, e incluye un apuntador al siguiente elemento de la pila.

```
Stack empty(){return NULL;}
```

La función empty() regresa un NULL que servirá para iniciar la pila vacía.

```
Stack push(char e, Stack l){
    Stack t=(Stack)malloc(sizeof(struct Nodo));
    t->dato=e;
    t->next=l;
    return t;
}
```

La función push(char e, Stack l) apila un char “e” a una pila “l”.

```
int isempty(Stack l){return l==NULL;}
```

La función isempty(Stack l) comprueba si la pila l es vacía o no, en caso afirmativo regresa un entero de valor 1.

```
char top(Stack l){return l ->dato;}
```

La función top(Stack l) regresa el dato que se encuentra apilado hasta la “cima” de la pila, o sea el último en entrar.

```
Stack pop(Stack l){return l ->next;}
```

La función pop(Stack l) regresa la pila l sin el elemento que se encuentra en la “cima”.

```
int Numchars(Stack l){
```

```
    if(isempty(l))
        return 0;
    else
        return 1+Numchars(pop(l));
}
```

La función Numchars(Stack l) regresa la cantidad de elementos (chars) apilados en la pila l.

```
void Impchar(char e){printf("%c ",e);}
```

La función Impchar(char e) imprime una variable de tipo char.

```
void Impchars(Stack l){
    if(!isempty(l)){
        Impchar(top(l));
        Impchars(pop(l));
    }
}
```

La función Impchars(Stack l) imprime en pantalla los elementos que se encuentran en la pila l.

```
Stack PegaStacks(Stack l1, Stack l2){
    if(isempty(l1))
        return l2;
    else
        return push(top(l1),PegaStacks(pop(l1),l2));
}
```

La función PegaStacks(Stack l1, Stack l2) une una pila l1 con otra pila l2 de modo que la pila l1 se apila en la pila l2.

```
Stack InvierteStacks(Stack l){
    if(isempty(l))
        return l;
    else
        return PegaStacks(InvierteStacks(pop(l)),push(top(l),empty()));
}
```

La función InvierteStacks(Stack l) cambia el orden de la pila invirtiendo esta.

Para la multiplicación se tiene el siguiente código:

```
long long int multiplica(Stack x, Stack y, int n, int base){

    //Comprueba si la candidad n de cifras es potencia de 2
```

```
    n = hazPotencia2(n);
    //Comprueba que ambos números tengan n cantidad de cifras
    if(Numchars(x)<n)
        x = agrega0(x,n-Numchars(x));
    if(Numchars(y)<n)
        y = agrega0(y,n-Numchars(y));

    if(n==1)
        return chartoint(top(x))*chartoint(top(y));
    else{
        Stack xi = dividir(x,'i',n);
        Stack xd = dividir(x,'d',n);
        Stack yi = dividir(y,'i',n);
        Stack yd = dividir(y,'d',n);
        Stack Sx = sumaStacks(xi,xd,base);
        Stack Sy = sumaStacks(yi,yd,base);
        long long int P1 = multiplica(xi,yi,n/2,base);
        long long int P2 = multiplica(Sx,Sy,mayorNumchars(Sx,Sy),base);
        long long int P3 = multiplica(xd,yd,n/2,base);

        long long int resultado = exp(base,n)*P1 + exp(base,n/2)*(P2-P1-
P3) + P3;
        return resultado;
    }
}
```

La última función “long long int multiplica(Stack x, Stack y, int n, int base)” es la que implementa la estrategia de Divide y Vencerás y la multiplicación de Gauss. Esta función recibe como argumentos dos pilas que representan dos números separados por sus dígitos, un entero n que representa la mayor cantidad de dígitos de los dos números y otro entero base que representa la base en la que se encuentran los números.

```
//Comprueba si la candidad n de cifras es potencia de 2
    n = hazPotencia2(n);
```

La primera parte, como dice la descripción en el código, comprueba si n es potencia de dos y en caso de que no lo sea esta función lo convertirá a la potencia de 2 próxima que sea mayor a n.

```
//Comprueba que ambos números tengan n cantidad de cifras
    if(Numchars(x)<n)
        x = agrega0(x,n-Numchars(x));
    if(Numchars(y)<n)
```

```
        y = agrega0(y,n-Numchars(y));
```

Esta parte comprobará que las pilas cuenten con n cantidad de elementos y de no ser así les apila n cantidad de 0 a la izquierda, por ejemplo: si se ingresan dos pilas que representan los siguientes números binarios [1][1][0][1][0] y [1][0][1][0] con n = 5. La primera parte cambiará n = 5 por n = 8 y posteriormente se asegurará que las pilas tengan n cantidad de dígitos haciendo que las pilas ahora sean [0][0][0][1][1][0][1][0] y [0][0][0][0][1][0][1][0] para que se pueda aplicar bien la estrategia divide y vencerás.

```
if(n==1)
        return chartoint(top(x))*chartoint(top(y));
    else{
        Stack xi = dividir(x,'i',n);
        Stack xd = dividir(x,'d',n);
        Stack yi = dividir(y,'i',n);
        Stack yd = dividir(y,'d',n);
        Stack Sx = sumaStacks(xi,xd,base);
        Stack Sy = sumaStacks(yi,yd,base);
        long long int P1 = multiplica(xi,yi,n/2,base);
        long long int P2 = multiplica(Sx,Sy,mayorNumchars(Sx,Sy),base);
        long long int P3 = multiplica(xd,yd,n/2,base);

        long long int resultado = exp(base,n)*P1 + exp(base,n/2)*(P2-P1-
P3) + P3;
        return resultado;
    }
```

Una vez que las pilas tengan una longitud aceptable se comprobará primero si se trata del caso base y de ser así regresa la multiplicación normal de los dos elementos como long long int (para facilitar los cálculos) con ayuda de la función chartoint(), la razón de usar long long se debe a que en el caso de la última prueba propuesta, el resultado en decimal es muy grande.

En caso de que no sea el caso base dividirá los números en mitades creando nuevas pilas con dichas mitades, tomando por ejemplo con los números anteriores:

X=[0][0][0][1][1][0][1][0] -> Xi=[0][0][0][1], Xd=[1][0][1][0]

Y=[0][0][0][0][1][0][1][0] -> Yi=[0][0][0][0], Yd=[1][0][1][0]

```
//Esta función suma dos pilas que representan un número dependiendo su base
//y regresa dicho número como una pila en la base que se especifica
Stack sumaStacks(Stack s1, Stack s2, int base){
    if(base==2){
        int ss1 = binToInt(s1,Numchars(s1)), ss2 = binToInt(s2,Numchars(s2))
, resultado;
        resultado = ss1 + ss2;
        return intToBin(resultado);
    }
    if(base==10){
        int ss1 = stackToInt(s1,Numchars(s1)), ss2 = stackToInt(s2,Numchars(
s2)), resultado;
        resultado = ss1 + ss2;
        return intToStack(resultado,cantDigInt(resultado));
    }
    if(base==16){
        int ss1 = hexToInt(s1,Numchars(s1)), ss2 = hexToInt(s2,Numchars(s2))
, resultado;
        resultado = ss1 + ss2;
        return intToHex(resultado);
    }
}
```

También creará dos pilas Sx & Sy que contendrán el resultado de Xi + Xd & Yi + Yd respectivamente con ayuda de la función sumaStacks, siguiendo con el ejemplo:

Xi + Xd = [0][0][0][1] + [1][0][1][0] internamente sumaStacks los convierte a decimales teniendo Xi + Xd = 1 + 10 = 11 y los regresa a la base binaria construyendo una nueva pila, Xi + Xd = [1][0][1][1] y repite lo mismo para Yi + Yd.

Posteriormente calculará P1, P2, P3 para la estrategia de Gauss:

P1 será igual al resultado de la multiplicación de Xi con Yi teniendo n/2 ya que se dividió a la mitad la pila y con la base siendo la misma.

Para P2 se multiplicarán las pilas resultantes de las sumas Xi + Xd & Yi + Yd (Sx & Sy), en esta ocasión n será determinado con ayuda de la función mayorNumchars() aplicado a Sx & Sy ya que puede suceder el siguiente caso hipotético:

Sx = Xi + Xd = [1][1] + [1][0] = [1][0][1] si anteriormente $n_1 = 2$ en este caso $n_2 \neq \frac{n_1}{2}$ sino $n_2 = 3$ y solo para esta suma, puede ser que la suma de Yi + Yd sea mayor y por tanto la función mayorNumchars() nos ayudará a saber que valor de n será

óptimo usar. Posteriormente la función recursiva multiplica() determinará si es necesario agregar 0 o no.

Y para P3 se seguirá el mismo procedimiento que para P1 pero con Xd & Yd, finalmente se calculará el resultado con la expresión exp(base,n)*P1 + exp(base,n/2)*(P2-P1-P3) + P3; regresando el resultado en entero para después utilizar las funciones intToStack(), intToBin() o intToHex() para mostrar el resultado de la multiplicación tanto en decimal como en una pila que representa el resultado en la base que se especificó.

El código completo se muestra a continuación:

```
//Sandoval Hernandez Eduardo
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include"Pila.h"

void imprime(Stack);
long int exp(long int,long int);
int chartoint(char);
char inttochar(int);
Stack dividir(Stack,char,int);
Stack agrega0(Stack,int);
int cantDigInt(int);
Stack intToStack(int,int);
int stackToInt(Stack,int);
Stack intToBin(int);
int binToInt(Stack,int);
long long int hexToInt(Stack,int);
Stack intToHex(long long int);
Stack sumaStacks(Stack,Stack,int);
int hazPotencia2(int);
int mayorNumchars(Stack,Stack);
long long int multiplica(Stack,Stack,int,int);

int main (){

    puts("*******BINARIO*******");
    puts("\n__Prueba 1__");
    Stack B1 = push('1',empty());
    Stack B2 = push('0',empty());
```

```
    int RB1B2 = multiplica(B1,B2,1,2);
    printf("Binario 1: ");
    imprime(B1);
    printf("Binario 2: ");
    imprime(B2);
    printf("Resultado decimal: %d\nResultado binario: ",RB1B2);
    imprime(intToBin(RB1B2));

    puts("\n__Prueba 2__");
    Stack B3 = push('1',push('1',push('0',push('1',push('0',empty()))));
    Stack B4 = push('1',push('0',push('1',push('0',empty()))));
    int RB3B4 = multiplica(B3,B4,5,2);
    printf("Binario 3: ");
    imprime(B3);
    printf("Binario 4: ");
    imprime(B4);
    printf("Resultado decimal: %d\nResultado binario: ",RB3B4);
    imprime(intToBin(RB3B4));

    puts("\n__Prueba 3__");
    Stack B5 = push('1',push('0',push('1',push('1',push('0',push('1',push('0
',push('1',empty()))))))));
    Stack B6 = push('1',push('0',push('1',empty())));
    int RB5B6 = multiplica(B5,B6,8,2);
    printf("Binario 5: ");
    imprime(B5);
    printf("Binario 6: ");
    imprime(B6);
    printf("Resultado decimal: %d\nResultado binario: ",RB5B6);
    imprime(intToBin(RB5B6));

    puts("\n\n*******DECIMAL*******");
    puts("\n__Prueba 4__");
    Stack D1 = push('8',empty());
    Stack D2 = push('9',empty());
    int RD1D2 = multiplica(D1,D2,1,10);
    printf("Decimal 1: ");
    imprime(D1);
    printf("Decimal 2: ");
    imprime(D2);
    printf("Resultado decimal: %d\nResultado en stack: ",RD1D2);
    imprime(intToStack(RD1D2,cantDigInt(RD1D2)));

    puts("\n__Prueba 5__");
    Stack D3 = push('1',push('2',push('3',push('4',push('5',empty()))));
```

```
    Stack D4 = push('3',push('4',empty()));
    int RD3D4 = multiplica(D3,D4,5,10);
    printf("Decimal 3: ");
    imprime(D3);
    printf("Decimal 4: ");
    imprime(D4);
    printf("Resultado decimal: %d\nResultado en stack: ",RD3D4);
    imprime(intToStack(RD3D4,cantDigInt(RD3D4)));

    puts("\n__Prueba 6__");
    Stack D5 = push('1',push('2',push('3',push('4',push('5',push('6',push('7
',empty()))))));
    Stack D6 = push('1',push('2',push('3',push('4',empty()))));
    int RD5D6 = multiplica(D5,D6,7,10);
    printf("Decimal 5: ");
    imprime(D5);
    printf("Decimal 6: ");
    imprime(D6);
    printf("Resultado decimal: %d\nResultado en stack: ",RD5D6);
    imprime(intToStack(RD5D6,cantDigInt(RD5D6)));

    puts("\n\n*******HEXADECIMAL*******");
    puts("\n__Prueba 7__");
    Stack H1 = push('A',empty());
    Stack H2 = push('4',empty());
    int RH1H2 = multiplica(H1,H2,1,16);
    printf("Hexadecimal 1: ");
    imprime(H1);
    printf("Hexadecimal 2: ");
    imprime(H2);
    printf("Resultado decimal: %d\nResultado hexadecimal: ",RH1H2);
    imprime(intToHex(RH1H2));

    puts("\n__Prueba 8__");
    Stack H3 = push('1',push('2',push('3',push('A',push('B',empty())))));
    Stack H4 = push('D',push('3',empty()));
    int RH3H4 = multiplica(H3,H4,5,16);
    printf("Hexadecimal 3: ");
    imprime(H3);
    printf("Hexadecimal 4: ");
    imprime(H4);
    printf("Resultado decimal: %d\nResultado hexadecimal: ",RH3H4);
    imprime(intToHex(RH3H4));

    puts("\n__Prueba 9__");
```

```
    Stack H5 = push('1',push('1',push('1',push('2',push('2',push('2',push('E
',empty()))))));
    Stack H6 = push('1',push('2',push('C',push('D',empty()))));
    long long int RH5H6 = multiplica(H5,H6,7,16);
    printf("Hexadecimal 5: ");
    imprime(H5);
    printf("Hexadecimal 6: ");
    imprime(H6);
    printf("Resultado decimal: %lld\nResultado hexadecimal: ",RH5H6);
    imprime(intToHex(RH5H6));

    return 0;
}

//Esta función sirve como complemento de Impchars ya que agrega un salto de
linea al final
void imprime(Stack P){
    Impchars(P);
    printf("\n");
}

//Esta función regresa el resultado de elevar un entero "a" a una potencia "
b"
long int exp(long int a, long int b){
    if(b==0)
        return 1;
    else
        return a*exp(a,b-1);
}

//Esta función regresa un char (desde 0 a 9 y de A a F) convertido a entero
int chartoint(char cc){
    if(isdigit(cc))
        return cc - '0';
    else
        return 10 + (toupper(cc) - 'A');
}

//Esta función hace lo contrario a la anterior pero solo con enteros del 0 a
l 9
char inttochar(int ii){
    return ii + '0';
}
```

```
//Esta función se encarga de dividir la pila a la mitad dependiento de que m
itad se desea
Stack dividir(Stack S, char mitad, int n){
    int i;
    Stack aux = empty();
    if(mitad=='i'){
        for(i=0 ; i<n/2 ; i++){
            aux = push(top(S),aux);
            S = pop(S);
        }
    }
    else{
        for(i=0 ; i<n/2 ; i++)
            S = pop(S);
        for(i=n/2 ; i<n ; i++){
            aux = push(top(S),aux);
            S = pop(S);
        }
    }
    aux = InvierteStacks(aux);
    return aux;
}

//Esta función agrega n canditad de 0 a la pila P
Stack agrega0(Stack P, int n){
    for(int i=0 ; i<n ; i++)
        P = push('0',P);
    return P;

}

//Esta función determina la cantidad de digitos que tiene un entero
int cantDigInt(int n){
    int i = 1;
    while((n/10)>0){
        n/=10;
        i++;
    }
    return i;
}

//Esta función divide los dígitos de un entero en una pila
Stack intToStack(int v, int n){
    if(n==0)
        return empty();
```

```
    else{
        int aux = ((v - v%(exp(10,n-1)))/exp(10,n-1));
        return push(inttochar(aux),intToStack(v%(exp(10,n-1)),n-1));
    }
}

//Esta función convierte una pila que representa un número decimal en un ent
ero
int stackToInt(Stack s, int n){
    if(isempty(s))
        return 0;
    else
        return chartoint(top(s))*exp(10,n-1) +  stackToInt(pop(s),n-1);
}

//Esta función convierte un entero a una pila que representa el mismo número
 en binario
Stack intToBin(int num){
    if(num==0)
        return push('0',empty());
    else if(num==1)
        return push('1',empty());
    else if(num%2==0)
        return PegaStacks(intToBin(num/2),push('0',empty()));
    else
        return PegaStacks(intToBin(num/2),push('1',empty()));
}

//Esta función toma una pila que representa un binario y la convierte a un e
ntero
int binToInt(Stack s, int n){
    if(isempty(s))
        return 0;
    else
        return chartoint(top(s))*exp(2,n-1) +  binToInt(pop(s),n-1);
}
//Esta función convierte una pila con hexadecimales a un entero
//Utiliza long long para enteros muy grandes como es el caso de la prueba 9
long long int hexToInt(Stack s, int n){
    if(isempty(s))
        return 0;
    else
        return chartoint(top(s))*exp(16,n-1) +  hexToInt(pop(s),n-1);
}
```

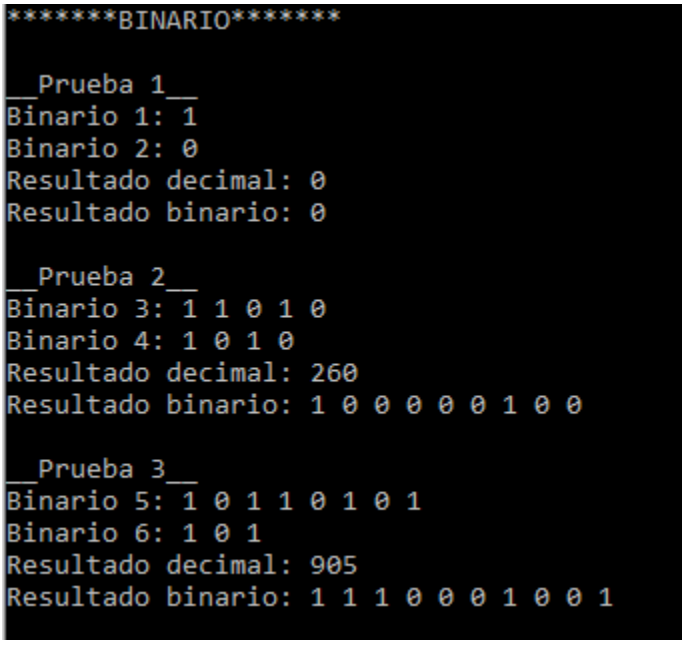
```
//Esta función convierte un entero a una pila que representa el mismo número
 en hexadecimal
Stack intToHex(long long int num){
    if(num<16){
        if(num<10)
            return push(inttochar(num),empty());
        else
            return push(num+55,empty());
    }
    else if(num/16 != 0){
        if(num%16==0)
            return PegaStacks(intToHex(num/16),push('0',empty()));
        else if(num%16<10)
            return PegaStacks(intToHex(num/16),push(inttochar(num%16),empty(
)));
        else
            return PegaStacks(intToHex(num/16),push(num%16+55,empty()));
    }
}

//Esta función suma dos pilas que representan un número dependiendo su base
//y regresa dicho número como una pila en la base que se especifica
Stack sumaStacks(Stack s1, Stack s2, int base){
    if(base==2){
        int ss1 = binToInt(s1,Numchars(s1)), ss2 = binToInt(s2,Numchars(s2))
, resultado;
        resultado = ss1 + ss2;
        return intToBin(resultado);
    }
    if(base==10){
        int ss1 = stackToInt(s1,Numchars(s1)), ss2 = stackToInt(s2,Numchars(
s2)), resultado;
        resultado = ss1 + ss2;
        return intToStack(resultado,cantDigInt(resultado));
    }
    if(base==16){
        int ss1 = hexToInt(s1,Numchars(s1)), ss2 = hexToInt(s2,Numchars(s2))
, resultado;
        resultado = ss1 + ss2;
        return intToHex(resultado);
    }
}

//Esta función determina si un entero num es potencia de 2 y si no regresa l
a potencia de 2
```

```
//siguiente a dicho número, ejemplo, si num = 5 entonces regresa 8.
int hazPotencia2(int num){
    int i = 2;
    if(num<=i)
        return num;
    else{
        while(1){
            i*=2;
            if(num<=i)
                return i;
        }
    }
}

//Esta función determina cuál de los pilas tiene la mayor cantidad de elemen
tos y regresa
//el número de elementos
int mayorNumchars(Stack s1, Stack s2){
    if(Numchars(s1)>=Numchars(s2))
        return Numchars(s1);
    else
        return Numchars(s2);
}

long long int multiplica(Stack x, Stack y, int n, int base){

    //Comprueba si la candidad n de cifras es potencia de 2
    n = hazPotencia2(n);
    //Comprueba que ambos números tengan n cantidad de cifras
    if(Numchars(x)<n)
        x = agrega0(x,n-Numchars(x));
    if(Numchars(y)<n)
        y = agrega0(y,n-Numchars(y));

    if(n==1)
        return chartoint(top(x))*chartoint(top(y));
    else{
        Stack xi = dividir(x,'i',n);
        Stack xd = dividir(x,'d',n);
        Stack yi = dividir(y,'i',n);
        Stack yd = dividir(y,'d',n);
        Stack Sx = sumaStacks(xi,xd,base);
        Stack Sy = sumaStacks(yi,yd,base);
        long long int P1 = multiplica(xi,yi,n/2,base);
        long long int P2 = multiplica(Sx,Sy,mayorNumchars(Sx,Sy),base);
```

```
        long long int P3 = multiplica(xd,yd,n/2,base);

        long long int resultado = exp(base,n)*P1 + exp(base,n/2)*(P2-P1-
P3) + P3;
        return resultado;
    }
}
```

Resultados para las pruebas.

Para las pruebas se muestran los números en la base que se especifica y posteriormente el resultado en decimal y su representación como pila en la base original.



```
*******DECIMAL*******

__Prueba 4__
Decimal 1: 8
Decimal 2: 9
Resultado decimal: 72
Resultado en stack: 7 2

__Prueba 5__
Decimal 3: 1 2 3 4 5
Decimal 4: 3 4
Resultado decimal: 419730
Resultado en stack: 4 1 9 7 3 0

__Prueba 6__
Decimal 5: 1 2 3 4 5 6 7
Decimal 6: 1 2 3 4
Resultado decimal: 1523455678
Resultado en stack: 1 5 2 3 4 5 5 6 7 8
```

```
*******HEXADECIMAL*******

__Prueba 7__
Hexadecimal 1: A
Hexadecimal 2: 4
Resultado decimal: 40
Resultado hexadecimal: 2 8

__Prueba 8__
Hexadecimal 3: 1 2 3 A B
Hexadecimal 4: D 3
Resultado decimal: 15754737
Resultado hexadecimal: F 0 6 5 F 1

__Prueba 9__
Hexadecimal 5: 1 1 1 2 2 2 E
Hexadecimal 6: 1 2 C D
Resultado decimal: 86153075414
Resultado hexadecimal: 1 4 0 F 1 F 9 A D 6
```