



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



## Practica 1 Algoritmos básicos

Integrantes:

Escutia López Arturo

López Santiago Daniel

28/Ago/2016

Unidad de aprendizaje: Análisis de algoritmos

Profesor: Díaz Santiago Sandra

# Teoría

Normalmente cuando se trata de representar un polinomio, ésta puede ser expresada comúnmente de la siguiente forma

$$f(x) = \sum_{k=0}^n a_k x^k$$

Como una sumatoria desde  $k=0$  hasta el elemento  $n$ -ésimo donde los coeficientes son valores constantes que multiplican un valor  $X$  con una potencia que depende del valor actual de  $k$  en la sumatoria.

Gracias a la regla de Horner es una forma que nos simplifica la manera en la que se evalúa un polinomio dividiendo el polinomio respectivamente a monomios (son polinomios de grado 1) mediante factorizaciones, haciendo así que sea sólo realizar una suma y una multiplicación por cada ciclo, la cual una vez calculado es sumado en el siguiente monomio, y así de manera sucesiva. En cambio de la manera tradicional habría que sacar la potencia mediante una sucesión de multiplicaciones, luego multiplicar por el coeficiente esto en cada iteración y posteriormente sumar los resultados con los otros  $a_k$  y  $x^k$ .

Supongamos que se quiere evaluar el polinomio

$$f(x_0) = 3 + 6x_0 + 10x_0^2 + x_0^3 + 2x_0^4 + 5x_0^5$$

Simplificando en monomios quedaría de la siguiente manera

$$f(x_0) = 3 + x_0(6 + x_0(10 + x_0(1 + x_0(2 + 5x_0))))$$

De esta forma se reduce el número de multiplicaciones comparadas con realizar las potencias de cada  $X$  por su coeficiente correspondiente, y como sabemos realizar una secuencia de multiplicaciones le toma tiempo a las computadoras.

Debemos empezar multiplicar y sumar desde la potencia más grande hacia la menor, o de manera visual desde la operación más anidada entre paréntesis hacia afuera

```
Suma=a[n]  
For i=n-1 to 0  
    Suma<-Suma*X0+a[i]  
End for
```

## Implementación

1.-Aquí vemos una simple evaluación, aquí simplemente la parte del código que nos importa es el ciclo for ya que ahí es donde se lleva a cabo la evaluación del polinomio, ya que se multiplica  $p$  que representa la potencia con  $\text{nums}[i]$  que sería su coeficiente correspondiente y se le suma el resultado de la misma operación pero de coeficientes y potencias diferentes que al final resultara el valor en la sumatoria.

```
#include <bits/stdc++.h>  
  
using namespace std;  
  
int main (int arg, char *c[]){  
  
    ifstream f(c[1]);  
    vector <int> nums;  
    int suma,p=1,x0;  
    string num,x;  
    if(f.is_open()){  
        while(getline(f,num))  
            nums.push_back(atoi(num.c_str()));  
  
        f.close();  
    }  
  
    x.append(c[2]);  
    x0=atoi(x.c_str());  
    for (int i =0;i<nums.size();i++){  
        suma=(nums[i]*p)+suma;  
        p=p*x0;  
    }  
    cout<<suma<<"\n";  
  
    return 0;  
  
}
```

2.-Aquí igual se evalúa el polinomio pero con la regla de horner, aquí solo nos interesa el ciclo for, empezamos por la potencia más grande con su coeficiente correspondiente hacia el menor del polinomio es por eso que decrementamos, aquí nos ahorramos la multiplicación demás del programa anterior de  $p=p*x$  debido a como está estructurado la forma de evaluar según la regla de horner como vimos en la teoría.

```
#include <bits/stdc++.h>

using namespace std;

int main (int arg,char *c[]){

    ifstream f(c[1]);
    vector <int> nums;
    int suma,x0;
    string num,x;
    if(f.is_open()){
        while(getline(f,num))
            nums.push_back(atoi(num.c_str()));

        f.close();
    }

    x.append(c[2]);
    x0=atoi(x.c_str());
    suma=nums[nums.size()-1];
    for (int i =nums.size()-2;i>=0;i--)
        suma=(suma*x0)+nums[i];

    cout<<suma<<"\n";

    return 0;

}
```



```
algoritmos : bash - Konsole
File Edit View Bookmarks Settings Help
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos > ./poli inumber 3
763086
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos > ./horner inumber 3
763086
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos > █
```

3.-Los nombres de un archivo los almacenamos en un vector llamado names, para poder realizar la búsqueda simplemente al final de names agregamos el nombre a buscar en la lista. Iteramos en el while y hasta que encuentra una coincidencia pararemos de iterar. Si nuestro contador se detuvo en una posición diferente a la última localidad de la lista, quiere decir que el nombre coincidió antes y ya estaba presente, de lo contrario si llega hasta el final quiere decir que el nombre no estaba en la lista ya que por eso insertamos al final de la lista el nombre a buscar como una forma de marcar una bandera.

```
#include <bits/stdc++.h>
using namespace std;
int main (int arg, char *c[]){
    ifstream f(c[1]);
    vector <string> names;
    string name;
    int i=0;
    if(f.is_open()){
        while(getline(f,name))
            names.push_back(name);
        f.close();
    }
    stringstream s;
    s<<c[2]<<" "<<c[3]<<" "<<c[4];
    name=s.str();
    names.push_back(name);
    while(name!=names[i]){
        //cout<<names[i]<<"\n";
        i=i+1;
    }
    if(i!=names.size()-1)
        cout<<"Nombre en la posición:"<<i<<"\n";
    else
        cout<<"El nombre no se encuentra en la lista \n";
    return 0;
}
```

4.-Con este programa a diferencia del anterior , la lista está obligada a estar ordenada ya que de lo contrario el algoritmo no funciona correctamente, la idea es ir dividiendo a la mitad la lista hasta que haya una coincidencia con el elemento de central o ya no sea más posible dividir la lista por la mitad, la razón por la que debe de estar ordenada la lista es que dependiendo de si el valor que se compara es menor o mayor al elemento ,determinara si se vuelve a dividir la primera mitad de la lista o la segunda mitad. En este caso llamamos de manera recursiva para dividir la lista con la función BinarySearch, dependiendo de la condición o retornamos un valor o volvemos a llamar la función definiendo de donde a donde

volveremos a partir en 2 la lista, en caso de una coincidencia solo regresamos la posición en el arreglo donde encontramos el nombre.

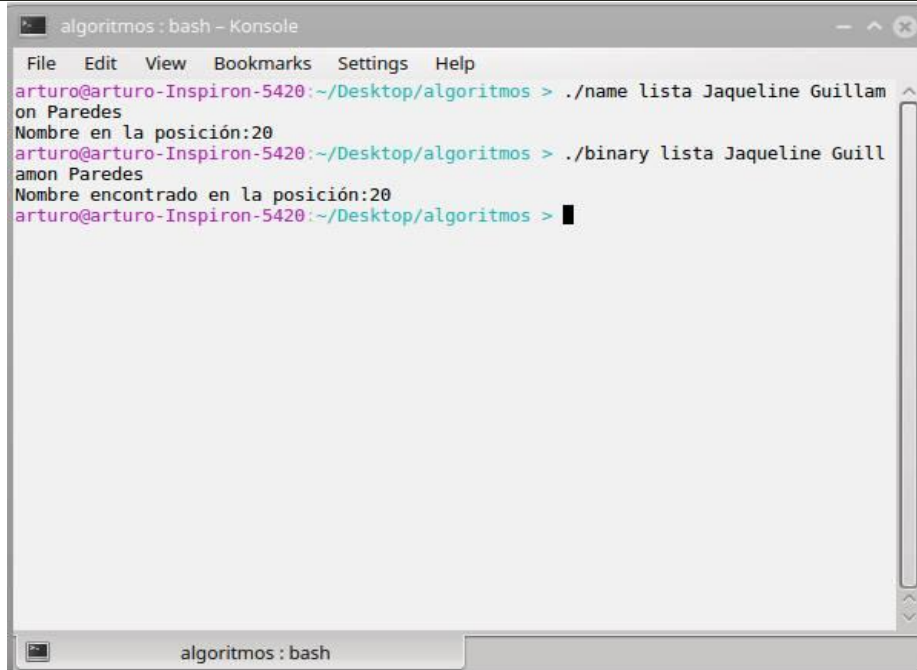
```
#include <bits/stdc++.h>

using namespace std;

int BinarySearch(vector <string> nms,string nm ,int low,int high);
int main (int arg,char *c[]){

    ifstream f(c[1]);
    vector <string> names;
    string name;
    int i=0;
    if(f.is_open()){
        while(getline(f,name))
            names.push_back(name);
        f.close();
    }
    stringstream s;
    s<<c[2]<<" "<<c[3]<<" "<<c[4];
    name=s.str();
    int flag=BinarySearch(names,name,0,names.size()-1);
    if (flag!=-1)
        cout<<"Nombre encontrado en la posición:"<<flag<<"\n";
    else
        cout<<"No se encontró en la lista \n";
    return 0;
}

int BinarySearch(vector <string> nms,string nm ,int low,int high){
    int middle=(high+low)/2;
    if(low>high) return -1;
    if(nms[middle]==nm) return middle;
    if(nms[middle]<nm)
        return BinarySearch(nms,nm,middle+1,high);
    else
        return BinarySearch(nms,nm,low,middle-1);
}
```



```
algoritmos : bash - Konsole
File Edit View Bookmarks Settings Help
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos > ./name lista Jaqueline Guillon Paredes
Nombre en la posición:20
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos > ./binary lista Jaqueline Guillon Paredes
Nombre encontrado en la posición:20
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos > █

algoritmos : bash
```

## Referencias

<https://www.math10.com/en/algebra/horner.html>

<http://www.uv.mx/anmarin/slides/MetNum2.4.pdf>