INSTITUTO POLITECNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA CAMPUS ZACATECAS

Ing. Sistemas Computacionales

Análisis de Algoritmos

Jesús Abelardo Dávila Mauricio

Presenta:

EMPAREJAMIENTO DE CADENAS

FUERZA BRUTA Y HASH

(Reporte)

A 14 de Noviembre del 2019

Introducción

Una cadena es una secuencia de caracteres sobre un alfabeto finito.

• $ATCTAGAGA$ es una cadena sobre Σ = $\{A, C, G, T\}$

El problema de emparejamiento de cadenas es encontrar todas las ocurrencias de una cadena p, llamada patrón, en una cadena más grande T del mismo alfabeto.

Dadas las cadenas x, y y z, se dice que x es:

• a) Un prefijo de xy,

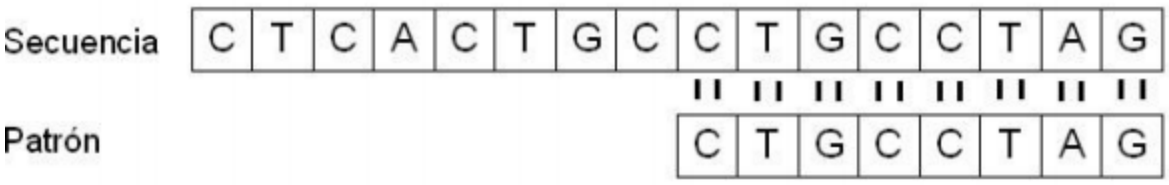
• b) Un sufijo de yx,

• c) Una subcadena de yxz.

Empate de cadenas

Implica la implementación de algoritmos de búsqueda de subcadenas y por lo tanto su objetivo es buscar la existencia de una subcadena dentro de una cadena.

La mayoría de los algoritmos para este problema se pueden modificar fácilmente para encontrar todas las ocurrencias del patrón en el texto, puesto que recorren el texto en secuencia y se pueden reinicializar en la posición situada inmediatamente después del comienzo de una concordancia, para encontrar la concordancia siguiente.



Descripción

Fuerza bruta

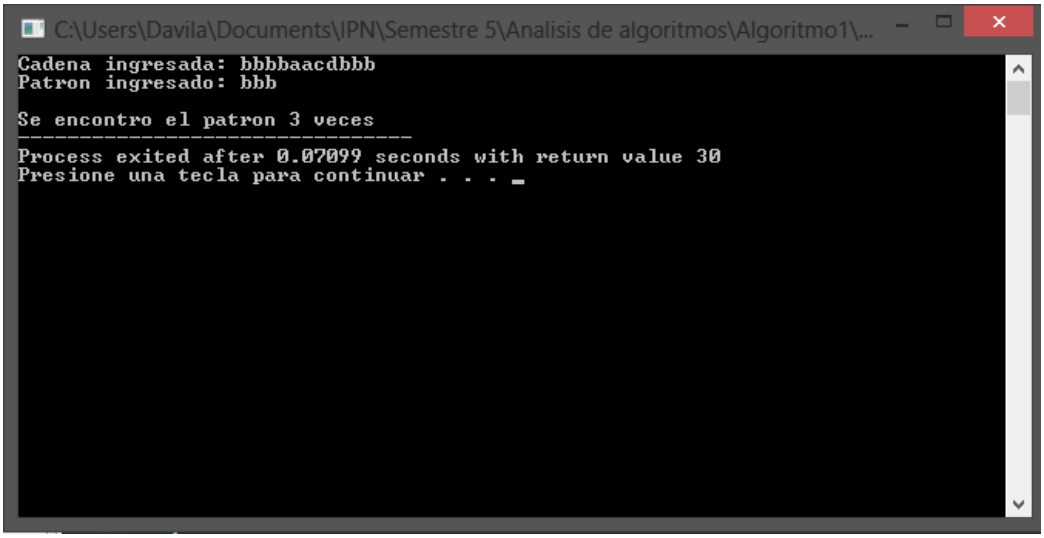
El método en el que se piensa de inmediato para el reconocimiento de patrones consiste simplemente en verificar, para cada posición posible del texto en la que el patrón pueda concordar, si efectivamente lo hace.

El algoritmo de Fuerza Bruta compara el patrón con el texto un carácter cada vez, hasta encontrar que no coinciden los caracteres.

Código y resultados.

```
#include <stdio.h>
#include <string.h>

int buscar_patron(char *cad, char *pat);

int main() {
	int n = 0;
	char *cadena = "bbbbaacdbbb";
	char *patron = "bbb";
	printf("Cadena ingresada: %s \nPatron ingresado: %s\n", cadena , patron);
	n = buscar_patron(cadena, patron);
	if(n != 0){
		printf("\nSe encontro el patron %d veces",n);
	}
	else{
		printf("No se encontro la cadena");
	}
}
int buscar_patron(char *cad, char *pat){
	int longitud1 = strlen(cad);
	int longitud2 = strlen(pat);
	int i, j;
	char c;
	if(longitud2 > longitud1)
	{
		printf("El patron es mas grande que la cadena");
		return 0;
```

```
        }else{
                i=0;


                j=0;
                c=pat[0];
                while(i < longitud1){
                        /*Si el primer elemento del patron es iguan al elemento de la cadena en la posicion i,
                        se puede avanzar y comparar los demas elementos,
                        si no, no se comparan y avanza hasta encontrarse de nuevo con el caso.
                        */
                        if(cad[i] == c){
                                if( strncmp( &cad[i], pat, longitud2 ) == 0 ){
                                        j++;
                                }
                        }
                i++;
                }
        return j;
        }
}
```



Rabin-karp

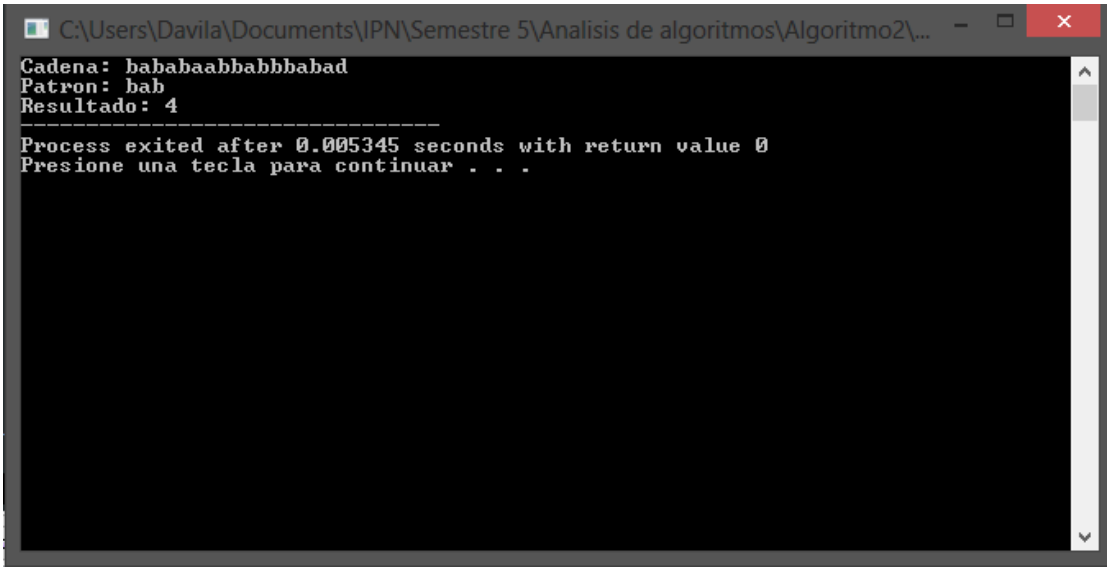
Rabin Karp es un algoritmo de búsqueda de cadenas, creado por Richard M. Karo y Michael O. Rabin en 1987 que utiliza funciones hash y la técnica hash rolling, para encontrar patrones en un texto, calcula un valor numérico(hash) para el patrón **p**, y para cada carácter de **m** subcadena del texto **t**. Luego compara los valores numéricos en lugar de comparar los símbolos reales. Si se encuentra una coincidencia, compara el patrón con la subcadena mediante un enfoque ingenuo. De lo contrario, se desplaza a la siguiente subcadena de t para comparar con p. Una de las aplicaciones prácticas de este algoritmo es la detección de copia, pues el algoritmo logra buscar frases en un texto con rapidez.

Código y resultados

```
#include <stdio.h>
#include<string.h>
#define d 256

int busqueda(char cad[], char pat[], int valorHash);

  int main()
```

```
{
 int q;
 char cadena[40]="babababbabbbabad";
        char patron[40]="bab";
        int total = busqueda(cadena,patron,1);
 printf("Cadena: %s\nPatron: %s\nResultado: %d",cadena,patron,total );
 return 0;
}


int busqueda(char cad[], char pat[], int valorHash){

        int longitud1= strlen(pat);
        int longitud2= strlen(cad);
        int i, j, patron=0, cadena=0, hash=1, contador=0;
        for(i=0; i<longitud1-1; i++){
        patron=((d*patron + pat[i])%valorHash);
                cadena=((d*cadena + cad[i])%valorHash);
        }
        for(i=0; i<= longitud2-longitud1; i++){
                if(patron==cadena){
        for(j=0; j<longitud1; j++){
                if(cad[i+j]!=pat[j])
                        break;
                        }
                if(j==longitud1){
        contador++;
                }
```

```
                }
        else if(i<longitud2-longitud1){
            cadena=(d*(cadena-cad[i]*hash)+cad[i+longitud2])%valorHash;
            if (cadena<0)
            cadena=(cadena+valorHash);
            }
            }
            return contador;
}
```



Conclusion

Durante el desarrollo de estos algoritmos, tuve una cierta dificultad a pesar de solo se tenían que comparar cadenas, pero al final pude resolverlos. Y además lo que observe en estos algoritmos es que, a pesar de que resuelven el mismo problema, se pueden solucionar de diferente manera.