UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

ALGORITMOS Y ESTRUCTURA DE DATOS

Practica #3. Algoritmos fuerza bruta con recursión

ALUMNO: CAUDILLO SANCHEZ DIEGO

MATRICULA: 1249199

GRUPO: 551

DOCENTE: Alma Leticia Palacios

FECHA DE ENTREGA: 08/Marzo/2019

Introducción

Los algoritmos de fuerza bruta son técnicas de resolución de problemas que consiste en enumerar sistemáticamente todos los posibles candidatos para una solución y verificar si cada candidato satisface el enunciado del problema.

La búsqueda de fuerza bruta es simple de implementar, y siempre encontrara la solución, si es que existe. El costo es proporcional al numero de posibles soluciones, lo cual en algunos casos tiende a crecer muy rápido, así como el tamaño del problema incrementa. Además, la búsqueda por fuerza bruta es típicamente usada cuando el tamaño del problema es limitado. El método es usado cuando la simplicidad de la implementación es mas importante que la velocidad.

Algoritmo básico

En orden de querer aplicar la búsqueda por fuerza bruta a una clase de problemas especifico, se debe implementar cuatro procedimientos, *primero, siguiente, valido y salida.* Estos procedimientos deben tomar como parámetro el dato *P* para la instancia del problema que se va ser resuelto, y debe seguir lo siguiente:

1. *Primero (P)*: genera un primer candidato para *P*
2. *Siguiente (P, c)*: genera el siguiente candidato para *P* después del actual candidato *c*.
3. *Valida (P, c)*: verifica si el candidato c es una solución para *P*.
4. *Salida (P, c)*: usa la solución *c* para *P* como apropiada para la aplicación.

El procedimiento *siguiente()* es el que debe decir cuando no haya mas candidatos para la instancia *P*, después del candidato actual *c*.

Competencia

Diferenciar las ventajas de los algoritmos recursivos sobre los algoritmos iterativos a través de la implementación de un algoritmo fuerza bruta recursivo, en el que se apliquen de manera apropiada el manejo de cadenas y recursión buscando el uso eficiente de los recursos mientras se asegura la integridad de los datos en forma responsable.

Problema

Una manera de recuperar una contraseña es a través de ataque fuerza bruta, en el que se prueban de manera exhaustiva todas las combinaciones posibles hasta encontrar aquella que permite el acceso. Diseñe e implemente un algoritmo recursivo que a partir de una cadena de proporcionada por el usuario genere todas las combinaciones que se puedan obtener a partir de los caracteres que la forman. Por ejemplo, si la cadena que el usuario introdujo contiene las letras S, t, r, i, n, g. La salida de su programa debería ser todas las posibles combinaciones.

- El usuario debe introducir las cadenas.
- El tamaño de la cadena es desconocido.
- El programador debe limitar el tamaño máximo de la cadena
- El algoritmo DEBE ser RECURSIVO
- La cadena puede contener cualquier carácter alfanumérico.
- La cadena puede contener caracteres repetidos.
- Debe respetarse el uso de minúsculas y mayúsculas.
- La cadena debe contener al menos un carácter.
- Las funciones solo deben realizar una tarea.
- Lenguaje de programación C.
- La estructura del programa debe ser siguiendo el estándar ANSI C.

Codificación

```
/* UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA
Asignatura: Algoritmos y estructura de datos
Alumno: Diego Caudillo Sanchez
Matricula: 1249199
Grupo: 551
Docente: Alma Leticia Palacios Guerrero
Fecha de entrega: 08/Marzo/2019

DESCRIPCION DEL PROGRAMA
Una manera de recuperar una contraseña es a través de ataque fuerza bruta,
en el que se prueban de manera exhaustiva todas las combinaciones posibles
hasta encontrar aquella que permite el acceso. Diseñe e implemente un algo-
ritmo recursivo que a partir de una cadena de proporcionada por el usuario
genere todas las combinaciones que se puedan obtener a partir de los carac-
teres que la forman. Por ejemplo, si la cadena que el usuario introdujo con-
```

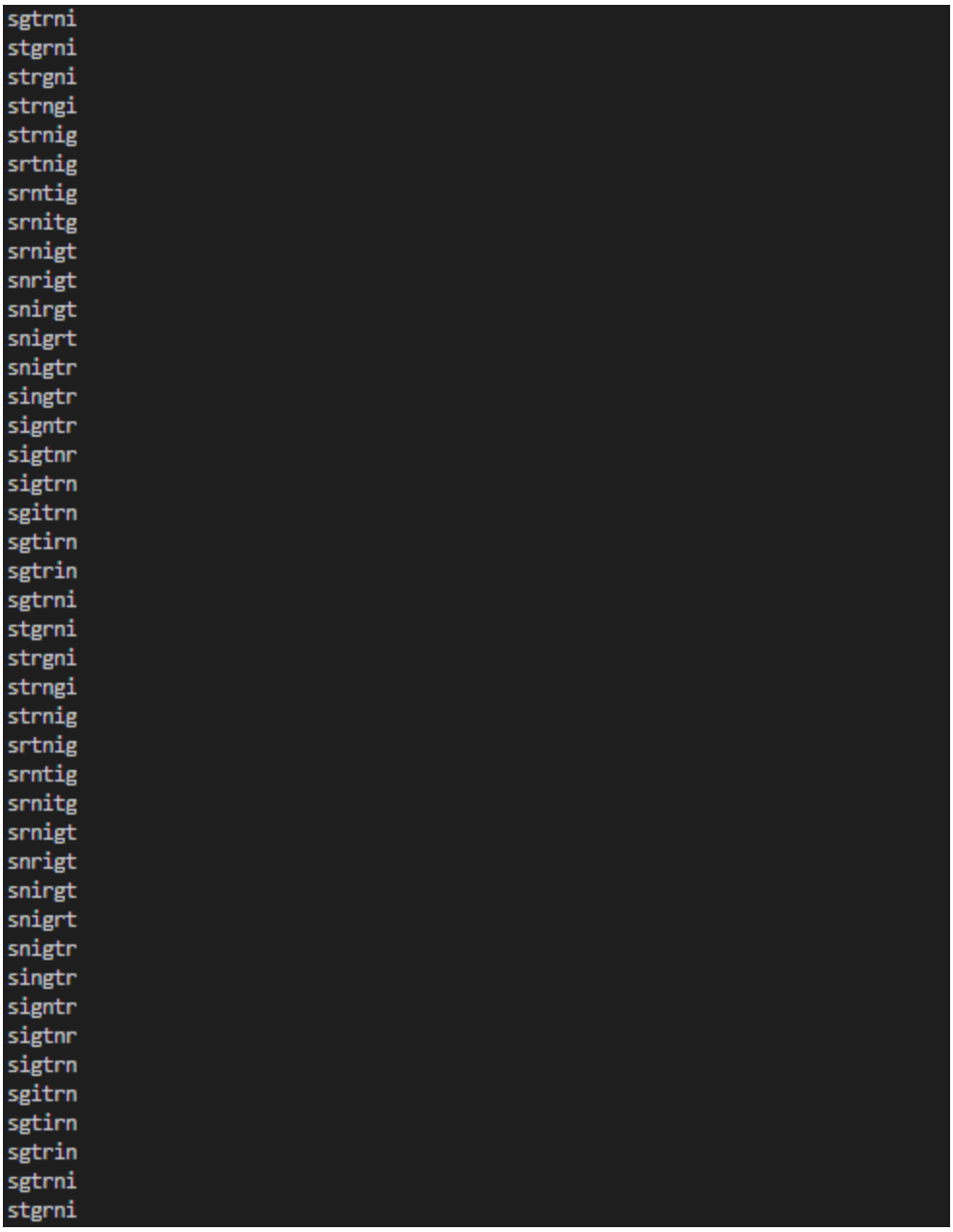
```
tiene las letras S, t, r, i, n, g. La salida de su programa debería ser to-
das las posibles combinaciones.

♣ El usuario debe introducir las cadenas.
♣ El tamaño de la cadena es desconocido.
♣ El programador debe limitar el tamaño máximo de la cadena
♣ El algoritmo DEBE ser RECURSIVO
♣ La cadena puede contener cualquier carácter alfanumérico.
♣ La cadena puede contener caracteres repetidos.
♣ Debe respetarse el uso de minúsculas y mayúsculas.
♣ La cadena debe contener al menos un carácter.
♣ Las funciones solo deben realizar una tarea.
♣ Lenguaje de programación C.
♣ La estructura del programa debe ser siguiendo el estándar ANSI C.

*/
/*Bibliotecas utilizadas en el programa*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/*HEADERS*/
void combinaciones(char* str, int init, int base, int str_size);
int factorial(int number);

/*Función principal*/
int main(int argc, char const *argv[])
{
    char str[20];
    int init = 1, base = 2;
    printf("Ingrese una palabra: "); gets(str);
    combinaciones(str, init, base, strlen(str)-1);
    getchar();
    return 0;
}
 /*
    DESCRIPCION
        Funcion que aplica la busqueda por fuerza bruta, la cual realiza todas
las
        combinaciones posibles para una cadena dada. La función no devuelve ni un
valor,
        pero cambia el valor de la cadena.
    PARAMETROS
        -str: Cadena la cual contiene la palabra ingresada por el usuario.
```

```
        -init: Inicializador que conforme va llamandose la recursion, va
aumentando hasta
        llegar al mismo número del tamaño de la cadena.

        -base: Es un número base el cual ayuda a determinar la cantidad de veces
que se va
        a realizar un intercambio de caracteres, por llamada a función. Siempre
se inicializa
        en 2 y se va incrementado por 1, siempre y cuando la cadena sea mayor a
1.

        -str_size: tamaño de la cadena menos 1. Se le resta un 1 debido a que al
momento de
        accesar al arreglo no haya confusión, ya que se accesa de derecha a
izquierda.
 */
void combinaciones(char* str, int init, int base, int str_size)
{
    char tmp; // caracter temporal donde se guarda un caracter para hacer un
swap.
    int i = 0, x = 0;
    if (init < strlen(str)) // La recursión termina cuando se llegue al tamaño de
la cadena.
    {
        while(init+x <= factorial(base)) // la cantidad de iteraciones es con
base a la cantidad de letras
        //iniciando de derecha a izquierda, tomando las ultimas 2 letras,
posteriormente se van agregando
        //letras para el intercambio. La cantidad de iteraciones esta dado por n!
(factorial), donde n es
        //la cantidad de letras, tomando que cuenta que ninguna letra se repite.
        {
            if(i == init) i = 0;    //La variable que recorre el arreglo se
reinicializa en 0 debido a que
            //si sigue aumentando se saldría de las dimensiones del arreglo,
imprimiendo así basura.
            tmp = str[str_size-init+i]; // se guarda la letra que se requiere en
un temporal para poder
            //asignarla y hacer el intercambio, de otro modo perderiamos la letra
ya que se sobre escribe.
            i++; //se aumenta la variable en 1 para ir recorriendo el arreglo.
            str[str_size-init+i-1] = str[str_size-init+i];
            str[str_size-init+i] = tmp;
            printf("%s\n", str);
```

```
            x++; //incremento de la variable para que no se cicle infinitamente.
        }combinaciones(str, init+1, base+1, str_size);
    }
}

/*
    DESCRIPCION
        Funcion que devuelve el factorial de un numero entero.
    PARAMETROS
        - number: numero entero al cual se le determina su factorial.
*/
int factorial(int number)
{
    int i, fact = 1;
    for(i = 1; i <= number; i++)
        fact *= i;
    return fact;
}
```

Evidencia

```
PS C:\Users\caudi\Documents\UABC\Semestre 5-2\Algoritmos\Laboratorio\Practica 3> ./d
Ingrese una palabra: string
strign
string
strnig
strngi
strgni
strgin
strign
stirgn
stigrn
stignr
stginr
stgnir
stgnri
stngri
stnrgi
stnrig
strnig
string
strign
stirgn
stigrn
stignr
stginr
stgnir
stgnri
stngri
stnrgi
stnrig
strnig
srtnig
srntig
srnitg
srnigt
snrigt
snirgt
snigrt
snigtr
singtr
signtr
```



Conclusión

Al finalizar la practica se concluye que la recursión en aplicaciones de ordenamiento o búsqueda pueden ser eficaces. En este caso al programar la búsqueda por fuerza bruta la cual, la búsqueda crece muchísimo en relación a la cantidad de caracteres que se este ordenando. La proporción de veces que se ejecuta esta dada por *n!*, donde *n* es el tamaño de la cadena ingresada por el usuario.

Fuente

Berstein J.D. (s.f). *Understanding brute force*. University of Illinois: Chicago.

Levitin A. (2012). *Brute force and Exhaustive search*. En Introduction to Design and Analysis of Algorithms(pp.105-108). New Jersey: Pearson.