Tiempos de ejecución y búsqueda de palabras

lntegrantes:

Manuel Silva 202173642-5

Nicolas Rodríguez 202173515-1

Introducción:

El presente informe se mostrará cuanto tiempo se demora el programa realizado, en buscar una palabra tanto de forma horizontal como vertical. Para medir el tiempo de ejecución se utilizó la función clock de la librería *time.h,* esta fue utilizada antes de llamar a la función de búsqueda y después de haber realizado la búsqueda para luego hacer una resta entre estos dos valores, el resultado obtenido son los pulsos de reloj (Clocks) que se demoró el programa en buscar la palabra. Un Clock corresponde a la millonésima parte de 1 segundo, por lo que para presentar el tiempo obtenido se dividieron los Clocks entre 1.000.000 para corresponderse al tiempo en segundos.

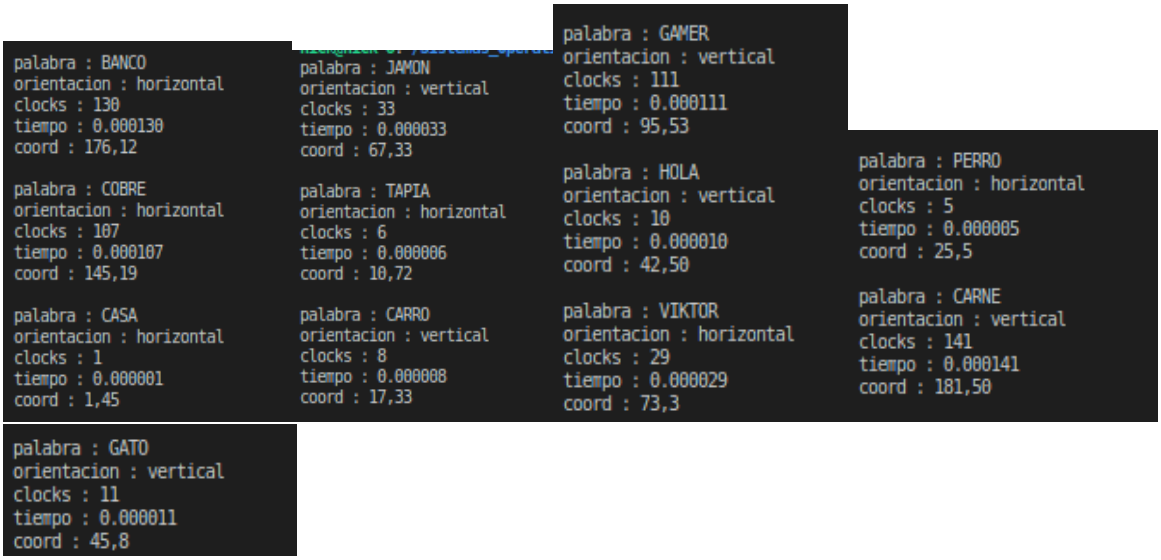
Desarrollo:

1. Tiempos de ejecución

Palabra	Tiempo en búsqueda	Orientación	Clocks
Gamer	0.000111 [s]	Vertical	111
Gato	0.000011 [s]	Vertical	11
Hola	0.000010 [s]	Vertical	10
Jamón	0.000033 [s]	Vertical	33
Carne	0.000141 [s]	Vertical	141
Carro	0.000008 [s]	Vertical	8
Banco	0.000130 [s]	Horizontal	130
Casa	0.000001 [s]	Horizontal	1
Cobre	0.000107[s]	Horizontal	107
Perro	0.000005 [s]	Horizontal	5
Viktor	0.000029 [s]	Horizontal	29
Tapia	0.000006 [s]	Horizontal	6

2. ¿Qué palabra tuvo un mayor tiempo de ejecución?

Según los datos obtenidos que están insertos en la tabla la palabra con mayor tiempo de ejecución fue Carne.



Aparte de la palabra y orientación, se incluyen las coordenadas donde se encontró la palabra dentro de la matriz correspondiente a la sopa de letras, estas apuntan al final o al principio de la palabra.

3. ¿Qué orientación tuvo un menor tiempo de ejecución?

La orientación Vertical tuvo $\frac{(11+33+8+111+10+141)}{6} = 52.3$ Clocks de duración promedio

Mientras que horizontal tuvo $\frac{(130+1+107+5+29+6)}{6} = 46.3$ Clocks en promedio

En promedio la orientación horizontal tuvo menor tiempo de ejecución. Pensamos que esto se debe que el acceso a la memoria de forma horizontal es de forma sucesiva dado que malloc() guarda regiones de memoria contiguas y en cambio en vertical se guardan los punteros que apuntan a las regiones de memoria por lo que eso puede llevar a un “mayor” tiempo de ejecución.

4. Código Optimizado

Para la correcta resolución del problema se implementaron dos formas de búsqueda las cuales fueron

-Al encontrar la primera letra se tomaban las N-1 letras siguientes dentro de la matriz, siendo N el largo de la palabra a buscar, para luego recorrer en un ciclo y comparar letra por letra para ver si correspondían unas a otras.

-Contar las veces que aparecían caracteres sucesivos hasta llegar a N implicando así que se había encontrado la palabra a buscar.

Se mostrarán las dos versiones del código para una orientación, dado que para ambas orientaciones las ideas son lo mismo:

```
coord *encontrar_horizontal(char **matriz,char *palabra, int n){
    int cont = 0,tope = strlen(palabra),o = 0;
    for(int i = 0;palabra[i]!='\0';++i){
        palabra[i] = toupper(palabra[i]);
    }
    coord *a = (coord*)malloc(sizeof(coord));
    char busqueda[tope];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if((j+tope-cont) > n){
                j=n;
            }
            if(matriz[i][j] == palabra[0]){
                for(int u = 0; u < tope; u++){
                    busqueda[u] = matriz[i][j+u];
                }
                busqueda[tope] = '\0';
                if(strcmp(busqueda,palabra) == 0){
                    a->row = i+1;
                    a->col = j+1;
                    return a;
                }
            }
        }
    }
}
```

```
coord *encontrar_horizontal_2(char **matriz,char *palabra, int n){
    int cont = 0,tope = strlen(palabra);
    for(int i = 0;palabra[i]!='\0';++i){
        palabra[i] = toupper(palabra[i]);
    }
    coord *a = (coord*)malloc(sizeof(coord));
    for(int i = 0;i < n;i++){
        for(int j = 0;j < n;j++){
            if((j+tope-cont) > n){
                j=n;
            }
            if(matriz[i][j] == palabra[cont]){
                cont++;
                if(cont == tope){
                    a->row = i+1;
                    a->col = j+1;
                    return a;
                }
            }else{
                cont = 0;
            }
        }
    }
}
```

El tipo de dato devuelto no es más que una estructura para guardar las coordenadas de donde se encontró la palabra.

Creemos que la 2da forma es más eficiente porque solo se cuenta el número de ocurrencias sucesivas y si la letra no corresponde el contador solo se resetea, pero de la primera forma se toma la palabra entera para compararla induciendo así un ciclo menor pero significativo, si es que el número de veces que se entra a ese pequeño ciclo es alto.

5. Nuevos Tiempos de Ejecución

Palabra	Tiempo en Búsqueda	Orientación	Clocks
Carne	0.000030 [s]	Vertical	30
Gamer	0.000024 [s]	Vertical	24
Gato	0.000003 [s]	Vertical	3
Hola	0.000012 [s]	Vertical	12
Jamón	0.000025 [s]	Vertical	25
Carro	0.000014 [s]	Vertical	14
Banco	0.000100 [s]	Horizontal	100
Casa	0.000001 [s]	Horizontal	1
Cobre	0.000080 [s]	Horizontal	80
Perro	0.000004 [s]	Horizontal	4
Viktor	0.000021 [s]	Horizontal	21
Tapia	0.000005 [s]	Horizontal	5

6. Solución al problema

La materia del curso que da una solución a este problema se encuentra en la parte de Rendimiento dado que sabemos que la cantidad de instrucciones afecta en el tiempo de ejecución a través del Tiempo de CPU, por lo que, si se realiza un código que haga más entradas al ciclo, se tendrán más instrucciones y no será lo más eficiente posible, por lo que se debe evitar esto.

Conclusión:

En conclusión, este informe presenta los resultados de un programa que realiza búsquedas de palabras tanto en forma horizontal como vertical. Se utilizó la función clock de la librería time.h para medir el tiempo de ejecución y se encontró que las búsquedas se realizan en cuestión de milisegundos. Además, se presentan los tiempos de ejecución para cada palabra buscada, lo que permite identificar cuáles palabras tomaron más tiempo en ser encontradas. En general, los resultados muestran que el programa es eficiente y rápido en la búsqueda de palabras.