Análisis del rendimiento de un programa y las aplicaciones de las herramientas dispuestas por los sistemas operativos

Nombre: Sebastián Arrieta rol: 202173511-9

Nombre: Jonathan Olivares rol: 202073096-2

Preguntas de análisis

1)

Palabra	Tiempo [s]	Orientación	Tamaño
casa	0,000295	Horizontal	50x50
hola	0,000306	Vertical	50x50
perro	0,000622	Horizontal	50x50
gato	0,000654	Vertical	50x50
carro	0,001193	Vertical	100x100
jamon	0,001457	Vertical	100x100
gamer	0,001710	Vertical	100x100
viktor	0,001790	Horizontal	100x100
tapia	0,002179	Horizontal	100x100
cobre	0,005662	Horizontal	200x200
banco	0,006098	Horizontal	200x200
carne	0,012913	Vertical	200x200

2)

La palabra que tuvo un mayor tiempo de ejecución fue carne, esto debido a 2 puntos importantes. El primero es que el tamaño es el más grande posible (200x200) por lo tanto la búsqueda será mayor que la de los otros tamaños. El segundo punto es la ubicación de la palabra carne en la sopa de letras, mientras más al final se encuentre más demorará el programa en encontrar la palabra.

3)

En promedio la orientación horizontal tiene un menor tiempo de ejecución con 0,002774s mientras que la vertical tiene un tiempo de ejecución en promedio de 0,003039s

Tomando en cuenta la forma en la que se estructuro el código se puede observar que la lógica utilizada en ambas orientaciones es la misma por lo que no se espera alguna diferencia significante en los tiempos de ejecución entre los dos tipos de orientación, aunque, en algunas palabras si se observa un

tiempo mayor, por ejemplo “carne” es vertical y es el que más tiempo consume, pero esto se puede deber a la ubicación de la palabra en la sopa de letras más que a la orientación en sí.

4)

```
int palabraEncontrada(char *p, char *orientacion, int dimension,FILE *dfile){
    char *palabra = (char*)malloc(strlen(p)* sizeof(char));
    strcpy(palabra, p);

    int i = 0;
    while (palabra[i]) {
        palabra[i] = toupper(palabra[i]);
        i++;
    }
    /*char **matriz = (char **)malloc(dimension * sizeof(char *));

    for (int i = 0; i < dimension; i++) {
        matriz[i] = (char *)malloc(dimension * sizeof(char));
    }*/
    bool res;
    if(*orientacion == 'v'){
        res = buscarVertical(palabra, dimension, dfile);
    }else{
        res = buscarHorizontal(palabra, dimension, dfile);
    }
    return res;
}
```

El primer aspecto por resaltar se encuentra en la función palabraEncontrada en la cual se buscó ahorrar el tiempo utilizado en la creación de la matriz y todos los subarrays que esta constituye borrando la variable matriz y eliminando la función crearMatriz, en cambio para la búsqueda de la palabra se crearon dos nuevas funciones según la orientación indicada (buscarVertical y buscarHorizontal.

```
int buscarVertical(char *palabra, int dimension, FILE *dfile){
    int size =  strlen(palabra);

    for (int i = 0; i < dimension; i++){
        fseek(dfile, 2*i + 10, SEEK_SET);
        char c = fgetc(dfile);
        int cont = 0;
        while(c != '\n' && c != EOF){
            if(palabra[cont] == c){
                cont++;
            }else{
                cont=0;
            }

            if(cont == size){
                return true;
            }
            fseek(dfile, dimension*2, SEEK_CUR);
            c = fgetc(dfile);
        }
    }

    return false;
}
```

```
int buscarHorizontal(char *palabra, int dimension, FILE *dfile){
    int size =  strlen(palabra);

    for (int i = 0; i < dimension; i++){
        char c = fgetc(dfile);
        int cont = 0;
        while(c != '\n' && c != EOF){
            if(palabra[cont] == c){
                cont++;
            }else{
                cont=0;
            }

            if(cont == size){
                return true;
            }
            fseek(dfile, 1, SEEK_CUR);
            c = fgetc(dfile);
        }
    }

    return false;
}
```

Estas imágenes corresponden a las dos funciones añadidas las cuales en vez de recorrer una matriz analiza el archivo directamente, la búsqueda horizontal se ejecuta con la función getc de la manera secuencial dispuesta en el archivo, en cuanto a la búsqueda vertical la función se enfoca en la utilización del mismo getc junto con la función fseek para realizar un recorrido vertical en la sopa de letras y así facilitar la búsqueda.

5)

Palabra	**Tiempo [s]**	**Orientación**	**Tamaño**
casa	0,000007	Horizontal	50x50
hola	0,000298	Vertical	50x50
perro	0,000141	Horizontal	50x50
gato	0,000065	Vertical	50x50
carro	0,000442	Vertical	100x100
Jamón	0,000467	Vertical	100x100
Gamer	0,000743	Vertical	100x100
viktor	0,001178	Horizontal	100x100
tapia	0,000113	Horizontal	100x100
Cobre	0,004033	Horizontal	200x200
banco	0,004607	Horizontal	200x200
Carne	0,001580	Vertical	200x200

6) La materia del curso que podría solucionar este problema de rendimiento y aumentarlo son las hebras o la programación multihilos (multithreading). Si bien es cierto que aún no se ve esta materia,

se ha mencionado en varias ocasiones su propósito y lo que hace. Con la programación multihilos se puede dividir la sopa de letras haciendo que en paralelo se busquen la primera mitad de líneas en 1 hilo y la otra mitad en otro, o incluso dividiendo la sopa de letras en más partes, de esta forma el tiempo de ejecución de reduce considerablemente.

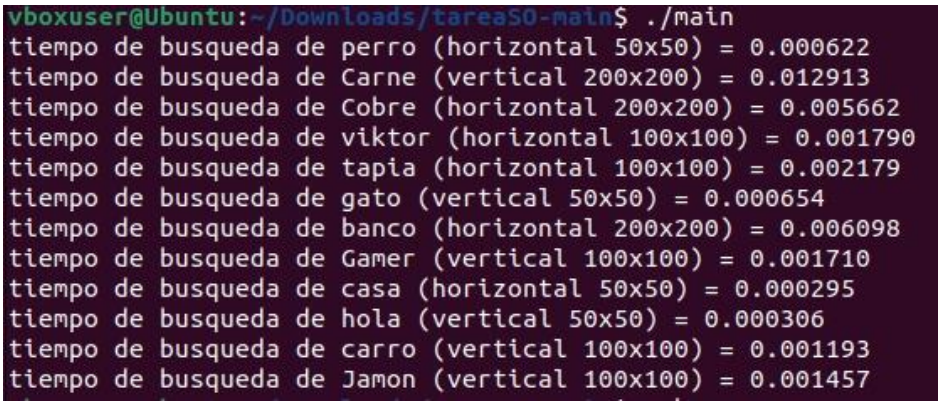
Conclusión

En conclusión, se puede observar un cambio considerable en los tiempos al combinar el input y output con la búsqueda. Ya que en el primer caso para las funciones se necesitaba leer y luego pasar a la matriz para comenzar la búsqueda, en el caso optimizado se realiza la búsqueda mientras va leyendo, de esta manera se aprovechan mejor los tiempos y se logra optimizar el programa. Se logra observar que la ubicación en la que se encuentre la palabra tiene un gran impacto. En nuestro caso el código posee un cortocircuito en el que al encontrar la palabra ya retorna "true", si no tuviera ese retorno y lo dejásemos, notaríamos unos tiempos muy similares para cada tamaño.

De todas formas, la mejor opción con el conocimiento que se tiene es usar multihilos, para lograr dividir el problema y trabajarlos en partes más pequeñas.

Se adjunta las imágenes de los tiempos del código.

Sin optimizar:



Optimizado: