UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

“Optimización de búsqueda en sopa de letras y manejo de directorios”

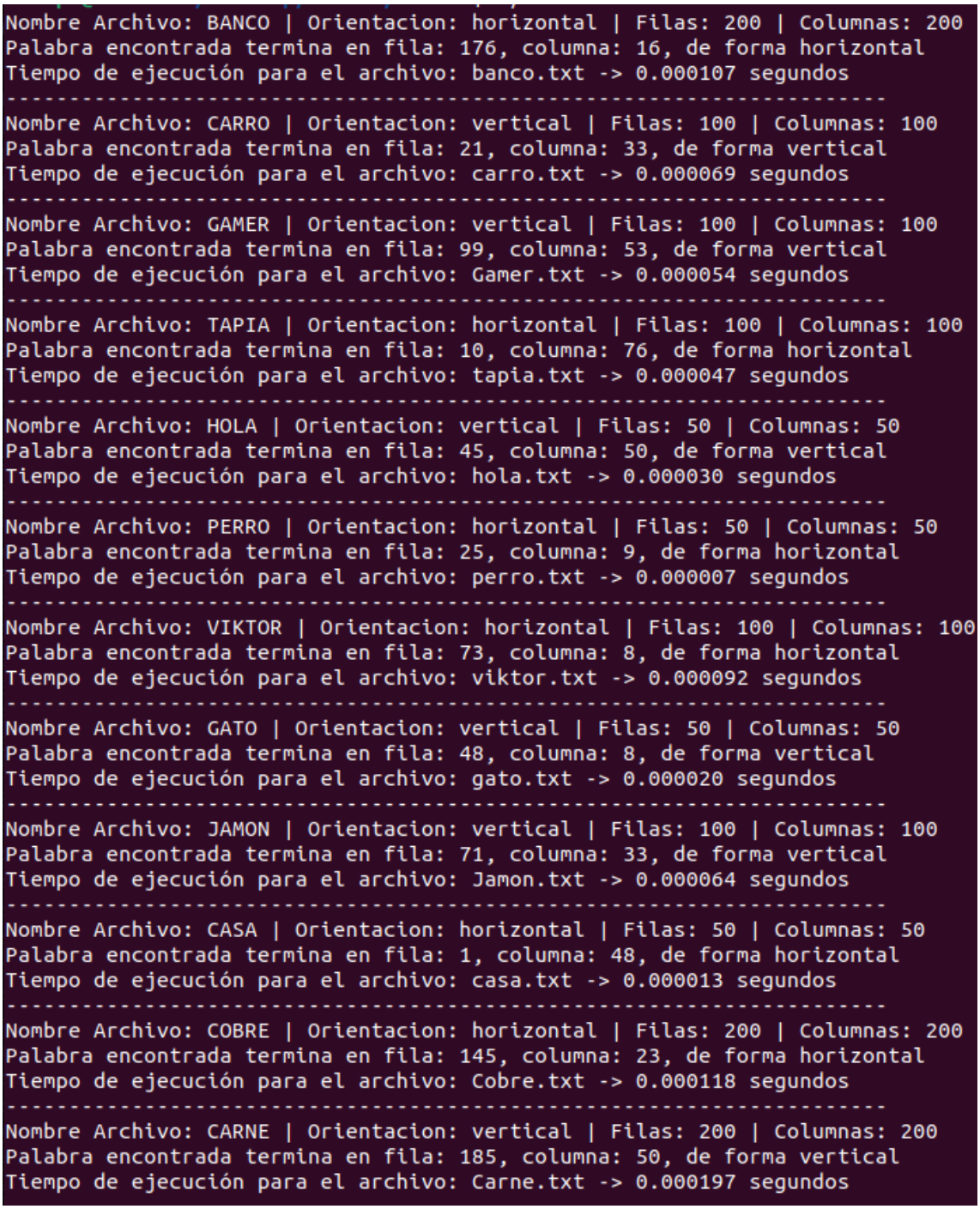
Benjamín López 202173533-k

Felipe León 202173598-4

Desarrollo de Preguntas y Análisis:

1. Cree una tabla con los tiempos de ejecución de cada palabra y su respectiva orientación.

En la siguiente imagen correspondiente a la terminal de Linux, se muestra por pantalla el tiempo de ejecución correspondiente a cada archivo de prueba:

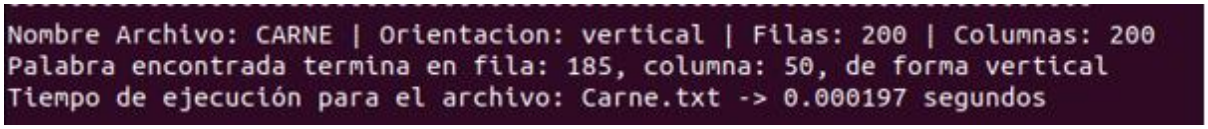


Con los tiempos de ejecución obtenidos, se presenta la siguiente tabla donde se puede encontrar la orientación y el tiempo correspondiente de cada palabra

Palabra	Tiempo de Ejecución (segundos)	Orientación
Banco	0.000107	Horizontal
Carro	0.000069	Vertical
Gamer	0.000054	Vertical
Tapia	0.000047	Horizontal
Hola	0.000030	Vertical
Perro	0.000007	Horizontal
Viktor	0.000092	Horizontal
Gato	0.000020	Vertical
Jamón	0.000064	Vertical
Casa	0.000013	Horizontal
Cobre	0.000118	Horizontal
Carne	0.000197	Vertical

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

De la tabla se puede observar que la palabra que más se demoró el código en buscar es la palabra Carne con 0.000197 segundos, como se muestra en la siguiente imagen:



Esta palabra se demoró más en buscar en comparación con las otras debido a dos principales razones, la primera es que la orientación que se obtuvo de la primera fila del archivo es vertical y nosotros para poder buscar una palabra con esta orientación, primero se guarda la sopa de letras en una matriz (esto se realiza para ambas orientaciones), pero al trabajar con una orientación vertical primero se debe realizar la transpuesta de la matriz para luego utilizar la misma función que se utiliza para buscar la palabra de forma horizontal (la función busqueda_horizontal()). La transpuesta se realiza con dos ciclos for anidados, de la siguiente forma:

```
// Primero, hacemos la matriz transpuesta
for (int i = 0; i < filas; i++){
    for (int j = 0; j < columnas; j++){
        transpuesta[i][j] = matriz[j][i];
    }
}
```

Esto agrega más tiempo de ejecución comparado con la orientación horizontal. La segunda causa de esto se debe a donde se ubica la palabra en la sopa de letras, como se puede observar, esta palabra fue encontrada en la fila 185 y columna 50, teniendo que recorrer casi todas las filas de la sopa de letra para poder ser localizada, por lo que se obtuvo un mayor tiempo de ejecución comparado con las demás.

Cabe destacar que cuando se busca de forma horizontal con la función busqueda_horizontal() (y también es utilizada para la búsqueda en forma vertical como fue mencionado anteriormente), se va recorriendo la sopa de letras que se encuentra guardada en la matriz fila por fila, por lo que encontrar la palabra en las primeras filas es fundamental para obtener un bajo tiempo de ejecución.

3. ¿Qué orientación tuvo un menor tiempo de ejecución? ¿A qué se debe esto?

Para responder esta pregunta, nos es conveniente obtener el promedio del tiempo de ejecución para cada orientación, el cual se presenta en la siguiente tabla

Orientación	Promedio tiempo de ejecución
Horizontal	0.000064
Vertical	0.0000723333…

En general, tuvo un mayor tiempo de ejecución la orientación vertical dado que lo que hace el programa en esta orientación es hacer la matriz traspuesta a la matriz original, para así buscar la palabra de forma horizontal ya que demora menos tiempo de esta forma, pero existe otro factor que es el tamaño de la matriz y en que fila y columna se encuentra la palabra, dado que si encuentra la

palabra en un fila más cercana del comienzo obviamente demorará menos tiempo que si encuentra la palabra en un fila casi al final de la sopa de letras.

4. ¿Como podría optimizar su código de forma que pueda minimizar sus tiempos de ejecución? Realice el código.

Lo primero que se cambió para poder optimizar los tiempos de ejecución, fue la forma en que se copiaba el contenido del archivo .txt al archivo creado en el directorio pedido en la tarea, a continuación, se muestra que en un comienzo se copiaba carácter por carácter, y para mejorar el tiempo de ejecución, se cambió a que se copiara línea por línea, ahorrando una buena cantidad de tiempo.

Antes:

```
char c;
while ( (c = fgetc(fp)) != EOF ){
    fputc(c, copia);
}
```

Después:

```
char linea[410];
while (fgets(linea, sizeof(linea), fp) != NULL) {
    fputs(linea, copia);
}
```

Otra parte que cambiamos fue la forma de buscar horizontalmente, ya que en un comienzo lo que se hizo fue que, al encontrar el primer carácter de la palabra se hacía un arreglo de caracteres de la misma longitud de la palabra original con las letras que seguían a la primera encontrada, y con esto comparábamos este arreglo con la palabra.

Lo que se hizo para optimizar esta parte del programa fue comparar carácter por carácter con la palabra original, esto se hacía de forma iterativa, dado que si encontraba coincidencia con el primer carácter de la palabra original se aumentaba en 1 un contador, pero si la siguiente letra no coincidía con la letra de la palabra original, este contador volvía a 0, de forma que cuando el contador sea igual a la longitud de la palabra, esto significaba que fue encontrada en la sopa de letras.

Antes:

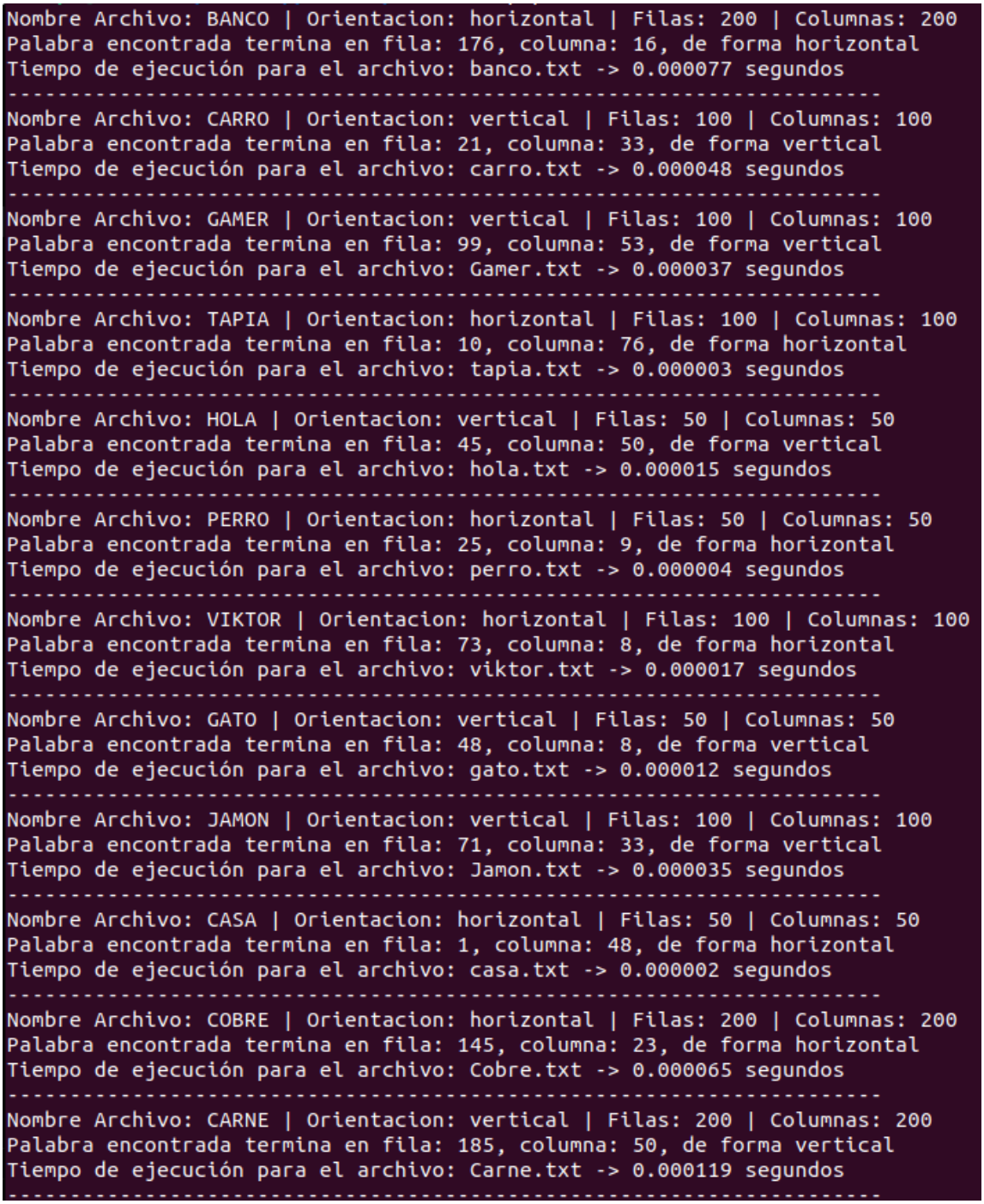
```c
void buscar_horizontal(char matriz[200][200], int filas, int columnas, char palabra_buscar[256], int horizontal) {
    int longitud = strlen(palabra_buscar);
    char palabra_comparar[longitud];
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++){
            if ( matriz[i][j] == palabra_buscar[0] ){
                if ( j + longitud <= columnas) { // Para asegurar que no nos salgamos de la fila
                    for (int k = 0; k < longitud; k++){
                        palabra_comparar[k] = matriz[i][j+k];
                    }
                    palabra_comparar[longitud] = '\0'; // Se agrega el carácter nulo al final
                    if (strcmp(palabra_comparar,palabra_buscar) == 0){
                        if (horizontal)
                            printf("Palabra encontrada termina en fila: %d, columna: %d, de forma horizontal\n",i+1,j+longitud);
                        else
                            printf("Palabra encontrada termina en fila: %d, columna: %d, de forma vertical\n",j+longitud,i+1);
                        }
                }
            }
        }
    }
```

Después:

```c
void buscar_horizontal(char matriz[200][200], int filas, int columnas, char palabra_buscar[256], int horizontal) {
    int longitud = strlen(palabra_buscar);
    int k = 0;
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++){
            if ( matriz[i][j] == palabra_buscar[k] ){
                //printf("sopa: %c\n",matriz[i][j]);
                //printf("palabra: %c\n",palabra_buscar[k]);
                k++;
            }
            else
                k = 0;
            if (k == longitud){
                if (horizontal)
                    printf("Palabra encontrada termina en fila: %d, columna: %d, de forma horizontal\n",i+1,j+1);
                else
                    printf("Palabra encontrada termina en fila: %d, columna: %d, de forma vertical\n",j+1,i+1);
                return;
            }
        }
    }
}
```

5. Cree una tabla con los nuevos tiempos de ejecución de cada palabra y su respectiva orientación utilizando el código realizado en la pregunta anterior.

Los nuevos tiempos de ejecución con el código optimizado se muestran por pantalla en la terminal de Linux en la siguiente imagen:



Estos tiempos se presentan en la siguiente tabla:

Palabra	Tiempo de Ejecución	Orientación
Banco	0.000077	Horizontal
Carro	0.000048	Vertical
Gamer	0.000037	Vertical
Tapia	0.000003	Horizontal
Hola	0.000015	Vertical
Perro	0.000004	Horizontal
Viktor	0.000017	Horizontal
Gato	0.000012	Vertical
Jamón	0.000035	Vertical
Casa	0.000002	Horizontal
Cobre	0.000065	Horizontal
Carne	0.000119	Vertical

Comparando con la tabla anterior, se puede observar que en todas las palabras el tiempo de ejecución disminuyó, por lo que el código fue optimizado correctamente para obtener una mayor rapidez.

6. ¿Qué materia del curso crees que podría ayudar a solucionar este problema? Justifique.

La materia del curso que nos puede ayudar a solucionar esto es el paralelismo (trabajar a través de hebras), que consiste en dividir una tarea en subtareas, con esto se pueden dar la posibilidad de dividir el programa en partes para que funcione de manera paralela y así poder optimizar su funcionamiento, por ejemplo, podríamos procesar varios archivos en paralelo para poder obtener un menor tiempo de ejecución. Un ejemplo de una librería en lenguaje C que permite esto es “Pthreads”. Se puede hacer uso de esta librería para dividir la tarea de procesar estos archivos en múltiples hebras. Cada hebra se encargaría de procesar un archivo específico. Esto se hace dividiendo la lista de archivos en partes iguales y asignando a cada hebra una parte de la lista. Todo esto permitiría mejorar la eficiencia del programa reduciendo los tiempos de ejecución para la búsqueda de la palabra.

Conclusión

A través del código realizado, se pudo determinar los tiempos de ejecución de diferentes palabras en distintas orientaciones. Se observó que, tras la optimización del código, hubo una disminución en los tiempos de ejecución, lo que indica que las mejoras implementadas fueron efectivas.

Además, se identificó que el concepto de paralelismo, específicamente la división de tareas en subtareas y el procesamiento en paralelo, podría ofrecer soluciones adicionales para mejorar la eficiencia del programa. La posibilidad de procesar varios archivos simultáneamente podría reducir aún más los tiempos de ejecución, optimizando la búsqueda de palabras en las sopas de letras proporcionadas.

En resumen, el informe destaca la importancia de la optimización del código y sugiere que, con la implementación de técnicas avanzadas como el uso de hebras, se pueden lograr mejoras significativas en velocidad de los programas.