Informe Laboratorio 1

Integrantes

Maximiliano Bardi, RUT: 21.030.899-7, ROL: 202173510-0
Bastian Ortega, RUT: 21.266.187-2, ROL: 202173547-K

Preguntas y análisis

Tabla de los tiempos de ejecución (en segundos)

Nombre	Orientación	Tiempo 1	Tiempo 2	Promedio
banco	Horizontal	0.003924	0.004100	0.004012
viktor	Horizontal	0.000931	0.001022	0.0009765
Cobre	Horizontal	0.003778	0.004098	0.003938
Jamon	Vertical	0.001111	0.001038	0.0010745
casa	Horizontal	0.000244	0.000346	0.000295
tapia	Horizontal	0.001012	0.001166	0.001089
gato	Vertical	0.000322	0.000303	0.0003125
Gamer	Vertical	0.001055	0.001309	0.001182
hola	Vertical	0.000251	0.000256	0.0002535
Carne	Vertical	0.004038	0.004178	0.004108
perro	Horizontal	0.000288	0.000271	0.0002795
carro	Vertical	0.001019	0.001029	0.001024

Promedio tiempos Vertical: 0.00132575

Tiempo total para Vertical: 0.0079545

Promedio tiempo Horizontal: 0.001765

Tiempo total para Horizontal: 0.01059

Palabra con mayor tiempo de ejecución: Carne

Debido a que puede variar un poco el tiempo de ejecución, decidimos medir 2 veces el tiempo y calcular el promedio. Como podemos observar, la palabra con mayor tiempo de ejecución fue “Carne”, esto es debido a que el código está estructurado de forma que tiene que leer letra por letra la sopa hasta encontrar la palabra, por lo que probablemente con “Carne” debió recorrer gran parte del archivo debido a la posición donde estaba, lo cual, justifica el tiempo de ejecución.

Orientación con menor tiempo de ejecución: Vertical

Analizando los tiempos obtenidos podemos notar que el menor tiempo se lo lleva la orientación vertical debido a que el factor posición de la palabra afecta directamente, esto quiere decir, que probablemente a Vertical le demoró menos tiempo encontrar la palabra debido a la posición de esta en la sopa, la cual estaría más cerca del inicio en comparación a las posiciones que tiene Horizontal. Además, el código Sopa contiene 2 funciones: Horizontal y Vertical, estas tienen el objetivo de encontrar la palabra, por ende, se mide el tiempo de ejecución de ellas, las cuales fueron creadas por cada uno de los integrantes, por ende, la naturaleza de los códigos es distinta. Esto puede afectar el tiempo de ejecución.

Cómo podríamos optimizar el código para minimizar sus tiempos de ejecución

Al realizar el código, se pudo observar una redundancia, la cual es la lectura de archivos, y es que Sopa lee y modifica el archivo, pero Horizontal y Vertical también lo hacen, provocando que el tiempo de ejecución para encontrar la palabra no sea óptimo ya que repite procesos innecesariamente.

Debido a esto, alteramos Horizontal y Vertical de forma que SOLO se dediquen a buscar la palabra en la sopa de letras (su función primordial), aumentando la cantidad de parámetros, pero disminuyendo la cantidad de procesos que debe hacer, disminuyendo la redundancia. Los códigos quedaron de la siguiente forma:

Vertical

```
char* Vertical(char letras[],char lista[], int largo,int contador) {
char lista_2[largo];
    for(int i=0; i<strlen(lista);i++){
        lista_2[i]=toupper(lista[i]);
    }
    bool flag = true;
    int letra = 8;
    int palabra = 0;
    int comprueba = 1;
    int correcto = 1;
    int len_sopa = round(sqrt(contador - 8));

    while (flag) {
        if (letras[letra] == lista_2[palabra]) {
            bool flag2 = true;
            while (flag2) {
                if (letras[letra + len_sopa * comprueba] == lista_2[comprueba]) {
                    comprueba++;
                    correcto++;
                    if (correcto == sizeof(lista_2)) {
                        printf("%s","Palabra encontrada.\n");
                        printf("Palabra:%s\nOrientacion: vertical\n", lista);
                        flag2 = false;
                        flag = false;
                    }
                } else {
                    comprueba = 1;
                    correcto = 1;
                    letra++;
                    flag2 = false;
                }
            }
        } else {
            letra++;
        }
    }
    return 0;
};
```

Horizontal

```
char* Horizontal(char letras[],char palabra[],int contador){
    for(int i=0; i<strlen(palabra);i++){
        palabra[i]=toupper(palabra[i]);
    }
    char* resultado= strstr(letras,palabra);
    if (resultado!=NULL){
        printf("Palabra encontrada.\nPalabra: %s\nOrientacion: Horizontal\n", palabra);
    }
    return 0;
};
```

Tabla de los tiempos de ejecución con el código mejorado

Nombre	Orientación	Tiempo 1	Tiempo 2	Promedio
banco	Horizontal	0.000023	0.000021	0.000022
viktor	Horizontal	0.000074	0.000011	0.0000425
Cobre	Horizontal	0.000016	0.000018	0.000017
Jamon	Vertical	0.000024	0.000072	0.000048
casa	Horizontal	0.000008	0.000021	0.0000145
tapia	Horizontal	0.000039	0.000019	0.000029
gato	Vertical	0.000016	0.000017	0.0000165
Gamer	Vertical	0.000044	0.000034	0.000049
hola	Vertical	0.000012	0.000014	0.000013
Carne	Vertical	0.000098	0.000099	0.0000985
perro	Horizontal	0.000011	0.000012	0.0000115
carro	Vertical	0.000014	0.000016	0.000015

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

A nivel general considerando que se analizan múltiples archivos nos ayudaría aplicar paralelismo para así lograr ejecutar 2 instrucciones al mismo tiempo. Mejorando el tiempo total del objetivo de la tarea. Además, reduciendo procesos a realizar también genera beneficios en cuanto al tiempo de ejecución.

Conclusión

Dependiendo de cómo está hecho el código, este se puede ejecutar con cierta cantidad de tiempo, y hay que intentar reducirlo lo máximo posible. Con este trabajo también se puede concluir que no necesariamente una función tiene constantemente el mismo tiempo de ejecución, esto lo podemos ver en las tablas, donde, dependiendo de la ubicación de la palabra en la sopa de letras, las funciones Horizontal y Vertical se demoran “x” cantidad de tiempo, incluso buscar la misma palabra en la misma sopa de letras no tendrá siempre el mismo tiempo de ejecución (se puede notar en el tiempo 1 y tiempo 2 de las palabras). Lo último que se puede concluir del trabajo es que a veces, al realizar un código, no necesariamente está realizado de la mejor forma, y que como programadores es importante “corregir” o “revisar” los códigos que se hacen, intentado eliminar redundancia y otros elementos que podrían afectar el tiempo de ejecución.