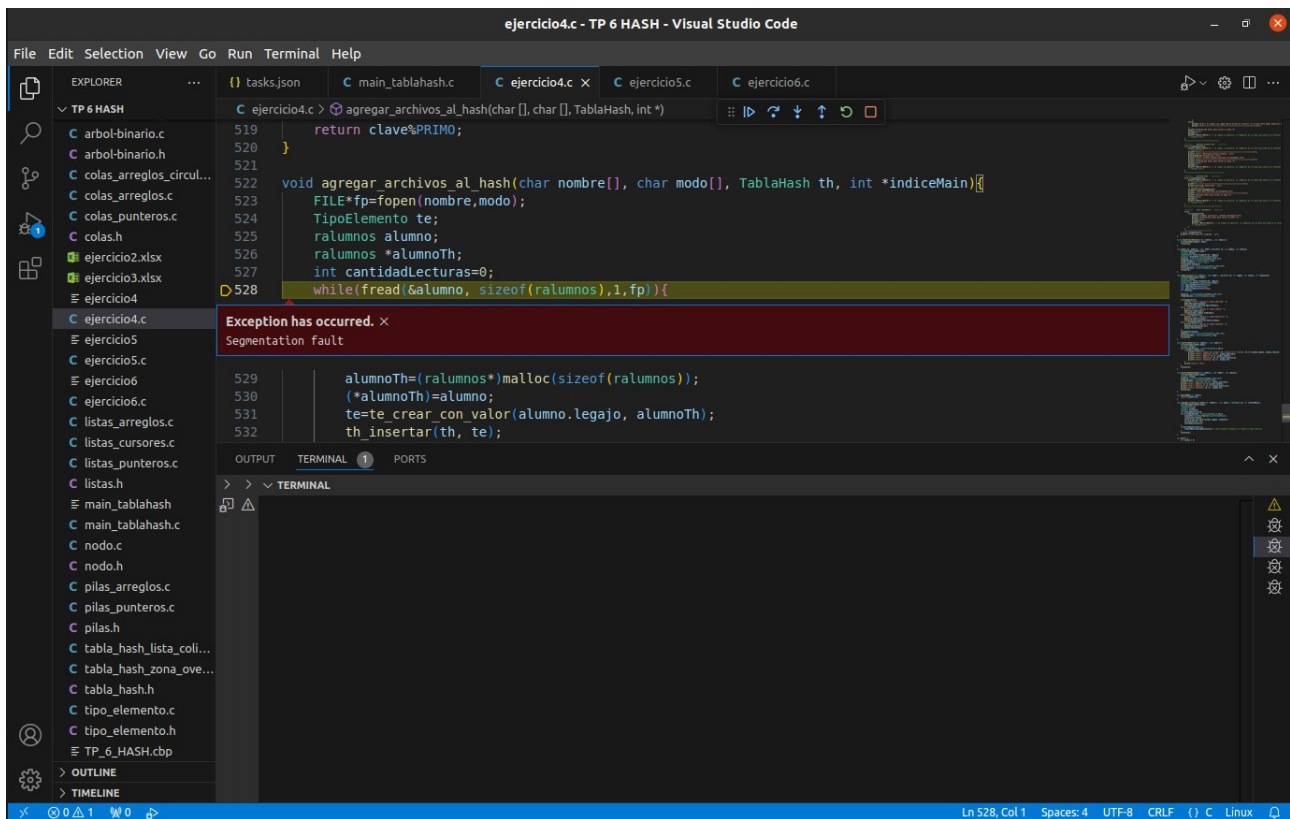


RESULTADO DE LA CORRECCIÓN: **DESAPROBADO**

OBSERVACIONES

En el ejercicio 4 intenta abrir un archivo .dat que no existe y no da la posibilidad de crearlo. En el ejercicio 5 no controla si la clave está repetida. No valida rango de valores a generar con la cantidad, con lo cual pueden repetirse valores. No imprimen las estructuras así que es imposible saber cuántas claves tiene realmente cada una de ellas. En el ejercicio 6 no muestra las estructuras así que no se puede saber cómo manejan las posibles colisiones



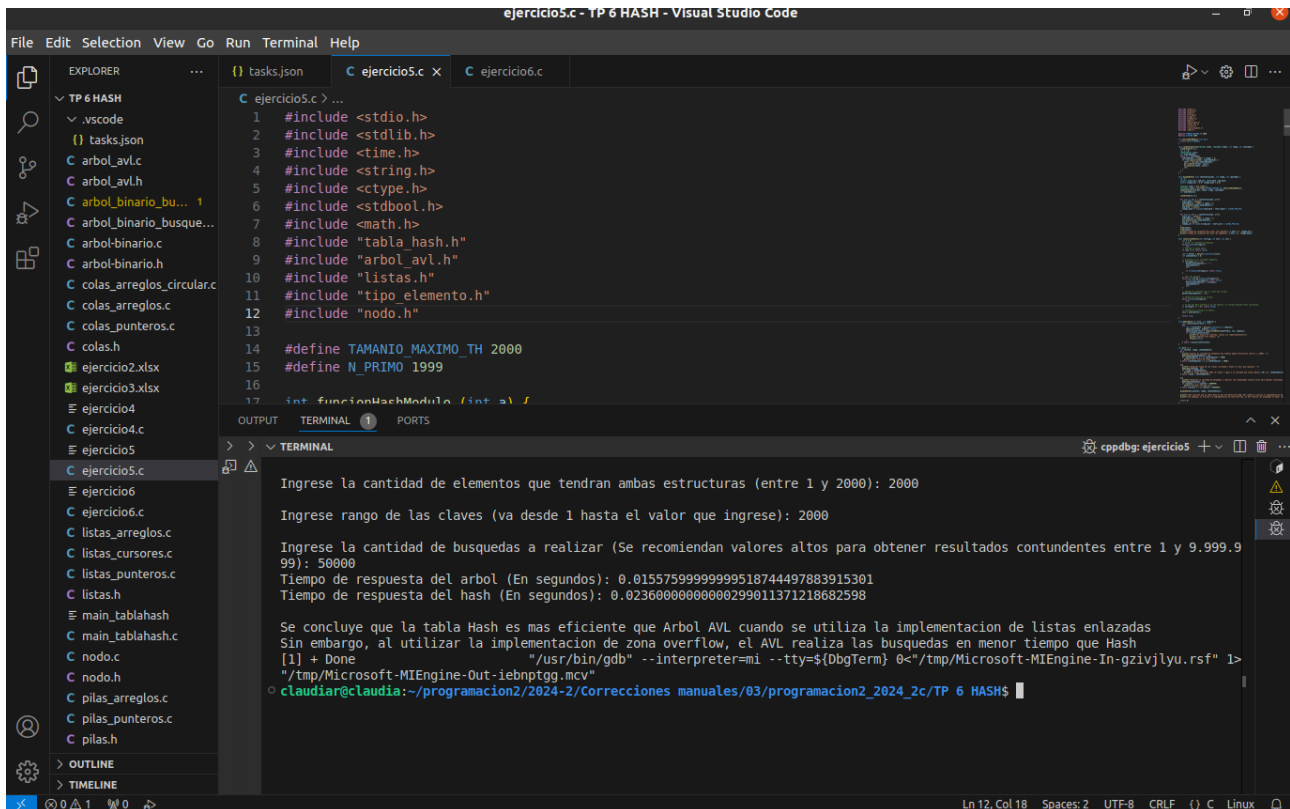
The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying a project named 'TP 6 HASH'. The file 'ejercicio4.c' is selected and open in the editor. The code in the editor shows a function 'agregar_archivos_al_hash' that attempts to open a file 'archivo.dat' in 'a' mode. A red error banner indicates a 'Segmentation fault' at line 528, which is a 'while' loop that reads from 'archivo.dat'. The terminal at the bottom is empty, and the status bar at the bottom indicates 'Ln 528, Col 1'.

```
File Edit Selection View Go Run Terminal Help
EXPLORER
TP 6 HASH
  C arbol-binario.c
  C arbol-binario.h
  C colas_arreglos_circul...
  C colas_arreglos.c
  C colas_punteros.c
  C colas.h
  C ejercicio2.xlsx
  C ejercicio3.xlsx
  C ejercicio4
  C ejercicio4.c
  C ejercicio5
  C ejercicio5.c
  C ejercicio6
  C ejercicio6.c
  C listas_arreglos.c
  C listas_cursoros.c
  C listas_punteros.c
  C listas.h
  C main_tablahash
  C main_tablahash.c
  C nodo.c
  C nodo.h
  C pilas_arreglos.c
  C pilas_punteros.c
  C pilas.h
  C tabla_hash_lista_coli...
  C tabla_hash_zona_ove...
  C tabla_hash.h
  C tipo_elemento.c
  C tipo_elemento.h
  C TP_6_HASH.cbp
  > OUTLINE
  > TIMELINE

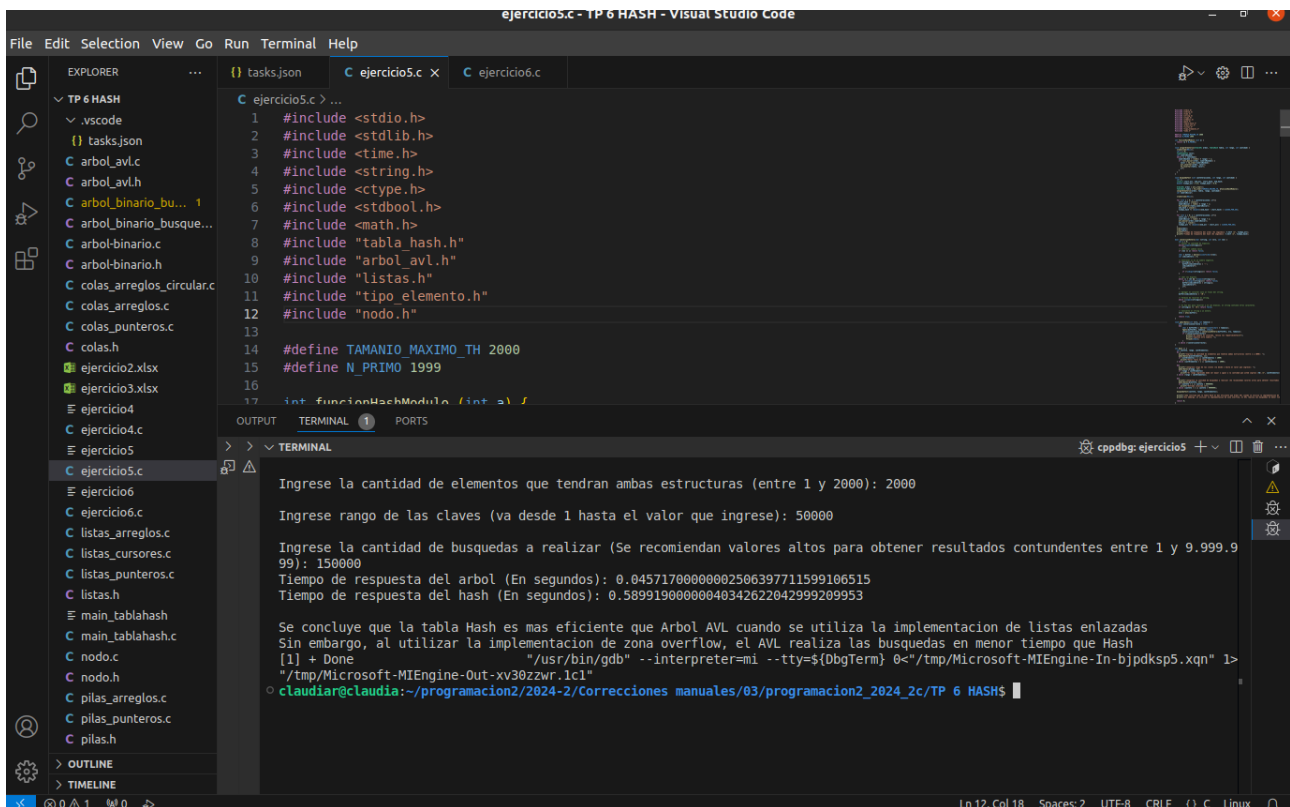
ejercicio4.c
519
520
521
522
523
524
525
526
527
528
529
530
531
532

void agregar_archivos_al_hash(char nombre[], char modo[], TablaHash th, int *indiceMain){
    FILE*fp=fopen(nombre,modo);
    TipoElemento te;
    ralumnos alumno;
    ralumnos *alumnoTh;
    int cantidadLecturas=0;
    while(fread(&alumno, sizeof(ralumnos),1,fp)){
        alumnoTh=(ralumnos*)malloc(sizeof(ralumnos));
        (*alumnoTh)=alumno;
        te=te_crear_con_valor(alumno.legajo, alumnoTh);
        th_insertar(th, te);
    }
}
```

GRUPO 3 – Correcciones Trabajo Práctico: TABLAS HASH



```
ejercicio5.c - TP 6 HASH - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
TP 6 HASH
tasks.json
arbol_avl.c
arbol_avl.h
arbol_binario_bu...
arbol_binario_busque...
arbol_binario.c
arbol_binario.h
colas_arreglos_circular.c
colas_arreglos.c
colas_punteros.c
colas.h
ejercicio2.xlsx
ejercicio3.xlsx
ejercicio4
ejercicio4.c
ejercicio5
ejercicio5.c
ejercicio6
ejercicio6.c
listas_arreglos.c
listas_cursor.c
listas_punteros.c
listas.h
main_tablahash
main_tablahash.c
nodo.c
nodo.h
pilas_arreglos.c
pilas_punteros.c
pilas.h
OUTLINE
TIMELINE
ejercicio5.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <string.h>
5 #include <ctype.h>
6 #include <stdbool.h>
7 #include <math.h>
8 #include "tabla_hash.h"
9 #include "arbol_avl.h"
10 #include "listas.h"
11 #include "tipo_elemento.h"
12 #include "nodo.h"
13
14 #define TAMANIO_MAXIMO_TH 2000
15 #define N_PRIMO 1999
16
17 int funcionHashModule (int a) {
    OUTPUT TERMINAL PORTS
    Ingrese la cantidad de elementos que tendran ambas estructuras (entre 1 y 2000): 2000
    Ingrese rango de las claves (va desde 1 hasta el valor que ingrese): 2000
    Ingrese la cantidad de busquedas a realizar (Se recomiendan valores altos para obtener resultados contundentes entre 1 y 9.999.999): 50000
    Tiempo de respuesta del arbol (En segundos): 0.0155759999999518744497883915301
    Tiempo de respuesta del hash (En segundos): 0.023600000000000299011371218682598
    Se concluye que la tabla Hash es mas eficiente que Arbol AVL cuando se utiliza la implementacion de listas enlazadas
    Sin embargo, al utilizar la implementacion de zona overflow, el AVL realiza las busquedas en menor tiempo que Hash
    [1] + Done "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-gzivjlyu.rsf" 1>
    "/tmp/Microsoft-MIEngine-Out-iebnptgg.mcv"
    claudiar@claudia:~/programacion2/2024-2/Correcciones manuales/03/programacion2_2024_2c/TP 6 HASH$
```



```
ejercicio5.c - TP 6 HASH - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
TP 6 HASH
tasks.json
arbol_avl.c
arbol_avl.h
arbol_binario_bu...
arbol_binario_busque...
arbol_binario.c
arbol_binario.h
colas_arreglos_circular.c
colas_arreglos.c
colas_punteros.c
colas.h
ejercicio2.xlsx
ejercicio3.xlsx
ejercicio4
ejercicio4.c
ejercicio5
ejercicio5.c
ejercicio6
ejercicio6.c
listas_arreglos.c
listas_cursor.c
listas_punteros.c
listas.h
main_tablahash
main_tablahash.c
nodo.c
nodo.h
pilas_arreglos.c
pilas_punteros.c
pilas.h
OUTLINE
TIMELINE
ejercicio5.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <string.h>
5 #include <ctype.h>
6 #include <stdbool.h>
7 #include <math.h>
8 #include "tabla_hash.h"
9 #include "arbol_avl.h"
10 #include "listas.h"
11 #include "tipo_elemento.h"
12 #include "nodo.h"
13
14 #define TAMANIO_MAXIMO_TH 2000
15 #define N_PRIMO 1999
16
17 int funcionHashModule (int a) {
    OUTPUT TERMINAL PORTS
    Ingrese la cantidad de elementos que tendran ambas estructuras (entre 1 y 2000): 2000
    Ingrese rango de las claves (va desde 1 hasta el valor que ingrese): 50000
    Ingrese la cantidad de busquedas a realizar (Se recomiendan valores altos para obtener resultados contundentes entre 1 y 9.999.999): 150000
    Tiempo de respuesta del arbol (En segundos): 0.04571700000002506397711599106515
    Tiempo de respuesta del hash (En segundos): 0.58991900000040342622042999209953
    Se concluye que la tabla Hash es mas eficiente que Arbol AVL cuando se utiliza la implementacion de listas enlazadas
    Sin embargo, al utilizar la implementacion de zona overflow, el AVL realiza las busquedas en menor tiempo que Hash
    [1] + Done "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-bjpdksps5.xqn" 1>
    "/tmp/Microsoft-MIEngine-Out-xv30zzwr.1cl"
    claudiar@claudia:~/programacion2/2024-2/Correcciones manuales/03/programacion2_2024_2c/TP 6 HASH$
```

GRUPO 3 – Correcciones Trabajo Práctico: TABLAS HASH

The screenshot shows the Visual Studio Code interface with the `tasks.json` file open in the editor. The file is located in the `TP 6 HASH` folder. The terminal output shows the execution of the tasks, including the input of the number of elements (2000) and the range of keys (50000). The output also displays the execution time for the hash table and the conclusion that the hash table is more efficient than the AVL tree.

```
.vscode > {} tasks.json > [ ] tasks > {} 0 > [ ] args
2
3   "tasks": [
4     {
5       "label": "TP 6 HASH",
6       "command": "gdb",
7       "args": [
8         "${file}",
9         "-o",
10        "${fileDirname}/${fileBasenameNoExtension}",
11        "${fileDirname}/nodo.c",
12        "${fileDirname}/tabla_hash_zona_overflow.c",
13        "${fileDirname}/tabla_hash_lista_colisiones.c",
14        "${fileDirname}/listas_punteros.c",
15        "${fileDirname}/arbol_avl.c",
16        "${fileDirname}/arbol_binario_busqueda.c",
17        "${fileDirname}/tipo_elemento.c",
18        "-lm",
19      ],
20      "type": "process",
21      "group": "test",
22    }
23  ],
24  "version": "2.0.0"
```

OUTPUT TERMINAL 1 PORTS

Ingrese la cantidad de elementos que tendran ambas estructuras (entre 1 y 2000): 2000
Ingrese rango de las claves (va desde 1 hasta el valor que ingrese): 50000
Ingrese la cantidad de busquedas a realizar (Se recomiendan valores altos para obtener resultados contundentes entre 1 y 9.999.999): 150000
Tiempo de respuesta del arbol (En segundos): 0.04571700000002506397711599106515
Tiempo de respuesta del hash (En segundos): 0.589919000000040342622042999209953

Se concluye que la tabla Hash es mas eficiente que Arbol AVL cuando se utiliza la implementacion de listas enlazadas Sin embargo, al utilizar la implementacion de zona overflow, el AVL realiza las busquedas en menor tiempo que Hash [1] + Done
"/usr/bin/gdb" --interpreter=mi --tty=\${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-bjpdksps5.xqn" 1>
"/tmp/Microsoft-MIEngine-Out-xv30zzwr.1c1"

claudiar@claudia:~/programacion2/2024-2/Correcciones manuales/03/programacion2_2024_2c/TP 6 HASH\$

The screenshot shows the Visual Studio Code interface with the `tasks.json` file open in the editor. The file is located in the `TP 6 HASH` folder. The terminal output shows the execution of the tasks, including the input of the number of elements (2000) and the range of keys (50000). The output also displays the execution time for the hash table and the conclusion that the hash table is more efficient than the AVL tree.

```
.vscode > {} tasks.json > [ ] tasks > {} 0 > [ ] args > 6
2
3   "tasks": [
4     {
5       "label": "TP 6 HASH",
6       "command": "gdb",
7       "args": [
8         "${file}",
9         "-o",
10        "${fileDirname}/${fileBasenameNoExtension}",
11        "${fileDirname}/nodo.c",
12        "${fileDirname}/tabla_hash_zona_overflow.c",
13        "${fileDirname}/tabla_hash_lista_colisiones.c",
14        "${fileDirname}/listas_punteros.c",
15        "${fileDirname}/arbol_avl.c",
16        "${fileDirname}/arbol_binario_busqueda.c",
17        "${fileDirname}/tipo_elemento.c",
18        "-lm",
19      ],
20      "type": "process",
21      "group": "test",
22    }
23  ],
24  "version": "2.0.0"
```

OUTPUT TERMINAL 1 PORTS

Ingrese la cantidad de elementos que tendran ambas estructuras (entre 1 y 2000): 2000
Ingrese rango de las claves (va desde 1 hasta el valor que ingrese): 50000
Ingrese la cantidad de busquedas a realizar (Se recomiendan valores altos para obtener resultados contundentes entre 1 y 9.999.999): 150000
Tiempo de respuesta del arbol (En segundos): 0.04583300000002508284291380391551
Tiempo de respuesta del hash (En segundos): 0.03566000000001479791622571724474

Se concluye que la tabla Hash es mas eficiente que Arbol AVL cuando se utiliza la implementacion de listas enlazadas Sin embargo, al utilizar la implementacion de zona overflow, el AVL realiza las busquedas en menor tiempo que Hash [1] + Done
"/usr/bin/gdb" --interpreter=mi --tty=\${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-3c00cdmd.trd" 1>
"/tmp/Microsoft-MIEngine-Out-eg4yivgx.snv"

claudiar@claudia:~/programacion2/2024-2/Correcciones manuales/03/programacion2_2024_2c/TP 6 HASH\$