CCC209 Q3

Proyecto: Codificación usando Árbol de Huffman

Entrega Miércoles Agosto 1

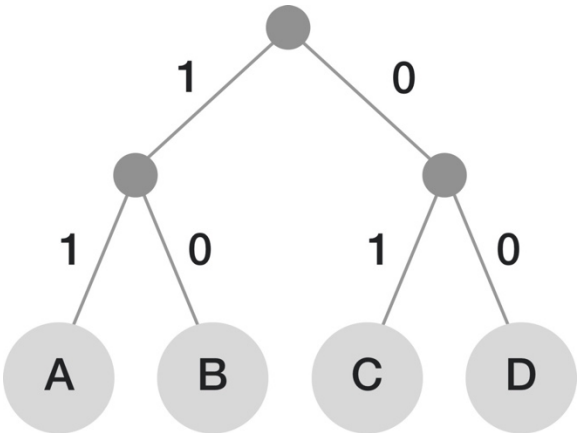
Objetivo: Su misión consiste en escribir un programa en C++ que lea un archivo de texto, construya un árbol de Huffman con él, y genere la codificación de cada carácter. Este proyecto le ayudará a practicar:

- Programación III
  - programación con múltiples archivos
  - clases
  - herencia
  - polimorfismo
  - memoria dinámica
- Estructura de Datos
  - implementación del TDA Árbol
  - Recorridos de árbol
  - Lectura de archivos
  - Uso de parámetros de línea de comando

Implementación

La tarea consiste a grandes rasgos en tres pasos:

1. Leer el archivo y crear una tabla de frecuencia
2. Usando la tabla de frecuencia crear el árbol de Huffman
3. Realizar el recorrido apropiado al árbol hasta llegar a cada hoja (que contiene los caracteres encontrados en el archivo) y generar la codificación binaria para cada carácter



Note que en esta figura se puede ver la codificación de cuatro caracteres A, B, C y D. Donde el código de A es 11, B es 10, C es 01 y D es 00.

En este proyecto el único requisito de programación es la implementación de la clase Árbol que se describe a continuación. El uso de esta clase es fundamental para el proyecto. El resto de clases o funciones quedará a su criterio.

Clase	Acceso	Miembro	Descripción
TreeNode	Private	`vector<TreeNode*> _children`	Sirve para almacenar la lista de hijos del nodo
	Private	`TreeNode* _parent`	Almacena el apuntador al padre
	Private	`TreeElement* _data`	Almacena la información del nodo, se recomienda que `TreeElement` tenga un `string element` y un `size_t frequency`.
	Public	`TreeNode(const TreeElement& data)`	Constructor que crea el nodo de árbol con los datos del parámetro
	Public	`TreeNode(const TreeElement& data, TreeNode* parent)`	Constructor que crea el nodo de árbol con los datos del primer parámetro y con padre `parent`
	Public	`~TreeNode()`	Destructor del nodo, recuerde liberar a los hijos si tiene.
	Public	`GetChildren(): vector<TreeNode*>&`	Recupera la referencia a la lista de hijos
	Public	`SetParent(TreeNode* parent): void`	Establece el padre del nodo
	Public	`AddChild(const TreeElement& data):void`	Agrega un hijo a la "izquierda" del último hijo agregado, usando el dato del parámetro
	Public	`AddChild(TreeNode* child): void`	Agrega un subárbol (`TreeNode*`) a la "izquierda" del último hijo agregado. Sirve para conectar árboles
	Public	`GetData(): TreeElement`	Recupera los datos almacenados en el nodo
	Public	`SetData(const TreeElement&): void`	Establece los datos a almacenar en el nodo
	Public	`IsRoot(): bool`	Verifica si un nodo es raíz
	Public	`IsLeaf(): bool`	Verifica si un nodo es hoja

Recuerde colocar correctamente los modificadores de métodos virtual y `const`. Revise el ejemplo en clase si tiene dudas.

Se le proveerán los siguientes archivos:

- `texto01.txt` Archivo 1 de prueba para generar el árbol
- `texto02.txt` Archivo 2 de prueba para general el árbol
- `output01.txt` Salida esperada de la codificación del árbol del archivo 1
- `output02.txt` Salida esperada de la codificación del árbol del archivo 2

Programa Principal (main.cpp)

Deberá crear un programa principal que reciba el nombre del archivo de línea de comando, así como se muestra en la figura. Note lo siguiente:

1. Cambie los siguientes caracteres no visibles
   a. 32 (Espacio) SP
   b. 10 (Salto de Línea) LF
   c. 13 (Enter) CR
2. Su programa deberá recibir el nombre de archivo de la línea de comando, si no escribe el nombre del archivo, o no es posible abrirlo deberá enviar un mensaje de error. EL NOMBRE DE ARCHIVO NO SE LEE CON UN cin, SE LEE DE LA LINEA DE COMANDO. Investigar: "C++ command line parameters".

```
ARIAS-CSC-MBP:ccc209-huffman arias$ make
make: `huffman' is up to date.
ARIAS-CSC-MBP:ccc209-huffman arias$ ./huffman
Not enough parameters
ARIAS-CSC-MBP:ccc209-huffman arias$ ./huffman test.txt
{key: SP, code: 11}
{key: a, code: 010}
{key: e, code: 0010}
{key: o, code: 0110}
{key: u, code: 0111}
{key: r, code: 1000}
{key: n, code: 1010}
{key: i, code: 1011}
{key: l, code: 00001}
{key: s, code: 00010}
{key: d, code: 00110}
{key: m, code: 10011}
{key: p, code: 000000}
{key: c, code: 000001}
{key: LF, code: 000110}
{key: t, code: 000111}
{key: g, code: 001111}
{key: q, code: 100100}
{key: b, code: 100101}
{key: h, code: 0011100}
{key: f, code: 00111010}
{key: z, code: 00111011}
ARIAS-CSC-MBP:ccc209-huffman arias$
```

Manejo de Errores:

Si el usuario no escribe el nombre de archivo como parámetro de línea de comando, o si el archivo no se puede abrir, el programa deberá reportar el error y terminar. Su programa no debe tener "fugas de memoria" (memory leaks), y no debe tener errores de tiempo de corrida (runtime errors).

Estilo de Programación

Su programa debe ser ordenado y de fácil lectura. Si está trabajando con un equipo de desarrollo y no escribe código limpio sus colegas no lo respetarán y posiblemente los saquen del equipo rápidamente. Esto implica que el código no tiene que tener grandes segmentos comentados, y que el código debe estar correctamente sangrado, entre otras cosas.

Todos los identificadores deben tener nombres significativos. Las funciones y clases seguirán el estándar CamelCase y las variables el estándar camelCase.

No utilice variables globales. Las variables globales a veces son apropiadas, pero no en los programas que desarrollará en esta clase.

Al inicio de los archivos deberá haber unos comentarios (header comments), por ejemplo:

```
// Titulo: Proyecto #2  Codificacion usando arbol de Huffman

//
// Proposito: crear un API que contiene todas las operaciones de lista usando listas
//            enlazadas
//
// Clase: CCC209 – Q3 – 2018
//
// Author: Luke Skywalker
```

Cada función deberá tener una breve descripción antes de su encabezado. Estos comentarios incluyen: qué hace la función, qué representa cada parámetro, qué retorna y cómo maneja los errores, por ejemplo:

```
// Inserta un elemento en la lista enlazada
// Params: Object* element, representa el elemento que se desea insertar
//         size_t position, representa la posición donde se desea insertar
// Returna: verdadero si la inserción fue exitosa, falso si no se pudo insertar
// Errores: si la posición es inválida la función termina retornando falso
```

Siempre es buena idea incluir comentarios adicionales en el código. Los mismos pueden ser breves. Si una variable no es autodescriptiva es buena idea colocar un comentario al lado de su declaración. Sea juicioso con la cantidad de comentarios, pues es posible poner demasiados comentarios.

Sea consistente en el estilo que utiliza para las llaves {}. Use el mismo estilo en todo y todos sus archivos.

Recuerde utilizar correctamente los modificadores `const`, y enviar las variables no-primitivas como parámetros por referencia. Los datos miembros de la clase deberán tener el prefijo _ (underscore)

Envío

Su solución debe estar contenida en al menos los archivos `treenode.h`, `treenode.cpp`, `makefile` y `main.cpp`

Antes de enviar:

1. Cree un directorio nuevo
2. Copie sus archivos en este directorio
3. Copie los archivos provistos en este directorio (`test01.txt`, `test02.txt`, `output01.txt`, `output02.txt`)
4. Abra una terminal (ventana de línea de comando)
5. Ubíquese en el directorio que acaba de crear
6. Ejecute: `make`
7. Asegúrese que todo compile correctamente
8. Ejecute el comando: `./huffman test01.txt > myoutput01.txt`
9. Ejecute el comando: `diff output01.txt myoutput01.txt`
10. No debe haber ninguna salida, lo cual indica que su salida es idéntica a la salida indicada en el archivo `output01.txt`
11. Haga lo mismo para `test02.txt`

```
ARIAS-CSC-MBP:ccc209-huffman arias$ ./huffman test01.txt > myoutput01.txt
ARIAS-CSC-MBP:ccc209-huffman arias$ diff output01.txt myoutput01.txt
ARIAS-CSC-MBP:ccc209-huffman arias$
```

Una vez pase todas las pruebas, está listo para enviar sus archivos.

¡NO MODIFIQUE NINGUNO DE LOS ARCHIVOS PROVISTOS!

Integridad Académica

Este proyecto debe ser desarrollado de manera individual. Ahora bien, se entiende que uno también aprende a través de las discusiones con los compañeros, lo cual es válido y se alienta. Está bien discutir con los compañeros, pero tenga claro que cada quien debe desarrollar su propio código para poder aprender correctamente. Por esta razón se establecen estas recomendaciones:

- Siéntase en libertad de discutir los proyectos y tareas, pero no permita que nadie mire o copie su código. No comparta su código ni impreso ni digital con nadie.
- No le niegue a otro compañero la oportunidad de desarrollar su lógica, está bien discutir, pero no resolver.
- Si luego de intentar sinceramente no logra resolver el problema, puede buscar a un estudiante avanzado de la carrera para que le apoye. Recuerde es apoyo, no solicitar que le hagan el trabajo.
- Si hay cualquier evidencia de que su programa u otra tarea asignada fue copiada de otro estudiante o de otra fuente, ni usted ni el(los) otro(s) estudiante(s) recibirán crédito por la asignación. **La nota será de cero y se enviará reporte a las autoridades académicas.**
- Protéjase y mantenga su código seguro.

Evaluación

En términos de ejecución, se revisará que su programa genere los mismos códigos y en el mismo orden que los que se le provee. Ahora bien, aun cuando su solución pase estas pruebas se tomará en cuenta:

- Seguir las instrucciones de diseño e instrucciones de entrega
- Seguir las convenciones de código
- Usar técnicas avanzadas no vistas en clase (y no poder explicarlas)
- Errores de programación no capturados
- Prueba con uno o más archivos diferentes de prueba