Trie

NOTA: Si usted está leyendo este documento sin haber extraído el compactado que se le entregó, ciérrelo ahora, extraiga todos los archivos en el escritorio, y siga trabajando desde ahí. Es un error común trabajar en la solución dentro del compactado, lo cual provoca que los cambios no se guarden. Si usted comete este error y entrega una solución vacía, no tendrá oportunidad de reclamar.

Un **Trie** es una estructura de datos de tipo árbol que permite la recuperación de información (de ahí su nombre, del inglés *re**trie**val*). Se utiliza para almacenar palabras como cadenas de caracteres y realizar consultas sobre este conjunto de forma eficiente. Por ejemplo: a partir de una cadena, obtener todas las palabras que tengan a dicha cadena como prefijo**.** Debido a este uso tan común del **Trie**, es conocido también como "árbol de prefijos".

> ***Un prefijo es cualquier porción del inicio de una palabra.*** *(Note que toda palabra tiene a la cadena vacía* `""` *como prefijo y una palabra en sí es prefijo de ella misma).*

En un **Trie** cada nodo tiene como valor un caracter (en este caso una letra). En la Figura 1 los nodos coloreados en naranja indican que en ese nodo hay un fin de palabra. Note, en el mismo ejemplo de la Figura 1, que no necesariamente los nodos fin de palabra son hojas porque a su vez pueden estar formando parte de otra palabra. Por ejemplo, en la figura un nodo `'L'` es naranja porque indica terminación de la palabra `"AZUL"`, pero no es hoja porque a su vez es parte de la palabra `"AZULEJO"`.

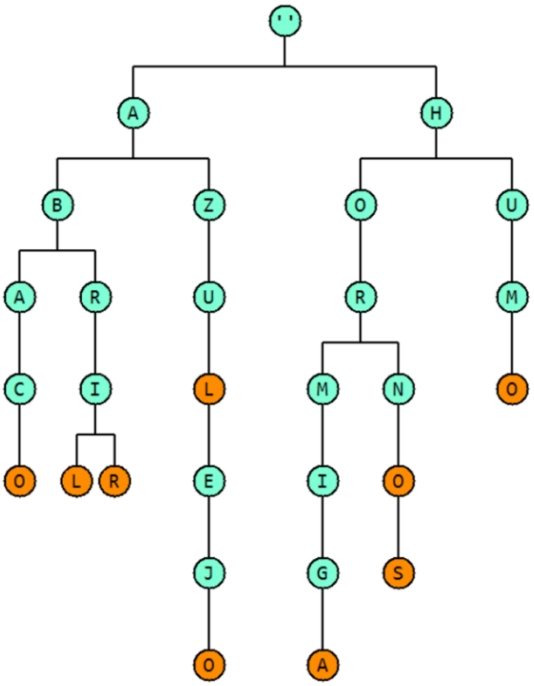


*Figura 1: Ejemplo de **Trie***

El **Trie** de la Figura 1 es el resultado de insertar las palabras: `"ABACO"`, `"ABRIL"`, `"ABRIR"`, `"AZUL"`, `"AZULEJO"`, `"HORMIGA"`, `"HORNO"`, `"HORNOS"` y `"HUMO"` en ese orden. Las palabras que se forman a partir de un mismo nodo tienen un prefijo común que es la secuencia que llega de la raíz a ese nodo. Por ejemplo, las palabras que se derivan del nodo `'B'` en la figura, es decir `"ABACO"`, `"ABRIL"` y `"ABRIR"`, tienen `"AB"` como prefijo común (nodos `'A'` y `'B'` a partir de la raíz).

El problema consiste en hacer una implementación de la *interface* `ITrie` para representar esta estructura de datos. Esta *interface* hace uso de la clase `NodoTrie` que a su vez es la que representa cada nodo de la estructura. Note que si la propiedad `FinDePalabra` devuelve `true` equivale a lo que se ha ilustrado como un nodo coloreado en naranja en la figura del ejemplo.

```
public class NodoTrie
{
    //Caracter que representa el nodo
    public char Valor { get; private set; }

    //Representa si una palabra termina en este nodo
    public bool FinDePalabra { get; set; }

    //Nodos Hijos
    public List<NodoTrie> Hijos { get; private set; }

    //Indexer para acceder al nodo hijo que corresponda a ese caracter.
    //De no existir devuelve null
    public NodoTrie this[char valor]
    {
        get {...}
    }

    //Constructor
    public NodoTrie(char valor, bool finDePalabra = false, params NodoTrie[] hijos)
    {
        ...
    }
}


public interface ITrie
{
    //Raíz del Trie. Representa el caracter vacío
    NodoTrie Raiz { get; }

    //Añade una nueva palabra al Trie
    void AgregarPalabra(string palabra);

    //Prefijo común de mayor cantidad de caracteres entre
    //todas las palabras del Trie
    string MayorPrefijoComun();

    //Palabras en el Trie que tienen el prefijo especificado
    IEnumerable<string> PalabrasConPrefijo(string prefijo);

    //Verifica si la palabra está o no en el Trie
    bool Contiene(string palabra);

    //Vacía el Trie
    void Vaciar();

    //Cantidad de palabras almacenadas en el Trie
    int CantidadDePalabras { get; }

    //Indexer para acceder al nodo hijo de la Raíz que corresponda a ese caracter.
    //De no existir devuelve null
    NodoTrie this[char valor] { get; }
}
```

La propiedad `Raiz` hace referencia al nodo raíz del **Trie**, identificado por el caracter `'\0'` (caracter "vacío"). Las inserciones de palabras se harán a partir de este nodo.

El método `AgregarPalabra` añade una nueva palabra al **Trie**. Se garantiza que para evaluar su solución no se insertarán palabras repetidas y que (para simplificar) siempre estarán conformadas con letras mayúsculas del alfabeto latino (pero sin tildes, diéresis, etc). Se debe insertar cada nuevo nodo en las listas de nodos `Hijos` de un nodo en el mismo orden en que se van insertando las palabras, es decir como el hijo "más a la derecha".

Si el nodo tiene la propiedad `FinDePalabra` en `true`, significa que en él finaliza una palabra, es parte de su implementación garantizar que esta propiedad devuelva el valor adecuado. En la Figura 2 se muestran los estados en los que queda un **Trie** luego de las inserciones consecutivas de las palabras `"SOLFEO"`, `"SOL"`, `"SOLEDAD"` y `"ZORRO"`.

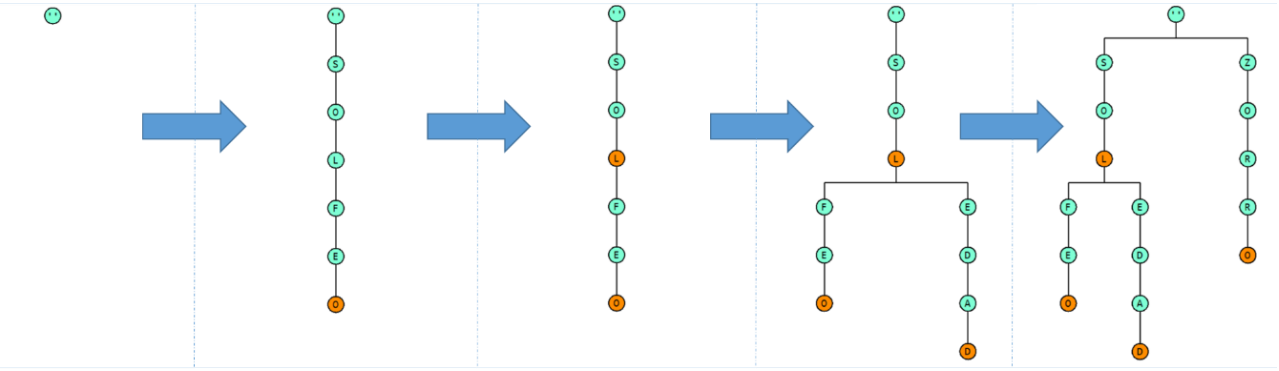


*Figura 2: Secuencia de inserciones en un **Trie**. Se insertan, en orden, las palabras **"SOLFEO"**, **"SOL"**, **"SOLEDAD"** y **"ZORRO"**.*

El método `MayorPrefijoComun` devuelve el prefijo con mayor cantidad de caracteres que es común a todas las palabras en el **Trie**. Por ejemplo si en un **Trie** las palabras insertadas son: `"SOLFEO"`, `"SOL"` y `"SOLEDAD"` el mayor prefijo común es `"SOL"` pero si a este conjunto se le añade la palabra `"ZORRO"` el mayor prefijo común es `""` (cadena vacía).

El método `PalabrasConPrefijo` devuelve un `IEnumerable<string>` con las palabras, de las insertadas en el **Trie**, que tienen como `prefijo` el especificado como argumento. Las palabras deben ser devueltas en un orden tal que **corresponda** al recorrido en **preorden** de los nodos del **Trie**. Por ejemplo, para el **Trie** de la Figura 2, si se invoca el método con el prefijo `"SO"` el `IEnumerable` debe iterar por las palabras en el orden `{"SOL", "SOLFEO","SOLEDAD"}` (ver Figura 3). Note que este no es necesariamente el orden en que fueron insertadas `{"SOLFEO", "SOL","SOLEDAD"}` ni tampoco el orden alfabético `{"SOL", "SOLEDAD","SOLFEO"}`.

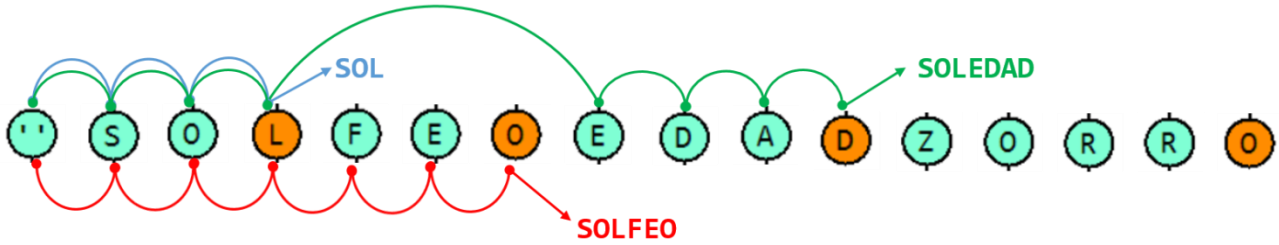


*Figura 3: Recorrido en preorden de los nodos del **Trie** de la Figura 2. Se muestra como se obtienen las palabras `"SOL"`, `"SOLFEO"` y `"SOLEDAD"` a partir de este recorrido para devolver el `IEnumerable` del llamado a `PalabrasConPrefijo("SOL")` en el orden correcto.*

El método `Contiene` verifica que la `palabra` que se pasa como parámetro se encuentra en el **Trie**. De encontrarse devuelve `true` sino devuelve `false`. La `palabra` especificada cumplirá siempre con el formato descrito anteriormente.

El método `Vaciar` elimina todas las palabras insertadas en el **Trie**.

La propiedad `CantidadDePalabras` devuelve la cantidad de palabras insertadas en el **Trie** en el momento del llamado.

El *indexer* recibe un `char` y devuelve el nodo hijo del la Raíz que tiene dicho caracter como valor, de no existir devuelve `null`.

Importante: Su implementación debe mantener el estado de los nodos del **Trie** de la forma que se ha descrito y de esta forma será evaluado en todo momento. Note que pueden existir varios **Trie** para un mismo conjunto de palabras, pero el **Trie** es único si se tiene en cuenta el orden en que fueron insertadas.

Usted debe haber recibido junto a este documento una solución de *Visual Studio* con dos proyectos: una biblioteca de clases (*Class Library*) y una aplicación de consola (*Console Application*). Usted debe completar la implementación de la clase **`Trie`** en el namespace *Weboo.Examen* que ya implementa la *interface* `ITrie`. Para esto debe seguir las especificaciones descritas en este documento.

```
public class Trie : ITrie
{
    public Trie()
    {
        Raiz = new NodoTrie('\0');
    }

    public NodoTrie Raiz { get; private set; }

    public void AgregarPalabra(string palabra)
    {
        throw new NotImplementedException();
    }

    public string MayorPrefijoComun()
    {
        throw new NotImplementedException();
    }
```

```
    public IEnumerable<string> PalabrasConPrefijo(string prefijo)
    {
        throw new NotImplementedException();
    }

    public bool Contiene(string palabra)
    {
        throw new NotImplementedException();
    }

    public void Vaciar()
    {
        throw new NotImplementedException();
    }

    public int CantidadDePalabras { get; private set; }

    public NodoTrie this[char valor]
    {
        get { throw new NotImplementedException(); }
    }
}
```

NOTA: Todo el código de la solución debe estar en este proyecto (biblioteca de clases), pues es el único código que será evaluado. Usted puede adicionar todo el código que considere necesario, pero no puede cambiar los nombres del namespace, clase o método mostrados. De lo contrario, el probador automático fallará. En particular, es imprescindible que usted no cambie el constructor de la clase `Trie`. Por supuesto, usted puede (y debe) adicionar todo el código que necesite al cuerpo del constructor para inicializar sus estructuras de datos.

Graficador de Trie

Junto al proyecto, debe haber recibido una aplicación llamada "Graficador de Trie.exe". Este sencillo programa le permitirá visualizar el estado de una instancia de su implementación de **Trie** (ver Figura 4). Use esta aplicación para probar de manera visual su implementación de los métodos `AgregarPalabra` y `Vaciar`.

Para que la aplicación funcione correctamente, el ensamblado con su implementación (archivo "Weboo.Examen.dll") y el graficador con sus dependencias deben estar en el mismo directorio. Cualquier error que ocurra, la aplicación le informará de la excepción lanzada y se cerrará.

Importante: Note que esta aplicación es una ayuda y no una garantía de que su implementación esté correcta. Su implementación se probará con una cantidad de cadenas que en la práctica es imposible de verificar visualmente. Con esta aplicación no podrá probar todas las funcionalidades que se le pide. Debe hacer todas las pruebas que necesite, para esto puede (y debe) utilizar también el *Console Application* en la solución de *Visual Studio* que se le entregó. No se podrá responsabilizar un posible malfuncionamiento de esta aplicación a cualquier resultado desfavorable que usted pueda tener en el examen.

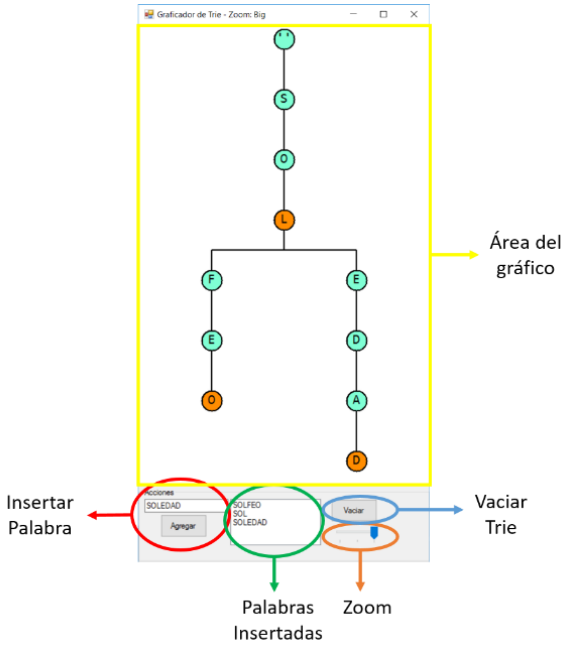
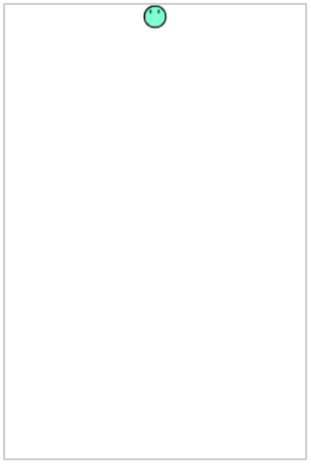


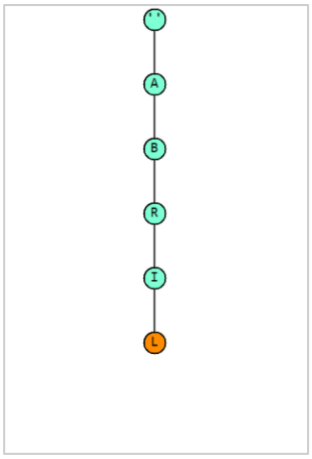
Figura 4: Funcionalidades del Graficador.

Ejemplos

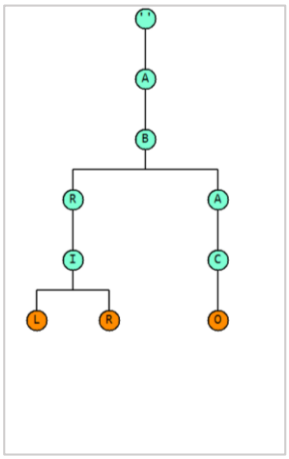
```
Trie t = new Trie(); //Se crea un nuevo trie
int total = t.CantidadDePalabras; // total = 0

string mayorPrefijo1 = t.MayorPrefijoComun(); //El mayor prefijo hasta el momento es ""
```



```
t.AgregarPalabra("ABRIL");
string mayorPrefijo2 = t.MayorPrefijoComun(); //mayorPrefijo2 = "ABRIL"
```

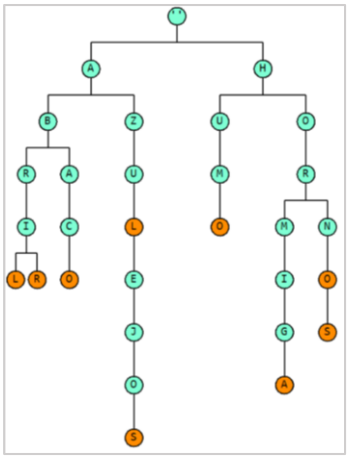



```
t.AgregarPalabra("ABRIR");
string mayorPrefijo3 = t.MayorPrefijoComun(); //mayorPrefijo3 = "ABRI"

t.AgregarPalabra("ABACO");
string mayorPrefijo4 = t.MayorPrefijoComun(); //mayorPrefijo4 = "AB"


NodoTrie nodoA = t['A']; //Único hijo de la raíz hasta el momento.
                         //Valor: 'A', fin de palabra: false, Hijos: 1

NodoTrie nodoH = t['H']; //null, ninguna palabra empieza con 'H'

NodoTrie nodoAB = nodoA['B']; //Nodo hijo de la letra 'A'. Parte del prefijo "AB"

bool iguales = nodoAB == nodoA.Hijos[0]; //iguales = true.
                                         //El nodo 'B' es el primer hijo del nodo 'A'.
```




```
total = t.CantidadDePalabras; // total = 3

t.AgregarPalabra("HUMO");

string mayorPrefijo5 = t.MayorPrefijoComun(); //mayorPrefijo5 = ""
```

```
t.AgregarPalabra("HORMIGA");
t.AgregarPalabra("AZULEJO");
t.AgregarPalabra("HORNOS");
t.AgregarPalabra("AZUL");
t.AgregarPalabra("HORNO");
```



```
nodoH = t['H']; //Segundo hijo de la raíz hasta el momento.
                //Valor 'H', fin de palabra: false, Hijos: 2

NodoTrie nodoHUMO = t['H']['U']['M']['O']; //Nodo con la letra 'O' de 'HUMO'.
                                           //Valor: 'O', fin de palabra: true, Hijos: 0


NodoTrie nodoABACO = t['A']['B']['A']['C']['O']; //Nodo con la letra 'O' de 'ABACO'.
                                                 //Valor: 'O', fin de palabra: true, Hijos: 0


bool iguales2 = nodoHUMO == nodoABACO; //iguales2 = false.
                                       //Mismo valor pero no la misma referencia.


var conPrefijoA = t.PalabrasConPrefijo("A");
//conPrefijoA = {ABRIL, ABRIR, ABACO, AZUL, AZULEJO}

var conPrefijoH = t.PalabrasConPrefijo("H");
//conPrefijoH = { HUMO, HORMIGA, HORNO, HORNOS}

var conPrefijoAZUL = t.PalabrasConPrefijo("AZUL");
//conPrefijoAZUL = { AZUL, AZULEJO}

var conPrefijoAB = t.PalabrasConPrefijo("AB");
//conPrefijoAB = { ABRIL, ABRIR, ABACO}

var conPrefijoHORN = t.PalabrasConPrefijo("HORN");
//conPrefijoHORN = { HORNO, HORNOS}

var conPrefijoX = t.PalabrasConPrefijo("X");
//conPrefijoX = { } (Iterador vacío)

var conPrefijoVacio = t.PalabrasConPrefijo("");
//conPrefijoVacio = { ABRIL, ABRIR, ABACO, AZUL, AZULEJO, HUMO, HORMIGA, HORNO, HORNOS}
//Todas las palabras en el trie
```

```
bool estaAZUL = t.Contiene("AZUL"); //estaAZUL = true
bool estaAZULES = t.Contiene("AZULES"); //estaAZULES = false (No existe secuencia de nodos)
bool estaAZULE = t.Contiene("AZULE"); //estaAZULE = false (La 'E' no es un fin de palabra)

total = t.CantidadDePalabras; // total = 9
t.Vaciar(); //El trie ha quedado vacío
nodoA = t['A']; //null, ninguna palabra empieza con 'A'
total = t.CantidadDePalabras; // total = 0
```

NOTA: Los casos de prueba que aparecen en este proyecto son solamente de ejemplo. Que usted obtenga resultados correctos con estos casos no es garantía de que su solución sea correcta y de buenos resultados con otros ejemplos. De modo que usted debe probar con todos los casos que considere convenientes para comprobar la validez de su implementación.