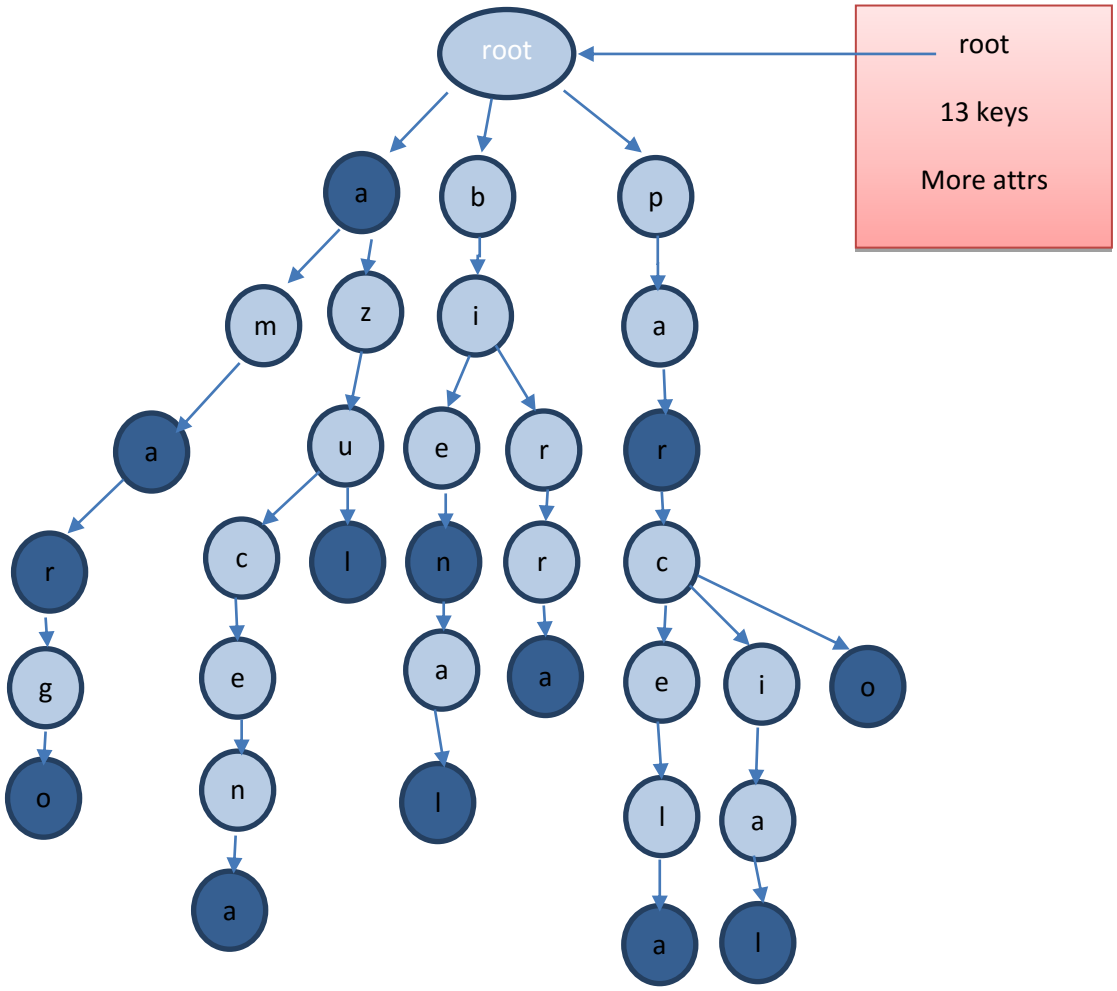
ICOM2015 – 2do Parcial

26 de noviembre de 2015

Notas:

1. Al finalizar, enviar por e-mail los archivos fuente de cada ejercicio con nombre APELLIDO_NOMBRE_Ejer_N.c a icom@ib.cnea.gov.ar
2. Uso de prácticos: **se pueden utilizar los trabajos prácticos propios realizados**.
3. Uso de Internet**: solo para la consulta de referencias de funciones de C**.

PROBLEMA 1 - Trie: Un **`Trie`** es una estructura de datos eficiente para la recuperación de información (information re**`trie`**val). Si almacenáramos claves (por ejemplo palabras) en un árbol de búsqueda binario (diccionario binario), bien balanceado, necesitaríamos un tiempo proporcional a M*log N, donde M es la longitud máxima de las claves y N el número de claves en el árbol. Utilizando un **`trie`**, se puede buscar la clave en un tiempo proporcional a M (Orden M). Cada nodo de un **`trie`** puede contener múltiples ramificaciones. Cada ramificación representa una nueva parte de la clave (un nuevo carácter en el caso de que las claves sean palabras) y en cada nodo se indica a través de un atributo si el nodo termina de definir una clave o es solo un prefix para claves que terminan en ramas inferiores (un nodo que termina de definir una clave puede también ser prefix para otras).



En la figura se notan en color oscuro los nodos que terminan de definir una clave.

Si se definen las siguientes estructuras de datos para manejar un **`Trie`** de strings.

```
typedef enum { FALSE, TRUE } BOOL;

#define ALPHABET_SIZE 26

typedef struct trie_node {
    BOOL defineKey;           // true/false si define o no una clave
    struct trie_node *children[ALPHABET_SIZE];
} Trie_node_t;

typedef struct {
    Trie_node_t *root;        // contenedor real de datos
    unsigned int numKeys;     // número de claves en el trie
} Trie_t;
```

Notar que con esta representación no se guardan los caracteres (que en la figura anterior parecerían estar incluidos), estos están implícitos en la topología de la estructura: un nodo dado define el prefix de todos sus hijos, si **`children[i]`** es no vacío, **`children[i]`** representa al carácter **`i`** siguiendo al prefix.

Se solicita implementar las siguientes funciones:

```
// crea un trie vacio y lo retorna
Trie_t *trieCreate();

// inserta una nueva clave si no está presente o
// lo marca con defineKey si ya existía como prefix de otras claves
void trieInsertKey(Trie_t *pTrie, const char *key);


// returna verdadero/falso indicando si la clave existe.
BOOL trieExistKey(Trie_t *pTrie, const char *key);

// Chequea la consistencia entre la cantidad de nodos que dice el
// trie y la cantidad de nodos que definen key.
// Retorna TRUE/FALSE indicando si hay o nó consistencia.
BOOL trieCheck(Trie_t *pTrie);

// Retorna la cantidad de claves que tienen un prefix determinado.
int trieNumWordsWithPrefix(Trie_t *pTrie, const char *prefix);

// destruye un trie liberando todos los recursos alocados
void trieDestroy(Trie_t *pTrie);
```

PROBLEMA 2 – Movimientos de alfiles en un tablero de ajedrez

Los alfiles en el ajedrez se mueven a lo largo de las diagonales y amenazan a aquellas piezas que se encuentran en su misma diagonal.

a) Escriba un programa que encuentre el número máximo de alfiles que se pueden colocar en las casillas negras del tablero de ajedrez sin que ninguno de los alfiles colocados se amenacen entre sí.

b) ¿Cuántas soluciones diferentes pueden encontrarse para el número máximo de alfiles colocados en las casillas negras del tablero?

c) Imprima en pantalla las soluciones encontradas.

PROBLEMA 3 – Page Rank

En 1999 se presentó un algoritmo, actualmente utilizado por Google, para pesar la relevancia de una página web. Si llamamos PR(A) a la relevancia de la página *A*, su valor se calcula con la siguiente expresión (1):

$$\mathrm{PR}(A) = (1-d) + d\sum_{i=1}^{n} \frac{\mathrm{PR}(i)}{C(i)}$$

Donde *i* son las páginas que apuntan por medio de hiperlinks a la página *A*, *C(i)* el número de hiperlinks de la página *i* y *d* es una constante de amortiguación que suele tomar el valor 0.55.

Si inicialmente solo se conoce la información de los hiperlinks de cada página, se solicita implementar un programa que pueda calcular el PR de cada página.

Hint: Una manera de resolver este problema es en forma iterativa: proponiendo valores iniciales (cualesquiera) para *PR(J,i)* e ir iterando utilizando (1) para calcular los valores siguientes *PR(J,i+1)*, y asi hasta que los valores converjan. *PR(J,i)* es el page Rank de la página *J* en la iteración *i*.