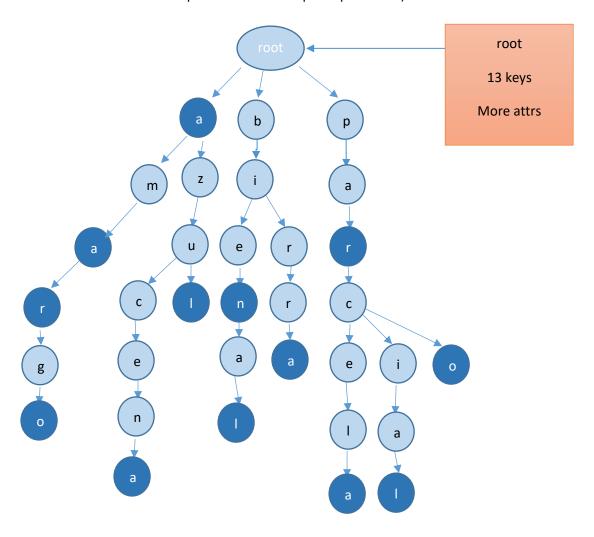
Tries

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 nodos 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.

```
#define ALPHABET SIZE 26
struct Trie node {
   BOOL defineKey;
                              // true/false si define o no una clave
    struct Trie_node *children[ALPHABET_SIZE];
};
struct Trie {
    Trie(); // TODO
    ~Trie(); // TODO
                            // contenedor real de datos
    Trie node *root;
    unsigned int numKeys; // número de claves en el trie
    // inserta una nueva clave si no está presente o
    // lo marca con defineKey si ya existía como prefix de
    // otras claves
    void insertKey(const char *key); // TODO
    // retorna verdadero/falso indicando si la clave existe.
   bool existKey(const char *key); // TODO
    // Chequea la consistencia entre la cantidad de nodos que dice el
    // trie y la cantidad de nodos que definen key.
    // Retorna un bool indicando si hay o nó consistencia.
   bool check();
    // Retorna la cantidad de claves que tienen un prefix
    // determinado.
    int numWordsWithPrefix(const char *prefix);
};
```

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.