Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

# Tarea 1

2do semestre 2018

**Entrega código:** 7 de septiembre
**Entrega informe:** 9 de septiembre

---

## Objetivos

1. Investigar, adaptar e implementar una estructura de datos desconocida para resolver un problema
2. Analizar la complejidad de la solución y compararlo con los resultados
3. Leer y procesar correctamente arreglos de caracteres en C

## Introducción

Para esta tarea, tu y tus amigos del club DCC (Dime Como Computo) quieren implementar un motor de búsqueda igual a `www.google.com`, el cual se va a llamar EDD (Este Diccionario Deslumbra). Lo primero que quieren implementar es que su motor de búsqueda va auto-completar la frase dada con la más popular. La idea es que dado un diccionario, el usuario ingrese un *string*, y que el sistema le responda con el *string* que complete la oración que sea el más frecuente. Por ejemplo, si tienen el siguiente diccionario, en donde el numero es la cantidad de veces que se ha buscado la frase que sale después de la coma:

```
10, me encanta el ramo EDD
100000, me encanta el pan con ketchup
```

y hacen la siguiente consulta:

```
me encanta
```

el sistema debería recomendar `me encanta el pan con ketchup`.

# Problema

Debe hacer un programa tal que dada una base de datos de búsquedas y un grupo de prefijos retorne las búsquedas más frecuentes dados los prefijos dados. Se espera que el tiempo en dar la respuesta a cada prefijo tome tiempo $O(k)$ donde $k$ es el largo de la respuesta. Para tu solución debes usar un Trie.

### Casos bordes

No habrá entradas duplicadas en la base de datos

En caso de empate de frecuencia de dos o más strings, se deberá entregar cualquiera.

Si el prefijo no está en la base de datos, el programa no debiese auto-completar, por lo que en este caso, deberá entregar el mismo prefijo.

Si el prefijo ya corresponde a la frase de mayor frecuencia, se debe entregar el mismo prefijo.

Si el prefijo está vacío (tiene largo 0) su programa debiera retornar la frase más frecuente de toda la base de datos

Por ejemplo, si tenemos la siguiente base de datos:

```
10 me gusta
10 me gusta EDD
10000000 amo EDD
```

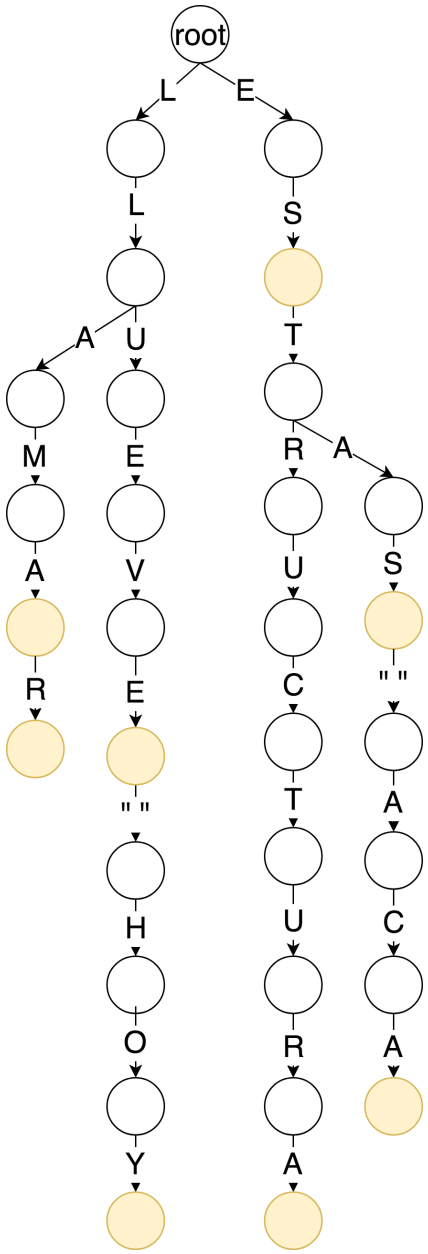
y la consulta: `me`, su programa debiera devolver: `me gusta` o `me gusta EDD`, cualquiera de las dos estaría correcta. Si tenemos la consulta `hola`, su programa debiera retornar `hola` ya que el prefijo no esta en el la base de datos. Si tenemos la consulta `me gusta EDD`, entonces el programa debiera devolver `me gusta EDD` ya que el prefijo ya corresponde a una frase completa. Finalmente, si la consulta está vacía, su programa debiera retornar `amo EDD` ya que es la frase más frecuente del diccionario.

## Trie

Para poder implementar una solución eficiente a este problema se puede usar una estructura de datos llamada Trie. Un Trie es una estructura que permite almacenar strings y hacer búsquedas eficientes sobre ella.
Para esto se almacenan los caracteres del string en las aristas que conectan los nodos. Además se marcan los nodos donde termina un string. En la figura 1 se muestra un ejemplo de esta estructura en donde podemos notar, primero, que los nodos donde terminan las palabras están pintados con amarillo y segundo, que el carácter espacio está representado por " ". Entonces, podemos darnos cuenta que el Trie de la figura 1 correspondería al siguiente conjunto de strings:

```
llama
llamar
llueve
llueve hoy
es
estructura
estas
estas aca
```

Figura 1: Visualización de un Trie



## Input

Tu programa debe ejecutarse de la siguiente forma:

```
./autocomplete [database.txt] [queries.txt] [output.txt]
```

Donde `[database.txt]` es la ruta al archivo con la base de datos de búsquedas, `[queries.txt]` es la ruta al archivo con las consultas y `[output.txt]` es la ruta al archivo con las respuestas a las consultas.

El archivo `[database.txt]` tiene el siguiente formato: La primera linea contiene un número $n$ que indica el número de strings en el diccionario. Las siguientes $n$ lineas contendrán primero un número que corresponde a la frecuencia con que se ha buscado el string, la cantidad de caracteres del string y el string propiamente tal (incluyendo el salto de linea). Puedes asumir que el string tiene entre 1 y 100 caracteres y que cada carácter puede ser cualquiera de las 26 letras del abecedario en minúscula (sin incluir la ñ) más el carácter " " (espacio). Por ejemplo:

```
8
7 5 llama
7 6 llamar
10 6 llueve
8 10 llueve hoy
15 2 es
8 10 estructura
10 5 estas
6 9 estas aca
```

Atención que es perfectamente posible que una entrada en la base de datos termine en un espacio, lo que es válido.

El archivo `[queries.txt]` tiene el siguiente formato: La primera linea contiene un número $m$ que indica el número de consultas. Las siguientes $m$ lineas contienen un entero $i \leq 100$ y un string de largo $i$ que corresponden a las consultas. Por ejemplo:

```
3
3 est
30 esto no esta en el diccionario
2 ll
```

## Output

El archivo `[output.txt]` deberá tener cada una de las respuestas en el orden en el que se dan los prefijos. Siguiendo el ejemplo de la sección **Input**, el programa debiese entregar:

```
estas
esto no esta en el diccionario
llueve
```

Noten que en el segundo caso, no se encontró el prefijo por lo que el programa imprimió el input.

## Informe y Análisis

Conteste **bevemente** las siguientes preguntas:

1. Explica cómo usaste un Trie para resolver el problema propuesto [2pts]. Se espera que sea una explicación con palabras, no código.
   OJO: Las respuestas a las siguientes preguntas están basadas en tu explicación.
2. *a*) ¿Cuál debería ser la complejidad en términos de $n$ de construir el Trie? [1pto]
   *b*) Calcule el tiempo que demora su programa en construir el Trie usando los archivos dados y construya un gráfico mostrando $n$ vs tiempo [1pto].
   *c*) Discuta brevemente sobre si se cumplió la complejidad esperada o no [1pto].
3. *a*) ¿Cuál debería ser la complejidad en términos de $m$ de contestar todas las consultas con el Trie ya construido? [1pto]
   *b*) Calcule el tiempo que demora su programa contestar las consultas con el Trie ya construido usando los archivos dados y construya un gráfico mostrando $m$ vs tiempo [1pto].
   *c*) Discuta brevemente sobre si se cumplió la complejidad esperada o no [1pto].
4. **BONUS:** Explique cómo se podría resolver este problema con otra estructura manteniendo la complejidad pero disminuyendo el uso de memoria [2pts].

## Evaluación

La nota de tu tarea se dividirá en dos partes:

- 50 % corresponde a la nota de tu código, el cual deberá pasar cada test en menos de 10 segundos.

- 50 % corresponde a la nota del informe.

Tu programa será probado con diferentes tests de dificultad creciente. Para cada uno de ellos, tu programa deberá ser capaz de entregar el output correcto dentro de un tiempo acorde a la complejidad esperada. Pasado ese tiempo el programa será terminado y se te asignará 0 puntos en ese test.

## Entrega

- Deberás entregar tu tarea en el repositorio que se te será asignado, asegúrate de seguir la estructura correcta de este.
- Se espera que tu código compile con `make` dentro de la carpeta **Programa** y genere un ejecutable de nombre `autocomplete` en esa misma carpeta.
- Se espera que dentro de la carpeta **Informe** entregues tu informe en formato PDF, con el nombre `Informe.pdf`.
- No debes subir a tu repositorio los archivos de los test ya que son muy pesados.

Estas reglas ayudan a recolectar y corregir las tareas de forma automática, por lo que su incumplimiento implicará un descuento en tu nota.
Se recogerá el estado de la rama `master` en tu repositorio 1 minuto pasadas las 23:59 horas del día de la entrega (tanto para el código como para el informe). Recuerda dejar ahí la versión final de tu tarea. No se permitirá la entrega de tareas atrasadas.

## Bonus: Manejo de memoria perfecto (+5 % a la nota del Código)

La nota de tu código aumentará en un 5 % si `valgrind` reporta 0 leaks y 0 errores de memoria en tu código. Todo esto considerando que tu programa haga lo que **tiene** que hacer. El bonus solo aplicará en caso que tu nota de código sea sobre 4.