Implementación de TinyURL usando Tabla Hash

Modelado y Programación

2 de septiembre de 2025

1. Descripción del Problema

En esta evaluación, implementarás un servicio básico similar a TinyURL que permite acortar URLs largas y convertirlas en versiones más cortas y manejables. El objetivo principal es crear una tabla hash eficiente que pueda almacenar y recuperar las URLs de manera rápida, junto con una clase TinyURL que utilice esta tabla hash para proporcionar la funcionalidad de acortamiento de URLs.

1.1. ¿Qué es TinyURL?

TinyURL es un servicio web que toma URLs largas y las convierte en versiones más cortas. Por ejemplo:

- URL original: `https://www.ejemplo.com/pagina/muy/larga//parametros?id=12345`
- URL acortada: `http://tiny.ly/abc123`

1.2. Funcionamiento del Sistema

El sistema debe funcionar de la siguiente manera:

1. **Acortar URL**: Cuando un usuario proporciona una URL larga, el sistema genera una clave única (código corto) y almacena la relación en la tabla hash.
2. **Expandir URL**: Cuando un usuario accede a la URL corta, el sistema busca la URL original en la tabla hash y redirige al usuario.
3. **Gestión de capacidad**: La tabla hash debe poder redimensionarse automáticamente cuando el factor de carga sea demasiado alto para mantener un rendimiento óptimo.

2. Requisitos de Implementación

Debes implementar dos clases que trabajen en conjunto:

2.1. Clase: HashTable

Una tabla hash genérica con los siguientes métodos:

1. `put(String key, String value)`: Almacena un valor con su clave correspondiente.
2. `get(String key)`: Recupera el valor asociado con la clave dada.
3. `getLoad()`: Calcula y retorna el factor de carga actual de la tabla hash.
4. `resize()`: Redimensiona la tabla hash cuando el factor de carga supera un umbral definido.
5. `toString()`: Retorna una representación en cadena del estado actual de la tabla hash.

2.2. Clase: TinyURL

Un servicio que utiliza la tabla hash para acortar URLs:

1. `insertUrl(String url)`: Recibe una URL larga y retorna un código corto único de 8 caracteres.
2. `redirect(String code)`: Recibe un código corto y retorna la URL original correspondiente.
3. `generateRandomCode()`: Método privado que genera códigos aleatorios de 8 caracteres.
4. `generateUniqueCode()`: Método privado que garantiza códigos únicos en el sistema.

2.3. Detalles de Implementación para TinyURL

2.3.1. Variable CHARACTERS

La clase TinyURL debe incluir una constante que defina los caracteres permitidos:

```
1 private static final String CHARACTERS =
2     "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
        ;
```

Listing 1: Definición de caracteres permitidos

Esta cadena contiene:

- 26 letras minúsculas (a-z)
- 26 letras mayúsculas (A-Z)
- 10 dígitos (0-9)
- Total: 62 caracteres diferentes

2.3.2. Implementación de generateRandomCode()

Para implementar este método:

1. Crear un `StringBuilder` para construir el código
2. Usar un bucle for de 8 iteraciones (DEFAULT_CODE_LENGTH = 8)
3. En cada iteración:
   - Generar un índice aleatorio: `random.nextInt(CHARACTERS.length())`
   - Obtener el carácter: `CHARACTERS.charAt(randomIndex)`
   - Agregarlo al StringBuilder
4. Retornar `code.toString()`

2.4. Consideraciones Técnicas

- **Función Hash**: Debes implementar una función hash que distribuya las claves de manera uniforme.
- **Manejo de Colisiones**: Utiliza encadenamiento (chaining) con listas enlazadas para manejar colisiones.
- **Factor de Carga**: Mantén un factor de carga máximo de 0.75 para garantizar un rendimiento óptimo.
- **Redimensionamiento**: Cuando el factor de carga supere 0.75, duplica el tamaño de la tabla y rehash todos los elementos.
- **Clase Entry**: Dentro de HashTable, utiliza una clase interna `Entry` para representar pares clave-valor. Esta clase debe contener los campos `key` y `value` de tipo String, junto con un constructor y el método `equals()` para comparar entradas por clave.

3. Especificaciones de Rendimiento

3.1. HashTable

- **Complejidad esperada para put()**: O(1) promedio
- **Complejidad esperada para get()**: O(1) promedio
- **Complejidad esperada para getLoad()**: O(1)
- **Complejidad para resize()**: O(n) donde n es el número de elementos

3.2. TinyURL

- **Complejidad esperada para insertUrl()**: O(1) promedio
- **Complejidad esperada para redirect()**: O(1) promedio

4. Entregables

1. Implementación completa de la clase `HashTable`
2. Implementación completa de la clase `TinyURL`
3. Documentación de código con comentarios explicativos
4. Tu implementación debe pasar todas las pruebas unitarias proporcionadas

5. Criterios de Evaluación

- **Correctitud (40 %)**: La implementación debe funcionar correctamente y pasar todas las pruebas.
- **Eficiencia (30 %)**: El código debe cumplir con las complejidades temporales esperadas.
- **Calidad del código (20 %)**: Código limpio, bien documentado y siguiendo buenas prácticas.
- **Manejo de casos edge (10 %)**: Manejo adecuado de casos especiales y errores.

6. Preguntas de Análisis

Responde las siguientes preguntas sobre el servicio TinyURL implementado:

1. **Capacidad del sistema**: Con códigos de 8 caracteres y 62 caracteres diferentes disponibles, ¿cuántas URLs únicas puede almacenar teóricamente el sistema? Justifica tu cálculo.
2. **Probabilidad de colisiones**: Si tenemos 1 millón de URLs almacenadas, ¿cuál es la probabilidad aproximada de que se genere un código duplicado al insertar una nueva URL? ¿Cómo afecta esto al rendimiento?
3. **Escalabilidad**: Si el sistema necesita manejar 100 millones de URLs, ¿qué modificaciones harías al diseño actual? Considera tanto la longitud de los códigos como la estructura de datos subyacente.

7. Instrucciones de Compilación y Ejecución

Para compilar y ejecutar las pruebas unitarias, sigue estos pasos:

7.1. Paso 1: Compilar las Clases

```
javac HashTable.java TinyURL.java
```

Listing 2: Compilación de las clases principales

7.2. Paso 2: Descargar JUnit 5

```
curl -L -o junit-platform-console-standalone-1.8.0.jar \
https://repo1.maven.org/maven2/org/junit/platform/\
junit-platform-console-standalone/1.8.0/\
junit-platform-console-standalone-1.8.0.jar
```

Listing 3: Descarga de JUnit 5 Standalone

7.3. Paso 3: Compilar las Pruebas

```
javac -cp ".:junit-platform-console-standalone-1.8.0.jar" \
TinyURLHashTableTest.java
```

Listing 4: Compilación de las pruebas unitarias

7.4. Paso 4: Ejecutar las Pruebas

```
java -cp ".:junit-platform-console-standalone-1.8.0.jar" \
org.junit.platform.console.ConsoleLauncher \
--classpath . --select-class TinyURLHashTableTest
```

Listing 5: Ejecución de las pruebas unitarias

7.5. Resultado Esperado

Al ejecutar las pruebas correctamente, deberías ver un resultado similar a:

```
Test run finished after 57 ms
[         26 tests found           ]
[          0 tests skipped         ]
[         26 tests started         ]
[          0 tests aborted         ]
[         26 tests successful      ]
[          0 tests failed          ]
```

Listing 6: Salida esperada de las pruebas

Nota: En sistemas Windows, reemplaza los dos puntos `:` en el classpath por punto y coma `;`.