Presentación Trabajo Práctico

Sistemas Operativos
DC - UBA - FCEN

Mayo de 2023

Problema a Resolver

- Se requiere manejar una estructura que, dado uno o varios textos, permita almacenar de manera **eficiente** las apariciones de cada palabra.
- Eficiente: Que permita aprovechar el uso de la concurrencia.

Estructura elegida

Se trabajará usando un **HashMapConcurrente**.

Estructura elegida

Se trabajará usando un **HashMapConcurrente**. ¿Qué es?

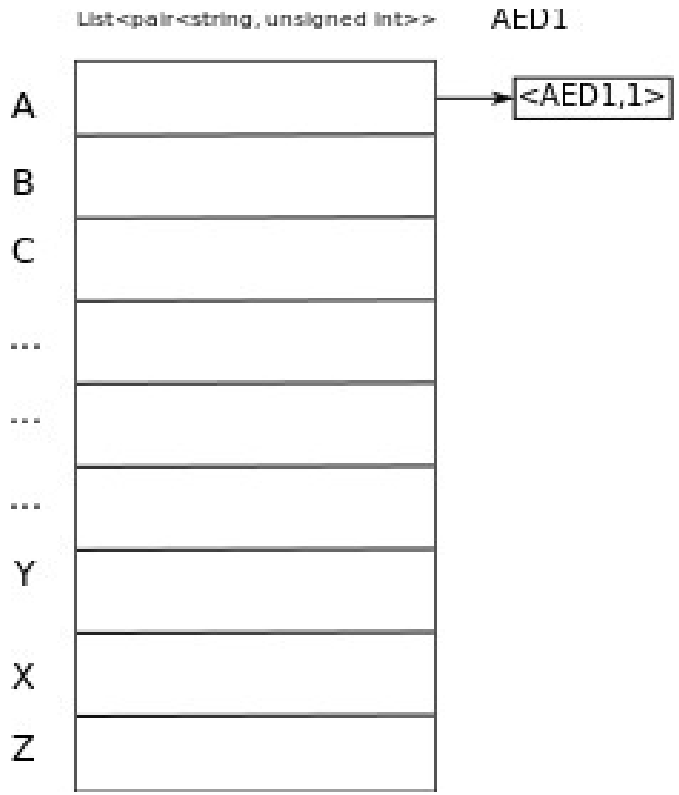
Estructura elegida

Se trabajará usando un **HashMapConcurrente**. ¿Qué es?

- Es un hashmap: Un diccionario implementado sobre una tabla de hash.
- Concurrente: Soporta accesos simultáneos.
  - No deben existir *deadlocks* ni *race conditions* ni *inanición*.
- Tabla de hash: Arreglo cuyos índices se determinan por una función hash del valor. Puede haber más de un valor en un mismo índice.
- Se requiere almacenar las palabras (claves) del diccionario en una tabla de hash usando como función hash la primera letra de cada palabra.

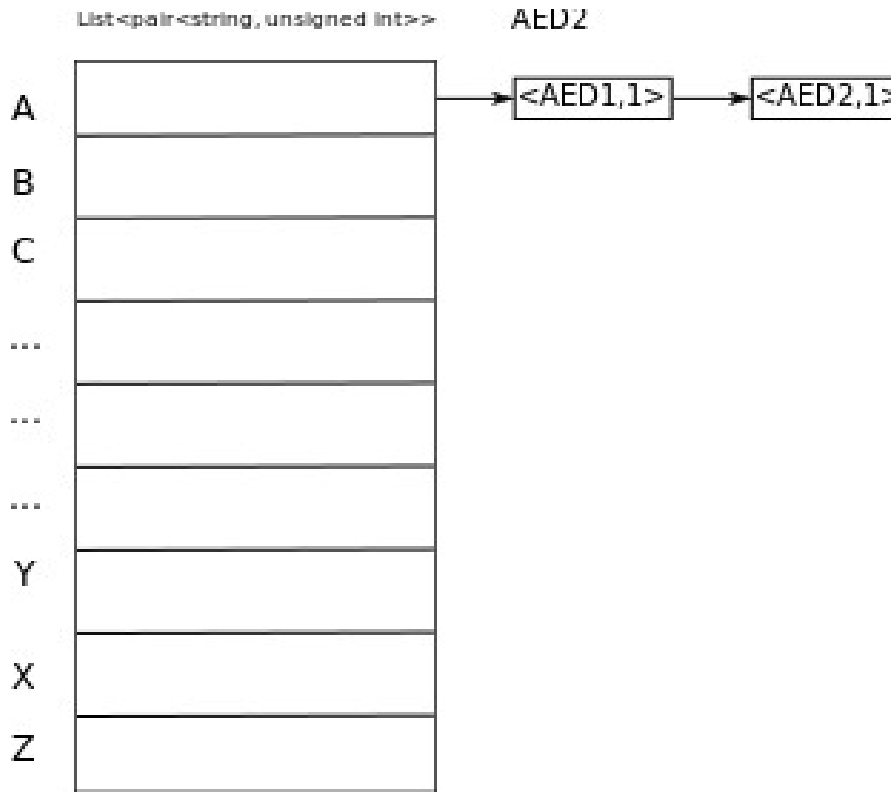
Operaciones

- **void insertar(T valor)**
  Está en el archivo ListaAtomica.hpp
- **void incrementar(string clave)**
  Este y los siguientes métodos están en HashMapConcurrente.cpp



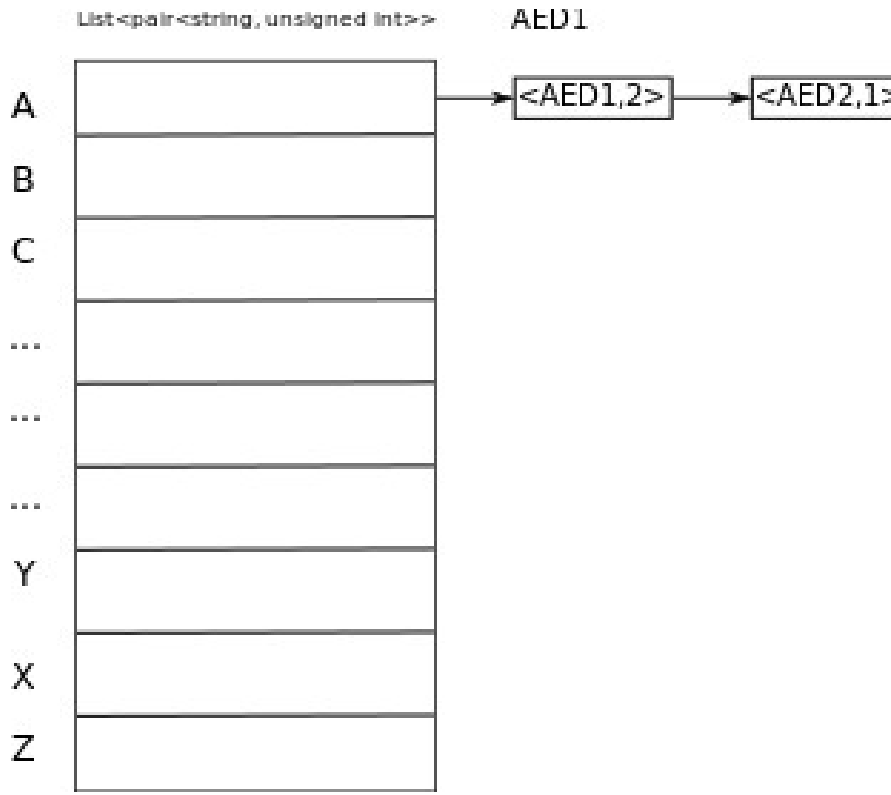
Operaciones

- **void insertar(T valor)**
  Está en el archivo ListaAtomica.hpp
- **void incrementar(string clave)**
  Este y los siguientes métodos están en HashMapConcurrente.cpp



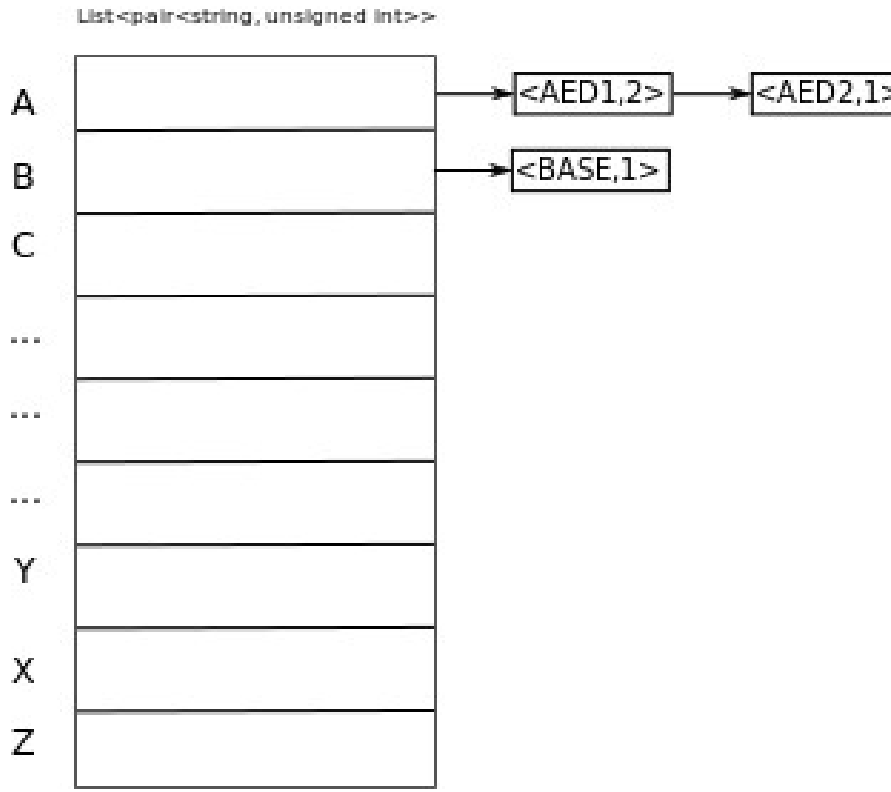
Operaciones

- **void insertar(T valor)**
  Está en el archivo ListaAtomica.hpp
- **void incrementar(string clave)**
  Este y los siguientes métodos están en HashMapConcurrente.cpp



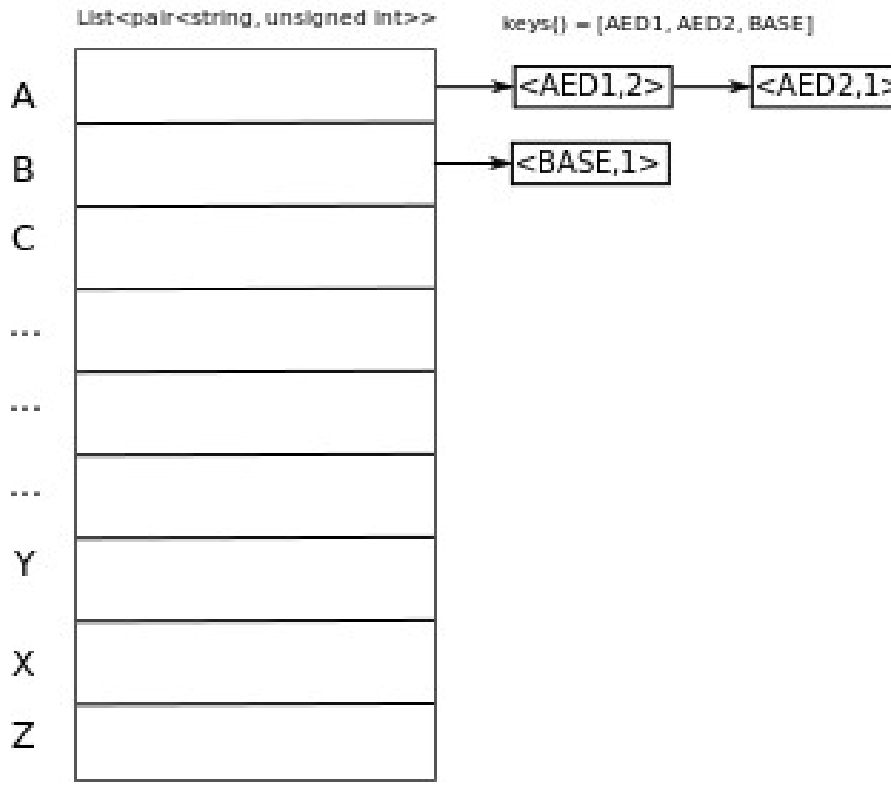
Operaciones

- **void insertar(T valor)**
  Está en el archivo ListaAtomica.hpp
- **void incrementar(string clave)**
  Este y los siguientes métodos están en HashMapConcurrente.cpp



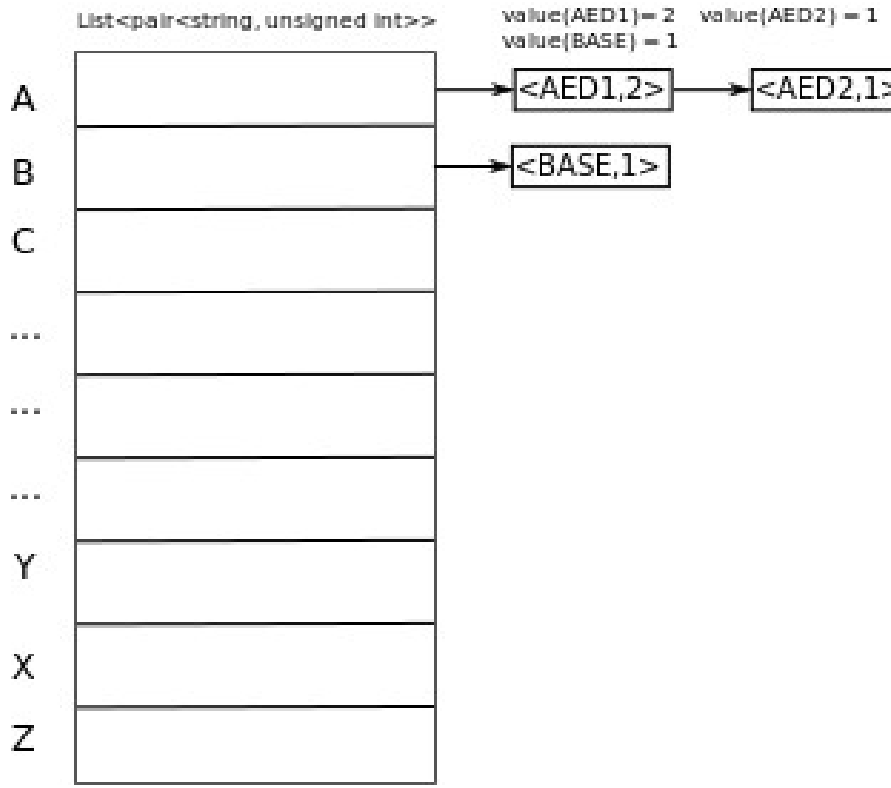
Operaciones

- **void insertar(T valor)**
  Está en el archivo ListaAtomica.hpp
- **void incrementar(string clave)**
  Este y los siguientes métodos están en HashMapConcurrente.cpp
- **vector<string>claves()**



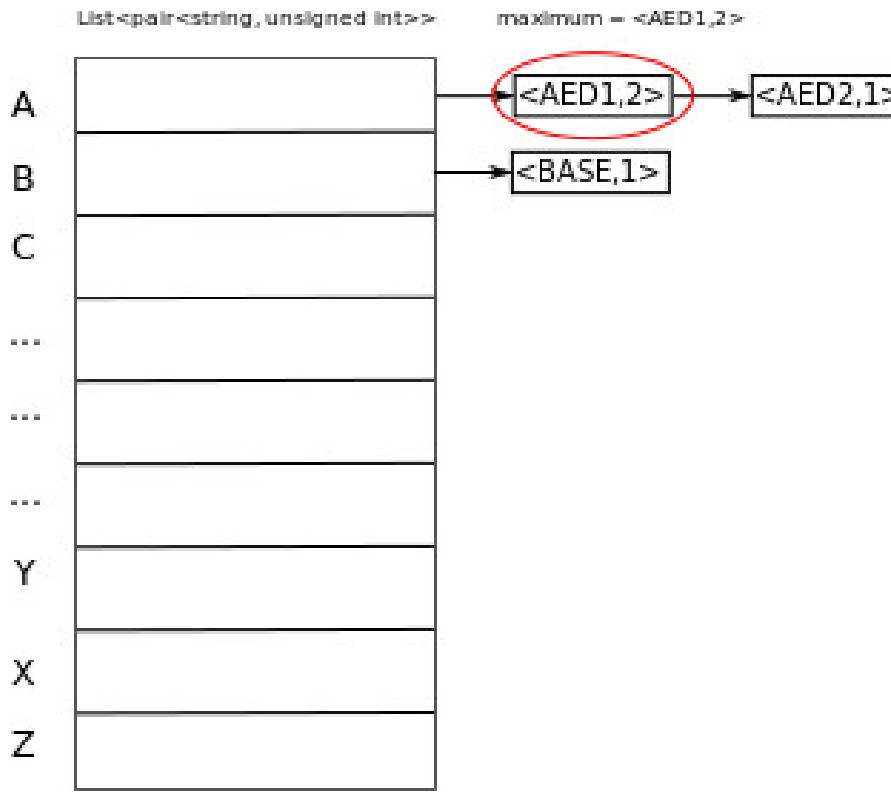
Operaciones

- **void insertar(T valor)**
  Está en el archivo ListaAtomica.hpp
- **void incrementar(string clave)**
  Este y los siguientes métodos están en HashMapConcurrente.cpp
- **vector<string>claves()**
- **unsigned int valor(string valor)**



Operaciones

- **void insertar(T valor)**
  Está en el archivo ListaAtomica.hpp
- **void incrementar(string clave)**
  Este y los siguientes métodos están en HashMapConcurrente.cpp
- **vector<string>claves()**
- **unsigned int valor(string valor)**
- **pair < string, unsigned int > maximo()**

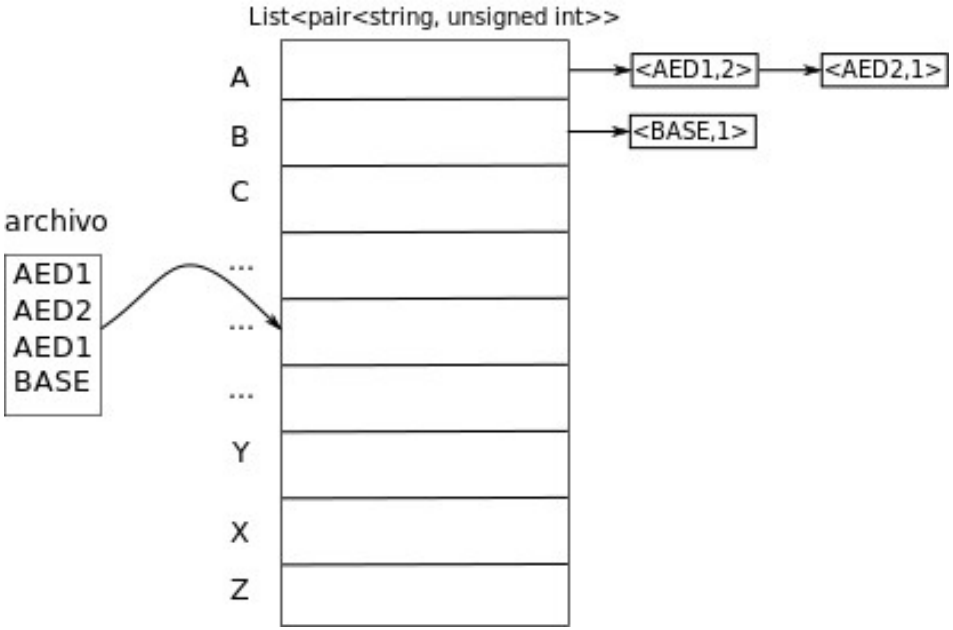


Operaciones

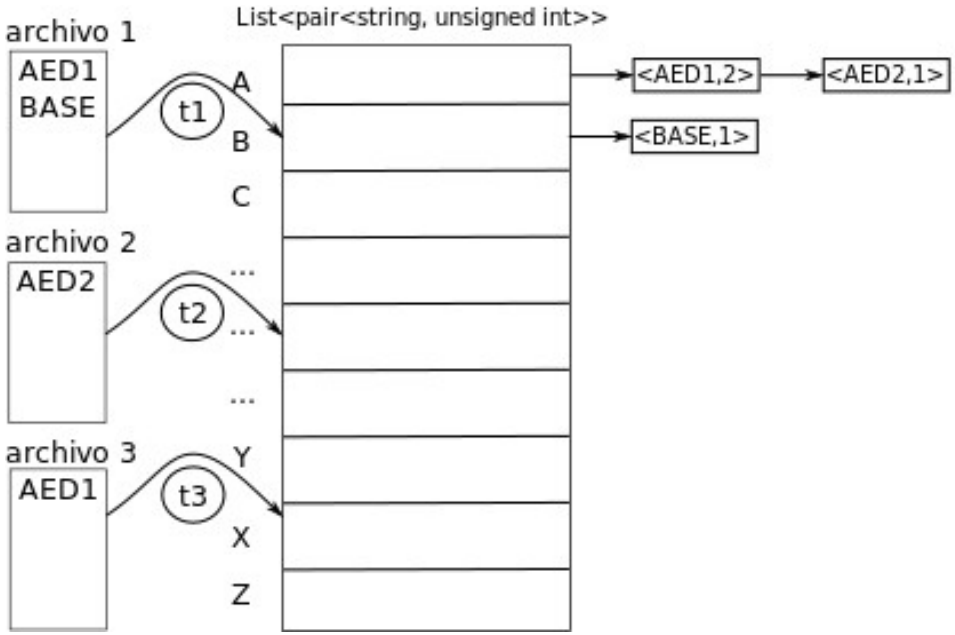
archivo

AED1
AED2
AED1
BASE

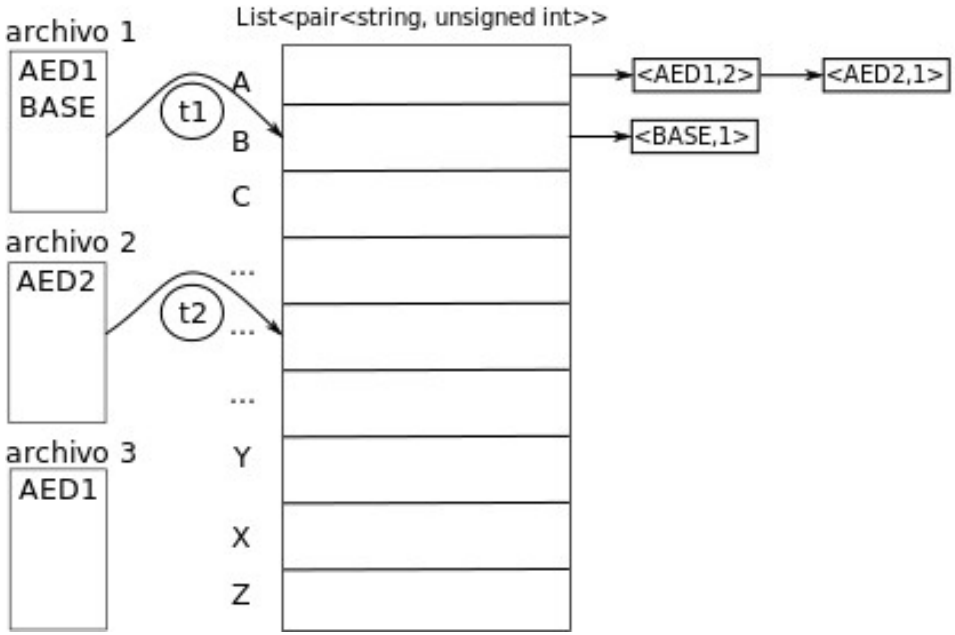
Operaciones



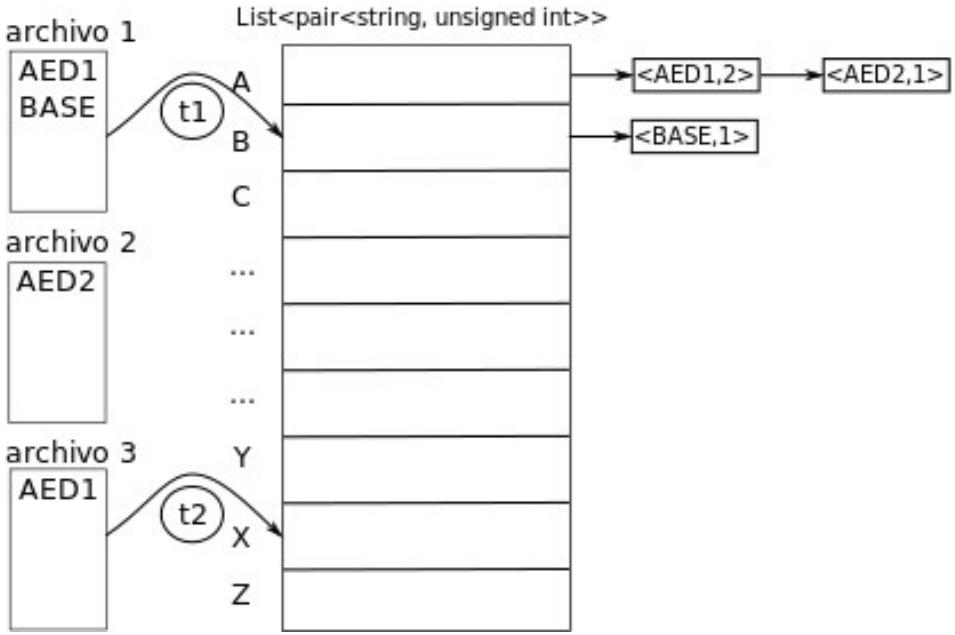
Operaciones



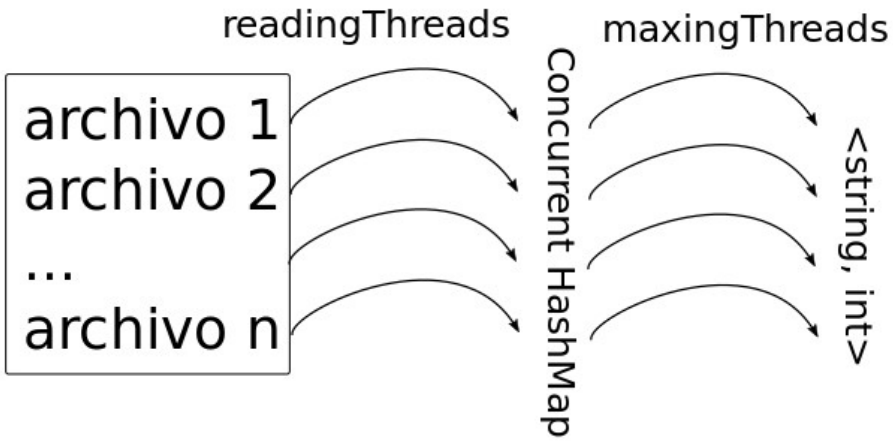
Operaciones



Operaciones



Operaciones



Ejercicio 1

- En el archivo `ListaAtomica.hpp`, completar la implementación del método:

```
void insertar(T valor)
```

Ejercicio 2

- Completar la implementación del archivo `HashMapConcurrente.cpp` con lo mencionado a continuación.

Ejercicio 2.a

- Complete la implementación de:

```
void incrementar(string clave)
```

- Si clave existe en la tabla (en la entrada correspondiente de la lista), se debe incrementar su valor en uno. Debe hacerse de manera atómica.

Ejercicio 2.b

- Complete las implementación de:

```
vector<string> claves()
```

- Devuelve todas las claves existentes en la tabla. Esta operación debe ser no bloqueante y libre de inanición.

Ejercicio 2.c

- Complete las implementación de:

```
unsigned int valor(string clave)
```

- Devuelve el valor de clave en la tabla, o 0 si la clave no existe en la tabla. Esta operación debe ser no bloqueante y libre de inanición.

Ejercicio 3

- Completar la implementación de la clase `HashMapConcurrente` con lo requerido.

Ejercicio 3.a

- La implementación provista de la siguiente función puede ejecutarse concurrentemente con `incrementar`, ¿cuáles problemas podría traer?.

```
pair<string, unsigned int>maximo()
```

- Modificar la implementación para que esto ya no pueda suceder.

Ejercicio 3.b

- Completar la implementación de:

```
pair<string, unsigned int>maximoParalelo(unsigned int cantThreads)
```

- Realiza lo mismo que máximo pero repartiendo el trabajo entre la cantidad de threads indicada.

Ejercicio 4

- Completar la implementación del archivo `CargarArchivos.cpp` con lo mencionado a continuación:

Ejercicio 4.a

- Completar la implementación de:

```
cargarArchivo(HashMapConcurrente hashMap, string filePath)
```

- La cual lee el archivo indicado mediante la ruta del archivo **filePath** y carga todas sus palabras en la **hashMap**.

Ejercicio 4.b

- Implementar la función:

```
void cargarMultiplesArchivos(HashMapConcurrente hashMap,
                             unsigned int cantThreads, vector<string>filePaths)
```

- La cual carga todos los archivos indicados por parámetro, repartiendo el trabajo entre la cantidad de threads indicada por **cantThreads**.

Ejercicio 5

- Evaluar, en términos de rendimiento, la ejecución concurrente a la hora de encontrar la palabra con mayor cantidad de apariciones en un conjunto de archivos.
- Elaboren tres o más hipótesis, basándose en los conocimientos que poseen sobre el tema.
- Diseñen y efectúen experimentos que permitan poner a prueba estas hipótesis.
- Se sugiere medir el tiempo de ejecución requerido para:
  - Cargar las palabras en la tabla de hash.
  - Computar el máximo.
- Variar la cantidad de threads utilizada en cada una de ellas.
- Se sugiere usar la función **clock gettime** (con la opción CLOCK REALTIME) de la biblioteca **time.h**.

Ejercicio 5

- Presenten sus resultados en el informe de manera clara, utilizando los recursos que les parezcan adecuados (gráficos, figuras, etc.).
- Realicen un análisis de los resultados.
- En el informe deben incluir:
  - Introducción.
  - Presentación del problema.
  - Enfoque para la presentación del problema.
  - Resultados.
  - Análisis de resultados y síntesis.
  - Conclusiones.

Consideraciones

- Implementación libre de condiciones de carrera.
- Ningún thread deberá escribir un resultado ya resuelto por otro thread.

Consideraciones

Realizar un informe justificando la implementación realizada. No **incluir** código en el informe.

Agregar al informe los resultados obtenidos.

Algunas preguntas disparadoras para las conclusiones:

- ¿Qué sentido le ve al uso de *threads* para resolver tareas de este tipo?
- ¿Cuáles son los casos que considera posibles, en este escenario, para que exista concurrencia?
- ¿Cuáles son los casos que considera posibles, en este escenario, para que surjan condiciones de carrera?
- En el enunciado se proponen más preguntas de esta índole. No se limiten a contestar solamente estas preguntas, ya que son solo disparadoras.

Por último: algunas pautas de entrega

- ★ Entrega vía campus.
- ★ Subir un archivo comprimido que deberá contener únicamente:
  1. El documento del informe (en PDF).
  2. El código fuente. **NO incluir código compilado: ejecutar "make clean" antes de enviar**.
  3. Tests mostrando la correcta implementación.
  4. Makefile para correr los test agregados (se puede modificar el que ya está).
- ★ Fecha límite: 11/06/2022 (OJO! es domingo).
- Si requieren una guía para saber cómo elaborar un informe técnico, pueden leer el siguiente enlace: `https://campus.fundec.org.ar/admin/archivos/MAETICO-TECyelinformetecnico.pdf`

¿Preguntas?