Presentación TP1

Sistemas Operativos
DC - UBA - FCEN

18 de septiembre de 2018

Problema a Resolver

- Queremos una estructura que, dado un texto (o varios), permita almacenar de manera **eficiente** las apariciones de cada palabra.
- Eficiente: En este caso queremos que ademas de mantener un desempeño razonable en términos de memoria y tiempo, permita aprovechar la ejecución en un entorno concurrente.

Estructura elegida

Realizaremos un **Concurrent HashMap**.

Estructura elegida

Realizaremos un **Concurrent HashMap**. Qué es ?

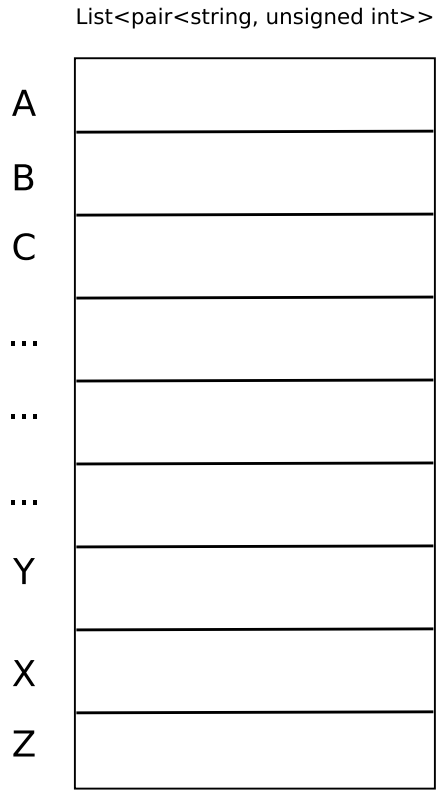
Estructura elegida

Realizaremos un **Concurrent HashMap**. Qué es ?

- Es un hashmap (a.k.a. un diccionario implementado sobre una tabla de hash)
- Concurrente: Soporta accesos simultáneos manteniendo la consistencia.
- Tabla de Hash (breve repaso de Algo 2): Arreglo cuyos valores están en un índice determinado por una función del valor (no es inyectiva, por lo que puede haber más de un valor en un índice).
- Queremos almacenar las claves del diccionario en una tabla de Hash usando como función de Hash la primera letra de la clave.

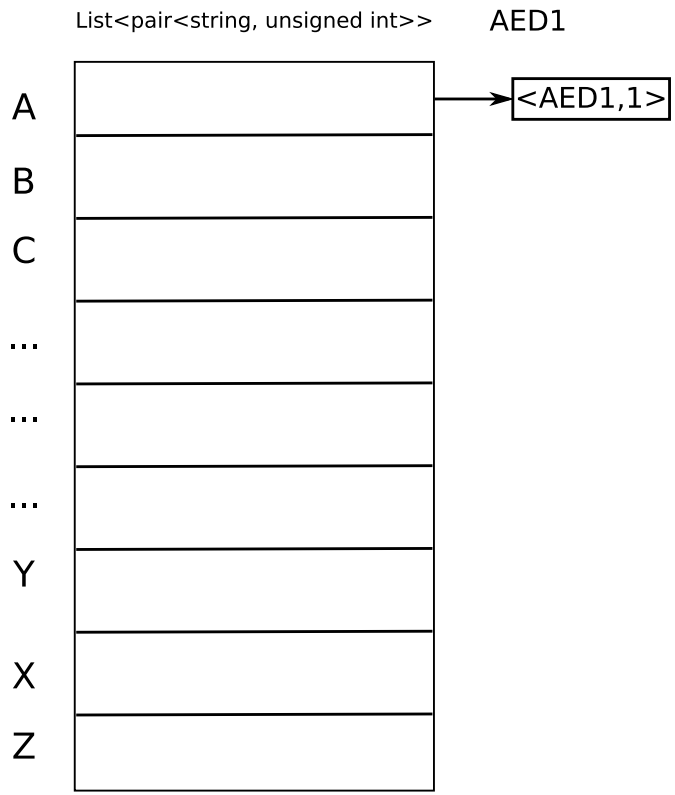
Operaciones

- `create()`



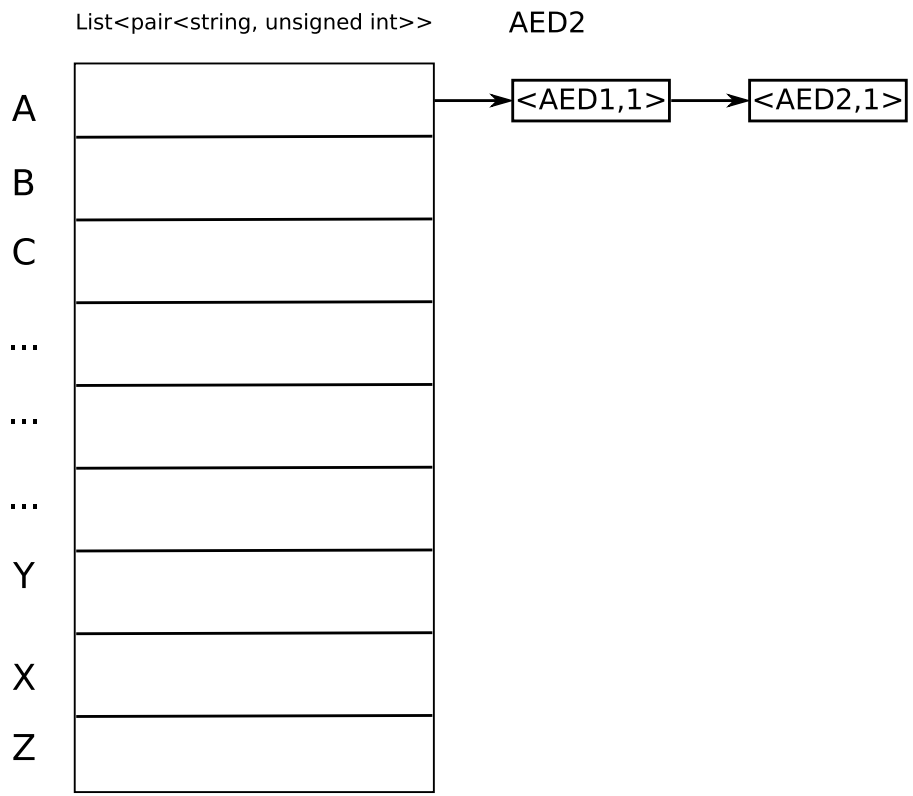
Operaciones

- `create()`
- `void addAndInc(string key)`



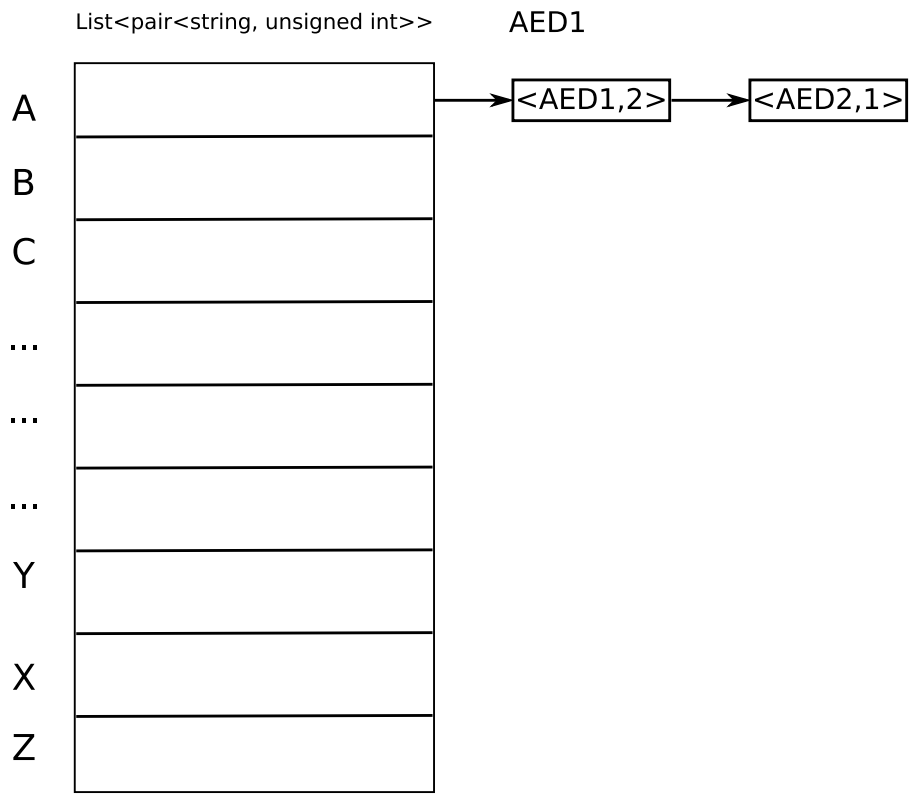
Operaciones

- `create()`
- `void addAndInc(string key)`



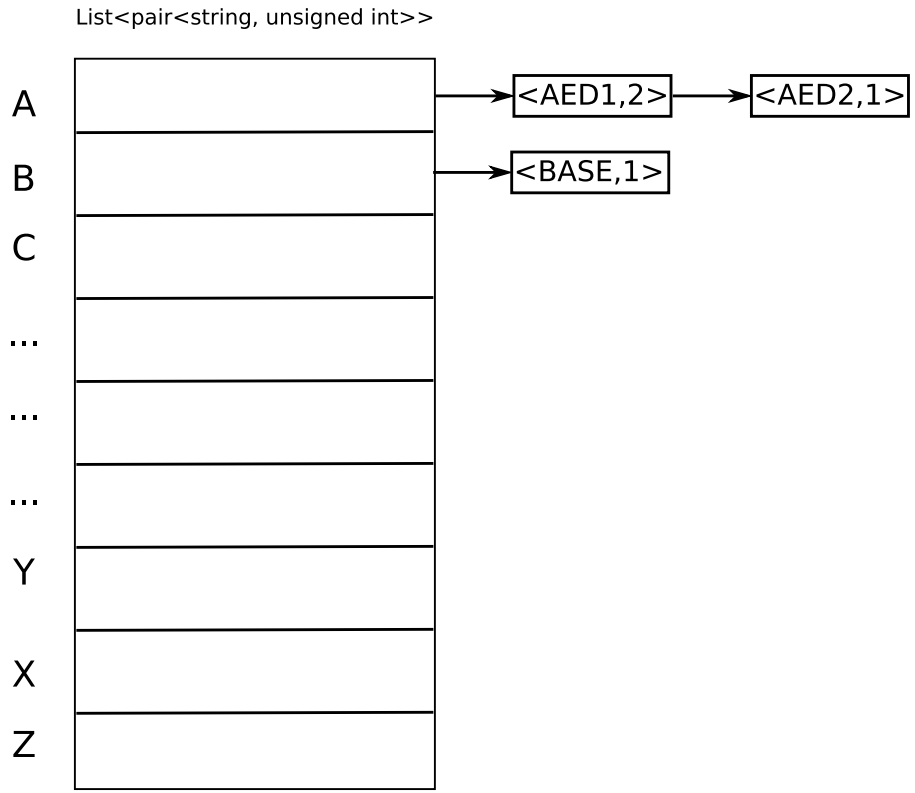
Operaciones

- `create()`
- `void addAndInc(string key)`



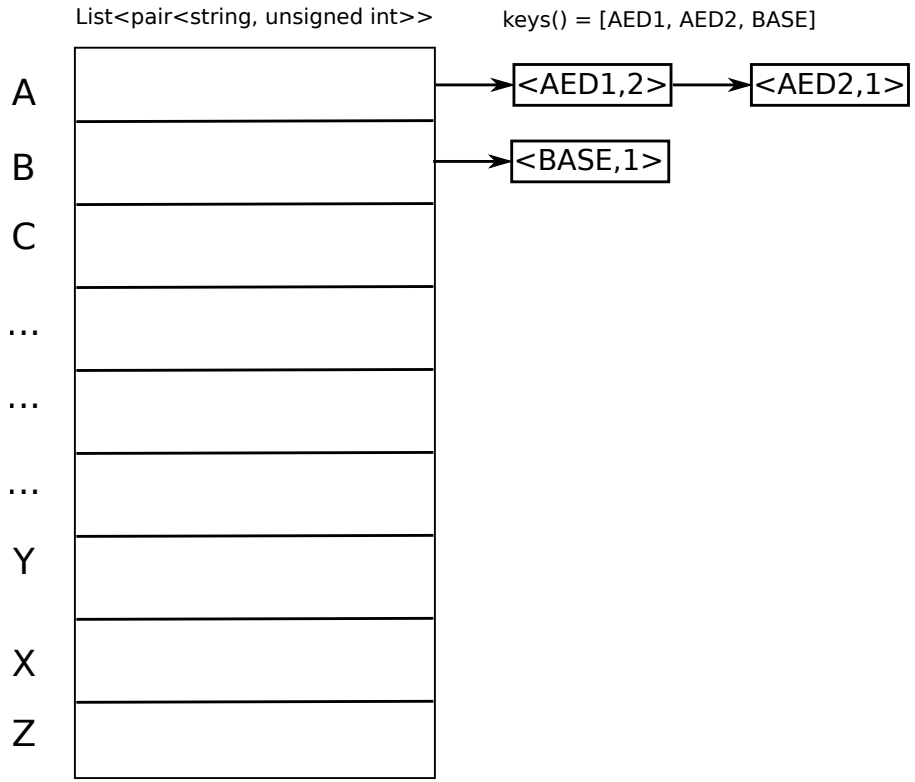
Operaciones

- `create()`
- `void addAndInc(string key)`



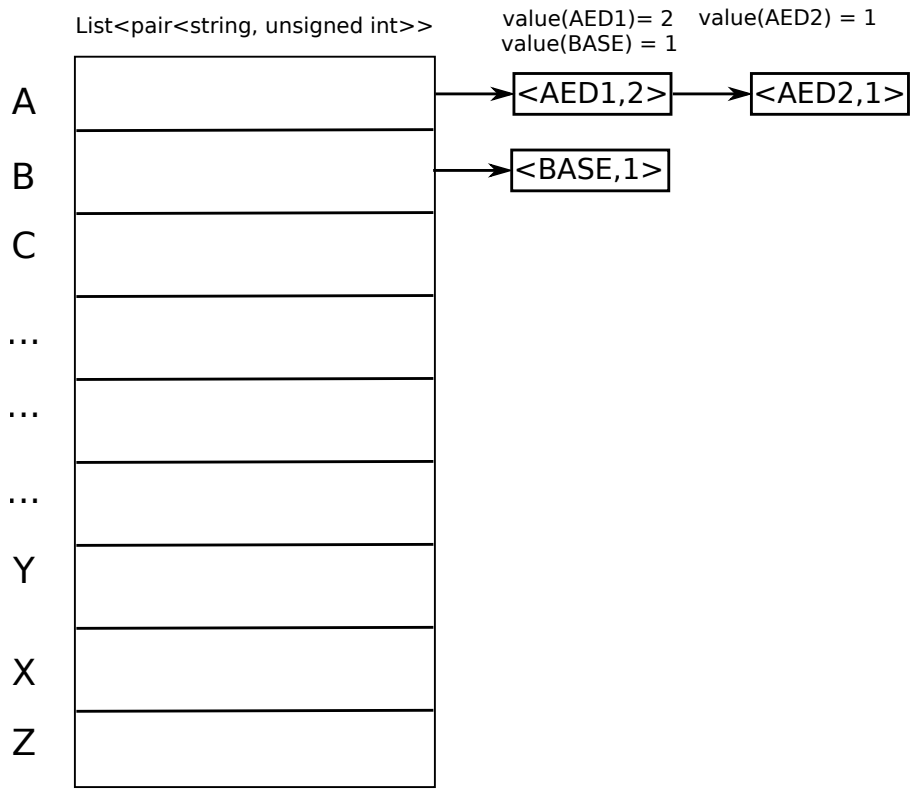
Operaciones

- `create()`
- `void addAndInc(string key)`
- `list<string> keys()`



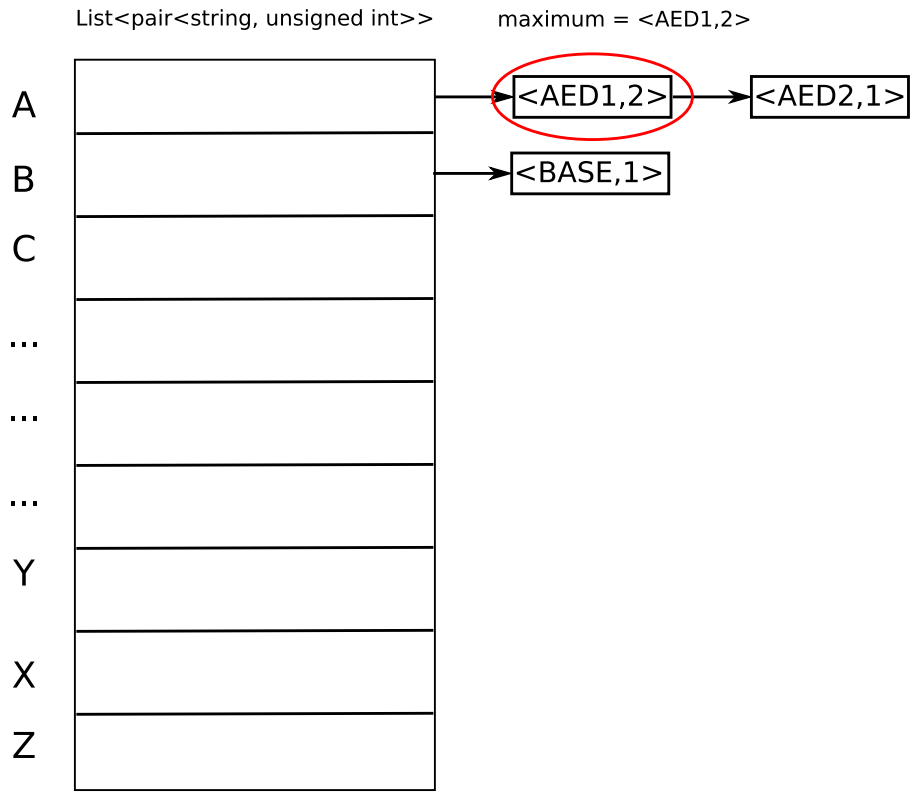
Operaciones

- `create()`
- `void addAndInc(string key)`
- `list<string> keys()`
- `unsigned int value(string key)`



Operaciones

- `create()`
- `void addAndInc(string key)`
- `list<string> keys()`
- `unsigned int value(string key)`
- `pair<string, unsigned int> maximum(unsigned int n)`



Ejercicio 1

Completar la implementación del `ConcurrentHashMap` con la interfaz pedida.

Ejercicio 2.a

```
ConcurrentHashMap countWordsInFile(string filePath)
```

Ejercicio 2.a

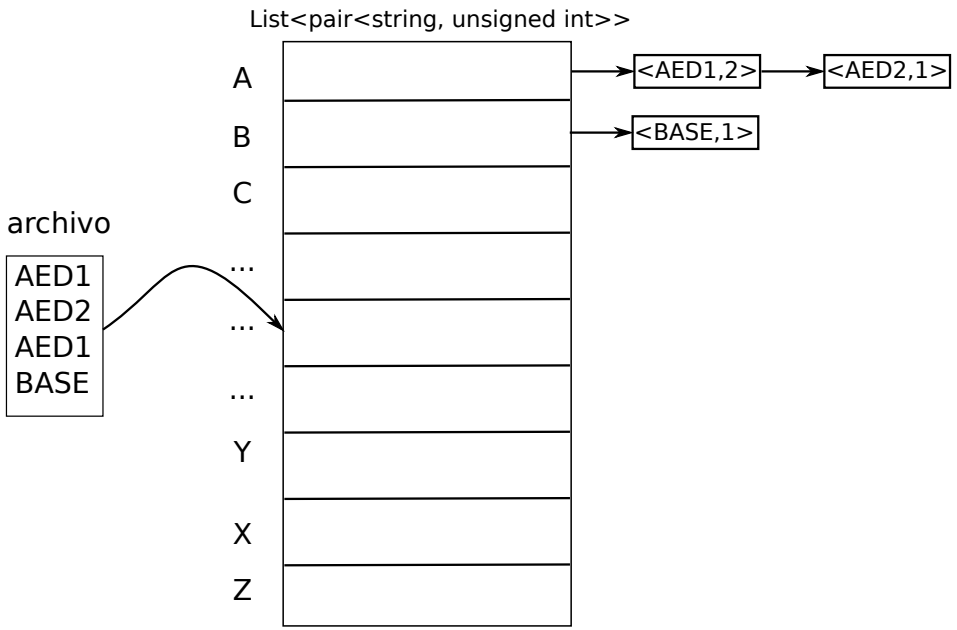
```
ConcurrentHashMap countWordsInFile(string filePath)
```

archivo

```
AED1
AED2
AED1
BASE
```

Ejercicio 2.a

```
ConcurrentHashMap countWordsInFile(string filePath)
```

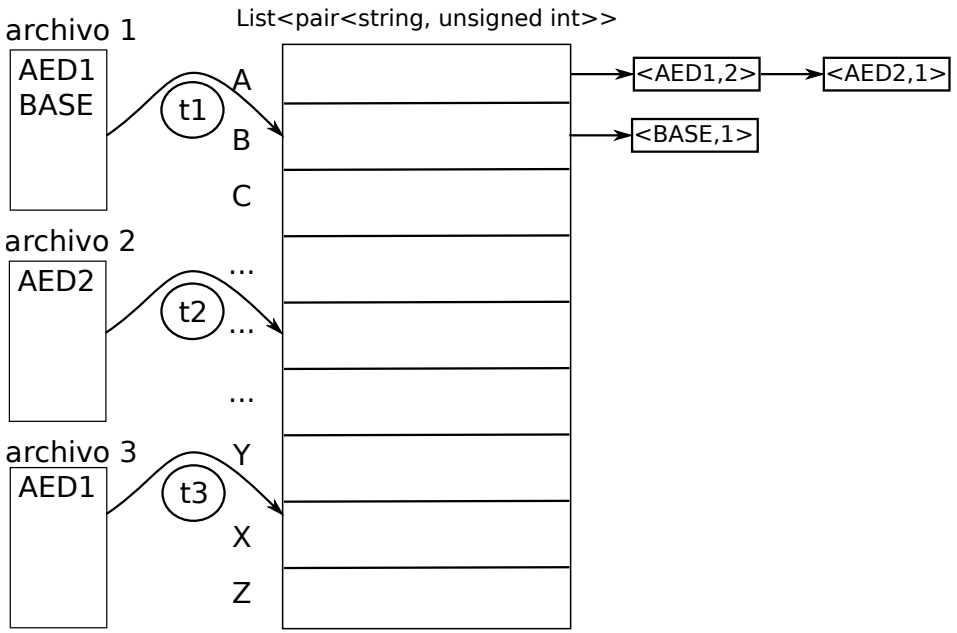


Ejercicio 2.b

```
ConcurrentHashMap countWordsOneThreadPerFile(
  list<string> filePaths)
```

Ejercicio 2.b

```
ConcurrentHashMap countWordsOneThreadPerFile(
  list<string> filePaths)
```

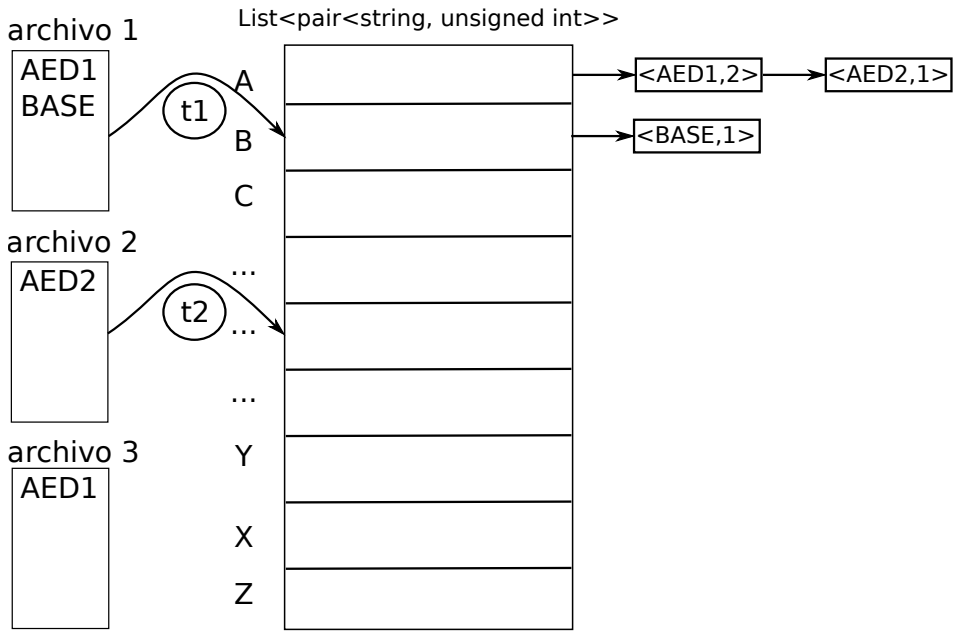


Ejercicio 2.c

```
ConcurrentHashMap countWordsArbitraryThreads(
  unsigned int n,
  list<string> filePaths)
```

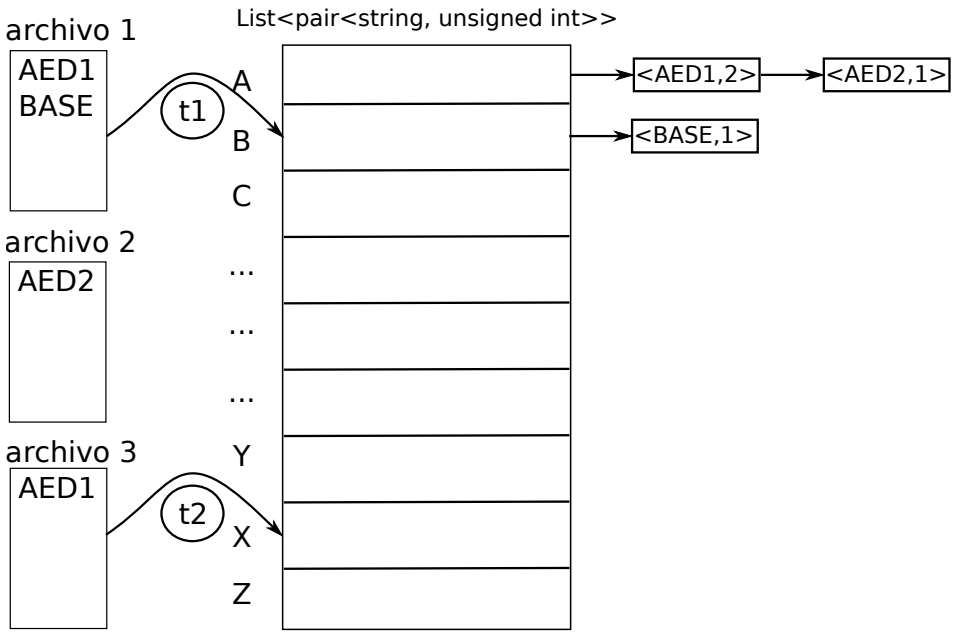
Ejercicio 2.c

```
ConcurrentHashMap countWordsArbitraryThreads(
  unsigned int n,
  list<string> filePaths)
```



Ejercicio 2.c

```
ConcurrentHashMap countWordsArbitraryThreads(
  unsigned int n,
  list<string> filePaths)
```

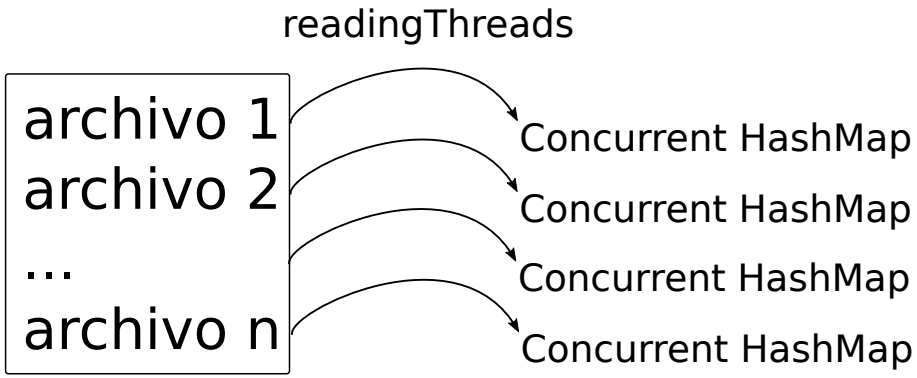


Ejercicio 2.d

```
pair<string, unsigned int> maximumOne(
  unsigned int readingThreads,
  unsigned int maxingThreads,
  list<string> filePaths)
```

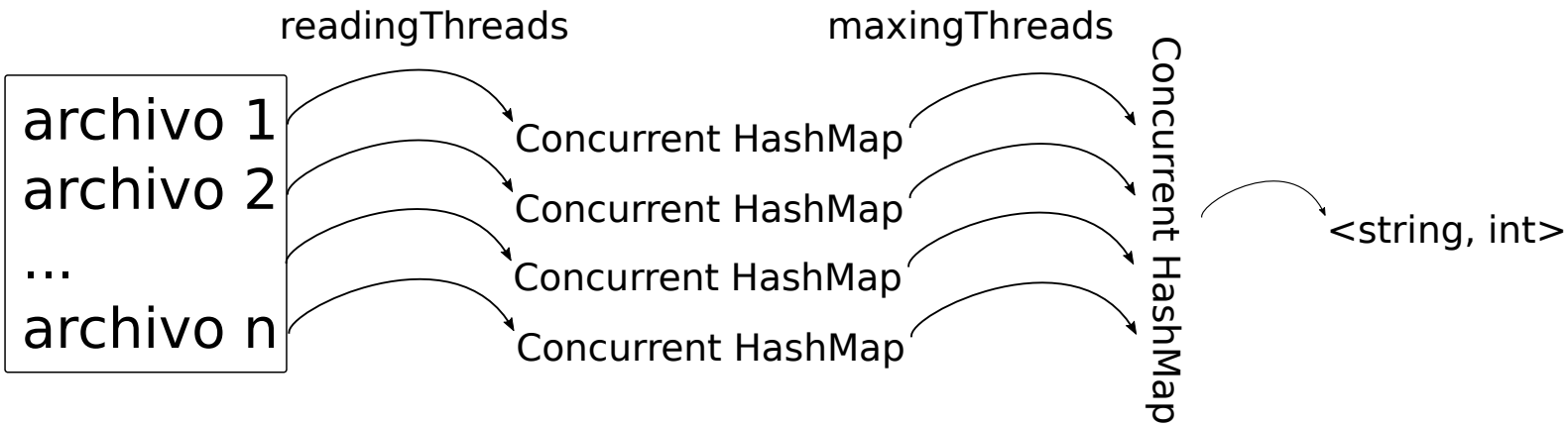
Ejercicio 2.d

```
pair<string, unsigned int> maximumOne(
  unsigned int readingThreads,
  unsigned int maxingThreads,
  list<string> filePaths)
```



Ejercicio 2.d

```
pair<string, unsigned int> maximumOne(
  unsigned int readingThreads,
  unsigned int maxingThreads,
  list<string> filePaths)
```

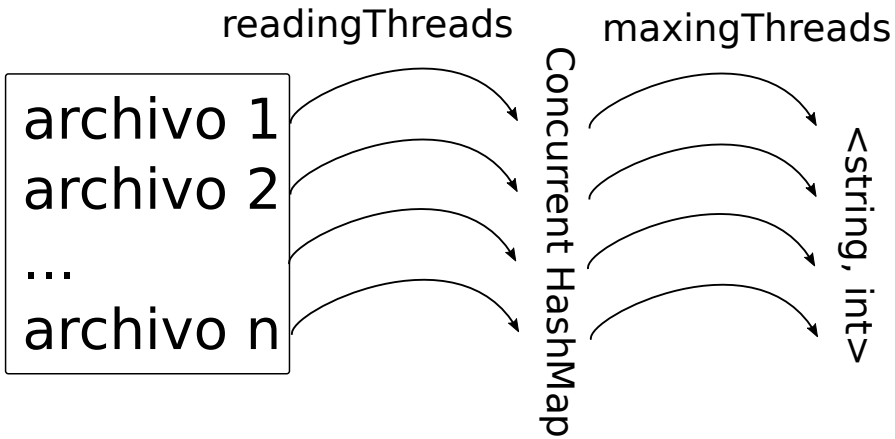


Ejercicio 2.e

```
pair<string, unsigned int> maximumTwo(
  unsigned int readingThreads,
  unsigned int maxingThreads,
  list<string> filePaths)
```

Ejercicio 2.e

```
pair<string, unsigned int> maximumTwo(
  unsigned int readingThreads,
  unsigned int maxingThreads,
  list<string> filePaths)
```



Consideraciones de la implementación

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

Ejercicio 3

Realizar un informe breve (max. 4 carillas) justificando la implementación realizada. No **copy-pastear** código en el informe.

Además, realizar pruebas para comparar la *performance* de los ejercicios `maximumOne` y `maximumTwo`. Agregar al informe los resultados obtenidos.

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?

Por último: algunas pautas de entrega

- Entrega electrónica vía mail a so-doc@dc.uba.ar con asunto `[SO;2018;C2;TP1]`. **NO enviar el TP a so-alu...**
- Enviar en el mail los datos de todos los integrantes del grupo y 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: 03/10/2018 (OJO! es miércoles)

¿Preguntas?