Presentación TP1

Sistemas Operativos
DC - UBA - FCEN

05 de abril de 2018

Problema a Resolver

- Queremos una estructura que, dado un texto (o varios), permita almacenar de manera **eficiente** las apariciones de cada palabra.

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. Diccionario implementado sobre una tabla de hash)

Estructura elegida

Realizaremos un **Concurrent HashMap** Qué es ?

- Es un hashmap (a.k.a. Diccionario implementado sobre una tabla de hash)
- Concurrente: Soporta accesos simultáneos manteniendo la consistencia.

Estructura elegida

Realizaremos un **Concurrent HashMap** Qué es ?

- Es un hashmap (a.k.a. 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)

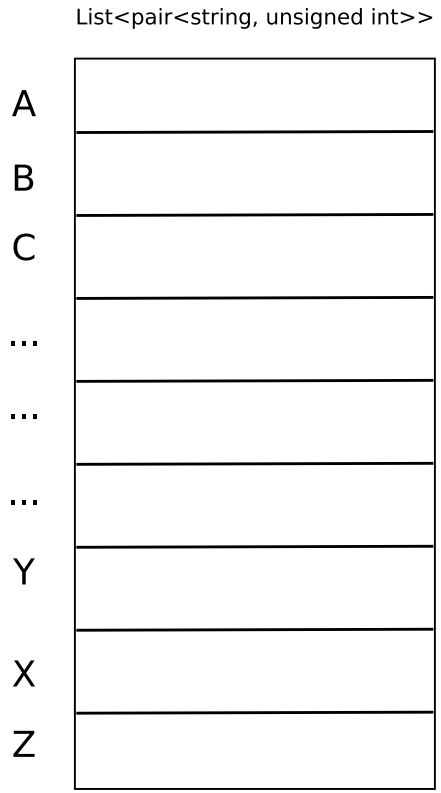
Estructura elegida

Realizaremos un **Concurrent HashMap** Qué es ?

- Es un hashmap (a.k.a. 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.

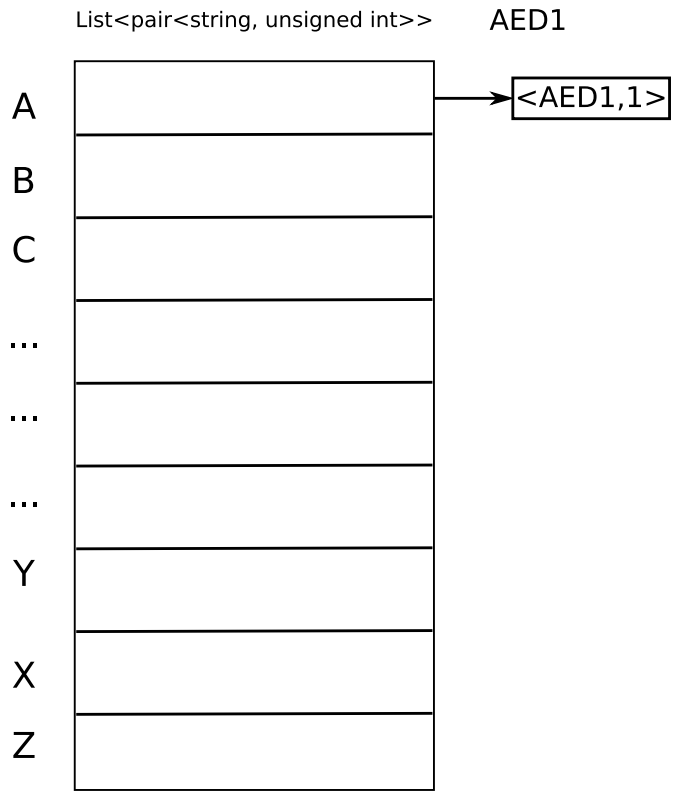
Operaciones

- `create()`



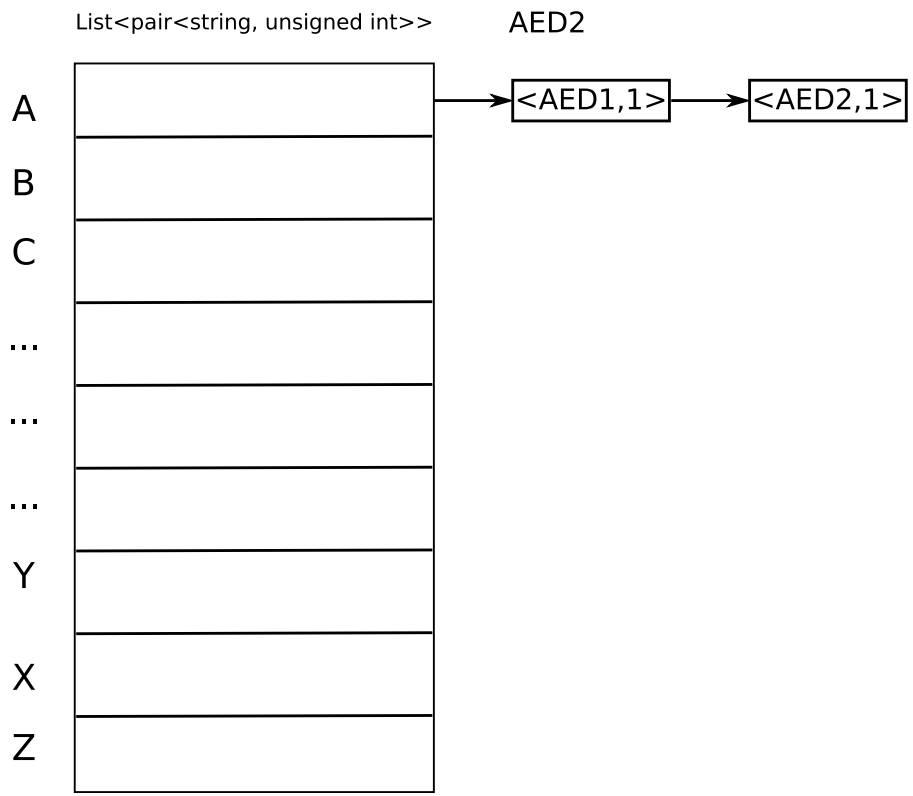
Operaciones

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



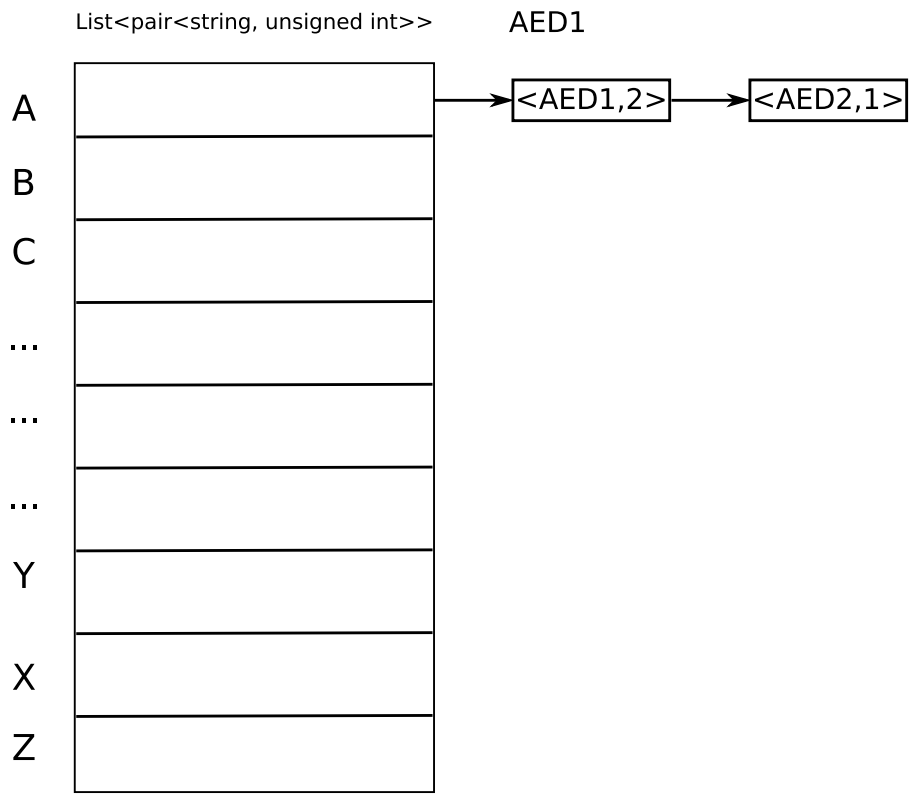
Operaciones

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



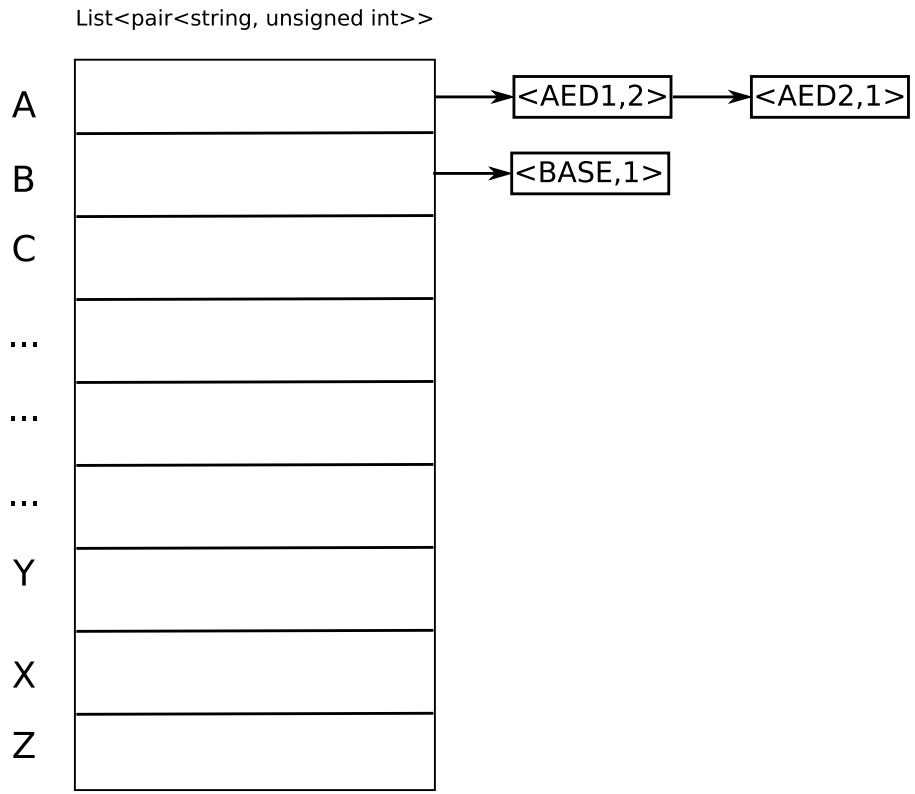
Operaciones

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



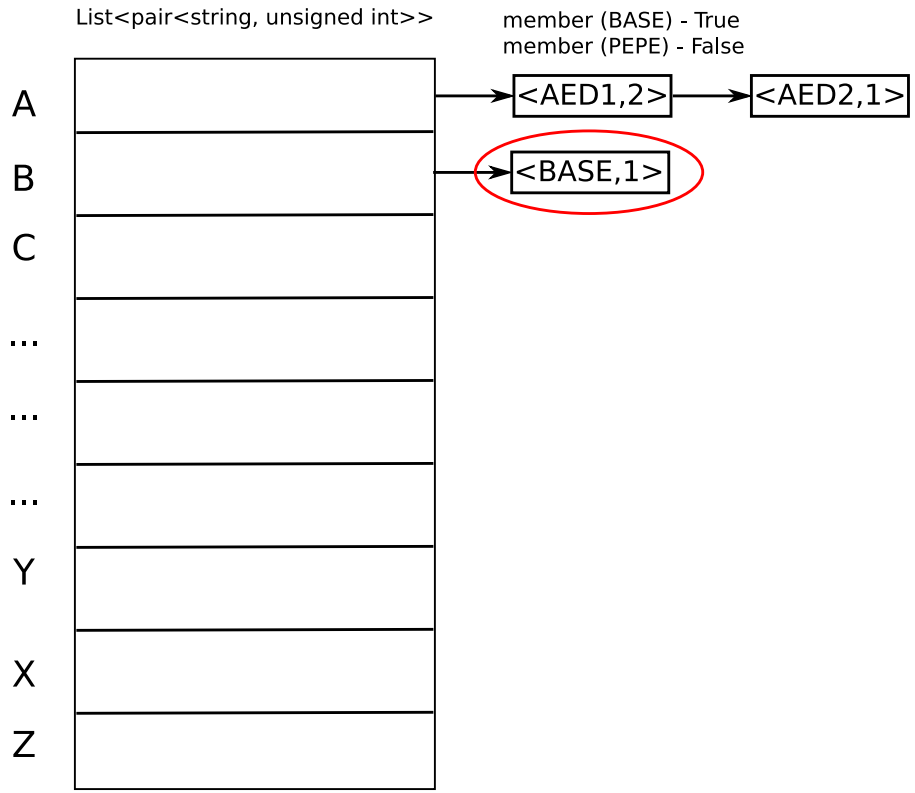
Operaciones

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



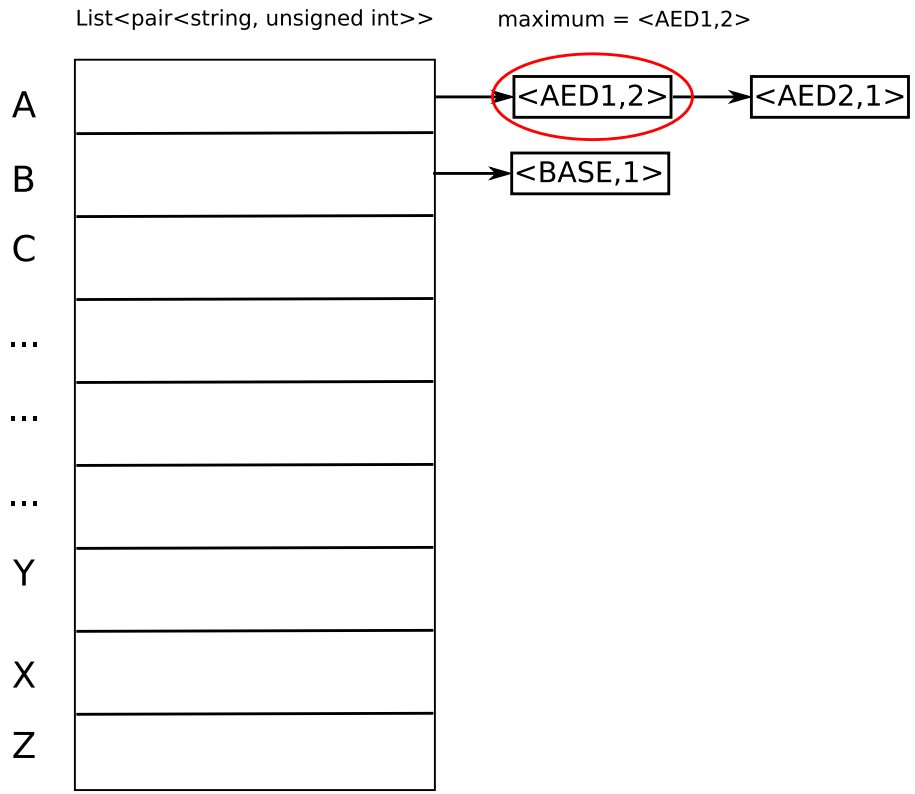
Operaciones

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



Operaciones

- `create()`
- `void addAndInc(string key)`
- `bool member(string key)`
- `pair<string, unsigned int> maximum(unsigned int nt)`



Ejercicios

- `ConcurrentHashMap()`.

Ejercicios

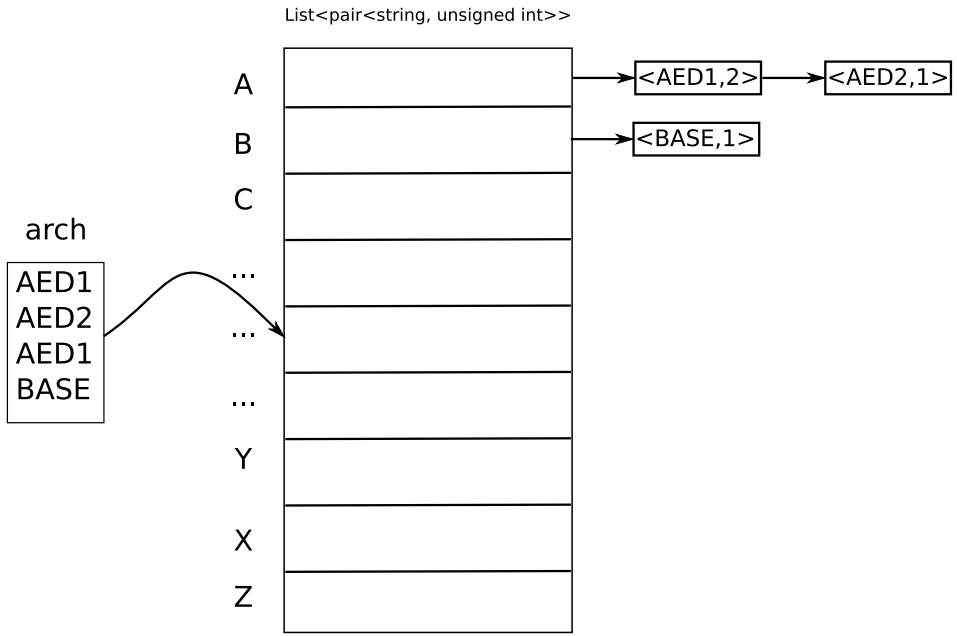
- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`

arch

AED1
AED2
AED1
BASE

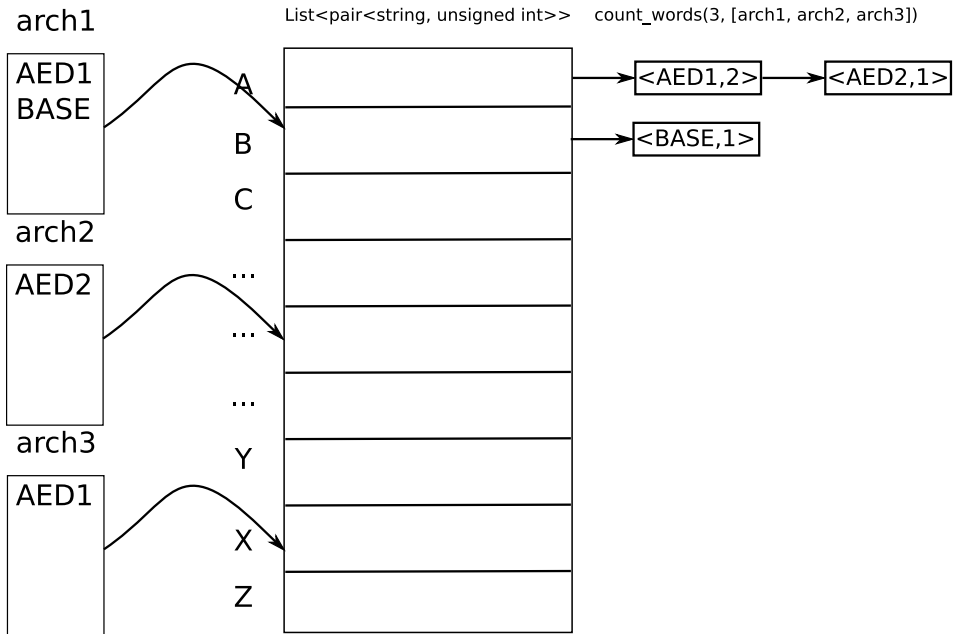
Ejercicios

- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`



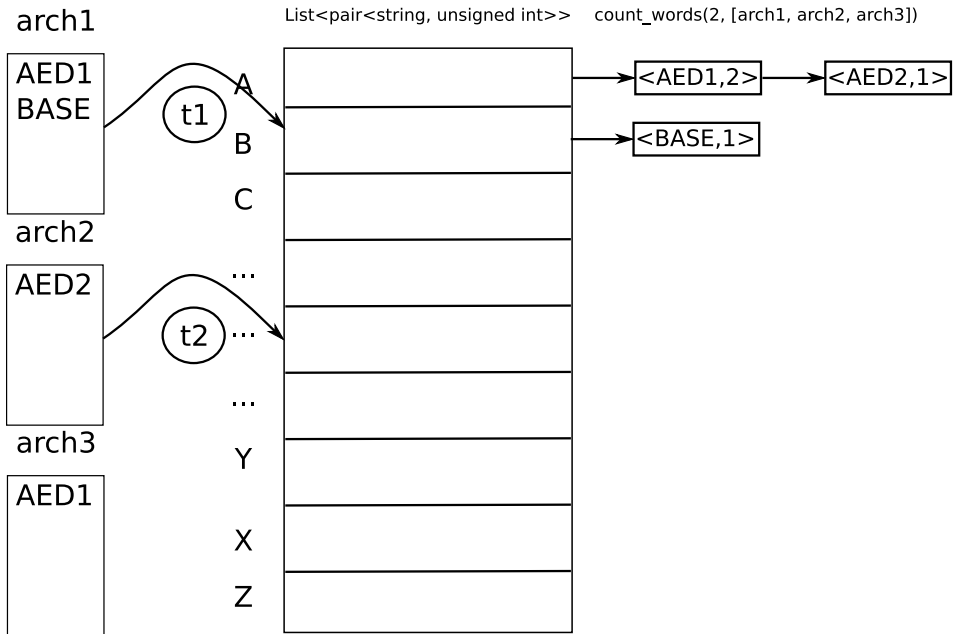
Ejercicios

- `ConcurrentHashMap().`
- `ConcurrentHashMap count_words(string arch)`
- `ConcurrentHashMap count_words(list<string> archs)`



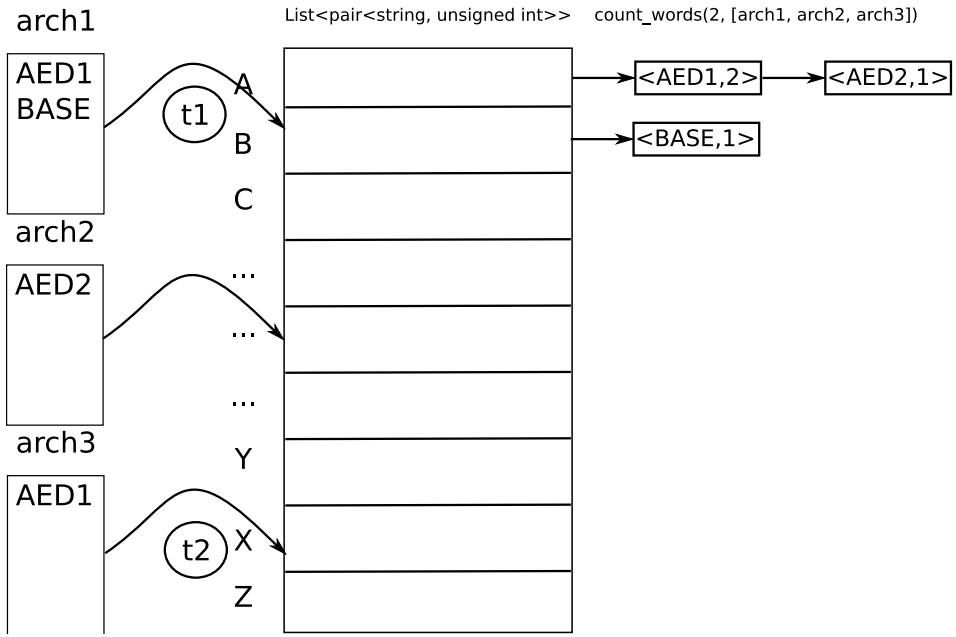
Ejercicios

- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`
- `ConcurrentHashMap count_words(list<string> archs)`
- `ConcurrentHashMap count_words(unsigned int p, list<string> archs)`
  $p < sizeof(archs)$



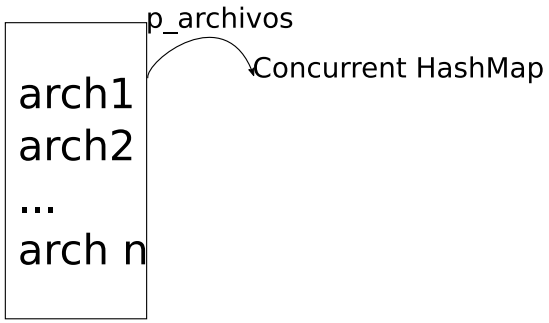
Ejercicios

- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`
- `ConcurrentHashMap count_words(list<string> archs)`
- `ConcurrentHashMap count_words(unsigned int p, list<string> archs)`
  $p < sizeof(archs)$



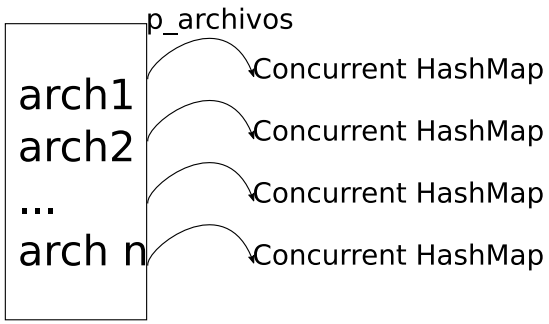
Ejercicios

- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`
- `ConcurrentHashMap count_words(list<string> archs)`
- `ConcurrentHashMap count_words(unsigned int p, list<string> archs)`
  $p < sizeof(archs)$
- `pair<string, unsigned int> maximum(unsigned int p_archivos, unsigned int p_maximos, list<string> archs)`



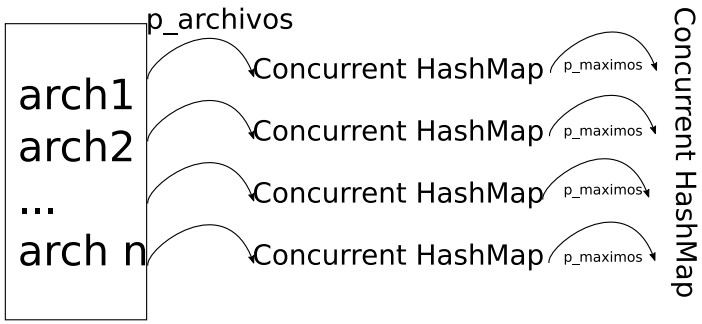
Ejercicios

- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`
- `ConcurrentHashMap count_words(list<string> archs)`
- `ConcurrentHashMap count_words(unsigned int p, list<string> archs)`
  $p < sizeof(archs)$
- `pair<string, unsigned int> maximum(unsigned int p_archivos, unsigned int p_maximos, list<string> archs)`



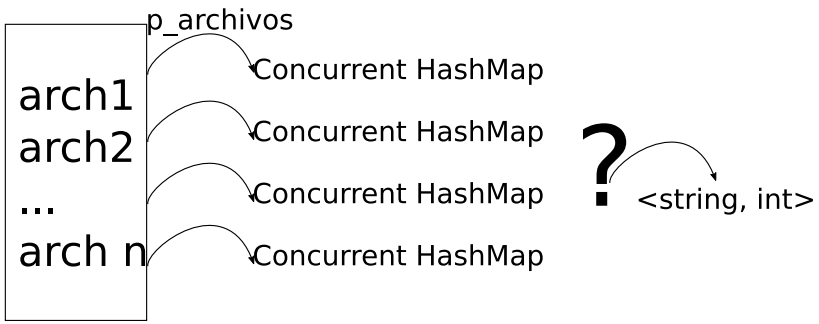
Ejercicios

- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`
- `ConcurrentHashMap count_words(list<string> archs)`
- `ConcurrentHashMap count_words(unsigned int p, list<string> archs)`
  $p < sizeof(archs)$
- `pair<string, unsigned int> maximum(unsigned int p_archivos, unsigned int p_maximos, list<string> archs)`



Ejercicios

- `ConcurrentHashMap()`.
- `ConcurrentHashMap count_words(string arch)`
- `ConcurrentHashMap count_words(list<string> archs)`
- `ConcurrentHashMap count_words(unsigned int p, list<string> archs)`
  $p < sizeof(archs)$
- `pair<string, unsigned int> maximum(unsigned int p_archivos, unsigned int p_maximos, list<string> archs)`



Por último: algunas pautas de entrega

- ★ Entrega electrónica vía formulario en la página de la materia o usando el link:
  `https://goo.gl/forms/r9gE6tUChBUu2Y6z1`
- ★ Completar con 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).
  3. Tests mostrando la correcta implementación.
  4. Makefile para correr los test agregados (se puede modificar el que ya está).

Por último: algunas pautas de entrega

- Fecha límite: 20/04/2018 (OJO! es viernes)
- Implementación libre de condiciones de carrera
- Ningún proceso deberá escribir un resultado ya resuelto por otro thread.
- Informe breve.

¿Preguntas?