

ESTRUCTURAS DE DATOS

DICCIONARIOS

El TAD Diccionario

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Motivación

- Leer un texto de entrada e imprimir el número de veces que aparece cada palabra.

Cuántas veces aparece cada palabra

David tomó la llave para entregársela a Laura. Esta última, no obstante, declinó hacer uso de la llave mientras que no fuera absolutamente necesario.

David 1
tomó 1
la 2
llave 2
para 1
...

contador para cada palabra. Cómo se almacenan esas palabras asociadas a sus respectivos contadores

- ¿Cómo almacenamos las palabras que nos encontramos?

Motivación

1a posibilidad

- Tabla en la que almacenamos el número de veces que aparece cada palabra encontrada hasta el momento.
- Con cada palabra recibida:
 - Si existe una entrada en la tabla con esa palabra, incrementamos su contador. ++
 - Si no, insertamos una nueva entrada con esa palabra y con su contador a 1.

Palabra	Contador
"David"	1
"tomó"	1
"la"	2
"llave"	2
...	...
"	1

¿Qué es un diccionario?

- Un tipo abstracto de datos que almacena un conjunto de pares.
- A cada par se le llama **entrada**. cada fila de la tabla.
- A la primera componente de cada par se le denomina **clave**.
- A la segunda componente se le denomina **valor asociado** a esa clave.
- No existen dos pares con la misma clave.

Esto va a ser muy útil para realizar búsquedas.

Cuando tenemos estructuras de datos de este tipo decimos que tenemos un diccionario.

Palabra	Contador
"David"	1
"tomó"	1
"la"	2
"llave"	2
...	...

clave

valor

Claves

Valores

Terminología

Nosotros en TP2 lo llamamos Map<>()

HashMap, TreeMap...

diccionarios

maps

tablas

associative arrays

arrays asociativos

dictionaries

symbol tables

Palabra	Contador
"David"	1
"tomó"	1
"la"	2
"llave"	2
...	...

También se llaman de esta forma los diccionarios

Modelo conceptual de diccionarios

ESTO ES COMO LO QUE EXPLICÓ
EL DE TP2. LOS MAPAS

- Sean:
 - K – conjunto de claves
 - V – conjunto de valores
- Un diccionario M es un conjunto de pares (k, v) , donde $k \in K, v \in V$.
- No existen pares $(k, v), (k, v') \in M$ tales que $v \neq v'$.

Se refiere a que no existen dos pares que tengan la misma clave k en el lado izquierdo tiene sentido

NOS OLVIDAMOS DE LAS TABLAS Y LOS CREAMOS
NOS LO IMAGINAMOS COMO CONJUNTOS.

Palabra	Contador
"David"	1
"tomó"	1
"la"	2
"llave"	2
...	...

NOSOTROS LAS REPRESENTAMOS MEJOR DE ESTA FORMA.
 $M = \{(\text{"David"}, 1), (\text{"tomó"}, 1), (\text{"la"}, 2), \dots\}$

COMO UN CONJUNTO M DE PARES (K, V)

Operaciones en el TAD Diccionario

- Constructoras:
 - Crear un diccionario vacío: **create_empty**
- Mutadoras:
 - Añadir una entrada al diccionario: **insert**
 - Eliminar una entrada del diccionario: **erase**
- Observadoras:
 - Saber si existe una entrada con una clave determinada: **contains**
 - Saber el valor asociado con una clave: **at**
 - Saber si el diccionario está vacío: **empty**
 - Saber el número de entradas del diccionario: **size**

Operaciones constructoras y mutadoras

$\{ \text{true} \}$

`create_empty()` $\rightarrow (M: \text{Map})$

$\{ \underline{M = \emptyset} \}$

VACÍO.

ESPECIFICACIÓN: ESTO ES DE FAL, ES DECIR CUAL ES LA PRECONDICIÓN Y LA POSTCONDICIÓN.

Creo que nosotros utilizabamos un MapEntry, que era un struct con clave valor y dos constructores.

Diccionario donde quiero meter la entrada

$\{ \text{true} \}$

`insert(k: Key, v: Value, M: Map)`

$M = \begin{cases} \text{old}(M) & \text{si } \exists v'. (k, v') \in \text{old}(M) \\ \text{old}(M) \cup \{(k, v)\} & \text{en otro caso} \end{cases}$

Si ya está la clave no la mete, si no está la mete.

$\{ \text{true} \}$

`erase(k: Key, M: Map)`

$M = \{ (k', v') \in \text{old}(M) \mid k' \neq k \}$

contiene todas las entradas del diccionario antiguo siempre que tengan una entrada distinta a la clave que queremos borrar.

Operaciones observadoras

$\{ \text{true} \}$

contains(k : Key, M : Map) \rightarrow (b : bool)

$\{ b \Leftrightarrow \exists v. (k, v) \in M \}$

Devuelve true si existe la clave.

$\{ \exists v'. (k, v') \in M \}$

la clave tiene que estar asociada algun valor dentro es decir la clave tiene que existir.

at(k : Key, M : Map) \rightarrow (v : value)

$\{ (k, v) \in M \}$

Devuelve el valor asociado a una clave concreta.

$\{ \text{true} \}$

empty(M : Map) \rightarrow (b : bool)

$\{ b \Leftrightarrow M = \emptyset \}$

true si el diccionario está vacío.

$\text{size}() == 0$

$\{ \text{true} \}$

size(M : Map) \rightarrow (n : int)

$\{ n = |M| \}$

cardinal o número de entradas que tiene nuestro diccionario.

Interfaz en C++

```
template <typename K, typename V>
```

```
class map {
```

```
public:
```

```
    map();
```

```
    map(const map &other);
```

```
    ~map();
```

CONSTRUCTORES

```
    void insert(const K &key, const V &value);
```

```
    void erase(const K &key);
```

```
    bool contains(const K &key) const;
```

```
    const V & at(const K &key) const;
```

```
    V & at(const K &key);
```

ACCESO AL VALOR ASOCIADO A UNA DETERMINADA CLAVE.

```
    int size() const;
```

```
    bool empty() const;
```

```
private:
```

```
    // ...
```

```
};
```

Interfaz en C++

```
template <typename K, typename V>
```

```
class map {
```

```
public:
```

```
    map();
```

```
    map(const map &other);
```

```
    ~map();
```

parámetro que agrupa ambas cosas

```
    void insert(const map_entry &entry);
```

```
    void erase(const K &key);
```

```
    bool contains(const K &key) const;
```

```
    const V & at(const K &key) const; Si no queremos modificar un dato
```

```
    V & at(const K &key); si queremos modificar un dato
```

```
    int size() const;
```

```
    bool empty() const;
```

```
private:
```

```
    // ...
```

```
};
```

Es razonable.

```
struct map_entry {
```

```
    K key;
```

```
    V value;
```

```
};
```

Esto es para acercar un poco a la definición de las librerías estándar de c++

Ejemplo

diccionario de personas vacío.

```
map<string, int> personas;
```

$personas = \emptyset$

```
personas.insert({"Aarón", 42});  
personas.insert({"Estela", 41});
```

$personas = \{("Aarón", 42), ("Estela", 41)\}$

```
cout << personas.contains("Aarón") << endl;  
cout << personas.at("Aarón") << endl;
```

true

42

Se ha insertado en orden pero en principio no tenemos que conocer el orden

```
personas.insert({"Carlos", 31});
```

$personas = \{("Aarón", 42), ("Carlos", 31), ("Estela", 41)\}$

```
personas.erase("Estela");
```

$personas = \{("Aarón", 42), ("Carlos", 31)\}$

```
personas.at("Aarón") = 43;
```

$personas = \{("Aarón", 43), ("Carlos", 31)\}$

La versión NO CONSTANTE.

también puedes modificar con at sin la const.

Ejemplo

```
string palabra;  
map<string, int> dicc;
```

```
cin >> palabra;
```

```
while (!cin.eof()) {  
    if (dicc.contains(palabra)) {  
        dicc.at(palabra)++;  
    } else {  
        dicc.insert({palabra, 1});  
    }
```

aumentamos

insertamos una nueva entrada

```
    cin >> palabra;  
}
```

Recordamos que recibe un struct map_entry que contiene dos constructores (aunque no aparecen ahí) y los atributos Clave K y Valor V.

Dos implementaciones

- Mediante árboles binarios de búsqueda (MapTree)
- Mediante tablas hash (MapHash)

Esta semana.

La semana que viene.

