Estructuras de Datos y Algoritmos

TAD diccionario

Lección 10

Universidad Zaragoza

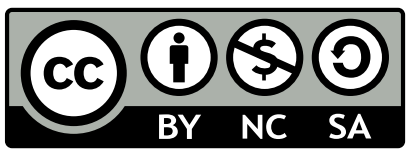
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Curso 2023/2024

Grado en Ingeniería Informática

Universidad de Zaragoza

Aula 0.04, Edificio Agustín de Betancourt



Índice

Universidad Zaragoza

Índice

Universidad Zaragoza

TAD diccionario

```
espec diccionarios
    usa boleanos, naturales
    parámetros formales
        géneros clave, valor
        operación
        {suponemos que en el género de las claves hay definidas funciones de comparación "=", "<"}
            _=_ : clave c1, clave c2 -> booleano
            _<_ : clave c1, clave c2 -> booleano
    fpf
género diccionario
    {Los valores del TAD representan conjuntos de pares (clave,valor) en los que
        no se permiten claves repetidas}
    operaciones
        crear: -> diccionario
        {Devuelve un diccionario vacío, sin elementos}
        añadir: diccionario d, clave c, valor v -> diccionario
        {Si en d no hay ningún par con clave c, devuelve el diccionario resultante de añadir el par (c, v) a d;
            si en d hay un par (c, v'), entonces devuelve el resultado de sustituirlo por el par (c, v)}
        pertenece?: clave c, diccionario d -> booleano
        {Devuelve verdad si y sólo si en d hay algún par (c, v)}
        parcial obtenerValor: clave c, diccionario d -> valor
        {Devuelve el valor asociado a la clave c en d.
        Parcial: la operación no está definida si c no está en d}
        quitar: clave c, diccionario d -> diccionario
        {Si c está en d, devuelve el diccionario resultante de borrar c y su valor de d;
            si c no está en d, devuelve un diccionario igual a d}
        cardinal: diccionario d -> natural
        {Devuelve el no de elementos en el diccionario d}
        esVacío?: diccionario d -> booleano
        {Devuelve verdad si y sólo si d no tiene elementos}
fespec
```

Universidad Zaragoza

Índice

Universidad Zaragoza

TAD diccionario
Posibles implementaciones

- Operaciones `pertenece?` y `obtenerValor` en una única operación
- Faltarían las operaciones de `duplicar`, `iguales` y para el iterador
  - Y operaciones para liberación de memoria, si la implementación es dinámica
- Implementación estática vs. dinámica (e.g., con listas ordenadas)

TAD diccionario

Implementación estática vs. dinámica – ventajas y desventajas

Implementación estática: aproximaciones

- Basadas en un vectores (de `max` componentes) de registros
- El registro almacenado es el par clave, valor (e.g., `par(c, v)`)

TAD diccionario

Implementación estática vs. dinámica – ventajas y desventajas

Implementación estática: aproximaciones

- Basadas en un vectores (de `max` componentes) de registros
- El registro almacenado es el par clave, valor (e.g., `par(c, v)`)

Representación desordenada

- Componentes insertadas **sin ningún criterio de ordenación**
- Costes:
  - Tiempo de buscar, borrar, insertar (caso peor): $O(n)$, siendo n el número de elementos almacenados
  - Espacio: $O(\texttt{max})$

Universidad Zaragoza

TAD diccionario

Implementación estática vs. dinámica – ventajas y desventajas

Implementación estática: aproximaciones

- Basadas en un vectores (de `max` componentes) de registros
- El registro almacenado es el par clave, valor (e.g., `par(c, v)`)

Representación desordenada

- Componentes insertadas **sin ningún criterio de ordenación**
- Costes:
  - Tiempo de buscar, borrar, insertar (caso peor): $O(n)$, siendo n el número de elementos almacenados
  - Espacio: $O(\texttt{max})$

Representación ordenada

- Componentes **ordenadas por valores crecientes de la clave**
- Costes:
  - Tiempo de borrar, insertar (caso peor): $O(n)$, siendo n el número de elementos almacenados
  - Tiempo de buscar (caso peor): $O(\log n)$ (búsqueda dicotómica)
  - Espacio: $O(\texttt{max})$

Universidad Zaragoza

TAD diccionario

Implementación dinámica

```
módulo genérico diccionarios
parámetros
    tipos clave, valor
    con función "="(c1, c2: clave) devuelve booleano
    con función "<"(c1, c2: clave) devuelve booleano
exporta
    tipo diccionario
    procedimiento crear(sal d: diccionario)
    procedimiento añadir(e/s d: diccionario; ent c: clave; ent v: valor)
    procedimiento buscar(ent d: diccionario; ent c: clave; sal éxito: booleano; sal v: valor)
    {pertenece? y obtenerValor se implementan en una única operación}
    procedimiento quitar(ent c: clave; e/s d: diccionario)
    función cardinal(d: diccionario) devuelve natural
    función esVacío?(d: diccionario) devuelve booleano

    {Operaciones para duplicar, comparar y liberar}
    procedimiento duplicar(sal dSal: diccionario; ent dEnt: diccionario)
    procedimiento liberar(e/s d: diccionario)
    función iguales?(d1, d2: diccionario) devuelve booleano

    {Operaciones del Iterador}
    procedimiento iniciarIterador(e/s d: diccionario)
    función existeSiguiente?(d: diccionario) devuelve booleano
    procedimiento siguiente(e/s d: diccionario; sal c: clave; sal v: valor; sal error: booleano)
implementación
...
```

Universidad Zaragoza

```
...
{El diccionario se representa en memoria dinámica con una lista
    enlazada tal que las claves se mantienen ordenadas ("<").}
    tipos punteroCelda = ↑unaCelda;
        unaCelda = registro
                       laClave: clave;
                       elValor: valor;
                       sig:     punteroCelda
                   freg
        diccionario = registro
                          primera: punteroCelda;
                          tamaño:  natural
                      freg

    procedimiento crear(sal d: diccionario)
    principio
        d.primera := nil;
        d.tamaño  := 0
    fin

    procedimiento añadir(e/s d: diccionario; ent c: clave; ent v: valor)
    variables pAux, nuevo: punteroCelda
    principio
        si d.primera = nil entonces {lista vacía}
            nuevoDato(d.primera);
            d.primera↑.laClave := c;
            d.primera↑.elValor := v;
            d.primera↑.sig := nil;
            d.tamaño:=1
        sino
            si c < d.primera↑.laClave entonces {inserción al principio}
                pAux := d.primera;
                nuevoDato(d.primera);
                d.primera↑.laClave := c;
                d.primera↑.elValor := v;
                d.primera↑.sig := pAux;
                d.tamaño := d.tamaño + 1
            sino
                ...
```

```
            ...
            si c = d.primera↑.laClave entonces {ya existe, cambiar valor}
                d.primera↑.elValor := v
            sino {c > d.primera↑.laClave => buscar punto de inserción}
                pAux := d.primera;
                mientrasQue pAux↑.sig ≠ nil andThen pAux↑.sig↑.laClave < c hacer
                    pAux := pAux↑.sig
                fmq;
                si pAux↑.sig ≠ nil andThen c = pAux↑.sig↑.laClave entonces
                    {clave ya existe, cambiar valor}
                    pAux↑.sig↑.elValor := v
                sino {inserción entre dos registros o al final}
                    nuevoDato(nuevo);
                    nuevo↑.laClave := c;
                    nuevo↑.elValor := v;
                    nuevo↑.sig := pAux↑.sig;
                    pAux↑.sig := nuevo;
                    d.tamaño := d.tamaño + 1
                fsi
            fsi
        fsi
    fsi
fin {de añadir}

{ Otra posible implementación}
procedimiento añadir_alternativa(e/s d: diccionario; ent c: clave; ent v: valor)
variables pAux, nuevo: punteroCelda; celdaAux: unaCelda;
principio
    si d.primera = nil entonces {lista vacía}
        nuevoDato(d.primera);
        d.primera↑.laClave := c;
        d.primera↑.elValor := v;
        d.primera↑.sig := nil;
        d.tamaño := 1;
    sino
        ...
```

```
        ...
        si c < d.primera↑.laClave entonces {inserción al principio}
            pAux := d.primera;
            nuevoDato(d.primera);
            d.primera↑.laClave := c;
            d.primera↑.elValor := v;
            d.primera↑.sig := pAux;
            d.tamaño := d.tamaño + 1
        sino {buscar punto de inserción}
            pAux:= d.primera;
            mientrasQue pAux↑.laClave < c and pAux↑.sig ≠ nil hacer
                pAux := pAux↑.sig;
            fmq;
            si c < pAux↑.laClave entonces {inserción entre dos registros}
                celdaAux.laClave := c;
                celdaAux.elValor := v;
                nuevoDato(nuevo);
                nuevo↑ := pAux↑;
                pAux↑ := celdaAux;
                pAux↑.sig := nuevo;
                d.tamaño := d.tamaño + 1
            sino
                si c = pAux↑.laClave entonces {clave ya existe, cambiar valor}
                    pAux↑.elValor := v
                sino {inserción al final}
                    nuevoDato(pAux↑.sig);
                    pAux:=pAux↑.sig;
                    pAux↑.laClave := c;
                    pAux↑.elValor := v;
                    pAux↑.sig := nil;
                    d.tamaño := d.tamaño + 1
                fsi
            fsi
        fsi
    fsi
fin {de añadir_alternativa}
```

```
procedimiento buscar(ent d: diccionario; ent c: clave; sal éxito: booleano; sal v: valor)
variable pAux: punteroCelda
principio
    pAux := d.primera;
    mientrasQue pAux ≠ nil andThen pAux↑.laClave < c hacer
        pAux := pAux↑.sig
    fmq;
    si pAux = nil entonces
        éxito := falso
    sino
        si pAux↑.laClave = c entonces
            v := pAux↑.elValor;
            éxito := verdad
        sino
            éxito := falso
        fsi
    fsi
fin

procedimiento quitar(ent c: clave; e/s d: diccionario)
variables pAux1,pAux2:punteroCelda; parar:booleano
principio
    si d.primera ≠ nil entonces {caso contrario, no hacer nada}
        si d.primera↑.laClave <= c entonces {caso contrario, no hacer nada}
            si d.primera↑.laClave = c entonces {borrar el primer elemento}
                pAux1 := d.primera;
                d.primera := d.primera↑.sig;
                disponer(pAux1)
                d.tamaño := d.tamaño - 1
            sino { d.primera↑.laClave < c
                    => buscar la clave c a partir del segundo elemento}
                parar := falso;
                pAux1 := d.primera↑.sig;
                pAux2 := d.primera;
                mientrasQue pAux1 ≠ nil and not parar hacer
                    si c < pAux1↑.laClave entonces {la clave no está, no hacer nada}
                        parar := verdad
                    sino_si c = pAux1↑.laClave entonces {borrar el registro}
                        pAux2↑.sig := pAux1↑.sig;
                        disponer(pAux1);
                        parar := verdad;
                        d.tamaño := d.tamaño - 1
                    sino {pAux1↑.laClave < c => avanzar}
                        pAux2 := pAux1;
                        pAux1 := pAux1↑.sig
                    fsi
                fmq
            fsi
        fsi
    fsi
fin
```

```
función cardinal(d: diccionario) devuelve natural
principio
    devuelve d.tamaño
fin
función esVacío?(d: diccionario) devuelve booleano
principio
    devuelve d.tamaño = 0
fin

procedimiento duplicar(sal dSal: diccionario; ent dEnt: diccionario)
variables pAuxEnt, pAuxSal: punteroCelda
principio
    si esVacío?(dEnt) entonces crear(dSal)
    sino {dEnt no vacío => tiene una primera celda}
        nuevoDato(dSal.primera);
        dSal.primera↑.laClave := dEnt.primera↑.laClave;
        dSal.primera↑.elValor := dEnt.primera↑.elValor;
        pAuxEnt := dEnt.primera↑.sig;
        pAuxSal := dSal.primera;
        mientrasQue pAuxEnt ≠ nil hacer
            nuevoDato(pAuxSal↑.sig);
            pAuxSal := pAuxSal↑.sig;
            pAuxSal↑.laClave := pAuxEnt↑.laClave;
            pAuxSal↑.elValor := pAuxEnt↑.elValor;
            pAuxEnt := pAuxEnt↑.sig
        fmq;
        pAuxSal↑.sig := nil;
        dSal.tamaño := dEnt.tamaño
    fsi
fin
...
```

```
procedimiento liberar(e/s d: diccionario)
variable pAux: punteroCelda
principio
    pAux := d.primera;
    mientrasQue pAux ≠ nil hacer
        d.primera := d.primera↑.sig;
        disponer(pAux);
        pAux := d.primera
    fmq;
    crear(d)
fin

función iguales?(d1, d2: diccionario) devuelve booleano
variables pAux1, pAux2: punteroCelda; igual: booleano
principio
    si esVacío?(d1) and esVacía?(d2) entonces devuelve verdad
    sino_si cardinal(d1) ≠ cardinal(d2) entonces devuelve falso
    sino {ambos tienen el mismo número (no nulo) de claves}
        igual := verdad;
        pAux1 := d1.primera;
        pAux2 := d2.primera;
        mientrasQue igual and pAux1 ≠ nil hacer
            igual := (pAux1↑.laClave = pAux2↑.laClave) and
                        (pAux1↑.elValor = pAux2↑.elValor);
            pAux1 := pAux1↑.sig;
            pAux2 := pAux2↑.sig
        fmq;
        devuelve igual
    fsi
fin
...
```

TAD diccionario – implementación del iterador

Requiere cambiar la implementación de `diccionario`:

```
diccionario = registro
                  primera, actual : punteroCelda;
                  tamaño: natural
              freg
```

Nuevo campo para el iterador

TAD diccionario – implementación del iterador

Requiere cambiar la implementación de `diccionario`:

```
diccionario = registro
                  primera, actual : punteroCelda;
                  tamaño: natural
              freg
```

Nuevo campo para el iterador

```
procedimiento iniciarIterador(e/s d: diccionario)
principio
    d.actual := d.primera
fin

función existeSiguiente?(d: diccionario) devuelve booleano
principio
    devuelve d.actual ≠ nil
fin

procedimiento siguiente(e/s d: diccionario; sal c: clave; sal v: valor; sal error: booleano)
principio
    si existeSiguiente?(d) entonces
        error := falso;
        c := d.actual↑.laClave;
        v := d.actual↑.elValor;
        d.actual := d.actual↑.sig
    sino
        error := verdad
    fsi
fin
```

Universidad Zaragoza

Trabajo encargado

Leer lección 10 de los apuntes.

Ejercicios hoja 1

Estructuras de Datos y Algoritmos

TAD diccionario

Lección 10

Universidad Zaragoza

1542

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, España

Curso 2023/2024

Grado en Ingeniería Informática

Universidad de Zaragoza

Aula 0.04, Edificio Agustín de Betancourt