Datu egiturak

Hash taulak

Aurkibidea

- ***Zer dira?***
  - Zergatik dira garrantzitsuak
  - Zertarako erabiltzen dira
  - Ezaugarriak
- ***Metodo nagusiak***
- ***Inplementazio adibideak***
  - Hash taula itxia
  - Hash taula irekia

Motibazioa

- Gako baten bidez bereizi daitezkeen elementuen multzo handiak kokatu eta atzitzeko beharra.
- Adibidez:
  - Hiztegi bateko definizioak
  - Datu pertsonalen erregistroak
  - Katalogo industrialetako elementuak: argitalpenak, mekanika, ...
  - Konpiladoreek kudeatzen dituzten sinboloak
  - Jokoetako egoerak
  - Datu-multzo handiak

Hash taulak

Zergatik dira garrantzitsuak: Helburuak

Elementu talde batean, oinarrizko eragiketak era eraginkorrenean exekutatzen dira:

- **Bilaketa**
- **Ezabaketa**
- **Txertaketa**

Hash taulen helburua eragiketa guztiak **ordena konstantean** egitea da O(1)

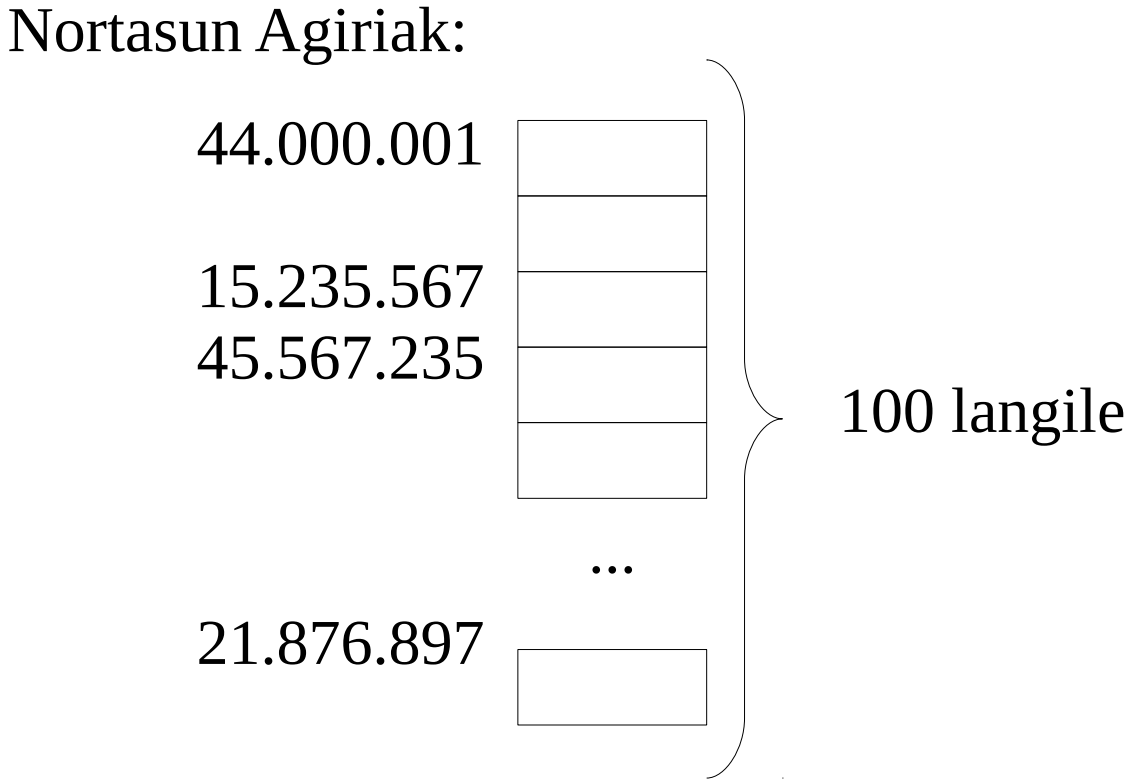
Hash Taulak

Zer dira? Zertarako erabiltzen dira?

- Datu egitura honek elementu multzo bat **indexatzen** du funtzio bat erabiliz (hash-funtzioa)

- **Array**-a da erabiltzen den datu egitura

Hash Taulak : Adibideak



Hash Taulak : Adibideak

Hiztegia:

katua	cat
txakurra	dog
etxea	house
	...
kaixo	hello

Hash taulak

Zer dira? Zertarako erabiltzen dira?

(katua, cat)
(txakurra, dog)
(etxea, house)
(kaixo, hello)

HASH FUNTZIOA

f(String) = n

HASH TAULA

1	
2	(katua, cat)
3	
4	
5	(txakurra, dog)
6	
7	(etxea, house)
8	
9	
10	(kaixo, hello)
11	
12	

Adibidea:

f(katua) = 2
f(txakurra) = 5
f(etxea) = 7

Hash taulak
hash funtzioa

- Elementuaren gakoa erabiliz array-aren indize bat itzultzen du

  f(gakoa) = Array posizioa

- Ezaugarriak (desiragarriak)
  - Elementu bakoitza posizio desberdinean
  - Konplexutasun ordena konstantea
  -
- Hash funtzioak gako bat hartuta zenbaki positibo bat lortu behar du. Adibidez:
  - hash(“JAVA”) = 10+1+23+1 = 35
  - Hash(“15564465”) = 36

Hash taulak: hash funtzioa Adibidea

- Hiztegi bateko 50.000 definizio gorde nahi ditugu. Definitzen den hitza gakoa da, eta definizioa balioa.

- 50.000 posizioko taula deifini genezake, eta hitz bakoitzari taulako indize bat esleitu

- Nola lotu hitz bat indize batekin?
  - Hash: Hitza -> indizea

Hash taulak: hash funtzioa Adibidea. Lehen saiakera

- Demagun hitzak minuskuletan daudela
- Letra bakoitza zenbaki batekin kode daiteke a=1, b=2, ..., z=27), eta hitz bakoitzari bere letren kodeen batura esleituko diogu:

  hash1(sei) = 20 + 5 + 9 = 34
- Arazoak:
  - hash1(baita) = 2 + 1 + 9 + 21 + 1 = 34
  - hash1(arpa) = hash1(para) = hash1(rapa)
  - 10 letrako hitzetara mugatzen baldin bagara, indizerik handiena 270 da. Gorde nahi diren 50.000 hitzetatik oso urrun!
  - Hitz askok indize bera izango dute lotuta

Hash taulak: hash funtzioa Adibidea. Bigarren saiakera

- Letra bakoitza zenbaki batekin kode daiteke a=1, b=2, ..., z=27), eta hitz bakoitzari kode hau esleituko diogu: “27 oinarrian dagokion zenbakia”

$$\text{hash2(baita)} = 2x27^4 + 1x27^3 + 9x27^2 + 21x27^1 + 1x27^0 =$$
$$1062882 + 19683 + 6561 + 567 + 1 = 1089694$$

- Arazoa: 10 letretako hitz batek indize handiegia izango luke: $27^9 > 7.000.000.000.000$

Hash taulak: hash funtzioa Adibidea. Hirugarren saiakera

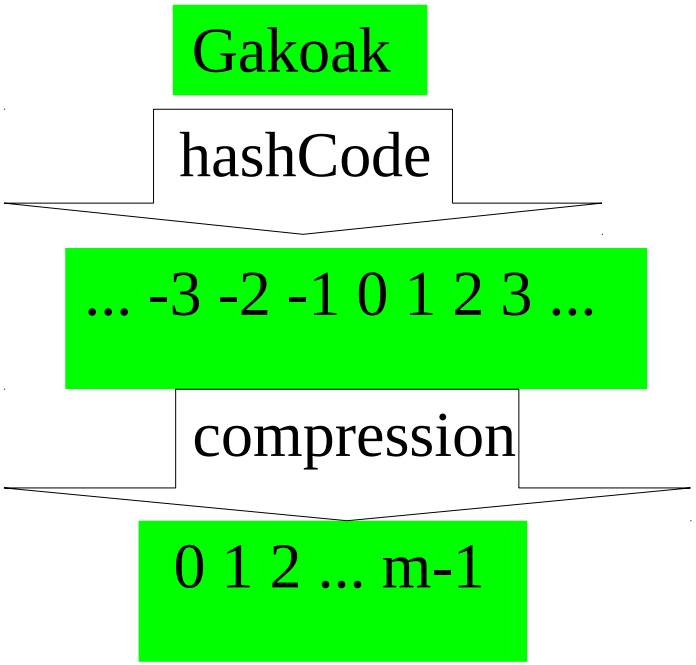
- 50.000 hitz gordetzeko, adibidez, 100.000 elementuko taula har dezakegu
- Zenbaki handi bat “konprimitu” dezakegu, 0..99.999 eremuan egoteko, zatiketaren hondarra erabiliz:
  - hash2(clase)%100.000 = 33.508
- Ez dira arazo guztiak ekiditzen:
  - 1733580%100.000 = 2433580%100.000

Hash taulak: hash funtzio baten forma tipikoa

– hashCode(baita) = $2x27^4 + 1x27^3 + 9x27^2 + 21x27^1 + 1x27^0$= 1089694

– compression(1089694) = 1089694 % 100.000 = 89.694

– hash(baita) = compression(hashCode(baita))



Hash taulak: hash funtzioa Adibidea.

- String motarako definituta dagoen hashCode funtzioa:

$$h(s) = \sum_{i=0}^{n-1} s[i] \cdot 31^{n-1-i}$$

- Adibidez: “Ea” eta “FB” kateek kode bera emango dute:

$$69 \times 31 + 97 = 70 \times 31 + 66$$

Hash taulak

Metodo nagusiak (interfazea)

public interface Map<K, V>

Metodoak	***Esanahia***
V **put (**K key, V value**)**	Gakoa eta balioa txertatzen ditu hash taulan. Gakoa badago, balio zaharra aldatzen du balio berriarengatik. Kasu honetan balio zaharra itzultzen du. Gakoa ez badago ***null*** itzultzen du.
V **get** (K key)	Gakoari dagokion balioa itzultzen du, edo ***null*** ez badago.
V **remove**(K key)	Gakoa eta bere balioa ezabatzen ditu. Gakoari dagokion balioa itzultzen du edo ***null*** existitzen ez bada.
void **clear** ()	Elementu guztiak ezabatzen ditu.
boolean **isEmpty**()	True hutsa bada, edo false beste kasuan.
boolean **containsKey** (K key)	True gakoa taulan badago, false beste kasuan.
int **size** ()	Zerrendaren elementu kopurua itzultzen du

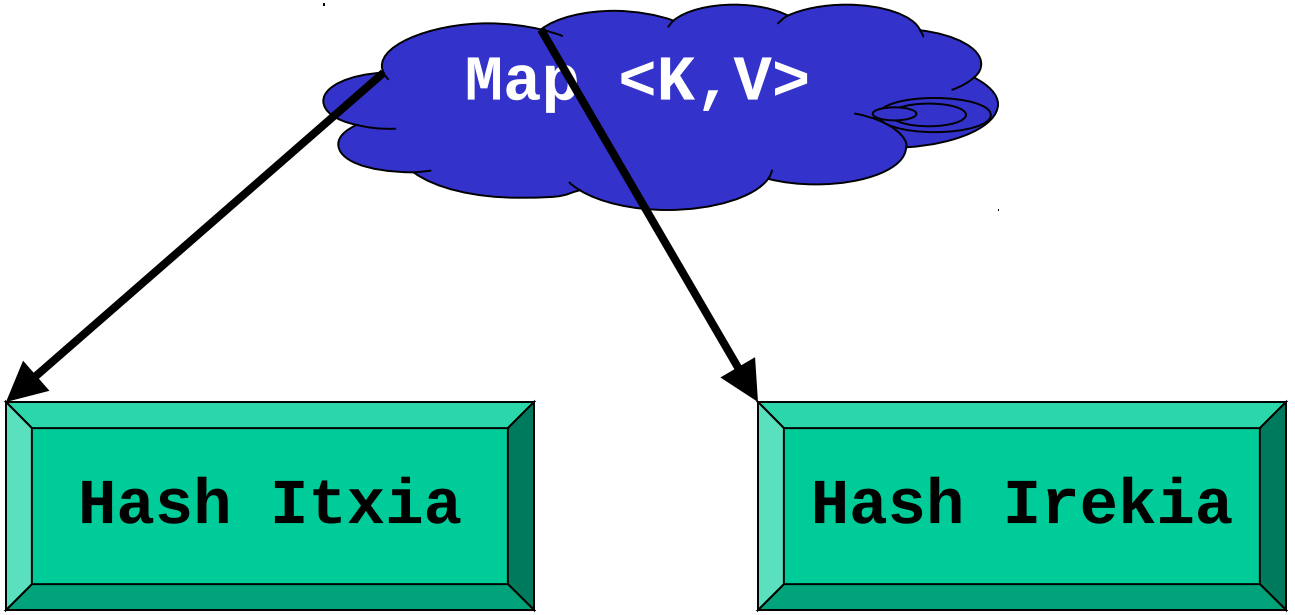
Informazio osagarria:
http://java.sun.com/j2se/1.5.0/docs/api/java/util/Map.html#get(java.lang.Object)

Hash taulak: interfazea

```
public interface Map <K,V> {
    public V put (K key, V value);
    public V get (K key);
    public V remove (K key);
    public void clear();
    public boolean isEmpty();
    public boolean containsKey(K key);
    public int size();
}
```

Hash taulak

Interfaze bat, inplementazio desberdinak



Hash taulak: Hash itxia

- Hash taula bat itxia da, onartzen dituen gehienezko elementu kopurua finkatuta dagoenean (MAXPOS)

- Eraginkortasuna bermatzeko, taula gorde nahi den elementu-kopurua baino %25 handiagoa izatea gomendatzen da (karga-faktorea < % 75 = 0.75)

Hash taulak

Hash itxia

- Hash funtzioaren emaitzarekin gakoari taulan dagokion posizioa lortzen da
  - indize=hash(gakoa) % MAXPOS
  - Baldin MAXPOS = 20
    - 15564465 gakoaren posizioa zein litzateke?
    - “JAVA” gakoaren posizioa zein litzateke?

Hash taulak
Hash itxia

- Zer gertatuko litzateke posizio hori beste gako **sinonimo** batek beteko balu (kolisioa edo talka)? ➔
  - ***Lehenbailehen libre dagoen posizio bat aurkitu taulan!!***

- ***Esaterako: "CC" gakoa "ADA" gakoaren sinonimoa da ikusitako hash funtzioarekin***

hash("CC") --> 3+3 = 6
hash("ADA") --> 1 + 4 + 1 = 6

Hash taula itxiak:
Talkak: Nola aurkitu posizio berria?

- Proba lineala: ondoren dagoen lehen posizio librea bilatu
  - ***while (gako posizioa erabilita) do***

    gakoa = (1 + gakoa) % MAXPOS

  Desabantaila: clusterrak sor daitezke

- Zenbaki lehenaren proba: MAXPOS zatitzen EZ duen zenbaki **lehen** bat aukeratu (MAXPOS = 15 → P = 13)

  gakoa = (P + gakoa) % MAXPOS

- *Elementu baten ezabaketak ez du esan nahi guztiz desagertu behar dela (nonItem)*

HASH TAULA

0	
1	(katua,cat)
2	
3	
4	(txakurra,dog)
	(etxea,house)
	(kaixo,hello)
MAXPOS-1	

Hash taula itxiak: Talkak: Nola aurkitu posizio berria?

- *Proba koadratikoa: x-en ondoren libre dagoen posizioa sekuentzia honekin bilatzen du:*

  $x+1, x+2^2, x+3^2, x+4^2, x+5^2,\ldots$

  – *Clusterrak ekiditu nahi ditu*

  – *Dena dela, posizio batera doazen gakoek bide bera jarraitzen dute*

Hash taula itxiak: Talkak: Nola aurkitu posizio berria?

- *Hashing bikoitza: x-en ondoren libre dagoen lehen posizioa bilatuko du, gakoaren gaineko beste hash funtzio batekin kalkulatutako Pauso baten bidez:*

$$x+Pauso,\ x+2Pauso,\ x+3Pauso$$

- *Adibidez: $Pauso = P - (gako\ \%\ P)$, P zenbaki lehena eta arrayaren tamaina baino txikiagoa izanik*
- *Pauso ezin da inoiz 0 izan*
- *Arrayaren tamainak zenbaki lehena izan behar du*

Hash taula itxiak:

Proba linealaren inplementazioa: datuak eta taula

```
class DataItem<K,V> {
      K key;                  // data item (key)
      V data;
      public DataItem(K k, V v) { // eraikitzailea
        key = k;
        data = v; }
   }  // end class DataItem
```

```
class HashTableItxia<K,V> implements Map<K,V> {
  private  DataItem<K,V>[] hashArray;   // hash-taula egitura
  private int arraySize;
  private DataItem<K,V> nonItem;
```

Hash taula itxiak:
Eraikitzailea eta hash funtzioa

```
public HashTableItxia(int size)  {     // eraikitzailea
      arraySize = size;
      hashArray = new DataItem[arraySize];
      nonItem = new DataItem(null, null);   // elementu ezabatua
}
```

```
private int hashFunc(K  key) {
      return (key.hashCode() % arraySize );
     // Objektu bakoitzak kode desberdin bat
    // dauka, sistemak emandakoa. Erreferentzietatik lortzen da
    // Beste bat nahi izanez gero, birdefinitu behar dugu
 }
```

Hash taula itxiak:
containsKey metodoa. Proba lineala

```
public boolean containsKey (K pkey) { // find item with key

    int hashVal = hashFunc(pkey);  // hash the key

    while ( (hashArray[hashVal] != null)  &&
                    (!hashArray[hashVal].key.equals(pkey)) ) { // not found
        hashVal++;                  // go to next cell
        hashVal = hashVal % arraySize; // wraparound if necessary
    }

    if (hashArray[hashVal] == null)
        return false;   // ez dugu aurkitu
    else
        return true;
  }
```

Hash taula itxiak:
remove metodoa. Proba lineala

```
public V remove (K pKey) { // delete a DataItem
      int hashVal = hashFunc(pKey);  // hash the key

      while ( (hashArray[hashVal] != null)  &&
                              (!hashArray[hashVal].key.equals(pKey)) ) {
                  hashVal++;                  // proba lineala
                  hashVal = hashVal % arraySize;
      }

      if ((hashArray[hashVal] == null) )
                   return null;
      else {
         V temp = hashArray[hashVal].data; // save item
          hashArray[hashVal] = nonItem;       // delete item
          return temp; }
}
```

Hash taula itxiak:
put metodoa. Proba lineala

```
public V put (K pKey, V pData) { // insert a DataItem
                    // (assumes table not full)

    int hashVal = hashFunc(pKey);
    while( (hashArray[hashVal] != null)  &&
      (!hashArray[hashVal].equals(nonItem)) &&
      (!hashArray[hashVal].key.equals(pKey)) ) {    // proba lineala
        hashVal++;
        hashVal = hashVal % arraySize;
    }
    if (hashArray[hashVal] == null){
       // gakoa ez dago: txertatu
        hashArray[hashVal] = new DataItem(pKey, pData);
        return null;
     }
     else
```

Hash taula itxiak:
put metodoaren jarraipena

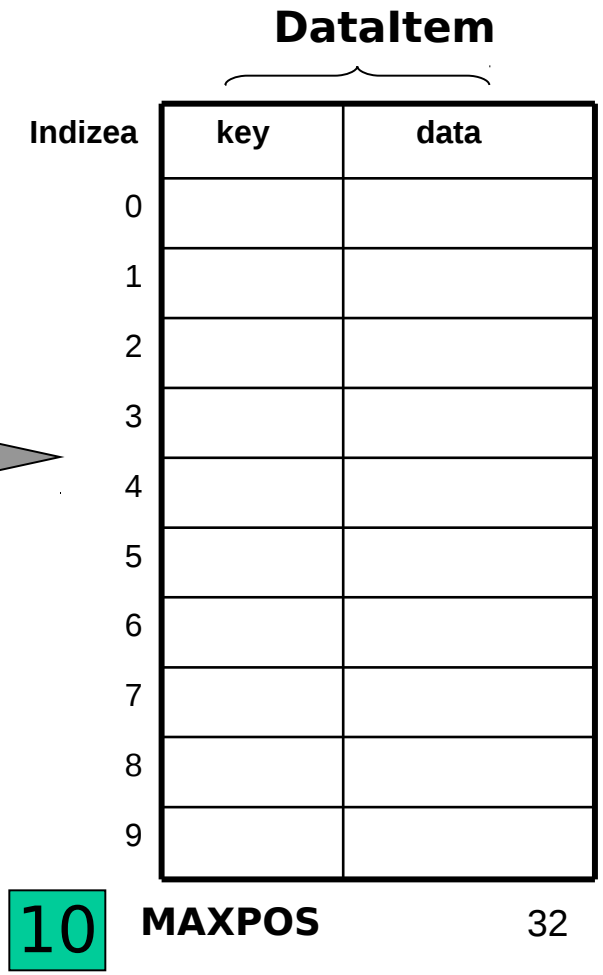
```
else if (hashArray[hashVal].equals(nonItem)) { // aurrerago begiratu
            int reserva = hashVal;
            hashVal = (hashVal + 1) % arraySize;
            while ( hashArray[hashVal] !=null &&
                    !hashArray[hashVal].key.equals(pKey) && hashVal != reserva ) {
                                hashVal++;
                                hashVal = hashVal % arraySize;
            }
            if ( (hashArray[hashVal] == null) || hashVal == reserva ) {
                                // gakoa ez dago: txertatu
                    hashArray[reserva] = new DataItem(pKey, pData);
                    return null;
            }
            else { // ( hashArray[hashVal].key.equals(pKey) )
                  //   aldatu elementu hau
                    V temp = hashArray[hashVal].data;
                    hashArray[hashVal].data=pData;
                    return temp;
            }
   }
    else // ( hashArray[hashVal].key.equals(pKey) )
```

Hash taula itxiak:
put metodoaren jarraipena

```
            else { // ( hashArray[hashVal].key.equals(pKey) )
              //   aldatu elementu hau
                        V temp = hashArray[hashVal].data;
                        hashArray[hashVal].data=pData;
                       return temp;
            }
} // end put()
```

Hash taula itxiaren adibidea

th = new HashTable<Integer, String>(10)



Hash taula itxiaren adibidea

Gakoek lau digitu osoak dituztela suposatuko dugu (dddd):
HASH (‘dddd’) → d+d+d+d
eta kolisioak proba linealaz ebazten direla.

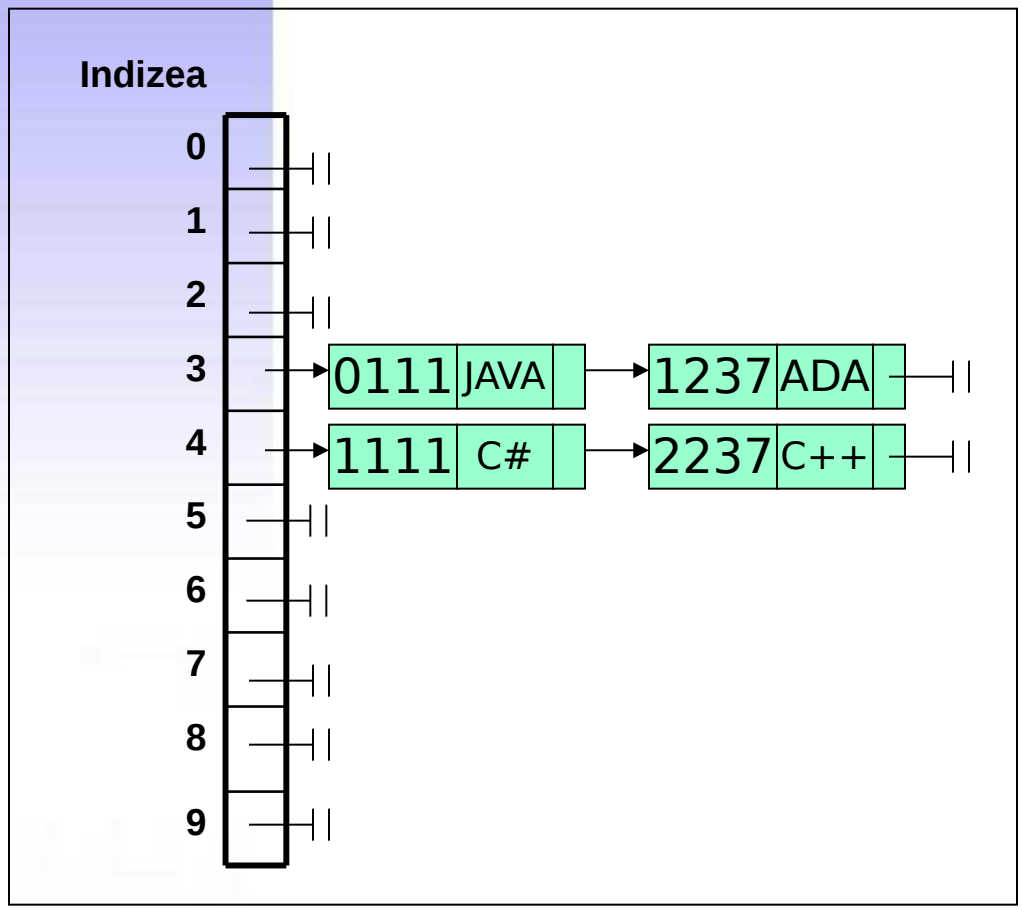
Ondorengo eragiketa egin ondoren hash taularen egoera zein den adierazi:

th.put (new Integer(1237), ”ADA”);
th.put (new Integer(2237), ”C”);
th.put (new Integer(0111), ”JAVA”));
th.put (new Integer(2237),”C++”);
th.remove (new Integer(0111));
th.put (new Integer(1111),”C#”));
th.remove (new Integer(1237));

Hash taulak:
Hash irekia

- **Ez dago** muga logikorik elementuak txertatzeko
- Hash funtzioa erabiliz gako bati dagokion posizioa lortzen da
- Posizio bereko sinonimoak zerrenda estekatu batean gordetzen dira
- Kolisioen kudeaketa ez da beharrezkoa

Hash taula irekiaren adibidea




```
Map th = new
        HashTaulaIrekia<Integer,
String>(10);
th.put(1237, ”ADA”);
th.put(2237, ”C”);
th.put(0111, ”JAVA”);
th.put(2237, ”C++”);
th.put(1111, ”C#”);
```

Hash taula irekiak:
Inplementazioa: datuak eta taula

```
class DataItem<K,V> {
      public K key;                  // data item (key)
      public V data;
      public DataItem(K pKey, V pData) { // eraikitzailea
        key=pKey;
        data=pData; }
   }  // end class DataItem
```

```
class HashTableIrekia<K,V> implements Map<K,V> {
   LinkedList<DataItem<K,V>>[] hashArray;        // array holds
                                                 //hash table
   int arraySize;
}
```

Hash taula irekiak:
Eraikitzailea eta hash funtzioa

```
public HashTableIrekia(int size)       // eraikitzailea
    {
    arraySize = size;
    hashArray = new LinkedList<K, V>[arraySize];
    for (int j = 0; j < arraySize; j++)
        hashArray[j] = new LinkedList<DataItem<K,V>>();
    }
```

```
public int hashFunc(K pKey)
    {
    return (k.hashCode() % arraysize); // hash function
    }
```

Hash taula irekiak:
put metodoa.

```
public V put (K pKey, V pData) { // insert a DataItem
        if (! containsKey(pKey)) {
            int hashVal = hashFunc(pKey);  // hash the key
            hashArray[hashVal].addFirst(new DataItem(pKey, pData));
            return null;}
        else {
            int hashVal = hashFunc(pKey);
            Iterator<DataItem<K, V> it  = hashArray[hashVal].iterator();
            boolean enc = 0;
            int index = 0;
            while (!(enc == 1)){ // ziur elementua badagoela!
                DataItem<K, V> elem = it.next();
                if elem.key.equals(pKey)) enc = 1;
                else index++;
            }
            V value;
            value = elem.data;
            hashArray[hashVal].remove(index);
            hashArray[hashVal].addFirst(new DataItem(pKey, pData));
            return value;
         }
        }  // end put()
```

Hash taula irekiak:
metodoak

```
public boolean containsKey (K pKey) {}   // find item with pKey
```

```
public V remove (K pKey) {} // delete item with pKey
```

```
public V get (K pKey) {} // return key associated value or null
```

Hash taulak:
Ez dira erabili behar

- Aldez aurretik elementuen kopurua oso aldakorra denean eta ezagutzen ez denean
- Elementuak ordena jakin batean atzitu behar direnean (adibidez, txikienetik handienera)

Hash taulak. Laburpena (1)

- Aplikazio askotan bilaketa eraginkorrak behar dira, gako baten bidez (NAN zenbakia, produktu baten kodea, etabar). Gako batek ez du onartzen balio errepikaturik
- Gako-kopurua >> elementu-kopurua
- Adibidez: gakoa: string(1 .. 20) => 26^{20} gako baina gehienez 10^5 elementu daudela dakigu!

Hash taulak. Laburpena (2)

- Gakoa ezin da erabili array baten indizetzat:
  - Elementu gehiegi (gehienak hutsik)
  - Gainera, gakoa ez bada numerikoa, lengoaia gehienek ez dute uzten gako hori arrayaren indizetzat erabiltzea. Adibidez: “jose luis”

Hash taulak. Laburpena (3)

(A) aukera: 10^5 elementuren arraya (indizea 1etik 100000raino)

- Arazoak:
  - Bilaketa
  - Txertaketa
  - Ezabaketa
  - Eragiketa sekuentzialak: O(N)
  - Gehienez bilaketa dikotomikoa (O(logN))
  - Baina txertaketa eta ezabaketa egitean elementuak mugitu behar dira => O(N)

Hash taulak. Laburpena (4)

(B) aukera: Hash taula (itxia)

- Bilaketa
- Txertaketa
- Ezabaketa

  Denbora ia konstantea ~ O(1) (bibliografia: analisi zehatza)
- Zeren arabera:
  - Hash funtzioa
  - Talkak
  - Taularen tamaina (karga-faktorea)

Hash taulak. Laburpena (5)

(C) aukera: Hash taula (irekia)

- k elementuren taula
- Azpilista baten batezbesteko elementuak: N / k

- Bilaketa
- Txertaketa
- Ezabaketa

  Denbora ia konstantea ~ O(N / k) (bibliografia: analisi zehatza)

Hash taulak. Laburpena (6)

(D) aukera. Zuhaitza DMA:

BZB (orekatua: AVL, ...)

- Denbora
  - Bilaketa
  - Txertaketa
  - Ezabaketa

Denak: ~ O(logN)

Datu-egituren eraginkortasuna

(denbora segundoetan)

Number of searches	BST (Lewis & Chase)	Balanced BST (Lewis & Chase)	TreeSet	HashSet	ArrayList
250	0.007	0.004	0.001	0	3.937
500	0.004	0.006	0.002	0.001	7.737
1000	0.008	0.006	0.004	0.001	14.95
2000	0.008	0.007	0.006	0.002	31.724
4000	0.016	0.015	0.012	0.005	60.775
8000	0.03	0.025	0.023	0.009	
16000	0.049	0.034	0.041	0.009	
32000	0.099	0.068	0.079	0.01	
64000	0.194	0.136	0.158	0.013	
128000	0.386	0.27	0.312	0.026	
256000	0.77	0.538	0.623	0.052	
512000	1.538	1.075	1.246	0.102	
1024000	3.072	2.147	2.501	0.202	
2048000	6.14	4.293	5.043	0.404	
4096000	12.27	8.594	9.938	0.809	
8192000	24.543	17.175	19.898	1.612	
16384000	49.137	34.351	39.771	3.229	

Datu-egituren eraginkortasuna

(denbora segundoetan)