ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Хеширање

Алгоритми и податочни структури

Аудиториска вежба 7

Хеширање

- Хеш табела со затворени кофички (Closed-bucket hash table - CBHT)
- Хеш табела со отворени кофички (Open-bucket hash table - OBHT)

- Двојно хеширање

Класата MapEntry

```
class MapEntry<K extends Comparable<K>,E> implements Comparable<K> {

        K key;
        E value;

        public MapEntry (K key, E val) {
                this.key = key;
                this.value = val;
        }

        public int compareTo (K that) {
                @SuppressWarnings("unchecked")
                MapEntry<K,E> other = (MapEntry<K,E>) that;
                return this.key.compareTo(other.key);
        }

        public String toString () {
                return "<" + key + "," + value + ">";
        }
}
```

Хеш табела со затворени кофички - CBHT

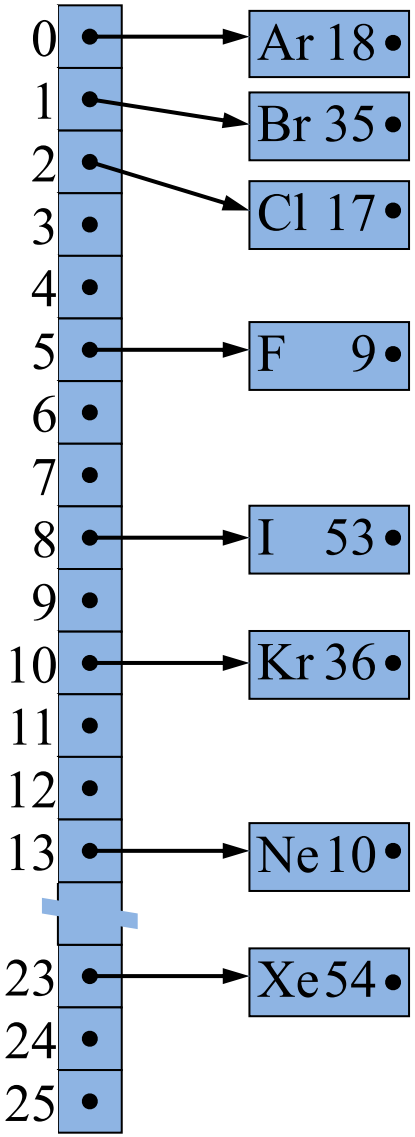
- Во секоја кофичка може да се сместат по повеќе елементи.
- Кофичките се потполно меѓусебно одвоени. Нема претекување, затоа се викаат затворени

- Наједноставна имплементација:
  - секоја кофичка е единечно поврзана листа
  - имаме низа од единечно поврзани листи

Хеш табела со затворени кофички - CBHT

- Илустрација (без колизии)

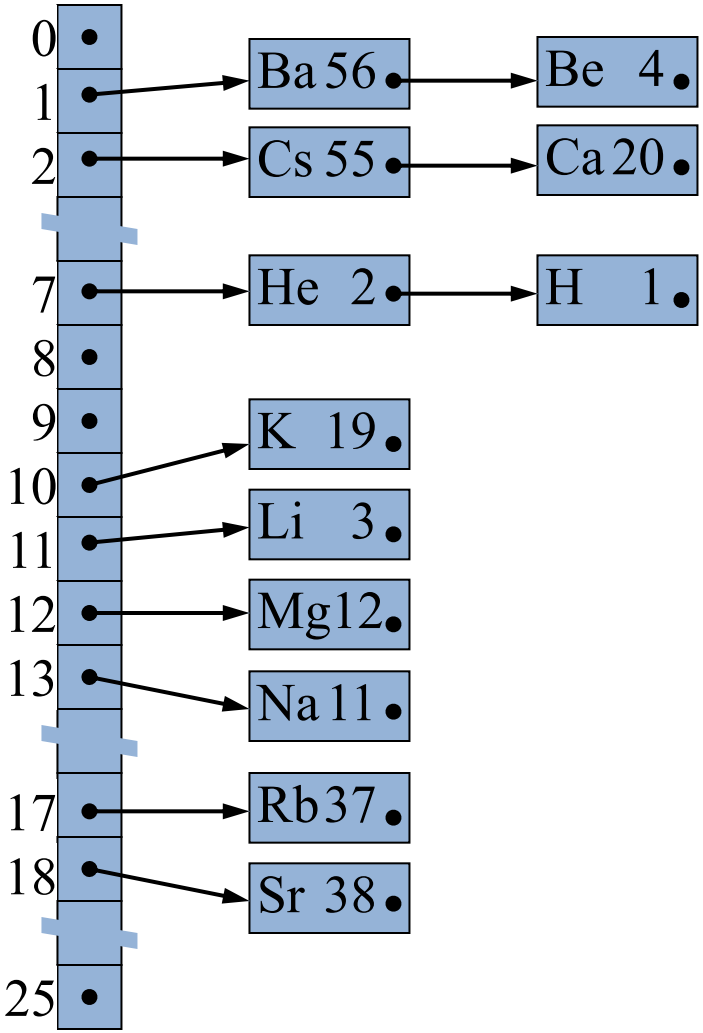
element	number
F	9
Ne	10
Cl	17
Ar	18
Br	35
Kr	36
I	53
Xe	54



Хеш табела со затворени кофички - СВНТ

- Илустрација (со колизии):

element	number
H	1
He	2
Li	3
Be	4
Na	11
Mg	12
K	19
Ca	20
Rb	37
Sr	38
Cs	55
Ba	56



Хеш табела со затворени кофички - класата CBHT

```
public class CBHT<K extends Comparable<K>, E> {

        //Табелата со кофички се состои од низа од јазли (SLLNode
        јазли) кои во себе чуваат MapEntry објекти.
        private SLLNode<MapEntry<K,E>>[] buckets;

        @SuppressWarnings("unchecked")
        public CBHT(int m) {

                //Креира празна хеш табела со m „кофички”.
                buckets = (SLLNode<MapEntry<K,E>>[]) new SLLNode[m];
        }

    //методи на CBHT
    //...
}
```

Хеш табела со затворени кофички – методи

```
private int hash(K key) {
        // Го преведува клучот во индекс на низата bucket
        return Math.abs(key.hashCode()) % buckets.length;
}

public SLLNode<MapEntry<K,E>> search(K targetKey) {
        //Го наоѓа јазолот од СВНТ кој содржи елемент чиј клуч е еднаков
        на targetKey. Враќа врска до тој јазол (или null ако нема таков
        јазол)
        int b = hash(targetKey);
        for (SLLNode<MapEntry<K,E>> curr = buckets[b]; curr != null; curr
        = curr.succ) {
                if (targetKey.equals(((MapEntry<K, E>)
        curr.element).key))
                        return curr;
        }
        return null;
}
```

Хеш табела со затворени кофички –методи

```
public void insert(K key, E val) {

        //Вметнување на парот <key, val> во СВНТ.
        MapEntry<K, E> newEntry = new MapEntry<K, E>(key, val);
        int b = hash(key);

        //Класата SLLNode од едностранo поврзани листи.
        for (SLLNode<MapEntry<K,E>> curr = buckets[b]; curr != null;
        curr = curr.succ) {

                if (key.equals(((MapEntry<K, E>) curr.element).key)) {
                        //newEntry го заменува постоечкиот елемент со
                        клуч key.
                        curr.element = newEntry;
                        return;
                }
        }
        //Додавање на newEntry на почетокот на листата во домашната
        „кофичка” со индекс b.
        buckets[b] = new SLLNode<MapEntry<K,E>>(newEntry, buckets[b]);
}
```

Хеш табела со затворени кофички – методи

```
public void delete(K key) {
        int b = hash(key);
        for (SLLNode<MapEntry<K,E>> pred = null, curr = buckets[b];
        curr != null; pred = curr, curr = curr.succ) {
                if (key.equals(((MapEntry<K,E>) curr.element).key)) {
                        if (pred == null)
                                buckets[b] = curr.succ;
                        else
                                pred.succ = curr.succ;
                        return;
                }
        }
}
```

Хеш табела со затворени кофички – методи

```
public String toString() {
        String temp = "";
        for (int i = 0; i < buckets.length; i++) {
                temp += i + ":";
                for (SLLNode<MapEntry<K,E>> curr = buckets[i]; curr !=
                null; curr = curr.succ) {
                        temp += curr.element.toString() + " ";
                }
                temp += "\n";
        }
        return temp;
}
```

Задача 1. Роденденди

Во Заводот на статистика се прави ново истражување каде што се открива бројот на луѓе родени во секој месец. Ваша задача е за даден месец да прикажете колку луѓе се родени во тој месец.

Влез: Во првиот ред од влезот е даден бројот на луѓе *N*, а во секој нареден ред е даден датумот на раѓање. Во последниот ред е даден месецот за кој треба да се прикаже бројот на луѓе родени во тој месец.

Излез: Број на луѓе кои се родени во тој месец. Доколку нема луѓе родени во тој месец да се испечати „Empty”.

Задача 1. Роденденi

Пример

Влез:

4

20.7.1976

16.7.1988

18.7.1966

5.6.1988

7

Излез:

3

Решение

```
public class Birthdays {

        public static void main(String[] args) throws NumberFormatException, IOException {

                BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
                int N = Integer.parseInt(bf.readLine());
                CBHT<String, Integer> birthdays = new  CBHT<>(23);

                for(int i = 0; i<N; i++){

                        String p [] = bf.readLine().split("\\.");

                        //доколку елементот со клуч p[1] не постои во хеш табелата
                        if(birthdays.search(p[1])==null)
                        {
                                birthdays.insert(p[1], 1);
                        }
                        else
                        {
                                //доколку елементот со клуч p[1] постои во хеш табелата
                                SLLNode<MapEntry<String, Integer>> br = birthdays.search(p[1]);
                                birthdays.insert(p[1], br.element.value+1);
                        }

                }

                //...
```

Решение

```
                //...

                String month = bf.readLine();
                SLLNode<MapEntry<String, Integer>> result =
                birthdays.search(month);

                //доколку не постои елемент со клуч mesec
                if(result==null)
                {
                        System.out.println("Empty");
                }
                else
                {
                        System.out.println(result.element.value);
                }
        }
}
```

Задача 2. Најдобра понуда

На еден светски познат предавач секојдневно му пристигнуваат понуди да држи предавања. За секоја понуда се дадени датуми, време на почеток, градот и износот на хонорарот за предавањето (во долари). Ваша задача е за даден датум да го прикажете предавањето кое би му донело најголема заработка на предавачот. Доколку нема понуди за дадениот датум да се испечати „No offers”.

Влез: Во првиот ред од влезот е даден бројот на понуди, а во секој нареден ред се дадени: датумот и времето на предавањето (формат dd/mm/yyyyhh:mm), градот во кој ќе се одржува предавањето и износот на хонорарот. Во последниот ред е даден датумот за кој треба да испечатите која понуда е најдобра за тој датум.

Излез: Деталите на понудата за тој датум.

Задача 2. Најдобра понуда

Пример

Влез:
7
27/01/2016 14:00 NewYork 6000
28/01/2016 08:00 Paris 3000
28/01/2016 14:00 Munich 5000
27/01/2016 09:00 Beijing 8000
27/01/2016 08:00 Seattle 4000
28/01/2016 09:00 SaltLakeCity 10000
28/01/2016 09:00 Lagos 12000
27/01/2016

Излез:
09:00 Beijing 8000

Решение – класата Lecture

```
class Lecture implements Comparable<Lecture>
{
        String date;
        String time;
        String place;
        Integer fee;

        public Lecture(String date, String time, String place, Integer fee)

        {
                this.date = date;
                this.time = time;
                this.place = place;
                this.fee = fee;
        }

        public String getTime() {
                return time;
        }
        public void setTime(String time) {
                this.time = time;
        }
        public String getDate() {
                return date;
        }
```

Решение – класата Lecture

```
public void setDate(String date) {
        this.date = date;
}
public String getPlace() {
        return place;
}
public void setPlace(String place) {
        this.place = place;
}
public Integer getFee() {
        return fee;
}
public void setFee(Integer fee) {
        this.fee = fee;
}
@Override
public int compareTo(Lecture obj) {
        if(this.fee > obj.fee)
                return 1;
        else if(this.fee < obj.fee)
                return -1;
        else
                return 0;
}
```

}

Решение

```
public class BestOffer{
public static void main(String[] args) throws IOException {

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int N = Integer.parseInt(br.readLine());

        CBHT<String, ArrayList<Lecture>> hashtable = new
        CBHT<String,ArrayList<Lecture>>(2*N);

    for(int i=0;i<N;i++)
    {
        String[] input=br.readLine().split(" ");
        Lecture p = new
        Lecture(input[0],input[1],input[2],Integer.parseInt(input[3]));

        if(hashtable.search(input[0])==null)
        {
                ArrayList<Lecture> lectures = new ArrayList<Lecture>();
                lectures.add(p);
                hashtable.insert(input[0], lectures);
        }
        else
        {
                SLLNode<MapEntry<String, ArrayList<Lecture>>> result =
                hashtable.search(input[0]);
                ArrayList<Lecture> lectures = result.element.value;
                lectures.add(p);
                Collections.sort(lectures,Collections.reverseOrder());
                hashtable.insert(input[0], lectures);
        }
    }
```

Решение

```
            String date=br.readLine();

            //Сложеноста на оваа операција е O(1) бидејќи низата е секогаш сортирана според хонорарот во опаѓачки редослед
            SLLNode<MapEntry<String, ArrayList<Lecture>>> tosearch = hashtable.search(date);

            if(tosearch!=null) {
                System.out.println(tosearch.element.value.get(0).getTime()+"
                "+tosearch.element.value.get(0).getPlace()+"
                "+tosearch.element.value.get(0).getFee());
            }
            else
            {
                System.out.println("No offers");
            }

        }
}
        String date=br.readLine();

            //Сложеноста на оваа операција е O(1) бидејќи низата е секогаш сортирана според хонорарот во опаѓачки редослед
            SLLNode<MapEntry<String, ArrayList<Lecture>>> tosearch = hashtable.search(date);

            if(tosearch!=null) {
                System.out.println(tosearch.element.value.get(0).getTime()+"
                "+tosearch.element.value.get(0).getPlace()+"
                "+tosearch.element.value.get(0).getFee());
            }
            else
            {
                System.out.println("No offers");
            }

        }
}
```

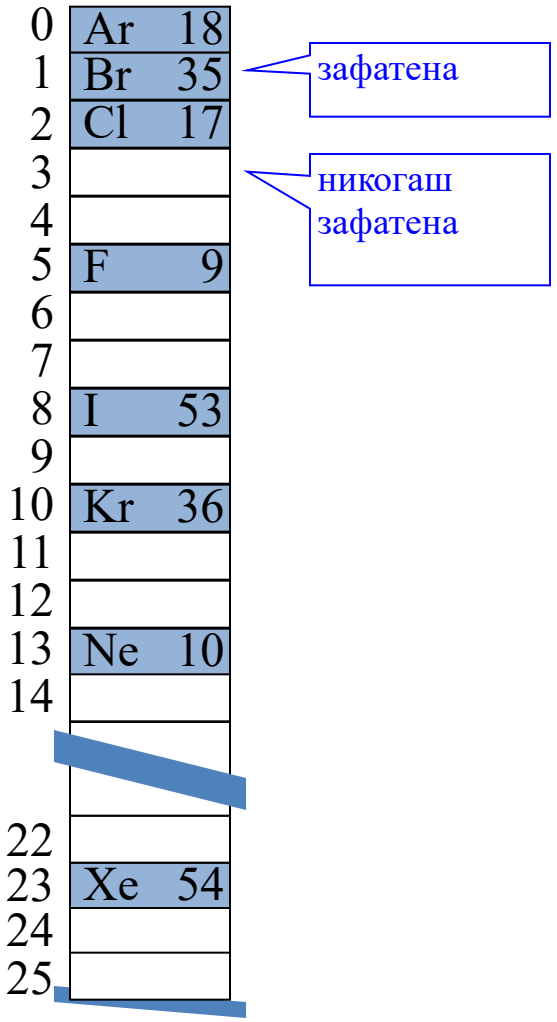
Хеш табела со отворени кофички - ОВНТ

- Секоја кофичка може да содржи најмногу еден елемент.
- Ако се појави колизија, новиот елемент се преместува во друга кофичка.
- Секоја кофичка може да се најде во една од трите состојби:
  - никогаш-зафатена (никогаш не содржела елемент)
  - зафатена (моментално има елемент)
  - претходно-зафатена (содржела елемент, кој е избришан и во моментов нема нов).

Хеш табела со отворени кофички (OBHT)

- Илустрација (без колизии):

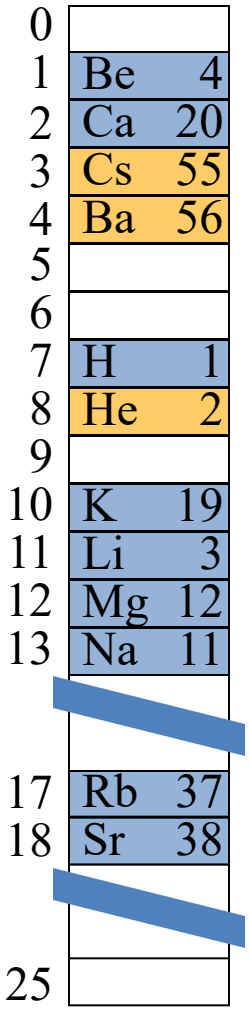
element	number
F	9
Ne	10
Cl	17
Ar	18
Br	35
Kr	36
I	53
Xe	54



Хеш табела со отворени кофички (OBHT)

- Илустрација (со колизии):

element	number
H	1
He	2
Li	3
Be	4
Na	11
Mg	12
K	19
Ca	20
Rb	37
Sr	38
Cs	55
Ba	56



Хеш табела со отворени кофички (OBHT)

```
public class OBHT<K extends Comparable<K>,E> {

    //Табелата се состои од MapEntry објекти.
    private MapEntry<K,E>[] buckets;

    //buckets[b] е null ако „кофичката” b не била никогаш зафатена.
    //buckets[b] е претходно зафатена ако во „кофичката” b имало претходно елемент кој
e избришан и моментално нема елемент во оваа „кофичка”.

    static final int NONE = -1; //различно од било кој индекс на „кофичка”.

    private static final MapEntry former = new MapEntry(null, null);
    //Ова гарантира дека за било кој елемент е e.key.equals(former.key) е false.

    private int occupancy = 0;
    //број на зафатени или претходно зафатени „кофички”.

    @SuppressWarnings("unchecked")
          public OBHT (int m) {
    //Се креира празна OBHT со m „кофички”.
        buckets = (MapEntry<K,E>[]) new MapEntry[m];
    }

    //методи на OBHT
    //...
}
```

Хеш табела со отворени кофички (OBHT) - методи

```
private int hash(K key) {
        //Го преведува клучот во индекс на низата buckets.
        return Math.abs(key.hashCode()) % buckets.length;
}

public int search (K targetKey) {
    int b = hash(targetKey);
    int n_search=0;
    for (;;) {
        MapEntry<K,E> oldEntry = buckets[b];
        if (oldEntry == null)
            return NONE;
        else if (targetKey.equals(oldEntry.key))
            return b;
        else
        {
            b = (b + 1) % buckets.length;
            n_search++;
                if(n_search==buckets.length)
                return NONE;

        }
    }
}
```

Хеш табела со отворени кофички (OBHT) - методи

```
public MapEntry<K,E> getBucket(int i){
          return buckets[i];
}

public void insert (K key, E val) {
          MapEntry<K,E> newEntry = new MapEntry<K,E>(key, val);
          int b = hash(key);
          int n_search=0;
          for (;;) {
                    MapEntry<K,E> oldEntry = buckets[b];
                    if (oldEntry == null) {
                              if (++occupancy == buckets.length) {
                                        System.out.println("Hash table is full!!!");
                              }
                              buckets[b] = newEntry;
                              return;
                    }
                    else if (oldEntry == former || key.equals(oldEntry.key)) {
                              buckets[b] = newEntry;
                              return;
                    }
                    else
                    {
                              b = (b + 1) % buckets.length;
                              n_search++;
                              if(n_search==buckets.length)
                                        return;

                    }
          }
}
```

Хеш табела со отворени кофички (OBHT) - методи

```
@SuppressWarnings("unchecked")
public void delete (K key) {
        int b = hash(key); int n_search=0;
        for (;;) {
                MapEntry<K,E> oldEntry = buckets[b];

                if (oldEntry == null)
                        return;
                else if (key.equals(oldEntry.key)) {
                        buckets[b] = former; //MapEntry<K,E>)former;
                        return;
                }
                else{
                        b = (b + 1) % buckets.length;
                        n_search++;
                        if(n_search==buckets.length)
                                return;

                }
        }
}
```

Задача 3. Црвен крст

Во рамки на една хуманитарна организација, потребно е да се направи статистика за крвните групи кои се на располагање за донација, и од кои донатори. Подгрупите A1+, A2+ припаѓаат на крвна група A+, додека A1- , A2- припаѓаат на група A-.

Влез: Во првиот ред од влезот е даден бројот на парови *N*, а во секој наредeн ред се дадени паровите (донатор, крвна група).

Излез: Да се испечати по колку донатори има од секоја крвна група согласно внесените податоци.

Задача 3. Црвен крст

Пример

Влез:
5
Alek A1+
Dejan B$-$
Sandra A+
Trajce 0+
Rebeka A1$-$

Излез:
A+=2
B$-$=1
0+=1
A$-$=1

Решение

```
public class RedCross {

        public static void main (String[] args) throws IOException {

                OBHT<String, Integer> hashtable = new OBHT<String,Integer>(11);
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                int N = Integer.parseInt(br.readLine());
                String input;

                for(int i=1;i<=N;i++){
                        input = br.readLine();
                        String[] row = input.split(" ");
                        String key = row[1];
                        key = key.replaceAll("[1-2]","");
                        if(hashtable.search(key)==-1)
                        {
                                hashtable.insert(key, 1);
                        }
                        else
                        {
                                hashtable.insert(key,
                                hashtable.getBucket(hashtable.search(key)).value+1);
                        }
                }
                System.out.println(hashtable);
        }
}
```

Вградени структури за хеш во Java

- Класата `HashMap<K,V>`
- Класата `Hashtable<K,V>`

Класата `Hashtable` во Java

- Класата `Hashtable<K,V>` претставува готова класа од Java за хеш табела со затворени кофички
- Тип на параметри: K – тип на клуч, V – тип на вредност
- Има два параметри: `int initialCapacity` и `float loadFactor`
- Конструкторот HashMap() креира празна мапа со иницијален капацитет 11 и фактор на пополнетост (load factor) 0.75
- Ако се зголеми бројот на елементи и се надмине факторот на пополнетост – тогаш табелата се зголемува и се ре-хашува
- Опис на методите:
  - `containsKey(Object key)` и `containsValue(Object value)`
    - враќа: true ако мапата содржи мапирање за дадениот клуч (вредност)
  - `get(Object key)`
    - враќа: објектот (од тип V) кој е внесен за дадениот клуч, во спротивно враќа null
  - `put(K key, V value)`
    - Ја поставува дадена вредност во табела за дадениот клуч
  - `remove(Object key)`
    - Го отстранува елементот што е внесен за дадениот клуч (ако постои) и го враќа

Класата `Hashtable` во Java - пример

```
import java.util.Hashtable;

public class TestHashtable {
        public static void main(String[] args) {
                Hashtable<Character, Integer> m =
                new Hashtable<Character, Integer>();
                String s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                for (int i = 0; i < s.length(); i++) {
                        m.put(s.charAt(i), i); //се полни хеш табелата
                }

                System.out.println(m);
                //ја земаме вредноста за буквата Z
                System.out.println("The value of letter Z is " + m.get('Z'));

        }
}
```

Класата `HashMap` во Java

- Класата `HashMap<K,V>` претставува еквивалент на `HashTable<K,V>`, со таа разлика што дозволува null за клуч и вредност
- Конструкторот HashMap() креира празна мапа со иницијален капацитет 16 и фактор на пополнетост (load factor) 0.75
- Опис на методите:
  - `containsKey(Object key)` и `containsValue(Object value)`
    - враќа: true ако мапата содржи мапирање за дадениот клуч (вредност)
  - `get(Object key)`
    - враќа: објектот (од тип V) кој е внесен за дадениот клуч, во спротивно враќа null
  - `put(K key, V value)`
    - Ја поставува дадена вредност во табела за дадениот клуч
  - `remove(Object key)`
    - Го отстранува елементот што е внесен за дадениот клуч (ако постои) и го враќа

Класата HashMap во Java - пример

```
import java.util.HashMap;
import java.util.Map;

public class TestHashMap {
        public static void main(String[] args) {
                Map<Character, Integer> m = new HashMap<Character, Integer>();
                String s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                for (int i = 0; i < s.length(); i++) {
                        m.put(s.charAt(i), i); //се полни хеш табелата
                }

                System.out.println(m);
                //ја земаме вредноста за буквата Z
                System.out.println("The value of letter Z is " + m.get('Z'));

        }
}
```

Вградени структури за хеш во Java

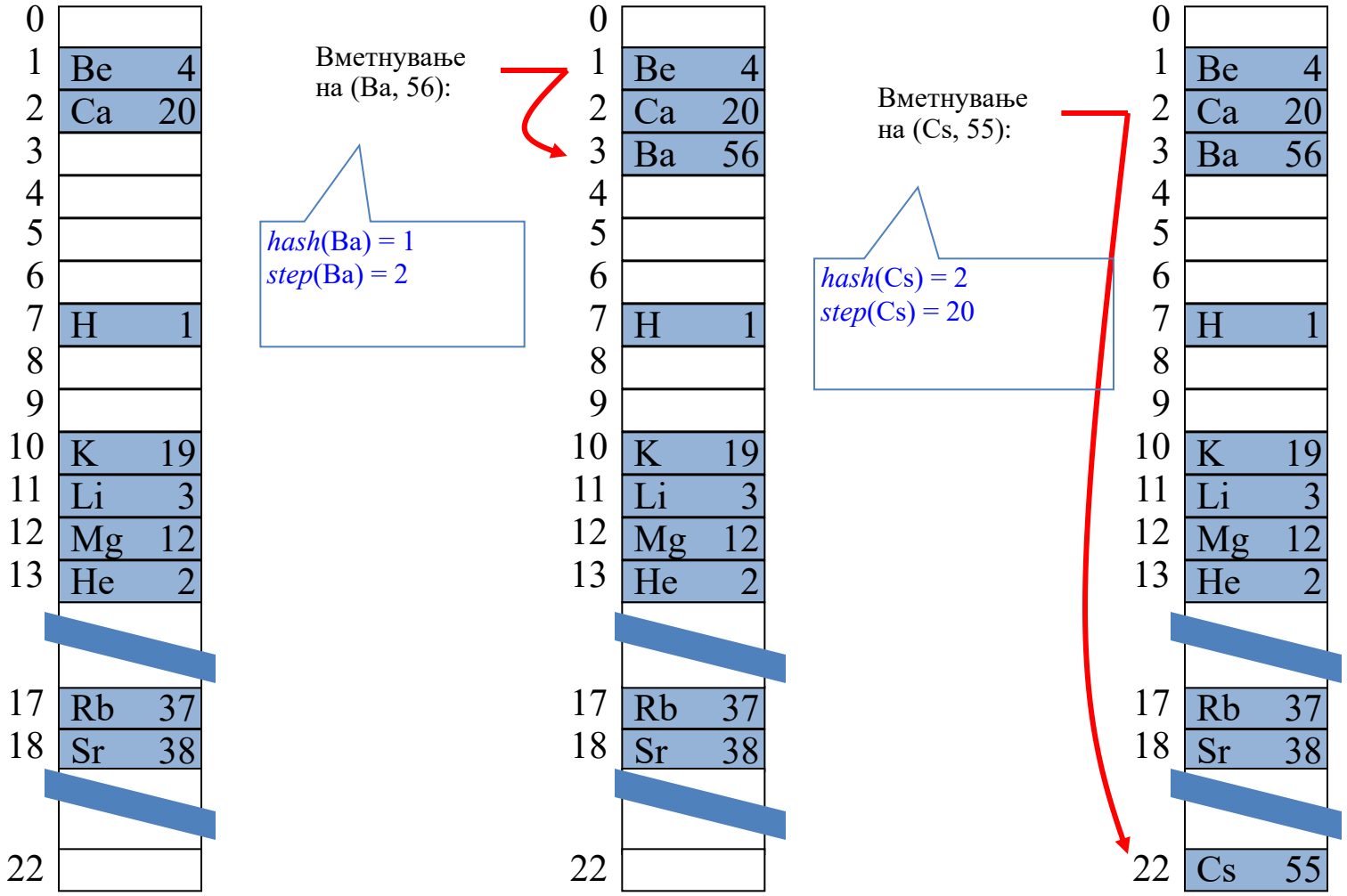
- Ако сакате сопствена класа да ја користите како клуч во Hashtable или HashMap тогаш вашата класа мора да ги имплементира следните методи:
  - hashCode - `int hashCode()`
  - equals - `boolean equals(Object obj)`

Двојно хеширање

- За вметнување/пребарување/бришење на елемент со клуч *k*, се пресметува *s* од *k*, користејќи втора хеш функција *s* = *step*(*k*).
- Функцијата *step(k)* во ОВНТ се користи на следниот начин:
  - При вметнување доколку веќе е вметнат друг елемент на позицијата *hash(k)* (односно веќе имаме ист хеш код за друг објект во табелата) тогаш елементот го вметнуваме на позиција *hash(k)+step*(*k*) (по модуло m)

Двојно хеширање - илустрација

- Илустрација:



Двојно хеширање - Java

```
public interface DoublyHashable<K> extends Comparable<K> {

    public int hashCode ();
    //Враќа хеш код за клучот.

    public int stepCode ();
    //Враќа должина на чекор за клучот.

}
```

Двојно хеширање - Java

```
public class DoublyHashedOBHT <K extends DoublyHashable<K>,E> {

    private MapEntry<K,E>[] buckets;

    static final int NONE = -1;

    private static final MapEntry former = new MapEntry(null, null);

    private int occupancy = 0;

    @SuppressWarnings("unchecked")
        public DoublyHashedOBHT (int m) {
         buckets = (MapEntry<K,E>[]) new MapEntry[m];
    }

    // ...
}
```

Двојно хеширање - Java

```
public void insert (K key, E val) {
        MapEntry<K,E> newEntry = new MapEntry<K,E>(key, val);
        int b = hash(key);
        int s = step(key);
        int n_search = 0;
        for (;;) {
                MapEntry<K,E> oldEntry = buckets[b];
                if (oldEntry == null) {
                        if (++occupancy == buckets.length) {
                                System.out.println("Hash table is full!!!");
                        }
                        buckets[b] = newEntry;
                        return;
                } else if (oldEntry == former || key.equals(oldEntry.key)) {
                        buckets[b] = newEntry;
                        return;
                }
                else
                {
                        b = (b + (s) % buckets.length;
                        n_search++;
                        if(n_search == buckets.length)
                                return;
                }
        }
}
```

Двојно хеширање - Java

```
class ChemicalElementDH implements DoublyHashable<ChemicalElementDH> {

    private char sym1, sym2;

    public ChemicalElementDH (String symbol) {
        if (symbol.length() >= 1)
            sym1 = Character.toUpperCase(symbol.charAt(0));
        else
            sym1 = ' '; // Should really fail.
        if (symbol.length() >= 2)
            sym2 = Character.toLowerCase(symbol.charAt(1));
        else
            sym2 = ' ';
    }

    public int hashCode () {
        return sym1 - 'A';
    }

    public int stepCode () {
         return (sym2 == ' ') ? 1 : sym2 - 'a' + 2;
    }
}
```

Двојно хеширање - Java

Пример за користење на структурата за илустрацијата од слајд 39

```
public static void main (String[] args) {
        DoublyHashedOBHT<ChemicalElementDH,Integer> table1 =
                            new
DoublyHashedOBHT<ChemicalElementDH,Integer>(23);
        table1.insert(new ChemicalElementDH("H"),  new Integer(1));
        table1.insert(new ChemicalElementDH("He"), new Integer(2));
        table1.insert(new ChemicalElementDH("Li"), new Integer(3));
        table1.insert(new ChemicalElementDH("Be"), new Integer(4));
        table1.insert(new ChemicalElementDH("Na"), new Integer(11));
        table1.insert(new ChemicalElementDH("Mg"), new Integer(12));
        table1.insert(new ChemicalElementDH("K"),  new Integer(19));
        table1.insert(new ChemicalElementDH("Ca"), new Integer(20));
        table1.insert(new ChemicalElementDH("Rb"), new Integer(37));
        table1.insert(new ChemicalElementDH("Sr"), new Integer(38));
        table1.insert(new ChemicalElementDH("Cs"), new Integer(55));
        table1.insert(new ChemicalElementDH("Ba"), new Integer(56));

        System.out.println(table1);
}
```

```
0:
1:<Be,4>
2:<Ca,20>
3:<Ba,56>
4:
5:
6:
7:<H ,1>
8:
9:
10:<K ,19>
11:<Li,3>
12:<Mg,12>
13:<He,2>
14:
15:<Na,11>
16:
17:<Rb,37>
18:<Sr,38>
19:
20:
21:
22:<Cs,55>
```