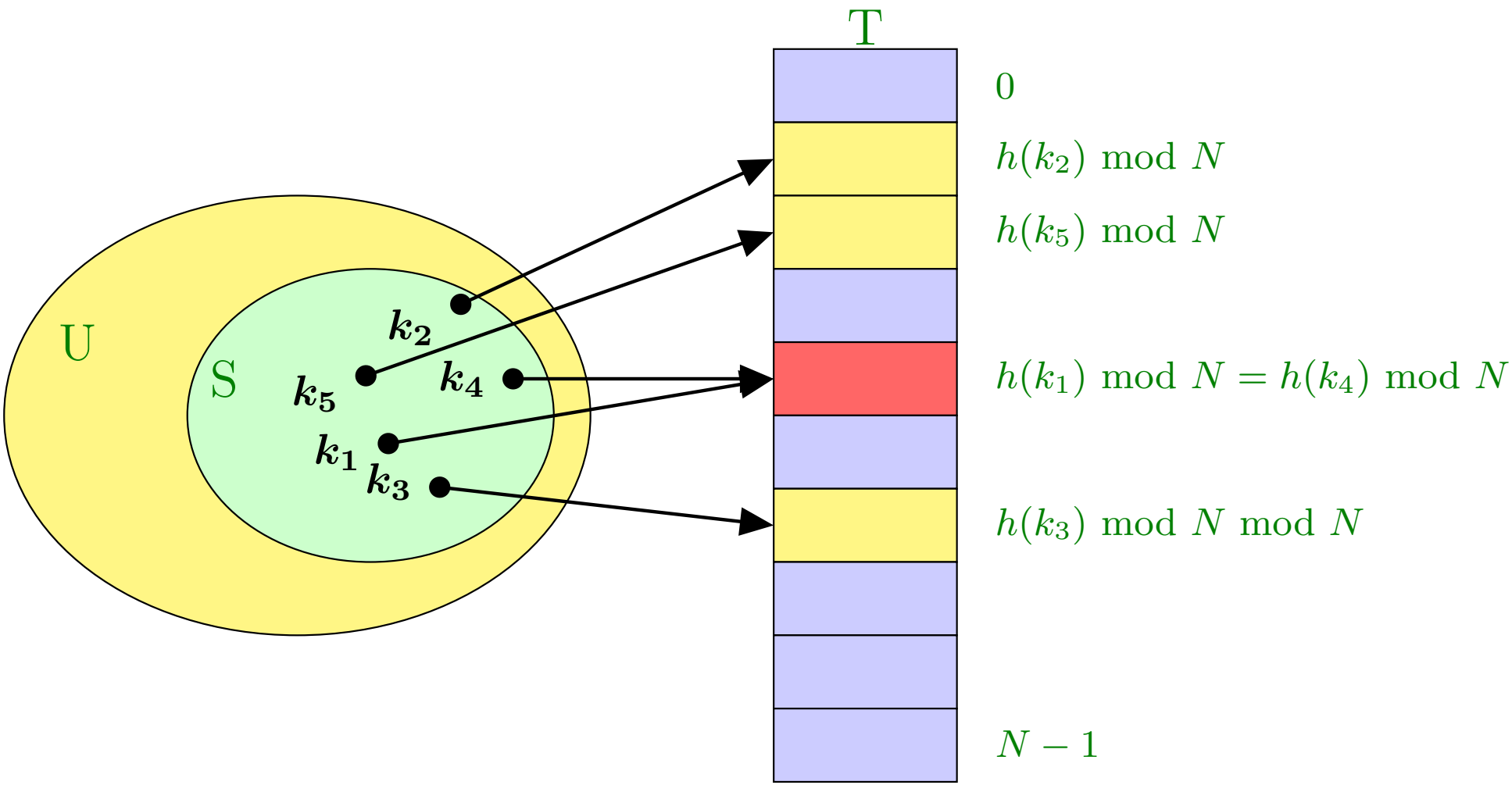
Algoritmusok és adatszerkezetek gyakorlat - 07 Hasítótáblák

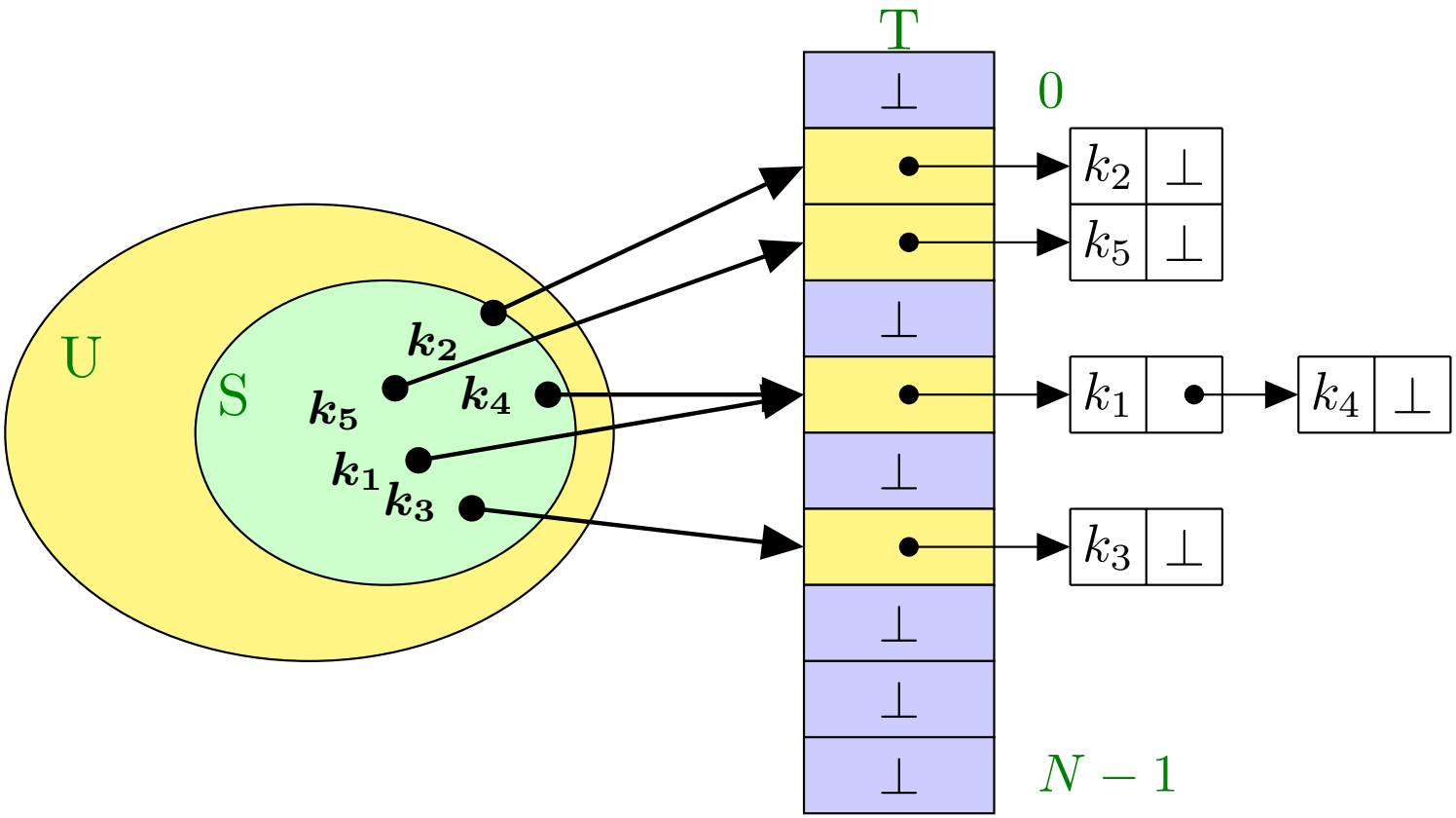
Gelle Kitti

2017. 10. 25.

Hasítótáblák



Ütközésfeloldás - Chaining



Load factor: vödrök száma / elemek száma

Ha ez túl nagy ⇒ több vödör és hash-eljük újra az összes eddigi elemet

⇒ Kétszer (vagy konstansszor) annyi vödröt éri meg létrehozni

Érdekesség: amikor a több vödör nem segít

Ha ugyanaz a hash code

```
println( "AaBBAa".hashCode() ) //1952508096
println( "BBAaAa".hashCode() ) //1952508096
println( "BBBBAa".hashCode() ) //1952508096
println( "AaAaAa".hashCode() ) //1952508096
println( "AaBBBB".hashCode() ) //1952508096
println( "BBAaBB".hashCode() ) //1952508096
println( "BBBBBB".hashCode() ) //1952508096
println( "AaAaBB".hashCode() ) //1952508096
```

Java-ban a `String.hashCode()`:

$$s[0] \cdot 31^{n-1} + s[1] \cdot 31^{n-2} + ... + s[n-1]$$

$$\texttt{"Aa".hashCode()} == 31 * 65 + 97 == 2112$$
$$== 31 * 66 + 66 == \texttt{"BB".hashCode()}$$

Open adressing

- Minden vödörbe max egy kulcsot teszünk
- A hash kód alapján számolt index csak kiindulópont
- Ha foglalt, akkor valamilyen módszerrel végigpróbálunk másokat.
  Jelölés: `probe(h(k),i)`: a `h(k)` kulcs i. próbája
  - **linear probing**: `probe( h(k), i ) = (h(k) + i) mod N`
  - **quadratic probing**: `probe( h(k), i )` = $\texttt{h(k)} + c_1 \cdot \texttt{i} + c_2 \cdot \texttt{i}^2$ `mod N`
  - **double hashing**: `probe( h(k), i )` = $h_1(\texttt{k}) + \texttt{i} \cdot h_2(\texttt{k})$ `mod N`
- Ha a load factor kb. 0.7-re növekszik, mindegyik módszer meglassul

Feladat

Szúrjuk be egy 10 hosszú hash táblába a következő objektumokat chaining és linear probing módszerekkel . Az elemek `h1` hash függvény által adott hash kódjai a következők:

- `h1("cím") = 210`
- `h1("foo") = 130`
- `h1("bar") = 65`
- `h1("fák") = 188`
- `h1("bin") = 785`
- `h1("one") = 960`

Megvalósításuk

Halmaz

- Egy elem csak egyszer kerülhet be
- Csak kulcsot tárol
- HashSet: hasítótáblával megvalósítva
- TreeSet: keresőfával megvalósítva

Map / Dictionary / Associative array

- Kulcs-érték párok tárolása
- HashMap: hasítótáblával megvalósítva
- TreeMap: keresőfával megvalósítva

Feladat

Szavak gyakoriságának leszámolása

Az input egy hosszú szöveget tartalmazó String. A feladat az, hogy adjuk meg a benne található összes szó előfordulásainak számát. Írjunk rá pszeudokódot!

Feladat - ugyanez javaban

```
public static Map<String, Integer>
     wordFrequencies(String[] words) {

  Map<String, Integer> result = new HashMap<>();
  for (String word : words) {
    int n = 1;

    if (result.containsKey(word)) {
      n = result.get(word) + 1;
    }

    result.put(word, n);
  }

  return result;
}
```

Keresőfa vs Hasítótábla

Keresőfa

- Keresés: $O(\log n)$
- Beszúrás: $O(\log n)$
- Rendezés: van (A kulcsok egy teljesen rendezett halmazból KELL jöjjenek)
- Megelőző/rákövetkező elem keresése: gyors

Hasítótábla

- Keresés: $O(1)$
- Beszúrás: $O(1)$
- Rendezés: nincs
- Megelőző/rákövetkező elem keresése: nem lehetséges

HashSet Feladat

Adott a következő 2D-s kordinátákat megvalósító osztály, melyeket egy HashSet-be szeretnénk gyűjteni.

```
public class Coord {
  int x;
  int y;

  public Coord(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

HashSet Feladat

```
public static void main(String[] args) {
  Set<Coord> s = new HashSet<Coord>();

  Coord p = new Coord(100, 100);
  s.add(p);
  System.out.println(s.contains(p));

  Coord q = new Coord(100,100);
  System.out.println(s.contains(q));

  Coord r = p;
  System.out.println(s.contains(r));
}
```

HashSet Feladat

```
public static void main(String[] args) {
  Set<Coord> s = new HashSet<Coord>();

  Coord p = new Coord(100, 100);
  s.add(p);
  System.out.println(s.contains(p)); //true

  Coord q = new Coord(100,100);
  System.out.println(s.contains(q)); //false

  Coord r = p;
  System.out.println(s.contains(r)); //true
}
```

HashSet Feladat

```
public class Coord {
...

  public int hashCode() {
    return x ^ y;
  }

  public boolean equals(Object o) {
    if (o == null) return false;
    if (!(o instanceof Coord)) return false;
    Coord oc = (Coord)o;
    return x == oc.x && y == oc.y;
  }
}
```

HashSet Feladat

```
public static void main(String[] args) {
  Set<Coord> s = new HashSet<Coord>();

  Coord p = new Coord(100, 100);
  s.add(p);
  System.out.println(s.contains(p)); //true

  Coord q = new Coord(100,100);
  System.out.println(s.contains(q)); //true :)

  Coord r = p;
  System.out.println(s.contains(r)); //true
}
```

HashSet Feladat

```
public static void main(String[] args) {
  Set<Coord> s = new HashSet<Coord>();

  Coord p = new Coord(100, 100);
  s.add(p);
  System.out.println(s.contains(p));

  p.x = 50;
  System.out.println(s.contains(p));

  Coord q = new Coord(50,100);
  System.out.println(s.contains(q));

  Coord r = new Coord(100,100);
  System.out.println(s.contains(r));
}
```

HashSet Feladat

```
public static void main(String[] args) {
  Set<Coord> s = new HashSet<Coord>();

  Coord p = new Coord(100, 100);
  s.add(p);
  System.out.println(s.contains(p)); //true

  p.x = 50;
  System.out.println(s.contains(p)); //false

  Coord q = new Coord(50,100);
  System.out.println(s.contains(q)); //false

  Coord r = new Coord(100,100);
  System.out.println(s.contains(r)); //false wtf
}
```

HashSet Feladat - tanulságok

- A `hashCode` és az `equals` metódusokat mindig párban kell felülírni
- A `hashCode` soha ne változhasson az objektum életciklusa alatt
- Ha két objektum lehet `equals` bármikor élettartamuk során, akkor a `hashCode`-juk ugyanaz kell legyen

Tehát `hashCode`ot csak immutable mezőkből (olyan adattagok, amiken nem tudunk változtatni) szabad számítani, különben ez (↑) lesz

TreeSet Feladat

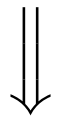
```
public static void main(String[] args) {
  Set<Coord> s = new TreeSet<Coord>();

  Coord p = new Coord(100, 100);
  s.add(p);
}
```

TreeSet Feladat

```
public static void main(String[] args) {
  Set<Coord> s = new TreeSet<Coord>();

  Coord p = new Coord(100, 100);
  s.add(p);
}
```



Exception in thread ”main” java.lang.ClassCastException: Coord cannot be cast to java.lang.Comparable

TreeSet Feladat

```
public class Coord implements Comparable<Coord> {
...

  // Sorfolytonos osszehasonlitas
  public int compareTo(Coord o) {
  int comp_y = Integer.compare(y, o.y);
  if (comp_y == 0) {
    return Integer.compare(x, o.x);
  }
  return comp_y;
  }
}
```

Tanulság: a TreeSet kulcsai mindig egy rendezett halmazból kell jöjjenek. Az osztály elemeit összehasonlító metódust meg kell írjuk!