Adatszerkezetek és algoritmusok

Horváth Géza

ötödik előadás

Előadások témái

1. Az algoritmusokkal kapcsolatos alapfogalmak bevezetése egyszerű példákon keresztül.
2. Az algoritmusok futási idejének aszimptotikus korlátai.
3. Az adatszerkezetekkel kapcsolatos alapfogalmak. A halmaz, a multihalmaz és a tömb adatszerkezet bemutatása.
4. Az adatszerkezetek folytonos és szétszórt reprezentációja. A verem, a sor és a lista.
5. Táblázatok, önátrendező táblázatok, hash függvények és hash táblák, ütközéskezelés.
6. Fák, bináris fák, bináris keresőfák, bejárás, keresés, beszúrás, törlés.
7. Kiegyensúlyozott bináris keresőfák: AVL fák.
8. Piros-fekete fák.
9. B-fák.
10. Gráfok, bejárás, legrövidebb út megkeresése.
11. Párhuzamos algoritmusok.
12. Eldönthetőség és bonyolultság, a P és az NP problémaosztályok.
13. Lineáris idejű rendezés. Összefoglalás.

A táblázat, mint absztrakt adattípus

A táblázat olyan összetett adatelemeket tartalmaz, melyek mindegyike alapvetően két részből áll:

- egy **kulcsból**, ami minden esetben egyedi, és
- egy **értékből,** mely maga is lehet összetett, és nem szükséges egyedinek lennie.

Az elemeket azonosítása a kulcs alapján történik.

A táblázat, mint adatszerkezet

A táblázat tulajdonságai:

- homogén
- dinamikus
- asszociatív
- néha folytonos reprezentációval tároljuk, míg máskor szétszórt (láncolt) reprezentációval

A táblázat típusai

Sokféle táblázatot használunk, ezek közül a legismertebbek:

- soros táblázat
- önátrendező táblázat
- kulcstranszformációs táblázat (magyarul: hash tábla)

A táblázat műveletei

- Insert – elem beszúrása a táblázatba.
- Delete – elem eltávolítása a táblázatból.
- Search – elem megkeresése a táblázatban a kulcs alapján.

A táblázat, mint adatszerkezet

Műveletek:

- adatszerkezetek **létrehozása**: folytonos vagy láncolt reprezentációval
- adatszerkezetek **módosítása**
  - elem hozáadása: INSERT
  - elem törlése: DELETE
  - elem cseréje: nem használjuk
- elem **elérése**: SEARCH

A táblázat lehetséges felépítése

	rendezett	rendezetlen
folytonos	van	van
láncolt	van	van

Önátrendező táblázat

- Az önátrendező táblázat használata abban az esetben hatékony, ha az adatelemek **feldolgozási gyakorisága** nagymértékben eltérő.
- Az önátrendező táblázat használata egyetlen művelet alkalmazásában tér el a soros táblázattól: a SEARCH művelet használatakor a megtalált elemet mindig előre hozzuk, a táblázat legelső pozíciójába. Ennek eredményeképpen a leggyakrabban használt elemek kerülnek a táblázat elejére.
- Az adatok tárolása javasolt **láncolt reprezentáció** és **lineáris keresés** használatával, mivel ebben az esetben az elemek első pozícióba mozgatása könnyen megvalósítható, és a lineáris keresés a legtöbb esetben gyorsan eredményre vezet, mivel a gyakran használt elemek a táblázat elején lesznek.
- Az önátrendező táblázat **gyorsan reagál** a feldolgozási gyakoriságban bekövetkező változásokra.

Önátrendező táblázat – példa

Kari büfé reggel:

200	coffee
300	sandwich
350	cola
150	tea
1.120	cigarette
400	cake
5.200	whisky
290	beer
4.800	steak
14.500.000	BMW

Önátrendező táblázat – példa

Kari büfé délben:

4.800	steak
200	coffee
350	cola
400	cake
290	beer
300	sandwich
150	tea
1.120	cigarette
5.200	whisky
14.500.000	BMW

Önátrendező táblázat – példa

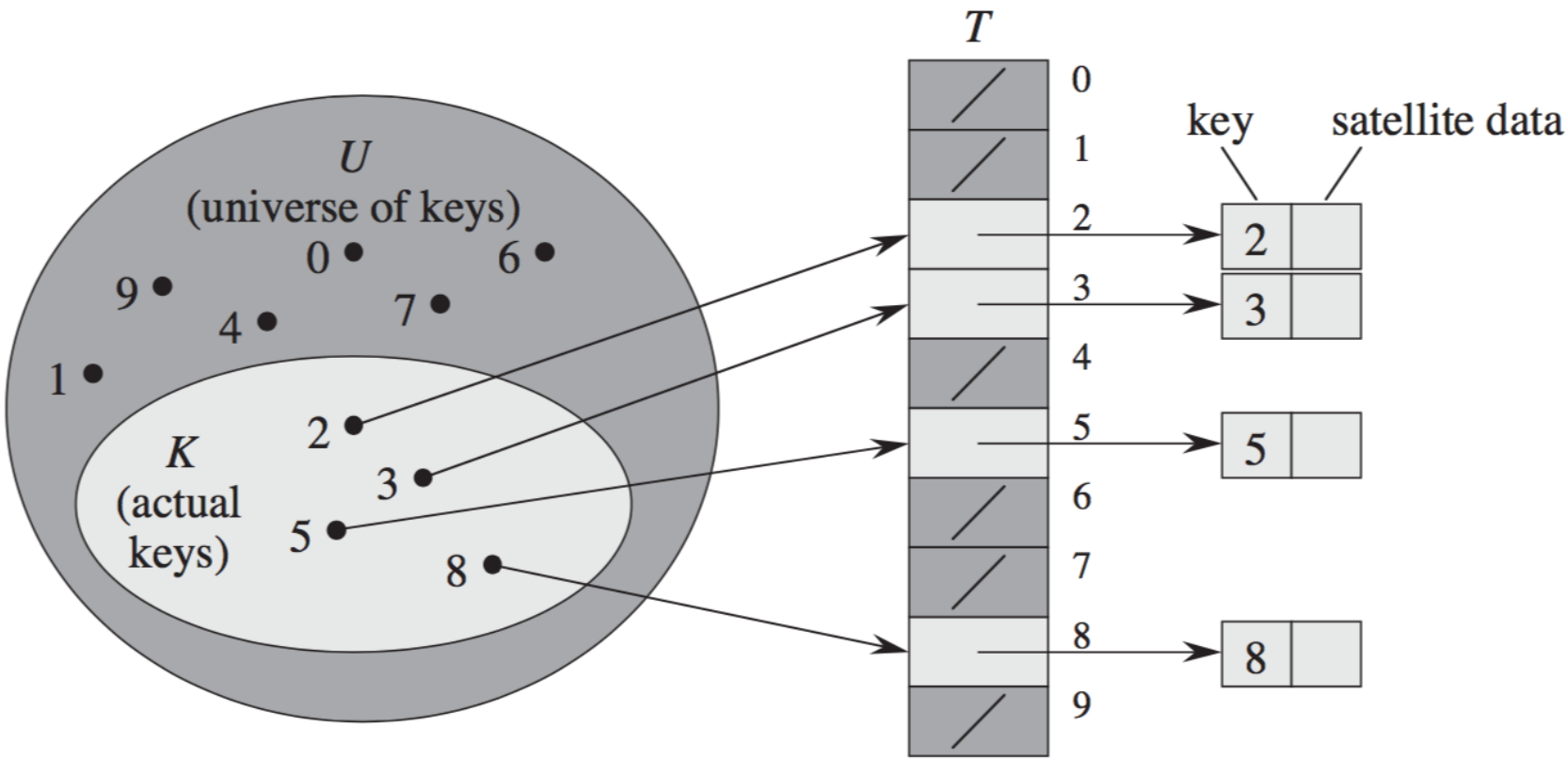
Kari büfé este:

290	beer
1.120	cigarette
5.200	whisky
4.800	steak
350	cola
300	sandwich
400	cake
150	tea
200	coffee
14.500.000	BMW

Közvetlen címzésű táblázat

- A közvetlen címzés módszere abban az esetben hatékony, ha az U kulcstér viszonylag kicsi.
- Tegyük fel, hogy a táblázat olyan kulcsokat használ, melyek az U=$\{0,1,\ldots,m\text{-}1\}$ elemei, és az m nem túl nagy szám.
- Ekkor a táblázatunkat tárolhatjuk egy olyan közvetlen címzésű $T[0 \ldots m-1]$ táblázatban, melyben minden elem pozíciója megegyezik az adott elem kulcsával.

A közvetlen címzésű táblázat reprezentációja



A közvetlen címzésű táblázat reprezentációja

DIRECT-ADDRESS-SEARCH(T, k)

1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)

1 $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)

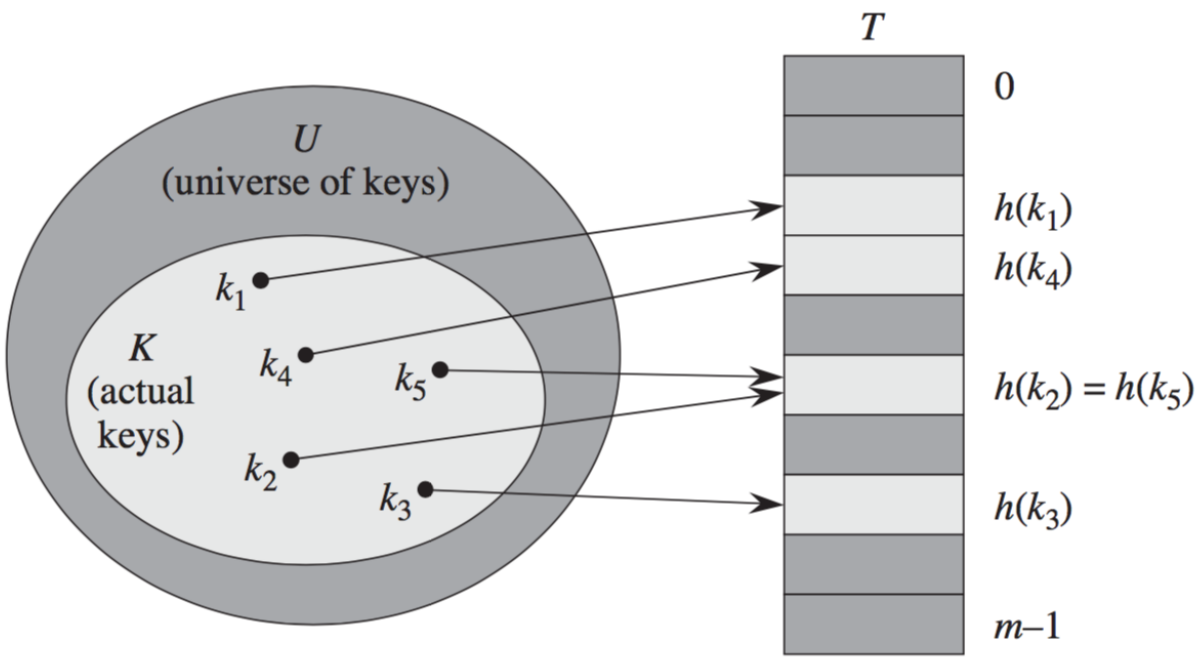
1 $T[x.key] =$ NIL

Hash tábla

- A közvetlen címzésű táblázat használatának megvannak a korlátai. Amenyiben az U kulcstér nagy, azaz a kulcsok nagy intervallumból kerülnek kiválasztásra, akkor a közvetlen címzés használata nem hatékony, hiszen ebben az esetben a T táblázat is nagy lesz, mivel a T elemeinek száma $|U|$. Ez különösen akkor szembetűnő, ha a táblázat elemeinek száma kicsi. Ebben az esetben hash függvény használata javasolt.
- Közvetlen címzés használatakor a k kulcsú elem pozíciója a T táblázatban: T[k]. Hash függvény használatakor a h(k) hash függvényt használjuk a k kulcsú elem T táblázatban elfoglalt pozíciójának kiszámítására: T[h(k)].

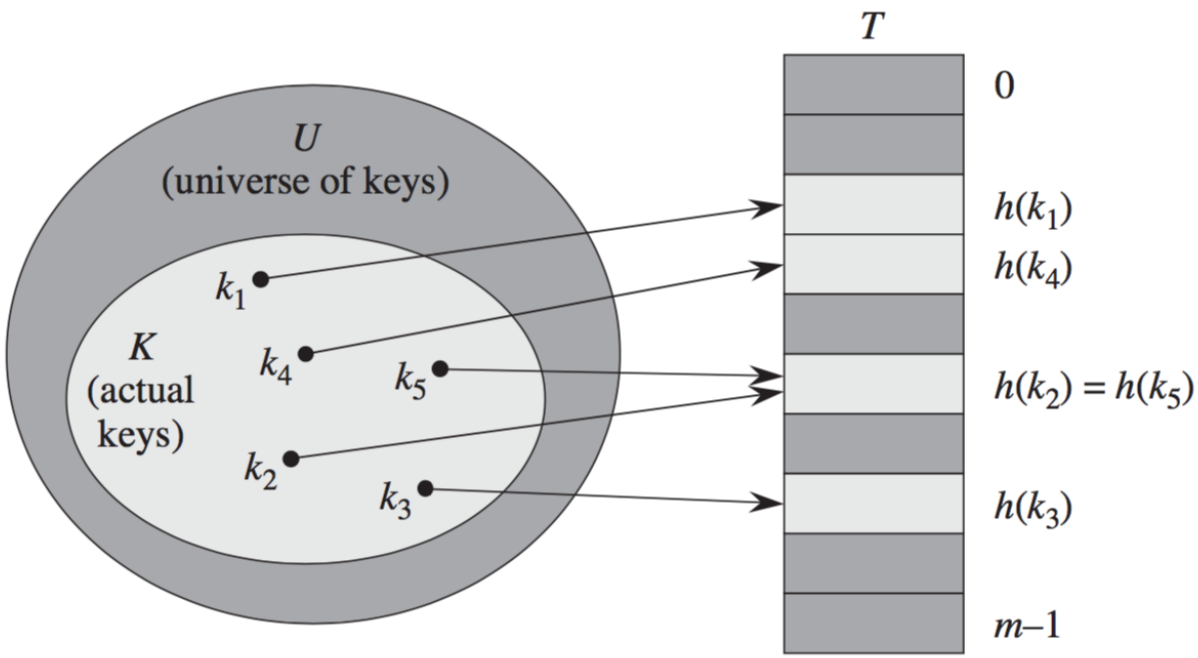
Hash tábla

- A h hash függvény az U kulcstér elemeihez rendeli egy adott T[0 . . . $m-1$] hash tábla pozícióit.
- h: U $\rightarrow$ {0,1,. . . ,m-1}, ahol m tipikusan jóval kisebb, mint |U|.



Hash tábla – ütközés

- Houston, we have a problem: több különböző kulcshoz is tartozhat ugyanaz a pozíció. Ezt az esetet hívjuk **ütközésnek.**
- Az alábbi példában a k_2 és a k_5 kulcsok ugyanazzal a **hash értékkel** rendelkeznek.
- Szerencsére hatékony megoldásaink vannak az ütközés esetére.

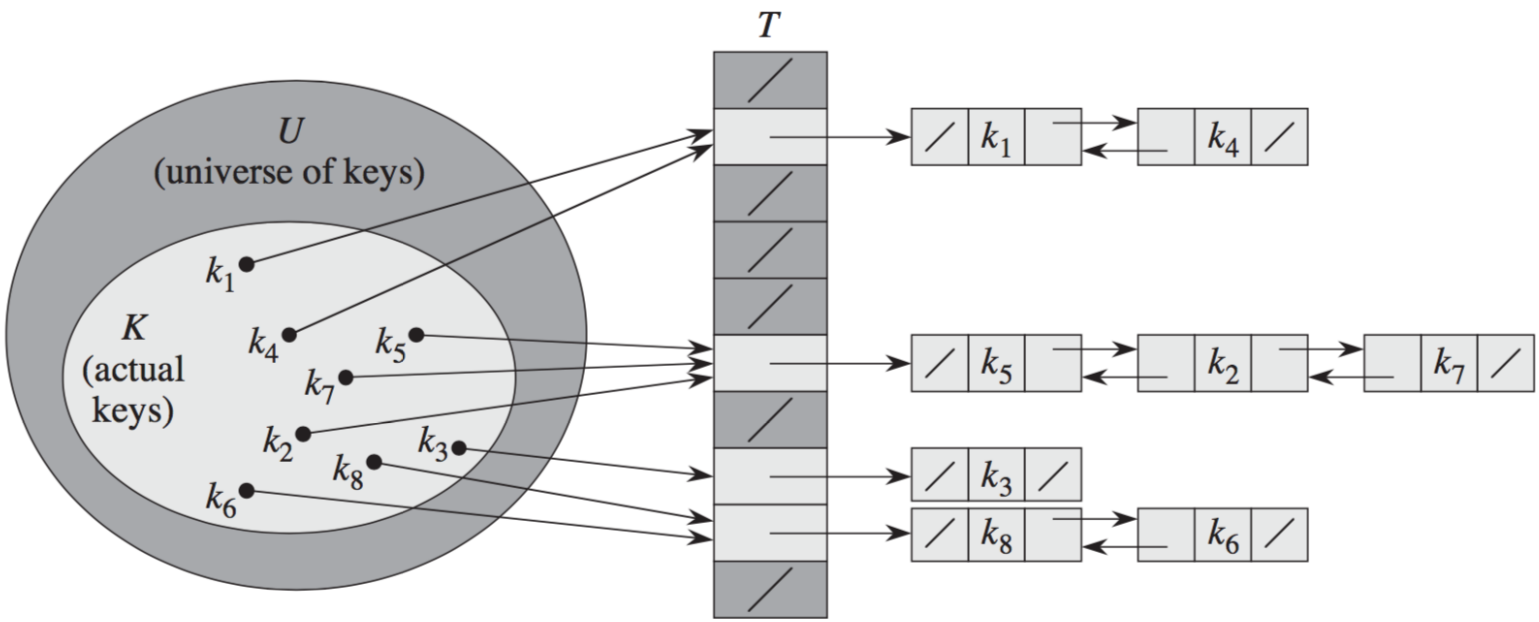


Hash függvény – az osztásos módszer

- Az **osztásos módszer** alkalmazásakor a h hash függvény a k kulcshoz az m különböző pozíció egyikét rendeli hozzá, maradékos osztás segítségével.
- A hash függvény alakja h(k) = k mod m.
- Például, ha a hash tábla mérete m=12, akkor a lehetséges hash értékek a 0 és 11 közötti egész számok.
- Ha a hash tábla mérete m=12 és a kulcs k=100, akkor a hash érték h(k)=4.

Ütközésfeloldás láncolással

- Láncolás esetén az azonos hash értékekkel rendelkező elemeket egy-egy láncolt listában tároljuk.
- A hash tábla j-edik pozíciójában egy mutatót találunk, mely azon láncolt lista első elemére mutat, amelyik mindazon elemeket tartalmazza, mely elemek hash értéke j. Ha nincs ilyen, akkor az értéke NIL.



Ütközésfeloldás láncolással

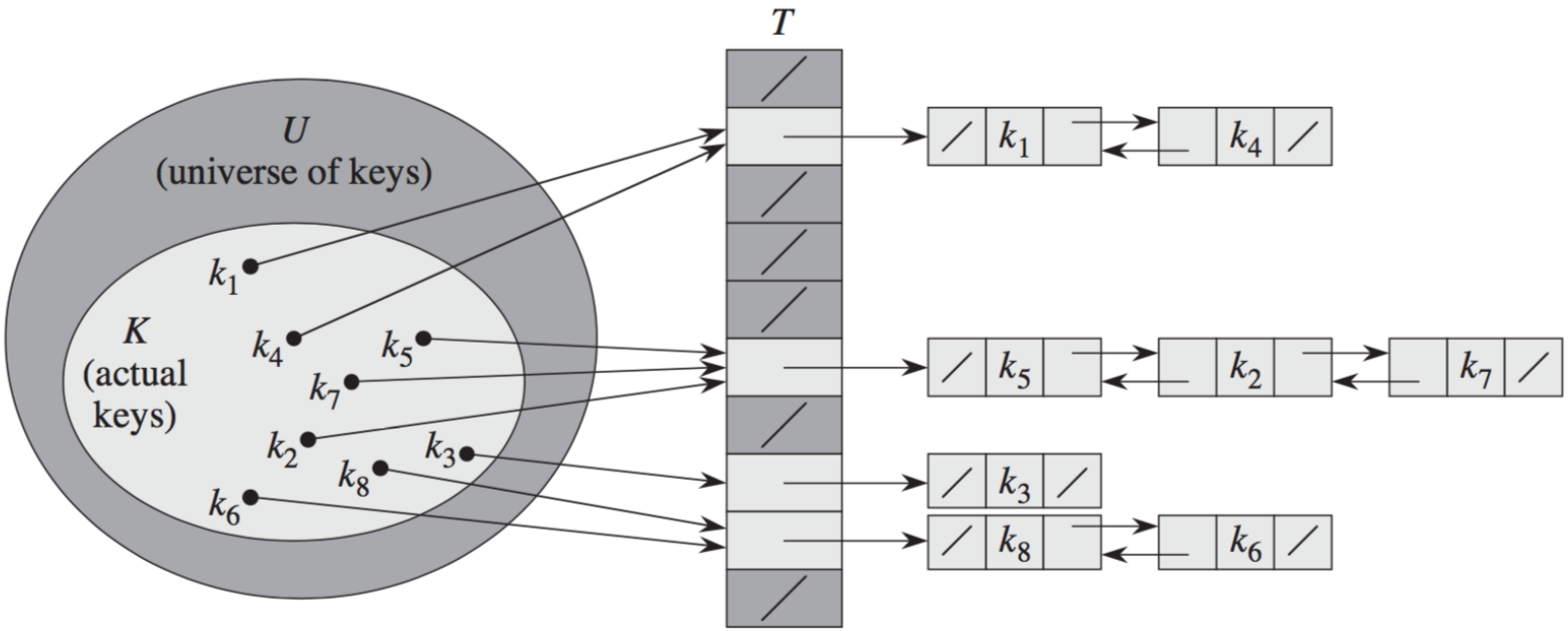


Figure 11.3 Collision resolution by chaining. Each hash-table slot $T[j]$ contains a linked list of all the keys whose hash value is j. For example, $h(k_1) = h(k_4)$ and $h(k_5) = h(k_7) = h(k_2)$. The linked list can be either singly or doubly linked; we show it as doubly linked because deletion is faster that way.

Ütközésfeloldás láncolással – műveletek

CHAINED-HASH-INSERT(T, x)

1 insert x at the head of list $T[h(x.key)]$

CHAINED-HASH-SEARCH(T, k)

1 search for an element with key k in list $T[h(k)]$

CHAINED-HASH-DELETE(T, x)

1 delete x from the list $T[h(x.key)]$

Ütközésfeloldás nyílt címzés módszerével

- A nyílt címzés módszerének használatakor minden elem magában a táblázatban tárolódik, ezért a táblázat egy pozíciójában vagy egy érték található, vagy pedig a NIL.
- Egy adott elem keresésekor végignézzük a táblázat elemeit, amíg vagy meg nem találjuk az adott elemet, vagy pedig ki nem tudjuk jelenteni, hogy az adott elem nincs benne a táblázatban.
- A legfontosabb előnye a nyílt címzés módszerének, hogy az adott táblázaton túl nincs szükség más adatszerkezet használatára, és mutatók kezelésére.

Nyílt címzés módszere – műveletek

```
HASH-INSERT(T, k)
1  i = 0
2  repeat
3       j = h(k, i)
4       if T[j] == NIL
5            T[j] = k
6            return j
7       else i = i + 1
8  until i == m
9  error "hash table overflow"
```

Nyílt címzés módszere – műveletek

```
HASH-SEARCH(T, k)
1  i = 0
2  repeat
3       j = h(k, i)
4       if T[j] == k
5            return j
6       i = i + 1
7  until T[j] == NIL or i == m
8  return NIL
```

Nyílt címzés módszere – műveletek

Deletion from an open-address hash table is difficult. When we delete a key from slot i, we cannot simply mark that slot as empty by storing NIL in it. If we did, we might be unable to retrieve any key k during whose insertion we had probed slot i and found it occupied. We can solve this problem by marking the slot, storing in it the special value DELETED instead of NIL. We would then modify the procedure HASH-INSERT to treat such a slot as if it were empty so that we can insert a new key there. We do not need to modify HASH-SEARCH, since it will pass over DELETED values while searching. When we use the special value DELETED, however, search times no longer depend on the load factor α, and for this reason chaining is more commonly selected as a collision resolution technique when keys must be deleted.

A többit gyakorlaton...

Irodalomjegyzék