ADATSZERKEZETEK ÉS ALGORITMUSOK

Leszámláló rendező, Edényrendező

Leszámláló rendezés

- Előadáson beláttuk, hogy összehasonlító rendezés nem lehet gyorsabb, mint O(n log n).

- Az edényrendezés időigénye O(n+k), tehát nem lehet összehasonlító rendezés.
  - k-t lásd később

- Akkor hogyan működik?

Leszámláló rendezés

- Tegyük fel, hogy a bemeneti sorozat értékkészlete véges.
  - Pl. csak 0 és 10000 közötti számok lehetnek benne
- k legyen a lehetséges értékek száma.
- Lefoglalunk egy k elemű t tömböt.
- Mindegyik elemet betesszük a megfelelő sorszámú „edénybe”.
- A t tömböt a bejövő értékekkel indexeljük, ez konstans idejű ⇒ nincs összehasonlítás.

Leszámláló rendezés

- Ha minden értéket beolvastunk, végigmegyünk a t tömbön és kumulatívan összeadjuk az edények méreteit.
  - Végül a bemeneti tömbön újra végig menve az edények kezdő indexei segítségével (és azokat inkrementálva) feltöltjük az eredmény tömböt.
- A teljes rendezés tehát csak végigiterálás két sorozaton
  - a bemeneten és az edényeken.
- A sorozat n elemű, a tömbünk k hosszú, így a teljes rendezés O(n+k) futásidejű komplexitású.
- Amire viszont figyelni kell, hogy a memóriabeli komplexitása: O(n+k), szemben az eddig tanult összehasonlító rendezések O(n) komplexitásával.

Rendezés egész számokra

- **A legegyszerűbb eset**: egész számok 0-tól k-ig
- A t tömb elemei számlálók: hány darab volt az adott számból?

- **Eggyel bonyolultabb**: egész számok a-tól a+k-ig
- Az indexelést arrébb kell tolni a-val

Edényrendezés

- **Következő szint**: Összetett objektumok rendezése
  - Számon kell tartani az objektumokat: legyenek az edények sorok!
  - Az adott objektumot mindig a megfelelő edénybe tesszük, a végén végigmegyünk az edényeken, és mindegyikből kivesszük a benne található összes objektumot.
  - Melyik a megfelelő edény?
    - Kell egy függvény, amelyik mindegyik objektumból egy számot képez, és ezzel indexeljük az edényeket
    - Eddigi rendezések során csak relációkat vizsgáltunk!.
    - Ennek a függvénynek azt kell tudnia, hogy ha a rendezési elv egy objektum nagyobb, mint a másik, akkor a belőle képezett szám is legyen nagyobb, mint a másiké.
    - Ha egyenlők, akkor tartozzon hozzájuk ugyanaz a szám (monoton függvény).
  - Ez a függvény a kulcsfüggvény.

Rendezés objektumokra

- **Például**: Car objektumokat szeretnénk rendezni, a numberOfDoors mezője alapján, növekvő sorrendbe.
  - Ekkor függvénynek megfelel egy olyan, amelyik a numberOfDoors mezőt adja vissza
  - Azért előnyös, hogy ha FIFO adatszerkezetet (sort) használunk edényként, mert ekkor a rendezés szerint egyenlő objektumok megőrzik eredeti sorrendjüket: stabil rendezés.

Kulcsfüggvény - függvénnyel

- Annak érdekében, hogy ugyanaz az edényrendező algoritmus többféleképp is tudjon rendezni, érdemes a kulcsfüggvényt paraméterként átadni neki.

```
template<typename T, typename Keyfun>
void bucketSort(T vec[], size_t size, Keyfun f) {
   //...
}
```

- Ez lehet ténylegesen egy függvény:

```
int keyfunc(const Car& c) {
    return c.numberOfDoors;
}
```

Kulcsfüggvény - funktor

- Lehet funktor is.
  - A funktor egy olyan osztály, amelynek meg van valósítva a zárójel operátora.
  - Jelen esetben egy számot kell visszaadnia:

```
class KeyFunction {
public:
  int operator () (const Car& car) {
    return car.numberOfDoors;
  }
};
```

Kulcsfüggvény - lambda

- Végezetül lehet egy lambda.
  - A lambda-kifejezés C++-ban egy anonim lokális függvény, ami esetleg hivatkozhat a környezetében lévő változókra.
  - Bővebben: http://en.cppreference.com/w/cpp/language/lambda

```
auto keyfunction = [](const Car& car) {
                return car.numberOfDoors;
        }
```

Edényrendezés – folytonos eset

- **Még bonyolultabb**: folytonos adatok rendezése
  - Pl. valós számok, vagy valós számokra leképeződő objektumok
- Ekkor az edényeink intervallumokat jelölnek
  - Az edényeink lehetnek láncolt listák
  - A listákban beszúrásos rendezéssel (insertion sort) tartunk rendet – vagy a végén egyéb rendezővel.
  - Ekkor már nem garantált az O(n+k) futásidő, okosan megválasztott intervallumokkal el lehet érni viszont, hogy ennyi legyen az átlagos futásidő.

Edényrendezésnél megfontolandó

Általában nem találkozunk végtelen adatsorral, tehát az értékkészlet mindig véges, mégsem használunk mindig edényrendezést.

Ha az adatok nagyon szétszórva helyezkednek el, mondjuk egy 10000 hosszú intervallumon 10 darab, akkor feleslegesen sok memóriát használnánk el, és időben is lehet túl sok lefoglalni és iterálni.

Akkor érdemes tehát használni, ha az adatok az intervallumon elég sűrűn laknak.

Radixrendezés (Radix sort)

- Tegyük fel, hogy nem tudunk az objektumainkhoz rendezéstartó módon egész számokat rendelni.
  - Pl. szöveg típus
- Tudunk viszont egy egészekből álló összetett kulcsot gyártani.
  - Szöveg típus esetén karakterkódok sorozata
- Ekkor lexikografikusan rendezünk: ha különbözik, akkor az első kulcs (karakter): k_m alapján, ha az első kulcs (karakter) megegyezik, akkor a második kulcs (karakter): k_{m-1} alapján, ha az is megegyezik, akkor a harmadik (k_{m-2}) alapján, stb.

Radixrendezés (Radix sort)

- A rendezés megírása során azonban visszafele gondolkozunk!
  - Először edényrendezzük k_m szerint.
  - Aztán edényrendezzük k_{m-1} szerint. Az edényrendezés konzervatív, tehát ez megtartja a k_{m-1} szerinti sorrendet.
  - …
  - Végül k_1 szerint is rendezzük.

Radixrendezés (Radix sort)

- **Például**: rendezzünk dátumokat (év,hó,nap)!
  - A gondolatmenet alapján, először rendezünk a napok szerint (k_3).
  - Majd rendezünk a hónapok szerint (k_2), ekkor ugye ha a hónapok megegyeznek, a napok különbsége számít (mivel a rendezés konzervatív). Végül rendezünk év (k_1) szerint.
  - Ekkor alapjában az évszám különbsége dönt (amelyik dátumnak későbbi évszáma van, az későbbinek is számít), hiszen ez alapján rendeztünk utoljára, viszont egyezés esetén a hónap számít, hiszen előtte az alapján rendeztük.
  - Ha a hónap is megegyezik, akkor számít a napok szerinti rendezés.