Bucket Sort a Poriadkové štatistiky

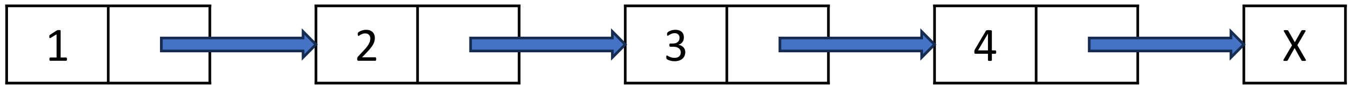
Bc. Katarína Olejková

KATEDRA INFORMATIKY

UNIVERZITA PALACKÉHO V OLOMOUCI

Bucket Sort

- Triedime čísla z intervalu [0, 1)
- Využíva dynamickú dátovu štruktúru nazývanú SPOJOVÝ SEZNAM:
  - Každý prvok seznamu obsahuje **hodnotu** a **odkaz** na ďalší prvok seznamu



Jednosmerný spojový seznam

Bucket Sort priebeh

1. Interval [0, 1) rozdelíme na n – podintervalov rovnakej veľkosti
2. Pre každý podinterval vytvoríme spojový seznam B[i] nazývané “buckets“
3. Prechádzame prvky vstupného poľa A a vložíme ich do príslušného bucketu/spojového seznamu B[i]
4. Každý bucket/spojový seznam B[i] zotriedime
5. Prvky zotriedených bucketov/spojových seznamov B[i]…B[n – 1] vložíme poporade do vstupného poľa A = máme zotriedené pole

```
Bucket-Sort(A[0..n-1])
    for i ← 0 to n - 1
        vlož A[i] do seznamu B[⌊n * A[i]⌋]
    for i ← 0 to n - 1
        Sort(B[i])
    vlož postupně prvky z B[0],...,B[n – 1] do pole A
```

Vloží prvok z A do príslušného B[i]

Zotriedi každý B[i] samostatne pomocou nejakého triediaceho algoritmu

A – vstupné pole
B – pole bucketov

i – index
n – počet prvkov

Bucket Sort - časová zložitosť

- V najhoršom prípade
  - Vtedy keby boli všetky prvky z A umiestnené v jednom buckete B[i]
  - Prvý cyklus + triediaci algoritmus + druhý cyklus

    $\Theta(n) \quad + \quad \Theta(f(n)) \quad + \quad \Theta(n) \quad = \quad \mathbf{\Theta(f(n))}$
  - Bucket-Sort bude mať potom čas. zlož. toho algoritmu, ktorý zvolíme za Sort(B[i])

- V priemernom prípade
  - $O(2 – 1/n) = \Theta(n)$

Poriadkové štatistiky

- i-tá poriadková štatistika = i-tý najmenší prvok (v množine s n prvkami)
  - Najmenší prvok ... prvá poriadková štatistika (i = 1)
  - Najväčší prvok ... n-tá poriadková štatistika (i = n)

- Výber i-tej štatistiky z poľa A
  - Naivný prístup – zotriedime pole A vzostupne a vrátime i-tý prvok
  - Existuje lepší algoritmus založený na podobnej myšlienke ako QuickSort

Poriadková štatistika v priemernom čase Θ(n)

- Rekurzívny algoritmus – narozdiel od QuickSort, ktorý sa rekurzívne volá na obidve časti (ľavá a pravá), tento algoritmus sa bude rekurzívne volať iba na **jednu** časť – tam, kde sa i-tá poriadková štatistika bude nachádzať

- Použijeme funkciu Partition:
  - podľa pivota rozdelí pole na ľavú a pravú polovicu
  - na rozdiel od QuickSortu, kde vyberáme vždy posledný prvok ako pivot, teraz použijeme Randomized-Partition, ktorý vyberie za pivota náhodný prvok
  - q – index pivota, k - pivot

- Môžu nastať 3 prípady:
  1. i = k   našli sme i-tú poriadkovú štatistiku prvok A[i] = k
  2. i < k   i-tá poriadková štatistika je i-tý prvok v zotriedenej ľavej časti
  3. i > k   i-tá poriadková štatistika je i-tý prvok v zotriedenej pravej časti

```
Randomized-Select(A, p, r, i)
if p = r
  then return A[p]
q ← Randomized-Partition(A, p, r)
k ← q − p + 1
if i = k
  then return A[q]
else if i < k
  then return Randomized-Select(A, p, q − 1, i)
else return Randomized-Select(A, q + 1, r, i − q + p − 1)
```

Vrátí i. pořádkovou statistiku pole A.

```
Randomized-Partition(A, p, r)
k ← Random(p, r)
swap(A[k], A[r])
return Partition(A, p, r)
```

```
Partition(A,p,r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
    do if A[j] ≤ x
          then i ← i + 1
               swap(A[i], A[j])
swap(A[i + 1],A[r])
return i + 1
```

Randomized-Select - časová zložitosť

- V najhoršom prípade:
  - Napr. keď hľadáme prvú poriadkovú štatistiku (najmenší prvok) a náhodne bude vždy zvolený najväčší prvok ako pivot
  - $\Theta(n^2)$
  - Existuje lepší algoritmus, ktorý má O(n)

- V priemernom prípade
  - Θ(n) – pomocou aparátu z teórie pravdepodobnosti