Föreläsning 16 i ADK

Algoritmkonstruktion: sortering i linjär tid

Stefan Nilsson

KTH

Räknesortering

function CountingSort(V[1..n], f : element $\to$ [1..k])

```
    C[1..k]: Hjälparray för räkning
    res[1..n]: Hjälparray för lagring av resultatet
    for i ← 1 to k do C[i] ← 0
    for j ← 1 to n do C[f(V[j])]++
    sum ← 0
    for i ← k downto 1 do
        sum ← sum + C[i]
        C[i] ← n − sum + 1
    for j ← 1 to n do
        res[C[f(v[j])]] ← v[j]
        C[f(v[j])]++
    return res
```

Tidskomplexitet: $\Theta(k + n)$

Räknesortering

Exempel

- `V[1..10]` $= [17, 2, 2, 4711, 2, 17, 17, 4711, 2, 17]$
- $f(2) = 1, f(17) = 2, f(4711) = 3$
- `C[1..3]` efter räkneslingan $= [4, 4, 2]$
- `C[1..3]` efter indexomräkningen $[1, 5, 9]$
- `res[1..10]` $= [2, 2, 2, 2 \mid 17, 17, 17, 17 \mid 4711, 4711]$

Några fall då man kan sortera i $o(n \log n)$

1. Bara ett konstant antal olika element ska sorteras:
   - $\Theta(n)$ med räknesortering (counting sort)
2. Elementen som ska sorteras är tal som är jämnt fördelade i ett visst intervall:
   - $\Theta(n)$ med bucket sort
3. Elementen som ska sorteras är strängar som består av d "siffror" ($v[i] = s_{i,1}s_{i,2}\ldots s_{i,d}$)
   - $\Theta(nd)$ med radixsortering
   - **for** $i \leftarrow d$ **downto** 1 **do**
     Sortera $v[1..n]$ efter siffra i med en stabil sorteringsalgoritm
   - Om d är konstant får vi linjär tidskomplexitet
   - Om vi räknar antalet siffror i indata få vi linjär tidskomplexitet $\Theta(N)$ där $N = nd$

Radixsorteringsexempel med $d = 3$

Osorterat:	Pass 1:	Pass 2:	Pass 3:
480	480	902	009
973	381	905	381
902	902	009	419
905	532	816	480
532	652	419	532
652	973	532	652
783	783	652	653
009	653	653	783
653	905	973	816
419	816	480	902
816	009	381	905
381	419	783	973