Sortering i linær tid

Søren Dahlgaard

Datalogisk Institut, Københavns Universitet

Diskret Matematik og Algoritmer, 2015

Motivation

Sammenligningsbaserede sorteringsalgoritmer bruger $\Omega(n \log n)$ tid.

Hvad hvis vi må andet end at sammenligne?

Kan vi antage noget om input der hjælper?

- Små heltal.
- God distribution.
- ...

Counting sort

Sortere små heltal

Virker for heltal i $\{0, 1, \ldots, k-1\}$. God når k er lille (f.eks. $O(n)$).

Hovedidé: Hold en tabel, C, af størrelse k, hvor plads i tæller hvor mange i'ere der er i input.
Brug C til at genskabe det sorterede array.

Bemærk, at vi ikke afhænger af sammenligninger! Derfor kan vi slå $\Omega(n \log n)$.

Counting sort

Beskrivelse af algoritmen

Lad A være input, med heltal mellem 0 og $k-1$, der skal sorteres.

Vi kan se algoritmen som tre faser:

1. Tæl hvor mange af hvert tal der er.
2. Akkumulér tællerne, så vi kan regne positioner ud.
3. Genskab den sorterede tabel.

Til alle faser bruger vi en tabel C af størrelse k.

Counting sort

Fase 1

1. Gå igennem hver indgang i A, $i = 0, 1, \ldots, n-1$.
2. For hver indgang, tæl $C[A[i]]$ en op
   $(C[A[i]] \leftarrow C[A[i]] + 1)$.

Eksempel med $n = 7$ og $k = 5$.

$$A = \begin{array}{|c|c|c|c|c|c|c|}\hline 3 & 2 & 0 & 4 & 2 & 2 & 3 \\ \hline\end{array}$$

$$\downarrow$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 1 & 0 & 3 & 2 & 1 \\ \hline\end{array}$$

Counting sort

Fase 2

Vi ved nu hvor mange der er af hver. Men hvordan placerer vi dem?

$$C = \boxed{1}\boxed{0}\boxed{3}\boxed{2}\boxed{1}$$

Observer, at det sidste 0 skal være på plads $1 - 1$. Det sidste 2 skal være på plads $3 + 1 - 1$. Det sidste 3 skal være på plads $2 + 3 + 1 - 1$, osv.

Idé: Akkumulér C tabellen så vi kan læse positionerne direkte! (se næste slide).

Counting sort

Fase 2

1. Gå igennem hver indgang i C (udover den første), $i = 1, 2, \ldots, k-1$.
2. For hver indgang akkumulér indtil videre (sæt $C[i] = C[i-1] + C[i]$).

Eksempel fortsat:

$$C = \boxed{1}\boxed{0}\boxed{3}\boxed{2}\boxed{1}$$
$$\downarrow$$
$$C = \boxed{1}\boxed{1}\boxed{4}\boxed{6}\boxed{7}$$

Bemærk, at det sidste 2 skal være på plads $4-1$ (0-indekseret) i sorteret rækkefølge osv.

Counting sort

Fase 3

1. Gå igennem hver indgang i A i omvendt rækkefølge, $i = n-1, n-2, \ldots, 0$.
2. For hver indgang, tæl $C[A[i]]$ en ned, og placér $A[i]$ på plads $C[A[i]]$.

$$A = \begin{array}{|c|c|c|c|c|c|c|}\hline 3 & 2 & 0 & 4 & 2 & 2 & 3 \\ \hline \end{array}$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 4 & 6 & 7 \\ \hline \end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline & & & & & & \\ \hline \end{array}$$

$$\downarrow$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 4 & 5 & 7 \\ \hline \end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline & & & & & 3 & \\ \hline \end{array}$$

$$\downarrow$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 3 & 5 & 7 \\ \hline \end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline & & & 2 & & 3 & \\ \hline \end{array}$$

Counting sort

Fase 3, eksempel fortsat

$$A = \begin{array}{|c|c|c|c|c|c|c|}\hline 3 & 2 & 0 & 4 & 2 & 2 & 3 \\ \hline\end{array}$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 2 & 5 & 7 \\ \hline\end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline & & 2 & 2 & & 3 & \\ \hline\end{array}$$

$$\downarrow$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 1 & 1 & 2 & 5 & 6 \\ \hline\end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline & & 2 & 2 & & 3 & 4 \\ \hline\end{array}$$

$$\downarrow$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 2 & 5 & 6 \\ \hline\end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline 0 & & 2 & 2 & & 3 & 4 \\ \hline\end{array}$$

$$\downarrow$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 1 & 5 & 6 \\ \hline\end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 2 & 2 & 2 & & 3 & 4 \\ \hline\end{array}$$

$$\downarrow$$

$$C = \begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 1 & 4 & 6 \\ \hline\end{array} \quad A' = \begin{array}{|c|c|c|c|c|c|c|}\hline 0 & 2 & 2 & 2 & 3 & 3 & 4 \\ \hline\end{array}$$

Counting sort

Opsamling og konklusion

Med counting sort kan vi sortere i $O(n+k)$ tid. Det kræver dog, at input er heltal i $\{0, 1, \ldots, k-1\}$.

Idéen er blot at tælle hvor mange af hvert element der er, og skabe en sorteret tabel ud fra dette.

Bemærk, at counting sort er stabil.

Se CLRS for pseudokode, mv.

Radix sort

Sortere tal med få cifre

Virker for tal, hvor vi kan læse et ciffer ad gangen.

Hovedidé: For hver cifferposition (startende med mindst betydende), brug en *stabil* sorteringsalgoritme, til at sortere tabellen mht. dette ciffer.

Bemærk, at da vi deler tallene op i cifre afhænger vi ikke af sammenligninger, og kan vi slå $\Omega(n \log n)$.

Radix sort

Pseudocode

Algorithm 1: `Radix-Sort`

input : Array A, number of digits d

for $i = 1 \to d$ **do**

    Sort A according to ith digit using stable sort

Observe, that we can use counting sort for each step!

Radix sort

Eksempel

24.6	120.1	120.1	106.2	24.6
120.1	54.1	31.6	120.1	31.6
34.9	106.2	54.1	24.6	34.9
54.1	24.6	24.6	31.6	54.1
31.6	31.6	34.9	34.9	106.2
106.2	34.9	106.2	54.1	120.1

Bemærk, at det er meget vigtigt, at der bliver brugt en stabil sortering. (Ellers kunne 106.2 og 120.1 f.eks. bytte plads til sidst).

Radix sort

Opsamling

Med radix sort kan vi sortere tal baseret på deres cifre.

Tidskompleksiteten er $O(n \cdot d)$ – under nogle antagelser. Se CLRS for flere detaljer.

Køretiden afhænger af den “interne sortering”.

Findes også variant der starter med mest betydende ciffer.

Bemærk, at vi ofte antager, at de tal der skal sorteres har w cifre[1] og $w = \Omega(\log n)$.

---

[1] w er ordlængden

Bucket sort

Sortere pænt distribuerede tal

Counting sort antog at tallene var små. Hvad vi de er store, men nogenlunde pænt fordelt?

Hovedidé: Put hvert tal i en af k spande (jo større tal, jo større spand index). Sortér hver spand for sig. Konkatenér spandene til det sorterede array.

Ved at tildele hvert tal en spand baseret på tallets størrelse undgår vi sammenligningsgrænsen på $\Omega(n \log n)$.

Bucket sort

Eksempel

$A =$

6	87	42	8	54	32	91	4	85	43	13

Lav en spand til intervallerne $[0, 10), [10, 19), \dots, [90, 100)$:

4									
8				43				85	
6	13	$\emptyset$	32	42	54	$\emptyset$	$\emptyset$	87	91

Nederst i spand = først i A. Brug nu insertion sort på hver spand:

8									
6				43				87	
4	13	$\emptyset$	32	42	54	$\emptyset$	$\emptyset$	85	91

Bucket sort

Eksempel fortsat

(kopieret fra forrige side)

8									
6				43				87	
4	13	$\emptyset$	32	42	54	$\emptyset$	$\emptyset$	85	91

Konkatenér spandene for at få sorteret version af A:

$A' =$

4	6	8	13	32	42	43	54	85	87	91

Bucket sort

Opsamling

Bucket sort kan sortere tal, der "er fordelt pænt". Hvis alle tal f.eks. er i $[0, 10)$, og vi har lavet spandene som i eksemplet vil det tage lang tid.

Intuitivt: Hvis $k = \Theta(n)$ kommer der cirka $O(1)$ element i hver spand, og hver insertion sort tager altså $O(1)$ tid!

Vi kan bruge andre sorteringsalgoritmer i spandende end insertion sort. Vi kan også lave analyse af input først for at vælge smarte spande!

Konklusion

- Sammenligningssortering kræver $\Omega(n \log n)$, men vi kan ofte gøre det bedre!
- Smartere sorteringsalgoritmer udnytter struktur i input.
- Counting sort er god til små heltal.
- Radix sort er god til tal med få cifre.
- Bucket sort er god til tal, der fordeler sig pænt i de valgte spande.