Kapitel 1: Einführung

(Effiziente Algorithmen, WS 2018)

Gerhard Woeginger

WS 2018, RWTH

Einführung: Einige Suchprobleme

- Grösste Zahl
- Maximum und Minimum
- Zweitgrösste Zahl

- Median (worst case)
- Median (randomisiert)

Zum Aufwärmen: Grösste Zahl

Maximum (1): Problemstellung

Algorithmisches Problem: Maximum

Eingabe: Ganze Zahlen $a_1, a_2, \ldots, a_n$

Gesucht: Das Maximum $\max\{a_1, a_2, \ldots, a_n\}$

- Teure Operation: Vergleich von zwei Zahlen a_i und a_j
- Liefert $a_i < a_j$ oder $a_i = a_j$ oder $a_i > a_j$ als Antwort
- Kein Zugriff auf den Zahlenwert von a_i
- Verwaltungsoperationen (FOR-Loops; Indexoperationen; Rechenoperationen; etc) sind gratis

Frage

Wie sieht ein guter Algorithmus fuer dieses Problem aus,
der auch im Worst Case nur wenige Vergleiche macht?

Maximum (2): Algorithmus

```
k= n;
for i= 1 to n-1 do
    if (a[i] > a[k]) then k= i;

return k;
```

Satz

Das Maximum Problem
kann mit $n-1$ Vergleichen (im Worst Case) gelöst werden.

Frage

Gibt es einen besseren Algorithmus?

Maximum (3): Untere Schranke

- Betrachte beliebigen Algorithmus A für das Maximum Problem
- Konstruiere Hilfsgraphen $G = (V, E)$ mit $V = \{1, 2, \ldots, n\}$
- Am Anfang ist die Kantenmenge E leer
- Wenn Algorithmus A die Zahlen a_i und a_j vergleicht, fügen wir die Kante $\{i, j\}$ ein
- Solange $G = (V, E)$ nicht zusammenhängend ist, hat A das Maximum Problem noch nicht gelöst (Warum?)

(Einfacher) Satz aus der Graphentheorie

Wenn $G = (V, E)$ zusammenhängend, dann $|E| \geq |V| - 1$.

Ergo: Die $n - 1$ Vergleiche sind bereits bestmöglich

Maximum und Minimum

Max+Min (1): Problemstellung

Problem: Max+Min

Eingabe: Ganze Zahlen $a_1, a_2, \ldots, a_n$

Gesucht: $\max\{a_1, a_2, \ldots, a_n\}$ und $\min\{a_1, a_2, \ldots, a_n\}$

Wir nehmen an: n ist eine gerade Zahl mit $n = 2q$

Algorithmus:

- Teile die Zahlen in q Paare auf
- Bestimme in jedem Paar die grössere Zahl (Gewinner) und die kleinere Zahl (Verlierer)
- Bestimme das Minimum der Verlierer
- Bestimme das Maximum der Gewinner

Satz

Das Max+Min Problem kann mit $3q - 2$ Vergleichen gelöst werden.

Max+Min (2a): Untere Schranke

Klassifikation der Zahlen in vier Typen:

- U: unberührt; wurde noch nie verglichen
- G: Gewinner; hat einen Vergleich gewonnen, noch nie verloren
- V: Verlierer; hat einen Vergleich verloren, noch nie gewonnen
- E: Eliminiert; hat Vergleich gewonnen und Vergleich verloren

Gewichte:

- Jede Zahl in U hat Gewicht 2
- Jede Zahl in G hat Gewicht 1
- Jede Zahl in V hat Gewicht 1
- Jede Zahl in E hat Gewicht 0

Max+Min (2b): Untere Schranke

Grundidee

Wir betrachten einen beliebigen Algorithmus A, und
lassen einen bösartigen Gegenspieler (“Adversary”) gegen A arbeiten.

Der Adversary beantwortet Vergleiche derart, dass möglichst wenig Gewicht verloren geht (aber konsistent mit vorherigen Vergleichen)

- U versus U: Gewichtsverlust $= 2$
- V versus V: Gewichtsverlust $= 1$
- G versus G: Gewichtsverlust $= 1$
- U versus $G/V/E$: Gewichtsverlust $= 1$
- E versus $G/V/E$: Gewichtsverlust $= 0$
- G versus V: Gewichtsverlust $= 0$

Max+Min (2c): Untere Schranke

Definition

v_0 := Anzahl der Vergleiche, die Gesamtgewicht nicht senken
v_1 := Anzahl der Vergleiche, die Gesamtgewicht um 1 senken
v_2 := Anzahl der Vergleiche, die Gesamtgewicht um 2 senken

- Gesamtgewicht fällt von $2n$ (am Anfang) auf 2 (am Ende)
  Ergo: $v_1 + 2v_2 = 2n - 2 = 4q - 2$
- Nur U versus U Vergleiche senken Gewicht um 2
  Ergo: $v_2 \leq n/2 = q$
- Daher gilt: $v_1 + v_2 \;=\; 4q - 2 - v_2 \;\geq\; 4q - 2 - q$

Satz

Im Worst Case braucht man $3q - 2$ Vergleiche für Max+Min.

Zweitgrösste Zahl

Zweitgrösste Zahl (1): Problemstellung

Problem: Zweitgrösste Zahl

Eingabe: Ganze Zahlen $a_1, a_2, \ldots, a_n$

Gesucht: Die grösste und die zweitgrösste Zahl unter $a_1, a_2, \ldots, a_n$

Wir nehmen an: n ist eine Zweierpotenz mit $n = 2^q$

Algorithmus:

- Wir schreiben $a_1, a_2, \ldots, a_n$ in die Blätter eines Binärbaumes
- Jeder Vaterknoten enthält das Maximum der beiden Söhne
- Die grösste Zahl steht in der Wurzel
- Die zweitgrösste Zahl hat Vergleich gegen grösste Zahl verloren

Satz

Die zweitgrösste Zahl kann mit $2^q + q - 2$ Vergleichen gefunden werden.

Zweitgrösste Zahl (2a): Untere Schranke

- Jede Zahl a_i wird als Partikel mit gewisser Energie angesehen
- Die Anfangsenergie jedes Partikels ist 1
- Werden zwei Zahlen mit einander verglichen, so erhält Gewinner gesamte Energie des Verlierers dazu (Verlierer hat dann Energie 0)

Anmerkungen:

- Die Gesamtenergie bleibt konstant.
- Am Ende hat Maximum die gesamte Energie $n = 2^q$ angesammelt; alle anderen Partikel haben Energie 0

Grundidee

Der Adversary beantwortet die Vergleiche immer derart,
dass der Partikel mit der höheren Energie den Vergleich gewinnt.
(Tiebreaking = beliebig; Partikel mit Energie 0 = konsistent)

Zweitgrösste Zahl (2b): Untere Schranke

Am Ende hat das Maximum an vielen Vergleichen teilgenommen:

- Vor seinem letzten Vergleich hatte Maximum Energie $\geq 2^{q-1}$
- Vor dem vorletzten Vergleich hatte Maximum Energie $\geq 2^{q-2}$
- Etc. Etc.
- Ergo: Maximum hat mindestens q Vergleiche gewonnen

Analyse

- Jedes Nicht-Maximum hat mindestens einen Vergleich verloren.
- Jede nicht-zweitgrösste Zahl, die gegen Maximum verloren hat, hat mindestens einen weiteren Vergleich verloren.

Ergo: Mindestens $(2^q - 1) + (q - 1)$ Vergleiche

k-kleinste Zahl und Median

k-kleinste Zahl: Problemstellung

Problem: k-kleinste Zahl

Eingabe: Ganze Zahlen $a_1, a_2, \ldots, a_n$ und k

Gesucht: Die k-kleinste Zahl unter $a_1, a_2, \ldots, a_n$

- $k = 1$: Minimum
- $k = n$: Maximum
- $k = n/2$: Median

Problem: Median

Eingabe: Ganze Zahlen $a_1, a_2, \ldots, a_n$

Gesucht: Der Median der Zahlen $a_1, a_2, \ldots, a_n$

k-kleinste Zahl: Ein schneller Algorithmus

Einfache Idee:

- Sortiere die Zahlen $a_1, a_2, \ldots, a_n$
- Gib die k-te Zahl in der Sortierung aus

Satz

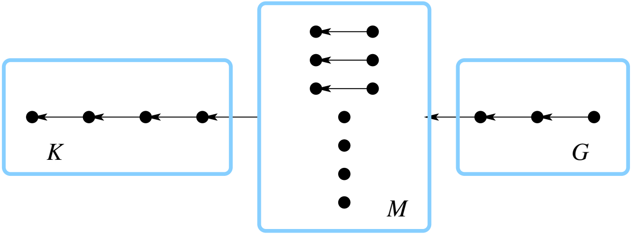
Die k-kleinste Zahl und der Median
können mit $O(n \log n)$ Vergleichen gefunden werden.

Median: Untere Schranke (1)

Der Adversary hält folgende Struktur aufrecht:

Die Zahlen $a_1, a_2, \ldots, a_n$ sind in drei Mengen aufgeteilt.

- Die Menge G der grossen Zahlen ist total geordnet.
- Die Menge K der kleinen Zahlen ist total geordnet.
- Die Menge M der mittleren Zahlen besteht aus vergleichbaren Paaren und Einzelelementen.
- Für alle $k \in K$, $m \in M$, $g \in G$ gilt $k < m < g$



Median: Untere Schranke (2a)

Angenommen, der Algorithmus möchte wissen, wie sich die beiden Elemente x und y zu einander verhalten:

- Falls $x, y \in K$, oder $x, y \in G$, so antwortet der Adversary gemäss der totalen Ordnung.
- Falls $x \in K$ und $y \in M \cup G$, oder $x \in M$ und $y \in G$, so antwortet der Adversary "$x < y$".

In den verbleibenden Fällen gilt $x, y \in M$.

Median: Untere Schranke (2b)

Wir definieren einen Hilfswert $\alpha(x)$ für jedes $x \in M$:

- $\alpha(x) = 0$, falls x Einzelement
- $\alpha(x) = -1$, falls x kleines Element in Paar
- $\alpha(x) = +1$, falls x grosses Element in Paar

Muss der Adversary zwei Elemente x und y mit $\alpha(x) \leq \alpha(y)$ vergleichen, so antwortet er "$x < y$".

- Falls $\alpha(x) = \alpha(y) = 0$, so passiert weiter nichts.
- Andernfalls: Wenn $\alpha(x) = -1$, so wird x aus M entfernt und wandert als neues grösstes Element nach K
- Andernfalls: Wenn $\alpha(x) \in \{0, 1\}$, so wird y aus M entfernt und wandert als neues keinstes Element nach G

Median: Untere Schranke (3a)

Es sei v die Anzahl der bisher gemachten Vergleiche.
Es sei p die Anzahl der vergleichbaren Paare in M.

Lemma

Der Adversary hält die Invariante $v \geq 2n + p - 2|M|$ aufrecht.

Beweisskizze:

Falls $\alpha(x) = \alpha(y) = 0$: ein neues Paar entsteht
Andernfalls: Ein Element verschwindet aus M, ein Paar löst sich auf

Median: Untere Schranke (3b)

- Wir betrachten den Zeitpunkt t,
  wenn zum ersten Mal $|G| = n/2 - 1$ oder $|K| = n/2 - 1$ gilt.
- Vor diesem Zeitpunkt t steht es dem Adversary völlig frei zu entscheiden, ob ein Element von M nach K oder nach G gehört.
- Zum Zeitpunkt t kann dann aber nur noch das kleinste/grösste Element von M nach K/G gehören.

Lemma

Zum Zeitpunkt t gilt die triviale Schranke $|M| \leq n/2 + 1$

Median: Untere Schranke (4)

Zum Zeitpunkt t gilt also:

- Invariante $v \geq 2n + p - 2|M|$
- Schranke $|M| \leq n/2 + 1$

Satz

Im Worst Case braucht man mindestens $3n/2 - 2$ Vergleiche um den Median zu finden.

Beweisskizze:

- Bis zum Zeitpunkt t hat der Algorithmus mindestens $2n + p - 2|M|$ Vergleiche gemacht.
- Nach dem Zeitpunkt t muss der Algorithmus dann noch das Minimum/Maximum von M herausfinden. Dazu braucht er $|M| - p - 1$ weitere Vergleiche.
- Insgesamt braucht der Algorithmus also mindestens $2n - |M| - 1$ Vergleiche.

k-kleinste Zahl (und Median) in linearer Zeit

k-kleinste Zahl in linearer Zeit: Hilfsprozedur

Eingabe: Ein Array $A[1 \dots n]$; eine Zahl x

Ausgabe: Eine Umsortierung der Zahlen in A, in der alle Zahlen $\leq x$ links von den Zahlen $> x$ stehen

```
PARTITION (A,x)
i= 0
for j= 1 to n do
    if A[j]<=x then
       {i= i+1;
        exchange A[i] with A[j]}
return i
```

Invariante: Jeder Index p mit $1 \leq p \leq i$ erfüllt $A[p] \leq x$

Invariante: Jeder Index p mit $i + 1 \leq p \leq j - 1$ erfüllt $A[p] > x$

k-kleinste Zahl in linearer Zeit: Hauptprogramm

1. Falls $n \leq 20$: Bestimme k-kleinste Zahl durch Sortieren. Stop.
2. Teile die n Zahlen in $\lfloor n/5 \rfloor$ Fünfergruppen auf
   (und eine Restgruppe mit den restlichen $n \bmod 5$ Zahlen)
3. Sortiere in jeder dieser $\lceil n/5 \rceil$ Gruppen die Zahlen,
   und bestimme den Median der Gruppe
4. Bestimme (rekursiv) den Median x dieser $\lceil n/5 \rceil$ Mediane
5. PARTITIONiere das Eingabearray um den Median-der-Mediane x.
   Danach sind die ersten q Zahlen im Array $\leq x$,
   und die restlichen $n - q$ Zahlen sind $> x$.
6. Wenn $k = q$: Gib x als k-kleinste Zahl aus.
   Wenn $k < q$: Berechne rekursiv die k-kleinste Zahl in $A[1 \ldots q-1]$.
   Wenn $k > q$: Berechne rekursiv die $(k-q)$-kleinste Zahl
   in $A[q+1 \ldots n]$.

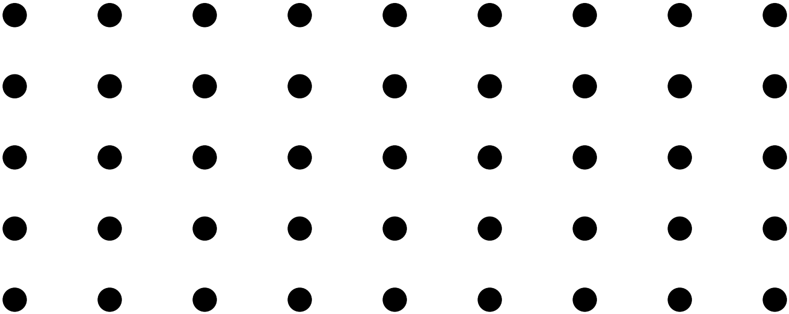
k-kleinste Zahl in linearer Zeit: Analyse (1)

- Aufteilung in Fünfergruppen: in linearer Zeit $O(n)$
- Sortieren der fünf Zahlen in jeder Gruppe: in linearer Zeit $O(n)$
- Bestimmung des Medians-der-Mediane: in Zeit $T(n/5)$
- PARTITION in Teil $\leq x$ und Teil $> x$: in linearer Zeit $O(n)$
- Rekursion: in Zeit $O(1)$ oder $T(q)$ oder $T(n-q)$

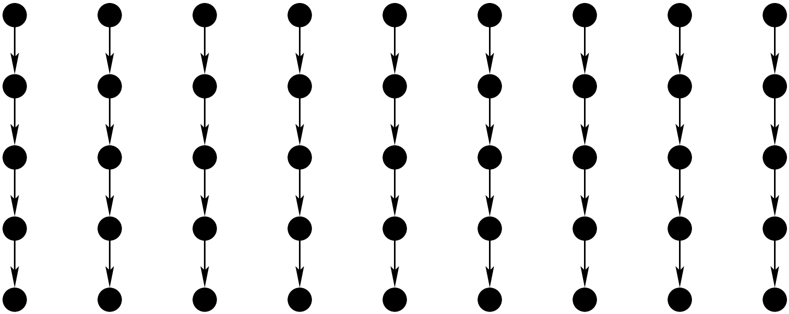
Wir erhalten die Zeitgleichung

$$T(n) \; = \; T(n/5) + \max\{\, T(q), \; T(n-q)\} + O(n)$$

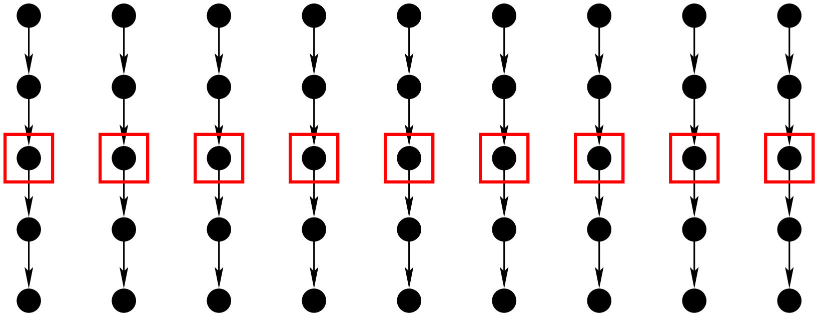
k-kleinste Zahl in linearer Zeit: Analyse (2)



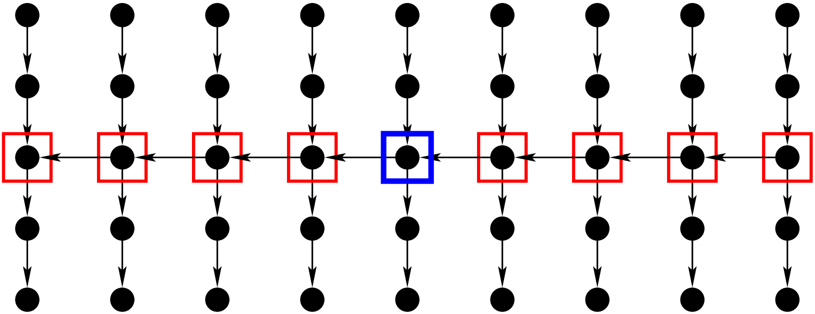
k-kleinste Zahl in linearer Zeit: Analyse (2)



k-kleinste Zahl in linearer Zeit: Analyse (2)



k-kleinste Zahl in linearer Zeit: Analyse (2)



k-kleinste Zahl in linearer Zeit: Analyse (3)

Zusammenfassende Beobachtung

Von den n Zahlen sind $\approx 3n/10$ garantiert $>$ Median-der-Mediane.
Von den n Zahlen sind $\approx 3n/10$ garantiert $\leq$ Median-der-Mediane.

Ergo: $3n/10 \leq q \leq 7n/10$

Die alte Zeitgleichung mit q

$$T(n) \;=\; T(n/5) + \max\{T(q),\; T(n-q)\} + O(n)$$

liefert uns nun

$$T(n) \;\leq\; T(n/5) + T(7n/10) + O(n).$$

k-kleinste Zahl in linearer Zeit: Analyse (4)

Satz

Die k-kleinste Zahl und der Median
können in linearer Zeit $O(n)$ berechnet werden.

Zur Erinnerung: Wenn für ein reelles $c > 0$

$$T(n) \;\le\; T(n/5) + T(7n/10) + cn$$

gilt, so kann man daraus mit vollständiger Induktion

$$T(n) \;\le\; d \cdot n$$

für jedes reelle $d \geq 10c$ herleiten.

Anmerkungen

Der $O(n)$ Algorithmus stammt von Manuel Blum, Robert Floyd, Vaughan Pratt, Ron Rivest und Robert Tarjan, und ist aus dem Jahr 1973.

Hausübung

(a) Zeige: Nimmt man Dreiergruppen statt Fünfergruppen, so erhält man keine lineare Laufzeit.

(b) Zeige: Nimmt man Siebenergruppen statt Fünfergruppen, so erhält man ebenfalls eine lineare Laufzeit.

Satz (Dorit Dor & Uri Zwick, 2001)

Der Median kann mit $2.95n$ Vergleichen gefunden werden.
Der Median kann nicht mit $(2 + 2^{-80})n$ Vergleichen gefunden werden.

k-kleinste Zahl (und Median) randomisiert

Randomisierter Algorithmus

- In der Praxis wird das Split-Element x einfach zufällig gewählt (und nicht in einer teuren Rekursion als Median-der-Mediane berechnet).
- Intuition: Mit hoher Wahrscheinlichkeit liegt der resultierende Wert q weit weg von 0 und weit weg von n.
- Die Worst Case Laufzeit dieses randomisierten Algorithmus ist quadratisch.

Wir werden zeigen:

Satz

Die erwartete Laufzeit dieses randomisierten Algorithmus ist $O(n)$.

Hilfsresultat

Fragestellung

Ein Experiment ist mit Wahrscheinlichkeit p erfolgreich.
Wie gross ist die erwartete Anzahl an Wiederholungen,
bis dass das Experiment einmal erfolgreich ausgeht?

- Es sei X die Zufallsvariable, die die Anzahl der Durchführungen zählt
- Dann gilt $Pr[X = k] = (1-p)^{k-1}p$

Antwort

$$\mathbb{E}[X] \;=\; \sum_{k=0}^{\infty} k \cdot Pr[X = k] \;=\; \sum_{k=1}^{\infty} k(1-p)^{k-1}p \;=\; \frac{1}{p}$$

Ergo: Für $p = 1/2$ gilt $\mathbb{E}[X] = 2$

Analyse (1)

- Wir sagen, dass der randomisierte Algorithmus in Phase k ist, falls die Anzahl der aktiven Zahlen kleiner gleich $(3/4)^k n$ und strikt grösser als $(3/4)^{k+1} n$ ist.
- Am Anfang ist der Algorithmus in Phase 0

- Eine Zahl x heisst zentral, wenn mindestens ein Viertel der aktiven Zahlen kleiner als x oder mindestens ein Viertel grösser als x ist
- Ist das Split-Element x zentral, dann wird mindestens ein Viertel der aktiven Zahlen weggeworfen

- Die Hälfte aller aktiven Zahlen ist zentral
- Die Wahrscheinlichkeit, dass das Split-Element zentral ist, beträgt $p = 1/2$
- Die erwartete Anzahl von Iterationen in Phase k ist 2

Analyse (2)

- Die Zufallsvariable X_k zählt die Anzahl der Vergleiche in Phase k
- In Phase k arbeitet der Algorithmus mit $\leq (3/4)^k n$ Elementen
- In Phase k wird im Erwartungswert zweimal partitioniert
- Zusammenfassend: $X_k \leq 2cn(3/4)^k$

$$
\begin{aligned}
\text{Erwartete Laufzeit} \quad &= \quad \sum_{k\geq 0} \mathbb{E}[X_k] \quad \leq \quad \sum_{k\geq 0} 2cn(3/4)^k \\
&= \quad 2cn \sum_{k\geq 0} (3/4)^k \quad = \quad 8cn \quad \in \quad O(n)
\end{aligned}
$$