Spotchecker

Übersicht:

- Ziel und Definition

- Beispiele
  1. Sort-Check
  2. Convex-Hull-Check
  3. Element-Distinctness

- Zusammenfassung

Ziel:

- Garantie einer Lösung, die nah am korrekten Ergebnis liegt
- Laufzeit asymp. schneller als Länge von Ein- und Ausgabe zusammen

Def.: $\epsilon-$Spotchecker

f: Funktion, die von Programm P berechnet werden soll

$\Delta(\cdot,\cdot)$: Abstandsfunktion

$\mathcal{C}$ ist ϵ-Spotchecker für f mit Abstandsfunktion Δ, wenn

1. Geg.: Programm P (soll f berechnen), ϵ, Eingabe x

   $\mathcal{C}$ gibt mit Wskt $\geq 3/4$ (bzgl. der internen Münzwürfe von $\mathcal{C}$) PASS aus, wenn $\Delta(\langle x, P(x)\rangle, \langle x, f(x)\rangle) = 0$ und FAIL, wenn für jede Eingabe y $\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle) > \epsilon$

2. Laufzeit von $\mathcal{C}$ ist $o(|x| + |f(x)|)$.

Beobachtung

1. Def. erfüllt starke Bedingung: Korrelation zw. Abstand Ausgabe zu richtigem Ergebnis und Wskt FAIL auszugeben

2. bei $O(\lg 1/\delta)$ Wdh: Konfidenz von $1 - \delta$

3. 3 mögliche Kombinationen für Länge der Eingabe / Länge der Ausgabe werden berücksichtigt:

   (a) Funktionen, bei denen Ein- und Ausgabe ähnliche Größe besitzen $|x| = \Theta(|f(x)|)$

   (b) Funktionen mit kleiner Ausgabe $|x| = o(|f(x)|)$

   (c) Funktionen mit großer Ausgabe $|f(x)| = o(|x|)$

1.Bsp: Sortieren

Ziel: Überprüfen der Programmausgabe eines Sortieralgorithmus auf Korrektheit;

Toleranz: Abweichung von $\leq \epsilon n$ Elementen von sortierter Liste

genauer: x, y Listen von Elementen,

$$\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle) = \begin{cases} \infty & x \neq y \text{ o. } |P(x)| \neq |f(y)| \\ \frac{\rho(P(x), f(y))}{|P(x)|} & \text{sonst} \end{cases}$$

Überprüfe Ergebnis in zwei Schritten:

1. Test, ob aufsteigende Subsequenz der Länge $(1-\epsilon)|x|$ in $P(x)$ vorhanden

2. Test, ob Mengen $P(x)$ und x einen Overlap $\geq (1-\epsilon)|x|$ besitzen

$\Rightarrow \Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle) \leq 2\epsilon$, falls $P(x)$ Test besteht
$\Rightarrow 2\epsilon-$Spotchecker

```
Procedure Sort-Check(A, ε)

repeat O(1/ε) times
choose i ∈_R [1, n]
for k ← 0...⌈lg i⌉ do
repeat O(1) times
choose j ∈_R [1, 2^k]
if(A[i − j] > A[i]) then return FAIL
for k ← 0...⌈lg(n − i)⌉ do
repeat O(1) times
choose j ∈_R [1, 2^k]
if (A[i] > A[i + j]) then return FAIL
return PASS
```

Theorem

Sort-Check(A, ϵ) läuft in $O((1/\epsilon) \lg n)$ und erfüllt:

1. A sortiert $\Rightarrow$ Sort-Check(A, ϵ)=PASS

2. A hat keine aufsteigende Untersequenz $\geq (1 - \epsilon)n$
   $\Rightarrow \rho > \epsilon n$
   $\Rightarrow \Delta > \epsilon n/n = \epsilon$
   $\Rightarrow$ Pr(Sort-Check(A, ϵ)=FAIL)$\geq$ 3/4

Hilfsdef.

$\forall i$: $\Gamma^{+}_{(t,t')}(i) = \{A[j] |\ A[j] > A[i],\ j > i,\ t \leq j \leq t'\}$

$\forall i$: $\Gamma^{-}_{(t,t')}(i) = \{A[j] |\ A[j] < A[i],\ j < i,\ t \leq j \leq t'\}$

schweres Element

$\forall k, 0 \leq k \leq \lg i$: $|\Gamma^{-}_{(i-2^k,i)}(i)| \geq \eta 2^k$ und

$\forall k, 0 \leq k \leq \lg(n-i)$: $|\Gamma^{+}_{(i,i+2^k)}(i)| \geq \eta 2^k$ mit $\eta = 3/4$

Lemma
Seien $A[i], A[j]$ schwere Elemente mit $i < j$. Dann $A[i] < A[j]$.

Idee:
Zeige: $\exists k$ mit $A[i] < A[k]$ und $A[k] < A[j]$,
$\Rightarrow$ Beh.

Beweis
Wähle m so, dass $2^m \leq (j-i)$, aber $(2m+1) \geq (j-i)$.
Sei $l = (j-i) - 2^m$ und $I = [j-2^m, i+2^m]$.
$\Rightarrow |I| = (i+2^m) - (j-2^m) + 1 = 2^m - l + 1$.
Idee:
Betrachte in I die Menge $G := \{A[k] | A[k] > A[i]\}$
und $K := \{A[k] | A[k] < A[j]\}$
$|G| + |K| > |I| \Rightarrow$(Schubfachprinzip) $\exists$ mind. ein k mit $A[k] \in I$
mit
$A[i] < A[k] < A[j]$

$A[i]$ schwer $\Rightarrow |G| \geq \eta 2^m - ((j - 2^m) - i) = \eta 2^m - l$
$A[j]$ schwer $\Rightarrow |K| \geq \eta 2^m - (j - (i + 2^m)) = \eta 2^m - l$

$\Rightarrow |G| + |K| = (\eta 2^m - l) + (\eta 2^m - l) \geq |I| = 2^m - l + 1$,
falls $l \leq 2^{m-1}$

Falls $l > 2^{m-1}$: Führe Beweis mit $m + 1$ statt mit m.

Beweis

Wenn A keine aufsteigende Untersequenz $\geq (1-\epsilon)n$ besitzt, dann Pr(Sort-Check(A, ϵ)=FAIL)$\geq$ 3/4.

Ann.: A besitze $< \epsilon n$ schwere Elemente $\Rightarrow$ A besitzt $> \epsilon n$ leichte Elemente Checker = PASS, wenn

1. nur schwere Elemente gezogen

2. leichtes Element für schweres Element gehalten

$\Rightarrow$ Chernoff-Schranken Pr(1.)$\leq \delta/2$
$\Rightarrow$ Chernoff-Schranken Pr(2.)$\leq \delta/2$
$\Rightarrow$ Gesamtirrtumswskt $< \delta$

Chernoff-Schranken

Geg. Bernoulli-Exp.

Erinnerung: Ziehen mit Zurücklegen ohne Reihenfolge

Trefferwskt p

Y_n zähle die Treffer

$\Rightarrow E[Y_n] = np$

1. $P(Y_n \geq (1+\epsilon)np \leq e^{-\epsilon^{2np/3}} 0 \leq \epsilon \leq 1$

2. $P(Y_n \geq (1-\epsilon)np \leq e^{-\epsilon^{2np/2}} 0 \leq \epsilon \leq 1$

Lemma

A, B zwei Listen mit $|A| = |B| = n$. A sortiert, aus unterschiedl. El. bestehend

$\Rightarrow \exists$ Spotchecker

- mit Laufzeit $O(\lg n)$
- A sortiert und $|A \cap B| = n \Rightarrow$ Ausgabe=PASS mit hoher Wskt
- $|A \cap B| \leq \epsilon n \Rightarrow$ Ausgabe=FAIL mit hoher Wskt

Beweis

Sei A sortiert.

choose $b \in B$ zufällig

search b in A mit bin. Suche

jeder Test: $O(\lg n)$, konst. Anz. Tests

$\Rightarrow$ $2\epsilon-$ Spotchecker für Sortierproblem

$O(\lg n)$**untere Schranke für Spotchecking Sorting**

gezeigt: $\exists 2\epsilon$ Spotchecker in $O(\lg n)$, n =Größe der Liste
Beh.:$\nexists$ Spotchecker mit Laufzeit $o(\lg n)$
Beweisidee: Ann.: $\mathcal{C}$ checkt Sortierproblem in $o(\lg n)$
Zeige: $\exists$ Eingabe y, die komplett sortiert ist mit
$Pr(\mathcal{C} =$FAIL$) = 1$ oder
$Pr(\mathcal{C} =$PASS$) = 1$ obwohl keine aufsteigende Sequenz der Länge $\Omega(n)$ ex.

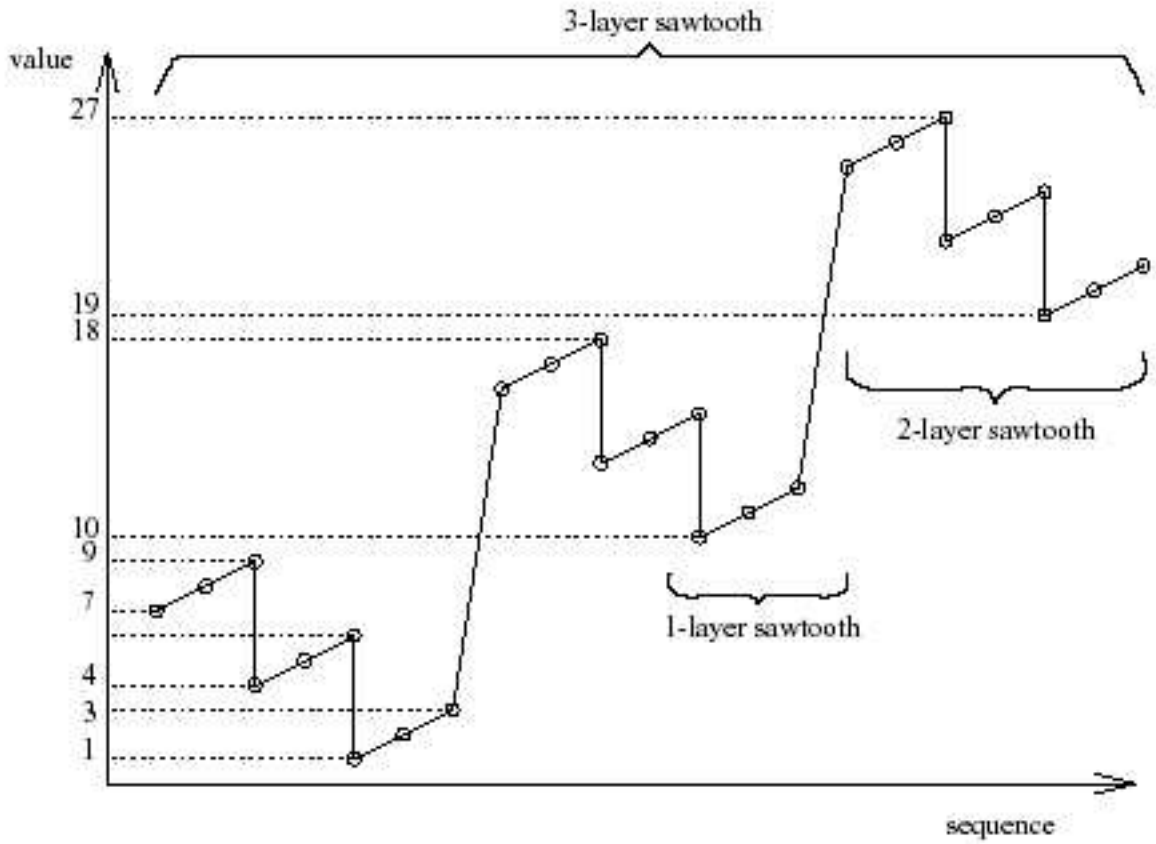


Figure 1: 3-layer saw-tooth: an $\mathrm{lst}_3(3,3,3)$ sequence.

Konvexe Hülle

Problem:

Programm P: Eingabe: M = Menge von n Punkten in euklid. Ebene

Ausgabe:$\langle x_0, x_1, ..., x_k \rangle$ Sequenz von Zeigern auf $k+1$ Punkte in M, die gegen Uhrzeigerrichtung eine konvexe Hülle ergeben sollen.

soll überprüft werden.

Idee für Spotchecker: Arbeite in zwei Phasen

1. Test auf Konvexität: Toleranz einer Abweichung von ϵk Elementen

2. Test, ob ausreichend Punkte in dem zurückgelieferten konvexen Polygon liegen (Hüllenbedingung)

Toleranz: Abweichung von ϵk Elementen.

Sei f die Funktion, die die konvexe Hülle einer Punktmenge korrekt bestimmt.
Def.

$$\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle) = \begin{cases} \infty & Y \neq P(X) \\ max\{d_{con}, d_{hull}\} & \text{sonst} \end{cases}$$

Zeige: $\exists\epsilon-$Spotchecker, mit Komplexität $O(\lg n)$

1. Spotchecker für Konvexität:
Geg.: CH eine Menge von $k+1$ Kanten, $e_i = (x_i, x_{i+1 \bmod k+1})$
Def.:$\angle e_0 = 0$
$\angle e_i =$ zw. e_0 und e_i eingeschlossener Winkel
$\forall \angle e_i$: $\angle e_i \in [0, 2\pi)$

Def.:
$0 \leq i, j \leq k$: $e_i \mathcal{R} e_j \Leftrightarrow$

1. $i < j$

2. $x_{i+1} = x_j$ und $0 < \angle e_j - \angle e_i < \pi$
   oder $0 < \angle e_j - \angle(x_{i+1}, x_j) < \pi$ und $0 < \angle(x_{i+1}, x_j) - \angle e_i < \pi$

$e_k \mathcal{R} e_0$, wenn $\angle e_k > \pi$

Lemma

Sei $S = \langle e_0, ..., e_l \rangle$ eine Sequenz von Kanten mit $e_i \mathcal{R} e_{i+1}$ $\forall 0 \leq i \leq l$. Sei C das Polygon, das sich ergibt, wenn man alle e_i aus S ggf. durch Einfügen neuer Kanten verbindet.

Beh.:

1. C besitzt keine Schleifen

2. C ist konvex

Procedure Convex-Check(CH, ϵ)

run Sort-Check($CH, \epsilon/2$), replacing $<$ with $\mathcal{R}$

if e_k or e_0 is not heavy return FAIL

if $\angle e_k \leq \pi$ return FAIL

return PASS

Theorem
Wenn CH Convex-Check(CH, ϵ) besteht, dann kann CH konvex gemacht werden, indem $\leq \epsilon k$ Knoten entfernt werden.

Test bestanden $\Rightarrow \exists 2$ disjunkte Subsequenzen von CH mit Gesamtlänge $\geq (1 - \epsilon)k \Rightarrow \leq \epsilon k$ Kanten entfernen
$\Rightarrow \leq \epsilon k$ Knoten entfernen

$\Rightarrow$ Convex-Check ist $\epsilon-$Spotchecker

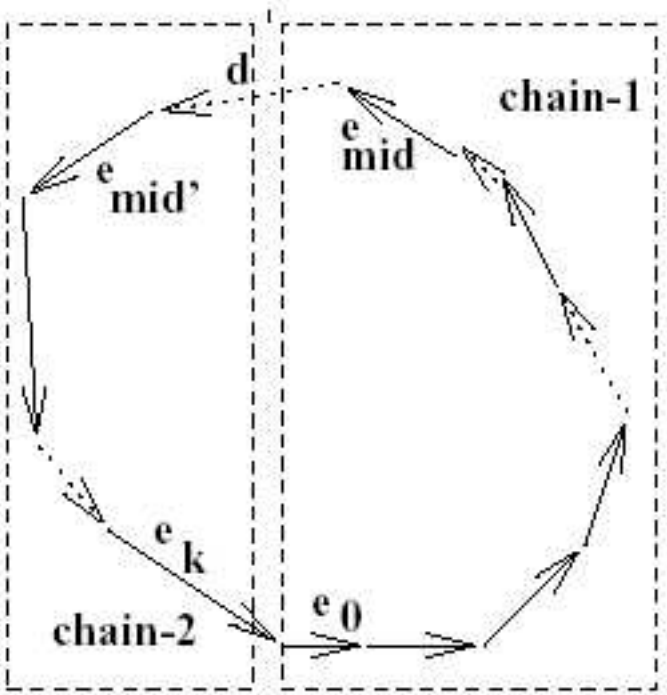


Figure 4: The two chains.

Lemma: eingeschränkte Transitivität

Wenn $e_i \mathcal{R} e_j$, $e_j \mathcal{R} e_k$ mit $\angle e_k - \angle e_i < \pi$
dann $e_i \mathcal{R} e_k$.

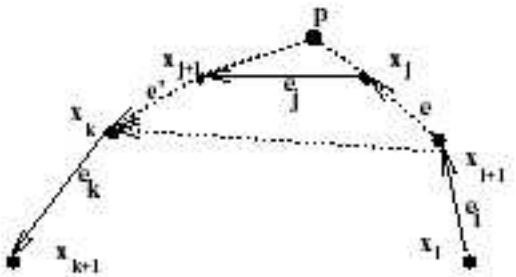


Figure 3: Transitivity under restricted conditions.

Beweis

Fall $x_{i+1} \neq x_j$ und $x_{j+1} \neq x_k$, e:=(x_{i+1}, x_j), e':=(x_{j+1}, x_k)

$\Rightarrow \angle e_i < \angle e < \angle e_j < \angle e' < \angle e_k$

$\Rightarrow \angle e' - \angle e < \pi \exists\ p$ so dass x_{i+1}, p, x_k bilden ein Dreieck

$\Rightarrow \angle e < \angle(x_{i+1}, x_k) < \angle e'$

$\Rightarrow e_i \mathcal{R} e_k$

Lemma

Wenn Convex-Check(CH, ϵ)=PASS, dann $e_{mid} \mathcal{R} e_{mid'}$.

Beweis

$e_{mid} \not\mathcal{R} e_{mid'}$

$\Rightarrow e_{mid}, e_{mid'}$ nicht adjazent

$\Rightarrow \exists e_r$ mit $e_{mid} \mathcal{R} e_r \mathcal{R} e_{mid'}$ und e_r kein schweres Element

$\Rightarrow \angle e_{mid'} - \angle e_{mid} > \pi$

d:=$(x_{mid+1}, x_{mid'})$

$e_{mid} \not\mathcal{R} e_{mid'} \Rightarrow$

1. $\angle d - \angle e_{mid} > \pi$ oder

2. $\angle e_{mid'} - \angle d > \pi$

Nimm o.B.d.A. (2.) an.

Für die Folge $e_0, ..., e_{mid}, d, e_{mid'}, ..., e_k$ gilt $e_i \mathcal{R} e_j$.
$\Rightarrow$ konvexes Polygon und aufsteigende Winkelfolge

(2.)$\Rightarrow$ $e_{mid'} > \pi$
$\Rightarrow \forall e_j$ mit $j > mid'$:$\angle e_j > \pi$ Widerspruch zu geschl. Polygon

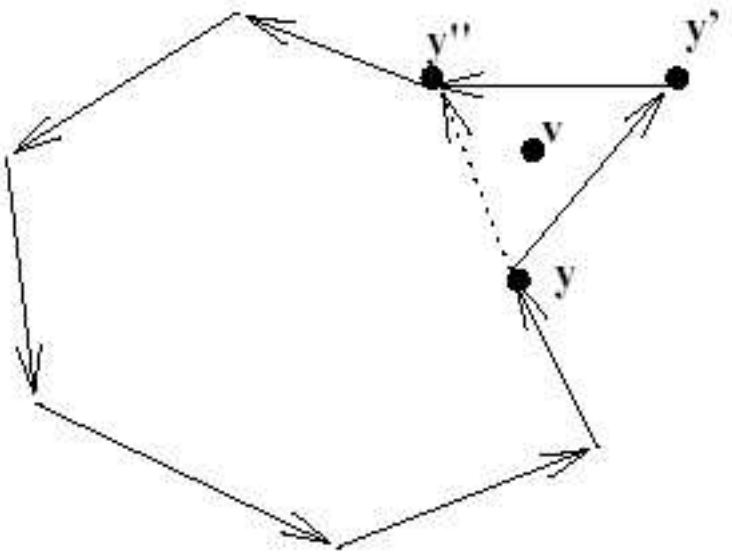
Hüllenbedingung

Idee: Wähle $O(1/\epsilon)$ Knoten, teste für jeden Knoten, ob er im konvexen Polygon liegt

Test:

$v \in CH$ innerhalb konvexem Polygon $\Rightarrow \exists$ Dreieck y, y', y'',
mit y' und y'' adjazent und v liegt im Dreieck
$\Rightarrow \angle(y, y') \leq \angle(v, y) \leq \angle(y'', y)$
Suche (y', y'') mit bin. Suche in $O(\lg n)$

Chernoff $\Rightarrow$ Fehlerwskt genügend klein


y''
y'
v
y

Element-Distinctness

Problem: Geg. Menge A,$|A| = n$

Frage: Alle Elemente unterschiedl.?

Toleranz: A hat $\geq (1 - \epsilon)n$ unterschiedl. Elemente

$f(A) = 0$, wenn A nicht n unterschiedl El enthält

$f(A) = 1$ sonst

$$\Delta(\langle x, P(x)\rangle, \langle y, f(y)\rangle) = \begin{cases} \infty & P(x) \neq f(y) \\ \frac{\rho(x,y)}{|x|} & \text{sonst} \end{cases}$$

mit $\rho(x, y) =$ Anz. El., die zu x hinzugefügt bzw. aus x gelöscht werden müssen um y zu erhalten.

Procedure Element-Distinctness-Check(A, ϵ)
choose random $\sqrt{(\epsilon n)}$ elements X from A
if X has any repeated elements **return** FAIL
return PASS

Element-Distinctness-Check(A, ϵ) ist $\epsilon-$Spotchecker mit Laufzeit $\tilde{O}(\sqrt{(\epsilon n)})$

Zusammenfassung

Effizienz von Spotcheckern wg. punktweisen Überprüfung der Programmausgabe

Vorteil:

1. Garantie, dass Programmausgabe nah am richtigen Ergebnis liegt

2. Laufzeit kürzer als o(Eingabe + Ausgabe)

Nachteil: "nah" am richtigen Ergebnis abh von Δ