

Effiziente Algorithmen für das Konvexe-Hülle-Problem (in 2D)

Jens Jägersküpfer

Vorlesung
Effiziente Algorithmen und Komplexitätstheorie
an der TU Dortmund im SoSe 2008

Problemstellung

Problem

Berechne die **Eckpunkte der konvexen Hülle** einer n -elementigen Menge $P \subset \mathbb{R}^2$ von Punkten (kartesische Koordinaten).

konvex Eine Menge $M \subseteq \mathbb{R}^2$ heißt konvex, wenn für alle $a, b \in M$ auch alle Punkte des **offenen Geradensegments**
 $\underline{(a, b)} := \{p \in \mathbb{R}^2 \mid \lambda a + (1 - \lambda)b, \lambda \in (0, 1)\}$ in M liegen.

konv. Hülle $H \subset \mathbb{R}^2$ einer Menge $M \subset \mathbb{R}^2$ ist der Schnitt aller konvexen Mengen, die M enthalten, d.h. $H = \bigcap_i K_i$ mit $K_i \subset \mathbb{R}^2$ konvex und $M \subseteq K_i$.
(H ist „die kleinste“ konvexe Obermenge von M .)

Eckpunkt Ein Punkt p einer konvexen Menge K ist ein Eckpunkt (Extrempunkt) von K , wenn es **keine** zwei Punkte $a, b \in K$ gibt, sodass $p \in \underline{(a, b)}$.

Motivation

konvexe Hülle in 2D „entspricht“ Sortieren in 2D

→ grundlegendes Problem

- Robotik: *motion planning* zur Umfahrung von Hindernissen
z.B. bei automatisierter Lagerhaltung
- Lassen sich zwei Punktmengen (aus \mathbb{R}^2) linear separieren?
gdw. Schnitt der konvexen Hüllen leer
- Statistik: Extrempunkte einer Menge von *samples* (aus \mathbb{R}^2)

Beobachtungen

- Die Menge E der Extrempunkte e_1, \dots, e_h der konvexen Hülle H ist eine Teilmenge der endlichen Punktmenge $P \subset \mathbb{R}^2$.
- Ferner ist die konv. Hülle H eine geschlossene Menge und es gibt eine Permutation π über $(1, \dots, h)$, sodass die Vereinigung $\bigcup_{i=1}^{h-1} \overline{[e_{\pi(i)}, e_{\pi(i+1)}]} \cup \overline{[e_{\pi(h)}, e_{\pi(1)}]}$ von Geradensegmenten gleich dem Rand von H ist.
- Der Rand von H ist ein einfaches Polygon, d.h. keine zwei der h o.g. Geradensegmente schneiden sich.

Problem Bestimme die h Punkte aus der n -elementigen Punktmenge $P \subset \mathbb{R}^2$, die auch Eckpunkte der konvexen Hülle von P sind.

P.-Variante Bestimme neben dieser Menge $E (\subseteq P)$ von Eckpunkten der konvexen Hülle von P auch eine Permutation mit o.g. Eigenschaft.

Annahmen

- n , die Anzahl Punkt in P , ist ≥ 3
- P ist in „allgemeiner Lage“ (*general position*), d.h. es gibt keine drei Punkte in P , die kollinear sind (d.h. sie liegen auf einer gemeinsamen Geraden).
- Insbesondere also Punkte aus P paarweise verschieden

Achtung

Das Anpassen von Algorithmen, die „ P in allgemeiner Lage“ voraussetzen, für die korrekte Verarbeitung beliebiger Punktmengen kann durchaus nicht-trivial sein.

Für die im Folgenden betrachteten Algorithmen ist das aber leicht möglich (→Übungsaufgabe!)

Trivialer Algorithmus

Da wir wissen, dass die Eckpunkte der konvexen Hülle H einer n -elementigen Menge $P \subset \mathbb{R}^2$ alle zu P gehören, müssen wir lediglich die „Nichteckpunkte“ aus P streichen.

Trivialer Algo. (Ausschlussverfahren)

- 1 Teste für jeden der n Punkte $p \in P$, ob es drei andere Punkte in P gibt, sodass p innerhalb (oder auf dem Rand) des durch sie definierten Dreiecks liegt. Wenn ja, markiere p .
- 2 Gib alle nicht markierten Punkt (aus P) aus.

Es werden $n \cdot \binom{n-1}{3} = \Theta(n^4)$ Tests benötigt.

Was für Tests? Wie testet man das? Wie teuer ist ein Test?

Test, ob p innerhalb (a, b, c, a)

Gegeben: vier Punkt $p, a, b, c \in \mathbb{R}^2$ in allgemeiner Lage
(zumindest a, b, c nicht kollinear)

Beobachtung

Der Punkt p liegt innerhalb des einfachen Polygons (a, b, c, a)
(ein Dreieck), wenn p für alle drei orientierten Geraden $\overrightarrow{a, b}, \overrightarrow{b, c}, \overrightarrow{c, a}$
auf derselben Seite liegt.

Für einen Vektor $v \in \mathbb{R}^2 \setminus \{0\}$ bezeichne $v^* := (-Y(v), X(v)) \in \mathbb{R}^2$
einen (bestimmten) Normalenvektor von/zu v .
(Die Abbildungen X und Y liefern die x - bzw. y -Koordinate.)

Für das Skalarprodukt/innere Produkt gilt $v \times v^* = 0$ (Orthogonalität)

Ferner gilt für $p \in \mathbb{R}^2$: $p \times v^* = 0 \implies p \in \overline{0, v}$

Falls $p \times v^* > 0$, so liegt p **links** von $\overrightarrow{0, v}$

Falls $p \times v^* < 0$, so liegt p **rechts** von $\overrightarrow{0, v}$

Links-/Rechtsknick

Definition

Das Punkte-Tripel (a, b, c) bildet
einen Linksknick, falls c links von $\overrightarrow{a, b}$ liegt (d.h. $(c - a) \times (b - a)^* > 0$)
einen Rechtsknick, falls c rechts von $\overrightarrow{a, b}$ (d.h. $(c - a) \times (b - a)^* < 0$)
(kollinear, falls $(c - a) \times (b - a)^* = 0$, was hier ausgeschlossen ist)

Die Knickbestimmung benötigt (höchstens) $2 \cdot 2$ Subtraktionen,
2 Multiplikationen, 1 Addition, **1 Vergleich**

Somit (unter der Annahme „allgemeine Lage“):

Der Punkt p liegt innerhalb des Dreiecks $(a, b, c, a) \iff$
 $\text{Knick}(a, b, p) = \text{Knick}(b, c, p) = \text{Knick}(c, a, p)$

Knick-Bestimmung bei „Konvexe Hülle“ $\hat{=}$ Vergleich beim Sortieren

Naiver Algorithmus

Unter der Voraussetzung „ P in allgemeiner Lage“ gilt:

Beobachtung

Falls für zwei unterschiedliche Punkte $a, b \in P$ alle anderen $n - 2$ Punkte auf derselben Seite von $\overrightarrow{a, b}$ liegen, so sind a und b Eckpunkte der konvexen Hülle von P .

Naiver Algorithmus („Positiveliste“)

- ① Teste für alle $\{a, b\} \subset P$ mit $a \neq b$, (und a oder b nicht markiert), ob alle anderen $n - 2$ Punkte aus $P \setminus \{a, b\}$ auf derselben Seite von $\overrightarrow{a, b}$ liegen. Falls ja, markiere a sowie b .
- ② Gib alle markierten Punkte aus.

Im *worst case* werden $\binom{n}{2} \cdot (n - 2) = \Theta(n^3)$ Tests durchgeführt.

Beobachtung (auch Jarvis 1973)

- Warum für alle Paare von Punkten aus P **naiv, d.h. in beliebiger Reihenfolge**, testen, ob sie ein Randsegment der konvexen Hülle definieren?
 - Sobald man einen Punkt als Eckpunkte markiert hat, weiß man, dass dieser (neben dem ebenfalls markierten Punkt) noch genau einen weiteren Nachbareckpunkt auf (dem Rand) der konvexen Hülle hat.
- ⇒ Wenn ein Eckpunkt gefunden, **suche gezielt** nach dem nächsten (d.h. benachbarten) Eckpunkt der konvexen Hülle.
- Wenn man von einem Eckpunkt die restlichen $n - 1$ Punkte betrachtet, werden der am weitesten rechts und der am weitesten links stehende Punkt gesucht.
 - Jarvis' Idee: Aufbau der Menge von Eckpunkten entgegen dem Uhrzeigersinn ihres Vorkommens auf der konvexen Hülle.

Algorithmus von Jarvis (*Jarvis' march, gift wrapping*)

Voraussetzung: $P = \{p_1, \dots, p_n\}$ in allgemeiner Lage

Algorithmus: Jarvis' march

- ① Bestimme i als Index eines Eckpunkts, z.B. minimale y -Koord.
- ② Setze $e := i$ sowie $E := \emptyset$.
- ③ WHILE $i \notin E$ DO
 - ① IF $e = 1$ THEN $r := 2$ ELSE $r := 1$
 - ② FOR $k \in \{1, \dots, n\} \setminus \{e, r\}$ IF (p_e, p_r, p_k) Rechtsknick THEN $r := k$.
 - ③ Setze $E := E \cup \{r\}$ sowie $e := r$.
- ④ Gib E als konvexe Hülle aus

In der WHILE-Schleife wird für den Eckpunkt e der Punkt r ($\neq e$) bestimmt, sodass alle anderen $n - 2$ Punkte links von $\overrightarrow{e, r}$ liegen. Folglich ist r der nächste Eckpunkt auf der konvexen Hülle.

Laufzeit von *Jarvis' march* und Anmerkungen

Laufzeit:

- Offensichtlich werden pro WHILE-Schleife $n - 2$ Tests (auf Rechtsknick) durchgeführt.
- In jedem Durchlauf wird ein neuer (nämlich der nächste) Eckpunkt in E aufgenommen, sodass maximal n Durchläufe nötig sind.
- Im *worst case* werden also $(n - 2) \cdot n = \Theta(n^2)$ Test benötigt.

Anmerkungen:

- Tatsächlich werden immer $(n - 2) \cdot h$ Tests benötigt! ($h = \#E$)
- Laufzeit ist eingabe- und **ausgabesensitiv**
- Bei sehr wenigen Eckpunkten gut ...
- ... z.B. bei zufälligen Punktmengen (Statistik), für die sehr wenige Eckpunkte erwarten werden.

Zwischenstand

- trivialer Algorithmus („Ausschlussverfahren“): $\Theta(n^4)$
- naiver Algorithmus („Positivliste“): $O(n^3)$
- Jarvis' march: $\Theta(h \cdot n) = O(n^2)$

Diese Schranken beziehen sich auf die Anzahl „Knick-Operationen“

Da eine „Knick-Operation“ mit 2 Multiplikationen, 1 Vergleich, 5 Additionen entschieden werden kann, gelten die o.g. oberen Schranken auch für #Multiplikationen, #Vergleiche, #Additionen.

Kann Division helfen, sodass $\#Mults + \#Divs = o(n^2)$ im *worst case*?

Algorithmus: *Graham's scan*

verfolgt Back-Tracking-Ansatz und verwendet „Sortieren“ als Unterprogramm

Graham's scan

- 1 Bestimme den Punkt $p_i \in P$ mit minimaler y -Koordinate
- 2 Verschiebe den Ursprung des Koordinatensystems nach p_i
- 3 Sortiere $P \setminus \{p_i\}$ nach dem Azimut in Polarkoordinaten (Winkel).
Durch Umnummerierung: doppelt verkettete Liste
 $p_1 \rightleftarrows \dots \rightleftarrows p_n \rightleftarrows p_1$, die ein einfaches Polygon bildet,
wobei HEAD:= p_1 Eckpunkt (nämlich ex p_i).
- 4 a:=HEAD, b:=SUCC(a), c:=SUCC(b).
- 5 WHILE b \neq HEAD
IF (a,b,c) Linksknick THEN { a:=b, b:=c, c:=SUCC(c) }
ELSE { DELETE(b) aus Liste, b:=a, a:=PREC(a) }
- 6 Gib die (verbliebenen) Listenelemente aus (jew. verschoben).

Analyse von *Graham's scan*

Korrektheit:

- Zunächst wird ein **einfaches** Polygon über P bestimmt.
- Dann werden nach und nach **innere** Punkte gelöscht, sodass alle Tripel (a,b,c) von aufeinander folgenden Punkten (Gegenuhrzeigersinn) des einfachen Rest-Polygons jeweils ein Linksknick. → konvexes Polygon

Laufzeit:

- ① $n - 1$ Vergleiche von y -Koordinaten (bubble)
- ② $2n$ Additionen für Translation
- ③ $\Theta(n \log n)$ Vergleiche, wenn wir die Winkel hätten
- ⑤ WHILE-Schleife (je eine Knick-Test und ggf. DELETE) wird keine $2n$ mal durchlaufen, sodass $O(n)$ Operationen (Mult., Vgl., Add.)
Beweis durch Widerspruch: WHILE-Schleife $2n$ mal durchlaufen \implies b erreicht zuvor HEAD oder mehr als n Konten entfernt.
- ⑥ ggf. $2h \leq 2n$ Additionen für Re-Translation der Punkte

Analyse von *Graham's scan* (2)

Das Sortieren nach Azimut dominiert: $\Theta(n \log n)$ Vergleiche

Sortierung der Punkte nach Azimut:

- Trigonometrische Funktionen teuer und „ungenau“
- Sortierung nach Quadrant und Steigung der Geraden $\overline{0, p_i}$ reicht aus, d.h. n Divisionen

Insgesamt also $\Theta(n \log n)$ Vergleiche und $O(n)$ Mults, Divs, Adds.

Anmerkung:

Da Laufzeit vom Sortieren dominiert, Schranke nicht ausgabesensitiv.

Kann es noch schneller gehen?

Komplexität von „Konvexe Hülle“

Reduziere Sortieren auf die KH-Variante, bei der Eckpunkt in der Reihenfolge ihres Vorkommens auf der konv. Hülle ausgegeben werden

Linearzeit-Reduktion: $A \leq_{\text{lin}} B : \iff$

\exists Programm für A, das auf der Eingabe (a_1, \dots, a_n) wie folgt arbeitet:

- 1 Berechne in Zeit $O(n)$ aus der A-Eingabe eine B-Eingabe (b_1, \dots, b_m) , folglich $m = O(n)$.
- 2 Rufe eine Programm für B auf der Eingabe (b_1, \dots, b_m) auf, Ergebnis (c_1, \dots, c_ℓ) .
- 3 Berechne in Zeit $O(n)$ aus der B-Ausgabe (c_1, \dots, c_ℓ) das Ergebnis für A (bei Eingabe (a_1, \dots, a_n))

Dann gilt: $T_A(n) \leq T_B(m) + O(n)$ mit $m = O(n)$.

- \exists Linearzeit-Algo. für B \implies auch A in Linearzeit lösbar
- Komplexität von A superlinear $\implies \nexists$ Linearzeit-Algo. für B

Anmerkungen zur Komplexität von KH

Wenn man weiß, dass Sortieren $\Omega(n \log n)$ Vergleiche benötigt, so bekommt man eine $\Omega(n \log n)$ Schranke für die #Vergleiche, die zur Lösung der KH-Variante nötig sind, wie folgt (Reduktion):

- Eingabemenge $\{a_1, \dots, a_n\}$, $a_i \in \mathbb{R}$ paarweise verschieden, die sortiert werden soll, abbilden auf $P := \{(a_1, a_1^2), \dots, (a_n, a_n^2)\}$ (kostet n Multiplikationen und Verdoppelung der Eingabeläge).
 \implies Offensichtlich sind alle n Punkte Eckpunkte der $KH(P)$.
- In der sortierten Ausgabe der KH-Eckpunkte den Punkt mit minimaler x -Koordinate bestimmen ($O(n)$ Vergleiche), um dann die x -Koordinaten der Eckpunkte sortiert auszugeben.

Problem: Lässt sich KH überhaupt nur mit Vergleichen lösen?

Was, wenn Sortieren mit anderen Operationen in $o(n \log n)$?

Achtung bei Reduktionen

Bei (Linearzeit-)Reduktionen ist zu Beachten, dass die Rechner-/Berechnungsmodelle „kompatibel“ sind.

Komplexität von Sortieren und Konvexe Hülle

„Zum Glück“:

Sortieren benötigt nicht nur im Entscheidungsbaum-Modell $\Omega(n \log n)$ Operationen (nämlich Vergleiche), sondern auch im Modell „algebraischer Berechnungsbaum“ mit den Operationen Vergleich, Mult, Div, Add, Wurzel.

⇒ Unsere obige lineare Reduktion liefert $\Omega(n \log n)$ -Schranke für die KH-Variante bez. algebraischen Berechnungsbäumen mit o.g. Operationen.

Ist das Bestimmen nur der Menge von Eckpunkten (unsortiert) der konvexen Hülle einer endlichen Punktmenge einfacher?

Nein, auch Komplexität $\Omega(n \log n)$ (ein wahrlich „tiefes“ Resultat)

⇒ *Graham's scan* ist ein worst-case-optimaler Algorithmus: Mit $O(n \log n)$ Operationen (im o.g. Modell) werden die Eckpunkte der konvexen Hülle bestimmt („sogar“ sortiert).

„Komplexität und Praxis“

Was heißt „Das Konvexe-Hülle-Problem hat Komplexität $\Omega(n \log n)$ “?

- \forall KH-Aglo. A und $n \in \mathbb{N} \quad \exists P \subset \mathbb{R}^2$ mit $P = n: T_A(P) = \Omega(n \log n)$
- Anders formuliert: $\forall n \in \mathbb{N} \quad \max_{P \subset \mathbb{R}^2: \#P=n} T_A(P) = \Omega(n \log n)$

Was, wenn in der Praxis diese „speziellen“ P nicht vorkommen?

Was, wenn die Eingabe keine Worst-Case-Eingabe ist?

- Für Punktfolgen mit $h = o(\log n)$ ist die $O(nh)$ -Laufzeit von *Jarvis' march* $o(n \log n)$
- Kann man schon für $h = o(n)$ Laufzeit $o(n \log n)$ erreichen?
- Kirkpatrick/Seidel: KH mit $O(n \log h)$ Operationen bestimmen ($h = \#$ Eckpunkte der konvexen Hülle)

Effiziente Algorithmen für das Konvexe-Hülle-Problem (in 2D) Teil 2

Jens Jägersküpfer

Vorlesung
Effiziente Algorithmen und Komplexitätstheorie
an der TU Dortmund im SoSe 2008

Was bisher geschah

Vier **Algorithmen** (o.k., zwei „echte“) für das Konvexe-Hülle-Problem in 2D, wobei $n = \#$ Punkte und $h = \#$ Eckpunkte der konvexen Hülle

- Trivialer Algo (Ausschluss von Nichteckpunkten) $\Theta(n^4)$
- Naiver Algo (Positivliste Geradensegmente) $O(n^3)$
- *Jarvis' march* (nächsten Eckpunkt suchen) $\Theta(nh) = O(n^2)$
- *Graham's scan* (einfaches Polygon über Sortieren, dann Nicht-KH-Eckpunkte löschen (*back tracking*)) $\Theta(n \log n)$

Komplexität des Konvexe-Hülle-Problems in 2D:

- $\Omega(n \log n)$ Operationen im Berechnungsmodell „algebraischer Berechnungsbaum“ mit den
 - Operationen: Vergleich, Mult, Div, Add/Sub (und Wurzel)
- ⇒ *Graham's scan* ist ein worst-case-optimaler Algorithmus

Ausgabesensitivität

Die Laufzeit von Jarvis' march hängt nicht nur von $n = \#P$ ab, sondern „stark“ von $h = \#\text{Eckpunkte in } KH(P)$.

z.B. für $h = O(1)$ ergibt sich die Laufzeit $O(n)$

Widerspruch zur Komplexität von KH?

Nein: Worst-Case-Modell (bisher) wie folgt:

„Betrachte unter allen $P \subset \mathbb{R}^2$ mit $\#P = n$ eine Instanz, die zu maximaler Laufzeit führt.“

Feineres – nämlich ausgabesensitives – Worst-Case-Modell:

„Betrachte unter allen $P \subset \mathbb{R}^2$ mit $\#P = n$ und $KH(P)$ hat genau h Eckpunkte eine Instanz, die zu maximaler Laufzeit führt.“

Die Menge der Instanzen wird nicht nur nach Eingabegröße partitioniert, sondern zusätzlich nach Ausgabegröße

„Ausgabesensitive Komplexität“ von KH

Worst-Case-Instanz hängt vom Algorithmus ab

\forall KH-Algo. $A \exists P \subset \mathbb{R}^2$ mit $\#P = n$ und $\#KH(P) = h : T_A(P) = \Omega(n \log h)$

Kirkpatrick/Seidel (SIAM J. Computing 1986)

- Entscheidungsvariante $\#$ Eckpunkte von $KH(P) \stackrel{?}{=} h$
 - Berechnungsmodell: „algebraischer Entscheidungsbaum“ mit „Polynom-Tests“ vom Grad $O(1)$
 - d.h. in jedem (Entscheidungsbaum-)Knoten wird das Signum des Werts eines (u.U. multivariaten) Polynoms bestimmt (+, 0, -).
 - „Unser Knick-Test“ ist ein quadratischer Test.
- ⇒ der Triviale Algo, der naive Algo. sowie *Jarvis' march* passen offensichtlich in das Modell „algebraischer Entscheidungsbaum mit quadratischen Tests“

D&C

Nein, nicht D&G, sondern: *divide & conquer*

Noch ein $O(n \log n)$ -Algorithmus
(Laufzeitschranke wie bei *Graham's scan*)

Grund: Vorbereitung eines $O(n \log h)$ -Algorithmus

D&C-Idee:

- 1 Berechne (rekursiv) die konvexen Hüllen der beiden Punktmenge P_1 und P_2 mit $P = P_1 \cup P_2$ (und $P_1 \cap P_2 = \emptyset$)
- 2 Berechne aus $KH(P_1)$ und $KH(P_2)$ schließlich $KH(P)$

Damit sich die Laufzeit $O(n \log n)$ ergibt:

- „Lehre“ aus Quick-Sort-Analyse: P_1 und P_2 möglichst gleich groß
- *Merge* von $KH(P_1)$ und $KH(P_2)$ zu $KH(P)$ mit $O(n)$ Operationen

D&C-Algorithmus: Anmerkungen und Annahmen

- Es handelt sich **nicht** um den Algorithmus „*quick hull*“
 - Eingabe: Punktmenge $P \subset \mathbb{R}^2$ in allgemeiner Lage
 - $n := \#P \geq 3$
 - und alle n X -Koordinaten paarweise verschieden (kann ebenso „gefixt“ werden wie Ann. „allg. Lage“)
- ⇒ Für gegebene Punkte(teil)menge sind Minimum und Maximum sowie Median bez. X -Koordinate eindeutig
- Median-Bestimmung in Linearzeit möglich
 - Beschränkung auf Berechnung der **oberen Hülle**: der obere Teil der KH, der die beiden Punkte aus P mit maximaler bzw. minimaler X -Koordinate verbindet
 - untere Hülle kann analog bestimmt werden (im hiesigen Spezialfall sogar gleichzeitig/parallel)

D&C-Algorithmus

ein D&C-Algorithmus für „Obere Hülle“ (OH)

- 1 Falls $n \leq 2$ gib die Punkte v.r.n.l. aus. ENDE.
- 2 Bestimme (mithilfe des Medians der X -Koordinaten) die P -Partition $\{P_L, P_R\}$, sodass P_L links von P_R und $|\#P_L - \#P_R| \leq 1$.
- 3 Bestimme rekursiv die OHs von P_L und P_R .
- 4 Verbinde Endpunkt von $\text{OH}(P_R)$ mit Anfangspunkt von $\text{OH}(P_L)$
- 5 Entferne aus diesem Polygonzug alle nicht zu $\text{OH}(P)$ gehörenden Punkte à la Graham
- 6 Gib den so „bereinigten“ Polygonzug v.r.n.l. aus

Anmerkung: Anweisung 4 kann in Anw. 5 integriert werden.

D&C-Algorithmus: Analyse

Korrektheit:

- Kein Punkt oberhalb des in Anw. 4 erzeugten Polygonzuges
- Korrektheit folgt somit aus der von *Graham's scan* (Anw. 5)

Laufzeit:

$T(n) \leq c = O(1)$ für $n \in \{1, 2\}$ klar.

Für $n = 2^i$ mit $i \in \{2, 3, \dots\}$ gilt $T(n) \leq c \cdot n \cdot \text{ld } n = O(n \log n)$:

$$\begin{aligned}
 T(n) &\leq 2 \cdot T(n/2) + c \cdot n \\
 &\leq 2 \cdot (c \cdot (n/2) \cdot \text{ld}(n/2)) + c \cdot n \\
 &= c \cdot n \cdot (\text{ld}(n) - 1) + c \cdot n = c \cdot n \cdot \text{ld } n
 \end{aligned}$$

- Konstante c so groß gewählt, dass Median-Berechnung, „Polygonzug-Bereinigung“ sowie Basis-Fälle ($n \leq 2$) abgedeckt
- n keine Zweierpotenz: $\text{ld} \rightarrow \log_{3/2}$, da Verhältnis 1:2 *worst case*

Anmerkungen zum D&C-Algorithmus

- Laufzeit „kaum“ ausgabesensitiv, d.h. „kaum“ von h abhängig
 - Rekursionstiefe immer $\log n$
- ⇒ Wie könnte man Rekursionstiefe $\log h$ erreichen?

In Anw. 5 wird „**die Brücke**“ zwischen P_L und P_R konstruiert, das **Liniensegment** $[\rho_L, \rho_R]$ der **OH**(P) mit $\rho_L \in P_L$ und $\rho_R \in P_R$

Kirkpatrick/Seidel: Was, wenn man diese Brücke(npunkte) vor der Rekursion in Zeit $O(n)$ bestimmen könnte?

- Zusammensetzen der OH trivial
- Man muss sich dann (rekursiv) nur noch kümmern um $P'_L := \{p \in P_L \mid X(p) \leq X(\rho_L)\}$ und $P'_R := \{p \in P_R \mid X(p) \geq X(\rho_R)\}$
- dabei können P'_L und P'_R viel kleiner sein als P_L bzw. P_R
- alle Punkte zwischen ρ_L und ρ_R „abgehakt“/nicht in Rekursion

Algorithmus nach Kirkpatrick/Seidel (K/S)

Dieser Algorithmus gibt – unter unseren Annahmen – die Knoten der oberen Hülle v.r.n.l. aus

Prinzip „*marriage before conquest*“ statt „*divide and conquer*“

$P'_L := \{p \in P_L \mid X(p) \leq X(\rho_L)\}$ und $P'_R := \{p \in P_R \mid X(p) \geq X(\rho_R)\}$

Algorithmus für „Obere Hülle“ nach Kirkpatrick/Seidel

- 1 Bestimme (mithilfe des Medians der X -Koordinaten) die P -Partition $\{P_L, P_R\}$, sodass P_L links von P_R und $|\#P_L - \#P_R| \leq 1$.
- 2 Bestimme die Brücke(npunkte): $\rho_L \in P_L$ und $\rho_R \in P_R$.
- 3 IF in P noch Punkte rechts von ρ_R THEN REKURSIV für P'_R
ELSE OUTPUT ρ_R .
- 4 IF in P noch Punkte links von ρ_L THEN REKURSIV für P'_L
ELSE OUTPUT ρ_L .

K/S-Algorithmus: Korrektheit

- Das in Anw. 1 bestimmte Gradensegment (p_L, p_R) gehört zur OH, da alle anderen anderen Punkte aus P unterhalb von $\overline{p_L, p_R}$
 - p_R wird zwar nur ausgegeben, falls in P keine P'e rechts von p_R
 - andernfalls aber ist p_R der **linkeste** Punkt in P'_R (welche noch mind. einen Punkt rechts von p_R enthält) und wird daher eine Rekursionstiefe weiter ausgegeben, denn analog gilt:
 - p_L wird zwar nur ausgegeben, falls in P keine P'kte links von p_L
 - andernfalls aber ist p_L der **rechtteste** Punkt in P'_L (welche noch mindestens einen Punkt links von p_L enthält) und wird daher eine Rekursionstiefe weiter ausgegeben.
- ⇒ Jedenfalls wird p_R direkt vor p_L ausgegeben.
- Ferner: Da der direkt vor p_R ausgegebene Punkt p' (wie oben bemerkt) unterhalb von $\overline{p_L, p_R}$ liegt, ist (p', p_R, p_L) ein Linksknick
 - da der direkt nach p_L ausgegebene Punkt p'' ebenso unterhalb von $\overline{p_L, p_R}$ liegt, ist auch (p_R, p_L, p'') ein Linksknick
- ⇒ Ausgabe ist ein Polygonzug vom rechtesten zum linkesten Punkt in P , der insb. ausschließlich Linksknicke enthält (oder 2 Punkte)

K/S-Algorithmus: Laufzeit

Hier $h := \# \text{Knoten in der OH}(P)$, wg. unserer Annahmen $h \geq 2$

$T(n, h) = c \cdot n$ für $h = 2$, da insb. Brücke in $O(n)$ berechenbar (Ann.)

Vermutung $T(n, h) \leq c \cdot n \cdot \text{ld } h$ wg. Vermutung Rekursionstiefe $\log h$

Für $h \geq 3$ und n Zweierpotenz gilt mit $h_L + h_R = h$

$$\begin{aligned}
 T(n, h) &\leq c \cdot n + \underbrace{T(\#P'_L, h_L)} + \underbrace{T(\#P'_R, h_R)} \\
 &\leq c \cdot n + c \cdot (n/2) \cdot \text{ld } h_L + c \cdot (n/2) \cdot \text{ld } h_R \\
 &= c \cdot n + c \cdot n \cdot (\text{ld } h_L + \text{ld } h_R) / 2 \\
 &= c \cdot n + c \cdot n \cdot \text{ld}(h_L \cdot h_R) / 2 \\
 (\text{da } h_L + h_R = h) &\leq c \cdot n + c \cdot n \cdot \text{ld}((h/2)^2) / 2 \\
 &= c \cdot n + c \cdot n \cdot (\text{ld}(h) - 1) = c \cdot n \cdot \text{ld } h
 \end{aligned}$$

Annahmen können überwunden werden, dass $O(n \log h)$ gültig bleibt.

Berechnung der Brücke in $O(n)$

Idee zum Finden der Brücke(nknoten) $p_L \in P_L$ und $p_R \in P_R$ für die P -Partition $\{P_L, P_R\}$ mit P_L links von P_R – unter Ann. P in allg. Lage

- Falls $\#P_L = 1 = \#P_R$, dann gib P_R und P_L aus. Sonst:
- Schätze die Steigung s der Brücke
- Finde den Punkt $p_L \in P_L$, sodass kein anderer Punkt aus P_L über der Geraden L durch p_L mit Steigung s liegt (eine Tangente)
- Teste für jeden Punkt $p \in P_R$, ob er auf, über oder unter L liegt.
- Falls einer (dies sei p_R) auf L liegt und alle anderen darunter, gib p_R, p_L als Brücke aus. Sonst:
- Falls alle $p \in P_R$ unter L , so war s zu groß geschätzt
- Falls ein $p \in P_R$ über L , so war s zu klein geschätzt

Was, wenn Schätzung der Brücke falsch?

Dann hat man zwar die Brücke nicht gefunden, aber man kann Punkte aus P als Brückenpunkte ausschließen

Dazu betrachte für ein Paar Punkte $a, b \in P$, sodass a links von b , die Steigung s_{ab} von $\overline{a, b}$.

Bezeichne s^* die tatsächliche Steigung der Brücke. Dann gilt:

$s_{ab} \leq s < s^*$ Es könnte zwar $a = p_L$ sein. Jedoch liegt b jedenfalls zu tief, sodass $b = p_R$ ausgeschlossen.

$s_{ab} \geq s > s^*$ Es könnte zwar $b = p_R$ sein. Jedoch liegt a jedenfalls zu tief, sodass $a = p_L$ ausgeschlossen

anderenfalls weiß man über $s_{ab} \leftrightarrow s^*$ nichts, sodass a wie auch b Brückenknoten sein könnten

⇒ Wir sollten die Schätzung s so wählen, dass wir möglichst viele Punkte als Brückenpunkte ausschließen können.

Passende Schätzung der Brückensteigung

Wir wollen für möglichst viele Punktepaare einen der beiden Punkte als Brückenpunkt ausschließen können. Also:

- Gruppiere P zunächst in Paare von Punkten (ggf. einer übrig)
 - Berechne dann für jedes Paar (a, b) , wobei a links von b sei, die Steigung von $\overline{a, b}$
 - Wähle den Median der $\lfloor n/2 \rfloor$ Steigungen als Schätzung s .
 - Falls $s = s^*$, so wird die Brücke gefunden (siehe oben).
 - Falls $s \neq s^*$, so trifft für mind. die Hälfte der Paare zu, dass die Steigung mindestens/höchstens s ist, d.h.
- ⇒ entw. $s_{ab} \leq s < s^*$ oder $s_{ab} \geq s > s^*$ für mind. die Hälfte der Paare
- ⇒ für mindestens $\lfloor \lfloor n/2 \rfloor / 2 \rfloor$ Punktepaare kann jeweils einer der beiden Punkte als Brückenpunkt ausgeschlossen werden

Bestimmen der Brücke: Laufzeit

Falls die Brücke (p_L, p_R) nicht gefunden wird, bleiben von $P = P_L \cup P_R$ noch $P'_L \subseteq P_L$ und $P'_R \subseteq P_R$ mit zusammen höchstens $n \cdot 3/4$ Punkten übrig, in denen analog erneut nach p_L und p_R gesucht wird.

Laufzeit eines Durchlaufs:

- Bis auf Bestimmung des Punktes p_L für die Tangente L an P_L mit Steigung s alles offensichtlich in $O(n)$.
 - p_L ist schlicht der Punkt in P_L , für den die Gerade (durch ihn) mit Steigung s zum größten Y -Achsen-Abschnitt führt → auch $O(n)$
- ⇒ nach $O(n)$ Schritten entweder Brücke gefunden oder Reduktion auf $n \cdot 3/4$ Punkte für „die nächste Runde“

Gesamtlaufzeit:

Sei Laufzeit pro Runde durch $c \cdot \#$ Punkte beschränkt. Dann ist auch die Gesamtlaufzeit durch $O(n)$ beschränkt, denn

$$cn + cn \cdot 3/4 + cn(3/4)^2 + cn(3/4)^3 + \dots \leq c \cdot n \cdot \sum_{i=0}^{\infty} (3/4)^i = c \cdot n \cdot 4$$

Rang- r -Element in erwarteter Linearzeit

Unser D&C-Algorithmus wie auch der Algorithmus nach Kirkpatrick/Seidel nehmen Median-Bestimmung in Zeit $O(n)$ an.

Aus DSA/DAP bekannt: QUICKSELECT bestimmt randomisiert den Median mit max. $4n$ „wesentlichen Vergleichen“ im Erwartungswert.

Idee zur Bestimmung des Elements mit Rang $r \in \{1, \dots, n\}$:

- ① Wähle ein Element x zufällig und partitioniere die Elemente nach kleiner/gleich/größer x (\rightarrow Partition in drei Teile).
- ② IF $\geq r$ Elemente kleiner x THEN suche rekursiv im entsprechenden Teil der Partition nach Element mit Rang r
- ③ ELSE IF $g > n - r$ Elemente größer x THEN suche rekursiv im entspr. Teil der Partition nach Element mit Rang $r - (n - g)$
- ④ ELSE gib x aus.

Hier: Ausgabe kann jedes Element sein, welches bei einer sortierten Reihenfolge der n Elemente an der r ten Position stehen könnte.

Median in deterministischer Linearzeit

Wenn man – statt rand. Wahl – deterministisch mit $O(n)$ Vergleichen ein Pivotelement findet, das sicherstellt, dass (ggf.) in der Rekursion höchstens $n/4$ Elemente betrachtet werden müssen, so würde die Abschätzung der $O(n)$ -Gesamtlaufzeit für „Brücke“ liefern, dass man den Median deterministisch mit $O(n)$ Vergleichen bestimmen kann.

Wahl des Pivots (Blum/Floyd/Pratt/Rivest/Tarjan STOC 1972):

- ① Falls $n \leq 14$, sortiere und gib Median (als Pivot) aus. Sonst:
- ② Gruppier die Elemente in 5er-Gruppen und
- ③ bestimme für jede 5er-Gruppe den Median direkt (7 Vergleiche).
- ④ Bestimme rekursiv den Median dieser Mediane (mit dem oben skizzierten determ. Algorithmus, der diesen hier zur Bestimmung des Pivotelements verwendet) und gib diesen (als Pivot) aus.

Im (interessanten) Fall $n \geq 15$ hat dieser Pivot Rang in $[n/4, n/4]$.

Abschätzung des Rangs des Medians der Mediane

Beweis-Idee/Skizze unter folgenden Annahmen:

- $n/5$ ganzzahlig und aus $\{3, 5, 7, \dots\}$
- alle $n (\geq 15)$ Elemente verschieden ($n < 15$ uninteressant)

Es gibt $m := n/5$ Mediane und $(m - 1)/2$ (ganzz.) dieser Mediane sind kleiner als der (rekursiv bestimmte) Median der Mediane (MdM).

Neben den zwei kleineren Elementen in der 5er-Gruppe des MdM gibt es also noch $(m - 1)/2$ 5er-Gruppen, in denen jeweils neben dem Gruppen-Median noch mindestens zwei weitere Elemente kleiner sind als der MdM.

Insgesamt sind also mindestens $2 + (1 + 2) \cdot (m - 1)/2 = 0,3n + 0,5$ Elemente kleiner als der MdM.

Analog erhält man: Es gibt mindestens $0,3n + 0,5$ größere Elemente.

Anmerkungen zum Algorithmus nach K/S

- Der Algorithmus hat immer eine Laufzeit von $O(n \log h)$ (da deterministisch)
- ⇒ Die ausgabesensitive Komplexität des Konvexe-Hülle-Problems in 2D ist $\Theta(n \log h)$
- Für die Praxis ist der deterministische Linearzeit-Algorithmus zur Medianbestimmung allerdings zu langsam
 - Statt eines randomisierten Median-Algorithmus (*quick select*) würde man „einfach“ direkt die Partitionierung von P in P_L, P_R anhand eines zufällig gewählten Punktes vornehmen
 - und auch direkt eine der (anfänglich $\lfloor n/2 \rfloor$) Steigungen zufällig für die Schätzung der Brückensteigung wählen.
- ⇒ Erwartete Laufzeit durch $c \cdot n \log h$ beschränkt, wobei die Konstante c vergleichsweise klein ist (praktisch relevant)

Zum Abschluss...

... eine Knobelaufgabe:

Gegeben sei (genau) ein fairer Würfel.

Wie oft muss man im Erwartungswert **hintereinander** würfeln, bis man erstmals zwei Sechsen direkt hintereinander gewürfelt hat?

Antwort: **42**

Markov-Kette mit drei Zuständen:

- s Startzustand bzw. letzter Wurf keine Sechs
- 6 letzter Wurf war eine Sechs, aber der davor nicht
- 66 Ziel erreicht

$$E(s \rightarrow 66) = 1 + \frac{1}{6}E(6 \rightarrow 66) + \frac{5}{6}E(s \rightarrow 66)$$
$$\implies E(s \rightarrow 66) = 6 + E(6 \rightarrow 66)$$

Mit $E(6 \rightarrow 66) = 1 + \frac{5}{6}E(s \rightarrow 66)$ erhalten wir

$$E(s \rightarrow 66) = 6 + 1 + \frac{5}{6}E(s \rightarrow 66), \text{ also } E(s \rightarrow 66) = (6 + 1) \cdot 6 = 42$$