Karlsruher Institut für Technologie

Algorithmische Geometrie

Point-Location und Persistenz – Wo bin ich? Und wann?

Einschub: Amortisierte Analyse mit Potentialen

Beispielproblem: binärer Zähler

- starte bei 00000000 (n Bits)
- addiere in jedem Schritt 1

Wie viele Bit-Flips pro Schritt?

- im schlimmsten Fall n
- amortisiert nur konstant viele

(d.h. für k Operationen benötigen wir $O(k)$ Zeit)

Intuition

- aktuell wenige 1en $\rightarrow$ wenige Bit-Flips
- aktuell viele 1en $\rightarrow$ irgendwo kommen die 1en her

Beweisidee

- erstelle ein Unordnungs-Konto
- wer 0en zu 1en macht muss einzahlen (der Zustand wird unordentlicher)
- wer 1en zu 0en macht darf abheben (der Zustand wird ordentlicher)

Beweismethode

- definiere Potential $\Phi(\text{Zustand}) \geq 0$, sodass $\Phi(\text{Start}) = 0$
- amortisierte Kosten einer Operation $=$ Kosten$+(\Phi(\text{nachher}) - \Phi(\text{vorher}))$
- im Beispiel: für $x \in \{0,1\}^n$ wähle $\Phi(x) =$ Anzahl 1en

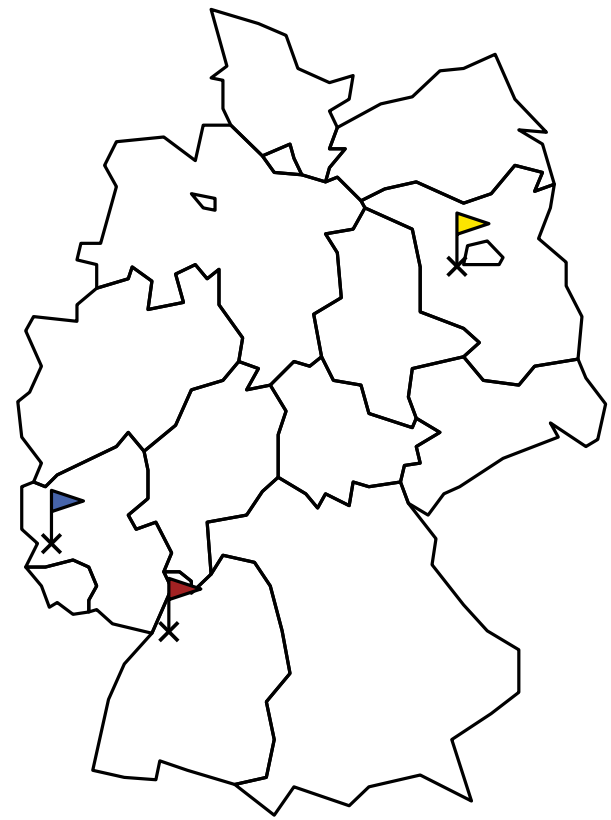
$\Rightarrow$ amortisierte Kosten pro Operation: 2

Wo bin ich?

Problem: Point-Location
In welcher Facette eines geom. Graphen liegt ein gegebener Punkt p?

Statische Variante

- Graph G ist fest
- beantworte Anfragen für viele Punkte $p \in \mathbb{R}^2$
- entwickle Datenstruktur für G, sodass
  - jede Anfrage ist schnell
  - Datenstruktur kann schnell aufgebaut werden
  - Datenstruktur benötigt wenig Platz

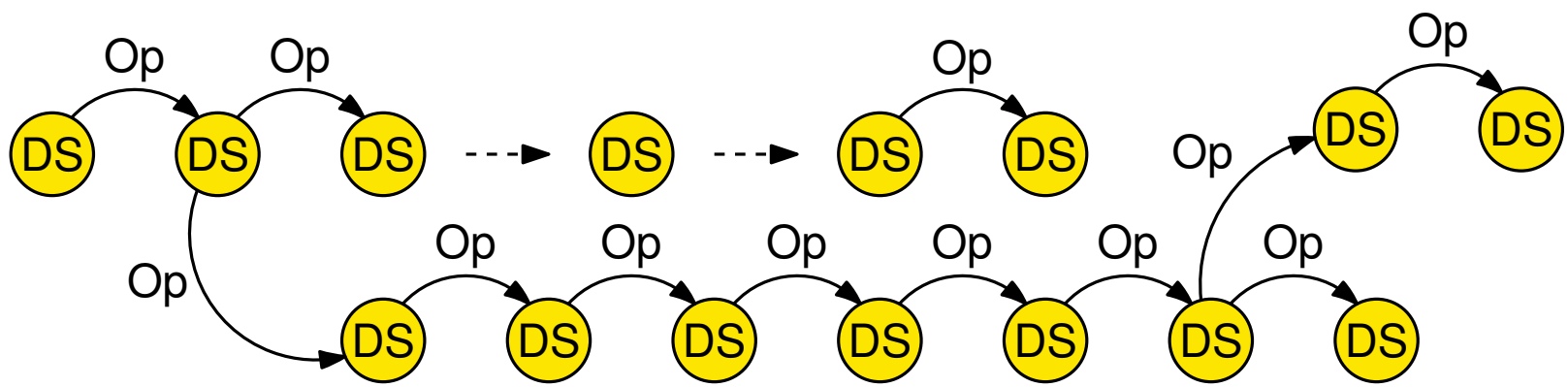


Was sind mögliche Anwendungen?

Datenstrukturen und Zeitreisen

Was kann man mit einer Datenstrukturen typischerweise machen?

- Anfrage an die Datenstruktur im aktuellen Zustand
  (Beispiel priority queue: get-min)
- Anwendung einer Operation $\rightarrow$ neuer Zustand für die Datenstruktur
  (Beispiel priority queue: insert, delete-min)



Partielle Persistenz

- merke dir vergangene Zustände der Datenstruktur
- erlaube Anfragen zu beliebigen (auch vergangenen) Zeitpunkten

Volle Persistenz

- erlaube außerdem Operationen in der Vergangenheit
- Zeit ist dann nicht mehr linear, sondern verzweigt sich

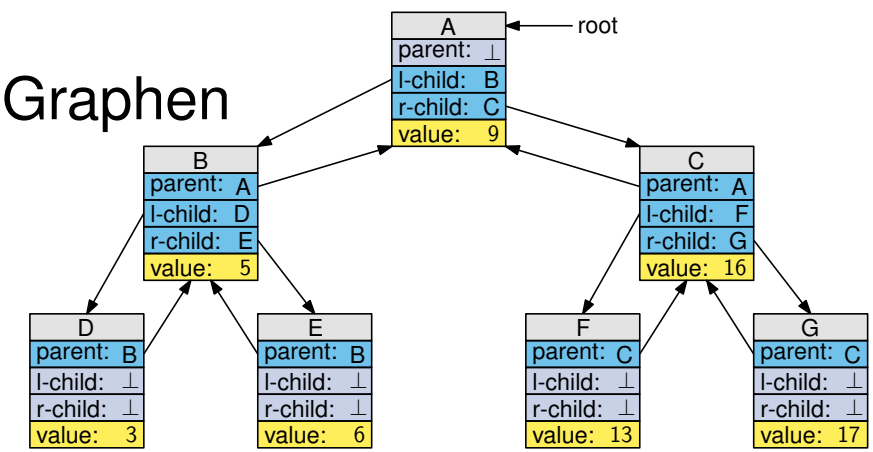
Verlinkte Datenstrukturen / Pointer Machine

Bestandteile einer verlinkten Datenstruktur

- konstant viele Typen von Knoten, jeweils mit konstant vielen Feldern
- jedes Feld enthält entweder
  - ein Datenelement (z.B. eine Zahl),
  - einen Zeiger zu einem Knoten oder einen NULL-Zeiger
- konstante Anzahl an Zeigern zu Startknoten

Anmerkung

- wir haben im Prinzip einen gerichteten Graphen
- mit konstantem Ausgangsgrad
- und konstant viel Speicher pro Knoten



Theorem (partielle Persistenz für alle)
Jede verlinkte Datenstruktur mit beschränktem Eingangsgrad kann mit (amortisiert) konstantem Overhead partiell persistent gemacht werden.

Anfragen, Operationen und Zeitstempel

Beobachtung

- wir wissen nicht, welche Anfragen und Operationen die DS erlaubt
- aber: jede ist Folge von kleineren Anfragen/Operationen
- jede der kleineren Anfragen/Operationen ist schnell ($O(1)$ teurer)
  $\Rightarrow$ gesamte Anfrage/Operation ist schnell ($O(1)$ teurer)

Atomare Anfrage

- lies ein Feld des aktuellen Knotens
- folge einem Zeiger des aktuellen Knotens zu einem anderen Knoten

Atomare Operation

- ändere ein Feld des aktuellen Knotens (Zeiger oder Daten)

Zeitstempel

- wird nach jeder (kompletten) Operation inkrementiert
- vorher: konstant viele Zeiger zu Startknoten
- jetzt: für jeden Zeitstempel konstant viele Zeiger zu Startknoten
  (z.B. konstant viele Arrays mit je einem Zeiger pro Zeitstempel)

Knoten mit Vergangenheit

Atomare Operationen

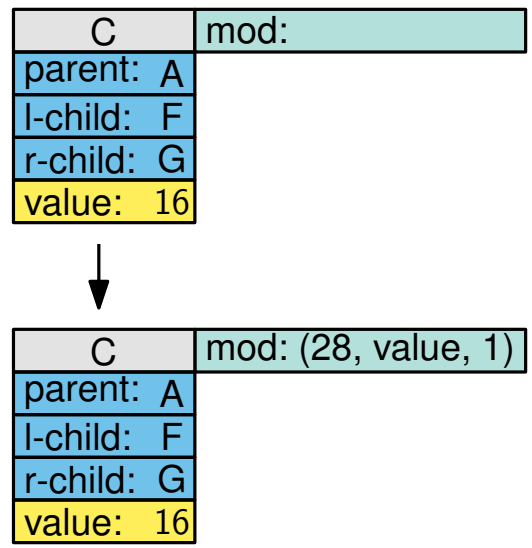
- Idee: jeder Knoten speichert sein eigenes Diff
- jeder Knoten hat ein zusätzliches Feld „mod“
- statt die atomare Operation anzuwenden speichert mod das Tupel (Zeitstempel, Feld, neuer Wert)

Atomare Anfragen

- ziehe Feld mod bei einer Anfrage in Betracht
- Beispielanfrage zum Zeitpunkt t: C.value
  - liefert 16, falls $t < 28$
  - liefert 1, falls $t \geq 28$

Beispieloperation

- Operation: C.value = 1
- aktueller Zeitstempel: 28



Problem: eine Modifikation pro Knoten reicht nicht aus

Idee

- speichere mehrere Modifikationen (wie viele genau überlegen wir uns später)
- alle mod-Felder belegt $\rightarrow$ erstelle neuen Knoten in aktuellem Zustand
- hänge Zeiger um auf neuen Knoten $\rightarrow$ wir brauchen Rück-Zeiger

Warum?

Eine atomare Operation

Felder eines persistenten Knotens: $f + p + m$

- ein Feld für jedes Feld im kurzlebigen Knoten
- p Felder für Rück-Zeiger
- m Felder für Modifikationen

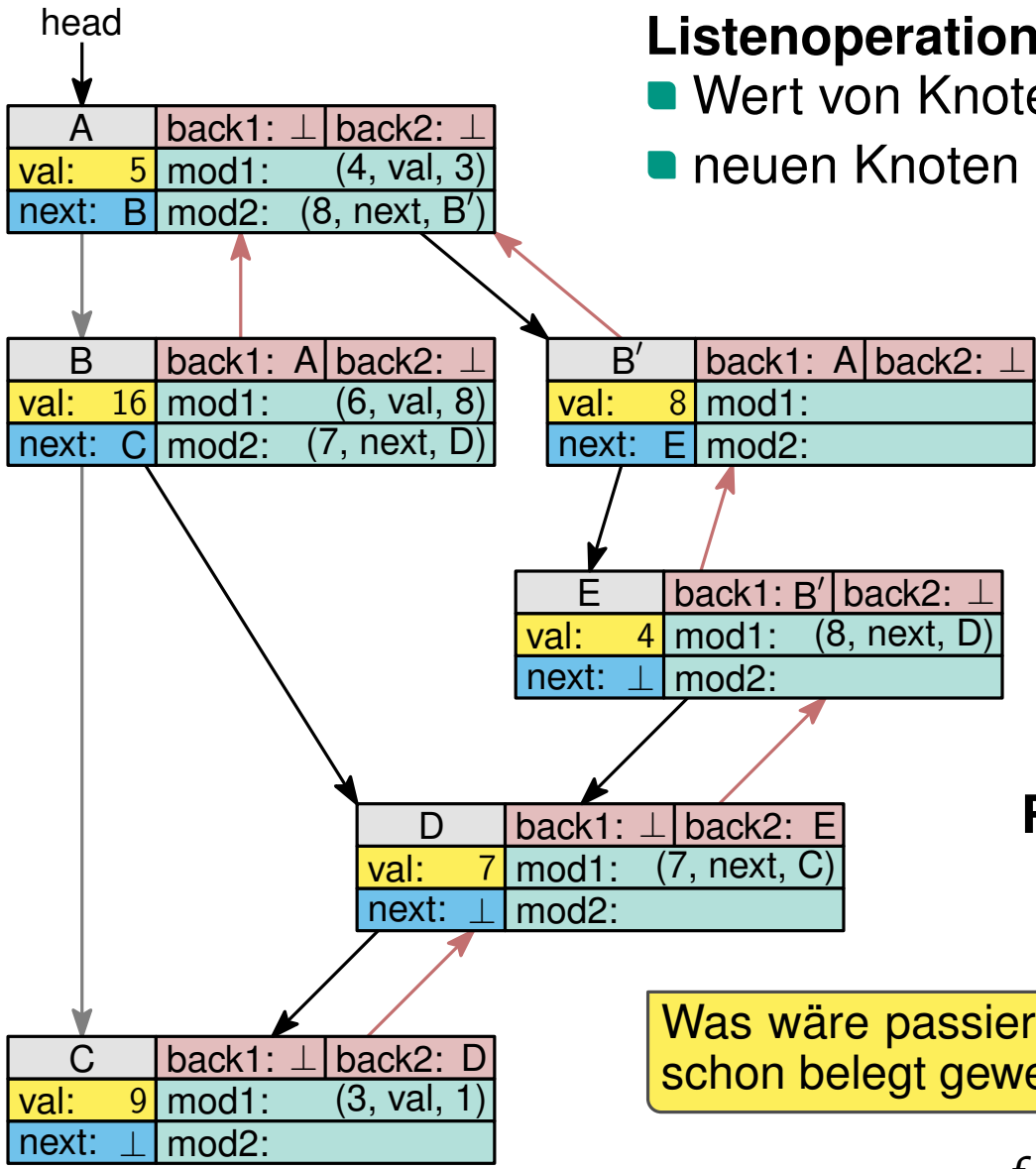
Konstanten
f: Anzahl Felder
p: max. Eingangsgrad
m: Anzahl mod-Felder

Rück-Zeiger: werden wir nur für die aktuellste Version der DS benötigen

Anwendung einer Operation

- Fall 1: es ist noch ein mod-Feld frei
  - trage Änderung in ein mod-Feld ein
  - aktualisiere ggf. Rück-Zeiger
- Fall 2: alle mod-Felder belegt
  - kopiere Knoten $\rightarrow$ alter Knoten: v_{alt}, neuer Knoten: v_{neu}
  - wende alle Modifikationen auf v_{neu} an (inklusive der gerade anstehenden)
  - aktualisiere Rück-Zeiger (entfernt Rück-Zeiger zu v_{alt}, erstellt welche zu v_{neu})
  - hänge rekursiv(!!) Zeiger um: zu v_{neu} statt zu v_{alt}
    $\rightarrow$ dafür brauchen wir die Rück-Zeiger
    $\rightarrow$ atomare Operation, die wir persistent machen müssen

Beispiel: einfach verkettete Liste



Listenoperationen

- Wert von Knoten X auf x setzten: set(X, x)
- neuen Knoten mit Wert y nach X einfügen: ins(X, y)

Zeit	Operationen	atomare Op.
6	set(B, 8)	B.val $=$ 8
7	ins(B, 7)	D $=$ newNode(7)
		D.next $=$ B.next
		B.next $=$ D
8	ins(B, 4)	E $=$ newNode(4)
		E.next $=$ B.next
		B.next $=$ E
		B.nextBack.next $=$ B′

Rekursiver Aufruf!

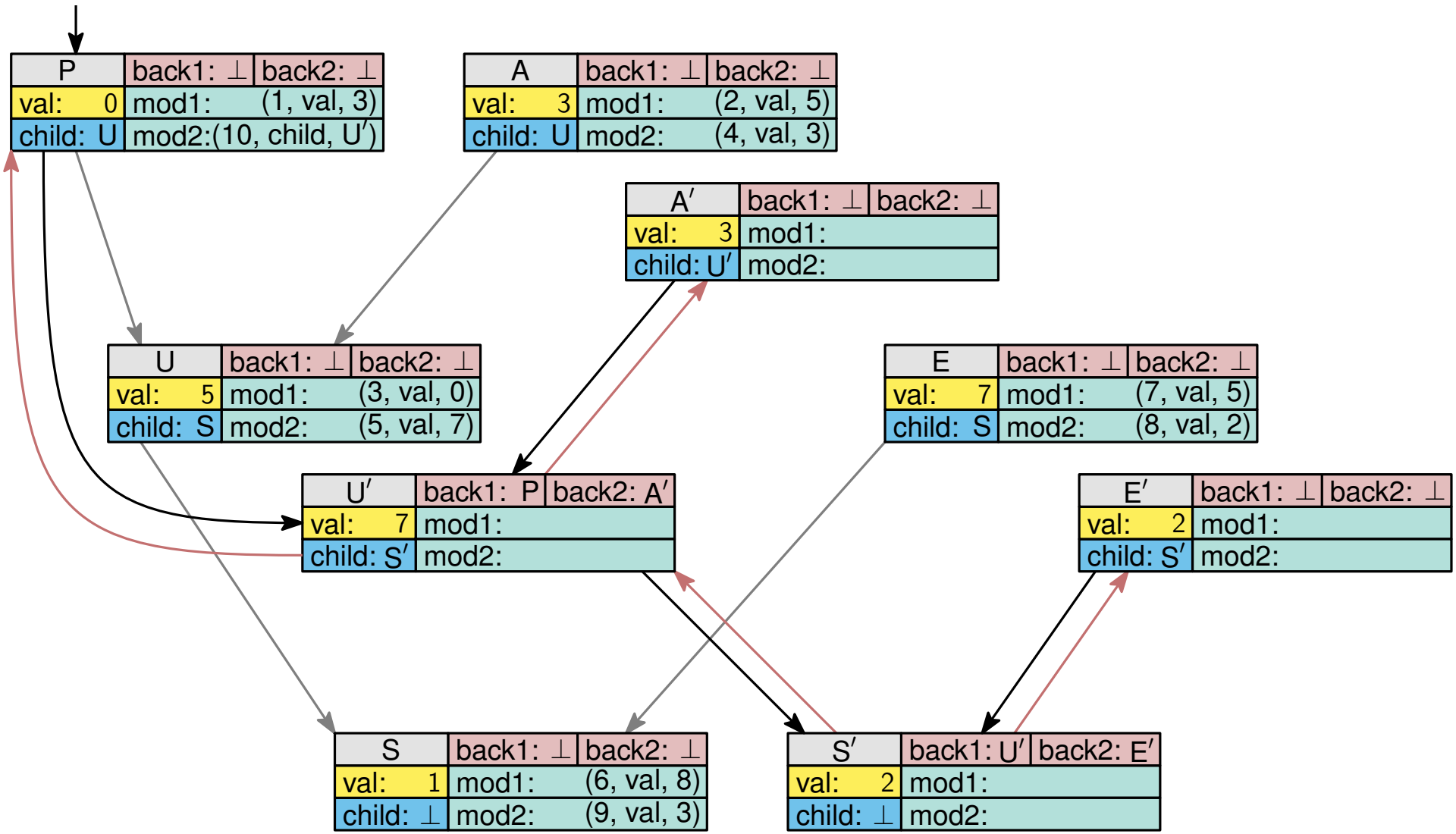
Was wäre passiert, wenn A.mod2 schon belegt gewesen wäre?

$f = 2, p = 2, m = 2$

Konstanten
f: Anzahl Felder
p: max. Eingangsgrad
m: Anzahl mod-Felder

Was passiert hier eigentlich?

Wie viele neue Knoten entstehen bei der Operation S.val = 2?



Laufzeit einer atomaren Operation

Beobachtung

- die rekursiven Aufrufe können kaskadieren
- Worst Case: alle mod-Felder schon belegt
- aber: irgendwer muss die dann schon belegt haben

Konstanten
f: Anzahl Felder
p: max. Eingangsgrad
m: Anzahl mod-Felder

Amortisierte Analyse

(im Beispiel: alle außer B am Ende)

- aktive Knoten: Knoten bei denen mod-Felder nicht übergelaufen sind
- Potential: $\Phi(\text{Zustand}) = c \cdot (\#\text{ belegter mod-Felder in aktiven Knoten})$
- amortisierte Kosten einer atomare Operation:

$$\begin{aligned}\text{Kosten} &\leq c + c + \big[p \cdot \text{Rekursionen} - cm\big] \\ &= 2c + \big[-cm + p \cdot \text{Rekursionen}\big]\end{aligned}$$

Notation: $[x] = 0$ oder x wenn Fall 1 bzw. Fall 2

- expandiere Kosten für Rekursionen:

$$\begin{aligned}&2c - cm + p \cdot \big(2c + \big[-cm + p \cdot \text{Rek.}\big]\big) \\ &= 2c - \cancel{cm} + \cancel{2pc} + p \cdot \big[-cm + p \cdot \text{Rek.}\big] \\ &= 2c + p \cdot \big[-cm + p \cdot \text{Rek.}\big] \\ &\leq 2c\end{aligned}$$

setze $m = 2p$

Anwendung einer Operation
- Fall 1: es ist ein mod-Feld frei
  - trage Änderung in mod-Feld ein
  - aktualisiere ggf. Rück-Zeiger
- Fall 2: alle mod-Felder belegt
  - kopiere Knoten $\to v_{\text{alt}}, v_{\text{neu}}$
  - wende Modifikationen auf v_{neu} an
  - aktualisiere Rück-Zeiger
  - hänge rekursiv(!!) Zeiger um: zu v_{neu} statt zu v_{alt}

Zusammenfassung: Persistenz

Kosten einer atomaren Operation

- amortisiert konstanter Overhead
- amortisiert konstant viel zusätzlicher Speicher

Kosten einer atomaren Anfrage

- konstanter Overhead

Theorem **(partielle Persistenz für alle)**
Jede verlinkte Datenstruktur mit beschränktem Eingangsgrad kann partiell persistent gemacht werden, sodass jede atomare Operation amortisiert konstanten Zeit- und Platz-Overhead hat. Jede Anfrage hat konstanten Overhead.

Anmerkung

- Operation besteht aus x atomaren Operationen $\rightarrow \Theta(x)$ zusätzlicher Platz
- binäre Suchbäume: $O(1)$ Speicher pro Operation möglich
- Aussage kann auf volle Persistenz verallgemeinert werden

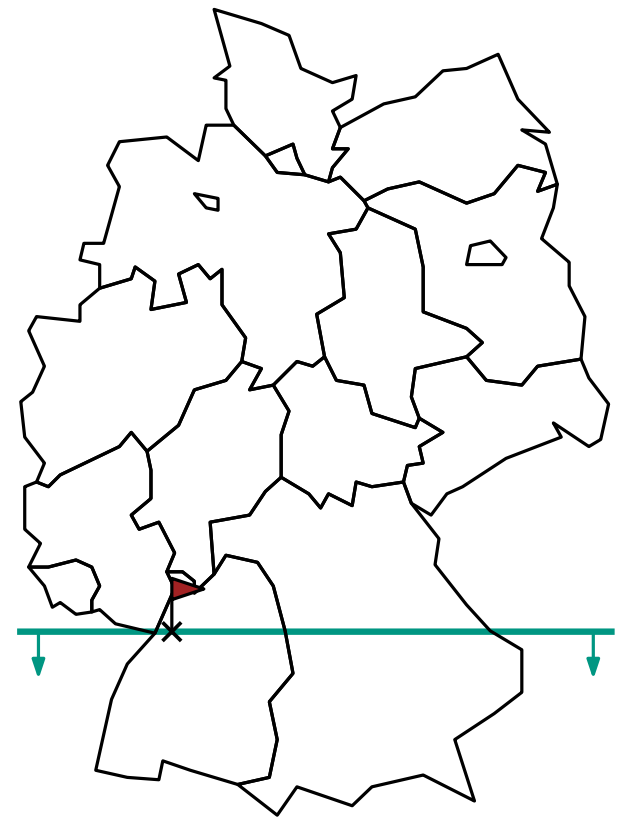
Wo geht unser Vorgehen schief, wenn wir volle Persistenz haben wollen?

Wo bin ich?

Problem: Point-Location
In welcher Facette eines geom. Graphen liegt ein gegebener Punkt p?

Statische Variante

- Graph G ist fest
- beantworte Anfragen für viele Punkte $p \in \mathbb{R}^2$



Idee

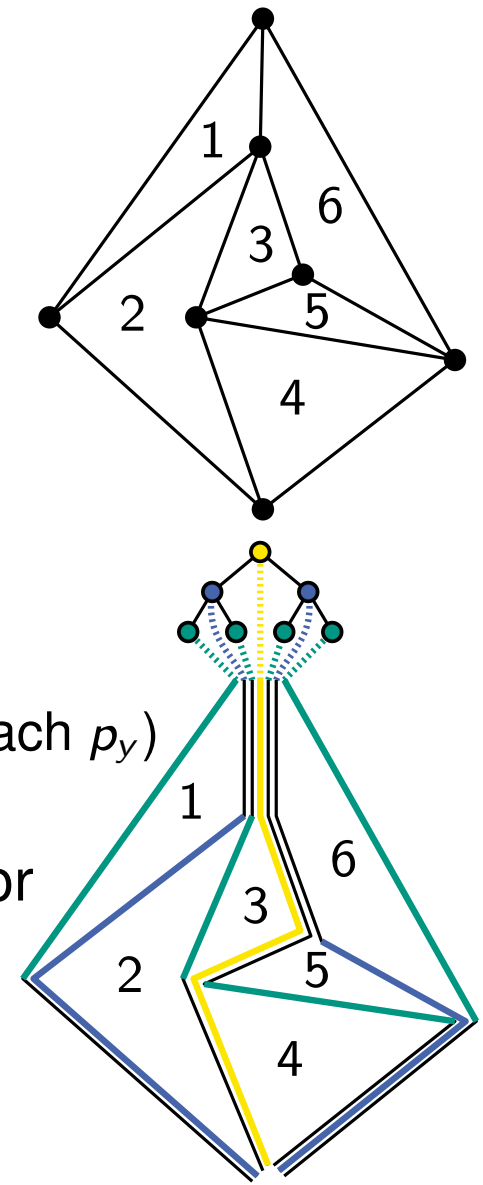
- lasse Sweep-Line Algo für Linienschnitt laufen
- Zeitpunkt p_y: finde Kante e links von p in $O(\log n)$
  (Vorgänger im Sweep-Line Zustand)
- gib Facette rechts neben e in $O(1)$ aus
- Benutze persistenten Suchbaum für SL-Zustand!
  - aus „Wo ist $p = (p_x, p_y)$?“
  - wird „Wo war p_x zum Zeitpunkt p_y?“

Vorberechnung: $O(n \log n)$ **Anfragen:** $O(\log n)$
Speicherplatz: $O(n)$

Das geht bestimmt auch anders!

- mache jede Facette y-monoton
- nummeriere Facette von links nach rechts
  (das geht dank der y-Monotonie)
- zerlege Graph in Pfade bezüglich dieser Ordnung
- jeder dieser Pfade ist monoton
- Ziel: finde zwei konsekutive Pfade, mit p dazwischen
- binärer Suchbaum auf den Pfaden
- jeder Knoten speichert sortierten Teilpfad
  (Kanten aus Vorgängern werden nicht erneut gespeichert)
- finde Punkt im Suchbaum:
  - pro Knoten: finde Kante e im Pfad neben p (Suche nach p_y)
  - laufe links/rechts wenn p links/rechts neben e
  - e existiert nicht $\rightarrow$ selbe Richtung wie ein Schritt zuvor
- Anfrage: $O(\log n)$ binäre Suchen $\rightarrow O(\log^2 n)$
- fractional cascading $\rightarrow$ **Anfragen:** $O(\log n)$

Vorberechnung: $O(n \log n)$ **Speicherplatz:** $O(n)$



Zusammenfassung

Heute gesehen

- Zeitreisen sind nicht nur in Filmen cool
- amortisierte Analyse mit Potentialfunktion
- verschiedene Lösungen für Point-Location
- Ausnutzung unserer Toolbox: Linienschnitt, Triangulierung, fractional cascading

Was gibt es sonst noch?

- weitere Ansätze für Point-Location mit den gleichen Kosten [1]
  (insbesondere gibt es auch einen randomisierten Algo mit ähnlicher Analyse wie bei 2D-LP)
- dynamische Variante
- retroaktive Datenstrukturen

[1] www.csun.edu/~ctoth/Handbook/chap38.pdf