ETH zürich

Übung 7

Datenstrukturen und Algorithmen, D-MATH, ETH Zurich

Programm von heute

Feedback letzte Übung(en)

Wiederholung Theorie
- Quadtrees
- Dynamische Programmierung

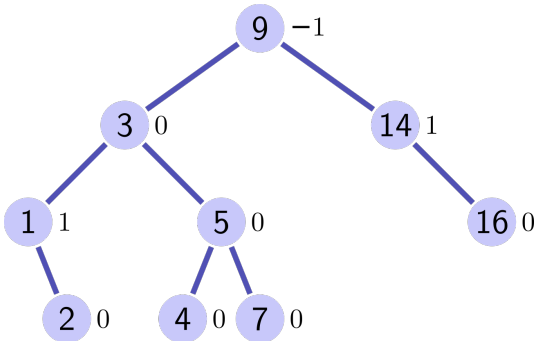
In-Class Exercises

Tipps für die nächsten Übungen

1. Feedback letzte Übung(en)

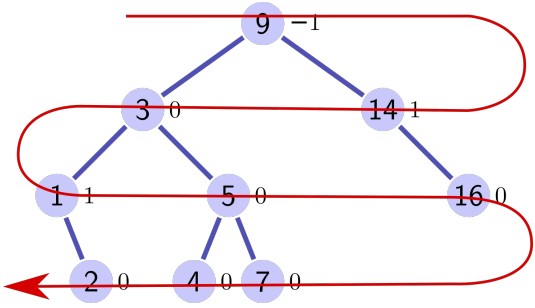
AVL Einfügesequenz

- Gegeben ein AVL Baum: Gibt es eine Einfügesequenz die den selben Baum erstellt und keine Rotation benötigt?



AVL Einfügesequenz

- Gegeben ein AVL Baum: Gibt es eine Einfügesequenz die den selben Baum erstellt und keine Rotation benötigt?



AVL Einfügesequenz - Beweisskizze

- Alle Sequenzen die die Höhenreihenfolge nicht ändern sind i.O.
- Beweis?
- Induktion über Baumhöhe

AVL Einfügesequenz - Beweisskizze

- Alle Sequenzen die die Höhenreihenfolge nicht ändern sind i.O.
- Beweis?
- Induktion über Baumhöhe
- Hypothese: Schlüssel an Höhe h und tiefer sind korrekt platziert und ihre Einfügeoperation verursacht keine Rotation.

AVL Einfügesequenz - Beweisskizze

- Alle Sequenzen die die Höhenreihenfolge nicht ändern sind i.O.
- Beweis?
- Induktion über Baumhöhe
- Hypothese: Schlüssel an Höhe h und tiefer sind korrekt platziert und ihre Einfügeoperation verursacht keine Rotation.
- Schritt: Zeige dass Traversierung gleich ist wie im Originalbaum, ergibt gleiche Positionierung. Dann, benutze AVL Eigenschaft um zu zeigen dass nie einen Höhenunterschied grösser als 1 eintreten kann und deshalb keine Rotationen nötig sind.

2. Wiederholung Theorie

2.1 Quadtrees

Funktionalminimierung zur Bildsegmentierung

$\mathcal{P}$ Partition $\quad$ $\gamma \geq 0$ Regularisierungsparameter
$f_{\mathcal{P}}$ Approximation $\quad$ z Bild = ‘Daten’

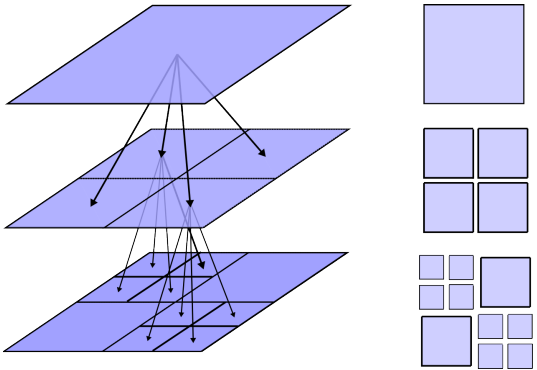
Ziel: Effiziente Minimierung des Funktionals

$$H_{\gamma,z} : \mathfrak{S} \to \mathbb{R}, \quad (\mathcal{P}, f_{\mathcal{P}}) \mapsto \gamma \cdot |\mathcal{P}| + \|z - f_{\mathcal{P}}\|_2^2.$$

Ergebnis $(\hat{\mathcal{P}}, \hat{f}_{\hat{\mathcal{P}}}) \in \operatorname{argmin}_{(\mathcal{P}, f_{\mathcal{P}})} H_{\gamma,z}$ interpretierbar als **optimaler Kompromiss zwischen Regularität und Datentreue**.

Minimierung eines Funktionals mithilfe Quadtrees

Partitionierung eines zweidimensionalen Bereiches in 4 gleich grosse Teile.

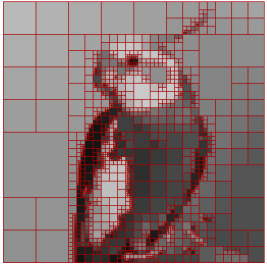


Algorithmus: Minimize(z,r,γ)

Input: Bilddaten $z \in \mathbb{R}^S$, Rechteck $r \subset S$, Regularisierung $\gamma > 0$
Output: $\min_T \gamma|L(T)| + \|z - \mu_{L(T)}\|_2^2$

if $|r| = 0$ **then return** 0
$m \leftarrow \gamma + \sum_{s \in r}(z_s - \mu_r)^2$
if $|r| > 1$ **then**
    Split r into r_{ll},r_{lr},r_{ul},r_{ur}
    $m_1 \leftarrow \text{Minimize}(z, r_{ll}, \gamma)$; $m_2 \leftarrow \text{Minimize}(z, r_{lr}, \gamma)$
    $m_3 \leftarrow \text{Minimize}(z, r_{ul}, \gamma)$; $m_4 \leftarrow \text{Minimize}(z, r_{ur}, \gamma)$
    $m' \leftarrow m_1 + m_2 + m_3 + m_4$
else
    $m' \leftarrow \infty$
if $m' < m$ **then** $m \leftarrow m'$
return m

Minimierung eines Funktionals mithilfe Quadtrees



2.2 Dynamische Programmierung

Dynamische Programmierung: Idee

- Aufteilen eines komplexen Problems in eine vernünftige Anzahl kleinerer Teilprobleme
- Die Lösung der Teilprobleme wird zur Lösung des komplexeren Problems verwendet
- Identische Teilprobleme werden nur einmal gerechnet

Dynamic Programming = Divide-And-Conquer ?

- In beiden Fällen ist das Ursprungsproblem (einfacher) lösbar, indem Lösungen von Teilproblemen herangezogen werden können. Das Problem hat **optimale Substruktur**.
- Bei Divide-And-Conquer Algorithmen (z.B. Mergesort) sind Teilprobleme unabhängig; deren Lösungen werden im Algorithmus nur einmal benötigt.
- Beim DP sind Teilprobleme nicht unabhängig. Das Problem hat **überlappende Teilprobleme**, welche im Algorithmus mehrfach gebraucht werden.
- Damit sie nur einmal gerechnet werden müssen, werden Resultate tabelliert. Dafür darf es **zwischen Teilproblemen keine zirkulären Abhängigkeiten** geben.

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**:

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags**:

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags**: Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags**: Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge**:

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags**: Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge**: In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags**: Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge**: In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?
- **Lösung und Laufzeit**:

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags**: Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge**: In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?
- **Lösung und Laufzeit**: Wie lässt sich die Lösung am Ende aus der Tabelle auslesen? Laufzeit des Algorithmus?

Dynamische Programmierung

Eine vollständige Beschreibung eines dynamischen Programms behandelt **immer** die folgenden Aspekte:

- **Definition der Teilprobleme / der DP-Tabelle**: Welche Dimensionen hat die Tabelle? Was ist die Bedeutung jedes Eintrags?
- **Rekursion: Berechnung eines Eintrags**: Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
- **Topologische Ordnung: Berechnungsreihenfolge**: In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?
- **Lösung und Laufzeit**: Wie lässt sich die Lösung am Ende aus der Tabelle auslesen? Laufzeit des Algorithmus?

3. In-Class Exercises

Longest Ascending Sequence on a Grid

Längste aufsteigende 2D Sequenz

Gegeben $n \times m$ Matrix A:

9	27	42	41	48
35	39	8	3	5
12	49	2	38	4
15	47	29	28	6
19	1	25	33	10

Gesucht längste aufsteigende Sequenz:

$$4, 6, 28, 29, 47, 49$$

Definition der DP-Tabelle

- Welche Dimensionen hat die Tabelle?

Definition der DP-Tabelle

- Welche Dimensionen hat die Tabelle?
  - $n \times m$

Definition der DP-Tabelle

- Welche Dimensionen hat die Tabelle?
  - $n \times m (\times 2)$

Definition der DP-Tabelle

- Welche Dimensionen hat die Tabelle?
  - $n \times m (\times 2)$
- Was ist die Bedeutung jedes Eintrags?

Definition der DP-Tabelle

- Welche Dimensionen hat die Tabelle?
  - $n \times m (\times 2)$
- Was ist die Bedeutung jedes Eintrags?
  - In $T[x][y]$ steht Länge der längsten aufsteigenden Sequenz, die im Feld $A[x][y]$ endet
  - In $S[x][y]$ steht Koordinaten des Vorgängers von (x, y) in aufsteigender Sequenz (falls existent)

Berechnung eines Eintrags

- Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?

Berechnung eines Eintrags

- Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
  - Betrachte Nachbarn mit kleineren Eintrag in A.

Berechnung eines Eintrags

- Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
  - Betrachte Nachbarn mit kleineren Eintrag in A.
  - Wähle von den kleineren Einträgen den mit dem grössten Eintrag in T

Berechnung eines Eintrags

- Wie berechnet sich ein Eintrag aus den Werten von anderen Einträgen? Welche Einträge hängen nicht von anderen Einträgen ab?
  - Betrachte Nachbarn mit kleineren Eintrag in A.
  - Wähle von den kleineren Einträgen den mit dem grössten Eintrag in T
  - Aktualisiere T und S (S erhält Koordinaten vom ausgewählten Nachbar, T erhält Wert um eins erhöht vom ausgewählten Nachbar).

Berechnungsreihenfolge

- In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?

Berechnungsreihenfolge

- In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?
- Bottom-Up: Beginne mit kleinstem Element in A und so weiter. (Bedeutet dass man A sortieren muss.)

Berechnungsreihenfolge

- In welcher Reihenfolge kann man die Einträge berechnen, so dass die jeweils benötigten anderen Einträge bereits vorher berechnet wurden?

- Bottom-Up: Beginne mit kleinstem Element in A und so weiter. (Bedeutet dass man A sortieren muss.)

- Rekursiv: Beliebige Reihenfolge, falls Eintrag schon berechnet überspringen sonst rekursiv von kleinern Nachbarn berechnen.

Auslesen der Lösung

- Wie lässt sich die Lösung am Ende aus der Tabelle auslesen?

Auslesen der Lösung

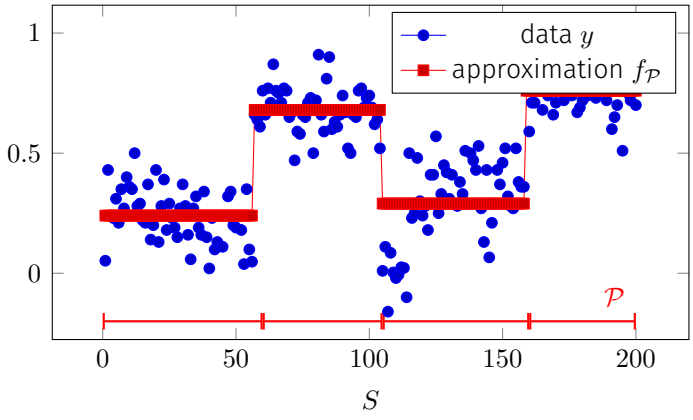
- Wie lässt sich die Lösung am Ende aus der Tabelle auslesen?
  - Betrachte alle Einträge um den Eintrag zu finden, in dem eine längste Sequenz endet. Von dort aus können wir die Lösung rekonstruieren, indem wir dem entsprechenden Vorgänger folgen.

3. In-Class Exercises

Implementieren Sie eine Lösung mit Dynamic Programming im Code Expert Programm $\longrightarrow$ CodeExpert

4. Tipps für die nächsten Übungen

Stückweise konstante Approximation



Stückweise konstante Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

Stückweise konstante Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I\in\mathcal{P}} \sum_{i\in I} (y_i - \mu_I)^2$$

- $\mathcal{P}$: Partition von S (Menge von Intervallen I_i, so dass $\bigcup_i I_i = S$).

Beispiel

$$S = \{1, \cdots, 128\}$$
$$\mathcal{P} = \{[1, 20], [21, 27], [28, 69], [70, 128]\}$$
$$H_{\gamma,y}(\mathcal{P}) = \gamma \cdot 4 + \sum_{I\in\mathcal{P}} \sum_{i\in I} (y_i - \mu_I)^2$$

- **Ziel:** Finde die Partition $\hat{\mathcal{P}}$, so dass $H_{\gamma,y}(\hat{\mathcal{P}})$ minimal

Stückweise konstante Approximation

Minimiere

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

Erklärung des (Hyper-)Parameters γ:

- $\gamma = 0$:
  Anzahl der Intervalle beliebig $\Rightarrow$ Approximation = Daten , viele Sprünge
- $\gamma \approx \infty$:
  Nur ein Intervall $\Rightarrow$ Approximation = Konstante, kein Sprung

γ kontrolliert Balance zwischen Datentreue und Regularität

Trick: Präfixsummen

Ziel: Schnelle Berechnung von

$$M_{i,j} := \sum_{k=i}^{j} y_k \quad (1 \leq i \leq j \leq n)$$

Präfixsummen:

$$Y_i = \sum_{k=1}^{i} y_k \quad (1 \leq k \leq n)$$

Dann

$$Y_i = Y_{i-1} + y_i \quad (1 \leq i \leq n) \quad \text{mit} Y_0 := 0$$
$$M_{i,j} = Y_j - Y_{i-1}$$

$\Rightarrow M_{i,j}$ kann für jedes Paar (i, j) in $\mathcal{O}(1)$ berechnet werden nachdem Y einmal initialisiert wurde in $\mathcal{O}(n)$.

Trick

$$\mu_{[i,j]} = \frac{1}{(j-i+1)} \sum_{k=i}^{j} y_i$$
$$= \frac{1}{(j-i+1)} (Y_j - Y_{i-1})$$

Wir können den Trick auch anwenden auf

$$e_{i,j} := \sum_{k=i}^{j} (y_k - \mu_{[i,j]})^2$$

(wie?)

Stückweise konstante Approximation

$$H_{\gamma,y} : \mathcal{P} \mapsto \gamma|\mathcal{P}| + \sum_{I \in \mathcal{P}} \sum_{i \in I} (y_i - \mu_I)^2$$

- **Ziel:** Finde die Partition $\hat{\mathcal{P}}$, so dass $H_{\gamma,y}(\hat{\mathcal{P}})$ minimal
- **Dynamische Programmierung**: Definition der Tabelle, Berechnung eines Eintrags, Berechnungsreihenfolge, Auslesen der Lösung
- Nutze aus*: $H_{\gamma,y}(\mathcal{P} \cup \{[l, r)\}) = H_{\gamma,y}(\mathcal{P}) + \gamma + e_{[l,r)}$

*Vorraussetzung: $\mathcal{P} \cup \{[l, r)\}$ ist eine Partition