Exercise Session 7

Data Structures and Algorithms, D-MATH, ETH Zurich

Program of today

Feedback of last exercise(s)

Repetition theory

- Quadtrees
- Dynamic Programming

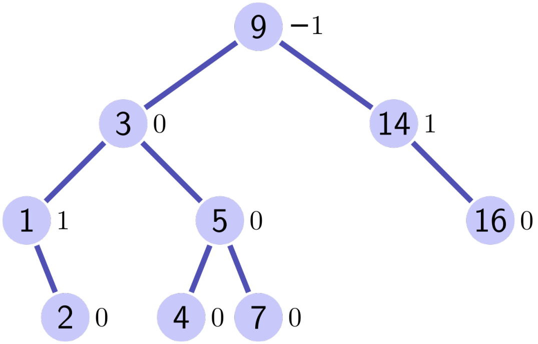
In-Class Exercises

Hints for the Upcoming Exercises

1. Feedback of last exercise(s)

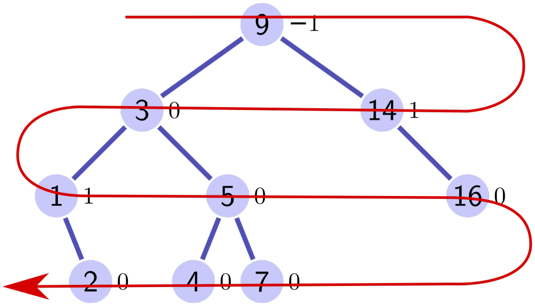
AVL insertion

- Given an AVL tree, is there an order that produces the same tree and does not cause any rotations



AVL insertion

- Given an AVL tree, is there an order that produces the same tree and does not cause any rotations



AVL insertion - sketch of proof

- Any sequence that keeps the height order intact is fine
- Proof?
- By induction over the height of the tree.

AVL insertion - sketch of proof

- Any sequence that keeps the height order intact is fine
- Proof?
- By induction over the height of the tree.
- Hypothesis: Keys at height h and lower are placed in the same place and do not cause rotation.

AVL insertion - sketch of proof

- Any sequence that keeps the height order intact is fine
- Proof?
- By induction over the height of the tree.
- Hypothesis: Keys at height h and lower are placed in the same place and do not cause rotation.
- Step: Show that the traversal is the same as in the original tree, yields same position. Use AVL property of T to show that there will not be a height difference bigger than 1, and therefore no rotation.

2. Repetition theory

2.1 Quadtrees

Minimization of a functional for signal segmentation

$\mathcal{P}$ Partition
$f_{\mathcal{P}}$ approxmation

$\gamma \geq 0$ regularization parameter
z image = 'data'

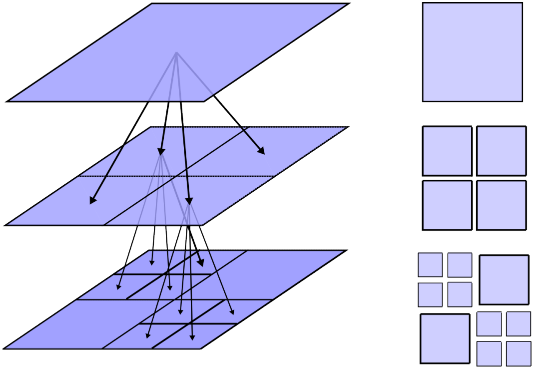
Goal: Efficient mimization of the functional

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

Result $(\hat{\mathcal{P}}, \hat{f}_{\hat{\mathcal{P}}}) \in \operatorname{argmin}_{(\mathcal{P}, f_{\mathcal{P}})} H_{\gamma,z}$ can be interpreted as **optimal compromise between regularity and fidelity to data.**

Minimization of a Functional using Quadtrees

Separation of a two-dimensional range into 4 equally seized parts.

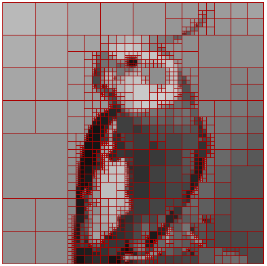


Algorithmus: Minimize(z,r,γ)

Input: Image data $z \in \mathbb{R}^S$, rectangle $r \subset S$, regularization $\gamma > 0$
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

Minimization of a Functional using Quadtrees



2.2 Dynamic Programming

Dynamic Programming: Idea

- Divide a complex problem into a reasonable number of sub-problems
- The solution of the sub-problems will be used to solve the more complex problem
- Identical problems will be computed only once

Dynamic Programming = Divide-And-Conquer ?

- In both cases the original problem can be solved (more easily) by utilizing the solutions of sub-problems. The problem provides **optimal substructure**.
- Divide-And-Conquer algorithms (such as Mergesort): sub-problems are independent; their solutions are required only once in the algorithm.
- DP: sub-problems are dependent. The problem is said to have **overlapping sub-problems** that are required multiple-times in the algorithm.
- In order to avoid redundant computations, results are tabulated. For **sub-problems there must not be any circular dependencies**.

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Recursion: Computation of an entry**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Recursion: Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Recursion: Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Toplogocial order: calculation order**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Recursion: Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Toplogocial order: calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Recursion: Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Toplogocial order: calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- **Solution and Running Time**:

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Recursion: Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Toplogocial order: calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- **Solution and Running Time**: How can the final solution be extracted once the table has been filled? Running time of the DP algorithm.

Dynamic programming

A complete description of a dynamic program **always** consists of the following aspects:

- **Definition of the subproblems / the DP table**: What are the dimensions of the table? What is the meaning of each entry?
- **Recursion: Computation of an entry**: How can an entry be computed from the values of other entries? Which entries do not depend on others?
- **Toplogocial order: calculation order**: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- **Solution and Running Time**: How can the final solution be extracted once the table has been filled? Running time of the DP algorithm.

3. In-Class Exercises

Longest Ascending Sequence on a Grid

Longest Ascending “2D" Sequence

Given $n \times m$ 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

Want the longest ascending sequence:

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

Definition of the DP table

- What are the dimensions of the table?

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m$

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m (\times 2)$

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m(\times 2)$
- What is the meaning of each entry?

Definition of the DP table

- What are the dimensions of the table?
  - $n \times m (\times 2)$
- What is the meaning of each entry?
  - In $T[x][y]$ is the length of the longest ascending sequence that ends in $A[x][y]$
  - In $S[x][y]$ are the coordinates of the predecessor in ascending sequence (if exists)

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?
  - Consider neighbors with smaller entry in A

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?
  - Consider neighbors with smaller entry in A
  - From the smaller entries choose entry with the largest entry in T

Computation of an entry

- How can an entry be computed from the values of other entries? Which entries do not depend on others?
    - Consider neighbors with smaller entry in A
    - From the smaller entries choose entry with the largest entry in T
    - Update T and S (S gets coordinate from selected neighbor, T gets value from selected neighbor increased by one).

Calculation order

- In which order can entries be computed so that values needed for each entry have been determined in previous steps?

Calculation order

- In which order can entries be computed so that values needed for each entry have been determined in previous steps?
- Bottom-Up: Start with smallest element in A and so on. (Means that one has to sort A)

Calculation order

- In which order can entries be computed so that values needed for each entry have been determined in previous steps?

- Bottom-Up: Start with smallest element in A and so on. (Means that one has to sort A)

- Recursively: Arbitrary order, if entry is already computed skip it otherwise compute for smaller neighbor recursively.

Extracting the solution

- How can the final solution be extracted once the table has been filled?

Extracting the solution

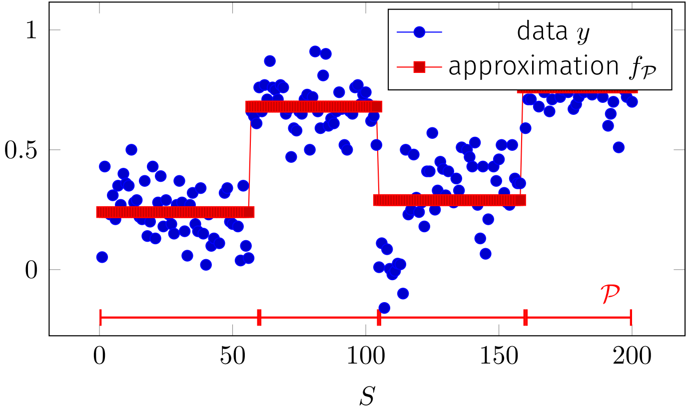
- How can the final solution be extracted once the table has been filled?
    - Consider all entries to find one with a longest sequence. From there, we can reconstruct the solution by following the corresponding predecessors.

3. In-Class Exercises

Implement a DP solution in the prepared CodeExpert program. $\longrightarrow$ CodeExpert

4. Hints for the Upcoming Exercises

Piecewise Constant Approximation



Piecewise Constant Approximation

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

Piecewise Constant Approximation

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

- $\mathcal{P}$: (set of intervals I_i, such that $\bigcup_i I_i = S$).

Example

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

- **Goal:** find the partition $\hat{\mathcal{P}}$ such that $H_{\gamma,y}(\hat{\mathcal{P}})$ is minimal

Piecewise Constant Approximation

Minimize

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

Explanation of the (Hyper-)parameter γ:

- $\gamma = 0$:
  Arbitrary number of intervals $\Rightarrow$ Approximation = Data, many steps
- $\gamma \approx \infty$:
  A single interval $\Rightarrow$ Approximation = Constant, no step

γ controls the balance between regularity and fidelity to data

Trick: Prefix-sums

Goal: fast computation of

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

Prefix-Sums:

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

Dann

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

$\Rightarrow M_{i,j}$ can be computed for each pair (i, j) in $\mathcal{O}(1)$ after Y has been initialized in $\mathcal{O}(n)$.

Trick

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

We can also apply the same trick on

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

(how?)

Piecewise Constant Approximation

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

- **Goal:** find the partition $\hat{\mathcal{P}}$ such that $H_{\gamma,y}(\hat{\mathcal{P}})$ is minimal
- **Dynamic programming**: definition of the table, computation of an entry, calculation order, extracting solution
- Utilize*: $H_{\gamma,y}(\mathcal{P} \cup \{[l,r)\}) = H_{\gamma,y}(\mathcal{P}) + \gamma + e_{[l,r)}$

*Assumption: $\mathcal{P} \cup \{[l,r)\}$ is a partition