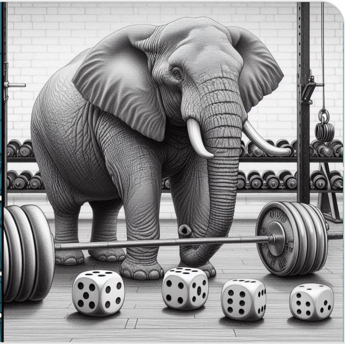
Karlsruhe Institute of Technology

Probability and Computing – The Power of Randomness

Stefan Walzer | WS 2025/2026





Content

Content

Organisation

Lectures by

Dr. Stefan Walzer

likes elephants

Exercises by

Stefan Hermann

- lectures every Thursday, 11:30
- exercises every second Tuesday, 9:45
- exception: switched in first week
- Website: `https://ae.iti.kit.edu/4994.php`
- Ilias: `https://ilias.studium.kit.edu/goto_produktiv_crs_2777317.html`
  - discuss exercises
  - ask questions
  - report typos / mistakes

- oral exam
- literature:
  - Probability and Computing (Mitzenmacher + Upfal)
  - Randomised Algorithms (Motwani + Raghavan)
  - Modern Discrete Probability (Roch)

Exercises

Karlsruhe Institute of Technology

Organisation

- one sheet published with each lecture
- one exercises session every two weeks
  $\Rightarrow$ two sheets per exercises session
- solutions provided after the exercise session
- optional, no hand-in, no grading. *But:*
- content of sheets relevant for exam
  - you may be asked to reproduce/rediscover solutions in the exam

Recommendation

- **You should**, prior to the exercise session
  - **think about** the exerices or
  - **discuss** them in your study group.
- **You should** do **at least one of** the following
  - **solve** the exercises
  - **attend** the exercise sessions and follow along
  - **work through** the provided solutions
- **We hope** that some of you will
  - **present** your own solutions during sessions

Content

1. Organisation

2. The Power of Randomness
   - Improve (Worst-Case) Running Time
   - Model Performance in the Real-World – Average Case Analysis
   - Achieve Load Balancing with Pseudorandomness
   - Approximate in Sublinear Time using Random Sampling

3. Semester Outline

Content

Can Randomness Improve (Worst-Case) Running Time?

it depends on what you mean by “worst case”...

Worst Input & Worst Luck

Any random decision is the worst decision.
$\hookrightarrow$ randomness is useless.

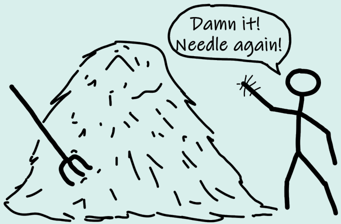
Can Randomness Improve (Worst-Case) Running Time?

it depends on what you mean by "worst case"...

Worst Input & Worst Luck

Any random decision is the worst decision.
$\hookrightarrow$ randomness is useless.

Finding Hay According to This View



Can Randomness Improve (Worst-Case) Running Time?

it depends on what you mean by “worst case”. . .

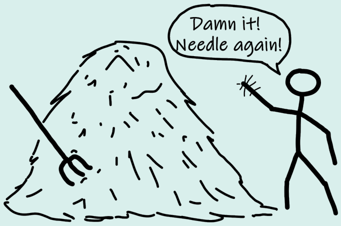
Worst Input & Worst Luck

Any random decision is the worst decision.
$\hookrightarrow$ randomness is useless.

Worst Input & Average Luck

Randomness *can* help. See next slide.

Finding Hay According to This View



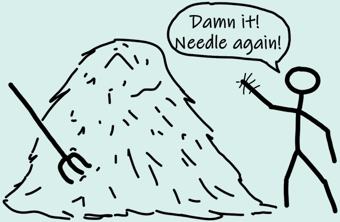
Can Randomness Improve (Worst-Case) Running Time?

it depends on what you mean by "worst case"...

Worst Input & Worst Luck

Any random decision is the worst decision.
$\hookrightarrow$ randomness is useless.

Finding Hay According to This View



Worst Input & Average Luck

Randomness *can* help. See next slide.

$\uparrow$ this is what we mean in the following

In other words:

1. *We* fix a randomised algorithm.
2. *Adversary* fixes an input.
3. Random choices made independently.

Example 1: Finding an Empty Slot

Karlsruhe Institute of Technology

Task

Input: array $A[1..n]$ where $n/2$ slots are empty
Output: $i \in [n]$ with $A[i] = \textsc{Empty}$

$A =$

1	2	3	4	5	6	7	8
x	z			y		w	

Example 1: Finding an Empty Slot

Task

Input: array $A[1..n]$ where $n/2$ slots are empty
Output: $i \in [n]$ with $A[i] = \textsc{Empty}$

Observation

For any *deterministic* algorithm D there exists an input A such that D inspects $\geq n/2$ entries of A.

$A =$

1	2	3	4	5	6	7	8
x	z			y		w	

Example 1: Finding an Empty Slot

Task

Input: array $A[1..n]$ where $n/2$ slots are empty
Output: $i \in [n]$ with $A[i] = \text{EMPTY}$

Observation

For any *deterministic* algorithm D there exists an input A such that D inspects $\geq n/2$ entries of A.

Observation

The randomised algorithm R that inspects slots of A at random finds an empty slot after X attempts where

$$\mathbb{E}[X] \overset{\text{TSF}}{=} \sum_{i \in \mathbb{N}_0} \Pr[X > i] = \sum_{i \in \mathbb{N}_0} 2^{-i} = 2.$$

$A =$

1	2	3	4	5	6	7	8
x	z			y		w	

Example 1: Finding an Empty Slot

Karlsruhe Institute of Technology

Task

Input: array $A[1..n]$ where $n/2$ slots are empty
Output: $i \in [n]$ with $A[i] = \text{EMPTY}$

Observation

For any *deterministic* algorithm D there exists an input A such that D inspects $\geq n/2$ entries of A.

Observation

The randomised algorithm R that inspects slots of A at random finds an empty slot after X attempts where

$$\mathbb{E}[X] \overset{\text{TSF}}{=} \sum_{i \in \mathbb{N}_0} \Pr[X > i] = \sum_{i \in \mathbb{N}_0} 2^{-i} = 2.$$

$A =$

1	2	3	4	5	6	7	8
x	z			y		w	

Note

- the analysis of R holds for *any input*
- "$\mathbb{E}$" relates to choices of R (not to input)
- input is fixed *before* random choices

Example 2 and 3: Verifying Identities

Exercise: Verifying Polynomial Identities

Let f and g be two polynomial functions over a field $\mathbb{F}$. For instance:

$$f(x) = (x^3 + 2x^2 - 5x - 6)(x^2 + x - 20)(x - 6) \text{ and } g(x) = x^6 - 7x^3 + 25.$$

Check whether $f \equiv g$ with a randomised algorithm![1]

[1] The algorithm may occasionally accept incorrect identities. Precise statements on the exercise sheet.

Example 2 and 3: Verifying Identities

Exercise: Verifying Polynomial Identities

Let f and g be two polynomial functions over a field $\mathbb{F}$. For instance:

$$f(x) = (x^3 + 2x^2 - 5x - 6)(x^2 + x - 20)(x - 6) \text{ and } g(x) = x^6 - 7x^3 + 25.$$

Check whether $f \equiv g$ with a randomised algorithm![1]

Exercise: Verifying Matrix Identities (Freivalds' Algorithm)

Let $A, B, C \in \mathbb{F}^{n \times n}$ be matrices over the field $\mathbb{F}$. Check whether $A \cdot B = C$ with a randomised algorithm![1]

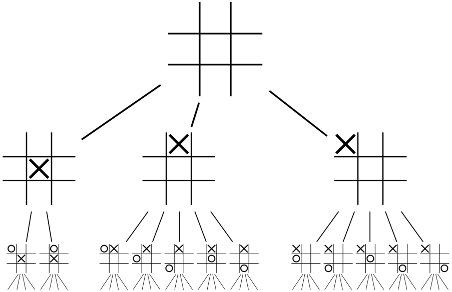
[1] The algorithm may occasionally accept incorrect identities. Precise statements on the exercise sheet.

Example 4: Evaluating Games

Three Types of Game States

value(S) $=$ W // active player has winning strategy
value(S) $=$ L // inactive player has winning strategy
value(S) $=$ D // draw in optimal play

Tic Tac Toe



Example 4: Evaluating Games

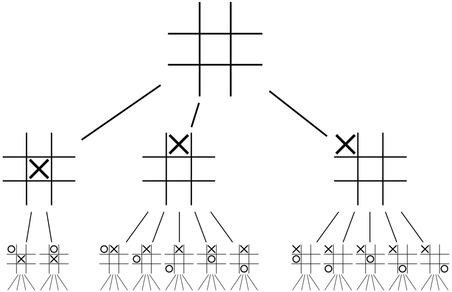
Three Types of Game States

value(S) = W // active player has winning strategy
value(S) = L // inactive player has winning strategy
value(S) = D // draw in optimal play

Task: Evaluating a Game

Input: (Implicit representation of) a game.
Output: value of start state.

Tic Tac Toe



Example 4: Evaluating Games without Draws

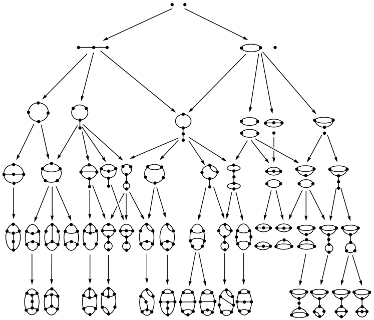
Three Types of Game States

value(S) $=$ W // active player has winning strategy
value(S) $=$ L // inactive player has winning strategy
~~value(S) $=$ D // draw in optimal play~~

Task: Evaluating a Game

Input: (Implicit representation of) a game.
Output: value of start state.

Game of Sprouts (see wikipedia)



Example 4: Evaluating Games without Draws

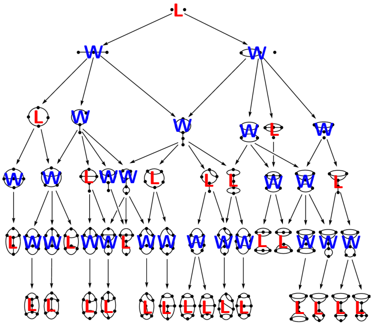
Three Types of Game States

value(S) $=$ W // active player has winning strategy
value(S) $=$ L // inactive player has winning strategy
~~value(S) $=$ D // draw in optimal play~~

Task: Evaluating a Game

Input: (Implicit representation of) a game.
Output: value of start state.

Game of Sprouts (see wikipedia)



Example 4: Evaluating Games without Draws

Karlsruhe Institute of Technology

Three Types of Game States

value$(S) = 1 =$ W // active player has winning strategy
value$(S) = 0 =$ L // inactive player has winning strategy
~~value$(S) = D$ // draw in optimal play~~

Task: Evaluating a Game

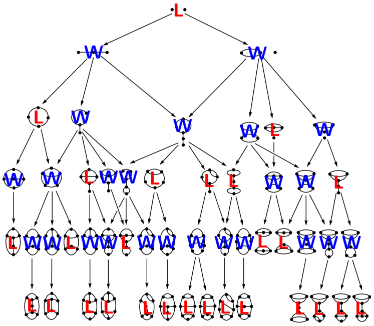
Input: (Implicit representation of) a game.
Output: value of start state.

Observation

A state S is winning if and only if some successor state is losing.

$$\text{value}(S) = \overline{\bigwedge_{S' \text{ successor of } S}} \text{value}(S').$$

Game of Sprouts (see wikipedia)



Example 4: Evaluating Games without Draws

Three Types of Game States

value$(S) = 1 =$ W // active player has winning strategy
value$(S) = 0 =$ L // inactive player has winning strategy
~~value$(S) = D$ // draw in optimal play~~

Task: Evaluating a Game

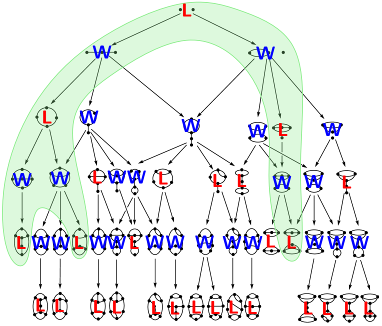
Input: (Implicit representation of) a game.
Output: value of start state.

Observation

A state S is winning if and only if some successor state is losing.

$$\text{value}(S) = \overline{\bigwedge_{S' \text{ successor of } S}} \text{value}(S').$$

Game of Sprouts (see wikipedia)



Observation

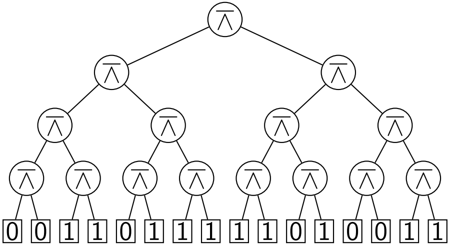
May not have to inspect entire tree to derive value at root.

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Karlsruhe Institute of Technology

Problem

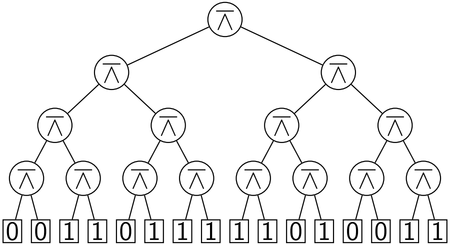
Input: $I \in \{0,1\}^n$ for $n = 2^d$.
Output: Value of complete binary $\overline{\wedge}$-tree with leaf values from I.
Cost Model: Number of inspected entries of I.



Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Problem

Input: $I \in \{0,1\}^n$ for $n = 2^d$.
Output: Value of complete binary $\overline{\wedge}$-tree with leaf values from I.
Cost Model: Number of inspected entries of I.



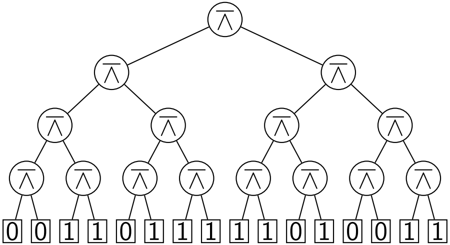
Exercise

For any deterministic algorithm A there exists an input $I_A \in \{0,1\}^n$ such that A inspects all n entries of I.

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Problem

Input: $I \in \{0,1\}^n$ for $n = 2^d$.
Output: Value of complete binary $\overline{\wedge}$-tree with leaf values from I.
Cost Model: Number of inspected entries of I.



Exercise

For any deterministic algorithm A there exists an input $I_A \in \{0,1\}^n$ such that A inspects all n entries of I.

Our Goal

Randomised algorithm that, for any input, inspects only X entries with

$$\mathbb{E}[X] = \mathcal{O}(n^{0.793}).$$

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Algorithm randEval(T)**:**

```
if T = Leaf(b) then
    return b
(T_0, T_1) ← T
// coin flip:
sample r ~ U({0, 1})
b_r ← randEval(T_r)
if b_r = 0 then
    return 1
return 1 − randEval(T_{1−r})
```

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Lemma

Assume randEval is excecuted for a tree T of depth $d \geq 2$. Let X be the number of resulting calls with subtrees of depth $d-2$. Then $\mathbb{E}[X] \leq 3$.

Algorithm randEval(T)**:**

```
if T = Leaf(b) then
    return b
(T_0, T_1) ← T
// coin flip:
sample r ~ U({0,1})
b_r ← randEval(T_r)
if b_r = 0 then
    return 1
return 1 − randEval(T_{1−r})
```

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Karlsruhe Institute of Technology

Algorithm randEval(T)**:**

if $T = \text{Leaf}(b)$ **then**
  return b
$(T_0, T_1) \leftarrow T$
// coin flip:
sample $r \sim \mathcal{U}(\{0, 1\})$
$b_r \leftarrow \text{randEval}(T_r)$
if $b_r = 0$ **then**
  return 1
return $1 - \text{randEval}(T_{1-r})$

Lemma

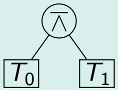
Assume randEval is excecuted for a tree T of depth $d \geq 2$. Let X be the number of resulting calls with subtrees of depth $d - 2$. Then $\mathbb{E}[X] \leq 3$.

Proof.

Let $T = (T_0, T_1) = ((T_{00}, T_{01}), (T_{10}, T_{11}))$.

Case 1: $\text{value}(T) = 1$.

- Then $\text{value}(T_0) = 0$ or $\text{value}(T_1) = 0$ (or both).
- Assume (wlog) $\text{value}(T_0) = 0$.
- With probability $1/2$ we select $r = 0$ and T_1 need not be evaluated.



$$\Rightarrow \mathbb{E}[X] \leq \tfrac{1}{2} \cdot 2 + \tfrac{1}{2} \cdot 4 = 3.$$

$\square$

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Algorithm randEval(T):
- **if** $T = \text{Leaf}(b)$ **then**
  - **return** b
- $(T_0, T_1) \leftarrow T$
- // coin flip:
- sample $r \sim \mathcal{U}(\{0,1\})$
- $b_r \leftarrow \text{randEval}(T_r)$
- **if** $b_r = 0$ **then**
  - **return** 1
- **return** $1 - \text{randEval}(T_{1-r})$

Lemma

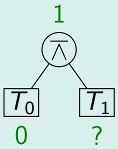
Assume randEval is excecuted for a tree T of depth $d \geq 2$. Let X be the number of resulting calls with subtrees of depth $d-2$. Then $\mathbb{E}[X] \leq 3$.

Proof.

Let $T = (T_0, T_1) = ((T_{00}, T_{01}), (T_{10}, T_{11}))$.

Case 1: $\text{value}(T) = 1$.

- Then $\text{value}(T_0) = 0$ or $\text{value}(T_1) = 0$ (or both).
- Assume (wlog) $\text{value}(T_0) = 0$.
- With probability $1/2$ we select $r = 0$ and T_1 need not be evaluated.



$$\Rightarrow \mathbb{E}[X] \leq \tfrac{1}{2} \cdot 2 + \tfrac{1}{2} \cdot 4 = 3.$$

□

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Algorithm randEval(T)**:**

- **if** $T = \text{Leaf}(b)$ **then**
  - **return** b
- $(T_0, T_1) \leftarrow T$
- // coin flip:
- sample $r \sim \mathcal{U}(\{0, 1\})$
- $b_r \leftarrow \text{randEval}(T_r)$
- **if** $b_r = 0$ **then**
  - **return** 1
- **return** $1 - \text{randEval}(T_{1-r})$

Lemma

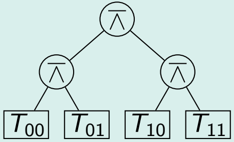
Assume randEval is excecuted for a tree T of depth $d \geq 2$. Let X be the number of resulting calls with subtrees of depth $d - 2$. Then $\mathbb{E}[X] \leq 3$.

Proof.

Let $T = (T_0, T_1) = ((T_{00}, T_{01}), (T_{10}, T_{11}))$.

Case 2: $\text{value}(T) = 0$.

- Then $\text{value}(T_0) = \text{value}(T_1) = 1$.
- Like before: T_{01} and T_{11} only evaluated with probability $1/2$ each.



$$\Rightarrow \mathbb{E}[X] \leq 2 + \tfrac{1}{2} \cdot 1 + \tfrac{1}{2} \cdot 1 = 3.$$

$\square$

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Karlsruhe Institute of Technology

Algorithm randEval(T)**:**

- **if** $T = \text{Leaf}(b)$ **then**
  - **return** b
- $(T_0, T_1) \leftarrow T$
- // coin flip:
- sample $r \sim \mathcal{U}(\{0, 1\})$
- $b_r \leftarrow \text{randEval}(T_r)$
- **if** $b_r = 0$ **then**
  - **return** 1
- **return** $1 - \text{randEval}(T_{1-r})$

Lemma

Assume randEval is excecuted for a tree T of depth $d \geq 2$. Let X be the number of resulting calls with subtrees of depth $d-2$. Then $\mathbb{E}[X] \leq 3$.

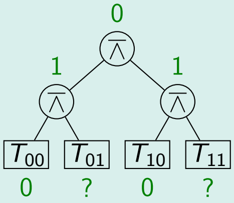
Proof.

Let $T = (T_0, T_1) = ((T_{00}, T_{01}), (T_{10}, T_{11}))$.

Case 2: $\text{value}(T) = 0$.

- Then $\text{value}(T_0) = \text{value}(T_1) = 1$.
- Like before: T_{01} and T_{11} only evaluated with probability $1/2$ each.

$$\Rightarrow \mathbb{E}[X] \leq 2 + \tfrac{1}{2} \cdot 1 + \tfrac{1}{2} \cdot 1 = 3.$$



$\square$

Example 4 Simplified: Evaluating $\overline{\wedge}$-Trees

Karlsruhe Institute of Technology

Algorithm randEval(T)**:**
- **if** $T = \text{Leaf}(b)$ **then**
  - **return** b
- $(T_0, T_1) \leftarrow T$
- // coin flip:
- sample $r \sim \mathcal{U}(\{0,1\})$
- $b_r \leftarrow \text{randEval}(T_r)$
- **if** $b_r = 0$ **then**
  - **return** 1
- **return** $1 - \text{randEval}(T_{1-r})$

Lemma

Assume randEval is excecuted for a tree T of depth $d \geq 2$. Let X be the number of resulting calls with subtrees of depth $d-2$. Then $\mathbb{E}[X] \leq 3$.

Corollary

Let T be a tree of depth $d \in \{0, 2, 4, \ldots\}$, i.e. $n = 2^d$.
The number L of leafs visited by randEval(T) satisfies

$$\underbrace{\mathbb{E}[L] \leq 3^{d/2}}_{\text{proof on blackboard}} = 4^{\log_4(3^{d/2})} = 4^{d/2\log_4(3)} = 2^{d\log_4(3)} = n^{\log_4(3)}.$$

Content

Karlsruhe Institute of Technology

Average Case Analysis

Theory-Practice Gap

$$\text{SAT is NP-complete} \overset{???}{\longleftrightarrow} \begin{array}{c}\text{modern SAT-solvers handle relevant}\\\text{instances with millions of clauses}\end{array}$$

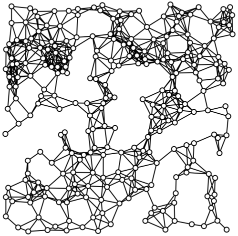
Similar observations for NP-hard graph problems on relevant graph classes, e.g. social networks.

Average Case Analysis

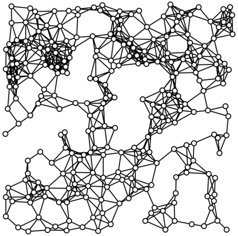
Theory-Practice Gap

SAT is NP-complete $\overset{???}{\longleftrightarrow}$ modern SAT-solvers handle relevant instances with millions of clauses

Similar observations for NP-hard graph problems on relevant graph classes, e.g. social networks.



Average Case Analysis



Theory-Practice Gap

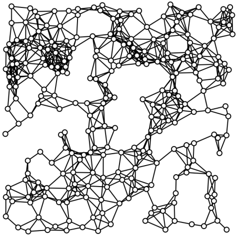
SAT is NP-complete $\overset{???}{\longleftrightarrow}$ modern SAT-solvers handle relevant instances with millions of clauses

Similar observations for NP-hard graph problems on relevant graph classes, e.g. social networks.

Explaining the Gap

1. Define a distribution $\mathcal{I}$ on inputs.
   - $\mathcal{I}$ should be realistic, i.e. model real world instances
   - $\mathcal{I}$ should have simple mathematical structure
2. Show that time to solve $I \sim \mathcal{I}$ is small *in expectation*.

Average Case Analysis



Theory-Practice Gap

SAT is NP-complete $\overset{???}{\longleftrightarrow}$ modern SAT-solvers handle relevant instances with millions of clauses

Similar observations for NP-hard graph problems on relevant graph classes, e.g. social networks.

Explaining the Gap

1. Define a distribution $\mathcal{I}$ on inputs.
   - $\mathcal{I}$ should be realistic, i.e. model real world instances
   - $\mathcal{I}$ should have simple mathematical structure
2. Show that time to solve $I \sim \mathcal{I}$ is small *in expectation*.

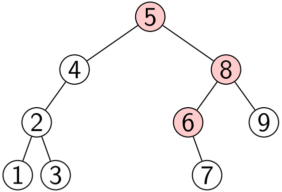
Goals

- model real world instances
- identify useful properties of these instances
- build algorithms exploiting these properties

Toy Example: Unbalanced Search Trees

Setting

Inserted $1, \dots, n$ into search tree *in some order*.
Consider: Depth of Element $y \in \{1, \dots, n\}$.



$\text{depth}(6) = 3$

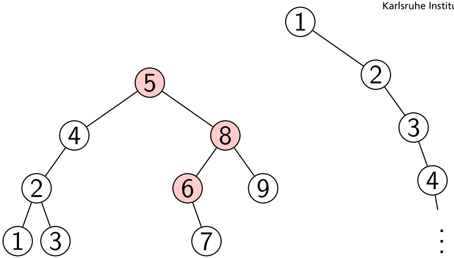
Toy Example: Unbalanced Search Trees

Setting

Inserted $1, \ldots, n$ into search tree *in some order*.
Consider: Depth of Element $y \in \{1, \ldots, n\}$.

Worst Case

Sorted order: $\text{depth}(y) = y$.



$\text{depth}(6) = 3$

Toy Example: Unbalanced Search Trees

Setting

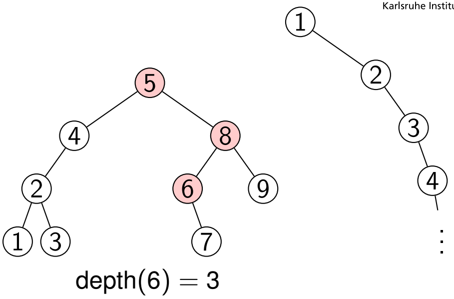
Inserted $1, \ldots, n$ into search tree *in some order*.
Consider: Depth of Element $y \in \{1, \ldots, n\}$.

Worst Case

Sorted order: $\text{depth}(y) = y$.

Possible Observation

- Alice sees good performance in her setting.
- Can we explain why that might be?



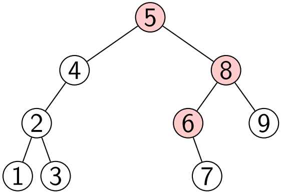
Toy Example: Unbalanced Search Trees

Setting

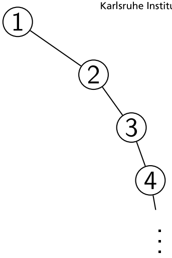
Inserted $1, \ldots, n$ into search tree *in some order*.
Consider: Depth of Element $y \in \{1, \ldots, n\}$.

Worst Case

Sorted order: $\text{depth}(y) = y$.



$\text{depth}(6) = 3$



Possible Observation

- Alice sees good performance in her setting.
- Can we explain why that might be?

Average Case Analysis

1. Model: Elements of $\{1, \ldots, n\}$ are inserted in random order.
   $\hookrightarrow$ Note: May or may not reflect Alice's setting...
2. Goal: Show that $y \in \{1, \ldots, n\}$ has expected depth $\mathcal{O}(\log n)$.
   $\hookrightarrow$ Proved on next slide!

Toy Example: Unbalanced Search Trees – Analysis

Karlsruhe Institute of Technology

Lemma

For any $x, y \in [n]$: $\Pr[E_{xy}] = \frac{1}{|y-x|+1}$.

Context

Elements $\{1, \ldots, n\}$ inserted into search tree in uniformly random order.

Definition

Event $E_{xy} = \{x \text{ is ancestor of } y\}$

// x counts as ancestor of x

Toy Example: Unbalanced Search Trees – Analysis

Karlsruhe Institute of Technology

Lemma

For any $x, y \in [n] : \Pr[E_{xy}] = \frac{1}{|y-x|+1}$.

Proof.

Assume wlog $x < y$.

$\square$

Context

Elements $\{1, \ldots, n\}$ inserted into search tree in uniformly random order.

Definition

Event $E_{xy} = \{x \text{ is ancestor of } y\}$

// x counts as ancestor of x

Toy Example: Unbalanced Search Trees – Analysis

Lemma

For any $x, y \in [n] : \Pr[E_{xy}] = \frac{1}{|y-x|+1}$.

Proof.

Assume wlog $x < y$.
Let v be the element of $\{x, \ldots, y\}$ inserted first.
Note: All elements of $\{x, \ldots, y\}$ are descendents of v.

$\square$

Context

Elements $\{1, \ldots, n\}$ inserted into search tree in uniformly random order.

Definition

Event $E_{xy} = \{x \text{ is ancestor of } y\}$
// x counts as ancestor of x

Toy Example: Unbalanced Search Trees – Analysis

Karlsruhe Institute of Technology

Lemma

For any $x, y \in [n] : \Pr[E_{xy}] = \frac{1}{|y-x|+1}$.

Proof.

Assume wlog $x < y$.
Let v be the element of $\{x, \dots, y\}$ inserted first.
Note: All elements of $\{x, \dots, y\}$ are descendents of v.

Case 1: $v = x$. Then x is ancestor of y.

Case 2: $v = y$. Then y is ancestor of x.

Case 3: $v \notin \{x, y\}$. Then x is in left subtree of v and y in right subtree of v.

$\square$

Context

Elements $\{1, \dots, n\}$ inserted into search tree in uniformly random order.

Definition

Event $E_{xy} = \{x \text{ is ancestor of } y\}$

// x counts as ancestor of x

Toy Example: Unbalanced Search Trees – Analysis

Karlsruhe Institute of Technology

Lemma

For any $x, y \in [n]$: $\Pr[E_{xy}] = \frac{1}{|y-x|+1}$.

Proof.

Assume wlog $x < y$.
Let v be the element of $\{x, \ldots, y\}$ inserted first.
Note: All elements of $\{x, \ldots, y\}$ are descendents of v.

Case 1: $v = x$. Then x is ancestor of y.

Case 2: $v = y$. Then y is ancestor of x.

Case 3: $v \notin \{x, y\}$. Then x is in left subtree of v and y in right subtree of v.

Hence E_{xy} occurs $\Leftrightarrow x = v \Leftrightarrow$ Case 1.
Therefore: $\Pr[E_{xy}] = \Pr[\text{Case 1}] = \frac{1}{|\{x,\ldots,y\}|} = \frac{1}{y-x+1}$. □

Context

Elements $\{1, \ldots, n\}$ inserted into search tree in uniformly random order.

Definition

Event $E_{xy} = \{x \text{ is ancestor of } y\}$

// x counts as ancestor of x

Toy Example: Unbalanced Search Trees – Analysis

Karlsruhe Institute of Technology

Lemma

For any $x, y \in [n]$: $\Pr[E_{xy}] = \frac{1}{|y-x|+1}$.

Theorem

Let $y \in [n]$ and ℓ_y the depth y. Then $\mathbb{E}[\ell_y] \leq 2\ln(n) + 2$.

Context

Elements $\{1, \ldots, n\}$ inserted into search tree in uniformly random order.

Definition

Event $E_{xy} = \{x \text{ is ancestor of } y\}$

// x counts as ancestor of x

Toy Example: Unbalanced Search Trees – Analysis

Karlsruhe Institute of Technology

Lemma

For any $x, y \in [n]$: $\Pr[E_{xy}] = \frac{1}{|y-x|+1}$.

Theorem

Let $y \in [n]$ and ℓ_y the depth y. Then $\mathbb{E}[\ell_y] \leq 2\ln(n) + 2$.

Proof.

We have $\ell_y = \sum_{x\in[n]} \mathbb{1}_{E_{xy}}$. Hence:

$$\mathbb{E}[\ell_y] \overset{\text{lin.}}{=} \sum_{x\in[n]} \mathbb{E}[\mathbb{1}_{E_{xy}}] = \sum_{x\in[n]} \Pr[E_{xy}] = \sum_{x\in[n]} \frac{1}{|y-x|+1}$$

$$\leq 2\sum_{i=1}^{n} \frac{1}{i} = 2 \cdot H_n \leq 2(\ln(n)+1).$$

$\square$

Context

Elements $\{1, \dots, n\}$ inserted into search tree in uniformly random order.

Definition

Event $E_{xy} = \{x \text{ is ancestor of } y\}$

// x counts as ancestor of x

Content

1. Organisation

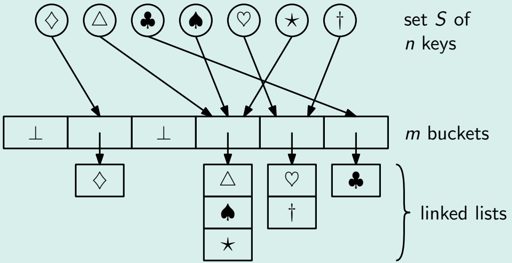
2. The Power of Randomness
   - Improve (Worst-Case) Running Time
   - Model Performance in the Real-World – Average Case Analysis
   - Achieve Load Balancing with Pseudorandomness
   - Approximate in Sublinear Time using Random Sampling

3. Semester Outline

Achieve Load Balancing with Pseudorandomness

Karlsruhe Institute of Technology

Hashing with Chaining:



Stay Tuned!

- Linear Probing
- Cuckoo Hashing
- Bloom Filters
- Retrieval
- Perfect Hashing

Content

Approximate in Sublinear Time using Random Sampling

Karlsruhe Institute of Technology

More $\times$ or more $\circ$?

XOXOXXXOXXXXXXXOXXXXOXXXXOOOOXXXXXOXOXOXXOXOXOXXXXXXX
XXOOOXXXXOXOXXXXOXXOOXXXXXOOOOOXOXXXOXXOOXXXXXXXOXOXX
XOOXXOXXOXOXOOOXOXOXXOOOOXXXOXXOXXXXOOXOOOOOOXXXOXXXX
XOXXXXOXXOOOOXOOXXXXXOOOXXXXXOOXOXXXXXXXOXOOOXXXXXXXX
XOOXXXXOXXOXOOOXXXXXXXXXOXXXOXOXOXOXXXXXXOOXXXXXXOOXX
OOXOXOXXXOOXXXXXOXOOOOXOOXOXXOXOOXXOXXOXXXXOXOXXOXX
XXOXXXXXXXXXXXOXXOXOXOXXXXXXOXXXOXXXXXXOXOXXXXOXXXXXX
XXXXXXOXOXXOOXOOXXOXXXXXXOXXXOXXOXOOXXXXOOXOOXXXXXXXX
OXXOXOXXXXXXXOXXXOXXXOXXXXXXXXXXXXXOXOXOXOXXXOOXOXOOX
OXXXOXXXOXXXXXXOOXXXOXOOXXXXXOXXXOXOXOXXXXOXXXXOXXXXO
OXOXXOXOXOXOOXXOXXOOXXXOXOOXXXOXXOOXOXOXOXXXXOXXOXXOX
XXXXOXXOXXOXXXXOXOXXXXXXXOOXXXXOXXXXOXOXXOXXXXXXXXXOO
OOXOXXXXXXXXXXOOOOOOXXXXXXOXXXXXOXXOXXXOXOOXXOXXXOOOX
XXXXOOXXOOXOXXXOXOXXOXXXXOOXOOOXXOOXXXXOOXXOOXOXXOXXX

Stay Tuned!

Approximation algorithms can estimate quantities by random sampling.

Content

Semester Outline

Karlsruhe Institute of Technology

Tools from Probability Theory

- Concentration Bounds
- Random Coupling
- Yao’s Principle
- Method of Bounded Differences

Random Graph Models

- Erdős-Renyi Random Graphs
- Branching Processes
- Random Geometric Graphs

Other Stuff

- Randomised Complexity Classes
- Probabilistic Method

Algorithm Design

- Random Sampling
- Approximation Algorithms
- Streaming Algorithms
- Probability Amplification

Randomised Data Structures

- Classic Hash Tables
- Cuckoo Hashing
- Bloom Filters
- Retrieval Data Structures
- Perfect Hash Functions

Conclusion

Avoiding the Worst Case with Randomness – Example: $\overline{\wedge}$-Tree Evaluation

Deterministic Algorithms:

- $\forall$Algo : $\exists$Input : Algo slow on Input.
- every algorithm is vulnerable to adversarial inputs

Our Randomised Algorithm:

- On *any* input: fast in expectation.
  on any input: slow if unlucky.
- not vulnerable to adversarial inputs

Average Case Analysis

- Model real world using probability distribution over inputs.
- In many cases random instances ...
  - ... are easier to solve than worst-case instances
    $\hookrightarrow$ NP-hard problems may be easy *on average*
  - ... admit simpler algorithms and data structures
    $\hookrightarrow$ e.g. search trees with random insertion order need no load balancing

Anhang: Mögliche Prüfungsfragen I

- Can we use randomness to improve worst-case running times?
  - In what sense?
  - What is an example?
- How can a randomized algorithm be used to verify a polynomial equation?
- How can a randomized algorithm be used to verify a matrix multiplication?
- With respect to the evaluation of $\overline{\wedge}$-trees:
  - What was our optimization goal?
  - What can be achieved with deterministic algorithms?
  - How does our randomized approach work?
  - What is its running time and why?
- What is average-case analysis, and what is it supposed to achieve?
- How do search trees behave under insertions in random order?
  - What holds for the expected depth of a node, and why?