Byzantinische Generäle

- Jede der n Divisionen der Armee von Byzanz wird von einem **loyalen** oder **illoyalen General** geführt

Byzantinische Generäle

- Jede der n Divisionen der Armee von Byzanz wird von einem **loyalen** oder **illoyalen General** geführt
- Die Anzahl der illoyalen Generäle ist $t < n/8$

Byzantinische Generäle

- Jede der n Divisionen der Armee von Byzanz wird von einem **loyalen** oder **illoyalen General** geführt
- Die Anzahl der illoyalen Generäle ist $t < n/8$
- Sie müssen sich auf **Angriff** oder **Rückzug** einigen und kommunizieren (paarweise) über Boten

Byzantinische Generäle

- Jede der n Divisionen der Armee von Byzanz wird von einem **loyalen** oder **illoyalen General** geführt
- Die Anzahl der illoyalen Generäle ist $t < n/8$
- Sie müssen sich auf **Angriff** oder **Rückzug** einigen und kommunizieren (paarweise) über Boten
- Sie stimmen in Runden ab: In jeder Runde schickt jeder General jedem anderen General sein aktuelles Votum

Byzantinische Generäle

- Jede der n Divisionen der Armee von Byzanz wird von einem **loyalen** oder **illoyalen General** geführt
- Die Anzahl der illoyalen Generäle ist $t < n/8$
- Sie müssen sich auf **Angriff** oder **Rückzug** einigen und kommunizieren (paarweise) über Boten
- Sie stimmen in Runden ab: In jeder Runde schickt jeder General jedem anderen General sein aktuelles Votum
- Ein (illoyaler) General kann verschiedene Voten an verschiedene Generäle schicken

Byzantinische Generäle

- Jede der n Divisionen der Armee von Byzanz wird von einem **loyalen** oder **illoyalen General** geführt
- Die Anzahl der illoyalen Generäle ist $t < n/8$
- Sie müssen sich auf **Angriff** oder **Rückzug** einigen und kommunizieren (paarweise) über Boten
- Sie stimmen in Runden ab: In jeder Runde schickt jeder General jedem anderen General sein aktuelles Votum
- Ein (illoyaler) General kann verschiedene Voten an verschiedene Generäle schicken
- **Ziel** loyaler General: **Einigung** der loyalen Generäle

  Ziel illoyaler General: **keine Einigung** der loyalen Generäle

Formalisierung für verteilte Systeme

Formalisierung für verteilte Systeme

Eingabe:

- Netzwerk mit n Prozessoren, wobei $t < n/8$ fehlerhaft sind

Formalisierung für verteilte Systeme

Eingabe:

- Netzwerk mit n Prozessoren, wobei $t < n/8$ fehlerhaft sind
- Die Prozessoren starten mit je einem Eingabebit

Formalisierung für verteilte Systeme

Eingabe:

- Netzwerk mit n Prozessoren, wobei $t < n/8$ fehlerhaft sind
- Die Prozessoren starten mit je einem Eingabebit

Aufgabe:

(1) Die fehlerfreien Prozessoren müssen sich auf ein Bit einigen

Formalisierung für verteilte Systeme

Eingabe:

- Netzwerk mit n Prozessoren, wobei $t < n/8$ fehlerhaft sind
- Die Prozessoren starten mit je einem Eingabebit

Aufgabe:

(1) Die fehlerfreien Prozessoren müssen sich auf ein Bit einigen

(2) Wenn alle fehlerfreien Prozessoren mit demselben Bit starten, dann müssen sie sich auf dieses Bit einigen;

Formalisierung für verteilte Systeme

Eingabe:

- Netzwerk mit n Prozessoren, wobei $t < n/8$ fehlerhaft sind
- Die Prozessoren starten mit je einem Eingabebit

Aufgabe:

(1) Die fehlerfreien Prozessoren müssen sich auf ein Bit einigen

(2) Wenn alle fehlerfreien Prozessoren mit demselben Bit starten, dann müssen sie sich auf dieses Bit einigen;

Regeln der Kommunikation

- Kommunikation in Runden;

Regeln der Kommunikation

- Kommunikation in Runden;
- In jeder Runde schickt jeder Prozessor sein aktuelles Votum (Bit) an jeden anderen Prozessor
- Er kann in einer Runde unterschiedliche Voten schicken:

Regeln der Kommunikation

- Kommunikation in Runden;
- In jeder Runde schickt jeder Prozessor sein aktuelles Votum (Bit) an jeden anderen Prozessor
- Er kann in einer Runde unterschiedliche Voten schicken:
  $\rightarrow$ in der gleichen Runde an unterschiedliche Prozessoren

Regeln der Kommunikation

- Kommunikation in Runden;
- In jeder Runde schickt jeder Prozessor sein aktuelles Votum (Bit) an jeden anderen Prozessor
- Er kann in einer Runde unterschiedliche Voten schicken:
  $\rightarrow$ in der gleichen Runde an unterschiedliche Prozessoren
  $\rightarrow$ in unterschiedlichen Runden an den gleichen Prozessor

Regeln der Kommunikation

- Kommunikation in Runden;
- In jeder Runde schickt jeder Prozessor sein aktuelles Votum (Bit) an jeden anderen Prozessor
- Er kann in einer Runde unterschiedliche Voten schicken:
  $\rightarrow$ in der gleichen Runde an unterschiedliche Prozessoren
  $\rightarrow$ in unterschiedlichen Runden an den gleichen Prozessor
- Vor jeder Runde können sich die fehlerhaften Prozessoren auf eine beliebig komplexe Strategie von Voten einigen

Regeln der Kommunikation

- Kommunikation in Runden;
- In jeder Runde schickt jeder Prozessor sein aktuelles Votum (Bit) an jeden anderen Prozessor
- Er kann in einer Runde unterschiedliche Voten schicken:
  $\rightarrow$ in der gleichen Runde an unterschiedliche Prozessoren
  $\rightarrow$ in unterschiedlichen Runden an den gleichen Prozessor
- Vor jeder Runde können sich die fehlerhaften Prozessoren auf eine beliebig komplexe Strategie von Voten einigen
- Die fehlerfreien Prozessoren kennen die fehlerhaften und ihre Strategien nicht

Regeln der Kommunikation

- Kommunikation in Runden;
- In jeder Runde schickt jeder Prozessor sein aktuelles Votum (Bit) an jeden anderen Prozessor
- Er kann in einer Runde unterschiedliche Voten schicken:
  $\rightarrow$ in der gleichen Runde an unterschiedliche Prozessoren
  $\rightarrow$ in unterschiedlichen Runden an den gleichen Prozessor
- Vor jeder Runde können sich die fehlerhaften Prozessoren auf eine beliebig komplexe Strategie von Voten einigen
- Die fehlerfreien Prozessoren kennen die fehlerhaften und ihre Strategien nicht

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

1. Sei b_i das Eingabebit von Prozessor i
2. $s \leftarrow 1$; $v_i(s) \leftarrow b_i$ (Votum von i in Runde s)

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

1. Sei b_i das Eingabebit von Prozessor i
2. $s \leftarrow 1$; $v_i(s) \leftarrow b_i$ (Votum von i in Runde s)
3. **repeat**
4. Schicke $v_i(s)$ an alle Prozessoren;

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

1. Sei b_i das Eingabebit von Prozessor i
2. $s \leftarrow 1$; $v_i(s) \leftarrow b_i$ (Votum von i in Runde s)
3. **repeat**
4. Schicke $v_i(s)$ an alle Prozessoren;
5. Empfange $v_j(s)$ von allen anderen. Sei Bit_i Mehrheit der Stimmen und $Stimmen_i$ Anzahl der Stimmen für Bit_i

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

1. Sei b_i das Eingabebit von Prozessor i
2. $s \leftarrow 1$; $v_i(s) \leftarrow b_i$ (Votum von i in Runde s)
3. **repeat**
4. Schicke $v_i(s)$ an alle Prozessoren;
5. Empfange $v_j(s)$ von allen anderen. Sei Bit_i Mehrheit der Stimmen und $Stimmen_i$ Anzahl der Stimmen für Bit_i
6. Erhalte das Zufallsbit $\tau(s)$
7. **if** $\tau(s) = 0$ **then** $T \leftarrow 5n/8$; **else** $T \leftarrow 6n/8$

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

1. Sei b_i das Eingabebit von Prozessor i
2. $s \leftarrow 1$; $v_i(s) \leftarrow b_i$ (Votum von i in Runde s)
3. **repeat**
4. Schicke $v_i(s)$ an alle Prozessoren;
5. Empfange $v_j(s)$ von allen anderen. Sei Bit_i Mehrheit der Stimmen und $Stimmen_i$ Anzahl der Stimmen für Bit_i
6. Erhalte das Zufallsbit $\tau(s)$
7. **if** $\tau(s) = 0$ **then** $T \leftarrow 5n/8$; **else** $T \leftarrow 6n/8$
8. **if** $Stimmen_i \geq T$ **then** $v_i(s+1) \leftarrow Bit_i$; **else** $v_i(s+1) \leftarrow 0$

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

1. Sei b_i das Eingabebit von Prozessor i
2. $s \leftarrow 1$; $v_i(s) \leftarrow b_i$ (Votum von i in Runde s)
3. **repeat**
4. Schicke $v_i(s)$ an alle Prozessoren;
5. Empfange $v_j(s)$ von allen anderen. Sei Bit_i Mehrheit der Stimmen und $Stimmen_i$ Anzahl der Stimmen für Bit_i
6. Erhalte das Zufallsbit $\tau(s)$
7. **if** $\tau(s) = 0$ **then** $T \leftarrow 5n/8$; **else** $T \leftarrow 6n/8$
8. **if** $Stimmen_i \geq T$ **then** $v_i(s+1) \leftarrow Bit_i$; **else** $v_i(s+1) \leftarrow 0$
9. $s \leftarrow s + 1$

Randomisiertes Protokoll

(Läuft auf den fehlerfreien Prozessoren, aber evtl. nicht auf den fehlerhaften)

Protokoll für Prozessor i :

1. Sei b_i das Eingabebit von Prozessor i
2. $s \leftarrow 1$; $v_i(s) \leftarrow b_i$ (Votum von i in Runde s)
3. **repeat**
4. Schicke $v_i(s)$ an alle Prozessoren;
5. Empfange $v_j(s)$ von allen anderen. Sei Bit_i Mehrheit der Stimmen und $Stimmen_i$ Anzahl der Stimmen für Bit_i
6. Erhalte das Zufallsbit $\tau(s)$
7. **if** $\tau(s) = 0$ **then** $T \leftarrow 5n/8$; **else** $T \leftarrow 6n/8$
8. **if** $Stimmen_i \geq T$ **then** $v_i(s+1) \leftarrow Bit_i$; **else** $v_i(s+1) \leftarrow 0$
9. $s \leftarrow s + 1$
10. **until** $Stimmen_i \geq 7n/8$ (endgültige Entscheidung)

Analyse

Korrektheit:

Analyse

Korrektheit:

- Wenn in einer Runde alle fehlerfreien Prozessoren dasselbe Bit schicken, dann entscheiden sich alle in dieser Runde.

Analyse

Korrektheit:

- Wenn in einer Runde alle fehlerfreien Prozessoren dasselbe Bit schicken, dann entscheiden sich alle in dieser Runde.

- Wenn sich mindestens einer in Runde s entscheidet, dann entscheiden sich alle bis Runde $s + 1$ für dasselbe Bit.

Analyse

Korrektheit:

- Wenn in einer Runde alle fehlerfreien Prozessoren dasselbe Bit schicken, dann entscheiden sich alle in dieser Runde.

- Wenn sich mindestens einer in Runde s entscheidet, dann entscheiden sich alle bis Runde $s + 1$ für dasselbe Bit.

Erwartete Anzahl der Runden:

Analyse

Korrektheit:

- Wenn in einer Runde alle fehlerfreien Prozessoren dasselbe Bit schicken, dann entscheiden sich alle in dieser Runde.

- Wenn sich mindestens einer in Runde s entscheidet, dann entscheiden sich alle bis Runde $s + 1$ für dasselbe Bit.

Erwartete Anzahl der Runden:

- Wenn für **mindestens einen** fehlerfreien Prozessor $Stimmen_i \geq 6n/8$, dann gilt mit W.keit mind. $1/2$:
  In der nächsten Runde schicken alle fehlerfreien Prozessoren dasselbe Bit.

Analyse

Korrektheit:

- Wenn in einer Runde alle fehlerfreien Prozessoren dasselbe Bit schicken, dann entscheiden sich alle in dieser Runde.

- Wenn sich mindestens einer in Runde s entscheidet, dann entscheiden sich alle bis Runde $s + 1$ für dasselbe Bit.

Erwartete Anzahl der Runden:

- Wenn für **mindestens einen** fehlerfreien Prozessor $Stimmen_i \geq 6n/8$, dann gilt mit W.keit mind. $1/2$:
  In der nächsten Runde schicken alle fehlerfreien Prozessoren dasselbe Bit.

- Wenn für **alle** fehlerfreien Prozessoren $Stimmen_i < 6n/8$, dann gilt mit W.keit mind. $1/2$:
  In der nächsten Runde schicken alle fehlerfreien Prozessoren $Bit_i = 0$.

Analyse

Korrektheit:

- Wenn in einer Runde alle fehlerfreien Prozessoren dasselbe Bit schicken, dann entscheiden sich alle in dieser Runde.

- Wenn sich mindestens einer in Runde s entscheidet, dann entscheiden sich alle bis Runde $s + 1$ für dasselbe Bit.

Erwartete Anzahl der Runden:

- Wenn für **mindestens einen** fehlerfreien Prozessor $Stimmen_i \geq 6n/8$, dann gilt mit W.keit mind. $1/2$:
  In der nächsten Runde schicken alle fehlerfreien Prozessoren dasselbe Bit.

- Wenn für **alle** fehlerfreien Prozessoren $Stimmen_i < 6n/8$, dann gilt mit W.keit mind. $1/2$:
  In der nächsten Runde schicken alle fehlerfreien Prozessoren $Bit_i = 0$.

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

1. Sei $P_0 = P$ und $i = 0$

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

1. Sei $P_0 = P$ und $i = 0$
2. **while** $P_i \neq \emptyset$ **do**

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

1. Sei $P_0 = P$ und $i = 0$
2. **while** $P_i \neq \emptyset$ **do**
3. Wähle Punkt $p \in P_i$ zufällig gleichverteilt
4. Sei $\delta_i \leftarrow \delta(p) \leftarrow \min_{p_j \neq p} d(p, p_j)$

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

1. Sei $P_0 = P$ und $i = 0$
2. **while** $P_i \neq \emptyset$ **do**
3. Wähle Punkt $p \in P_i$ zufällig gleichverteilt
4. Sei $\delta_i \leftarrow \delta(p) \leftarrow \min_{p_j \neq p} d(p, p_j)$
5. In einem Gitter mit Seitenlänge $\delta_i/3$ entferne alle **isolierten** Punkte. Sei P_{i+1} die restliche Punktmenge

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

1. Sei $P_0 = P$ und $i = 0$
2. **while** $P_i \neq \emptyset$ **do**
3. Wähle Punkt $p \in P_i$ zufällig gleichverteilt
4. Sei $\delta_i \leftarrow \delta(p) \leftarrow \min_{p_j \neq p} d(p, p_j)$
5. In einem Gitter mit Seitenlänge $\delta_i/3$ entferne alle **isolierten** Punkte. Sei P_{i+1} die restliche Punktmenge
6. $i \leftarrow i + 1$

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

1. Sei $P_0 = P$ und $i = 0$
2. **while** $P_i \neq \emptyset$ **do**
3. Wähle Punkt $p \in P_i$ zufällig gleichverteilt
4. Sei $\delta_i \leftarrow \delta(p) \leftarrow \min_{p_j \neq p} d(p, p_j)$
5. In einem Gitter mit Seitenlänge $\delta_i/3$ entferne alle **isolierten** Punkte. Sei P_{i+1} die restliche Punktmenge
6. $i \leftarrow i + 1$
7. Sei $k \leftarrow i - 1$ und δ_k der letzte Minimalabstand
8. In einem Gitter mit Seitenlänge δ_k bestimme für jeden Eingabepunkt seinen Minimalabstand von allen Punkten in der selben und den 8 benachbarten Zellen

Randomisierter Algorithmus für Closest-Pair

Eingabe: $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$

1. Sei $P_0 = P$ und $i = 0$
2. **while** $P_i \neq \emptyset$ **do**
3. Wähle Punkt $p \in P_i$ zufällig gleichverteilt
4. Sei $\delta_i \leftarrow \delta(p) \leftarrow \min_{p_j \neq p} d(p, p_j)$
5. In einem Gitter mit Seitenlänge $\delta_i/3$ entferne alle **isolierten** Punkte. Sei P_{i+1} die restliche Punktmenge
6. $i \leftarrow i + 1$
7. Sei $k \leftarrow i - 1$ und δ_k der letzte Minimalabstand
8. In einem Gitter mit Seitenlänge δ_k bestimme für jeden Eingabepunkt seinen Minimalabstand von allen Punkten in der selben und den 8 benachbarten Zellen
9. **return** globalen Minimalabstand μ

Das Yao-Prinzip

Das Yao-Prinzip

allgemeine untere Schranken für randomisierte Algorithmen

Das Yao-Prinzip

allgemeine untere Schranken für randomisierte Algorithmen

Theorem:

Jeder randomisierte online Algorithmus für Paging mit Cache-Größe k hat Wettbewerbsfaktor mindestens

$$H_k = 1 + 1/2 + 1/3 + \ldots + 1/k.$$

Das Yao-Prinzip

allgemeine untere Schranken für randomisierte Algorithmen

Theorem:

Jeder randomisierte online Algorithmus für Paging mit Cache-Größe k hat Wettbewerbsfaktor mindestens

$$H_k = 1 + 1/2 + 1/3 + \ldots + 1/k.$$

Der Beweis verwendet das Yao-Prinzip:

Das Yao-Prinzip

allgemeine untere Schranken für randomisierte Algorithmen

Theorem:

Jeder randomisierte online Algorithmus für Paging mit Cache-Größe k hat Wettbewerbsfaktor mindestens

$$H_k = 1 + 1/2 + 1/3 + \ldots + 1/k.$$

Der Beweis verwendet das Yao-Prinzip:

Statt der **schlechtesten Instanz** für die
beste Verteilung über det. Algorithmen (= rand. Alg.)

Das Yao-Prinzip

allgemeine untere Schranken für randomisierte Algorithmen

Theorem:

Jeder randomisierte online Algorithmus für Paging mit Cache-Größe k hat Wettbewerbsfaktor mindestens

$$H_k = 1 + 1/2 + 1/3 + \ldots + 1/k.$$

Der Beweis verwendet das Yao-Prinzip:

Statt der **schlechtesten Instanz** für die
beste Verteilung über det. Algorithmen (= rand. Alg.)

betrachte den **besten det. Algorithmus** für die
schlechteste Verteilung über Instanzen.

Lemma

Lemma

mit dem Yao-Prinzip folgt:

Lemma

mit dem Yao-Prinzip folgt:

Lemma:

Wenn für irgendeine Verteilung π über (allen) Eingabefolgen σ gilt, dass für jeden deterministischen Algorithmus ALG

Lemma

mit dem Yao-Prinzip folgt:

Lemma:

Wenn für irgendeine Verteilung π über (allen) Eingabefolgen σ gilt, dass für jeden deterministischen Algorithmus ALG

$$\mathbb{E}_\pi[ALG(\sigma)] \quad \geq \quad H_k \cdot \mathbb{E}_\pi[OPT(\sigma)]$$

dann hat jeder randomisierte online Algorithmus für Paging Wettbewerbsfaktor mindestens H_k.

Hintergrund zum Yao-Prinzip

Hintergrund zum Yao-Prinzip

Bei randomisierten Strategien der Spieler

(Alg. Designer vs. Bösartiger Gegner) gelten:

Hintergrund zum Yao-Prinzip

Bei randomisierten Strategien der Spieler
(Alg. Designer vs. Bösartiger Gegner) gelten:

(1) Der Spieler der als Zweiter seine Strategie wählt, kann genauso gut eine deterministische Strategie wählen.

Hintergrund zum Yao-Prinzip

Bei randomisierten Strategien der Spieler

(Alg. Designer vs. Bösartiger Gegner) gelten:

(1) Der Spieler der als Zweiter seine Strategie wählt, kann genauso gut eine deterministische Strategie wählen.

(2) (Minimax-Theorem:) Bei randomisierten Strategien ist egal, wer als Erster seine Strategie wählt, weil

Hintergrund zum Yao-Prinzip

Bei randomisierten Strategien der Spieler
(Alg. Designer vs. Bösartiger Gegner) gelten:

(1) Der Spieler der als Zweiter seine Strategie wählt, kann genauso gut eine deterministische Strategie wählen.

(2) (Minimax-Theorem:) Bei randomisierten Strategien ist egal, wer als Erster seine Strategie wählt, weil

$$\min_a \max_b \quad a^T \cdot M \cdot b \qquad = \qquad \max_a \min_b \quad a^T \cdot M \cdot b$$

Hintergrund zum Yao-Prinzip

Hintergrund zum Yao-Prinzip

Der erwartete Spielwert ist gleich für...

Hintergrund zum Yao-Prinzip

Der erwartete Spielwert ist gleich für...

schlechteste Instanz

$\rightarrow$ für beste Verteilung über det. Algorithmen

Hintergrund zum Yao-Prinzip

Der erwartete Spielwert ist gleich für...

schlechteste Instanz

$\rightarrow$ für beste Verteilung über det. Algorithmen

schlechteste Verteilung über Instanzen

$\rightarrow$ für beste Verteilung über det. Algorithmen

Hintergrund zum Yao-Prinzip

Der erwartete Spielwert ist gleich für...

schlechteste Instanz

$\rightarrow$ für beste Verteilung über det. Algorithmen

schlechteste Verteilung über Instanzen

$\rightarrow$ für beste Verteilung über det. Algorithmen

beste Verteilung über det. Algorithmen

$\rightarrow$ für schlechteste Verteilung über Instanzen

Hintergrund zum Yao-Prinzip

Der erwartete Spielwert ist gleich für...

schlechteste Instanz

$\rightarrow$ für beste Verteilung über det. Algorithmen

schlechteste Verteilung über Instanzen

$\rightarrow$ für beste Verteilung über det. Algorithmen

beste Verteilung über det. Algorithmen

$\rightarrow$ für schlechteste Verteilung über Instanzen

bester det. Algorithmus

$\rightarrow$ für schlechteste Verteilung über Instanzen

Hintergrund zum Yao-Prinzip

Der erwartete Spielwert ist gleich für...

schlechteste Instanz

$\rightarrow$ für beste Verteilung über det. Algorithmen

schlechteste Verteilung über Instanzen

$\rightarrow$ für beste Verteilung über det. Algorithmen

beste Verteilung über det. Algorithmen

$\rightarrow$ für schlechteste Verteilung über Instanzen

bester det. Algorithmus

$\rightarrow$ für schlechteste Verteilung über Instanzen