CS 577 - Randomized Algorithms

Marc Renault

Department of Computer Sciences
University of Wisconsin – Madison

Spring 2023
TopHat Section 001 Join Code: 020205
TopHat Section 002 Join Code: 394523

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

QuickSort

Recall: Linear Time Selection

Problem

Find the kth value in an unsorted array A of n numbers if A were sorted.

Recall: Linear Time Selection

Problem

Find the kth value in an unsorted array A of n numbers if A were sorted.

Algorithm: QuickSelect

Input : A array $A[1..n]$ and an int k.
Output: The kth element of A if A were sorted.

if $n = 1$ **then return** $A[1]$
Choose a pivot $A[p]$
$r :=$ Partition$(A[1..n], p)$
if $k < r$ **then**
  return *QuickSelect*$(A[1..r-1], k)$
else if $k > r$ **then**
  return *QuickSelect*$(A[r+1..n], k-r)$
else
  return $A[r]$
end

QuickSort

Algorithm: QuickSort

Input : An array $A[1..n]$.

Output: A sorted from 1 to n.

Choose a pivot $A[p]$

$r :=$ Partition$(A[1..n], p)$

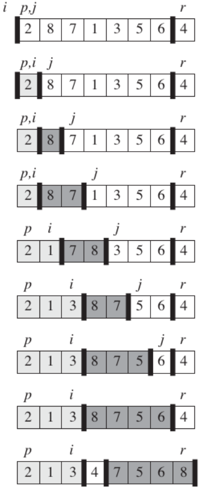
QuickSort$(A[1..r-1])$

QuickSort$(A[r+1..n])$

return A

QuickSort

QuickSort partition step:



QuickSort

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r := \textsc{Partition}(A[1..n], p)$
$\textsc{QuickSort}(A[1..r-1])$
$\textsc{QuickSort}(A[r+1..n])$
return A

Why no combine step?

QuickSort

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r :=$ Partition$(A[1..n], p)$
QuickSort$(A[1..r-1])$
QuickSort$(A[r+1..n])$
return A

Why no combine step?

Because QuickSort sorts in-place.

QuickSort

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r := \textsc{Partition}(A[1..n], p)$
$\textsc{QuickSort}(A[1..r-1])$
$\textsc{QuickSort}(A[r+1..n])$
return A

TopHat 1: What is the complexity of the partition step?

QuickSort

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r := \textsc{Partition}(A[1..n], p)$
$\textsc{QuickSort}(A[1..r-1])$
$\textsc{QuickSort}(A[r+1..n])$
return A

TopHat 1: What is the complexity of the partition step? $O(n)$.

QuickSort Analysis

Worst Case

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r := \text{Partition}(A[1..n], p)$
$\text{QuickSort}(A[1..r-1])$
$\text{QuickSort}(A[r+1..n])$
return A

TopHat 2: What is the worst-case recurrence?

QuickSort Analysis

Worst Case

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r := \textsc{Partition}(A[1..n], p)$
$\textsc{QuickSort}(A[1..r-1])$
$\textsc{QuickSort}(A[r+1..n])$
return A

Worst-case recurrence

$$\begin{aligned} T(n) &\leq T(n-1) + T(0) + O(n) \\ &\leq T(n-2) + 2T(0) + 2O(n) \\ &\leq n(T(0) + O(n)) \\ &= O\left(n^2\right) \end{aligned}$$

QuickSort Analysis

Best Case

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r := \textsc{Partition}(A[1..n], p)$
$\textsc{QuickSort}(A[1..r-1])$
$\textsc{QuickSort}(A[r+1..n])$
return A

TopHat 3: What is the best-case recurrence?

QuickSort Analysis

Best Case

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$
$r := \textsc{Partition}(A[1..n], p)$
$\textsc{QuickSort}(A[1..r-1])$
$\textsc{QuickSort}(A[r+1..n])$
return A

Best-case recurrence

$$\begin{aligned} T(n) &\leq 2T(n/2) + O(n) \\ &= O(n \log n) \end{aligned}$$

QuickSort Analysis

Average Case

Observation 1

For $0 < \varepsilon < 1$,

$$\begin{aligned} T(n) &= T(\varepsilon n) + T((1-\varepsilon)n) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

QuickSort Analysis

Average Case

Observation 1

For $0 < \varepsilon < 1$,

$$\begin{aligned} T(n) &= T(\varepsilon n) + T((1-\varepsilon)n) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

Probabilistic Argument

Expected Runtime:

$$T(n) \leq \Pr[\Theta(n) \text{ split}] \cdot \Theta(n \log n) + \Pr[o(n) \text{ split}] \cdot \Theta(n^2)$$

QuickSort Analysis

Average Case

Observation 1

For $0 < \varepsilon < 1$,

$$\begin{aligned} T(n) &= T(\varepsilon n) + T((1-\varepsilon)n) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

Probabilistic Argument

Expected Runtime:

$$\begin{aligned} T(n) &\le \Pr[\Theta(n) \text{ split}] \cdot \Theta(n \log n) + \Pr[o(n) \text{ split}] \cdot \Theta(n^2) \\ &= (1 - \Pr[o(n) \text{ split}]) \cdot \Theta(n \log n) + \Pr[o(n) \text{ split}] \cdot \Theta(n^2) \end{aligned}$$

QuickSort Analysis

Average Case

Observation 1

For $0 < \varepsilon < 1$,

$$\begin{aligned} T(n) &= T(\varepsilon n) + T((1-\varepsilon)n) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$

Probabilistic Argument

Expected Runtime:

$$\begin{aligned} T(n) &\leq \Pr[\Theta(n) \text{ split}] \cdot \Theta(n \log n) + \Pr[o(n) \text{ split}] \cdot \Theta(n^2) \\ &= (1 - \Pr[o(n) \text{ split}]) \cdot \Theta(n \log n) + \Pr[o(n) \text{ split}] \cdot \Theta(n^2) \\ &= \Theta(n \log n), \text{ if } \Pr[o(n) \text{ split}] = O\left(\frac{\log n}{n}\right) \end{aligned}$$

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$T(n) \leq \frac{1}{n} \sum_{i=1}^{n} (T(i-1) + T(n-i)) + O(n)$$

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$T(n) \leq \frac{1}{n}\sum_{i=1}^{n}(T(i-1)+T(n-i))+O(n)$$
$$= \frac{2}{n}\sum_{i=1}^{n}(T(i-1))+O(n)$$

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$
\begin{aligned}
T(n) &\leq \frac{1}{n}\sum_{i=1}^{n}(T(i-1)+T(n-i))+O(n) \\
&= \frac{2}{n}\sum_{i=1}^{n}(T(i-1))+O(n)=O(n\log n)
\end{aligned}
$$

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$
\begin{aligned}
T(n) &\leq \frac{1}{n}\sum_{i=1}^{n}\left(T(i-1)+T(n-i)\right)+O(n)\\
&= \frac{2}{n}\sum_{i=1}^{n}\left(T(i-1)\right)+O(n) = O(n\log n)
\end{aligned}
$$

Uniform Assumption Realistic?

- Probably not...

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$T(n) \leq \frac{1}{n} \sum_{i=1}^{n} (T(i-1) + T(n-i)) + O(n)$$
$$= \frac{2}{n} \sum_{i=1}^{n} (T(i-1)) + O(n) = O(n \log n)$$

Uniform Assumption Realistic?

- Probably not...
- Improve QuickSort by more complicated pivot choice.

QuickSort with MomPivot

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$ using MomPivot
$r := \textsc{Partition}(A[1..n], p)$
$\textsc{QuickSort}(A[1..r-1])$
$\textsc{QuickSort}(A[r+1..n])$
return A

MomPivot Recurrence Worst-Case

$$\begin{aligned} T(n) &\leq T(7n/10) + T(3n/10) + O(n) \\ &= O(n \log n) \end{aligned}$$

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$\begin{aligned} T(n) &\leq \frac{1}{n}\sum_{i=1}^{n}(T(i-1)+T(n-i))+O(n) \\ &= \frac{2}{n}\sum_{i=1}^{n}(T(i-1))+O(n)=O(n\log n) \end{aligned}$$

Uniform Assumption Realistic?

- Probably not...

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$
\begin{aligned}
T(n) &\leq \frac{1}{n} \sum_{i=1}^{n} \left(T(i-1) + T(n-i)\right) + O(n) \\
&= \frac{2}{n} \sum_{i=1}^{n} \left(T(i-1)\right) + O(n) = O(n \log n)
\end{aligned}
$$

Uniform Assumption Realistic?

- Probably not...
- Improve QuickSort by more complicated pivot choice.

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$
\begin{aligned}
T(n) &\leq \frac{1}{n} \sum_{i=1}^{n} (T(i-1) + T(n-i)) + O(n) \\
&= \frac{2}{n} \sum_{i=1}^{n} (T(i-1)) + O(n) = O(n \log n)
\end{aligned}
$$

Uniform Assumption Realistic?

- Probably not...
- Improve QuickSort by more complicated pivot choice.
- What would be an easy way to get this average case performance?

QuickSort Analysis

Average Case

Average Case Recurrence (uniform dist on orderings)

$$
\begin{aligned}
T(n) &\le \frac{1}{n}\sum_{i=1}^{n}\left(T(i-1)+T(n-i)\right)+O(n)\\
&= \frac{2}{n}\sum_{i=1}^{n}\left(T(i-1)\right)+O(n)=O(n\log n)
\end{aligned}
$$

Uniform Assumption Realistic?

- Probably not...
- Improve QuickSort by more complicated pivot choice.
- What would be an easy way to get this average case performance? UAR choose the pivot.

Randomized Algorithms

Randomization and Algorithms

Random Input

- Average Case analysis:
  - Input is drawn from some distribution π.
  - Under distribution π, average run-time, memory, etc...
- We saw an example when we analyzed QuickSort for a uniform distribution.

Randomization and Algorithms

Random Input

- Average Case analysis:
  - Input is drawn from some distribution π.
  - Under distribution π, average run-time, memory, etc...
- We saw an example when we analyzed QuickSort for a uniform distribution.

Randomized Algorithms

- Algorithm flips a coin to make some decisions.

Randomization and Algorithms

Random Input

- Average Case analysis:
  - Input is drawn from some distribution π.
  - Under distribution π, average run-time, memory, etc...
- We saw an example when we analyzed QuickSort for a uniform distribution.

Randomized Algorithms

- Algorithm flips a coin to make some decisions.
- Non-Deterministic: simultaneously considers multiple algorithms weighted by the probability distribution.

Randomized Algorithms

Types of Randomized Algorithms:

Monte Carlo

- With probability p returns the correct answer:

Randomized Algorithms

Types of Randomized Algorithms:

Monte Carlo

- With probability p returns the correct answer:
  - Run multiple times to boost the probability of correct answer.

Randomized Algorithms

Types of Randomized Algorithms:

Monte Carlo

- With probability p returns the correct answer:
  - Run multiple times to boost the probability of correct answer.
  - Provide an approximation guarantee in expectation.

Randomized Algorithms

Types of Randomized Algorithms:

Monte Carlo

- With probability p returns the correct answer:
  - Run multiple times to boost the probability of correct answer.
  - Provide an approximation guarantee in expectation.

Las Vegas

- Always returns the correct solution, or informs about failure.
- Has a run-time that is polynomial in expectation.

Randomized Algorithms

Types of Randomized Algorithms:

Monte Carlo

- With probability p returns the correct answer:
  - Run multiple times to boost the probability of correct answer.
  - Provide an approximation guarantee in expectation.

Las Vegas

- Always returns the correct solution, or informs about failure.
- Has a run-time that is polynomial in expectation.

Atlantic City

- Probabilistic run-time and correctness.

Randomization and Approximation

Guarantee in Expectation

Returns a solution that has a r approximation ratio in expectation:

$$\forall I, \mathbb{E}[\textsc{alg}(I)] \leq r \cdot \textsc{opt}(I) + \eta$$

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.
- Total probability mass is 1: $\sum_{i \in \Omega} p(i) = 1$.

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.
- Total probability mass is 1: $\sum_{i \in \Omega} p(i) = 1$.

Probability Event

- An event ε is a set of outcomes of Ω.

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.
- Total probability mass is 1: $\sum_{i \in \Omega} p(i) = 1$.

Probability Event

- An event ε is a set of outcomes of Ω.
- $\Pr[\varepsilon] = \sum_{i \in \varepsilon} p(i)$.
- Note: $\Pr[\overline{\varepsilon}] = 1 - \Pr[\varepsilon]$

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.
- Total probability mass is 1: $\sum_{i \in \Omega} p(i) = 1$.

Probability Event

- An event ε is a set of outcomes of Ω.
- $\Pr[\varepsilon] = \sum_{i \in \varepsilon} p(i)$.
- Note: $\Pr[\overline{\varepsilon}] = 1 - \Pr[\varepsilon]$

TH: $\Pr[\text{Roll 1 on a fair D4}] =$

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.
- Total probability mass is 1: $\sum_{i \in \Omega} p(i) = 1$.

Probability Event

- An event ε is a set of outcomes of Ω.
- $\Pr[\varepsilon] = \sum_{i \in \varepsilon} p(i)$.
- Note: $\Pr[\overline{\varepsilon}] = 1 - \Pr[\varepsilon]$

TH: Pr[Roll 1 on a fair D4] = 1/4

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.
- Total probability mass is 1: $\sum_{i \in \Omega} p(i) = 1$.

Probability Event

- An event ε is a set of outcomes of Ω.
- $\Pr[\varepsilon] = \sum_{i \in \varepsilon} p(i)$.
- Note: $\Pr[\overline{\varepsilon}] = 1 - \Pr[\varepsilon]$

TH: Pr[Roll 1 on a fair D4] = 1/4;
TH: Pr[Roll 2, 3, or 4 on a fair D4] =

Probability Review / Primer

Probability Space

- *Sample space* Ω of all possible outcomes.
  - Can be infinite, but we will focus on finite.
  - Ex: 4-sided die (D4): $\Omega = \{1, 2, 3, 4\}$.
- *Probability mass*: each $i \in \Omega$ has a nonnegative probability mass: $1 \geq p(i) \geq 0$.
- Total probability mass is 1: $\sum_{i \in \Omega} p(i) = 1$.

Probability Event

- An event ε is a set of outcomes of Ω.
- $\Pr[\varepsilon] = \sum_{i \in \varepsilon} p(i)$.
- Note: $\Pr[\overline{\varepsilon}] = 1 - \Pr[\varepsilon]$

TH: Pr[Roll 1 on a fair D4] = 1/4;
TH: Pr[Roll 2, 3, or 4 on a fair D4] = 3/4

Conditional Probability and Independence

Conditional Probability

Probability of ε given $\mathcal{F}$.

$$\Pr[\varepsilon|\mathcal{F}] = \frac{\Pr[\varepsilon \cap \mathcal{F}]}{\Pr[\mathcal{F}]}$$

Conditional Probability and Independence

Conditional Probability

Probability of ε given $\mathcal{F}$.

$$\Pr[\varepsilon|\mathcal{F}] = \frac{\Pr[\varepsilon \cap \mathcal{F}]}{\Pr[\mathcal{F}]}$$

Independent Events

- Events ε and $\mathcal{F}$ are independent if $\Pr[\varepsilon|\mathcal{F}] = \Pr[\varepsilon]$ and $\Pr[\mathcal{F}|\varepsilon] = \Pr[\mathcal{F}]$.

Conditional Probability and Independence

Conditional Probability

Probability of ε given $\mathcal{F}$.

$$\Pr[\varepsilon|\mathcal{F}] = \frac{\Pr[\varepsilon \cap \mathcal{F}]}{\Pr[\mathcal{F}]}$$

Independent Events

- Events ε and $\mathcal{F}$ are independent if $\Pr[\varepsilon|\mathcal{F}] = \Pr[\varepsilon]$ and $\Pr[\mathcal{F}|\varepsilon] = \Pr[\mathcal{F}]$.
- This implies $\Pr[\varepsilon \cap \mathcal{F}] = \Pr[\varepsilon] \cdot \Pr[\mathcal{F}]$.

Conditional Probability and Independence

Conditional Probability

Probability of ε given $\mathcal{F}$.

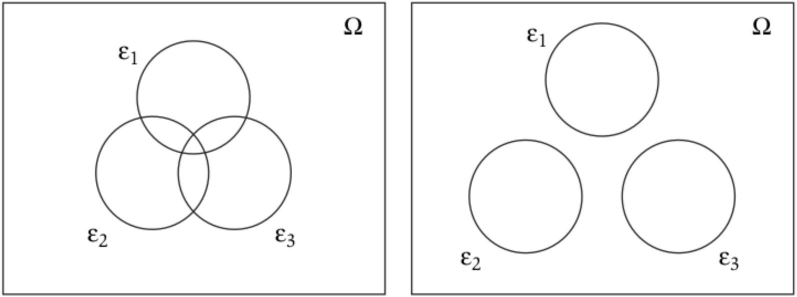
$$\Pr[\varepsilon|\mathcal{F}] = \frac{\Pr[\varepsilon \cap \mathcal{F}]}{\Pr[\mathcal{F}]}$$

Independent Events

- Events ε and $\mathcal{F}$ are independent if $\Pr[\varepsilon|\mathcal{F}] = \Pr[\varepsilon]$ and $\Pr[\mathcal{F}|\varepsilon] = \Pr[\mathcal{F}]$.
- This implies $\Pr[\varepsilon \cap \mathcal{F}] = \Pr[\varepsilon] \cdot \Pr[\mathcal{F}]$.
- Generalization: Say $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ are independent.

$$\Pr\left[\bigcap_{i=1}^{n} \varepsilon_i\right] = \prod_{i=1}^{n} \Pr[\varepsilon_i]$$

Union Bound



Union Bound

$$\Pr\left[\bigcup_{i=1}^{n} \varepsilon_i\right] \leq \sum_{i=1}^{n} \Pr[\varepsilon_i],$$

where equality *only* if events are mutually exclusive.

Random Variables and Expectation

Random Variables

- Technical: Given a probability space, a random variable X is a function from the sample space to the natural (finite – real if infinite) numbers, such that, for number j, $X^{-1}(j)$ is the set of all sample points taking the value j is an event.

Ex: $\Pr[X = 1] = 1/4$, where X is a toss of a 4-sided die.

Random Variables and Expectation

Random Variables

- Informally: A random variable X takes on a value that depends on a random process.

Ex: $\Pr[X = 1] = 1/4$, where X is a toss of a 4-sided die.

Random Variables and Expectation

Random Variables

- Informally: A random variable X takes on a value that depends on a random process.

Ex: $\Pr[X = 1] = 1/4$, where X is a toss of a 4-sided die.

Expected Value

- "Weighted average value"
- $\mathbb{E}[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$

Random Variables and Expectation

Random Variables

- Informally: A random variable X takes on a value that depends on a random process.

Ex: $\Pr[X = 1] = 1/4$, where X is a toss of a 4-sided die.

Expected Value

- "Weighted average value"
- $\mathbb{E}[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$

TH: What is $\mathbb{E}[X]$, where X is a toss of a 4-sided die?

Random Variables and Expectation

Random Variables

- Informally: A random variable X takes on a value that depends on a random process.

Ex: $\Pr[X = 1] = 1/4$, where X is a toss of a 4-sided die.

Expected Value

- "Weighted average value"
- $\mathbb{E}[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$

TH: What is $\mathbb{E}[X]$, where X is a toss of a 4-sided die? 2.5

Random Variables and Expectation

Expected Value

- "Weighted average value"
- $\mathbb{E}[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$

TH: What is $\mathbb{E}[X]$, where X is a toss of a 4-sided die? 2.5

Expectation Properties

Let X and Y be random variables, and a be a constant.

- Linearity of expectation:
  - $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
  - $\mathbb{E}[aX] = a\,\mathbb{E}[X]$

Random Variables and Expectation

Expected Value

- "Weighted average value"
- $\mathbb{E}[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$

TH: What is $\mathbb{E}[X]$, where X is a toss of a 4-sided die? 2.5

Expectation Properties

Let X and Y be random variables, and a be a constant.

- Linearity of expectation:
    - $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
    - $\mathbb{E}[aX] = a\,\mathbb{E}[X]$
- If X and Y are independent, $\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$.

Random QuickSort

QuickSort with Random Pivot

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$ UAR
$r := \text{Partition}(A[1..n], p)$
$\text{QuickSort}(A[1..r-1])$
$\text{QuickSort}(A[r+1..n])$
return A

QuickSort with Random Pivot

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$ UAR
$r := \text{Partition}(A[1..n], p)$
$\text{QuickSort}(A[1..r-1])$
$\text{QuickSort}(A[r+1..n])$
return A

Expected Runtime (Pivot UAR)

$$
\begin{aligned}
T(n) &\leq \frac{1}{n} \sum_{i=1}^{n} (T(i-1) + T(n-i)) + O(n) \\
&= \frac{2}{n} \sum_{i=1}^{n} (T(i-1)) + O(n) = O(n \log n)
\end{aligned}
$$

QuickSort with Random Pivot

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$ UAR
$r := \text{Partition}(A[1..n], p)$
$\text{QuickSort}(A[1..r-1])$
$\text{QuickSort}(A[r+1..n])$
return A

Expected Runtime (Pivot UAR)

$$T(n) = \frac{2}{n} \sum_{i=1}^{n} (T(i-1)) + O(n) = O(n \log n)$$

TH: What kind of randomized algorithm is this?

QuickSort with Random Pivot

Algorithm: QuickSort

Input : An array $A[1..n]$.
Output: A sorted from 1 to n.
Choose a pivot $A[p]$ UAR
$r := \text{Partition}(A[1..n], p)$
$\text{QuickSort}(A[1..r-1])$
$\text{QuickSort}(A[r+1..n])$
return A

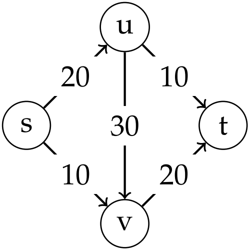
Expected Runtime (Pivot UAR)

$$T(n) = \frac{2}{n}\sum_{i=1}^{n}(T(i-1)) + O(n) = O(n\log n)$$

TH: What kind of randomized algorithm is this? Las Vegas

Min-Cut

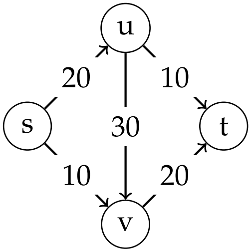
Random Min-Cut



Why?

- We saw a polynomial time algorithm (flows).

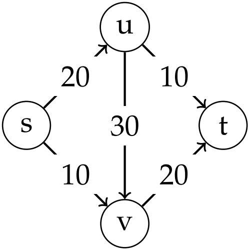
Random Min-Cut



Why?

- We saw a polynomial time algorithm (flows).
- Because:
  - Nice example of a Monte Carlo algorithm.
  - Has a good run-time for dense graphs.

Random Min-Cut



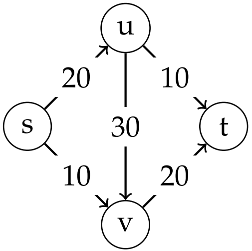
Why?

- We saw a polynomial time algorithm (flows).
- Because:
  - Nice example of a Monte Carlo algorithm.
  - Has a good run-time for dense graphs.

Min-Cut

- A Cut: Partition of V into sets (A, B) with $s \in A$ and $t \in B$.

Random Min-Cut



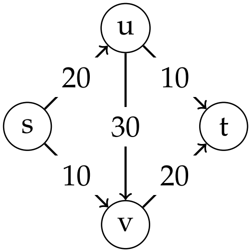
Why?

- We saw a polynomial time algorithm (flows).
- Because:
  - Nice example of a Monte Carlo algorithm.
  - Has a good run-time for dense graphs.

Min-Cut

- A Cut: Partition of V into sets (A, B) with $s \in A$ and $t \in B$.
- Cut capacity: $c(A, B) = \sum_{e \text{ out of } A} c_e$

Random Min-Cut



Why?

- We saw a polynomial time algorithm (flows).
- Because:
  - Nice example of a Monte Carlo algorithm.
  - Has a good run-time for dense graphs.

Min-Cut

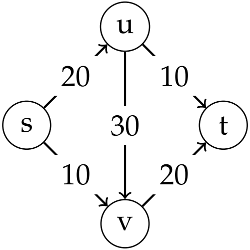
- A Cut: Partition of V into sets (A, B) with $s \in A$ and $t \in B$.
- Cut capacity: $c(A, B) = \sum_{e \text{ out of } A} c_e$
- Minimum-cut of G: The cut (A^*, B^*) that minimizes $c(A^*, B^*)$ for G.

Global Min-Cut

Some Notations

- Global meaning for any (s, t) pair.

Global Min-Cut

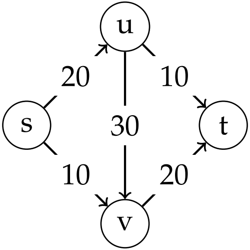


TH: What is the global min-cut?

Some Notations

- Global meaning for any (s, t) pair.

Global Min-Cut



TH: What is the global min-cut? 0

Some Notations

- Global meaning for any (s, t) pair.

Global Min-Cut

Some Notations

- Global meaning for any (s, t) pair.
- An undirected multigraph $G = (V, E)$:

Global Min-Cut

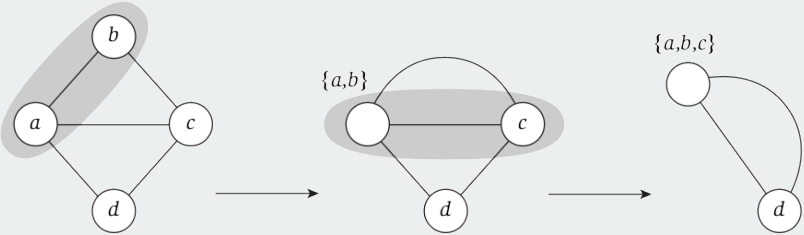
Some Notations

- Global meaning for any (s, t) pair.
- An undirected multigraph $G = (V, E)$:
- Every edge has capacity 1.
    - E is a multiset: (u, v) might be in E more than once.

Global Min-Cut

Some Notations

- Global meaning for any (s, t) pair.
- An undirected multigraph $G = (V, E)$:
- Every edge has capacity 1.
  - E is a multiset: (u, v) might be in E more than once.
- (u, v) edge contraction:
  - create a supernode $\{u, v\}$



Karger's Algorithm

Algorithm: Contraction Algorithm

Input : Multigraph $G = (V, E)$
Output: Edge set representing a cut.

```
if G has exactly 2 nodes u and v then
    return the set of edges between u and v
else
    Choose an edge (u, v) uniformly at random.
    G' := G after contracting (u, v).
    return Contraction Algorithm(G')
end
```

Karger's Algorithm

Algorithm: Contraction Algorithm

Input : Multigraph $G = (V, E)$

Output: Edge set representing a cut.

if *G has exactly 2 nodes u and v* **then**

  return *the set of edges between u and v*

else

  Choose an edge (u, v) uniformly at random.

  $G' := G$ after contracting (u, v).

  return *Contraction Algorithm*(G')

end

TH: Will this algorithm always return the correct answer?

Karger's Algorithm

Algorithm: Contraction Algorithm

Input : Multigraph $G = (V, E)$

Output: Edge set representing a cut.

if *G has exactly 2 nodes u and v* **then**

  return *the set of edges between u and v*

else

  Choose an edge (u, v) uniformly at random.

  $G' := G$ after contracting (u, v).

  return *Contraction Algorithm*(G')

end

TH: Will this algorithm always return the correct answer? No

Karger's Algorithm

Algorithm: Contraction Algorithm

Input : Multigraph $G = (V, E)$
Output: Edge set representing a cut.

if *G has exactly 2 nodes u and v* **then**
  return *the set of edges between u and v*
else
  Choose an edge (u, v) uniformly at random.
  $G' := G$ after contracting (u, v).
  return *Contraction Algorithm*(G')
end

TH: Will this algorithm always return the correct answer? No
TH: What kind of randomized algorithm is this?

Karger's Algorithm

Algorithm: Contraction Algorithm

Input : Multigraph $G = (V, E)$

Output: Edge set representing a cut.

if *G has exactly 2 nodes u and v* **then**

  return *the set of edges between u and v*

else

  Choose an edge (u, v) uniformly at random.

  $G' := G$ after contracting (u, v).

  return *Contraction Algorithm*(G')

end

TH: Will this algorithm always return the correct answer? No

TH: What kind of randomized algorithm is this? Monte Carlo

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Suppose that the global min-cut (A, B) has a size of k, and let F be the edge set.

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Suppose that the global min-cut (A, B) has a size of k, and let F be the edge set.
- Every node has degree $\geq k \implies |E| \geq \frac{1}{2}kn$.

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Suppose that the global min-cut (A, B) has a size of k, and let F be the edge set.
- Every node has degree $\geq k \implies |E| \geq \frac{1}{2}kn$.
- $\Pr[\text{Edge in } F \text{ is contracted at step } 1] \leq \frac{k}{\frac{1}{2}kn} = \frac{2}{n}$.

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Suppose that the global min-cut (A, B) has a size of k, and let F be the edge set.
- Every node has degree $\geq k \implies |E| \geq \frac{1}{2}kn$.
- $\Pr[\text{1st edge in } F \text{ is contracted at step } i] \leq \frac{k}{\frac{1}{2}k(n-(i-1))} = \frac{2}{n-i+1}$.
    - Conditioned on no edge from F having been previously contracted.

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Suppose that the global min-cut (A, B) has a size of k, and let F be the edge set.
- Every node has degree $\geq k \implies |E| \geq \frac{1}{2}kn$.
- $\Pr[\text{1st edge in } F \text{ is contracted at step } i] \leq \frac{k}{\frac{1}{2}k(n-(i-1))} = \frac{2}{n-i+1}$.
  - Conditioned on no edge from F having been previously contracted.
- There are $n-2$ steps in Contraction Algorithm.

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Let ε_i be the event that an edge $\in F$ is <u>not</u> contracted at step i:

$$\Pr[\text{success}] = \Pr[\varepsilon_1] \cdot \Pr[\varepsilon_2|\varepsilon_1] \cdots \Pr[\varepsilon_{n-2}|\varepsilon_1 \cap \varepsilon_2 \cap \cdots \cap \varepsilon_{n-3}]$$

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Let ε_i be the event that an edge $\in F$ is not contracted at step i:

$$\Pr[\text{success}] = \Pr[\varepsilon_1] \cdot \Pr[\varepsilon_2 | \varepsilon_1] \cdots \Pr[\varepsilon_{n-2} | \varepsilon_1 \cap \varepsilon_2 \cap \cdots \cap \varepsilon_{n-3}]$$

$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{n-i}\right) \cdots \left(1 - \frac{2}{3}\right)$$

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Let ε_i be the event that an edge $\in F$ is <u>not</u> contracted at step i:

$$\begin{aligned}
\Pr[\text{success}] &= \Pr[\varepsilon_1] \cdot \Pr[\varepsilon_2 | \varepsilon_1] \cdots \Pr[\varepsilon_{n-2} | \varepsilon_1 \cap \varepsilon_2 \cap \cdots \cap \varepsilon_{n-3}] \\
&\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{n-i}\right) \cdots \left(1 - \frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\left(\frac{n-5}{n-3}\right) \cdots \left(\frac{2}{4}\right)\left(\frac{1}{3}\right)
\end{aligned}$$

Analysis of Karger's Algorithm

Theorem 1

The Contraction Algorithm returns a global min-cut of G with probability of at least $1/\binom{n}{2}$.

Proof.

- Let ε_i be the event that an edge $\in F$ is <u>not</u> contracted at step i:

$$
\begin{aligned}
\Pr[\text{success}] &= \Pr[\varepsilon_1] \cdot \Pr[\varepsilon_2|\varepsilon_1] \cdots \Pr[\varepsilon_{n-2}|\varepsilon_1 \cap \varepsilon_2 \cap \cdots \cap \varepsilon_{n-3}] \\
&\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\cdots\left(1 - \frac{2}{n-i}\right)\cdots\left(1 - \frac{2}{3}\right) \\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\left(\frac{n-5}{n-3}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\
&= \frac{2}{n(n-1)} = \binom{n}{2}^{-1}
\end{aligned}
$$

Multiple Runs of Contraction Algorithm

Multiple Runs

- With $\binom{n}{2}$ runs, we get:

$$\Pr[\text{failure}] \leq \left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}} \leq \frac{1}{e} \approx 0.368$$

Multiple Runs of Contraction Algorithm

Multiple Runs

- With $\binom{n}{2}$ runs, we get:

$$\Pr[\text{failure}] \leq \left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}} \leq \frac{1}{e} \approx 0.368$$

- With $\binom{n}{2} \ln n$ runs, we get:

$$\Pr[\text{failure}] \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n}$$

Hashing

Hashing

Definition

A function that converts some input value into a hash value.

- Input: A large universe of values U. Typically, assume $|U| \gg n$.
- Output: A hash value for $u \in U$ to $\{0, 1, 2, \ldots, n-1\}$.

Hashing

Definition

A function that converts some input value into a hash value.

- Input: A large universe of values U. Typically, assume $|U| \gg n$.
- Output: A hash value for $u \in U$ to $\{0, 1, 2, \ldots, n-1\}$.

Why?

Typically used to generate keys for a dictionary data structure.

Dictionary Data Structure

Dictionary

- Storage of a subset of values from U.
- A map, where the key is generated/hashed (efficiently) from the value.

Dictionary Data Structure

Dictionary

- Storage of a subset of values from U.
- A map, where the key is generated/hashed (efficiently) from the value.

Dictionary Operations

- MakeDictionary: Initializes a fresh dictionary that can maintain a subset S of U that is initially empty.

Dictionary Data Structure

Dictionary

- Storage of a subset of values from U.
- A map, where the key is generated/hashed (efficiently) from the value.

Dictionary Operations

- MakeDictionary: Initializes a fresh dictionary that can maintain a subset S of U that is initially empty.
- Insert(u): Adds $u \in U$ to the dictionary (S).

Dictionary Data Structure

Dictionary

- Storage of a subset of values from U.
- A map, where the key is generated/hashed (efficiently) from the value.

Dictionary Operations

- MakeDictionary: Initializes a fresh dictionary that can maintain a subset S of U that is initially empty.
- Insert(u): Adds $u \in U$ to the dictionary (S).
- Delete(u): Remove u from S.

Dictionary Data Structure

Dictionary

- Storage of a subset of values from U.
- A map, where the key is generated/hashed (efficiently) from the value.

Dictionary Operations

- MakeDictionary: Initializes a fresh dictionary that can maintain a subset S of U that is initially empty.
- Insert(u): Adds $u \in U$ to the dictionary (S).
- Delete(u): Remove u from S.
- Lookup(u): Determine if u is in S; if so retrieve u.

Hashing

Motivation

- The values in U may be huge. Ex: Blog posts.

Hashing

Motivation

- The values in U may be huge. Ex: Blog posts.
- Take a value $u \in U$ and build a smaller key.

Hashing

Motivation

- The values in U may be huge. Ex: Blog posts.
- Take a value $u \in U$ and build a smaller key.

Hashing

- *Hash Table*: a n-length array H to store the values.
- *Hash Function*: Map $u \in U$ to an index in H;
  $h : U \to [0..n-1]$

Hashing

Motivation

- The values in U may be huge. Ex: Blog posts.
- Take a value $u \in U$ and build a smaller key.

Hashing

- *Hash Table*: a n-length array H to store the values.
- *Hash Function*: Map $u \in U$ to an index in H;
  $h : U \to [0..n-1]$

Dictionary Hashing

- TH: Let $u, v \in U$. Say $|U| \gg n$, can $h(u) = h(v)$?

Hashing

Motivation

- The values in U may be huge. Ex: Blog posts.
- Take a value $u \in U$ and build a smaller key.

Hashing

- *Hash Table*: a n-length array H to store the values.
- *Hash Function*: Map $u \in U$ to an index in H; $h : U \to [0..n-1]$

Dictionary Hashing

- TH: Let $u, v \in U$. Say $|U| \gg n$, can $h(u) = h(v)$? Yes.
- Collision: $h(u) = h(v)$ – At $H[i]$ is a linked-list (bucket) to store any values where $h(u) = i$.

Hashing

Motivation

- The values in U may be huge. Ex: Blog posts.
- Take a value $u \in U$ and build a smaller key.

Hashing

- *Hash Table*: a n-length array H to store the values.
- *Hash Function*: Map $u \in U$ to an index in H; $h : U \to [0..n-1]$

Dictionary Hashing

- Collision: $h(u) = h(v)$ – At $H[i]$ is a linked-list (bucket) to store any values where $h(u) = i$.
- TH: Say $|S| = n$, what is the worst-case number of comparisons to $\textsc{Lookup}(u)$?

Hashing

Motivation

- The values in U may be huge. Ex: Blog posts.
- Take a value $u \in U$ and build a smaller key.

Hashing

- *Hash Table*: a n-length array H to store the values.
- *Hash Function*: Map $u \in U$ to an index in H; $h : U \to [0..n-1]$

Dictionary Hashing

- Collision: $h(u) = h(v)$ – At $H[i]$ is a linked-list (bucket) to store any values where $h(u) = i$.
- TH: Say $|S| = n$, what is the worst-case number of comparisons to $\textsc{Lookup}(u)$? $O(n)$

Hash Function Design

Good Hash Function

- Compact and efficient.
- Minimize the collisions.

Hash Function Design

Good Hash Function

- Compact and efficient.
- Minimize the collisions.

Some ideas for hash functions

- Hash as a prefix: Collisions can result from similar prefixes. E.g. many phrases in English start with “The”.

Hash Function Design

Good Hash Function

- Compact and efficient.
- Minimize the collisions.

Some ideas for hash functions

- Hash as a prefix: Collisions can result from similar prefixes. E.g. many phrases in English start with “The”.
- $u \mod n$: Risk of collision can be large especially if say n is a power of 2.

Hash Function Design

Good Hash Function

- Compact and efficient.
- Minimize the collisions.

Some ideas for hash functions

- Hash as a prefix: Collisions can result from similar prefixes. E.g. many phrases in English start with "The".
- $u \mod n$: Risk of collision can be large especially if say n is a power of 2.
- $u \mod p$, where p is a prime: Less risk than n especially if p is not tiny, but $p \approx n$.

Random Hash Function

$h(x)$: Return a value from 0 to $n-1$ UAR.

Lemma 2

Given $h(x)$, the probability that $h(u) = h(v)$ for any $u, v \in U$ is [TopHat]

Random Hash Function

$h(x)$: Return a value from 0 to $n-1$ UAR.

Lemma 2

Given $h(x)$, the probability that $h(u) = h(v)$ for any $u, v \in U$ is $1/n$.

Random Hash Function

$h(x)$: Return a value from 0 to $n-1$ UAR.

Lemma 2

Given $h(x)$, the probability that $h(u) = h(v)$ for any $u, v \in U$ is $1/n$.

Proof.

- There are n^2 possible pairs of values $(h(u), h(v)$. Exactly n of them have $h(u) = h(v)$, Hence, $\Pr[h(u) = h(v)] = \frac{n}{n^2} = \frac{1}{n}$.

Random Hash Function

$h(x)$: Return a value from 0 to $n-1$ UAR.

Lemma 2

Given $h(x)$, the probability that $h(u) = h(v)$ for any $u, v \in U$ is $1/n$.

Proof.

- There are n^2 possible pairs of values $(h(u), h(v)$. Exactly n of them have $h(u) = h(v)$, Hence, $\Pr[h(u) = h(v)] = \frac{n}{n^2} = \frac{1}{n}$.
- Alternate proof: Since $h(u)$ and $h(v)$ are independent:
  - Fix $h(u)$. What is the probability that $h(u) = h(v)$?

Random Hash Function

$h(x)$: Return a value from 0 to $n-1$ UAR.

Lemma 2

Given $h(x)$, the probability that $h(u) = h(v)$ for any $u, v \in U$ is $1/n$.

Proof.

- There are n^2 possible pairs of values $(h(u), h(v)$. Exactly n of them have $h(u) = h(v)$, Hence, $\Pr[h(u) = h(v)] = \frac{n}{n^2} = \frac{1}{n}$.
- Alternate proof: Since $h(u)$ and $h(v)$ are independent:
  - Fix $h(u)$. What is the probability that $h(u) = h(v)$?
  - $\Pr[h(v) = x | h(u) = x] = \frac{1}{n}$.

$\square$

Random Hash Function

$h(x)$: Return a value from 0 to $n-1$ UAR.

Lemma 2

Given $h(x)$, the probability that $h(u) = h(v)$ for any $u, v \in U$ is $1/n$.

Proof.

- There are n^2 possible pairs of values $(h(u), h(v)$. Exactly n of them have $h(u) = h(v)$, Hence, $\Pr[h(u) = h(v)] = \frac{n}{n^2} = \frac{1}{n}$.
- Alternate proof: Since $h(u)$ and $h(v)$ are independent:
  - Fix $h(u)$. What is the probability that $h(u) = h(v)$?
  - $\Pr[h(v) = x | h(u) = x] = \frac{1}{n}$.

$\square$

What is the problem with this random hash function?

Random Hash Function

$h(x)$: Return a value from 0 to $n-1$ UAR.

Lemma 2

Given $h(x)$, the probability that $h(u) = h(v)$ for any $u, v \in U$ is $1/n$.

Proof.

- There are n^2 possible pairs of values $(h(u), h(v)$. Exactly n of them have $h(u) = h(v)$, Hence, $\Pr[h(u) = h(v)] = \frac{n}{n^2} = \frac{1}{n}$.
- Alternate proof: Since $h(u)$ and $h(v)$ are independent:
  - Fix $h(u)$. What is the probability that $h(u) = h(v)$?
  - $\Pr[h(v) = x | h(u) = x] = \frac{1}{n}$.

□

What is the problem with this random hash function?
For a dictionary, Delete(u) and Lookup(u) won't work since $h(u)$ returns a random value!

Universal Class of Hash Functions

Randomly choosing a hash function

Definition

Let $\mathcal{H}$ be a class of functions such that:

- Universal property: For any pair of values $u, v \in U$, the probability that a randomly chosen $h \in \mathcal{H}$ has $h(u) = h(v)$ is $\leq \frac{1}{n}$.

Universal Class of Hash Functions

Randomly choosing a hash function

Definition

Let $\mathcal{H}$ be a class of functions such that:

- Universal property: For any pair of values $u, v \in U$, the probability that a randomly chosen $h \in \mathcal{H}$ has $h(u) = h(v)$ is $\leq \frac{1}{n}$.
- Each $h \in \mathcal{H}$ is represented compactly and can be computed efficiently.

Universal Class of Hash Functions

Randomly choosing a hash function

Definition

Let $\mathcal{H}$ be a class of functions such that:

- Universal property: For any pair of values $u, v \in U$, the probability that a randomly chosen $h \in \mathcal{H}$ has $h(u) = h(v)$ is $\leq \frac{1}{n}$.
- Each $h \in \mathcal{H}$ is represented compactly and can be computed efficiently.

MakeDictionary: Given $\mathcal{H}$, choose h from $\mathcal{H}$ UAR for the dictionary.

Universal Class of Hash Functions

Randomly choosing a hash function

Definition

Let $\mathcal{H}$ be a class of functions such that:

- Universal property: For any pair of values $u, v \in U$, the probability that a randomly chosen $h \in \mathcal{H}$ has $h(u) = h(v)$ is $\leq \frac{1}{n}$.
- Each $h \in \mathcal{H}$ is represented compactly and can be computed efficiently.

Theorem 3

Let $\mathcal{H}$ be a universal class of hash functions mapping U to $[0..n-1]$. Let $S \subseteq U$ be of size $\leq n$. The expected number of elements $s \in S$ where $h(s) = h(u)$ for any $u \in U$ when h is chosen UAR from $\mathcal{H}$ is ≤ 1.

Universal Class of Hash Functions

Randomly choosing a hash function

Theorem 3

Let $\mathcal{H}$ be a universal class of hash functions mapping U to $[0..n-1]$. Let $S \subseteq U$ be of size $\leq n$. The expected number of elements $s \in S$ where $h(s) = h(u)$ for any $u \in U$ when h is chosen UAR from $\mathcal{H}$ is ≤ 1.

Proof.

- Fix $u \in U$. Let X_s be a random variable that is 1 if $h(s) = h(u)$; 0 otherwise.

Universal Class of Hash Functions

Randomly choosing a hash function

Theorem 3

Let $\mathcal{H}$ be a universal class of hash functions mapping U to $[0..n-1]$. Let $S \subseteq U$ be of size $\leq n$. The expected number of elements $s \in S$ where $h(s) = h(u)$ for any $u \in U$ when h is chosen UAR from $\mathcal{H}$ is ≤ 1.

Proof.

- Fix $u \in U$. Let X_s be a random variable that is 1 if $h(s) = h(u)$; 0 otherwise.
- Let $X = \sum_{s \in S} X_s$.

Universal Class of Hash Functions

Randomly choosing a hash function

Theorem 3

Let $\mathcal{H}$ be a universal class of hash functions mapping U to $[0..n-1]$. Let $S \subseteq U$ be of size $\leq n$. The expected number of elements $s \in S$ where $h(s) = h(u)$ for any $u \in U$ when h is chosen UAR from $\mathcal{H}$ is ≤ 1.

Proof.

- Fix $u \in U$. Let X_s be a random variable that is 1 if $h(s) = h(u)$; 0 otherwise.
- Let $X = \sum_{s \in S} X_s$.

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{s \in S} X_s\right] = \sum_{s \in S} \mathbb{E}[X_s]$$

Universal Class of Hash Functions

Randomly choosing a hash function

Theorem 3

Let $\mathcal{H}$ be a universal class of hash functions mapping U to $[0..n-1]$. Let $S \subseteq U$ be of size $\leq n$. The expected number of elements $s \in S$ where $h(s) = h(u)$ for any $u \in U$ when h is chosen UAR from $\mathcal{H}$ is ≤ 1.

Proof.

- Fix $u \in U$. Let X_s be a random variable that is 1 if $h(s) = h(u)$; 0 otherwise.
- Let $X = \sum_{s \in S} X_s$.
- By linearity of expectation:

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{s \in S} X_s\right] = \sum_{s \in S} \mathbb{E}[X_s]$$

Universal Class of Hash Functions

Randomly choosing a hash function

Theorem 3

Let $\mathcal{H}$ be a universal class of hash functions mapping U to $[0..n-1]$. Let $S \subseteq U$ be of size $\leq n$. The expected number of elements $s \in S$ where $h(s) = h(u)$ for any $u \in U$ when h is chosen UAR from $\mathcal{H}$ is ≤ 1.

Proof.

- Fix $u \in U$. Let X_s be a random variable that is 1 if $h(s) = h(u)$; 0 otherwise.
- Let $X = \sum_{s \in S} X_s$.
- By linearity of expectation:

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{s \in S} X_s\right] = \sum_{s \in S} \mathbb{E}[X_s] \leq |S| \cdot \frac{1}{n} \leq 1 \, .$$

$\square$

Designing a Universal Class of Hash Functions

Defining $\mathcal{H}$

- Choose a prime $p \approx n$.

Designing a Universal Class of Hash Functions

Defining $\mathcal{H}$

- Choose a prime $p \approx n$.
- Bootstrapping: All values in U are associated with a vector coordinate $x = (x_1, x_2, \ldots, x_r)$ for some r, where $0 \leq x_i < p$.
  - $r \approx \frac{\log |U|}{\log n}$ for unique x per item in U.

Designing a Universal Class of Hash Functions

Defining $\mathcal{H}$

- Choose a prime $p \approx n$.
- Bootstrapping: All values in U are associated with a vector coordinate $x = (x_1, x_2, \ldots, x_r)$ for some r, where $0 \leq x_i < p$.
  - $r \approx \frac{\log |U|}{\log n}$ for unique x per item in U.
- Let $\mathcal{A}$ be the set of all vectors of the form $a = (a_1, a_2, \ldots, a_r)$, where $0 \leq a_i < p$.

Designing a Universal Class of Hash Functions

Defining $\mathcal{H}$

- Choose a prime $p \approx n$.
- Bootstrapping: All values in U are associated with a vector coordinate $x = (x_1, x_2, \ldots, x_r)$ for some r, where $0 \le x_i < p$.
  - $r \approx \frac{\log |U|}{\log n}$ for unique x per item in U.
- Let $\mathcal{A}$ be the set of all vectors of the form $a = (a_1, a_2, \ldots, a_r)$, where $0 \le a_i < p$.
- $\mathcal{H}$ contains $h_a(x) = \left(\sum_{i=1}^{r} a_i x_i\right) \mod p$ for all $a \in \mathcal{A}$.

Analyze our definition of $\mathcal{H}$

Lemma 4 (Technical Lemma)

For any prime p and any integer $z \not\equiv 0 \mod p$, and any two integers α, β, if $\alpha z \equiv \beta z \mod p$, then $\alpha \equiv \beta \mod p$.

Analyze our definition of $\mathcal{H}$

Lemma 4 (Technical Lemma)

For any prime p and any integer $z \not\equiv 0 \mod p$, and any two integers α, β, if $\alpha z \equiv \beta z \mod p$, then $\alpha \equiv \beta \mod p$.

Proof.

Suppose $\alpha z \equiv \beta z \mod p$:

- $\iff z(\alpha - \beta) \equiv 0 \mod p$

Analyze our definition of $\mathcal{H}$

Lemma 4 (Technical Lemma)

For any prime p and any integer $z \not\equiv 0 \mod p$, and any two integers α, β, if $\alpha z \equiv \beta z \mod p$, then $\alpha \equiv \beta \mod p$.

Proof.

Suppose $\alpha z \equiv \beta z \mod p$:

- $\iff z(\alpha - \beta) \equiv 0 \mod p$
- z is not divisible by p, so $(\alpha - \beta) \equiv 0 \mod p$.

Analyze our definition of $\mathcal{H}$

Lemma 4 (Technical Lemma)

For any prime p and any integer $z \not\equiv 0 \mod p$, and any two integers α, β, if $\alpha z \equiv \beta z \mod p$, then $\alpha \equiv \beta \mod p$.

Proof.

Suppose $\alpha z \equiv \beta z \mod p$:

- $\iff z(\alpha - \beta) \equiv 0 \mod p$
- z is not divisible by p, so $(\alpha - \beta) \equiv 0 \mod p$.
- Hence, $\alpha \equiv \beta \mod p$.

$\square$

Analyze our definition of $\mathcal{H}$

Lemma 4 (Technical Lemma)

For any prime p and any integer $z \not\equiv 0 \mod p$, and any two integers α, β, if $\alpha z \equiv \beta z \mod p$, then $\alpha \equiv \beta \mod p$.

Theorem 5

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Analyze our definition of $\mathcal{H}$

Lemma 4 (Technical Lemma)

For any prime p and any integer $z \not\equiv 0 \mod p$, and any two integers α, β, if $\alpha z \equiv \beta z \mod p$, then $\alpha \equiv \beta \mod p$.

Theorem 5

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Proof.

- Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U. ($r \approx \frac{\log |U|}{\log n}$)

Analyze our definition of $\mathcal{H}$

Lemma 4 (Technical Lemma)

For any prime p and any integer $z \not\equiv 0 \mod p$, and any two integers α, β, if $\alpha z \equiv \beta z \mod p$, then $\alpha \equiv \beta \mod p$.

Theorem 5

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Proof.

- Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U. ($r \approx \frac{\log |U|}{\log n}$)
- We need to show that $\Pr[h_a(x) = h_a(y)] \le 1/p$ for a randomly chosen $a \in \mathcal{A}$.

Analyze our definition of $\mathcal{H}$

Theorem 4

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Proof.

- Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U. ($r \approx \frac{\log |U|}{\log n}$)
- Let j be an index such that $x_j \neq y_j$.

Analyze our definition of $\mathcal{H}$

Theorem 4

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Proof.

- Let $x = (x_1, x_2, \ldots, x_r)$ and $y = (y_1, y_2, \ldots, y_r)$ be two distinct elements of U. ($r \approx \frac{\log |U|}{\log n}$)
- Let j be an index such that $x_j \neq y_j$.
- Define $a := \{\text{arb fix } a_i \text{ for } i \neq j\}$, a_j defined later.

Analyze our definition of $\mathcal{H}$

Theorem 4

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Proof.

- Let j be an index such that $x_j \neq y_j$.
- Define $a := \{\text{arb fix } a_i \text{ for } i \neq j\}$, a_j defined later.
- Consider $h_a(x) = h_a(y)$:

$$\sum_{i=1}^{r} a_i x_i \equiv \sum_{i=1}^{r} a_i y_i \iff a_j(x_j - y_j) \equiv \sum_{i \neq j} a_i(y_i - x_i) \mod p \tag{1}$$

Analyze our definition of $\mathcal{H}$

Theorem 4

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Proof.

- Let j be an index such that $x_j \neq y_j$.
- Define $a := \{\text{arb fix } a_i \text{ for } i \neq j\}$, a_j defined later.
- Consider $h_a(x) = h_a(y)$:

$$\sum_{i=1}^{r} a_i x_i \equiv \sum_{i=1}^{r} a_i y_i \iff a_j(x_j - y_j) \equiv \sum_{i \neq j} a_i(y_i - x_i) \mod p \tag{1}$$

- Lemma 4 shows there is a single value for a_j to satisfy (1).

Analyze our definition of $\mathcal{H}$

Theorem 4

The class of linear functions $\mathcal{H}$ as defined previously is universal.

Proof.

- Let j be an index such that $x_j \neq y_j$.
- Define $a := \{\text{arb fix } a_i \text{ for } i \neq j\}$, a_j defined later.
- Consider $h_a(x) = h_a(y)$:

$$\sum_{i=1}^{r} a_i x_i \equiv \sum_{i=1}^{r} a_i y_i \iff a_j(x_j - y_j) \equiv \sum_{i \neq j} a_i(y_i - x_i) \mod p \tag{1}$$

- Lemma 4 shows there is a single value for a_j to satisfy (1).
- So, $\Pr[h_a(x) = h_a(y)] \leq \frac{1}{p}$.

$\square$

MAX SAT

Satisfiability Problem (SAT)

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.

Satisfiability Problem (SAT)

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.
- For a given variable x_i, x_i is the assigned value and $\overline{x_i}$ is the negation of the assigned value.

Satisfiability Problem (SAT)

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.
- For a given variable x_i, x_i is the assigned value and $\overline{x_i}$ is the negation of the assigned value.
- A clause C_j is a *disjunction* of (distinct) terms, e.g., $(x_1 \vee \overline{x_2})$.

Satisfiability Problem (SAT)

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.
- For a given variable x_i, x_i is the assigned value and $\overline{x_i}$ is the negation of the assigned value.
- A clause C_j is a *disjunction* of (distinct) terms, e.g., $(x_1 \vee \overline{x_2})$.
- Length of C_j is the # of terms in C_j.

Satisfiability Problem (SAT)

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.
- For a given variable x_i, x_i is the assigned value and $\overline{x_i}$ is the negation of the assigned value.
- A clause C_j is a *disjunction* of (distinct) terms, e.g., $(x_1 \vee \overline{x_2})$.
- Length of C_j is the # of terms in C_j.
- A collection/*conjunction* of k clauses: $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$.

Satisfiability Problem (SAT)

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.
- For a given variable x_i, x_i is the assigned value and $\overline{x_i}$ is the negation of the assigned value.
- A clause C_j is a *disjunction* of (distinct) terms, e.g., $(x_1 \vee \overline{x_2})$.
- Length of C_j is the # of terms in C_j.
- A collection/*conjunction* of k clauses: $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$.
- Truth assignment function $v : X \to \{0, 1\}$, assigns values to the terms and returns the conjunction of the clauses.

Satisfiability Problem (SAT)

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.
- For a given variable x_i, x_i is the assigned value and $\overline{x_i}$ is the negation of the assigned value.
- A clause C_j is a *disjunction* of (distinct) terms, e.g., $(x_1 \vee \overline{x_2})$.
- Length of C_j is the # of terms in C_j.
- A collection/*conjunction* of k clauses: $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$.
- Truth assignment function $v : X \rightarrow \{0, 1\}$, assigns values to the terms and returns the conjunction of the clauses.
- v is a *satisfying assignment* if $\mathcal{C}$ is 1, i.e., all C_i evaluate to 1.

Satisfiability Problem (SAT)

TH: What values will satisfy the example?

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_3})$$

Preliminaries

- A set of boolean terms/literals: $X : x_1, \ldots, x_n$.
- For a given variable x_i, x_i is the assigned value and $\overline{x_i}$ is the negation of the assigned value.
- A clause C_j is a *disjunction* of (distinct) terms, e.g., $(x_1 \vee \overline{x_2})$.
- Length of C_j is the # of terms in C_j.
- A collection/*conjunction* of k clauses: $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$.
- Truth assignment function $v : X \rightarrow \{0, 1\}$, assigns values to the terms and returns the conjunction of the clauses.
- v is a *satisfying assignment* if $\mathcal{C}$ is 1, i.e., all C_i evaluate to 1.

MAX 3-SAT

3SAT Problem

Given a set of literals: $X : x_1, \ldots, x_n$, and a collection of clauses $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$, does there exist a satisfying assignment?

MAX 3-SAT

3SAT Problem

Given a set of literals: $X : x_1, \ldots, x_n$, and a collection of clauses $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$, each of length 3, does there exist a satisfying assignment?

MAX 3SAT Problem

Given a 3SAT problem satisfying as many clauses as possible.

MAX 3-SAT

3SAT Problem

Given a set of literals: $X : x_1, \ldots, x_n$, and a collection of clauses $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$, each of length 3, does there exist a satisfying assignment?

MAX 3SAT Problem

Given a 3SAT problem satisfying as many clauses as possible.

TH: Suggest a randomized algorithm.

MAX 3-SAT

3SAT Problem

Given a set of literals: $X : x_1, \ldots, x_n$, and a collection of clauses $\mathcal{C} : C_1 \wedge C_2 \wedge \cdot \wedge C_k$, each of length 3, does there exist a satisfying assignment?

MAX 3SAT Problem

Given a 3SAT problem satisfying as many clauses as possible.

Random Assignment

For each x_i, independently assign a value of 0 or 1 with probability $\frac{1}{2}$ each.

Analyze Random Assignment

Clause C_i

- Let Z_i be a random variable: 1 if clause is satisfied, 0 otherwise.

Analyze Random Assignment

Clause C_i

- Let Z_i be a random variable: 1 if clause is satisfied, 0 otherwise.
- TH: What is $\Pr[Z_i = 0]$?

Analyze Random Assignment

Clause C_i

- Let Z_i be a random variable: 1 if clause is satisfied, 0 otherwise.
- TH: What is $\Pr[Z_i = 0]$? $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$

Analyze Random Assignment

Clause C_i

- Let Z_i be a random variable: 1 if clause is satisfied, 0 otherwise.
- TH: What is $\Pr[Z_i = 0]$? $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$
- Each clause has 3 variables x_i each with $\Pr[x_i = 0] = \frac{1}{2}$:

$$\Pr[Z_i = 1] = 1 - \Pr[Z_i = 0] = 1 - \left(\frac{1}{2}\right)^3 = \frac{7}{8}$$

Analyze Random Assignment

Clause C_i

- Let Z_i be a random variable: 1 if clause is satisfied, 0 otherwise.
- TH: What is $\Pr[Z_i = 0]$? $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$
- Each clause has 3 variables x_i each with $\Pr[x_i = 0] = \frac{1}{2}$:

$$\Pr[Z_i = 1] = 1 - \Pr[Z_i = 0] = 1 - \left(\frac{1}{2}\right)^3 = \frac{7}{8}$$

- So, $\mathbb{E}[Z_i] = 1 \cdot \frac{7}{8} + 0 \cdot \frac{1}{8} = \frac{7}{8}$.

Analyze Random Assignment

Clause C_i

- Let Z_i be a random variable: 1 if clause is satisfied, 0 otherwise.
- So, $\mathbb{E}[Z_i] = 1 \cdot \frac{7}{8} + 0 \cdot \frac{1}{8} = \frac{7}{8}$.

Overall

Let $Z = \sum_{i=1}^{k} Z_i$:

Analyze Random Assignment

Clause C_i

- Let Z_i be a random variable: 1 if clause is satisfied, 0 otherwise.
- So, $\mathbb{E}[Z_i] = 1 \cdot \frac{7}{8} + 0 \cdot \frac{1}{8} = \frac{7}{8}$.

Overall

Let $Z = \sum_{i=1}^{k} Z_i$:

$$\begin{aligned}
\mathbb{E}[Z] &= \mathbb{E}\left[\sum_{i=1}^{k} Z_i\right] \\
&= \mathbb{E}[Z_1] + \mathbb{E}[Z_2] + \cdots + \mathbb{E}[Z_k] \text{ , by Linearity of Expectation,} \\
&= \frac{7}{8}k
\end{aligned}$$

Interesting Corollaries

Theorem 5

Random Assign satisfies 7/8 of the clauses in expectation.

Interesting Corollaries

Theorem 5

Random Assign satisfies $7/8$ of the clauses in expectation.

Corollary 6

For every 3-SAT, there is an assignment that satisfies $7/8$ of the clauses.

Interesting Corollaries

Theorem 5

Random Assign satisfies $7/8$ of the clauses in expectation.

Corollary 6

For every 3-SAT, there is an assignment that satisfies $7/8$ of the clauses.

Proof.

Since the expectation is a weighted average, its value is between the maximum and minimum possible values. □

Interesting Corollaries

Theorem 5

Random Assign satisfies $7/8$ of the clauses in expectation.

Corollary 6

For every 3-SAT, there is an assignment that satisfies $7/8$ of the clauses.

Corollary 7

Every 3-SAT with ≤ 7 clauses is satisfiable.

Interesting Corollaries

Theorem 5

Random Assign satisfies 7/8 of the clauses in expectation.

Corollary 6

For every 3-SAT, there is an assignment that satisfies 7/8 of the clauses.

Corollary 7

Every 3-SAT with ≤ 7 clauses is satisfiable.

Proof.

For $k \leq 7$, $\frac{7}{8}k > k - 1$. $\square$

Waiting For a Good Assignment

Theorem 8

There exists a randomized algorithm with a polynomial expectation running time that is guaranteed to produce a truth assignment satisfying at least 7/8 of all k clauses.

Waiting For a Good Assignment

Theorem 8

There exists a randomized algorithm with a polynomial expectation running time that is guaranteed to produce a truth assignment satisfying at least $7/8$ of all k clauses.

Proof.

- Let p_j be the probability that j clauses are satisfied.

Waiting For a Good Assignment

Theorem 8

There exists a randomized algorithm with a polynomial expectation running time that is guaranteed to produce a truth assignment satisfying at least 7/8 of all k clauses.

Proof.

- Let p_j be the probability that j clauses are satisfied.
- We need to calculate $p = \sum_{j \geq \frac{7}{8}k} p_j$.

Waiting For a Good Assignment

Theorem 8

There exists a randomized algorithm with a polynomial expectation running time that is guaranteed to produce a truth assignment satisfying at least 7/8 of all k clauses.

Proof.

- Let p_j be the probability that j clauses are satisfied.
- We need to calculate $p = \sum_{j \geq \frac{7}{8}k} p_j$.
- By Definition of expectation:

$$\frac{7}{8}k = \sum_{j=0}^{k} jp_j = \sum_{j < \frac{7}{8}k} jp_j + \sum_{j \geq \frac{7}{8}k} jp_j$$

$$\leq \left(\frac{7k}{8} - \frac{1}{8}\right) \sum_{j < \frac{7}{8}k} p_j + k \sum_{j \geq \frac{7}{8}k} p_j$$

Waiting For a Good Assignment

Theorem 8

There exists a randomized algorithm with a polynomial expectation running time that is guaranteed to produce a truth assignment satisfying at least 7/8 of all k clauses.

Proof.

- Let p_j be the probability that j clauses are satisfied.
- We need to calculate $p = \sum_{j \geq \frac{7}{8}k} p_j$.
- By Definition of expectation:

$$\frac{7}{8}k = \sum_{j=0}^{k} j p_j \leq \left(\frac{7k}{8} - \frac{1}{8}\right) \sum_{j < \frac{7}{8}k} p_j + k \sum_{j \geq \frac{7}{8}k} p_j$$

$$\iff \frac{7}{8}k \leq \left(\frac{7k}{8} - \frac{1}{8}\right)(1-p) + kp \leq \left(\frac{7k}{8} - \frac{1}{8}\right) + kp$$

Waiting For a Good Assignment

Theorem 8

There exists a randomized algorithm with a polynomial expectation running time that is guaranteed to produce a truth assignment satisfying at least 7/8 of all k clauses.

Proof.

- Let p_j be the probability that j clauses are satisfied.
- We need to calculate $p = \sum_{j \geq \frac{7}{8}k} p_j$.
- By Definition of expectation:

$$\iff \frac{7}{8}k \leq \left(\frac{7k}{8} - \frac{1}{8}\right)(1-p) + kp \leq \left(\frac{7k}{8} - \frac{1}{8}\right) + kp$$

$$\iff p \geq \frac{\frac{7}{8}k - \left(\frac{7k}{8} - \frac{1}{8}\right)}{k} = \frac{1}{8k}.$$

Waiting For a Good Assignment

Theorem 8

There exists a randomized algorithm with a polynomial expectation running time that is guaranteed to produce a truth assignment satisfying at least 7/8 of all k clauses.

Proof.

- Let p_j be the probability that j clauses are satisfied.
- We need to calculate $p = \sum_{j \geq \frac{7}{8}k} p_j$.
- With $p = \frac{1}{8k}$, we have a Bernoulli trial: Within $8k$ tries, we expect an assignment that satisfies $\frac{7}{8}$ of the clauses.
- I.e., the expected runtime is $8k$ runs of random assignment.

$\square$

Appendix

References

Image Sources I

https://brand.wisc.edu/web/logos/