Generating Random Structures

Manuel Kauers

Seminar on Selected Algorithms in Symbolic Computation
JKU · Summer Semester 2012

1. Random Bits

Task: Input: none; Output: 0 or 1 with equal probability.

Task: Input: none; Output: 0 or 1 with equal probability.

- By hand: Toss a coin!

Task: Input: none; Output: 0 or 1 with equal probability.

- By hand: Toss a coin!
- On a computer: *theoretically impossible*

Task: Input: none; Output: 0 or 1 with equal probability.

- By hand: Toss a coin!
- On a computer: *theoretically impossible*
- In practice: use a "pseudo-random number generator"

Task: Input: none; Output: 0 or 1 with equal probability.

- By hand: Toss a coin!
- On a computer: *theoretically impossible*
- In practice: use a "pseudo-random number generator"
- Better: extract random bits from some chaotic physical process (e.g., quantom phenomena, atmospheric fluctuations)

Task: Input: none; Output: 0 or 1 with equal probability.

- By hand: Toss a coin!
- On a computer: *theoretically impossible*
- In practice: use a "pseudo-random number generator"
- Better: extract random bits from some chaotic physical process (e.g., quantom phenomena, atmospheric fluctuations)
- www.random.org

Task: Input: none; Output: 0 or 1 with equal probability.

- By hand: Toss a coin!
- On a computer: *theoretically impossible*
- In practice: use a "pseudo-random number generator"
- Better: extract random bits from some chaotic physical process (e.g., quantom phenomena, atmospheric fluctuations)
- www.random.org
- Today: We simply assume that we can generate uniformly distributed random bits somehow.

Task: Input: none; Output: 0 or 1 with equal probability.

- By hand: Toss a coin!
- On a computer: *theoretically impossible*
- In practice: use a "pseudo-random number generator"
- Better: extract random bits from some chaotic physical process (e.g., quantom phenomena, atmospheric fluctuations)
- www.random.org
- Today: We simply assume that we can generate uniformly distributed random bits somehow.
- Question: How can we use them to create other random objects?

2. Random Integers

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

If n is a power of 2, say $n = 2^{\nu}$, this is easy:

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

If n is a power of 2, say $n = 2^\nu$, this is easy:

- Choose ν random bits $k_0, k_1, \ldots, k_{\nu-1} \in \{0, 1\}$

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

If n is a power of 2, say $n = 2^\nu$, this is easy:

- Choose ν random bits $k_0, k_1, \ldots, k_{\nu-1} \in \{0, 1\}$
- Return $k = k_0 + 2k_1 + 4k_2 + 8k_3 + \cdots + 2^{\nu-1}k_{\nu-1}$.

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

If n is a power of 2, say $n = 2^\nu$, this is easy:

- Choose ν random bits $k_0, k_1, \ldots, k_{\nu-1} \in \{0, 1\}$
- Return $k = k_0 + 2k_1 + 4k_2 + 8k_3 + \cdots + 2^{\nu-1}k_{\nu-1}$.

It is clear that every output k is equally likely.

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

If n is a power of 2, say $n = 2^\nu$, this is easy:

- Choose ν random bits $k_0, k_1, \ldots, k_{\nu-1} \in \{0, 1\}$
- Return $k = k_0 + 2k_1 + 4k_2 + 8k_3 + \cdots + 2^{\nu-1}k_{\nu-1}$.

It is clear that every output k is equally likely.

But what if n is not a power of two?

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

First version:

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

First version:

- Let ν be the smallest integer with $2^\nu \geq n$

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

First version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \dots, 2^\nu - 1\}$

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

First version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- Return $k \bmod n$

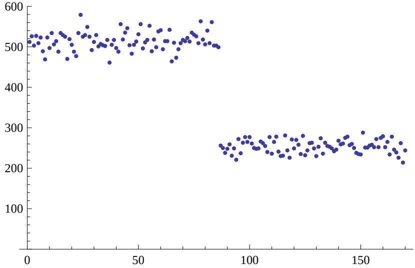
Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

First version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- Return $k \bmod n$

THIS IS FLAWED!

Example: Randomly choosing 65536 integers k with $0 \leq k < 170$ by this method gives the following output distribution:



Some outputs are more likely than others.

(A point (k, u) in the plot indicates that k appeared u times as output.)

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

First version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- Return $k \bmod n$

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Second version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- Return $\lfloor kn/2^\nu \rfloor$

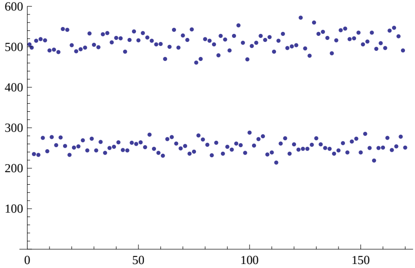
Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Second version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- Return $\lfloor kn/2^\nu \rfloor$

THIS IS NOT BETTER!

Example: Randomly choosing 65536 integers k with $0 \leq k < 170$ by this method gives the following output distribution:



Some outputs are more likely than others.

(A point (k, u) in the plot indicates that k appeared u times as output.)

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Third version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Third version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- If $k \geq n$, try again

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Third version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- If $k \geq n$, try again
- Otherwise return k.

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Third version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- If $k \geq n$, try again
- Otherwise return k.

Now every output k with $0 \leq k < n$ is equally likely.

Task: Given $n \in \mathbb{N}$, choose a random integer k with $0 \leq k < n$.

Third version:

- Let ν be the smallest integer with $2^\nu \geq n$
- Choose $k \in \{0, \ldots, 2^\nu - 1\}$
- If $k \geq n$, try again
- Otherwise return k.

Now every output k with $0 \leq k < n$ is equally likely.

(But more random bits are generated. Is there a better way?)

3. Random Subsets

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Naive:

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Naive:

- Construct the power set $\mathcal{P}(S) = \{\emptyset, \{x_1\}, \{x_2\}, \ldots, S\}$ of S

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Naive:

- Construct the power set $\mathcal{P}(S) = \{\emptyset, \{x_1\}, \{x_2\}, \ldots, S\}$ of S
- Choose a random integer $k \in \{0, 1, \ldots, 2^{|S|} - 1\}$

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Naive:

- Construct the power set $\mathcal{P}(S) = \{\emptyset, \{x_1\}, \{x_2\}, \ldots, S\}$ of S
- Choose a random integer $k \in \{0, 1, \ldots, 2^{|S|} - 1\}$
- Return the kth element of $\mathcal{P}(S)$

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Better:

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Better:

- Choose a random integer $k \in \{0, 1, \ldots, 2^{|S|} - 1\}$

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Better:

- Choose a random integer $k \in \{0, 1, \ldots, 2^{|S|} - 1\}$
- Let $k = k_1 k_2 k_3 \cdots k_{|S|}$ be the binary digit representation of k

Task: Given a set $S := \{x_1, \ldots, x_n\}$, choose a random subset.

Better:

- Choose a random integer $k \in \{0, 1, \ldots, 2^{|S|} - 1\}$
- Let $k = k_1 k_2 k_3 \cdots k_{|S|}$ be the binary digit representation of k
- Return the subset $\{x_i : k_i = 1\} \subseteq S$

More generally: if A is some finite (but possibly very big) set of combinatorial objects (e.g., $\mathcal{P}(S)$), we can efficiently pick a random element if we know a *bijection*

$$b\colon \{0, 1, 2, \ldots, |A| - 1\} \to A$$

which can be computed efficiently:

More generally: if A is some finite (but possibly very big) set of combinatorial objects (e.g., $\mathcal{P}(S)$), we can efficiently pick a random element if we know a *bijection*

$$b\colon \{0, 1, 2, \ldots, |A| - 1\} \to A$$

which can be computed efficiently:

- Choose a random integer $x \in \{0, 1, \ldots, |A| - 1\}$

More generally: if A is some finite (but possibly very big) set of combinatorial objects (e.g., $\mathcal{P}(S)$), we can efficiently pick a random element if we know a *bijection*

$$b\colon \{0, 1, 2, \ldots, |A| - 1\} \to A$$

which can be computed efficiently:

- Choose a random integer $x \in \{0, 1, \ldots, |A| - 1\}$
- Return $b(x)$

4. Random Binary Trees

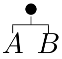
Definition:

Definition:

- $\circ$ is a binary tree of size 0.

Definition:

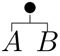
- $\circ$ is a binary tree of size 0.
- If A is a binary tree of size a and B is a binary tree of size b, then

  is a binary tree of size $a + b + 1$, of which A is called the left subtree and B is called the right subtree.

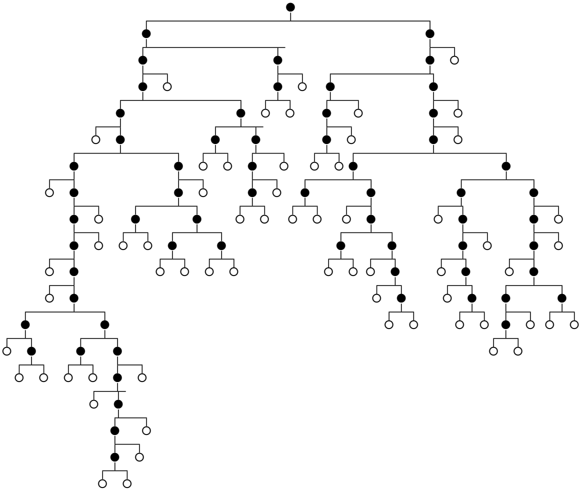
Definition:

- $\circ$ is a binary tree of size 0.
- If A is a binary tree of size a and B is a binary tree of size b, then

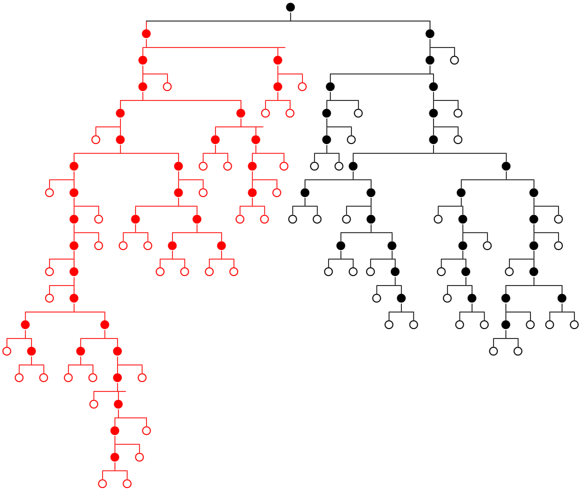
  

  is a binary tree of size $a + b + 1$, of which A is called the left subtree and B is called the right subtree.
- There are no other binary trees.

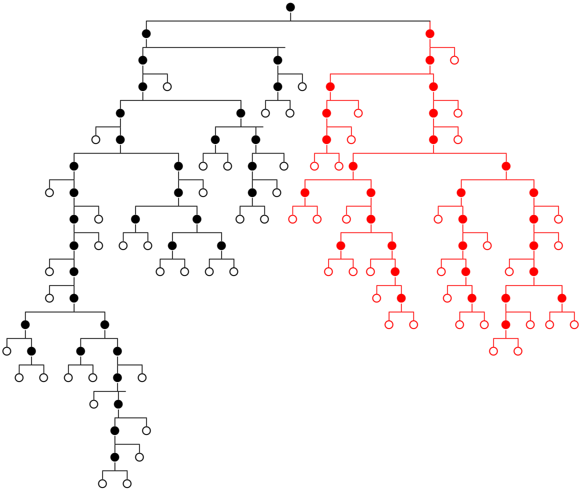
Example: Here is a random binary tree of size 63:



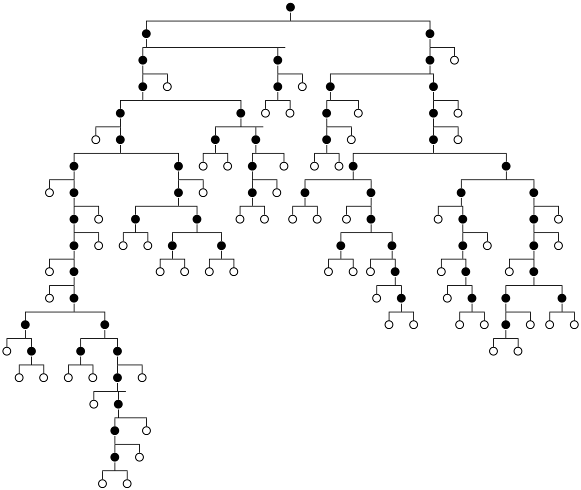
Example: Here is a random binary tree of size 63:



Example: Here is a random binary tree of size 63:



Example: Here is a random binary tree of size 63:



How many binary trees are there?

How many binary trees are there?

If C_n is the number of binary trees of size n, then

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} \quad (n \geq 1), \qquad C_0 = 1.$$

This follows directly from the definition.

How many binary trees are there?

If C_n is the number of binary trees of size n, then

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} \quad (n \geq 1), \qquad C_0 = 1.$$

This follows directly from the definition.

By induction, it can be shown that the solution of this recurrence is

$$C_n = \frac{1}{n+1}\binom{2n}{n} \qquad (n \geq 1).$$

How many binary trees are there?

If C_n is the number of binary trees of size n, then

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} \quad (n \geq 1), \qquad C_0 = 1.$$

This follows directly from the definition.

By induction, it can be shown that the solution of this recurrence is

$$C_n = \frac{1}{n+1}\binom{2n}{n} \qquad (n \geq 1).$$

These numbers are known as *Catalan numbers.*

Goal: Construct a bijection between $\{0, 1, \ldots, C_n - 1\}$ and the set of binary trees of size n.

Goal: Construct a bijection between $\{0, 1, \ldots, C_n - 1\}$ and the set of binary trees of size n.

Need:

Goal: Construct a bijection between $\{0, 1, \ldots, C_n - 1\}$ and the set of binary trees of size n.

Need:

- **Encoding:** Given a binary tree of size n, encode it faithfully into an integer $x \in \{0, \ldots, C_n - 1\}$.

Goal: Construct a bijection between $\{0, 1, \ldots, C_n - 1\}$ and the set of binary trees of size n.

Need:

- **Encoding:** Given a binary tree of size n, encode it faithfully into an integer $x \in \{0, \ldots, C_n - 1\}$.
- **Decoding:** Given an integer $x \in \{0, \ldots, C_n - 1\}$, reconstruct the corresponding binary tree of size n.

Encoding.

Encoding.

- There are precisely $C_k C_{n-1-k}$ binary trees of size n whose left subtree has size exactly k.

Encoding.

- There are precisely $C_k C_{n-1-k}$ binary trees of size n whose left subtree has size exactly k.
- Consequently, there are $\sum_{i=0}^{k} C_i C_{n-1-i}$ binary trees of size n whose left subtree has size at most k.

Encoding.

- There are precisely $C_k C_{n-1-k}$ binary trees of size n whose left subtree has size exactly k.
- Consequently, there are $\sum_{i=0}^{k} C_i C_{n-1-i}$ binary trees of size n whose left subtree has size at most k.
- We choose to use the numbers in the segment

$$\left\{ \sum_{i=0}^{k-1} C_i C_{n-1-i}, \quad \ldots\ldots, \quad \sum_{i=0}^{k} C_i C_{n-1-i} - 1 \right\}$$

for representing trees of size n with left subtrees of size k.

Encoding.

- The tree T of size 0 is encoded by $\mathrm{enc}(T) := 0$.

Encoding.

- The tree T of size 0 is encoded by $\mathrm{enc}(T) := 0$.
- Suppose we already know for every $k < n$ how to encode trees of size k into an integer $x \in \{0, \ldots, C_k - 1\}$.

Encoding.

- The tree T of size 0 is encoded by $\mathrm{enc}(T) := 0$.
- Suppose we already know for every $k < n$ how to encode trees of size k into an integer $x \in \{0, \ldots, C_k - 1\}$.
- Then, for a given tree T of size n with subtrees A and B of sizes k and $n - 1 - k$, we set

$$\mathrm{enc}(T) := \sum_{i=0}^{k-1} C_i C_{n-1-i} + \ \mathrm{enc}(A) + C_k \ \mathrm{enc}(B)$$

Encoding.

- The tree T of size 0 is encoded by $\mathrm{enc}(T) := 0$.
- Suppose we already know for every $k < n$ how to encode trees of size k into an integer $x \in \{0, \ldots, C_k - 1\}$.
- Then, for a given tree T of size n with subtrees A and B of sizes k and $n-1-k$, we set

$$\mathrm{enc}(T) := \sum_{i=0}^{k-1} C_i C_{n-1-i} + \underbrace{\mathrm{enc}(A)}_{0 \le \cdot < C_k} + C_k \; \mathrm{enc}(B)$$

Encoding.

- The tree T of size 0 is encoded by $\mathrm{enc}(T) := 0$.
- Suppose we already know for every $k < n$ how to encode trees of size k into an integer $x \in \{0, \ldots, C_k - 1\}$.
- Then, for a given tree T of size n with subtrees A and B of sizes k and $n-1-k$, we set

$$\mathrm{enc}(T) := \sum_{i=0}^{k-1} C_i C_{n-1-i} + \underbrace{\mathrm{enc}(A)}_{0 \le \cdot < C_k} + C_k \underbrace{\mathrm{enc}(B)}_{0 \le \cdot < C_{n-1-k}}$$

Encoding.

- The tree T of size 0 is encoded by $\mathrm{enc}(T) := 0$.
- Suppose we already know for every $k < n$ how to encode trees of size k into an integer $x \in \{0, \dots, C_k - 1\}$.
- Then, for a given tree T of size n with subtrees A and B of sizes k and $n - 1 - k$, we set

$$\mathrm{enc}(T) := \sum_{i=0}^{k-1} C_i C_{n-1-i} + \underbrace{\underbrace{\mathrm{enc}(A)}_{0 \le \cdot < C_k} + C_k \underbrace{\mathrm{enc}(B)}_{0 \le \cdot < C_{n-1-k}}}_{0 \le \cdot < C_k C_{n-1-k}}$$

Encoding.

- The tree T of size 0 is encoded by $\mathrm{enc}(T) := 0$.
- Suppose we already know for every $k < n$ how to encode trees of size k into an integer $x \in \{0, \dots, C_k - 1\}$.
- Then, for a given tree T of size n with subtrees A and B of sizes k and $n-1-k$, we set

$$\mathrm{enc}(T) := \underbrace{\sum_{i=0}^{k-1} C_i C_{n-1-i} + \underbrace{\underbrace{\mathrm{enc}(A)}_{0 \le \cdot < C_k} + C_k \underbrace{\mathrm{enc}(B)}_{0 \le \cdot < C_{n-1-k}}}_{0 \le \cdot < C_k C_{n-1-k}}}_{\sum_{i=0}^{k-1} C_i C_{n-1-i} \le \cdot < \sum_{i=0}^{k} C_i C_{n-1-i}}$$

Decoding.

Decoding.

- Set $\mathrm{dec}(0, 0)$ to the tree of size 0.

Decoding.

- Set $\mathrm{dec}(0,0)$ to the tree of size 0.
- Suppose we already know how to compute $\mathrm{dec}(x,k)$ for every $x \in \{0,\ldots,C_k-1\}$ and every $k<n$.

Decoding.

- Set $\mathrm{dec}(0, 0)$ to the tree of size 0.
- Suppose we already know how to compute $\mathrm{dec}(x, k)$ for every $x \in \{0, \ldots, C_k - 1\}$ and every $k < n$.
- Then a given $x \in \{0, \ldots, C_n - 1\}$ can be decoded as follows.

Decoding.

- Set $\mathrm{dec}(0, 0)$ to the tree of size 0.
- Suppose we already know how to compute $\mathrm{dec}(x, k)$ for every $x \in \{0, \ldots, C_k - 1\}$ and every $k < n$.
- Then a given $x \in \{0, \ldots, C_n - 1\}$ can be decoded as follows.
  - Find $k < n$ with $\sum_{i=0}^{k-1} C_i C_{n-1-i} \leq x < \sum_{i=0}^{k} C_i C_{n-1-i}$.

Decoding.

- Set $\mathrm{dec}(0,0)$ to the tree of size 0.
- Suppose we already know how to compute $\mathrm{dec}(x,k)$ for every $x \in \{0,\dots,C_k-1\}$ and every $k<n$.
- Then a given $x \in \{0,\dots,C_n-1\}$ can be decoded as follows.
  - Find $k<n$ with $\sum_{i=0}^{k-1} C_i C_{n-1-i} \le x < \sum_{i=0}^{k} C_i C_{n-1-i}$.
  - Set $x := x - \sum_{i=0}^{k-1} C_i C_{n-1-i}$.

Decoding.

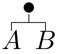
- Set $\mathrm{dec}(0,0)$ to the tree of size 0.
- Suppose we already know how to compute $\mathrm{dec}(x,k)$ for every $x \in \{0,\dots,C_k - 1\}$ and every $k < n$.
- Then a given $x \in \{0,\dots,C_n - 1\}$ can be decoded as follows.
  - Find $k < n$ with $\sum_{i=0}^{k-1} C_i C_{n-1-i} \leq x < \sum_{i=0}^{k} C_i C_{n-1-i}$.
  - Set $x := x - \sum_{i=0}^{k-1} C_i C_{n-1-i}$.
  - Set $a := \mathrm{rem}(x, C_k)$ and $b := \mathrm{quo}(x, C_k)$.

Decoding.

- Set $\operatorname{dec}(0, 0)$ to the tree of size 0.
- Suppose we already know how to compute $\operatorname{dec}(x, k)$ for every $x \in \{0, \ldots, C_k - 1\}$ and every $k < n$.
- Then a given $x \in \{0, \ldots, C_n - 1\}$ can be decoded as follows.
    - Find $k < n$ with $\sum_{i=0}^{k-1} C_i C_{n-1-i} \leq x < \sum_{i=0}^{k} C_i C_{n-1-i}$.
    - Set $x := x - \sum_{i=0}^{k-1} C_i C_{n-1-i}$.
    - Set $a := \operatorname{rem}(x, C_k)$ and $b := \operatorname{quo}(x, C_k)$.
    - Compute $A := \operatorname{dec}(a, k)$ and $B := \operatorname{dec}(b, n - 1 - k)$.

Decoding.

- Set $\mathrm{dec}(0,0)$ to the tree of size 0.
- Suppose we already know how to compute $\mathrm{dec}(x,k)$ for every $x \in \{0,\ldots,C_k - 1\}$ and every $k < n$.
- Then a given $x \in \{0,\ldots,C_n - 1\}$ can be decoded as follows.
  - Find $k < n$ with $\sum_{i=0}^{k-1} C_i C_{n-1-i} \le x < \sum_{i=0}^{k} C_i C_{n-1-i}$.
  - Set $x := x - \sum_{i=0}^{k-1} C_i C_{n-1-i}$.
  - Set $a := \mathrm{rem}(x, C_k)$ and $b := \mathrm{quo}(x, C_k)$.
  - Compute $A := \mathrm{dec}(a,k)$ and $B := \mathrm{dec}(b, n-1-k)$.
  - Return



5. Random Topics

Possible topics for seminar talks: similar constructions for other combinatorial objects

- Permutations (Knuth-shuffle)
- Young Tableaux (Robinson-Schensted-Knuth algorithm)
- Unrooted labeled trees with arbitrary number of subtrees (Prüfer transform)
- Subsets with prescribed number of elements
- Integer Partitions