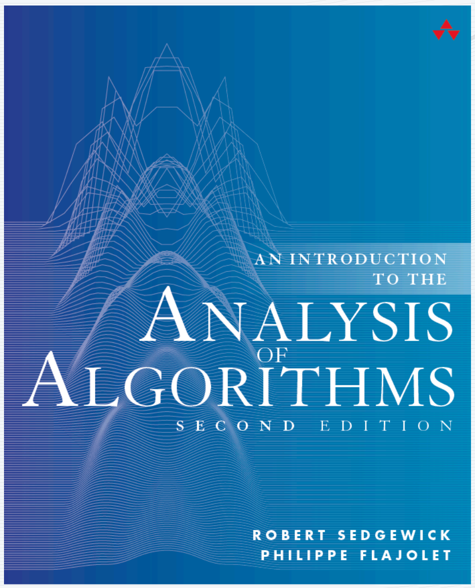
ANALYTIC COMBINATORICS

PART ONE



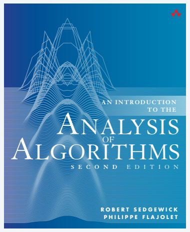
http://aofa.cs.princeton.edu

8. Strings and Tries

Orientation

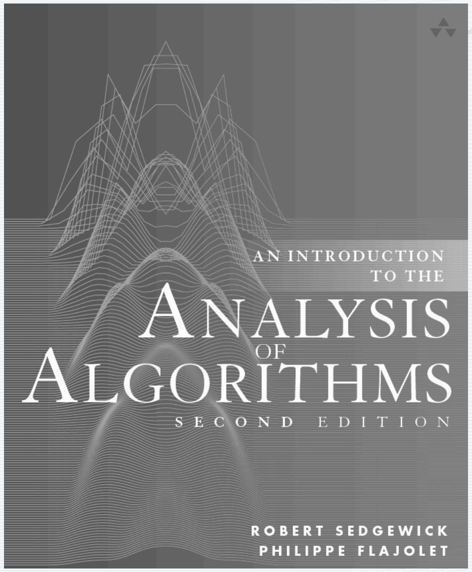
Second half of class

- Surveys fundamental combinatorial classes.
- Considers techniques from analytic combinatorics to study them .
- Includes applications to the analysis of algorithms.



chapter	*combinatorial classes*	*type of class*	*type of GF*
6	Trees	unlabeled	OGFs
7	Permutations	labeled	EGFs
8	Strings and Tries	unlabeled	OGFs
9	Words and Mappings	labeled	EGFs

Note: Many more examples in book than in lectures.



http://aofa.cs.princeton.edu

8. Strings and Tries

- **Bitstrings with restrictions**
- Languages
- Tries
- Trie parameters

Bitstrings

23	10111110100101001100111**000**1001111101101101000001111000011001110111011111101011000
9	110100101**000**111101001111001101001110110101111100000101101110011010000000111001110
29	11101110101100111010111001101**000**011000111001010111110011001000011001000101010010
6	10111**000**011011000110011101110011011011110111110011101011000011001100101000000110
13	1010110011101**000**1101101110110010010110100101001101111100110000001111101000001111
1	1**000**001001100000110001100010000111100111001111000001100111111001101100010010011 1
24	100010101011100011101011**000**0011000001110101010001011000100110111111001111 10110010
18	0011101100101110010**00**0110000100111101001001100110000110011101001101000010100011 1
42	0011111110011011011101101110101001101101111**000**111111110101110100110000000100101110
5	10101**000**111100001010000011001000001101010010100011001100101010101110110111111110
2	11**000**000101111011011000101011010110010010000001110111001000000110101000000001010 00
70	11101111011011111011111111111010011101001011111110111010011101**000**1100010010001001 0
25	0011111110011101011011111**000**0100010001110000111010111100101011111001110101011111
0	**000**000100011110011101101010111001100000111110010010010101001100110011010011011110
24	101111001010001001101111**000**1100100011100100001010011010111011111101011001001110 0
7	0101001**000**101111011000011011010101101010111101100110110110100010011000111110011 1
23	01110110010011001110111**000**101010001101101001111111001101010111010001100110100001
3	001**000**1101101000110001111111001110011001111001011000110011001101000111011 1011101

Q. What is the *expected wait time for the first occurrence* of 000 in a random bitstring?

Q. What is the probability that an *N*-bit random bitstring *does not contain* 000?

Symbolic method for unlabelled objects (review)

Warmup: How many binary strings with N bits?

Class	B, the class of all binary strings
Size	$\lvert b \rvert$, the number of bits in b
OGF	$B(z) = \sum_{b \in B} z^{\lvert b \rvert} = \sum_{N \geq 0} B_N z^N$

Atoms

type	*class*	*size*	*GF*
0 bit	Z_0	1	z
1 bit	Z_1	1	z

Construction $$B = SEQ(Z_0 + Z_1)$$

"a binary string is a sequence of 0 bits and 1 bits"

OGF equation $$B(z) = \frac{1}{1 - 2z}$$

$$[z^N]B(z) = 2^N \quad \checkmark$$

Symbolic method for unlabelled objects (review)

Warmup: How many binary strings with N bits (alternate proof)?

Class	B, the class of all binary strings
Size	$\lvert b\rvert$, the number of bits in b
OGF	$B(z) = \sum_{b \in B} z^{\lvert b\rvert} = \sum_{N \geq 0} B_N z^N$

Atoms

type	*class*	*size*	*GF*
0 bit	Z_0	1	z
1 bit	Z_1	1	z

Construction $\quad B = E + (Z_0 + Z_1) \times B$

"a binary string is empty or
a bit followed by a binary string"

OGF equation $\quad B(z) = 1 + 2zB(z)$

Solution $\quad B(z) = \dfrac{1}{1-2z}$

$$[z^N]B(z) = 2^N \quad ✓$$

Ex. How many N-bit binary strings have no two consecutive 0s?

Class	B_{00}, the class of binary strings with no 00
OGF	$B_{00}(z) = \sum_{b \in B_{00}} z^{\lvert b \rvert}$

Construction $$B_{00} = E + Z_0 + (Z_1 + Z_0 \times Z_1) \times B_{00}$$

"a binary string with no 00 is either empty or 0 or it is 1 or 01 followed by a binary string with no 00"

OGF equation $$B_{00}(z) = 1 + z + (z + z^2)B_{00}(z)$$

Solution $$B_{00}(z) = \frac{1+z}{1-z-z^2}$$

Extract cofficients $$[z^N]B_{00}(z) = F_N + F_{N+1} = F_{N+2}$$

1, 2, 5, 8, 13, ... ✓

$$= \frac{\phi^2}{\sqrt{5}}\phi^N \sim c_2\beta_2^N \quad \text{with} \begin{cases} \beta_2 \doteq 1.61803 \\ c_2 \doteq 1.17082 \end{cases}$$

Binary strings without long runs of 0s

Ex. How many N-bit binary strings have no runs of P consecutive 0s?

Class	B_P, the class of binary strings with no 0^P
OGF	$B_P(z) = \sum_{b \in B_P} z^{\lvert b \rvert}$

Construction $$B_P = Z_{<P}(E + Z_1 B_P)$$

"a string with no 0^P is a string of 0s of length <P followed by an empty string or a 1 followed by a string with no 0^P"

OGF equation $$B_P(z) = (1 + z + \ldots + z^P)(1 + zB_P(z))$$

Solution $$B_P(z) = \frac{1 - z^P}{1 - 2z + z^{P+1}}$$

$1/\beta_k$ is the smallest root of

Extract cofficients $$[z^N]B_k(z) \sim c_k \beta_k^N \quad \text{where} \begin{cases} \beta_k \text{ is the dominant root of } 1 - 2z + z^k \\ c_k = \text{ [explicit formula available]} \end{cases}$$

See "Asymptotics" lecture

Binary strings without long runs

Theorem. The number of binary strings of length N with no runs of P 0s is $\sim c_P \beta_P^N$ where c_P and β_P are easily-calculated constants.

```
sage: f2 = 1 - 2*x + x^3
sage: 1.0/f2.find_root(0, .99, x)
1.61803398874989
sage: f3 = 1 - 2*x + x^4
sage: 1.0/f3.find_root(0, .99, x)
1.83928675521416
sage: f4  = 1 - 2*x + x^5
sage: 1.0/f4.find_root(0, .99, x)
1.92756197548293
sage: f5 = 1 - 2*x + x^6
sage: 1.0/f5.find_root(0, .99, x)
1.96594823664510
sage: f6 = 1 - 2*x + x^7
sage: 1.0/f6.find_root(0, .99, x)
1.98358284342432
```

β_2 β_3 β_4 β_5 β_6

Information on consecutive 0s in GFs for strings

$$S_P(z) = \sum_{s \in \mathcal{S}_P} z^{|s|} = \frac{1 - z^P}{1 - 2z + z^{P+1}} = \sum_{N \geq 0} \{\text{\# of bitstrings of length } N \text{ with no } 0^P\} z^N$$

$$S_P(z/2) = \sum_{N \geq 0} \left(\{\text{\# of bitstrings of length } N \text{ with no runs of } P \text{ 0s}\}/2^N\right) z^N$$

$$S_P(1/2) = \sum_{N \geq 0} \{\text{\# of bitstrings of length } N \text{ with no runs of } P \text{ 0s}\}/2^N$$

$$= \sum_{N \geq 0} \Pr\,\{\text{1st } N \text{ bits of a random bitstring have no runs of } P \text{ 0s}\}$$

$$= \sum_{N \geq 0} \Pr\,\{\text{position of end of first } 0^P \text{ is} > N\,\} = \text{Expected position of end of first } 0^P$$

Theorem. Probability that an N-bit random bitstring has no 0^P : $[z^N]S_P(z/2) \sim c_P(\beta_P/2)^N$

Theorem. Expected wait time for the first 0^P in a random bitstring: $S_P(1/2) = 2^{P+1} - 2$

Consecutive 0s in random bitstrings

P	$S_P(z)$	approx. probability of no 0^P in N random bits			wait time
		N	10	100	
1	$\dfrac{1-z}{1-2z+z^2}$	$.5^N$	0.0010	$<10^{-30}$	2
2	$\dfrac{1-z^2}{1-2z+z^3}$	$1.1708 \times .80901^N$	0.1406	$<10^{-9}$	6
3	$\dfrac{1-z^3}{1-2z+z^4}$	$1.1375 \times .91864^N$	0.4869	0.0023	14
4	$\dfrac{1-z^4}{1-2z+z^5}$	$1.0917 \times .96328^N$	0.7510	0.0259	30
5	$\dfrac{1-z^5}{1-2z+z^6}$	$1.0575 \times .98297^N$	0.8906	0.1898	62
6	$\dfrac{1-z^6}{1-2z+z^7}$	$1.0350 \times .99174^N$	0.9526	0.4516	126

Validation of mathematical results

is always worthwhile when analyzing algorithms

```
public class TestOccP
{
   public static int find(int[] bits, int k)
   // See code at right.

   public static void main(String[] args)
   {
      int w = Integer.parseInt(args[0]);
      int maxP = Integer.parseInt(args[1]);
      int[] bits = new int[w];
      int[] sum = new int[maxP+1];

      int T = 0;
      int cnt = 0;
      while (!StdIn.isEmpty())
      {
         T++;
         for (int j = 0; j < w; j++)
            bits[j] = BitIO.readbit();
         for (int P = 1; P <= maxP; P++)
            if (find(bits, P) == bits.length) sum[P]++;
      }

      for (int P = 1; P <= maxP; P++)
          StdOut.printf("%8.4f\n", 1.0*sum[P]/T);
      StdOut.println(T + "trials");
   }
}
```

N/w trials.

- Read w-bits from StdIn
- For each P, check for 0^P

Print empirical probabilities.

```
public static int find(int[] bits, int P)
{
   int cnt = 0;
   for (int i = 0; i < bits.length; i++)
   {
      if (cnt == P) return i;
      if (bits[i] == 0) cnt++; else cnt = 0;
   }
   return bits.length;
}
```

```
% java TestOccP 100 6 < data/random1M.txt
  0.0000    .0000
  0.0000    .0000
  0.0004    .0023
  0.0267    .0259
  0.1861    .1898
  0.4502    .4516
10000 trials
```

predicted by theory

✓

Wait time for specified patterns

9	23	10111110100101001100111000100111110110110100000111100001100111011101111101011000
4	9	11010010100011110100111100110100111011010111110000010110111001101000000111001110
12	29	11101110101100111010111001101000011000111001010111110011001000011001000101010010
8	5	10111000011011000110011101110011011011110111110011101011000011001100101000000110
6	13	10101100111010001101101110110010010110100101001101111100110000001111101000001111
4	1	10000010011000001100011000100001111001110011110000011001111110011011000100100111
2	24	10001010101110001110101100000110000011101010100010110001001101111110011110110010
0	18	00111011001011100100001100001001111010010011001100001100111010011010000101000111
0	42	00111111100110110111011011101010011011011100011111111010111010011000000100101110
6	5	10101000111110000101000001100100000110101001010001100110010101010111011011111110
6	2	11000000101111011011000101011010110010010000011101110010000001101010000000101000
30	70	11101111011011111011111111110100111010010111111011101001110100011000100100010010
0	25	00111111100111010110111110000100010001110000111010111100101011111001110101011111
4	0	00000010001111001110110101011100110000011110010010010101001100110011010011011110
6	24	10111100101000100110111100011001000111001000010100110101110111111010110010011100
4	7	01010010001011110110000110110101011010101111011001101101101000100110001111100111
7	23	01110110010011001110111000101010001101101001111111001101010111010001100110100001
0	3	00100011011010001100011111110011100110011110010110001100110011010001110111011101

Expected wait time for the first occurrence of 000: 17.9

Expected wait time for the first occurrence of 001: 6.0

Are these bitstrings random??

Autocorrelation

The probability that an N-bit random bitstring does not contain 0000 is $\sim 1.0917 \times .96328^N$

The expected wait time for the first occurrence of 0000 in a random bitstring is 30.

Q. Do the same results hold for 0001?

0001 occurs much earlier than 0000

A. NO!

1011111010010100110011**1000**1001111101101101**0000**0111100001

Observation. Consider first occurrence of 000.

- 0000 and 0001 equally likely, BUT
- mismatch for 0000 means 0001, so need to wait four more bits
- mismatch for 0001 means 0000, so *next* bit could give a match.

Q. What is the probability that an N-bit random bitstring does not contain 0001?

Q. What is the expected wait time for the first occurrence of 0001 in a random bitstring?

Constructions for strings without specified patterns

Cast of characters:

p — a pattern

S_p — binary strings that do not contain p

T_p — binary strings that *end in p* *and have no other occurrence of p*

p 101001010

S_p 10111110101101001100110000011111

T_p 10111110101101001100110101001010

First construction

- S_p and T_p are disjoint
- the empty string is in S_p
- adding a bit to a string in S_p gives a string in S_p or T_p

$$S_p + T_p = E + S_p \times \{Z_0 + Z_1\}$$

Constructions for bitstrings without specified patterns

Every pattern has an autocorrelation polynomial

- slide the pattern to the left over itself.
- for each match of i trailing bits with the leading bits include a term $z^{|p|-i}$

101001010	
101001010	z^0
101001010	
101001010	
101001010	
101001010	
101001010	z^5
101001010	
101001010	z^7
101001010	

autocorrelation polynomial → $c_{101001010}(z) = 1 + z^5 + z^7$

Constructions for bitstrings without specified patterns

Second construction

- for each 1 bit in the autocorrelation of any string in T_p add a "tail"
- result is a string in S_p followed by the pattern

p 101001010

a string in T_p 101111101011010011001101**01001010**

first tail is null

strings in S_p

101111101011010011001101**01001010**

10111110101101001100110101001010**01010**

10111110101101001100110101001010**1001010**

```
        101001010
        101001010
       101001010
      101001010
     101001010
    101001010
   10100101001010
  101001010
 10100101010010 10
101001010
```

$$S_p \times \{p\} = T_p \times \sum_{c_i \neq 0} \{t_i\}$$

Bitstrings without specified patterns

How many N-bit strings do not contain a specified pattern p ?

Classes	S_p — the class of binary strings with no p
	T_p — the class of binary strings that end in p *and have no other occurence*

OGFs	$S_p(z) = \sum_{s \in S_p} z^{\|s\|}$
	$T_p(z) = \sum_{s \in T_p} z^{\|s\|}$

Constructions $\quad S_p + T_p = E + S_p \times \{Z_0 + Z_1\} \qquad S_p \times \{p\} = T_p \times \sum_{c_i \neq 0} \{t_i\}$

OGF equations $\quad S_p(z) + T_p(z) = 1 + 2zS_p(z) \qquad S_p(z)z^P = T_p(z)c_p(z)$

Solution
$$S_p(z) = \frac{c_p(z)}{z^P + (1-2z)c_p(z)}$$

$1/\beta_p$ is the smallest root of

See "Asymptotics" lecture

Extract cofficients $\quad [z^N]S_p(z) \sim c_p \beta_p^N \quad$ where $\begin{cases} \beta_p \text{ is the dominant root of } z^P + (1-2z)c_p(z) \\ c_p = \text{[explicit formula available]} \end{cases}$

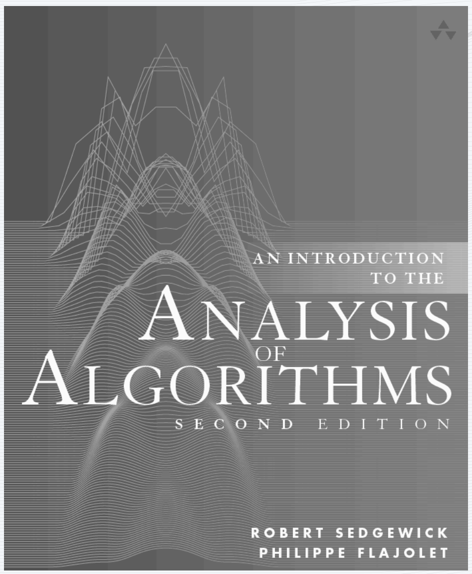
Autocorrelation for 4-bit patterns

p	auto-correlation	OGF	Probability that p does not occur in N random bits			wait time
			N	10	100	
0000 1111	1111	$\dfrac{1-z^4}{1-2z+z^5}$	$.96328^N$	0.7510	0.0259	30
0001 0011 0111 1000 1100 1110	1000	$\dfrac{1}{1-2z+z^4}$	$.91964^N$	0.4327	0.0002	16
0010 0100 0110 1001 1011 1101	1001	$\dfrac{1+z^3}{1-2z+z^3-z^4}$	$.93338^N$	0.5019	0.0010	18
0101 1010	1010	$\dfrac{1+z^2}{1-2z+z^2-2z^3+z^4}$	$.94165^N$	0.5481	0.0024	20

off by < 10%
but indicative

constants omitted
(close to 1)

Example. In 100 random bits,
0000 is ~10 times more likely to be absent than 0101
~100 times more likely to be absent than 0001.



http://aofa.cs.princeton.edu

8. Strings and Tries

- Bitstrings with restrictions
- Languages
- Tries
- Trie parameters

Formal languages and the symbolic method

Definition. A formal language is a *set* of strings.

Q. How many strings of length N in a given language?

A. Use an OGF to enumerate them.

$$S(z) = \sum_{s \in \mathcal{S}} z^{|s|}$$

Remark. The symbolic method provides a systematic approach to this problem.

Issue. Ambiguity.

Regular expressions

Theorem. Let A and B be *unambiguous* REs with OGFs $A(z)$ and $B(z)$. If A + B, AB, and A* are also unambiguous, then

$$A(z) + B(z) \quad \text{enumerates A + B}$$

$$A(z)B(z) \quad \text{enumerates AB}$$

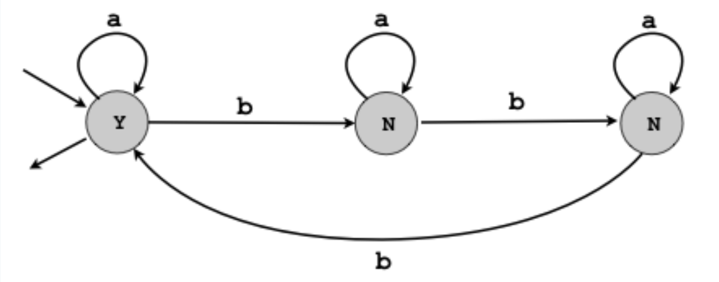
$$\frac{1}{1 - A(z)} \quad \text{enumerates A*}$$

Proof.
Same as for symbolic method—different notation.

OGF for an unambiguous RE is *rational* — can be written as the ratio of two polynomials.

Corollary. OGFs that enumerate *regular languages* are rational.

Proof.
1. There exists an FSA for the language.
2. *Kleene's theorem* gives an unambiguous RE for the language defined by any FSA.



`a* | (a*ba*ba*ba*)*`

Example 1. Binary strings with no 000

RE. $$(1 + 01 + 001 + 001)^*(\epsilon + 0 + 00 + 00)$$

OGF. $$\begin{aligned} S_4(z) &= \frac{1 + z + z^2 + z^3}{1 - (z + z^2 + z^3 + z^4)} \\ &= \frac{\dfrac{1 - z^4}{1 - z}}{1 - z\dfrac{1 - z^4}{1 - z}} \\ &= \frac{1 - z^4}{1 - 2z + z^5} \quad \checkmark \end{aligned}$$

Expansion. $$[z^N]S_4(z) \sim c_4\beta_4^N \quad \text{with} \begin{cases} \beta_4 \doteq 1.92756 \\ c_4 \doteq 1.09166 \end{cases}$$

Example 2. Binary strings that represent multiples of 3

RE. $$(1(01^*0)^*10^*)^*$$

OGF. $$D_3(z) = \cfrac{1}{1 - \cfrac{z^2}{1 - \cfrac{z^2}{1-z}}\left(\cfrac{1}{1-z}\right)} = \cfrac{1}{1 - \cfrac{z^2}{1-z-z^2}}$$

$$= 1 - \frac{z^2}{(1-2z)(1+z)}$$

Expansion. $$[z^N]D_3(z) \sim \frac{2^{N-1}}{3} \quad \checkmark$$

11
110
1001
1100
1111
10010
10101
11000
11011
11110
100001
100100
...

Context-free languages

Theorem. Let <A> and <B> be nonterminals in an *unambiguous* CFG with OGFs $A(z)$ and $B(z)$. If <A> | <B> and <A><B> are also unambiguous, then

$A(z) + B(z)$ enumerates <A> | <B>

$A(z)B(z)$ enumerates <A><B>

Proof.
Same as for symbolic method—different notation.

Corollary. OGFs that enumerate unambiguous CF languages are *algebraic*.

Proof.
"Gröbner basis" elimination—see text.

An *algebraic function* is a function that satisfies a polynomial equation whose coefficients are polynomials with rational coefficients

Context-free languages

The unlabelled constructions we have considered *are* CFGs, using different notation.

class	construction	CFG	OGF (algebraic)
Binary Trees	$T = E + T \times Z \times T$	<T> := <E> <T> := <T><Z><T>	$T(z) = 1 + zT(z)^2$
Bitstrings	$B = E + (Z_0 + Z_1) \times B$	<B> := <E> <Y> := $\langle Z_0\rangle$ | $\langle Z_1\rangle$ <B> := <Y> × <B>	$B(z) = 1 + 2zB(z)$
Bitstrings with no 00	$B_{00} = (E + Z_0)$ $\times (E + Z_1 \times B_{00})$	$\langle Y_0\rangle$:= <E> | $\langle Z_0\rangle$ $\langle Y_1\rangle$:= $\langle Z_1\rangle$ × $\langle B_{00}\rangle$ $\langle Y_2\rangle$:= <E> + $\langle Y_1\rangle$ $\langle B_{00}\rangle$:= $\langle Y_0\rangle$ | $\langle Y_2\rangle$	$B_{00}(z) = 1 + z$ $+ (z + z^2)B_{00}(z)$

Note 1. Not all CFGs correspond to combinatorial classes (ambiguity).

Note 2. Not all constructions are CFGs (many other operations have been defined).

Walks

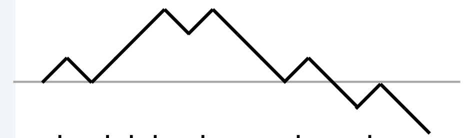
Definition. A walk is a sequence of + and – characters.

Sample applications:

- Parenthesis systems
- Gambler's ruin problems
- Inversions in 2-ordered permutations (see text)

()((()()))())(())

+-+++-+---+--+--

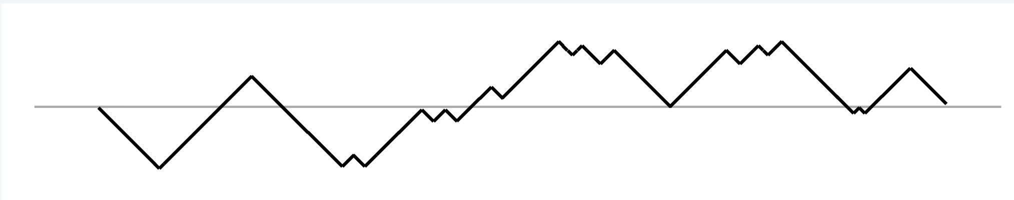


+-+++-+---+--+--

Q. How many different walks of length N ?

Q. How many different walks of length N where every prefix has more + than – ?

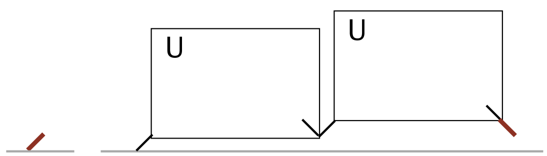
Unambiguous decomposition of walks



<U>:

- start with +
- end at +1
- never hit 0

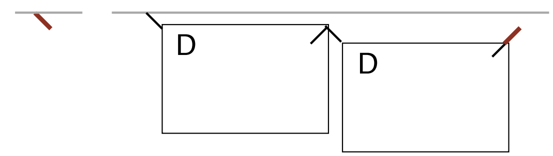
<U> := <+> | <U><U><–>



<D>:

- start with –
- end at –1
- never hit 0

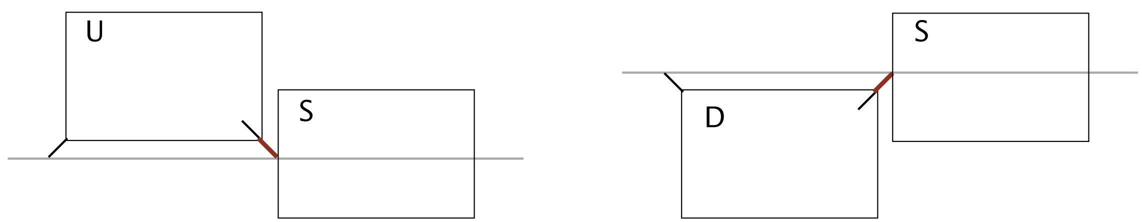
<D> := <–> | <D><D><+>



<S>:

- begin at 0
- end at 0

<S> := <U><–><S> | <D><+><S>



Context-free languages

Example. Walks of length $2N$ that start at and return to 0

CFL.

<S> := <U><–><S> | <D><+><S> | ε
<U> := <U><U><–> | <+>
<D> := <D><D><+> | <–>

OGFs.

$$S(z) = zU(z)S(z) + zD(z)S(z) + 1$$

$$U(z) = z + zU^2(z)$$

$$D(z) = z + zD^2(z)$$

Solve simultaneous equations.

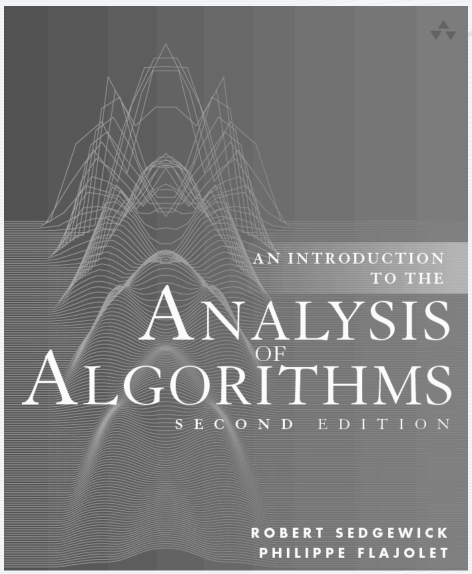
$$U(z) = D(z) = \frac{1}{2z}\left(1 - \sqrt{1 - 4z^2}\right)$$

$$S(z) = \frac{1}{1 - 2zU(z)} = \frac{1}{\sqrt{1 - 4z^2}}$$

Expand.

$$[z^{2N}]S(z) = \binom{2N}{N}$$

Elementary example, but extends to similar, more difficult problems



http://aofa.cs.princeton.edu

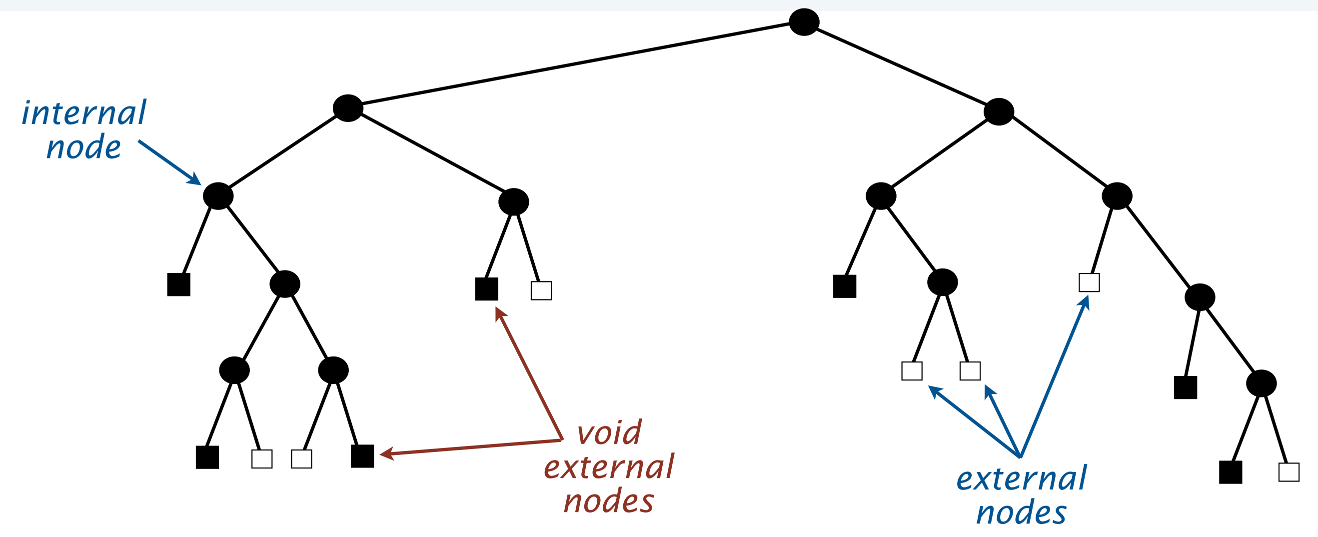
8. Strings and Tries

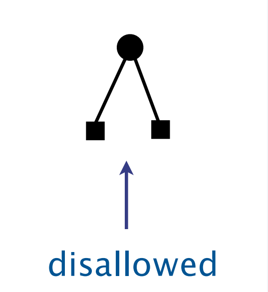
- Bitstrings with restrictions
- Languages
- **Tries**
- Trie parameters

Tries

Definition. A trie is a binary tree with the following properties:

- External nodes may be void (■)
- Siblings of void nodes are *not* void (● or □).



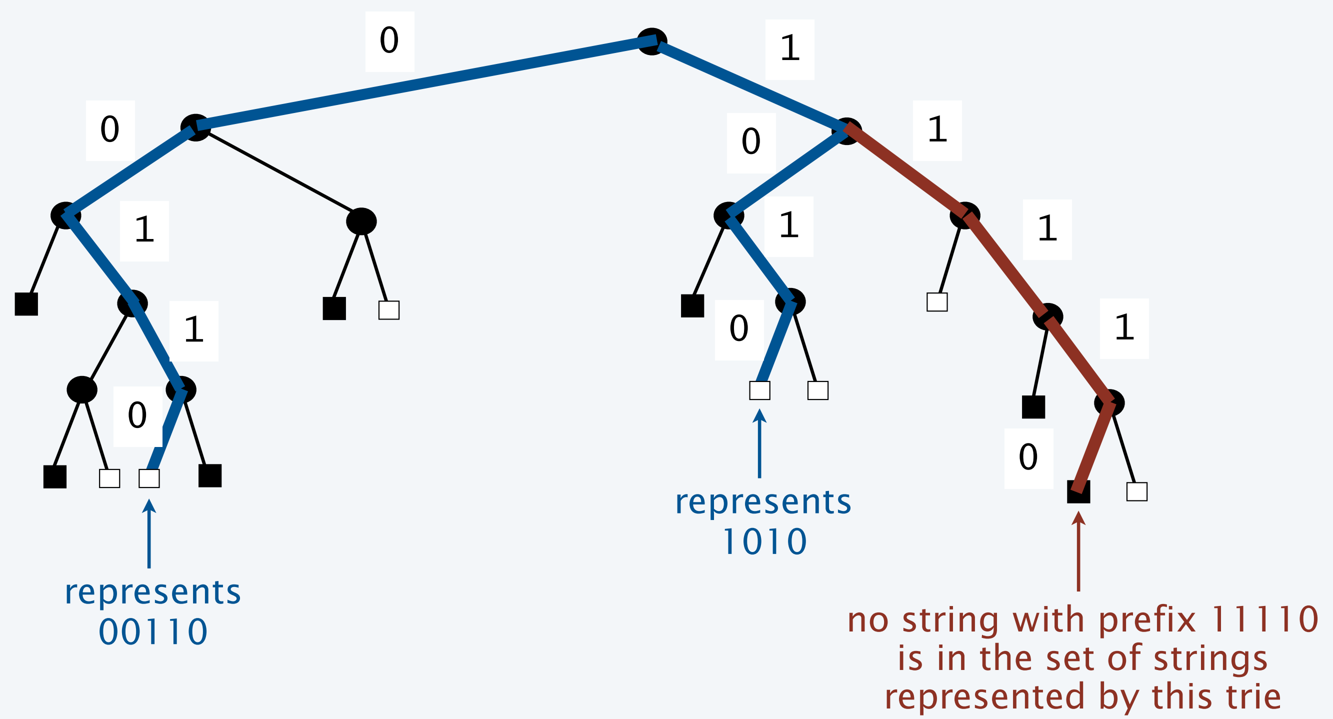


Ex. Give a recursive definition.

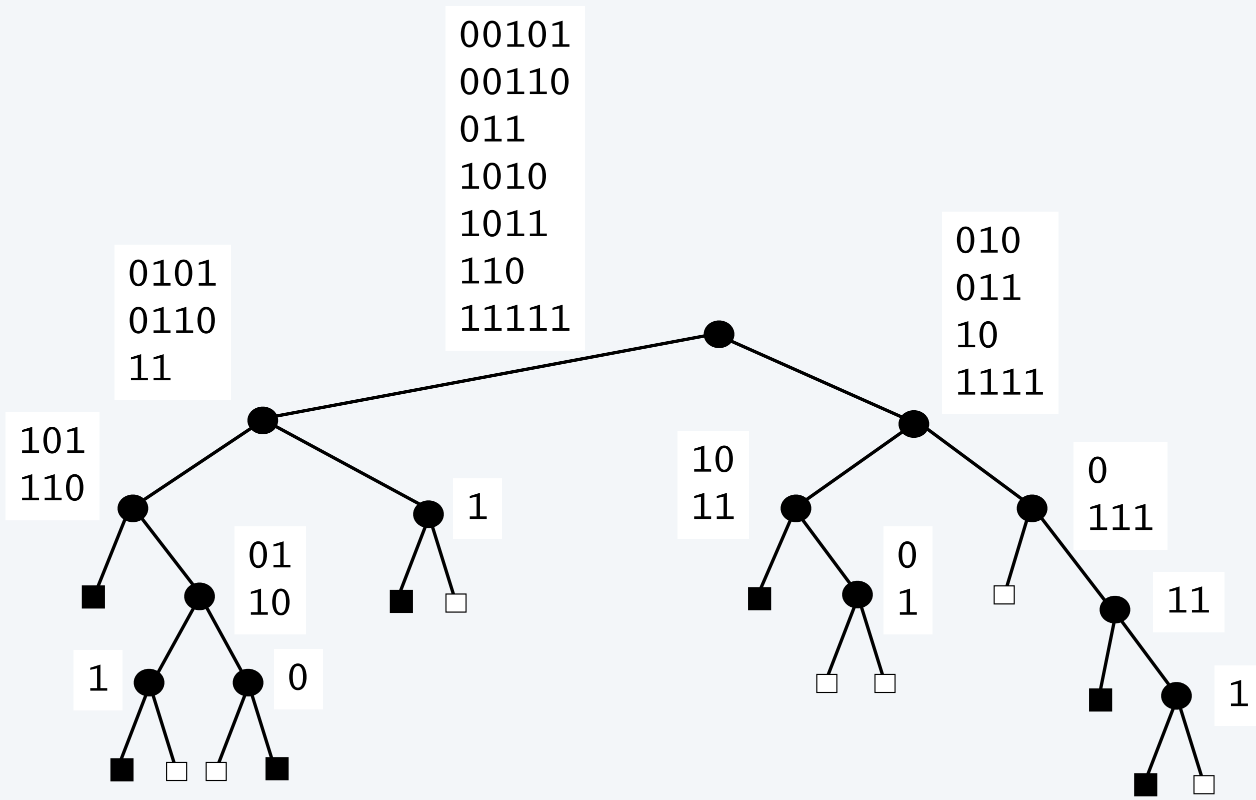
Tries and sets of bitstrings

Each trie corresponds to a set of bitstrings.

- Each nonvoid external node represents one bitstring.
- Path from the root to a node defines the bitstring



Tries and sets of bitstrings



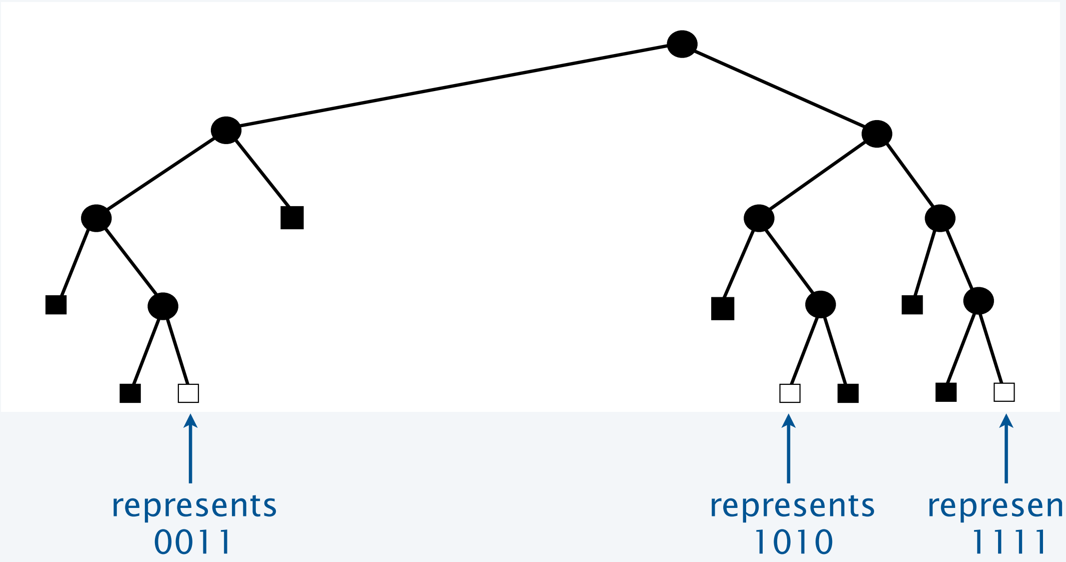
Note: Works only for *prefix-free* sets of bitstrings (or use void/nonvoid *internal* nodes).

no member is a prefix of another

Tries and sets of bitstrings (fixed length)

If all the bitstrings in the set are the same length, it is prefix-free.

0011
1010
1111



Trie applications

Searching and sorting

- MSD radix sort
- Symbol tables with string keys
- Suffix arrays

Data compression

- Huffman and prefix-free codes
- LZW compression

Application areas:
Network systems
Bioinformatics
Internet search
Commercial data processing

Decision making

- Collision resolution
- Leader election

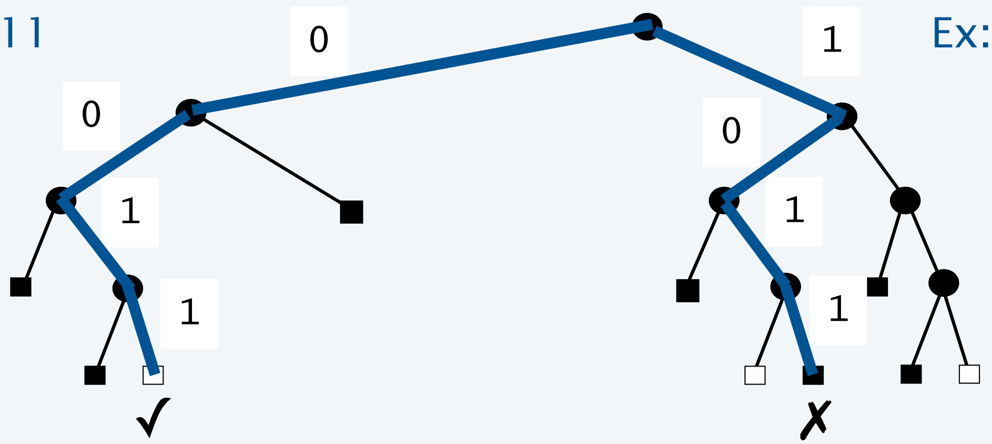
Trie application 1: Symbol tables

Search

- If at nonvoid external node and no bits left in bitstring, report success.
- If at void external node, report failure.
- If leading bit is 0, search in the left subtrie (using remainder of string).
- If leading bit is 1, search in the right subtrie (using remainder of string).

Ex: search for 0011

Ex: search for 10110



Q. Expected search time ?

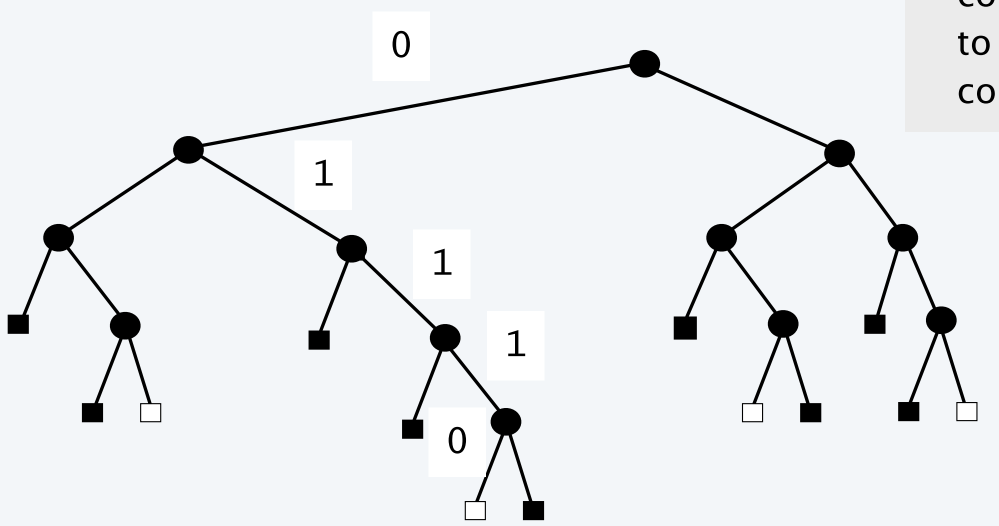
Trie application 1: Symbol tables

Insert

- Search to void external node (prefix-free violation if nonvoid external node hit).
- Add internal nodes (each with one void external child) for each remaining bit.

variant:
convert the void external node
to a nonvoid external node that
contains a pointer to the "tail"

Ex: insert 01110



Q. How many void nodes ?

Trie application 2: Substring search index

Problem: Build an index that supports fast *substring search* in a given string *S*.

	0	1	2	3	4	5	6	7	8	9
Ex. S ⟶	A	C	C	T	A	G	G	C	C	T

Q. Is ACCTA in S?
A. Yes, starting at 0.

Q. Is CCT in S?
A. Yes, in multiple places.

Q. Is TGA in S?
A. No.

Solution: Use a *suffix multiway trie*.

Application 1: Search in genomic data.

Application 2: Internet search.

Trie application 2: Substring search index

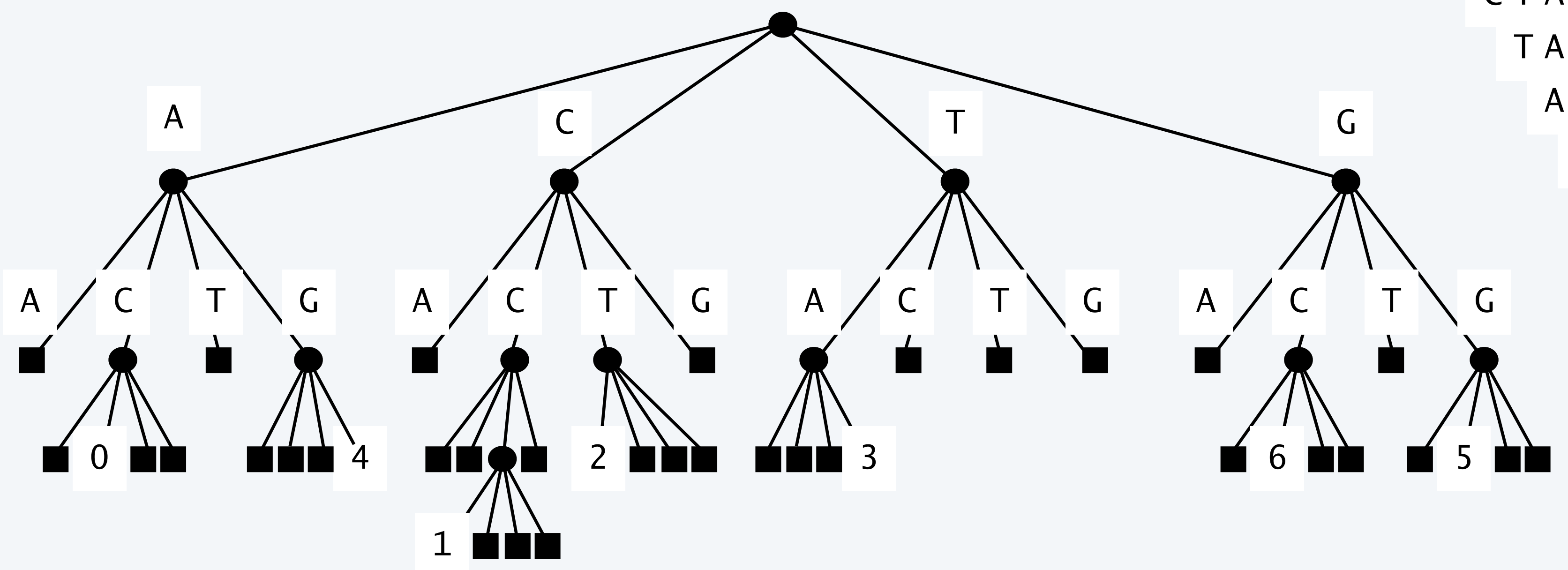
To build the *suffix multiway trie* associated with a string *S*

- Insert the substrings starting at each position into an initially empty trie.
- Associate a string index with each nonvoid external node.

a prefix-free set

0	1	2	3	4	5	6	7	8	9
A	C	C	T	A	G	G	C	C	T
	C	C	T	A	G	G	C	C	T
		C	T	A	G	G	C	C	T
			T	A	G	G	C	C	T
				A	G	G	C	C	T
					G	G	C	C	T
						G	C	C	T
							C	C	T
								C	T
									T

Property: *Every* internal node corresponds to a substring of *S*

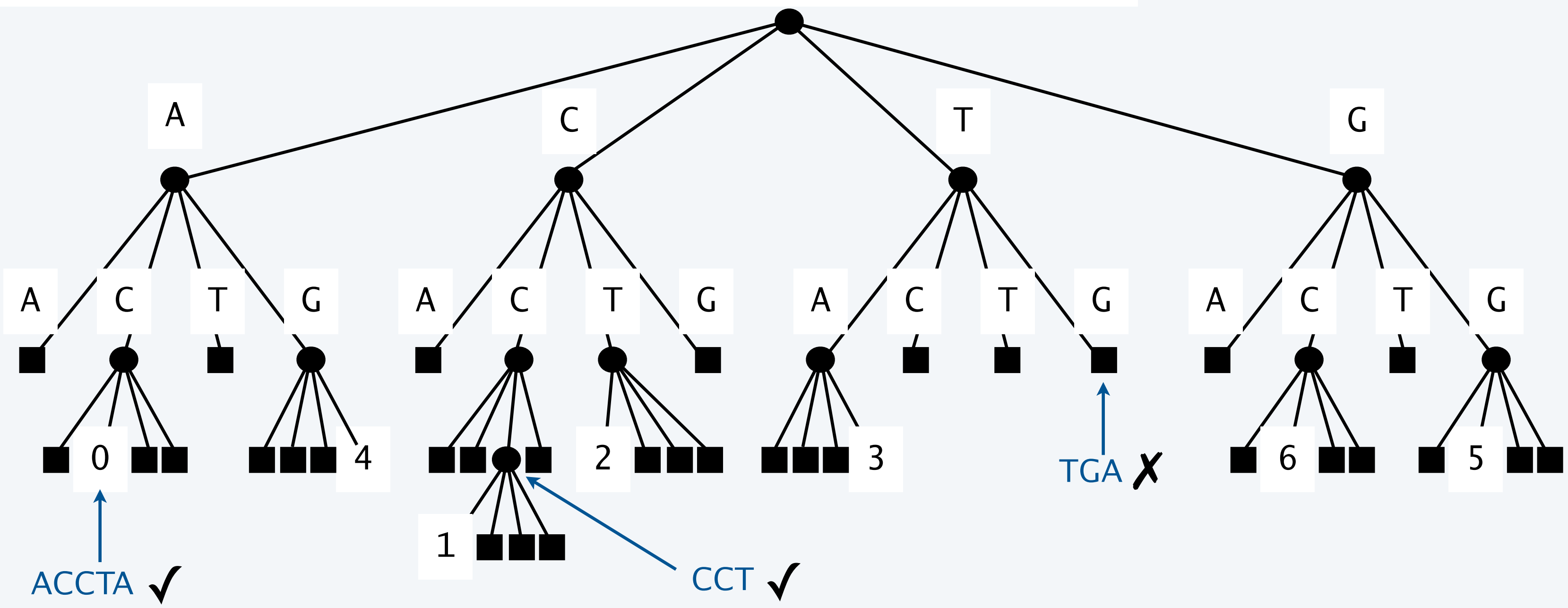


Trie application 2: Substring index

To use a suffix tree to answer the query *Is X a substring of S* ?

- Use the characters of *X to traverse the trie.*
- Continue in string when nonvoid node encountered.
- Report failure if void node encountered.
- Report success when end of *X* reached.

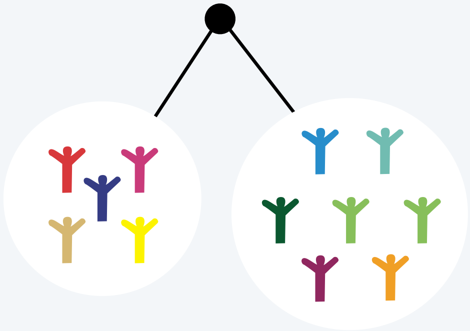
0	1	2	3	4	5	6	7	8	9
A	C	C	T	A	G	G	C	C	T



Trie application 3: Elect a leader

Problem: Elect a *leader* among a group of individuals.

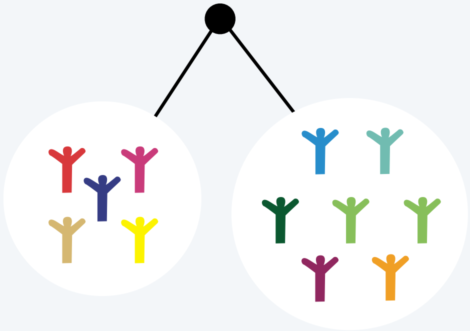
Trie application 3: Elect a leader



Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

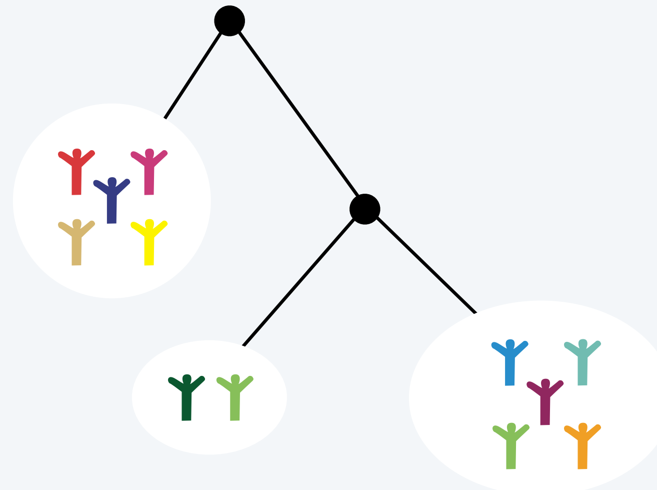
Trie application 3: Elect a leader



Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

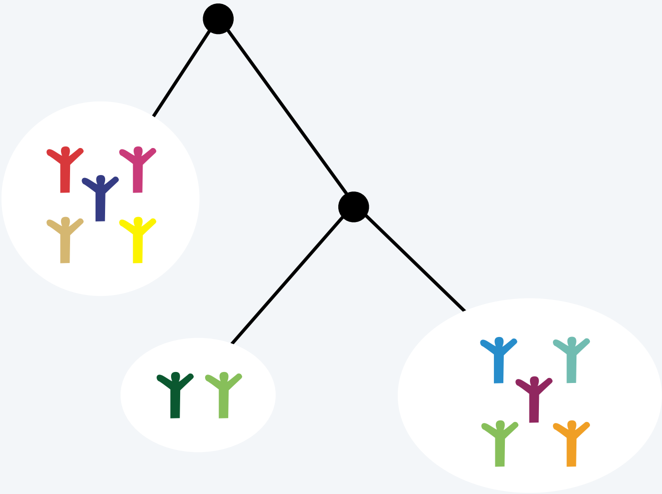
Trie application 3: Elect a leader



Method.

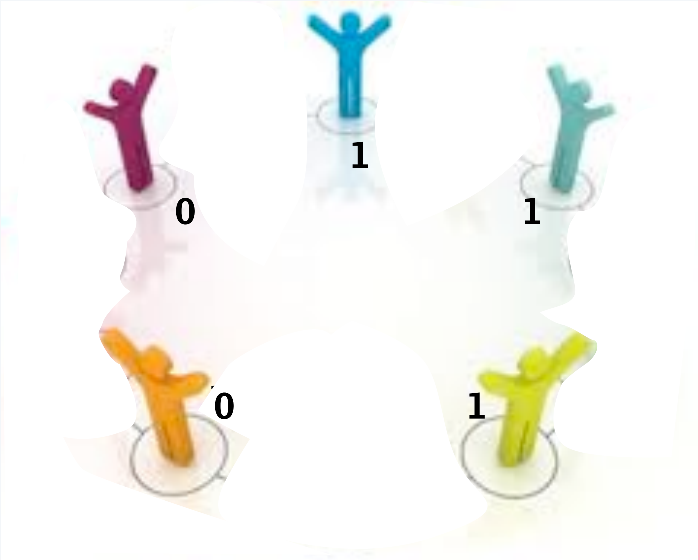
- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

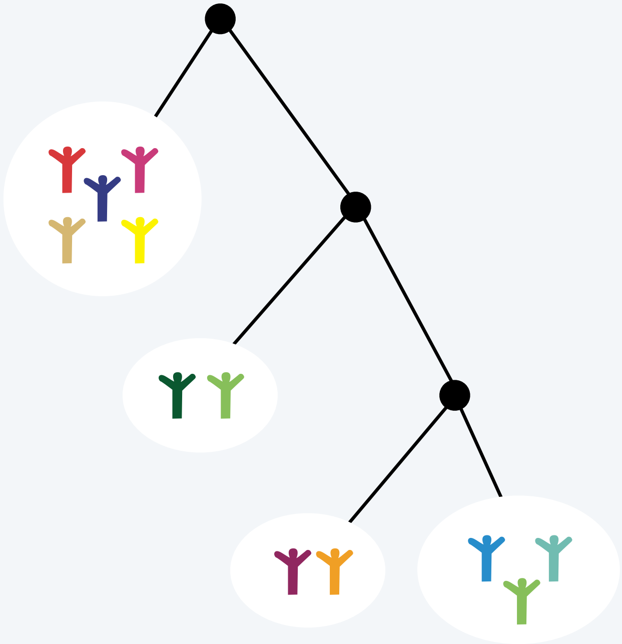
Trie application 3: Elect a leader



Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

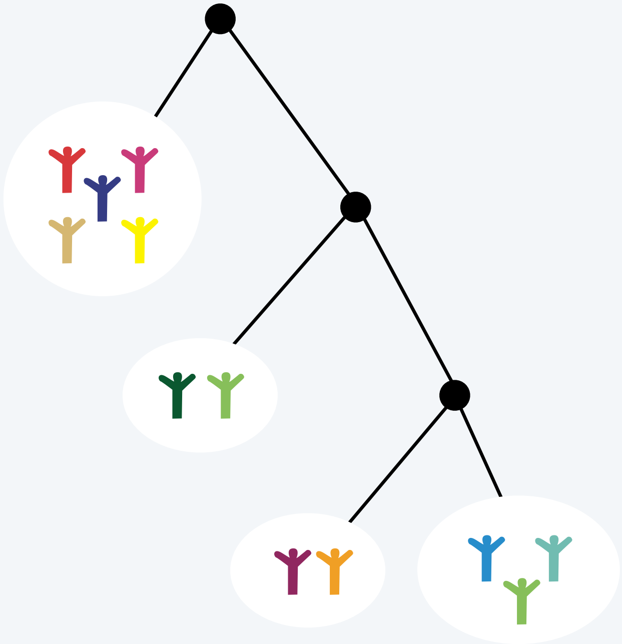




Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

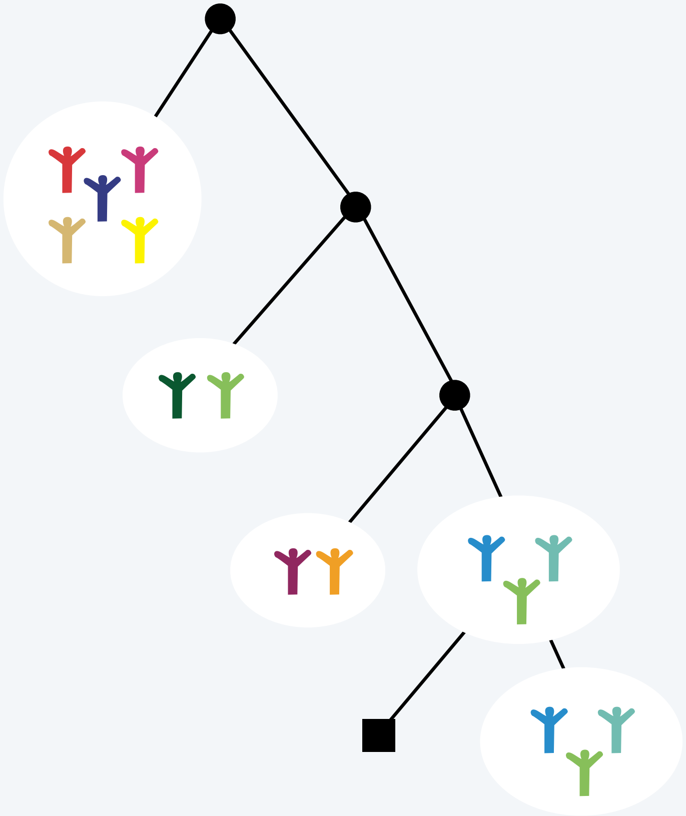
Trie application 3: Elect a leader



Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

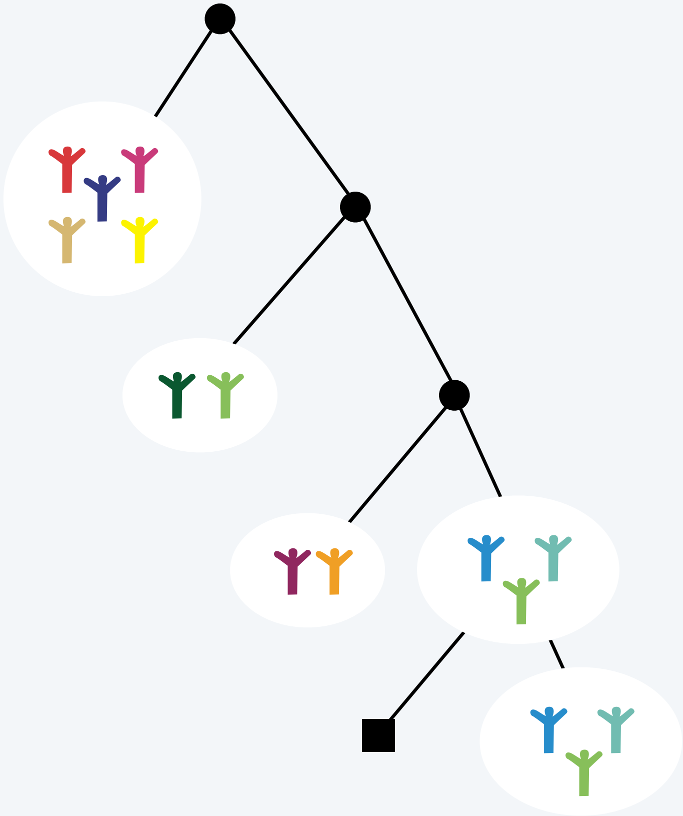
Trie application 3: Elect a leader



Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

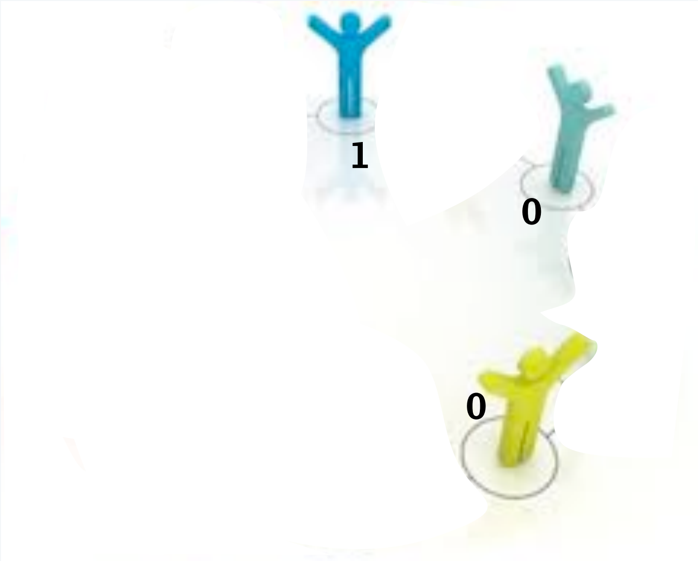
Trie application 3: Elect a leader

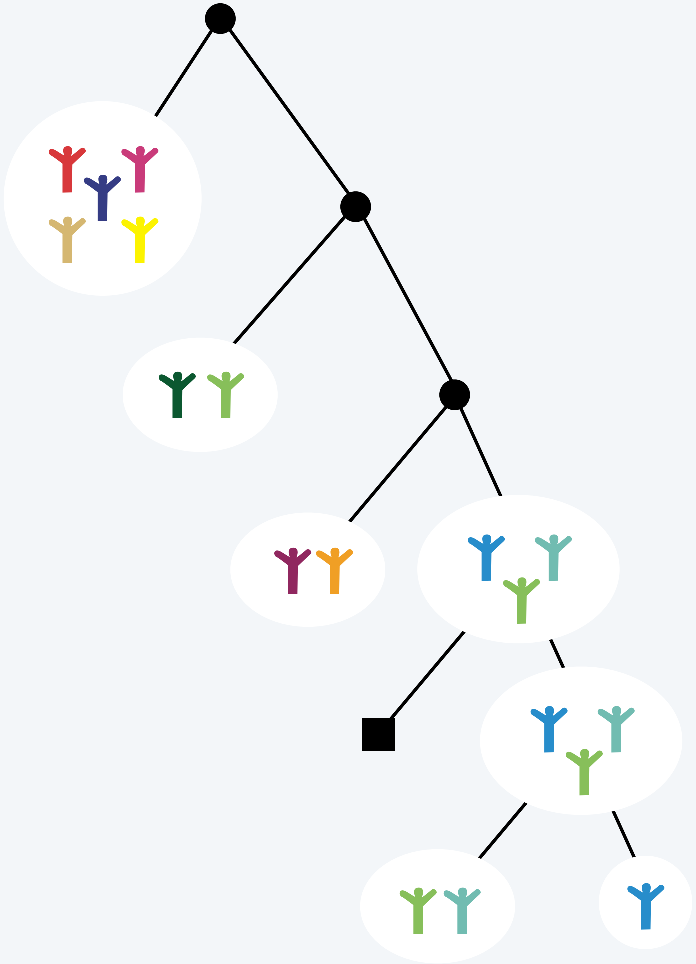


Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

Trie application 3: Elect a leader

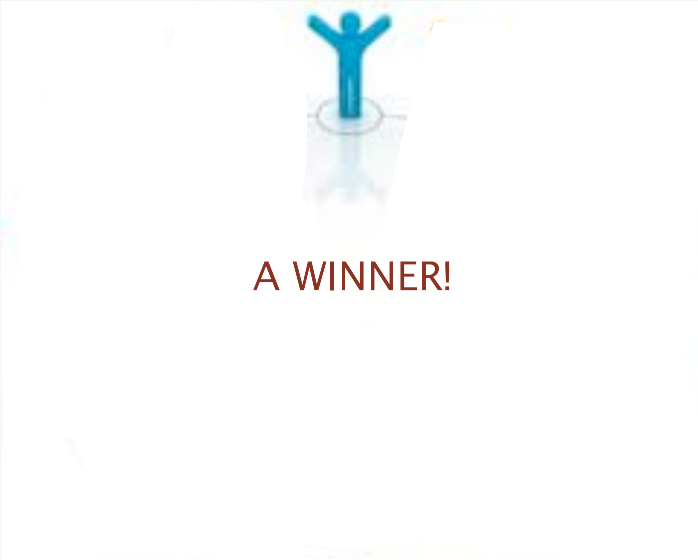


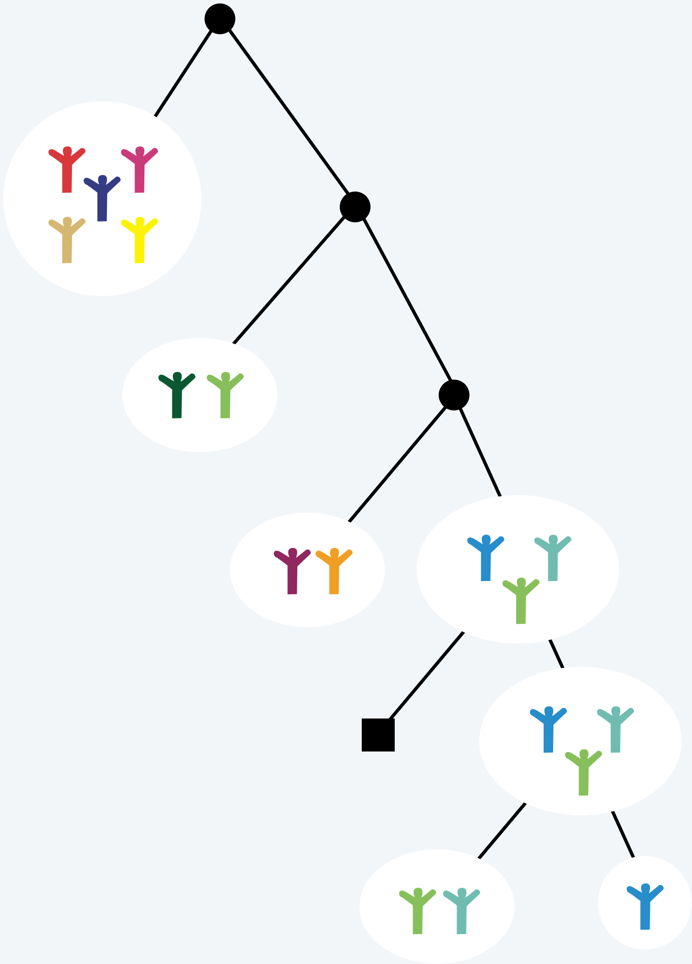


Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

Trie application 3: Elect a leader



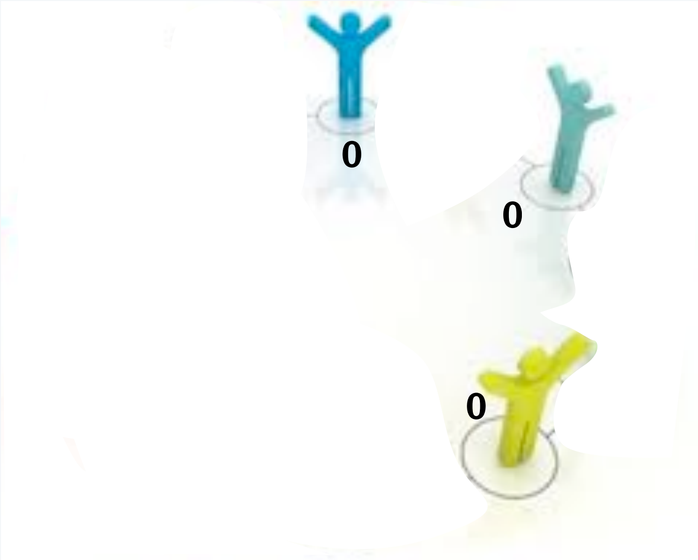


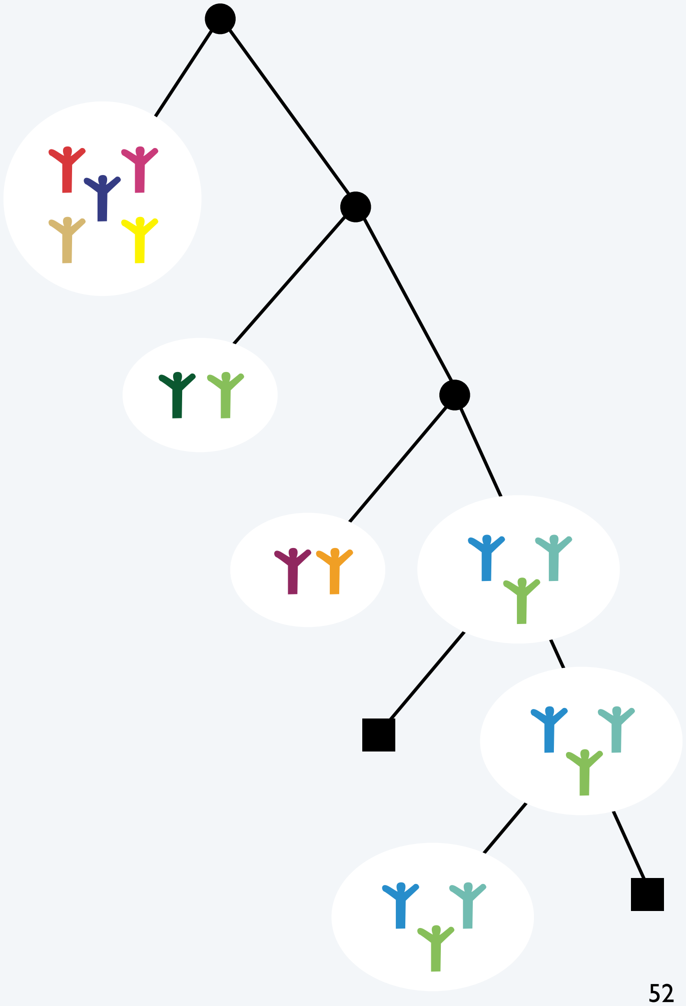
Method.

- Each person flips a 0-1 coin.
- 1 wins, 0 loses
- Winners continue to next round.

Trie application 3: Elect a leader

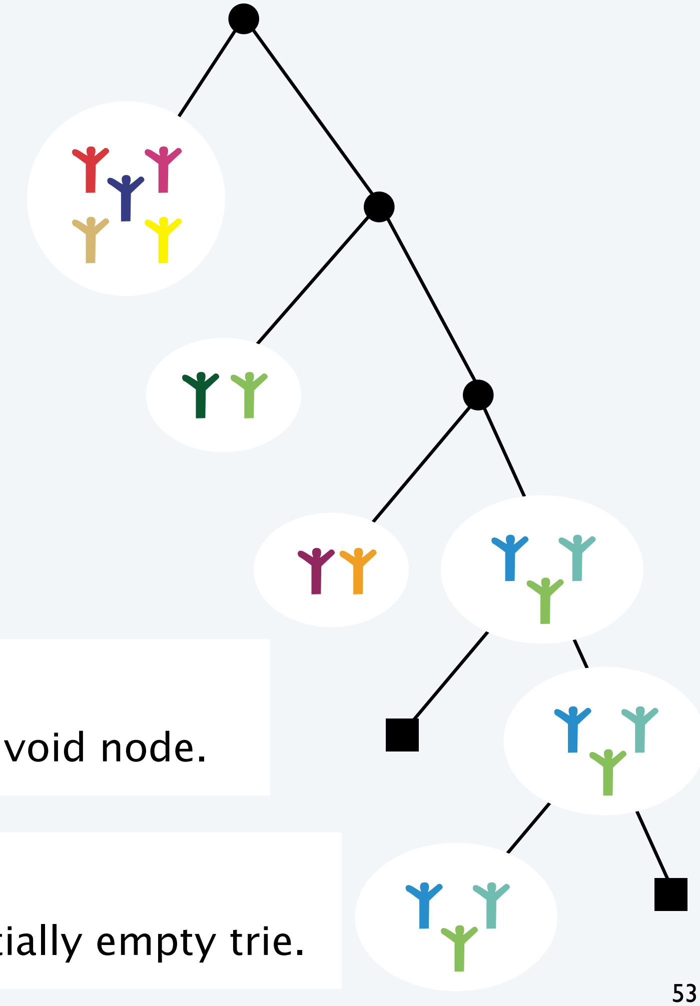
Procedure might fail!





Trie application 3: Elect a leader

a set of losers



Procedure might fail!

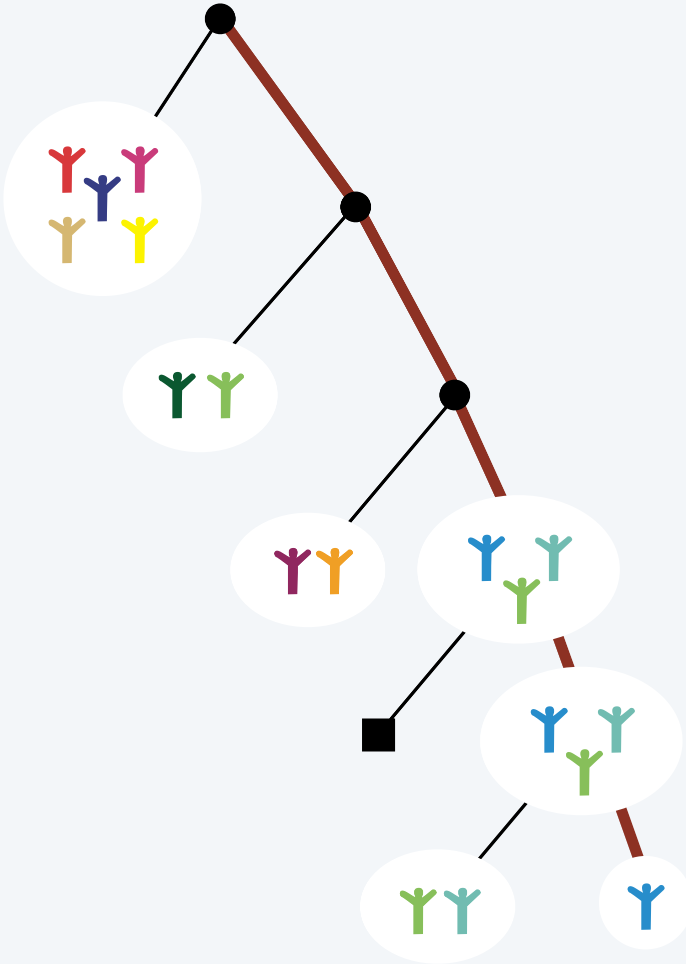
Q. What is the chance of failure?

A. Probability that the rightmost path in a random trie ends in a void node.

Q. What is a random trie?

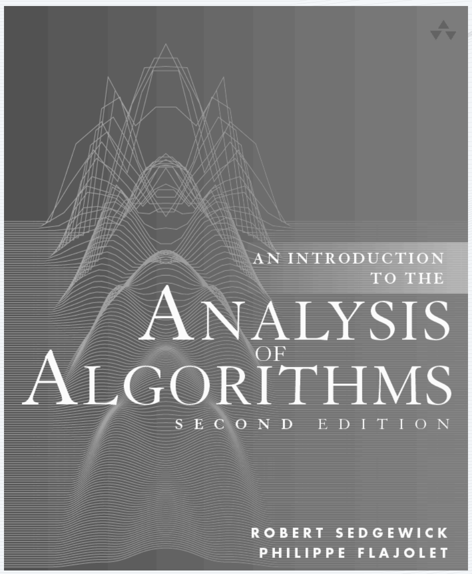
A. Built by inserting infinite-length random bitstrings into an initially empty trie.

Trie application 3: Elect a leader



Q. How many rounds in a distributed leader election?

A. Expected length of the rightmost path in a random trie.



http://aofa.cs.princeton.edu

8. Strings and Tries

- Bitstrings with restrictions
- Languages
- Tries
- **Trie parameters**

Analysis of trie parameters

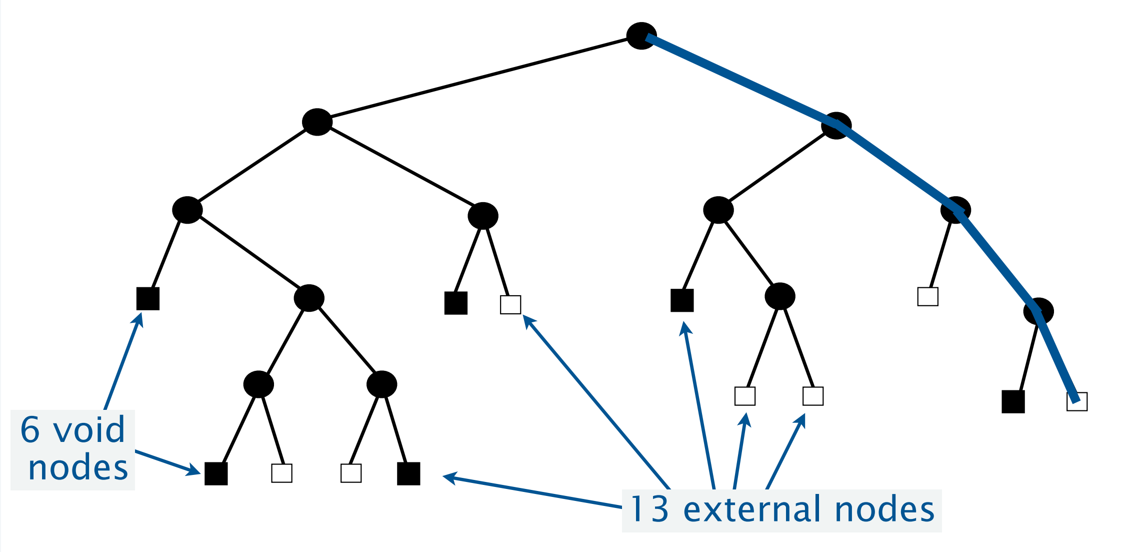
is the basis of understanding performance in numerous large-scale applications.

Q. Space requirement?
A. Number of external nodes.

Q. "Extra" space ?
A. Number of void nodes.

Q. Expected search cost?
A. External path length.

Q. Rounds in leader election?
A. Length of rightmost path.

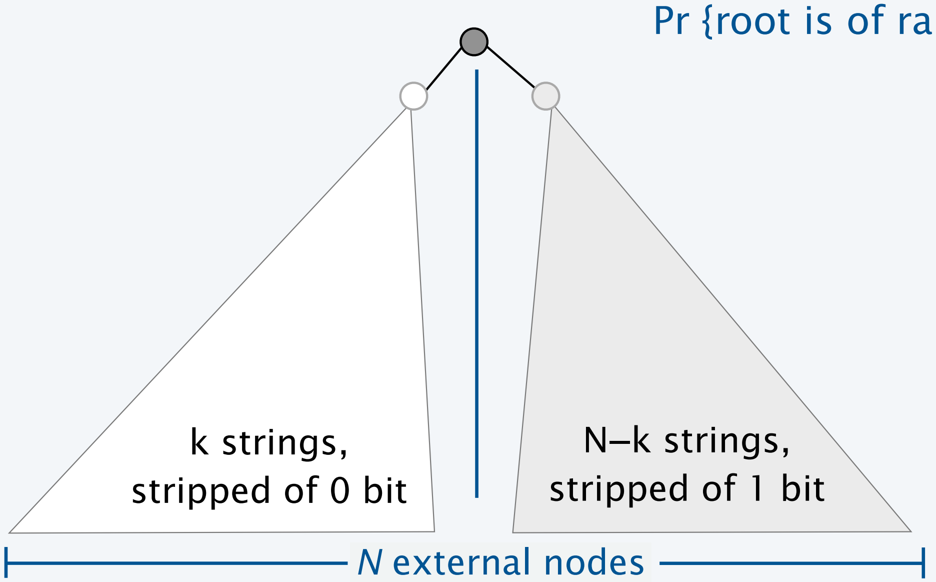


$(3 + 5 + 5 + 5 + 5 + 3 + 3 + 3 + 4 + 4 + 3 + 4 + 4)/13$
$\doteq 3.92$

Usual model: Build trie from N *infinite* random bitstrings (nonvoid nodes represent tails)

Average external path length in a trie

Recurrence. [For comparison with BST and Catalan models.]



Pr {root is of rank k}

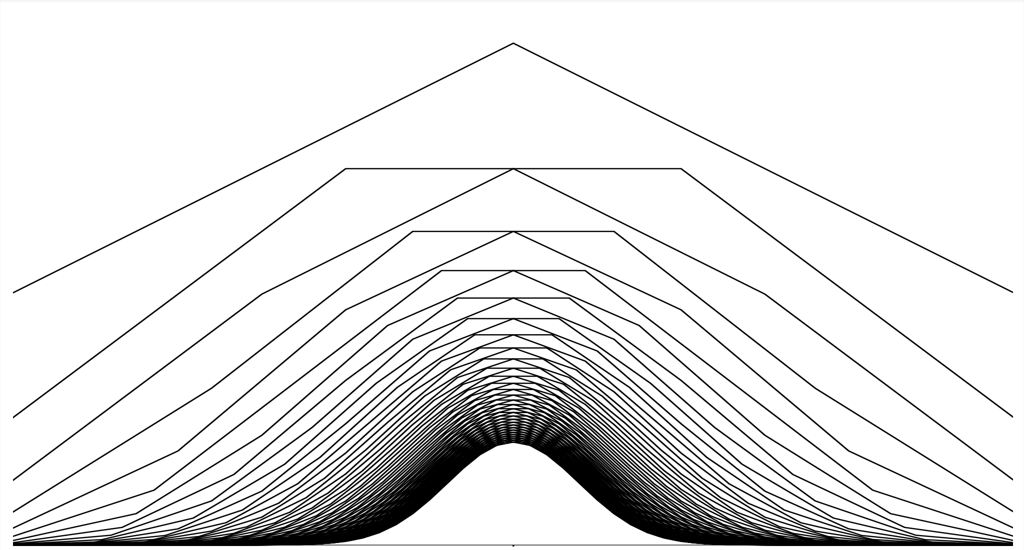
BST	$\dfrac{1}{N}$
Catalan	$\dfrac{\dfrac{1}{k}\dbinom{2k-2}{k}\dfrac{1}{N-k+1}\dbinom{2N-2k}{N-k}}{\dfrac{1}{N+1}\dbinom{2N}{N}}$
Trie	$\dfrac{1}{2^N}\dbinom{N}{k}$

$$C_N = N + \frac{1}{2^N}\sum_k \binom{N}{k}(C_k + C_{N-k}) \text{ for } N > 1 \text{ with } C_0 = C_1 = 0$$

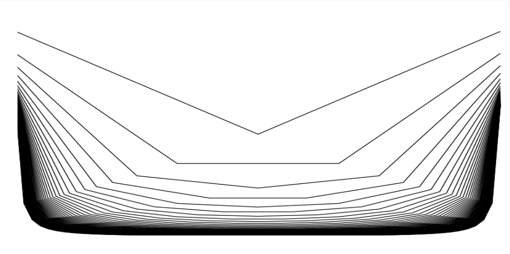
Caution: When $k = 0$ and $k = N$, C_N appears on right-hand side.

Probability that the root is of rank *k* in a random tree.

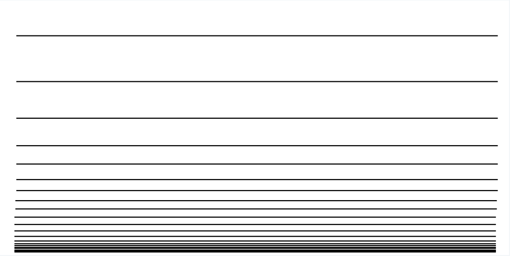
Trie built from random bitstrings



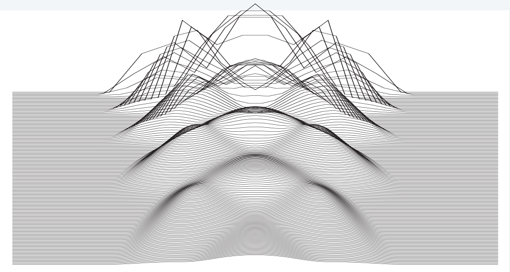
Random binary tree



BST built from random perm



AVL tree



Average external path length in a trie

Recurrence. $$C_N = N + \frac{1}{2^N}\sum_k \binom{N}{k}(C_k + C_{N-k}) \text{ for } N > 1 \text{ with } C_0 = C_1 = 0$$

EGF

$$C(z) = \sum_{N\geq 0} C_N \frac{z^N}{N!}$$

GF equation. $$C(z) = ze^z - z + 2e^{z/2}C(z/2)$$ ← Also available directly through symbolic method

$$= ze^z - z + 2e^{z/2}\Big(\frac{z}{2}e^{z/2} - \frac{z}{2} + 2e^{z/4}C(z/4)\Big)$$

$$= z(e^z - 1) + z(e^z - e^{z/2}) + 4e^{3z/4}C(z/4)$$

$$= z(e^z - 1) + z(e^z - e^{z/2}) + z(e^z - e^{3z/4}) + 8e^{7z/8}C(z/8)$$

Iterate. $$C(z) = z\sum_{j\geq 0}\Big(e^z - e^{(1-2^{-j})z}\Big)$$

Expand. $$C_N = N![z^N]C(z) = N\sum_{j\geq 0}\Big(1 - \Big(1 - \frac{1}{2^j}\Big)^{N-1}\Big)$$

Approximate (exp-log) $$C_N \sim N\sum_{j\geq 0}(1 - e^{-N/2^j}) \sim N\lg N$$ ← See next slide

Goal: isolate periodic terms

$$\sum_{j\ge 0}(1-e^{-N/2^j}) = \sum_{0\le j<\lfloor \lg N\rfloor}(1-e^{-N/2^j}) + \sum_{j\ge\lfloor \lg N\rfloor}(1-e^{-N/2^j})$$

$$= \lfloor \lg N\rfloor - \sum_{0\le j<\lfloor \lg N\rfloor}(e^{-N/2^j}) + \sum_{j\ge\lfloor \lg N\rfloor}(1-e^{-N/2^j})$$

$$= \lfloor \lg N\rfloor - \sum_{j<\lfloor \lg N\rfloor}(e^{-N/2^j}) + \sum_{j\ge\lfloor \lg N\rfloor}(1-e^{-N/2^j}) + O(e^{-N})$$

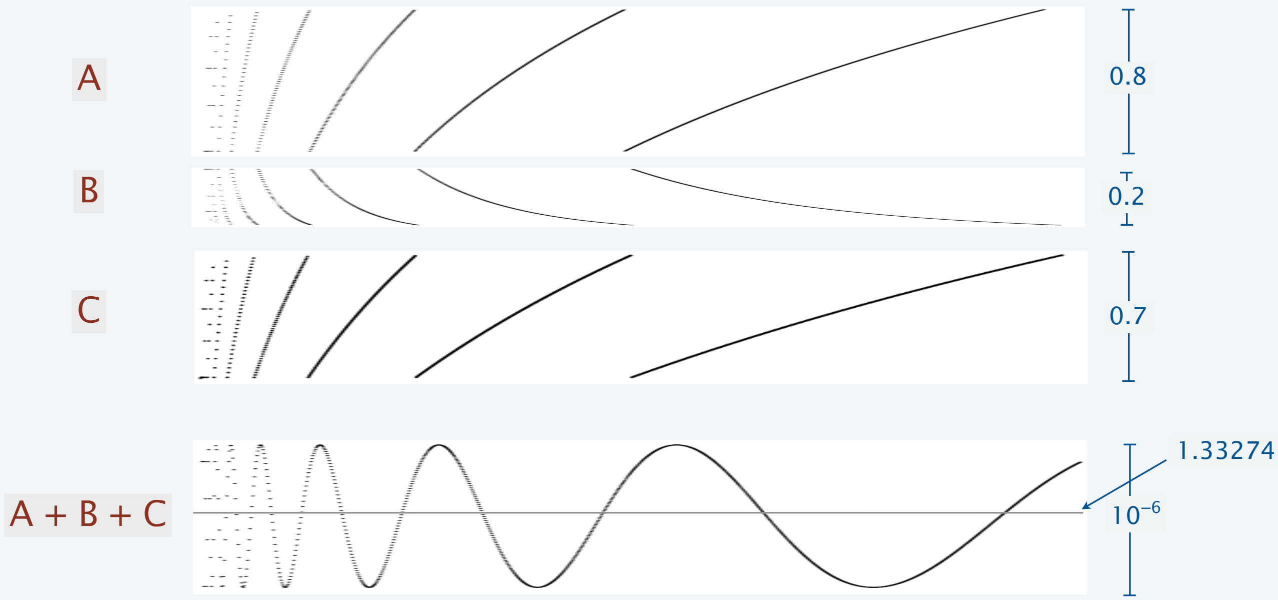
$$= \lfloor \lg N\rfloor - \sum_{j<0}(e^{-N/2^{j+\lfloor \lg N\rfloor}}) + \sum_{j\ge 0}(1-e^{-N/2^{j+\lfloor \lg N\rfloor}}) + O(e^{-N})$$

$$= \lg N - \{\lg N\} - \sum_{j<0} e^{-2^{\{\lg N\}-j}} + \sum_{j\ge 0}(1-e^{-2^{\{\lg N\}-j}}) + O(e^{-N}) \quad ✓$$

Average external path length in a trie

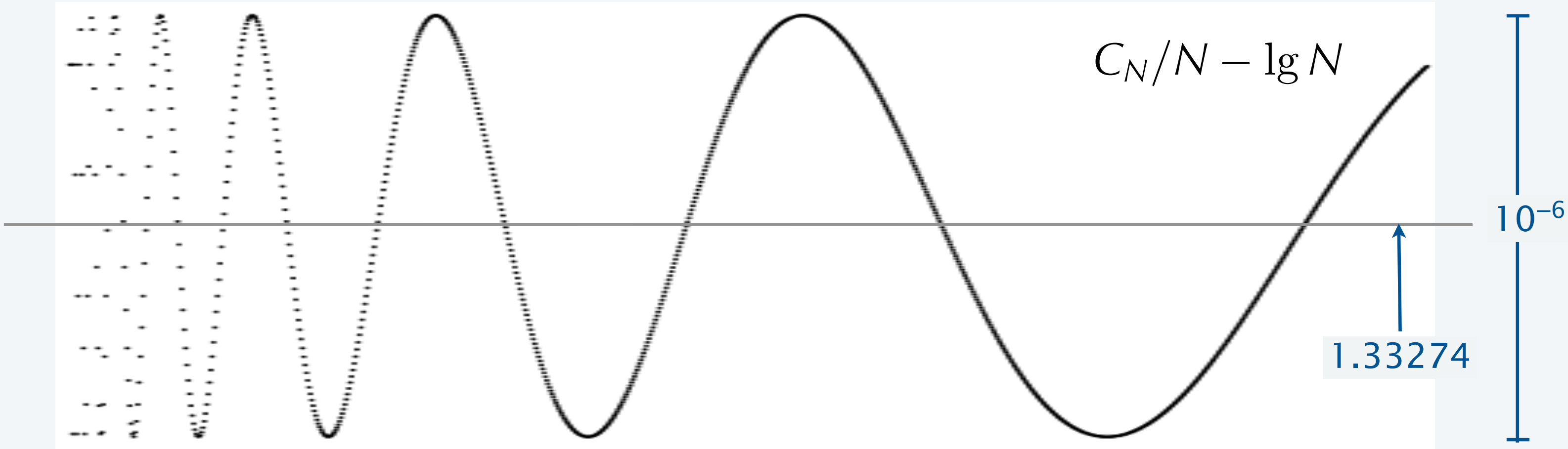
Q. $$C_N = N + \frac{1}{2^N}\sum_k \binom{N}{k}(C_k + C_{N-k}) \text{ for } N > 1 \text{ with } C_0 = C_1 = 0$$

A. $$C_N/N = \underbrace{\lg N - \{\lg N\}}_{A} - \underbrace{\sum_{j<0} e^{-2^{\{\lg N\}-j}}}_{B} + \underbrace{\sum_{j\geq 0}(1 - e^{-2^{\{\lg N\}-j}})}_{C} + O(e^{-N})$$



Fluctuating term in trie (and other AofA) results

$$C_N = N + \frac{1}{2^N}\sum_k \binom{N}{k}(C_k + C_{N-k}) \text{ for } N > 1 \text{ with } C_0 = C_1 = 0$$

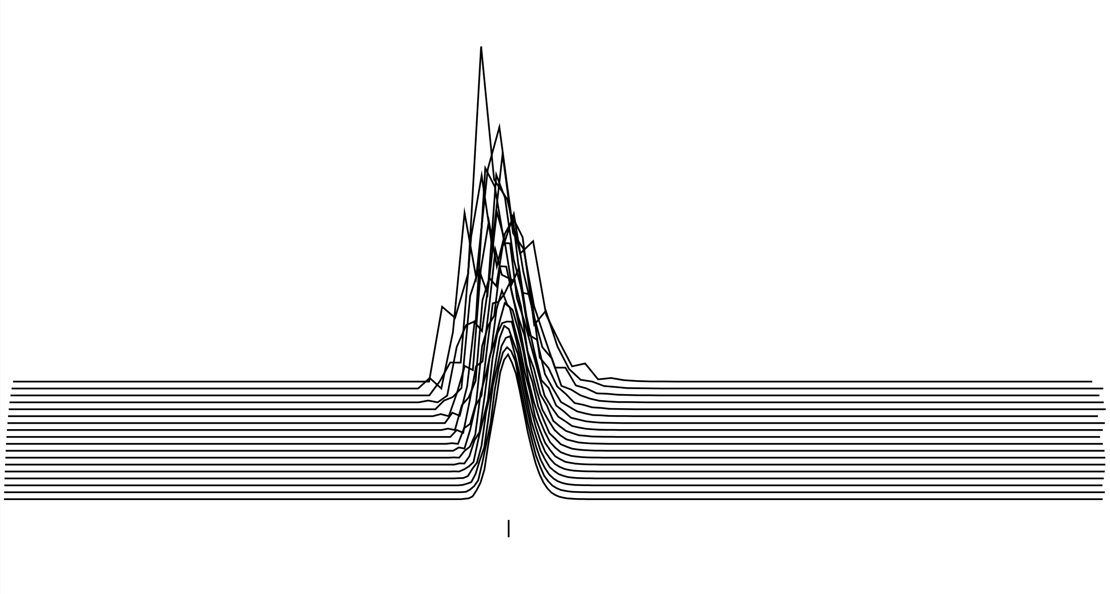


Q. Is there a reason that such a recurrence should imply such periodic behavior?

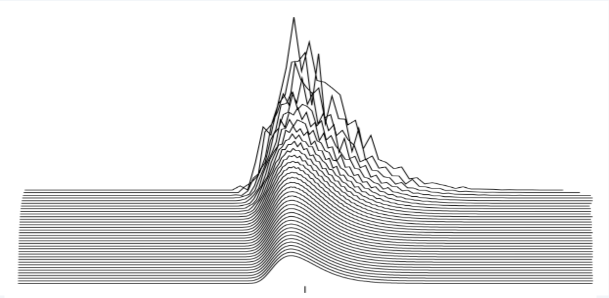
A. Yes. Stay tuned for the Mellin transform and related topics in Part II.

Average external path length distribution

Trie built from random bitstrings



BST built from random perm



Analysis of trie parameters

is the basis of understanding performance in numerous large-scale applications.

Q. Space requirement?

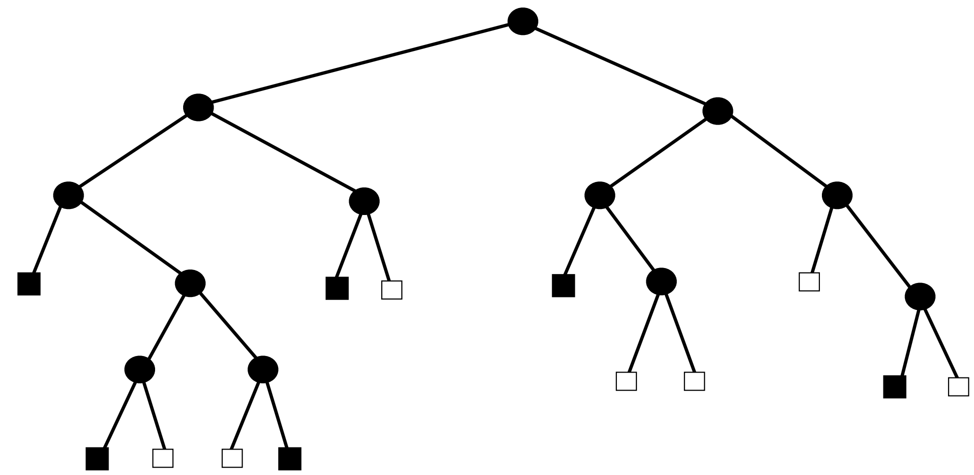
A. $\sim N/\ln 2 \doteq 1.44\ N$.

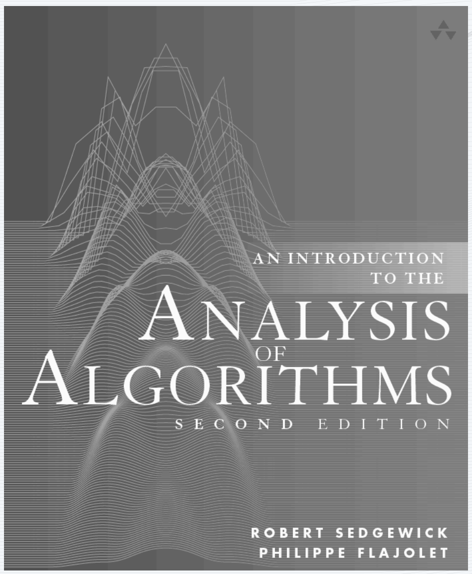
Q. Expected search cost?

A. About $N \lg N - 1.333\ N$.

Q. Rounds in leader election?

A. [see exercise 8.57].





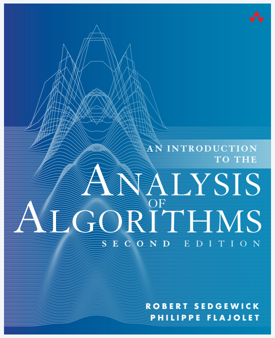
http://aofa.cs.princeton.edu

8. Strings and Tries

- Bitstrings with restrictions
- Languages
- Tries
- Trie parameters
- **Exercises**

Exercise 8.3

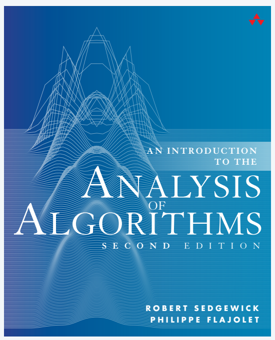
Good chance of a long run of 0s.



Exercise 8.3 How long a string of random bits should be taken to be 50% sure that there are at least 32 consecutive 0s?

Exercise 8.14

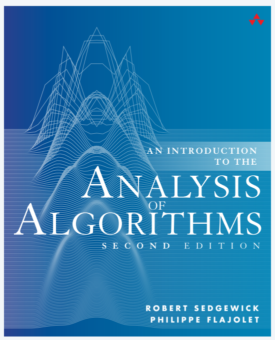
Monkey at a keyboard.



Exercise 8.14 Suppose that a monkey types randomly at a 32-key keyboard. What is the expected number of characters typed before the monkey hits upon the phrase `THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG`?

Exercise 8.57

Leader-election success probability.

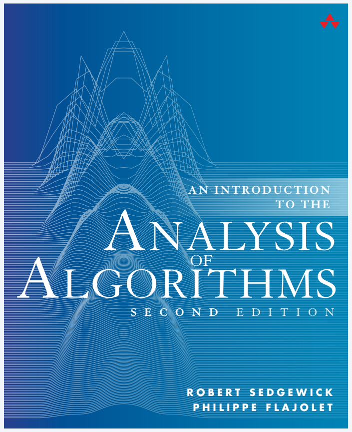


Exercise 8.57 Solve the recurrence for p_N given in the proof of Theorem 8.9, to within the oscillating term.

$$p_N = \frac{1}{2^N}\sum_k \binom{N}{k} p_k \qquad \text{for } N > 1 \text{ with } p_0 = 0 \text{ and } p_1 = 1$$

Assignments for next lecture

1. Read pages 415-472 in text.



2. Run experiments to validate mathematical results.

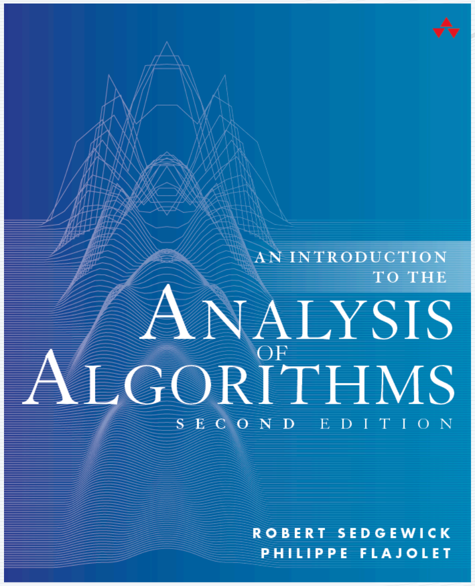
Experiment 1. Write a program to generate and draw random tries (see lecture on Trees) and use it to draw 10 random tries with 100 nodes.

Experiment 2. *Extra credit.* Validate the results of the trie path length analysis by running experiments to build 100 random tries of size *N* for *N* = 1000, 2000, 3000, ... 100,000, producing a plot like Figure 1.1 in the text. Build the tries by inserting *N* random strings into an initially empty trie.

3. Write up solutions to Exercises 8.3, 8.14, and 8.57.

ANALYTIC COMBINATORICS

PART ONE



http://aofa.cs.princeton.edu

8. Strings and Tries