

Shannon Coding for the Discrete Noiseless Channel and Related Problems

Man DU

Mordecai GOLIN

Qin ZHANG

HKUST

Barcelona
Sept 16, 2009



Overview

- *Shannon Coding* was introduced by Shannon as a proof technique in his noiseless coding theorem
- *Shannon-Fano coding* is what's primarily used for algorithm design
- This talk's punchline: Shannon Coding *can* be algorithmically useful



Outline

- Huffman Coding and Generalizations
- Previous Work & Background
- New Work
- A “Counterexample”
- Open Problems

Prefix-free coding

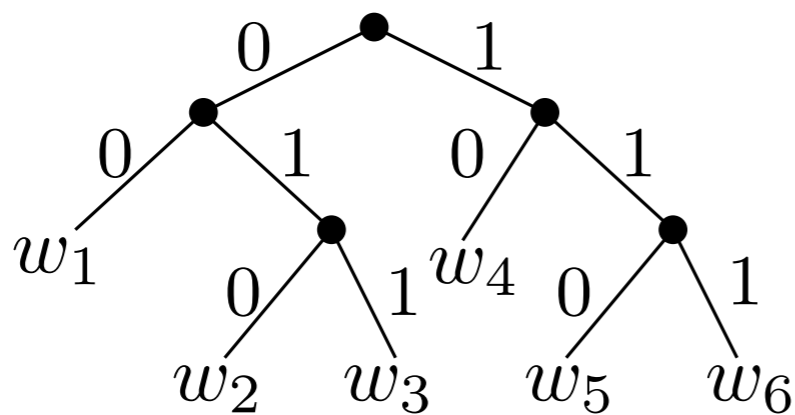
- Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_r\}$ be an *encoding alphabet*. Word $w \in \Sigma^*$ is a *prefix* of word $w' \in \Sigma^*$ if $w' = wu$ where $u \in \Sigma^*$ is a non-empty word. A *Code* over Σ is a collection of words $C = \{w_1, \dots, w_n\}$.

Prefix-free coding

- Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_r\}$ be an *encoding alphabet*. Word $w \in \Sigma^*$ is a *prefix* of word $w' \in \Sigma^*$ if $w' = wu$ where $u \in \Sigma^*$ is a non-empty word. A *Code* over Σ is a collection of words $C = \{w_1, \dots, w_n\}$.
- Code C is *prefix-free* if for all $i \neq j$ w_i is not a prefix of w_j .
 $\{0, 10, 11\}$ is prefix-free. $\{0, 00, 11\}$ isn't.

Prefix-free coding

- Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_r\}$ be an *encoding alphabet*. Word $w \in \Sigma^*$ is a *prefix* of word $w' \in \Sigma^*$ if $w' = wu$ where $u \in \Sigma^*$ is a non-empty word. A *Code* over Σ is a collection of words $C = \{w_1, \dots, w_n\}$.
- Code C is *prefix-free* if for all $i \neq j$ w_i is not a prefix of w_j .
 $\{0, 10, 11\}$ is prefix-free. $\{0, 00, 11\}$ isn't.
- A prefix-free code can be modelled as (leaves of) a tree



$$\begin{array}{ll} w_1 = 00 & w_4 = 10 \\ w_2 = 010 & w_5 = 110 \\ w_3 = 011 & w_6 = 111 \end{array}$$

The prefix coding problem

- Let $cost(w)$ be the *length* or number of characters in w . Let $P = \{p_1, p_2, \dots, p_n\}$ be a fixed discrete probability distribution (P.D.).

Define $cost(C) = \sum_{i=1}^n cost(w_i)p_i$

The prefix coding problem

- Let $cost(w)$ be the *length* or number of characters in w . Let $P = \{p_1, p_2, \dots, p_n\}$ be a fixed discrete probability distribution (P.D.).

$$\text{Define } cost(C) = \sum_{i=1}^n cost(w_i)p_i$$

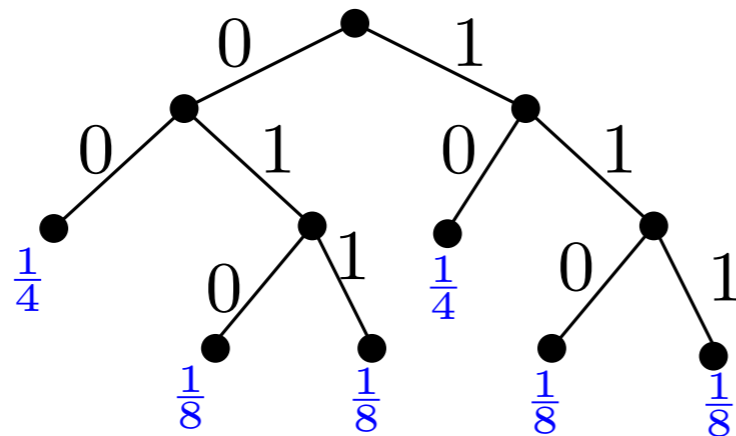
- The *prefix coding* problem, sometimes known as the *Huffman encoding* problem is to find a prefix-free code over Σ of minimum cost.

The prefix coding problem

- Let $cost(w)$ be the *length* or number of characters in w . Let $P = \{p_1, p_2, \dots, p_n\}$ be a fixed discrete probability distribution (P.D.).

$$\text{Define } cost(C) = \sum_{i=1}^n cost(w_i)p_i$$

- The *prefix coding* problem, sometimes known as the *Huffman encoding* problem is to find a prefix-free code over Σ of minimum cost.



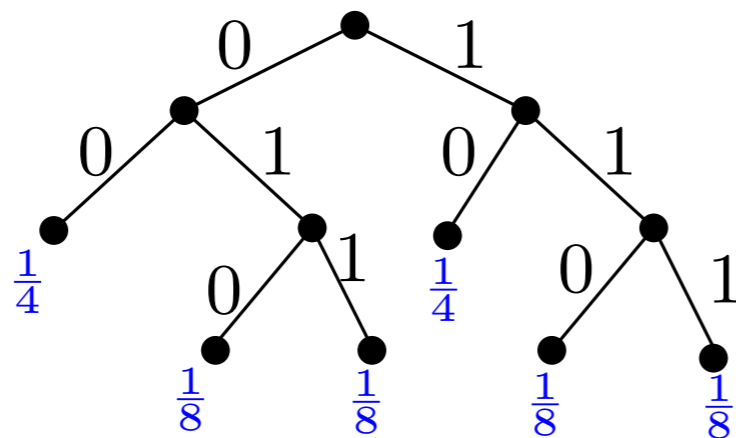
Equivalent to finding tree with *minimum external path-length*

The prefix coding problem

- Let $cost(w)$ be the *length* or number of characters in w . Let $P = \{p_1, p_2, \dots, p_n\}$ be a fixed discrete probability distribution (P.D.).

$$\text{Define } cost(C) = \sum_{i=1}^n cost(w_i)p_i$$

- The *prefix coding* problem, sometimes known as the *Huffman encoding* problem is to find a prefix-free code over Σ of minimum cost.



Equivalent to finding tree with *minimum external path-length*

$$2 \times \left[\frac{1}{4} + \frac{1}{4} \right] + 3 \times \left[\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} \right]$$



The prefix coding problem

- Useful for Data transmission/storage.
- Modelling search problems
- Very well studied

What's known

- Sub-optimal codes

Shannon coding: (from noiseless coding theorem)

There exists a prefix-free code with word lengths

$$l_i = \lceil -\log_r p_i \rceil, \quad i = 1, 2, \dots, n.$$

Shannon-Fano coding: **probability splitting**

Try to put $\sim \frac{1}{r}$ of the probability in each node.

What's known

- Sub-optimal codes

Shannon coding: (from noiseless coding theorem)

There exists a prefix-free code with word lengths

$$l_i = \lceil -\log_r p_i \rceil, \quad i = 1, 2, \dots, n.$$

Shannon-Fano coding: **probability splitting**

Try to put $\sim \frac{1}{r}$ of the probability in each node.

Both methods have cost within 1 of optimal

What's known

- Sub-optimal codes

Shannon coding: (from noiseless coding theorem)

There exists a prefix-free code with word lengths

$$l_i = \lceil -\log_r p_i \rceil, \quad i = 1, 2, \dots, n.$$

Shannon-Fano coding: **probability splitting**

Try to put $\sim \frac{1}{r}$ of the probability in each node.

Both methods have cost within 1 of optimal

- Optimal codes

Huffman 1952: a well-known $O(rn \log n)$ -time greedy-algorithm ($O(rn)$ -time if the p_i are sorted in non-decreasing order)



What's not as well known

- The fact that the greedy Huffman algorithm “works” is quite amazing
- Almost any possible modification or generalization to the original problem causes greedy to fail
- For some simple modifications, we don't even have polynomial time algorithms.

Generalizations: Min cost prefix coding

- Unequal-cost coding

Allow letters to have different costs, say, $c(\sigma_j) = c_j$.

- Discrete Noiseless Channels (in Shannon's original paper)

This can be viewed as a strongly connected aperiodic **directed graph** with k vertices (states).

1. Each edge leaving a vertex is labelled by an encoding letter $\sigma \in \Sigma$, with at most one σ -edge leaving each vertex.
2. An edge labelled by σ leaving vertex i has cost $c_{i,\sigma}$.

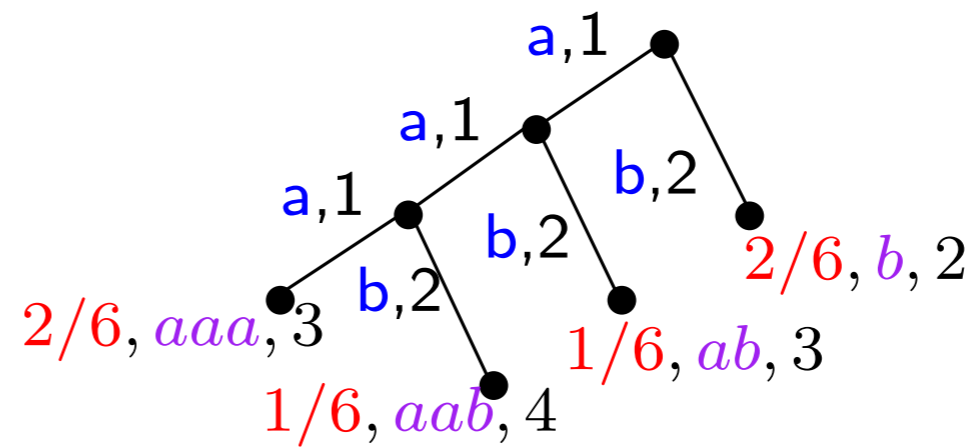
- Language restrictions

Require all codewords to be contained in some given Language \mathcal{L}

Generalizations: Prefix-free coding

- With Unequal-cost letters

$$c_1 = 1; c_2 = 2.$$

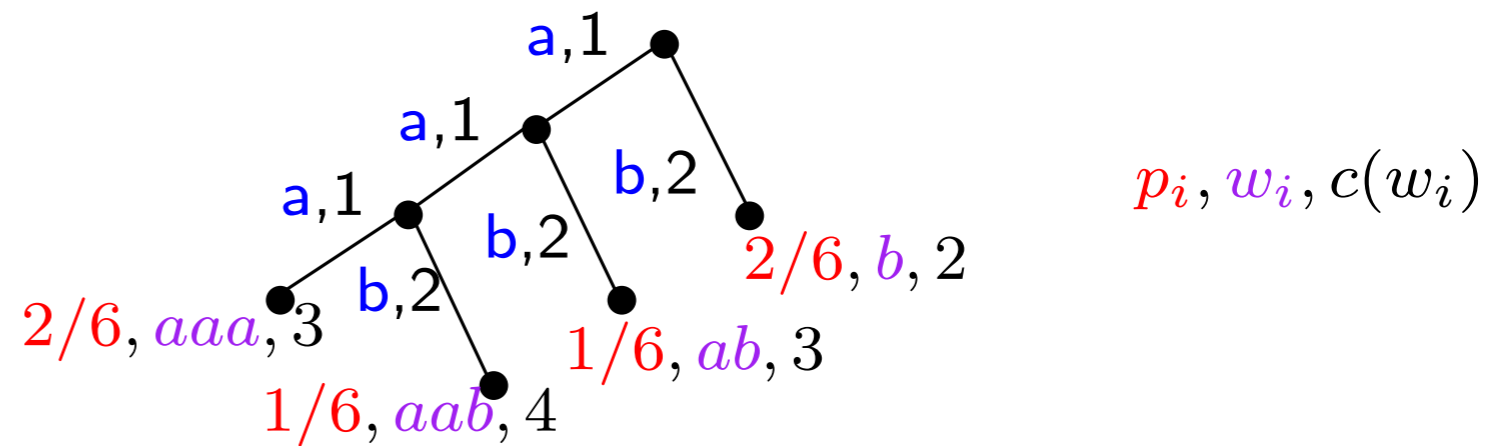


$$p_i, w_i, c(w_i)$$

Generalizations: Prefix-free coding

- With Unequal-cost letters

$$c_1 = 1; c_2 = 2.$$

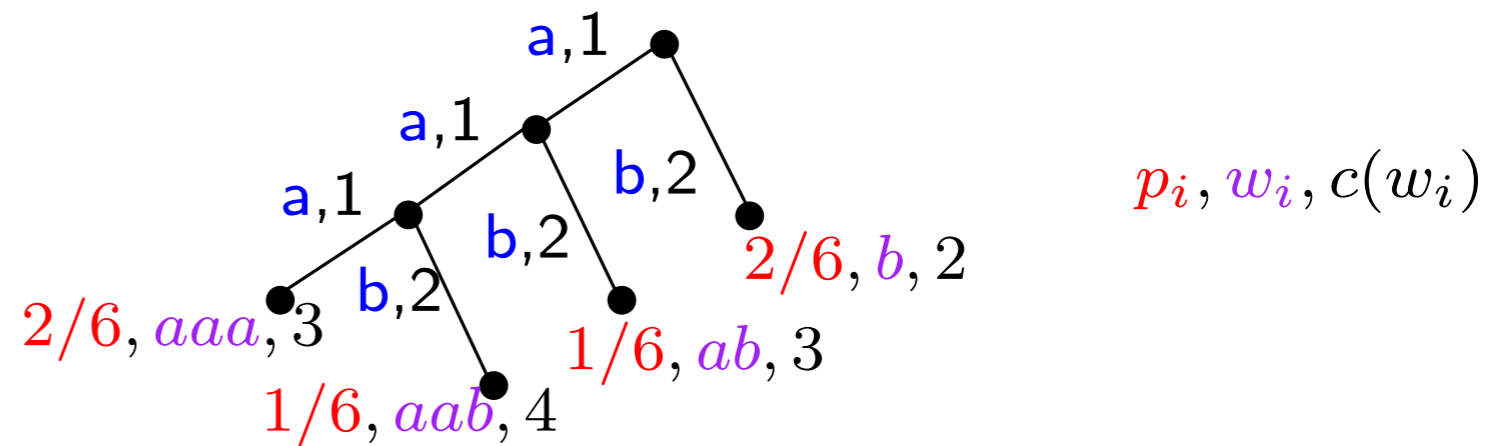


- Corresponds to different letter transmission/storage costs, e.g., the **Telegraph Channel**.
Also, to different costs for evaluating test outcomes in, e.g., group testing.

Generalizations: Prefix-free coding

- With Unequal-cost letters

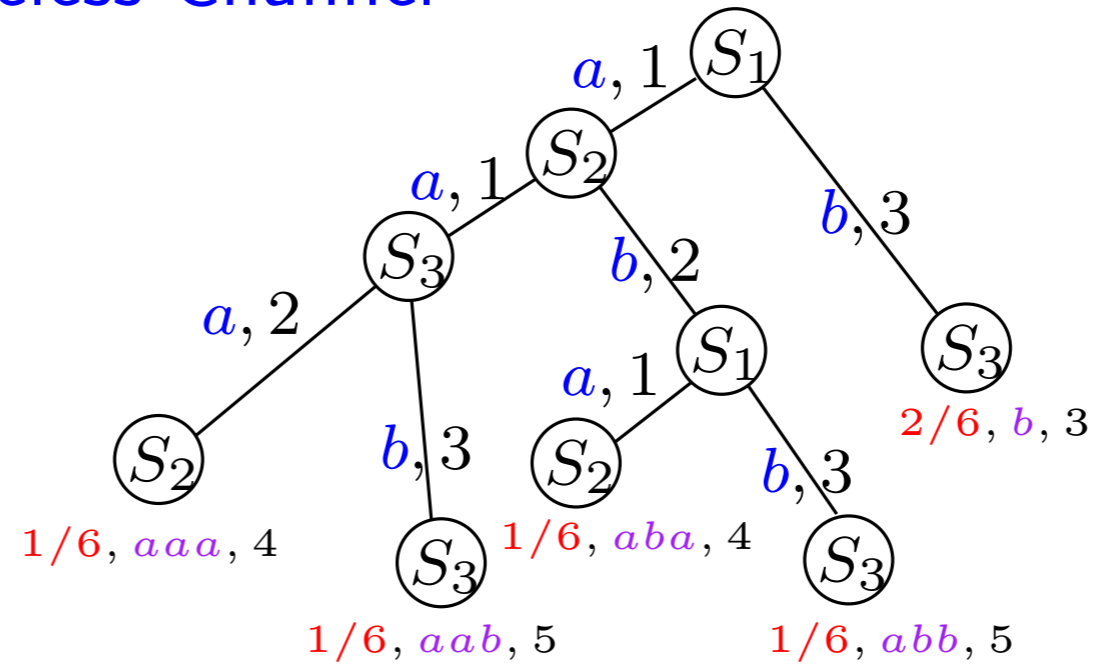
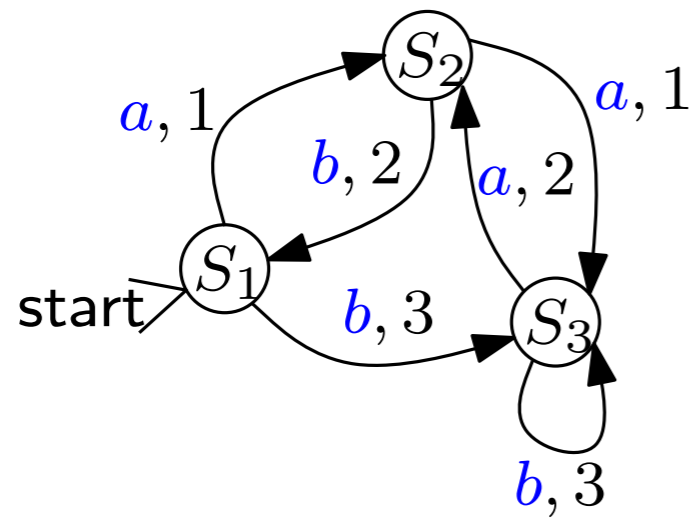
$$c_1 = 1; c_2 = 2.$$



- Corresponds to different letter transmission/storage costs, e.g., the **Telegraph Channel**.
Also, to different costs for evaluating test outcomes in, e.g., group testing.
- Size of encoding alphabet, Σ , *could* be countably infinite!

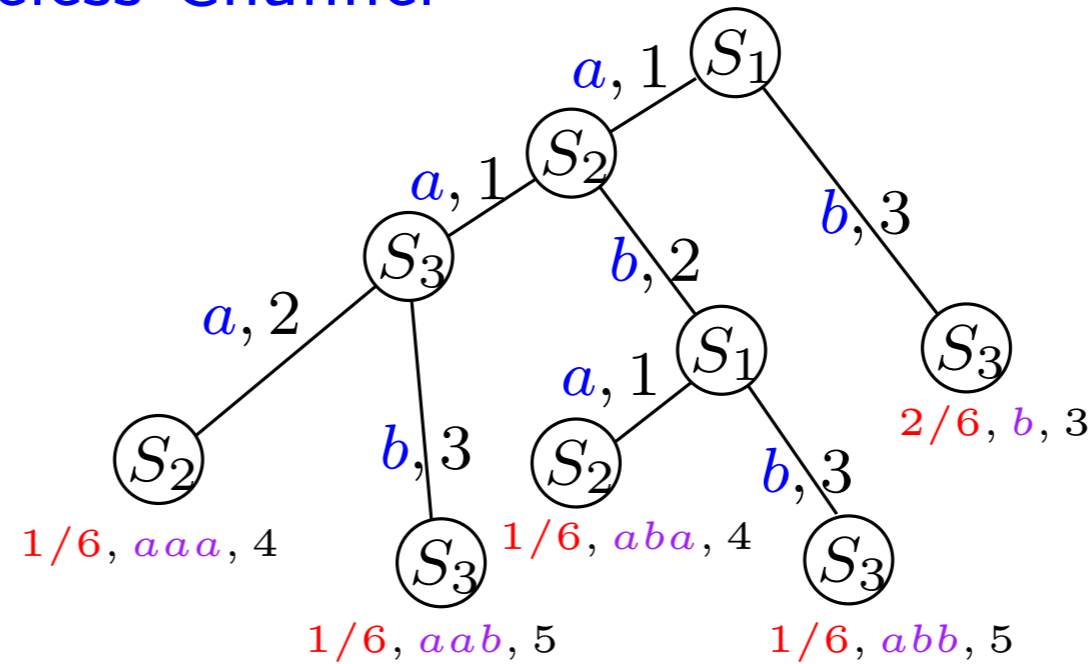
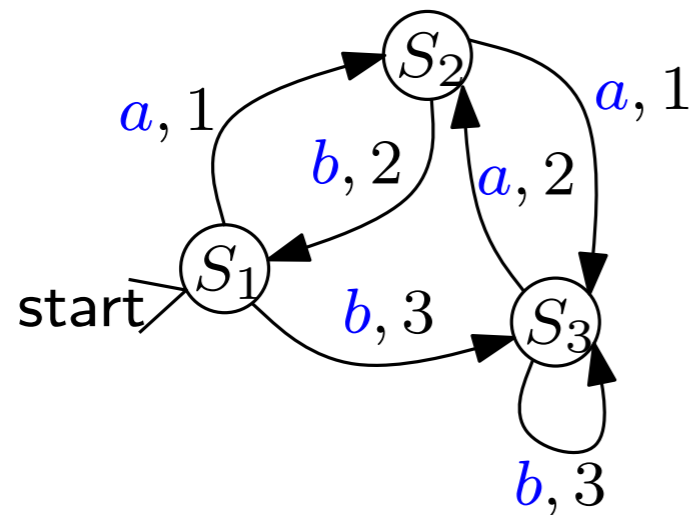
Generalizations: Prefix-free coding

- In a Discrete Noiseless Channel



Generalizations: Prefix-free coding

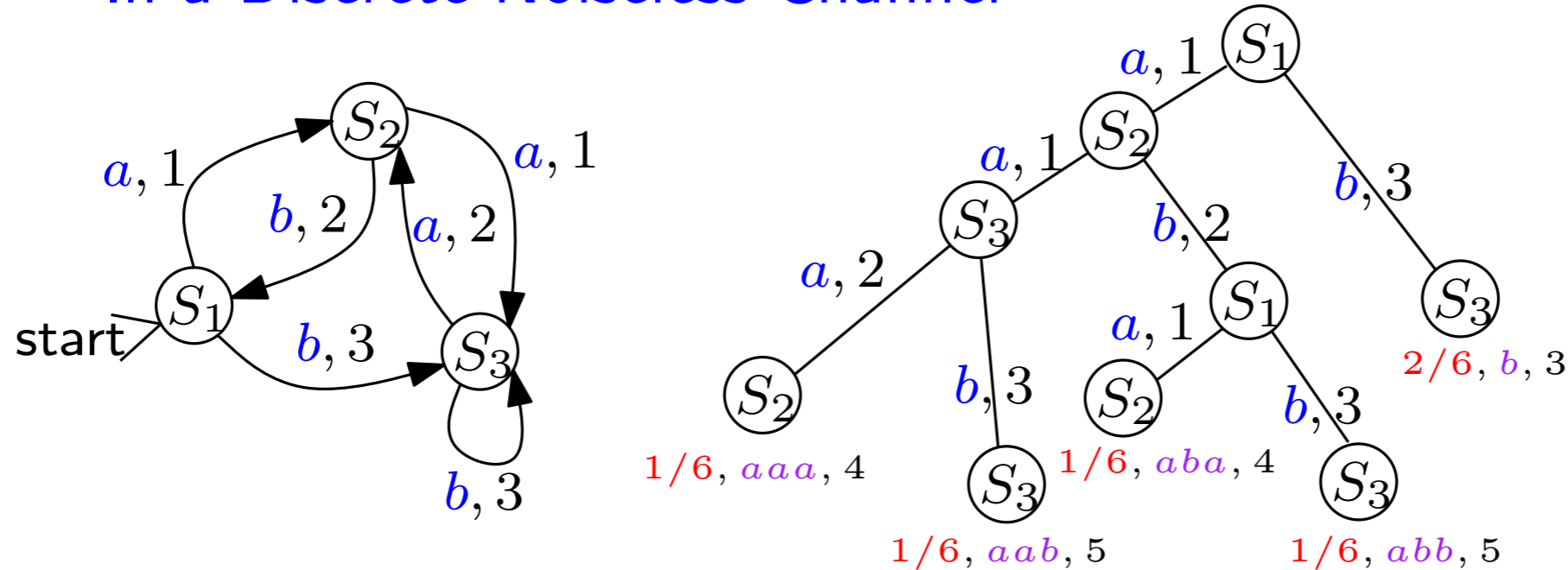
- In a Discrete Noiseless Channel



- Cost of letter depends upon current state.
In Shannon's original paper, $k = \#$ states and $|\Sigma|$ are both finite

Generalizations: Prefix-free coding

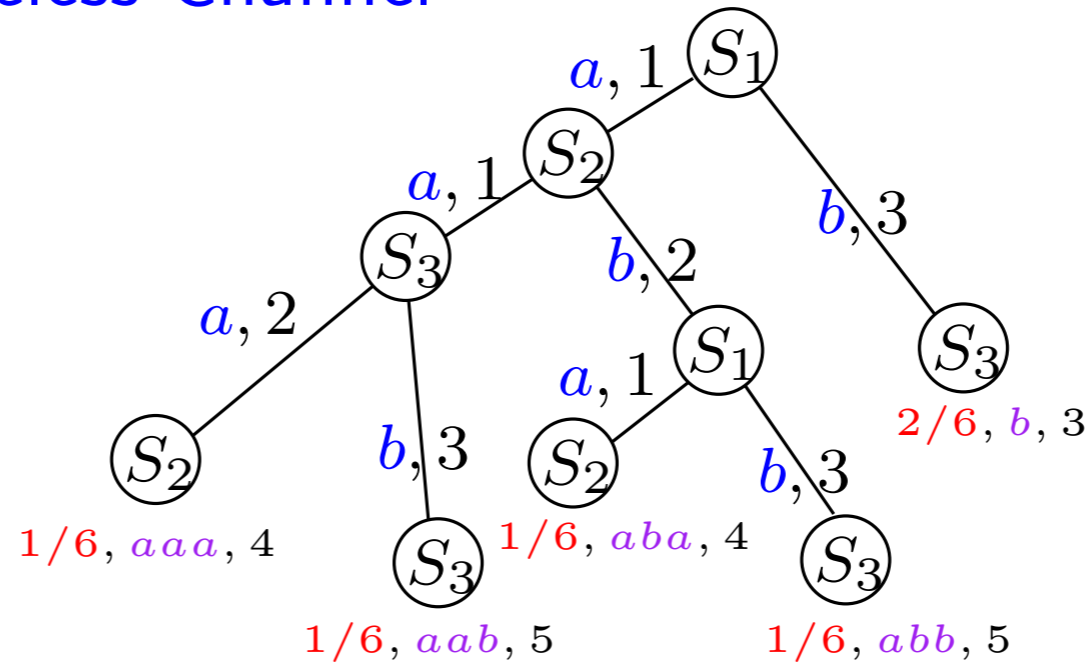
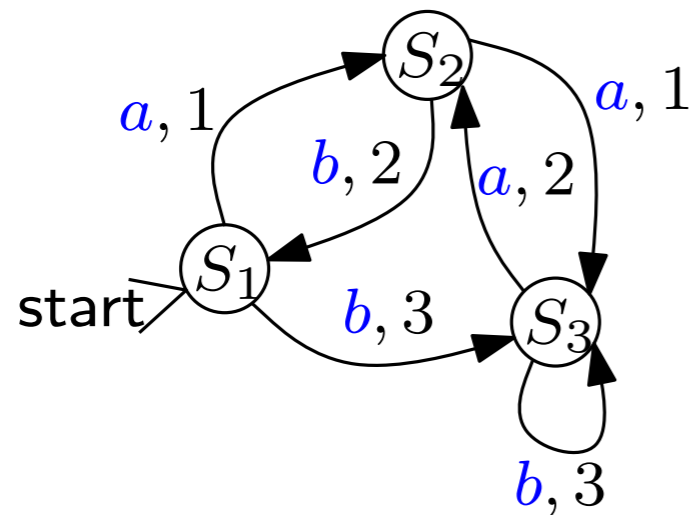
- In a Discrete Noiseless Channel



- Cost of letter depends upon current state.
In Shannon's original paper, $k = \# \text{ states}$ and $|\Sigma|$ are both finite
- A codeword has both start and end states. In coded message, new codeword must start from final state of preceding one.

Generalizations: Prefix-free coding

- In a Discrete Noiseless Channel



- Cost of letter depends upon current state.
In Shannon's original paper, $k = \# \text{ states}$ and $|\Sigma|$ are both finite
- A codeword has both start and end states. In coded message, new codeword must start from final state of preceding one.
- \Rightarrow Need k code trees; each one rooted with different state



Generalizations: Prefix-free coding

- With Language Restrictions



Generalizations: Prefix-free coding

- With Language Restrictions
- Find min-cost prefix code in which all words belong to given language \mathcal{L} .

Generalizations: Prefix-free coding

- With Language Restrictions
- Find min-cost prefix code in which all words belong to given language \mathcal{L} .
- Example: $\mathcal{L} = 0^*1$, all binary words ending in '1'.
Used in constructing self-synchronizing codes.

Generalizations: Prefix-free coding

- With Language Restrictions
- Find min-cost prefix code in which all words belong to given language \mathcal{L} .
- Example: $\mathcal{L} = 0^*1$, all binary words ending in '1'.
Used in constructing self-synchronizing codes.
- One of the problems that motivated this research.
Let \mathcal{L} be the set of all binary words that do *not* contain a given pattern, e.g., 010.
No previous good way of finding min cost prefix code with such restrictions.



Generalizations: Prefix-free coding

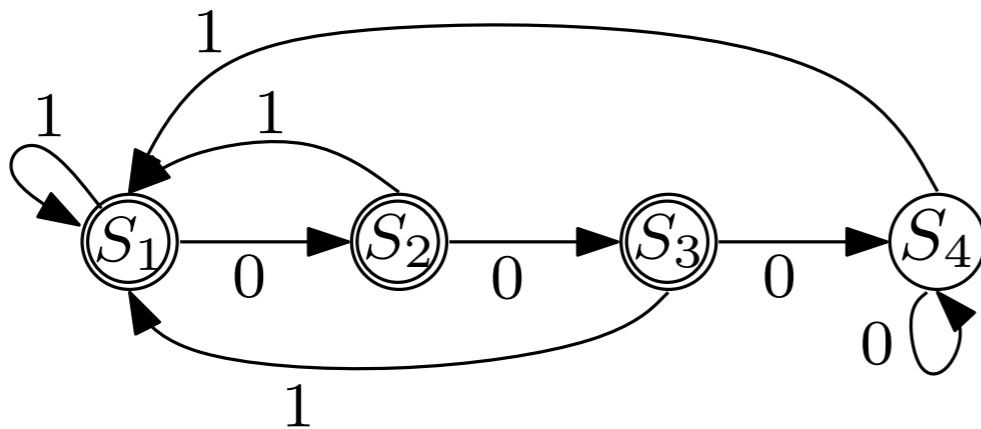
- With **Regular** Language Restrictions

Generalizations: Prefix-free coding

- With **Regular** Language Restrictions
- In this case, there is a DFA \mathcal{M} accepting Language \mathcal{L} .

Generalizations: Prefix-free coding

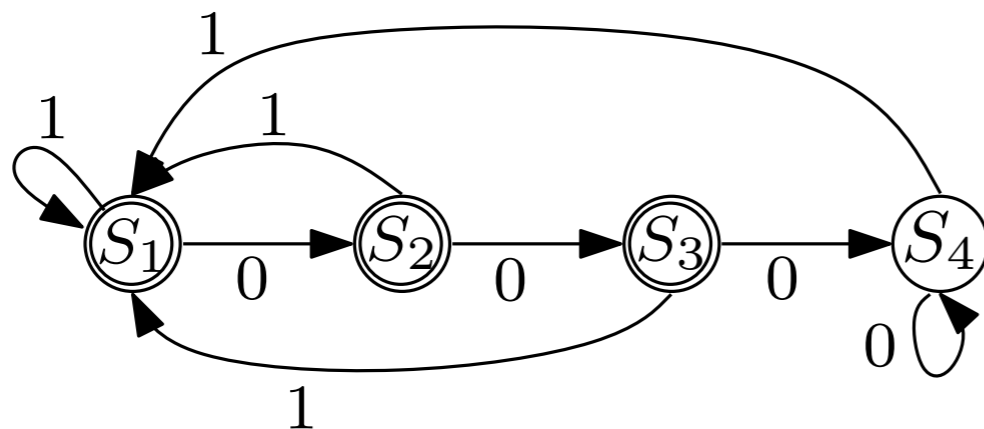
- With **Regular** Language Restrictions
- In this case, there is a DFA \mathcal{M} accepting Language \mathcal{L} .



$$\begin{aligned}\mathcal{L} &= ((0 + 1)^* 000)^C \\ &= \text{binary strings not ending in } 000\end{aligned}$$

Generalizations: Prefix-free coding

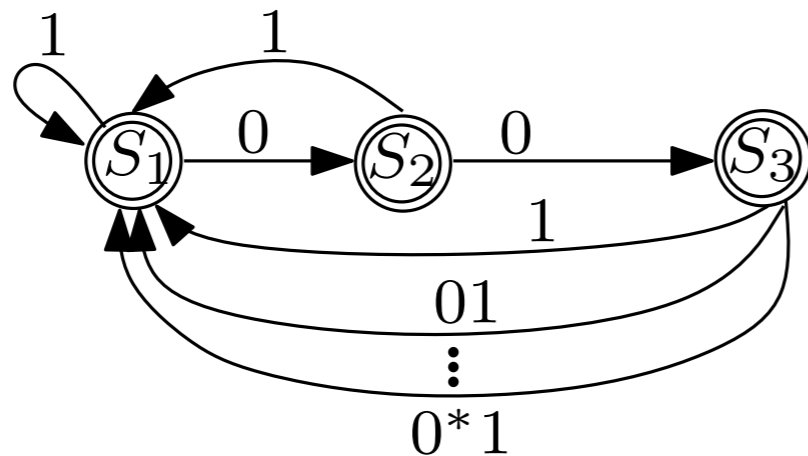
- With **Regular** Language Restrictions
- In this case, there is a DFA \mathcal{M} accepting Language \mathcal{L} .



$$\mathcal{L} = ((0 + 1)^* 000)^C$$

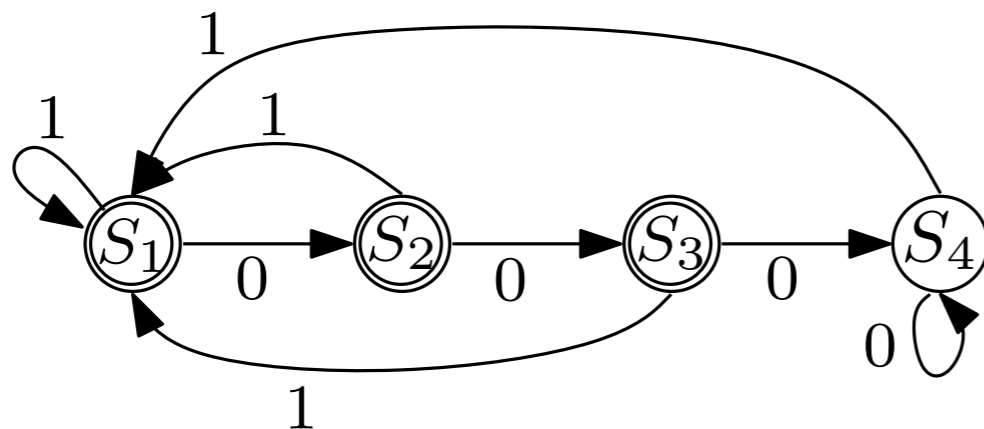
= binary strings not ending in 000

- Erasing the nonaccepting states, \mathcal{M} can be drawn with a finite # of states but a countably **infinite** encoding alphabet.



Generalizations: Prefix-free coding

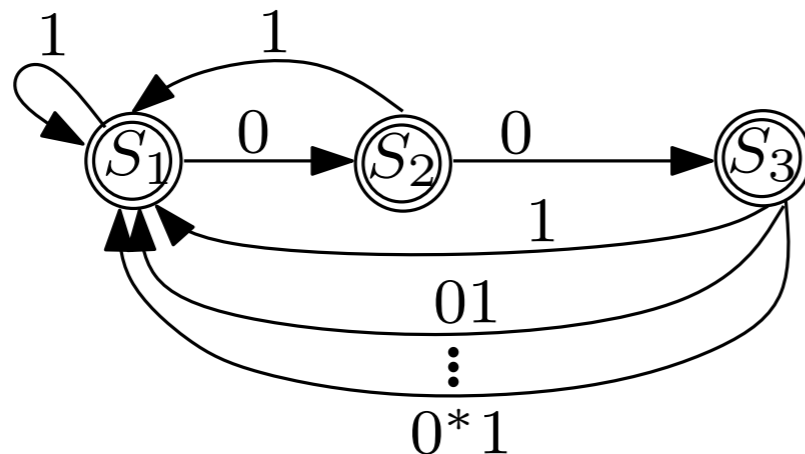
- With **Regular** Language Restrictions
- In this case, there is a DFA \mathcal{M} accepting Language \mathcal{L} .



$$\mathcal{L} = ((0 + 1)^* 000)^C$$

= binary strings not ending in 000

- Erasing the nonaccepting states, \mathcal{M} can be drawn with a finite # of states but a countably **infinite** encoding alphabet.



Note: graph doesn't need to be strongly connected. It might even have sinks!

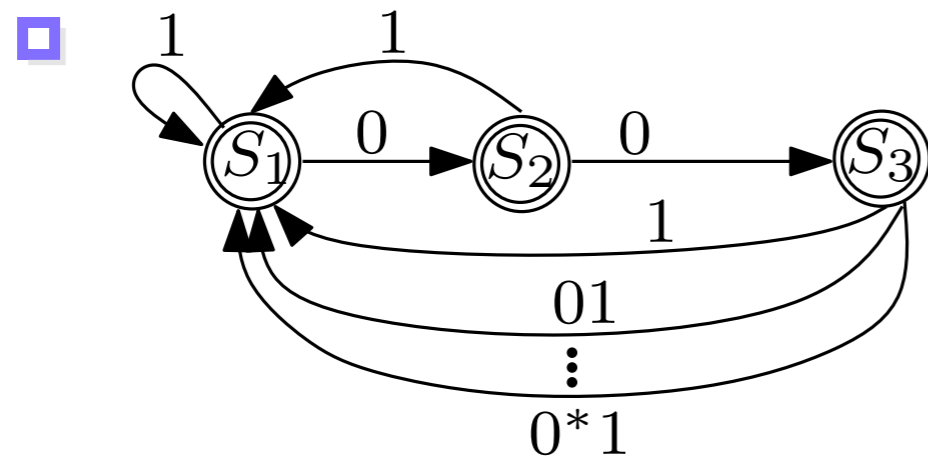


Generalizations: Prefix-free coding

- With **Regular** Language Restrictions

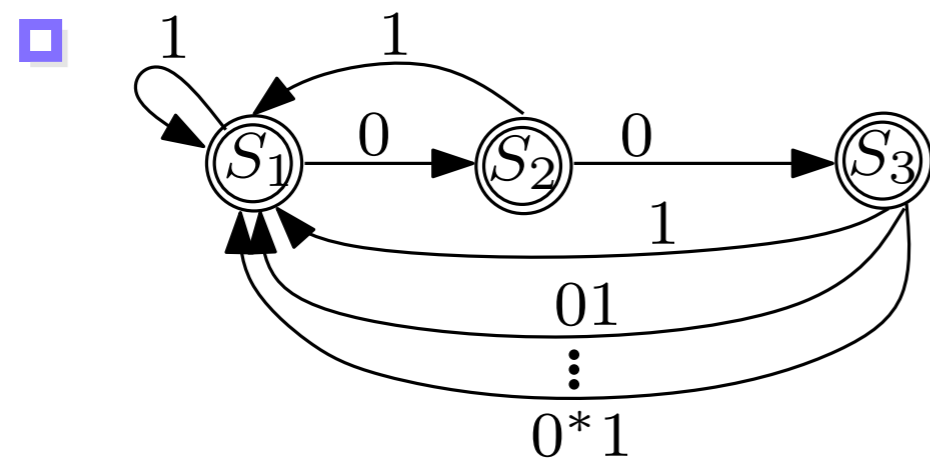
Generalizations: Prefix-free coding

- With **Regular** Language Restrictions

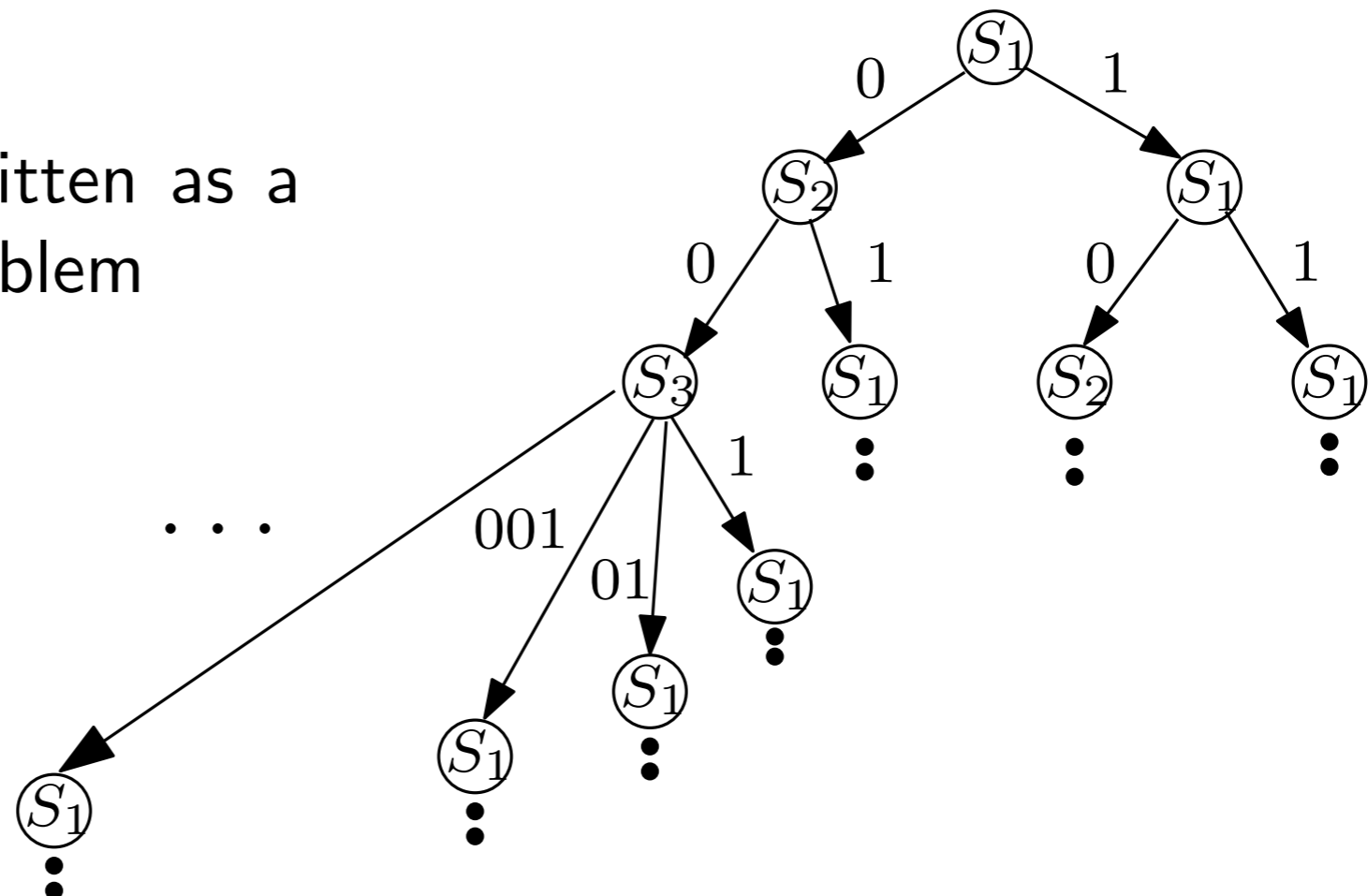


Generalizations: Prefix-free coding

- With **Regular** Language Restrictions



- Can still be rewritten as a min-cost tree problem





Outline

- Huffman Coding and Generalizations
- Previous Work & Background
- New Work
- A “Counterexample”
- Open Problems



Previous Work: Unequal Cost Coding

- Letters in Σ have **different** costs $c_1 \leq c_2 \leq c_3 \leq \dots \leq c_r$.
Models different transmission/storage costs

Previous Work: Unequal Cost Coding

- Letters in Σ have **different** costs $c_1 \leq c_2 \leq c_3 \leq \dots \leq c_r$.
Models different transmission/storage costs
- Blachman (1954), Marcus (1957), Gilbert (1995) – Heuristics
Karp (1961) – Integer Linear Programming Solution
G., Rote (1998) – $O(n^{c_r+2})$ DP solution
Bradford, et. al. (2002), Dumitrescu(2006) – $O(n^{c_r})$
G., Kenyon, Young (2002) – A PTAS

Previous Work: Unequal Cost Coding

- Letters in Σ have **different** costs $c_1 \leq c_2 \leq c_3 \leq \dots \leq c_r$.
Models different transmission/storage costs
- Blachman (1954), Marcus (1957), Gilbert (1995) – Heuristics
Karp (1961) – Integer Linear Programming Solution
G., Rote (1998) – $O(n^{c_r+2})$ DP solution
Bradford, et. al. (2002), Dumitrescu(2006) – $O(n^{c_r})$
G., Kenyon, Young (2002) – A PTAS
- Big Open Question
Still don't know if it's NP-Hard, in P or something between.

Previous Work: Unequal Cost Coding

- ▣ Letters in Σ have **different** costs $c_1 \leq c_2 \leq c_3 \leq \dots \leq c_r$.
Models different transmission/storage costs
- ▣ Blachman (1954), Marcus (1957), Gilbert (1995) – Heuristics
Karp (1961) – Integer Linear Programming Solution
G., Rote (1998) – $O(n^{c_r+2})$ DP solution
Bradford, et. al. (2002), Dumitrescu(2006) – $O(n^{c_r})$
G., Kenyon, Young (2002) – A PTAS
- ▣ Big Open Question
Still **don't know** if it's NP-Hard, in P or something between.
- ▣ Most Practical Solutions are arithmetic error approximations

A diagram consisting of a horizontal orange bar at the top and a vertical blue bar on the left. Inside the vertical blue bar, there is a smaller vertical cyan bar. The bars are outlined in black.

Previous Work: Unequal Cost Coding



Previous Work: Unequal Cost Coding

- Efficient algorithms ($O(n \log n)$ or $O(n)$) that create codes which are within an additive error of optimal.

$$COST \leq OPT + K$$

Previous Work: Unequal Cost Coding

- Efficient algorithms ($O(n \log n)$ or $O(n)$) that create codes which are within an additive error of optimal.

$$COST \leq OPT + K$$

- - Krause (1962)
 - Csiszar (1969)
 - Cott (1977)
 - Alenkamp and Mehlhorn (1980)
 - Mehlhorn (1980)
 - G. and Li (2007)

Previous Work: Unequal Cost Coding

- Efficient algorithms ($O(n \log n)$ or $O(n)$) that create codes which are within an additive error of optimal.

$$COST \leq OPT + K$$

- - Krause (1962)
 - Csiszar (1969)
 - Cott (1977)
 - Altenkamp and Mehlhorn (1980)
 - Mehlhorn (1980)
 - G. and Li (2007)
- K is a function of letter costs c_1, c_2, c_3, \dots
 $K(c_1, c_2, c_3, \dots)$ are incomparable between different algorithms
 K is often function of longest letter length c_r , problem when $r = \infty$.

Previous Work: Unequal Cost Coding

- Efficient algorithms ($O(n \log n)$ or $O(n)$) that create codes which are within an additive error of optimal.

$$COST \leq OPT + K$$

- Krause (1962)
 - Csiszar (1969)
 - Cott (1977)
 - Altenkamp and Mehlhorn (1980)
 - Mehlhorn (1980)
 - G. and Li (2007)
- K is a function of letter costs c_1, c_2, c_3, \dots
 $K(c_1, c_2, c_3, \dots)$ are incomparable between different algorithms

All algorithms above are Shannon-Fano type codes; differ in how they define “approximate” split



Previous Work:

- **The Discrete Noiseless Channel:** Only previous result seems to be [Csiszar \(1969\)](#) who gives additive approximation to optimal code, again using a generalization of Shannon-Fano splitting.

Previous Work:

- **The Discrete Noiseless Channel:** Only previous result seems to be [Csiszar \(1969\)](#) who gives additive approximation to optimal code, again using a generalization of Shannon-Fano splitting.
- **Language Constraints**
 - “1”-ended codes:
[Capocelli, et.al., \(1994\)](#) [Berger, Yeung\(1990\)](#) – Exponential Search
[Chan, G. \(2000\)](#) – $O(n^3)$ DP algorithm
 - Sound of Silence – Binary Codes with at most k zeros
[Dolev, et. al. \(1999\)](#) – $n^{O(k)}$ DP algorithm
 - General Regular Language Constraint
Folk theorem: If \exists a DFA with m states accepting \mathcal{L} , optimal code can be built in $n^{O(m)}$ time. ($O(m) \leq 3m.$)

Previous Work:

- **The Discrete Noiseless Channel:** Only previous result seems to be [Csiszar \(1969\)](#) who gives additive approximation to optimal code, again using a generalization of Shannon-Fano splitting.
- **Language Constraints**
 - “1”-ended codes:
 - [Capocelli, et.al., \(1994\)](#) [Berger, Yeung\(1990\)](#) – Exponential Search
 - [Chan, G. \(2000\)](#) – $O(n^3)$ DP algorithm
 - Sound of Silence – Binary Codes with at most k zeros
 - [Dolev, et. al. \(1999\)](#) – $n^{O(k)}$ DP algorithm
 - General Regular Language Constraint
 - Folk theorem: If \exists a DFA with m states accepting \mathcal{L} , optimal code can be built in $n^{O(m)}$ time. ($O(m) \leq 3m$.)
 - **No good efficient algorithm known**



Previous Work:

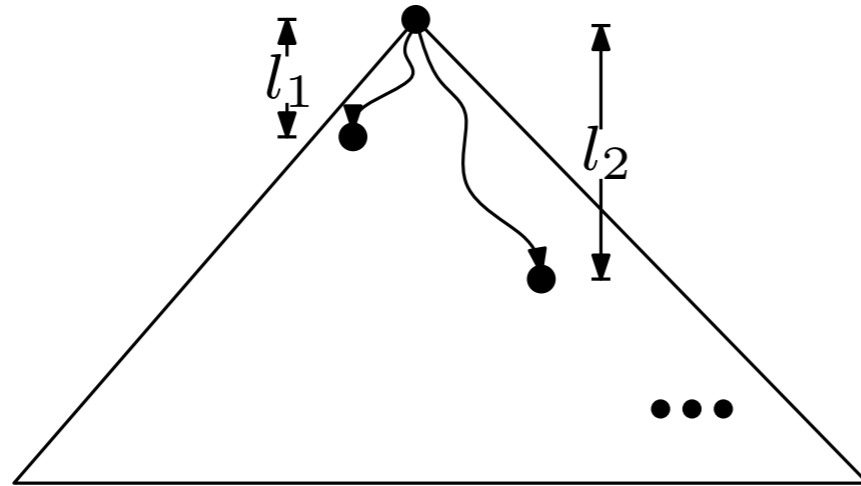
- Pre-Huffman there were two Sub-optimal constructions for basic case

Previous Work:

- Pre-Huffman there were two Sub-optimal constructions for basic case
- **Shannon coding:** (from noiseless coding theorem)
There exists a prefix-free code with word lengths $l_i = \lceil -\log_r p_i \rceil, i = 1, 2, \dots, n.$
- **Shannon-Fano coding:** **probability splitting**
Try to put $\sim \frac{1}{r}$ of the probability in each node.

Shannon Coding vs. Shannon-Fano Coding

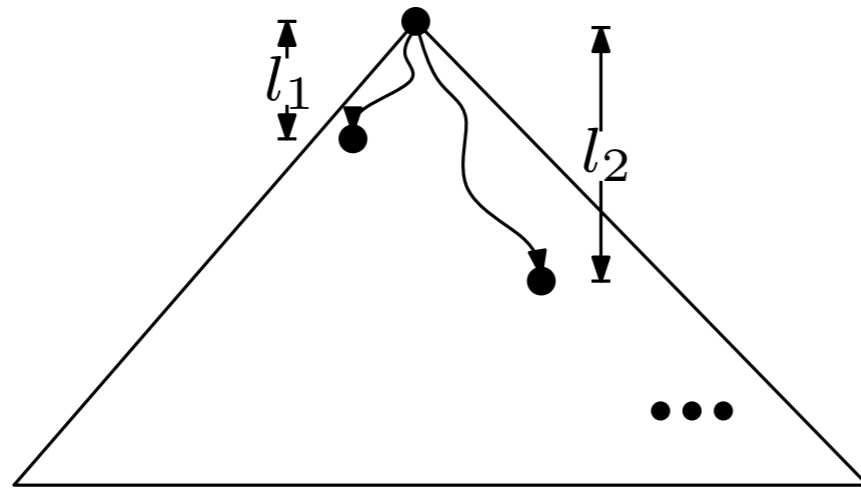
□ Shannon Coding



$$l_i = \lceil -\log_r p_i \rceil$$

Shannon Coding vs. Shannon-Fano Coding

□ Shannon Coding

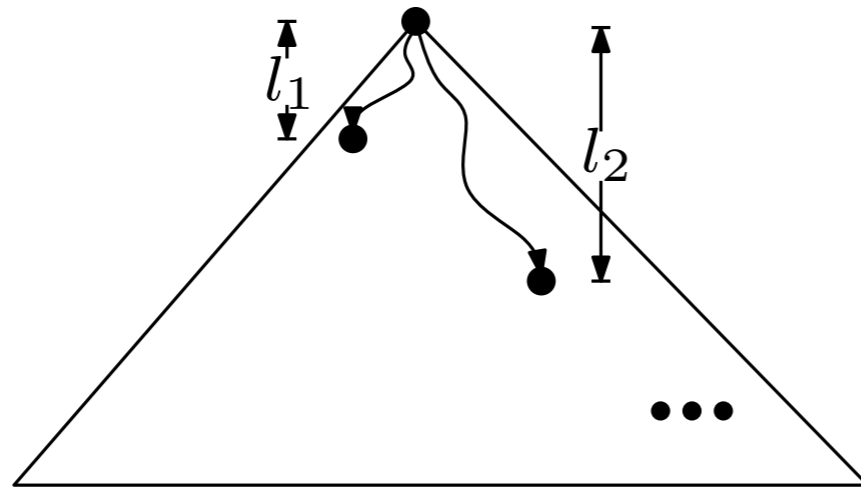


$$l_i = \lceil -\log_r p_i \rceil$$

Given depths l_i , can build tree via top-down “linear” scan. When moving down a level, expand *all* non-used leaves to be parents.

Shannon Coding vs. Shannon-Fano Coding

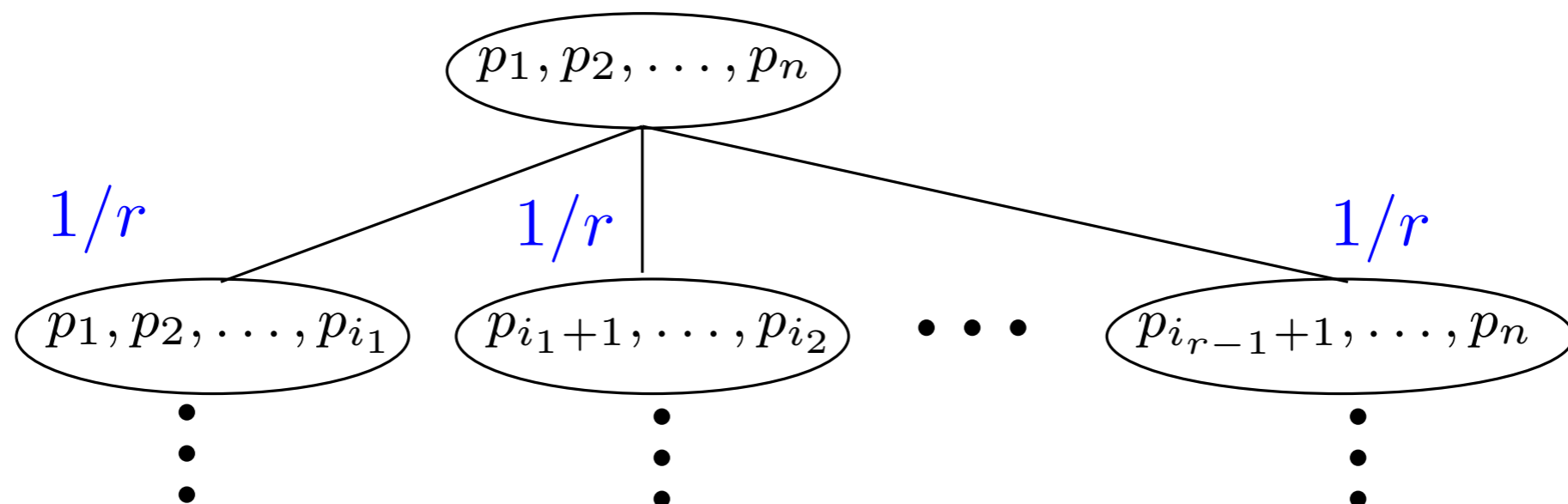
Shannon Coding



$$l_i = \lceil -\log_r p_i \rceil$$

Given depths l_i , can build tree via top-down “linear” scan. When moving down a level, expand *all* non-used leaves to be parents.

Shannon-Fano Coding



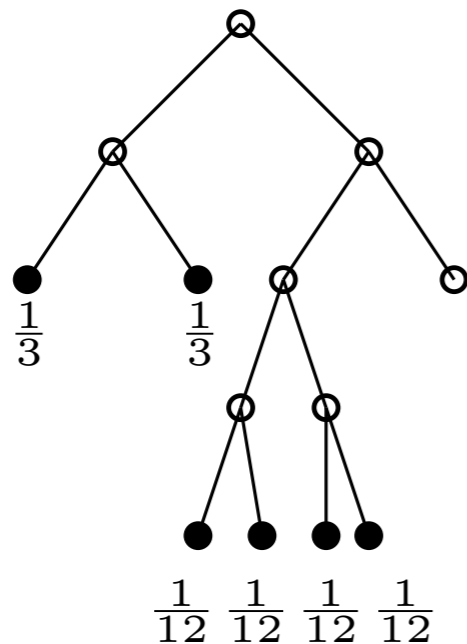


Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$

Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$



Shannon coding

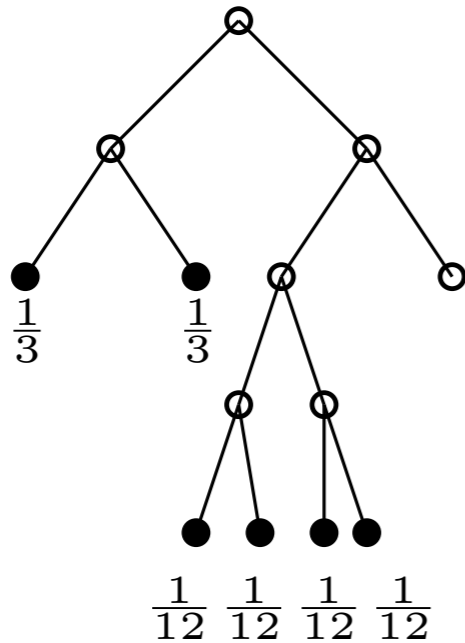
$$l_1 = l_2 = 2 = \lceil -\log_2 \frac{1}{3} \rceil$$

$$l_3 = l_4 = l_5 = l_6 = 4 = \lceil -\log_2 \frac{1}{12} \rceil$$

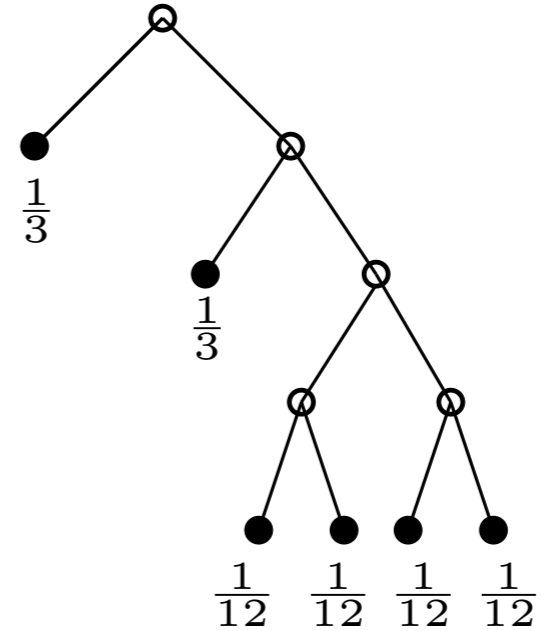
Has empty "slots"
can be improved

Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$



Shannon coding



Shannon-Fano coding

Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$

Shannon-Fano: First, sort items and insert at root.

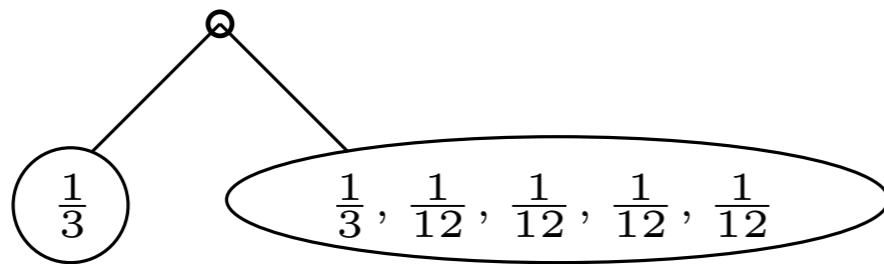
While a node contains more than 1 item, split its items' weights as evenly as possible. At most $1/2$ node's weight in left child.

Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$

Shannon-Fano: First, sort items and insert at root.

While a node contains more than 1 item, split its items' weights as evenly as possible. At most $1/2$ node's weight in left child.

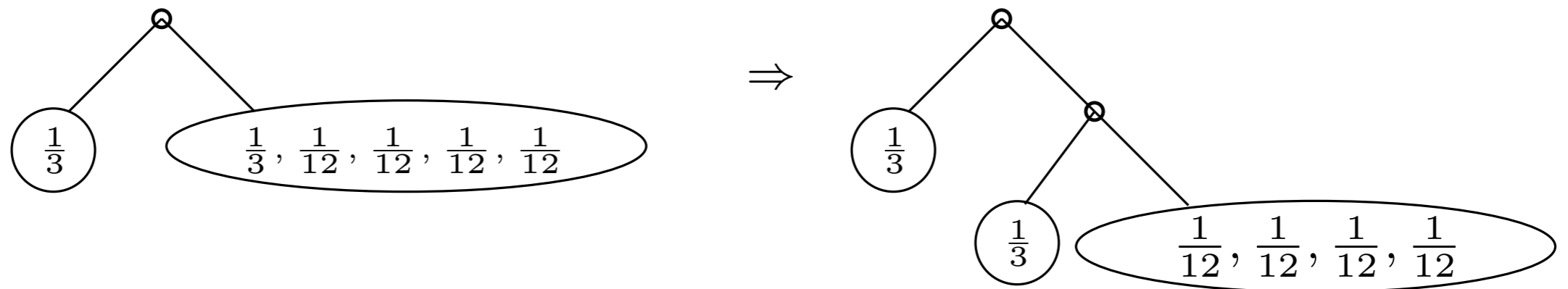


Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$

Shannon-Fano: First, sort items and insert at root.

While a node contains more than 1 item, split its items' weights as evenly as possible. At most $1/2$ node's weight in left child.

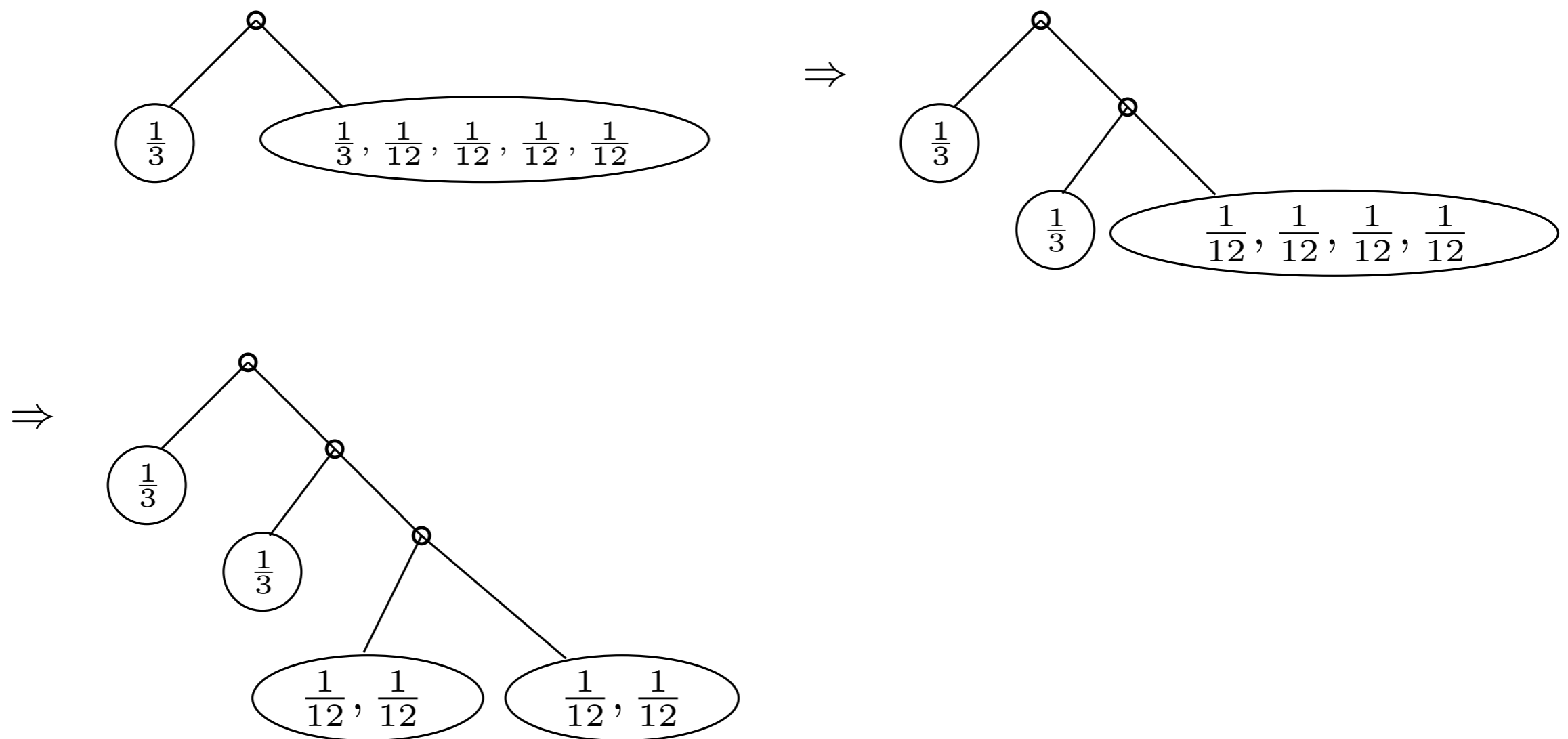


Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$

Shannon-Fano: First, sort items and insert at root.

While a node contains more than 1 item, split its items' weights as evenly as possible. At most $1/2$ node's weight in left child.

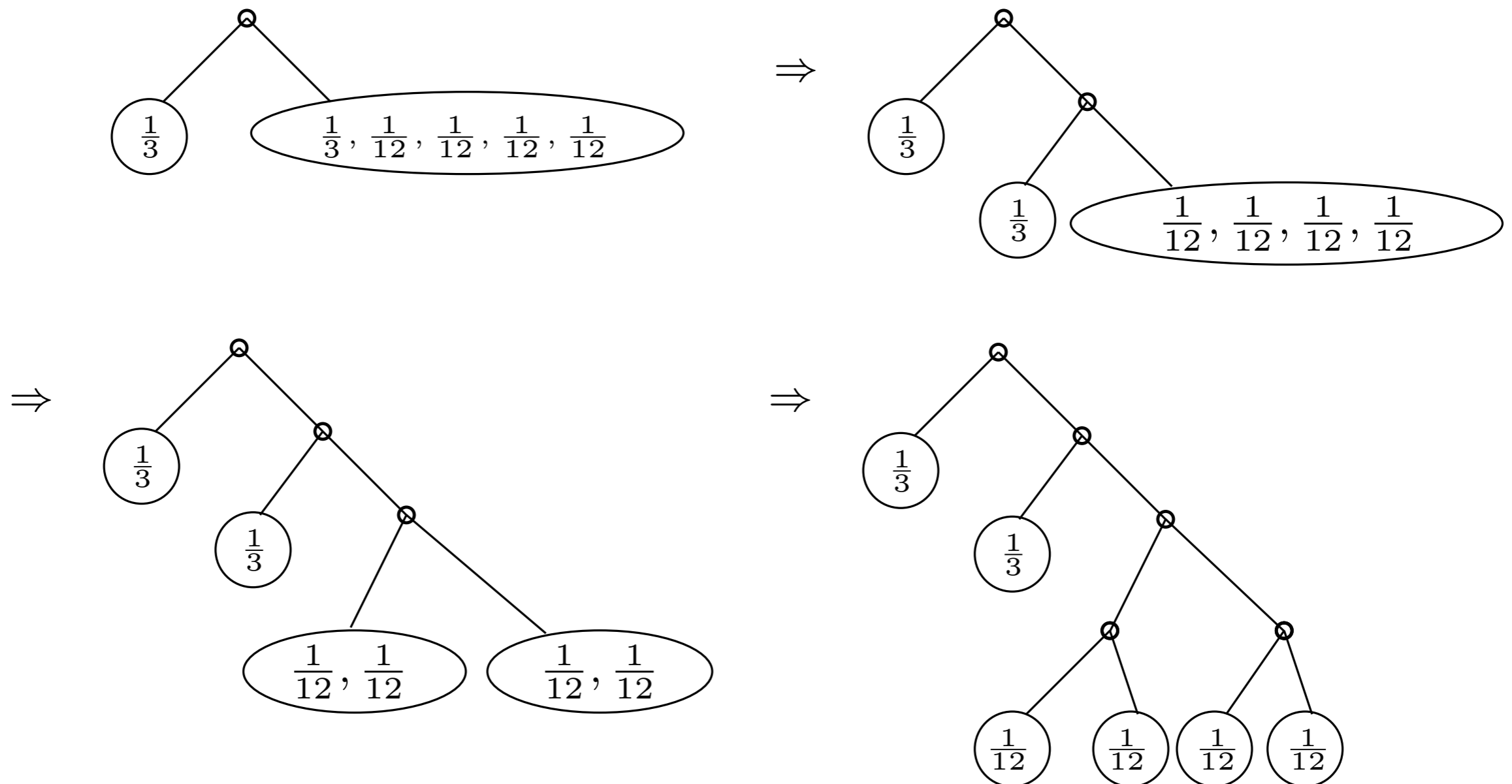


Shannon Coding vs. Shannon-Fano Coding

Example: $p_1 = p_2 = \frac{1}{3}$, $p_3 = p_4 = p_5 = p_6 = \frac{1}{12}$

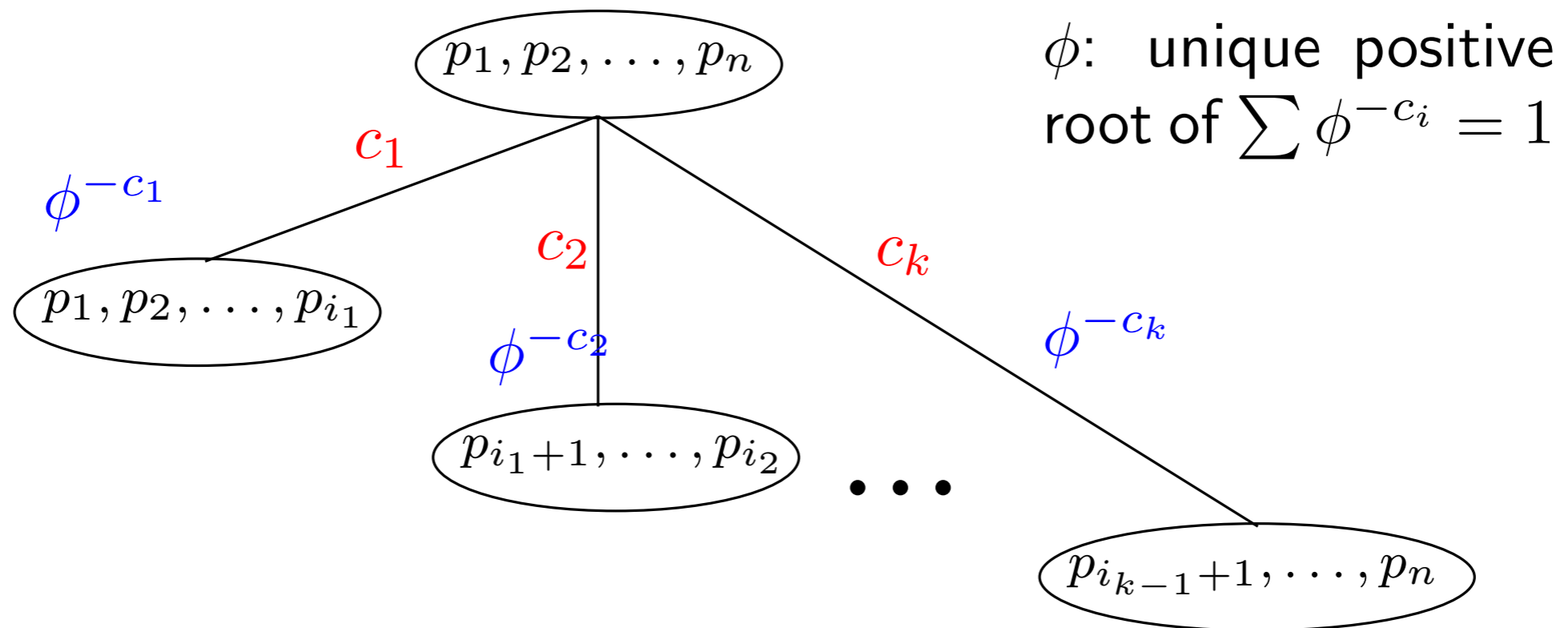
Shannon-Fano: First, sort items and insert at root.

While a node contains more than 1 item, split its items' weights as evenly as possible. At most $1/2$ node's weight in left child.



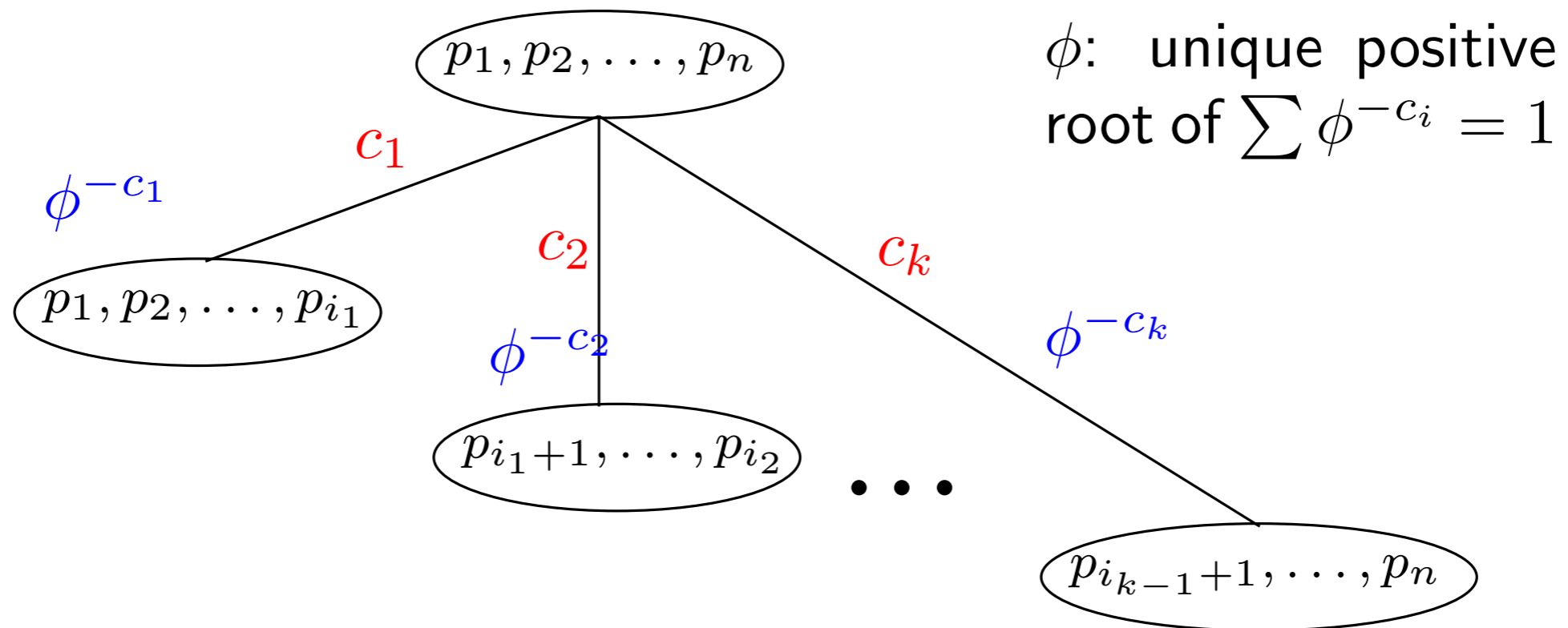
Previous Work. Unequal Cost Codes

- Shannon Fano coding for unequal cost codes



Previous Work. Unequal Cost Codes

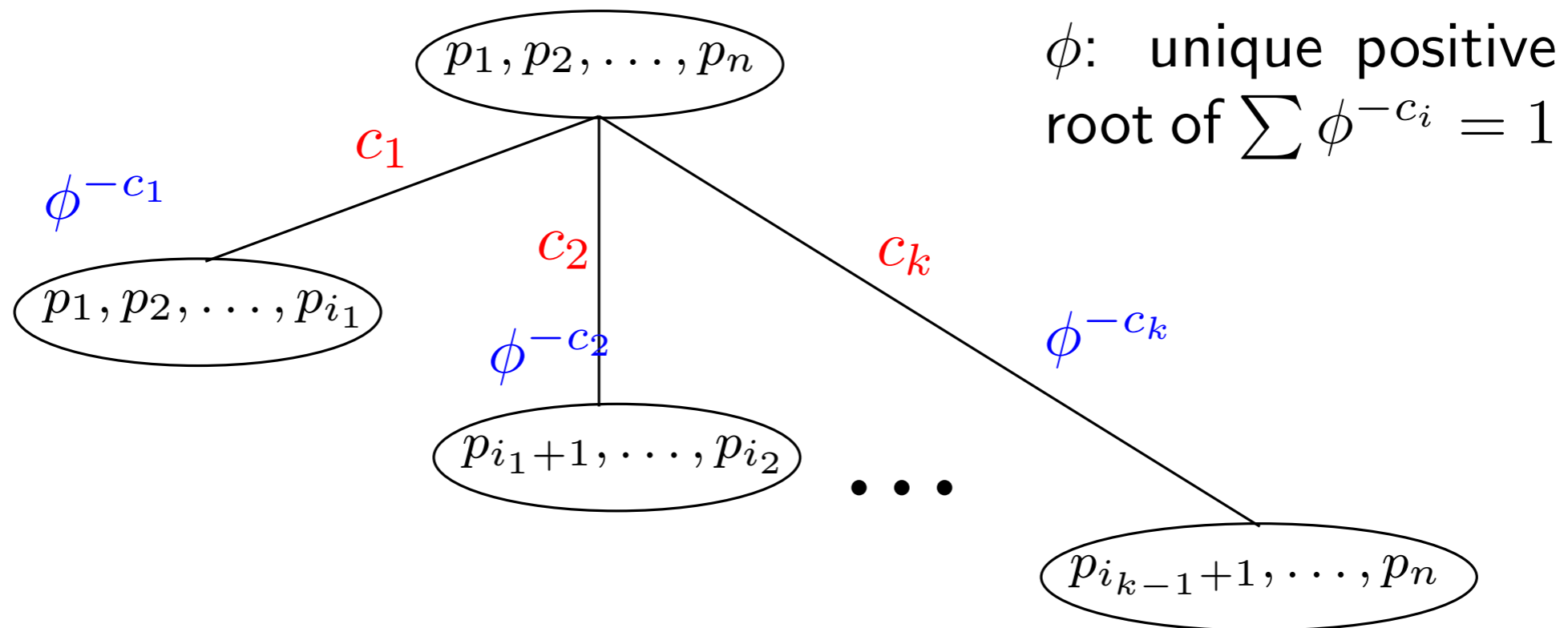
- Shannon Fano coding for unequal cost codes



- Split probabilities so “approximately” ϕ^{-c_i} of the probability in a node is put into its i^{th} child.

Previous Work. Unequal Cost Codes

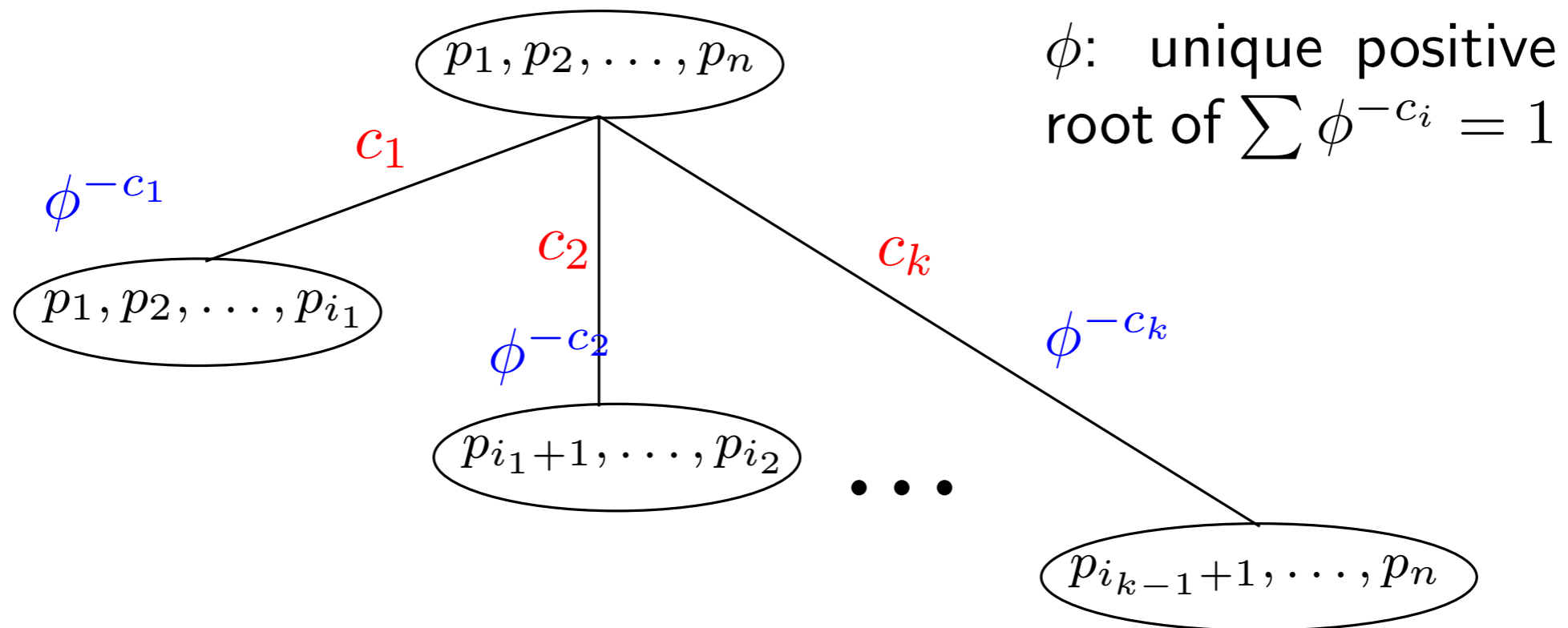
- Shannon Fano coding for unequal cost codes



- Split probabilities so “approximately” ϕ^{-c_i} of the probability in a node is put into its i^{th} child.
- Note: This “can” work for infinite alphabets, as long as ϕ exists.

Previous Work. Unequal Cost Codes

- Shannon Fano coding for unequal cost codes



- Split probabilities so “approximately” ϕ^{-c_i} of the probability in a node is put into its i^{th} child.
- All previous algorithms were Shannon-Fano like. They differed in how they implemented “approximate split”.

- 
- Shannon-Fano coding for unequal cost codes

ϕ : unique positive root of $\sum \phi^{-c_i} = 1$

- Split probabilities so “approximately” ϕ^{-c_i} of the probability in a node is put into its i^{th} child.

- Shannon-Fano coding for unequal cost codes

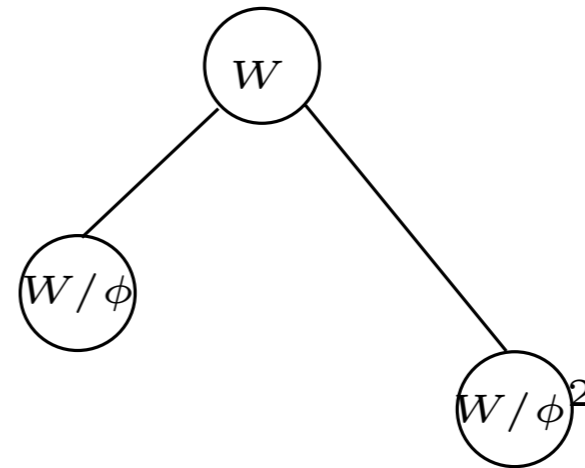
ϕ : unique positive root of $\sum \phi^{-c_i} = 1$

- Split probabilities so “approximately” ϕ^{-c_i} of the probability in a node is put into its i^{th} child.

- Example: Telegraph Channel: $c_1 = 1, c_2 = 2$

$$\phi^{-1} = \frac{\sqrt{5}-1}{2}$$

Put $\sim \phi^{-1}$ of the root's weight in the left subtree and $\sim \phi^{-2}$ of the weight in the right



- 
- Shannon-Fano coding for unequal cost codes

ϕ : unique positive root of $\sum \phi^{-c_i} = 1$

- Split probabilities so “approximately” ϕ^{-c_i} of the probability in a node is put into its i^{th} child.

- Shannon-Fano coding for unequal cost codes

ϕ : unique positive root of $\sum \phi^{-c_i} = 1$

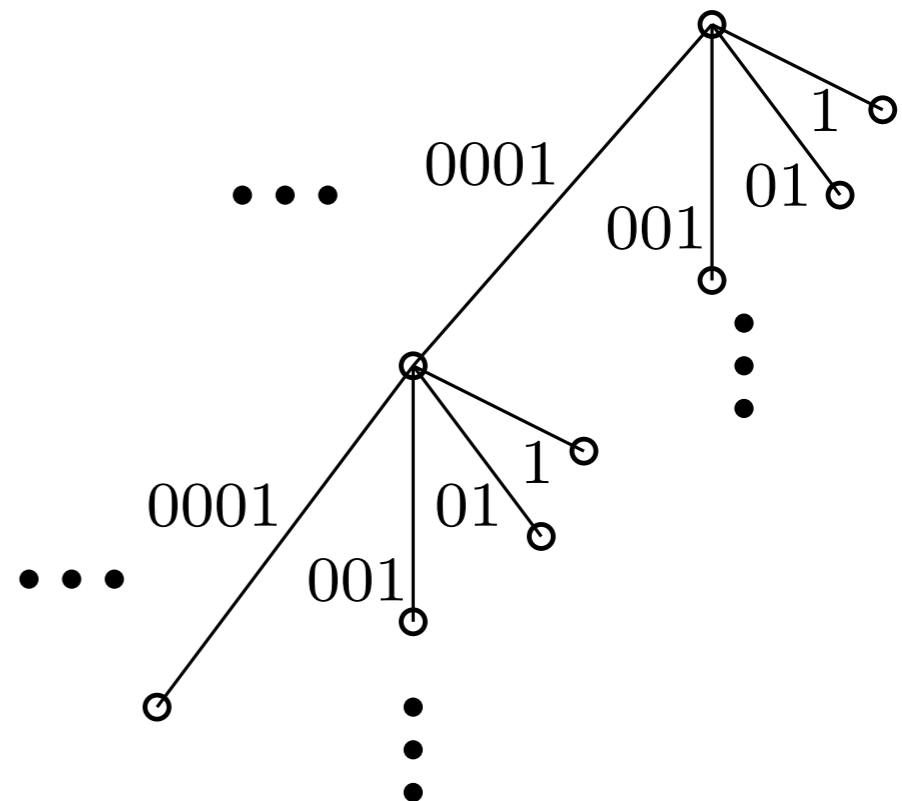
- Split probabilities so “approximately” ϕ^{-c_i} of the probability in a node is put into its i^{th} child.

- Example: 1-ended coding. $\forall i > 0, c_i = i$.

i^{th} encoding letter denotes string $0^{i-1}1$.

$\sum \phi^{-c_i} = 1$ gives $\phi^{-1} = \frac{1}{2}$

Put $\sim 2^{-i}$ of a node's weight into its i^{th} subtree



Previous Work. Well Known Lower Bound

- Given coding letter lengths $\mathcal{C} = \{c_1, c_2, c_3, \dots\}$, $\gcd(c_i) = 1$, let ϕ be the unique positive root of $g(z) = 1 - \sum_j \phi^{-c_j}$

Previous Work. Well Known Lower Bound

- Given coding letter lengths $\mathcal{C} = \{c_1, c_2, c_3, \dots\}$, $\gcd(c_i) = 1$, let ϕ be the unique positive root of $g(z) = 1 - \sum_j \phi^{-c_j}$

Note: ϕ sometimes called the “capacity”

Previous Work. Well Known Lower Bound

- Given coding letter lengths $\mathcal{C} = \{c_1, c_2, c_3, \dots\}$, $\gcd(c_i) = 1$, let ϕ be the unique positive root of $g(z) = 1 - \sum_j \phi^{-c_j}$

Note: ϕ sometimes called the “capacity”

- For given P.D. set $H_\phi = -\sum p_i \log_\phi p_i$.

Previous Work. Well Known Lower Bound

- Given coding letter lengths $\mathcal{C} = \{c_1, c_2, c_3, \dots\}$, $\gcd(c_i) = 1$, let ϕ be the unique positive root of $g(z) = 1 - \sum_j \phi^{-c_j}$

Note: ϕ sometimes called the “capacity”

- For given P.D. set $H_\phi = -\sum p_i \log_\phi p_i$.

Note: If $c_1 = c_2 = 1$ then $\phi = 2$ and H_ϕ is standard entropy

Previous Work. Well Known Lower Bound

- Given coding letter lengths $\mathcal{C} = \{c_1, c_2, c_3, \dots\}$, $\gcd(c_i) = 1$, let ϕ be the unique positive root of $g(z) = 1 - \sum_j \phi^{-c_j}$

Note: ϕ sometimes called the “capacity”

- For given P.D. set $H_\phi = -\sum p_i \log_\phi p_i$.

Note: If $c_1 = c_2 = 1$ then $\phi = 2$ and H_ϕ is standard entropy

- Theorem:

Let OPT be cost of min-cost code for given P.D. and letter costs. Then

$$H_\phi \leq OPT$$

Previous Work. Well Known Lower Bound

- Given coding letter lengths $\mathcal{C} = \{c_1, c_2, c_3, \dots\}$, $\gcd(c_i) = 1$, let ϕ be the unique positive root of $g(z) = 1 - \sum_j \phi^{-c_j}$

Note: ϕ sometimes called the “capacity”

- For given P.D. set $H_\phi = -\sum p_i \log_\phi p_i$.

Note: If $c_1 = c_2 = 1$ then $\phi = 2$ and H_ϕ is standard entropy

- Theorem:

Let OPT be cost of min-cost code for given P.D. and letter costs. Then

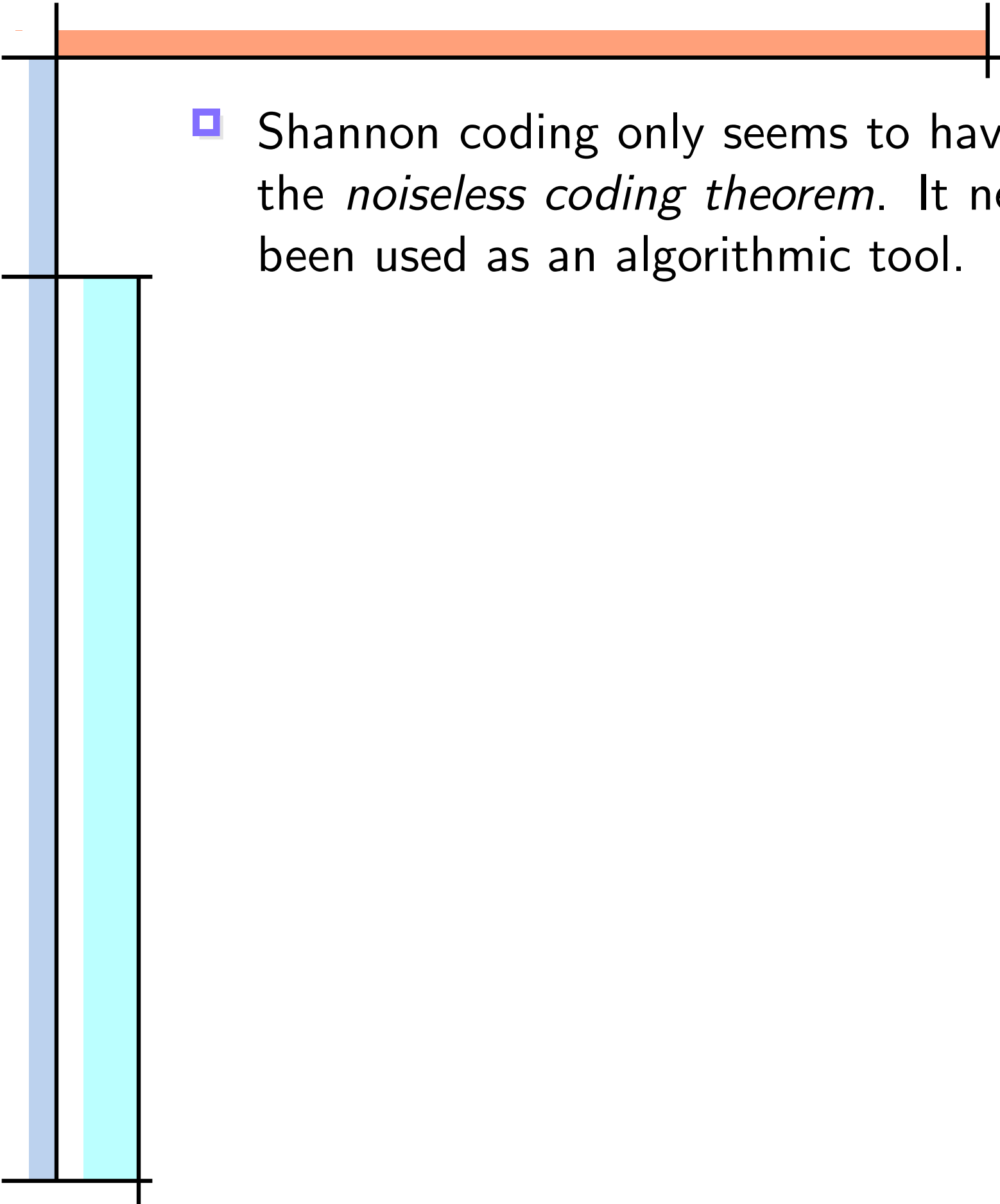
$$H_\phi \leq OPT$$

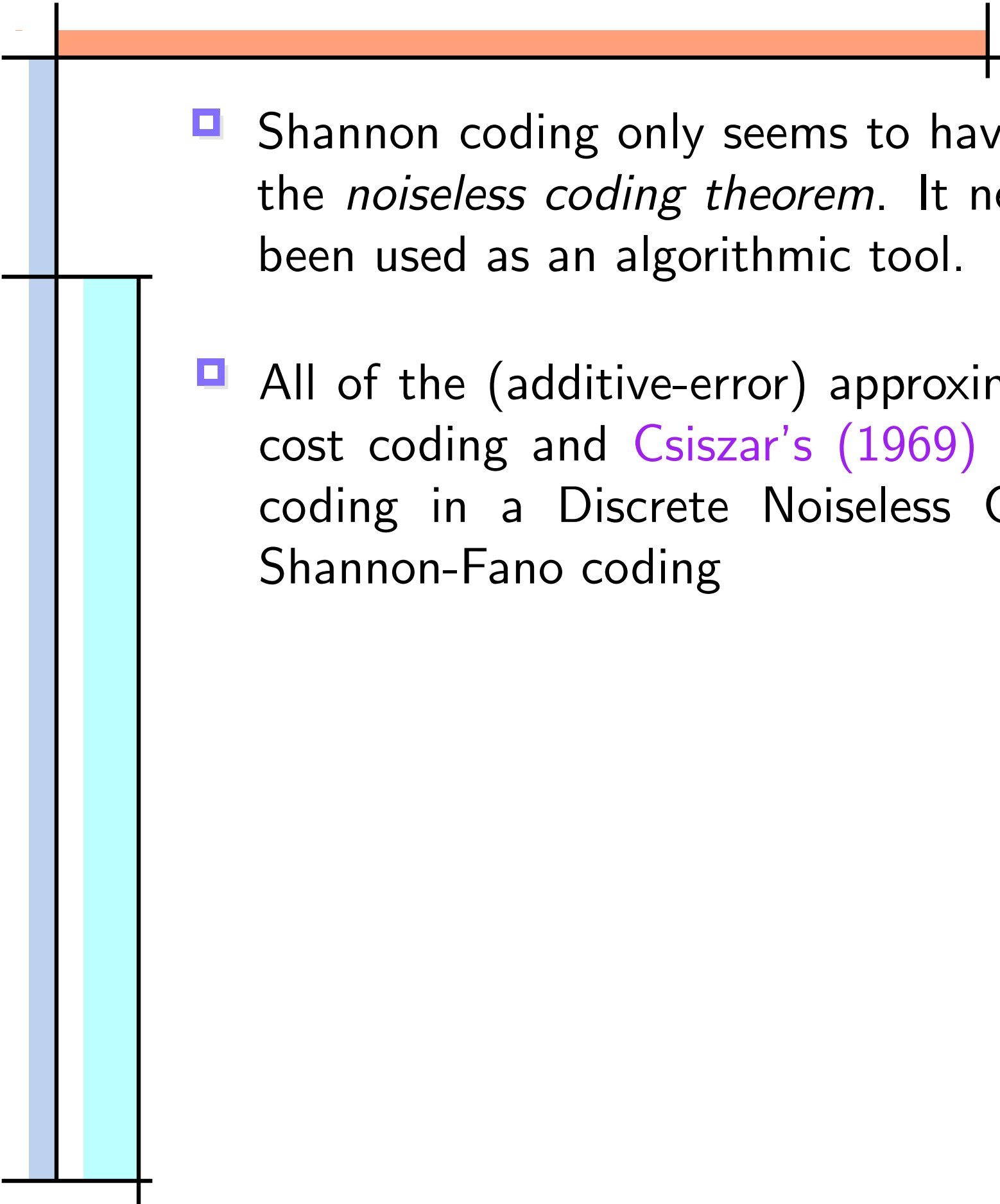
Note: If $c_1 = c_2 = 1$ then $\phi = 2$ and this is classic “Shannon Information Theoretic Lower Bound”

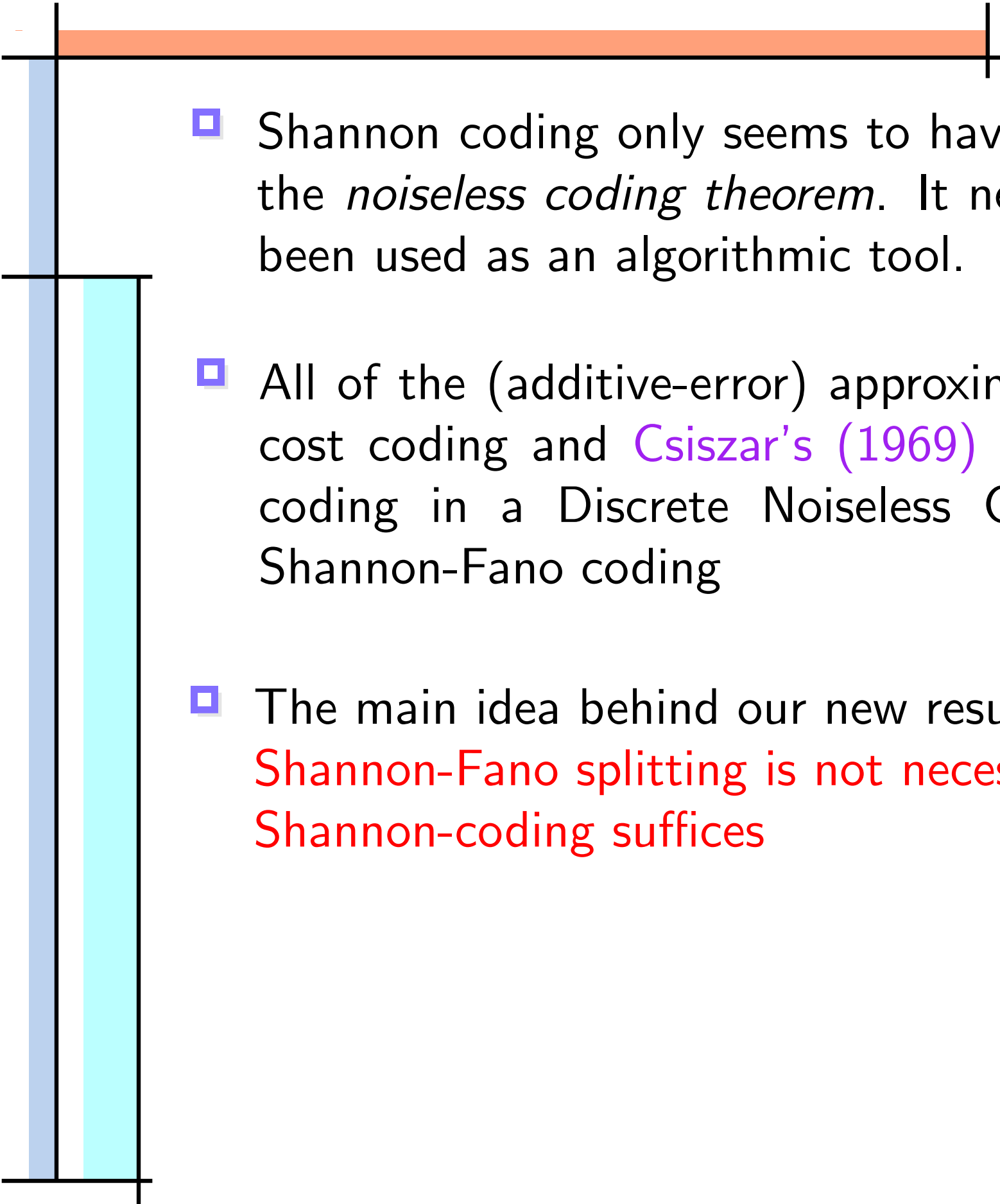


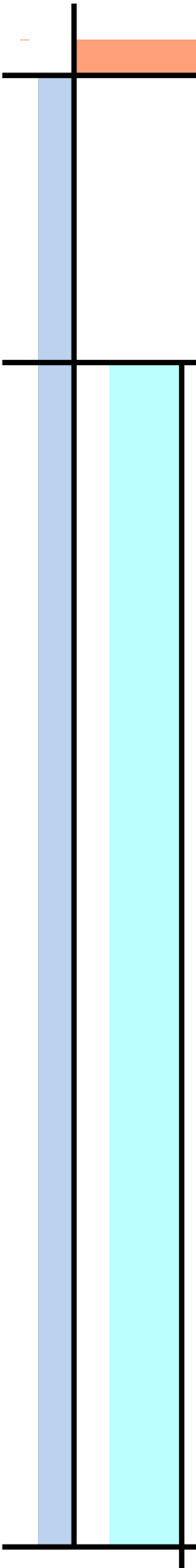
Outline

- Huffman Coding and Generalizations
- Previous Work & Background
- **New Work**
- A “Counterexample”
- Open Problems

- 
- Shannon coding only seems to have been used in the proof of the *noiseless coding theorem*. It never seems to have actually been used as an algorithmic tool.

- 
- Shannon coding only seems to have been used in the proof of the *noiseless coding theorem*. It never seems to have actually been used as an algorithmic tool.
 - All of the (additive-error) approximation algorithms for unequal cost coding and [Csiszar's \(1969\)](#) approximation algorithm for coding in a Discrete Noiseless Channel, were variations of Shannon-Fano coding

- 
- Shannon coding only seems to have been used in the proof of the *noiseless coding theorem*. It never seems to have actually been used as an algorithmic tool.
 - All of the (additive-error) approximation algorithms for unequal cost coding and Csiszar's (1969) approximation algorithm for coding in a Discrete Noiseless Channel, were variations of Shannon-Fano coding
 - The main idea behind our new results is that **Shannon-Fano splitting is not necessary; Shannon-coding suffices**

- 
- Shannon coding only seems to have been used in the proof of the *noiseless coding theorem*. It never seems to have actually been used as an algorithmic tool.
 - All of the (additive-error) approximation algorithms for unequal cost coding and [Csiszar's \(1969\)](#) approximation algorithm for coding in a Discrete Noiseless Channel, were variations of Shannon-Fano coding
 - The main idea behind our new results is that **Shannon-Fano splitting is not necessary; Shannon-coding suffices**
 - Yields efficient additive-error approximation algorithms for unequal cost coding and the Discrete Noiseless Channel, as well as for regular language constraints.



New Results for Unequal Cost Coding

New Results for Unequal Cost Coding

- Given coding letter lengths \mathcal{C} , let ϕ be capacity. Then $\exists K > 0$, depending only upon \mathcal{C} , such that if
 1. $P = \{p_1, p_2, \dots, p_n\}$ is any P.D., and
 2. l_1, l_2, \dots, l_n any set of integers such that
$$\forall i, l_i \geq K + \lceil -\log_{\phi} p_i \rceil,$$then there exists a prefix free code for which the l_i are the word lengths.

New Results for Unequal Cost Coding

- Given coding letter lengths \mathcal{C} , let ϕ be capacity. Then $\exists K > 0$, depending only upon \mathcal{C} , such that if
 - $P = \{p_1, p_2, \dots, p_n\}$ is any P.D., and
 - l_1, l_2, \dots, l_n any set of integers such that $\forall i, l_i \geq K + \lceil -\log_{\phi} p_i \rceil$,

then there exists a prefix free code for which the l_i are the word lengths.

$$\Rightarrow \sum_i p_i l_i \leq K + 1 + H_{\phi}(P) \leq OPT + K + 1$$

New Results for Unequal Cost Coding

- Given coding letter lengths \mathcal{C} , let ϕ be capacity. Then $\exists K > 0$, depending only upon \mathcal{C} , such that if
 - $P = \{p_1, p_2, \dots, p_n\}$ is any P.D., and
 - l_1, l_2, \dots, l_n any set of integers such that $\forall i, l_i \geq K + \lceil -\log_{\phi} p_i \rceil$,

then there exists a prefix free code for which the l_i are the word lengths.

$$\Rightarrow \sum_i p_i l_i \leq K + 1 + H_{\phi}(P) \leq OPT + K + 1$$

- This gives an additive approximation of same type as Shannon-Fano splitting without the splitting (same time complexity but many fewer operations on reals).

New Results for Unequal Cost Coding

- Given coding letter lengths \mathcal{C} , let ϕ be capacity. Then $\exists K > 0$, depending only upon \mathcal{C} , such that if
 - $P = \{p_1, p_2, \dots, p_n\}$ is any P.D., and
 - l_1, l_2, \dots, l_n any set of integers such that $\forall i, l_i \geq K + \lceil -\log_{\phi} p_i \rceil$,

then there exists a prefix free code for which the l_i are the word lengths.

$$\Rightarrow \sum_i p_i l_i \leq K + 1 + H_{\phi}(P) \leq OPT + K + 1$$

- Same result holds for DNC and regular language restrictions. ϕ is a function of the DNC or \mathcal{L} -accepting automaton graph

Proof of the Theorem

- We first prove the following lemma.

Given \mathcal{C} and corresponding ϕ then

$\exists \beta > 0$ depending only upon \mathcal{C} such that if

$$\sum_{i=1}^n \phi^{-l_i} \leq \beta,$$

then there exists a prefix-free code with word lengths l_1, l_2, \dots, l_n .

Proof of the Theorem

- We first prove the following lemma.

Given \mathcal{C} and corresponding ϕ then

$\exists \beta > 0$ depending only upon \mathcal{C} such that if

$$\sum_{i=1}^n \phi^{-l_i} \leq \beta,$$

then there exists a prefix-free code with word lengths l_1, l_2, \dots, l_n .

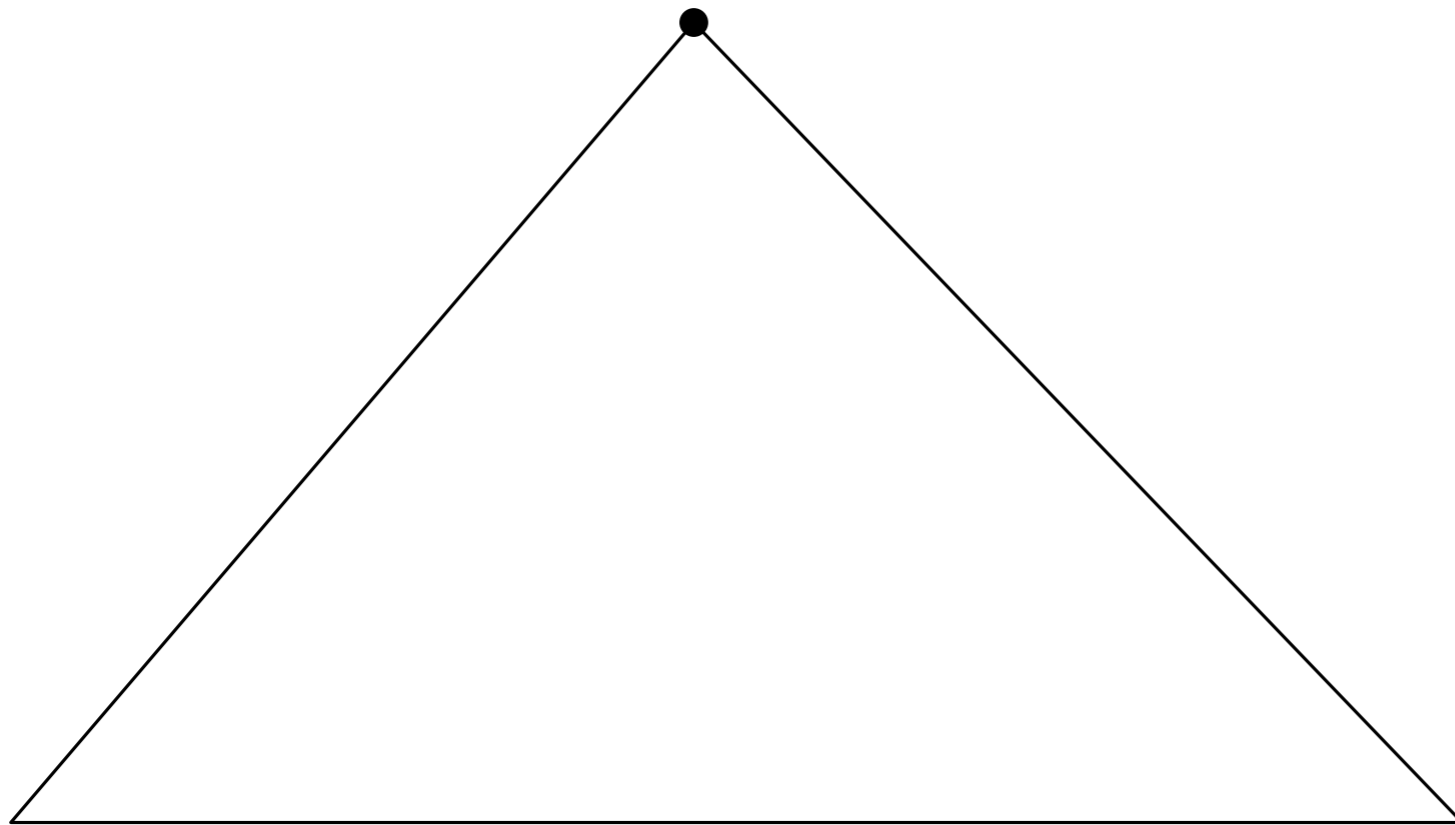
- *Note: if $c_1 = c_2 = 1$ then $\phi = 2$. Let $\beta = 1$ and condition becomes $\sum 2^{-l_i} \leq 1$.*

Lemma then becomes one direction of **Kraft Inequality**.

Proof of the Lemma

- Let $L(n)$ be the number of nodes on level n of the infinite tree corresponding to \mathcal{C}

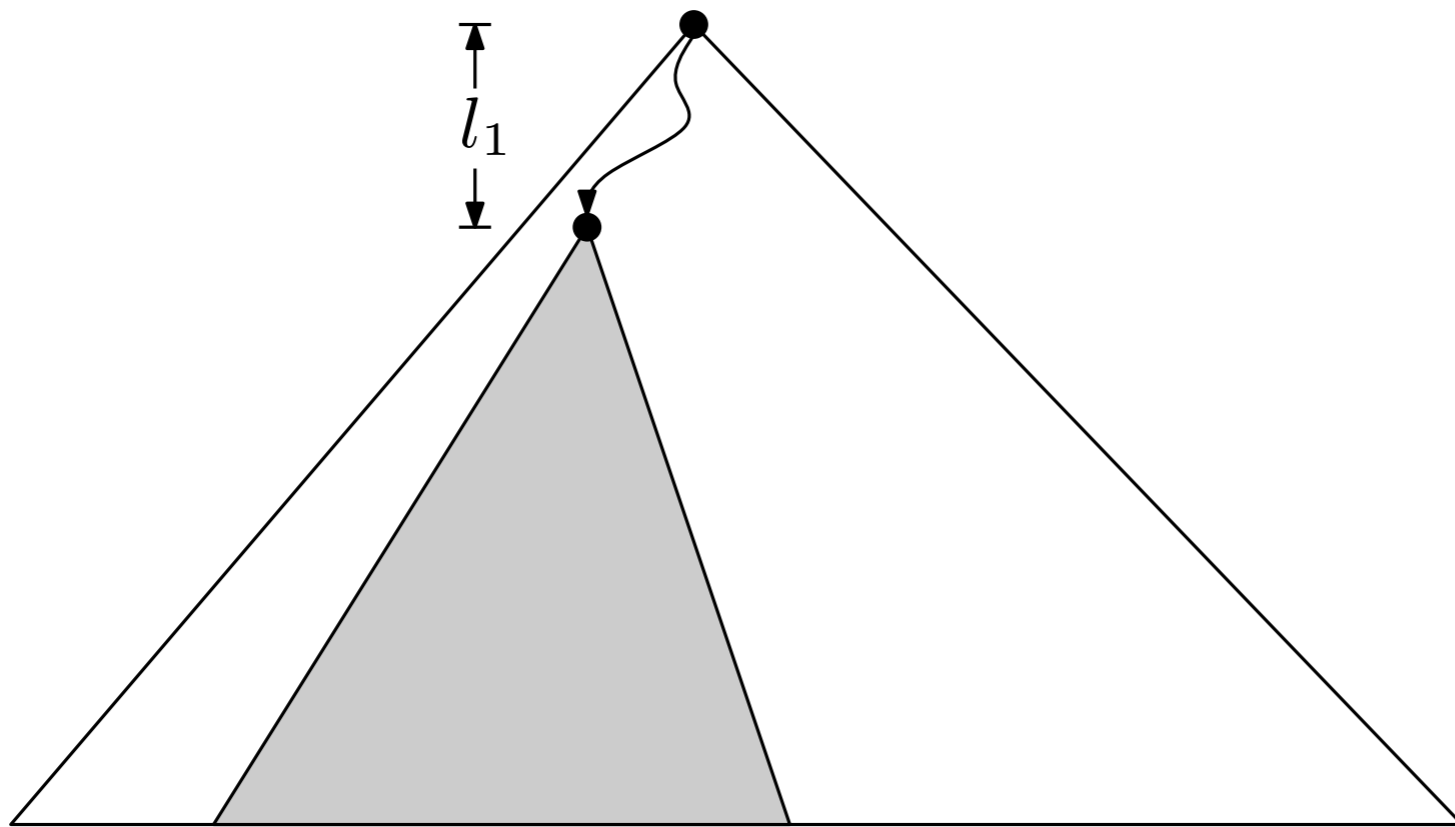
Can show $\exists t_1, t_2$ s.t., $t_1\phi^n \leq L(n) \leq t_2\phi^n$.



Proof of the Lemma

- Let $L(n)$ be the number of nodes on level n of the infinite tree corresponding to \mathcal{C}

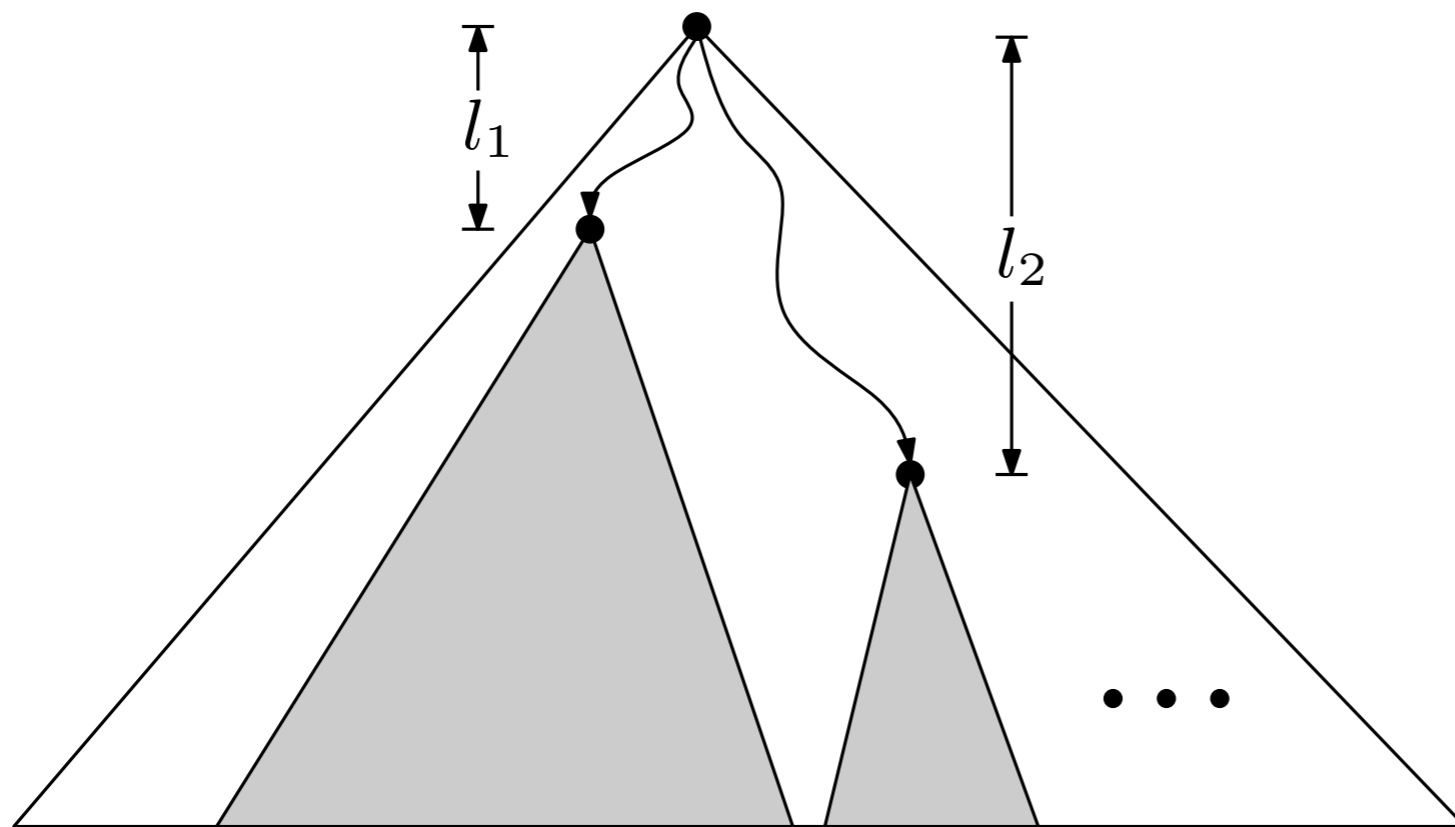
Can show $\exists t_1, t_2$ s.t., $t_1\phi^n \leq L(n) \leq t_2\phi^n$.



Proof of the Lemma

- Let $L(n)$ be the number of nodes on level n of the infinite tree corresponding to \mathcal{C}

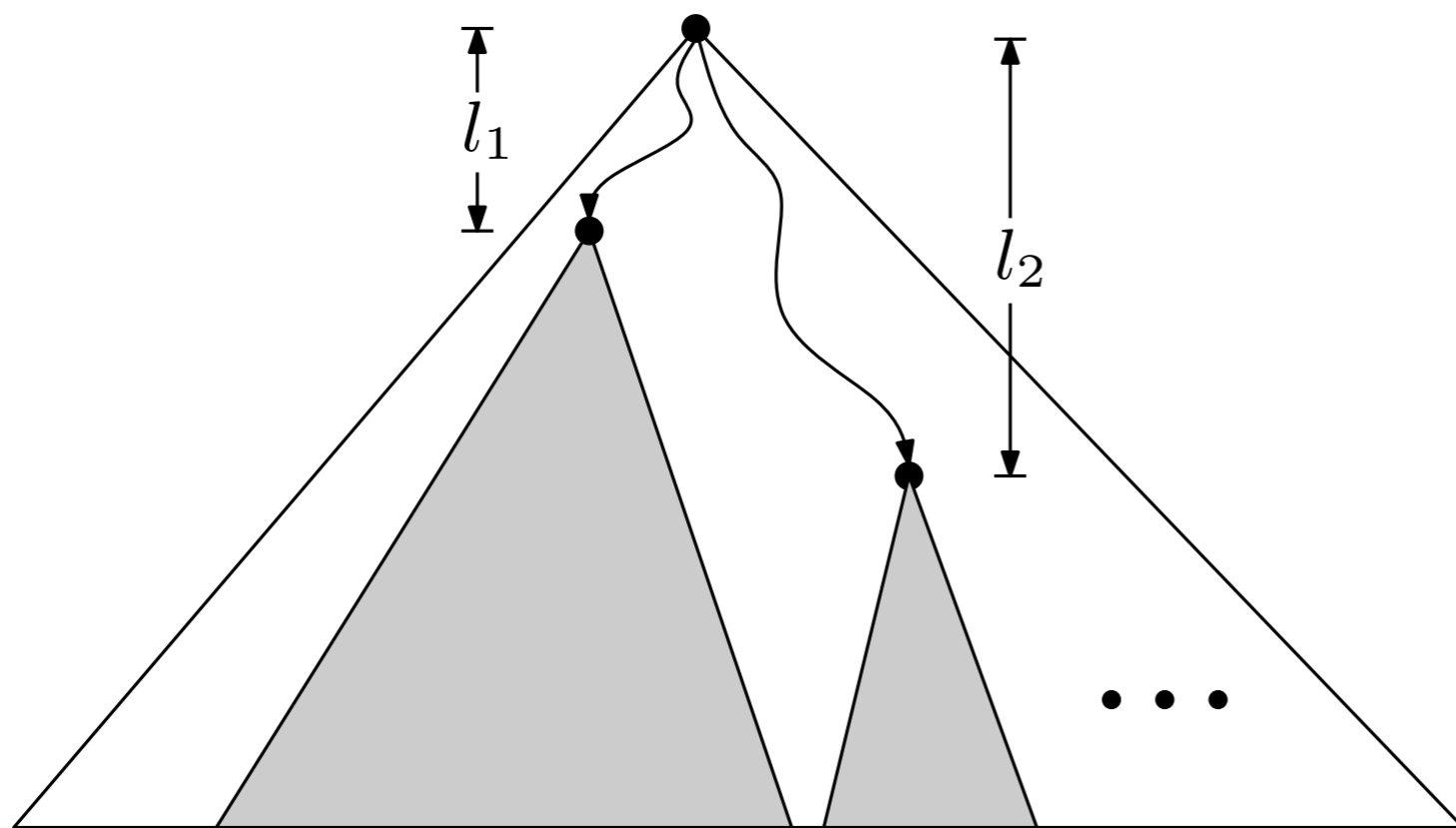
Can show $\exists t_1, t_2$ s.t., $t_1\phi^n \leq L(n) \leq t_2\phi^n$.



Proof of the Lemma

- Let $L(n)$ be the number of nodes on level n of the infinite tree corresponding to \mathcal{C}

Can show $\exists t_1, t_2$ s.t., $t_1\phi^n \leq L(n) \leq t_2\phi^n$.

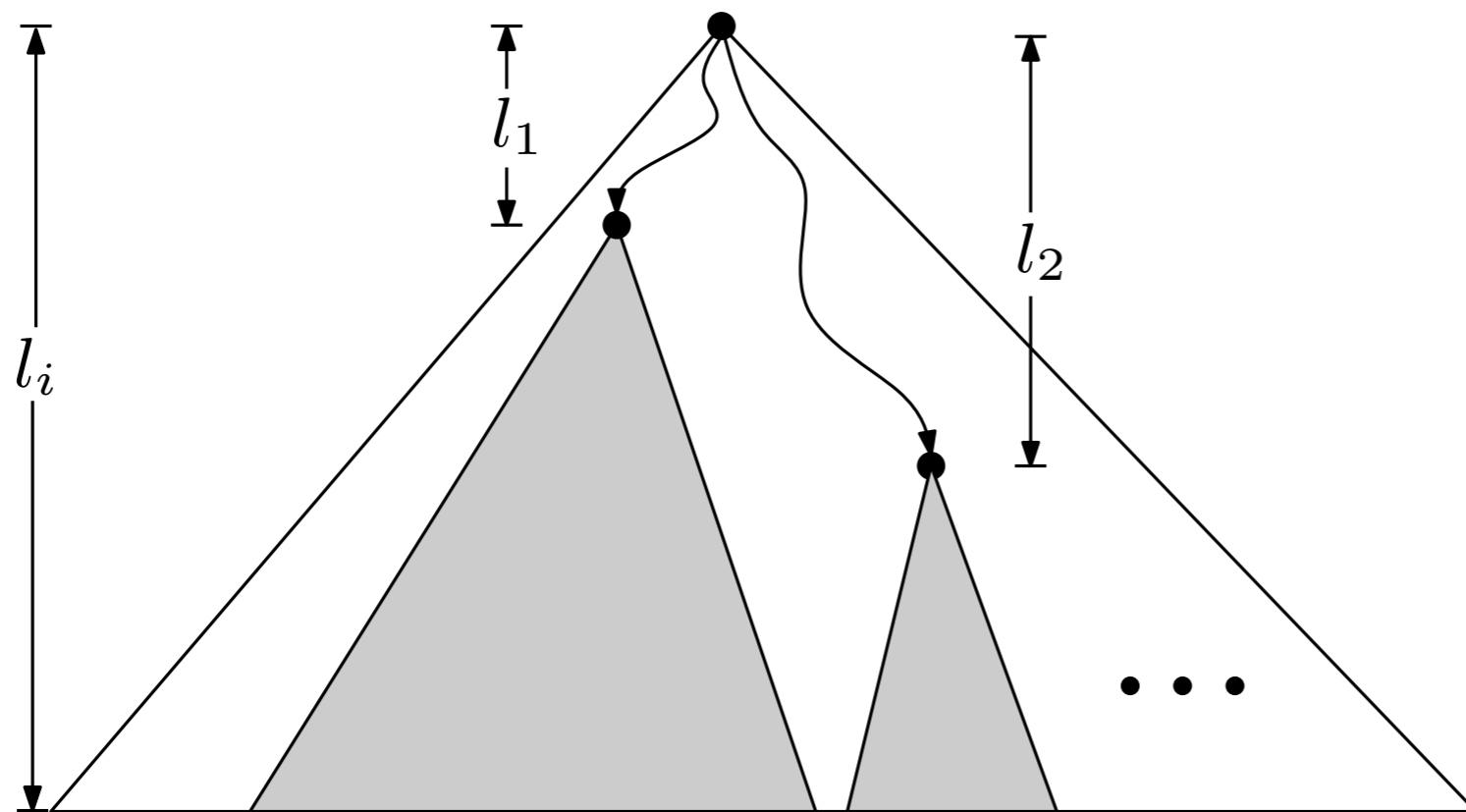


Grey regions are parts of infinite tree that are erased when node k on l_k becomes leaf.

Proof of the Lemma

- Let $L(n)$ be the number of nodes on level n of the infinite tree corresponding to \mathcal{C}

Can show $\exists t_1, t_2$ s.t., $t_1\phi^n \leq L(n) \leq t_2\phi^n$.



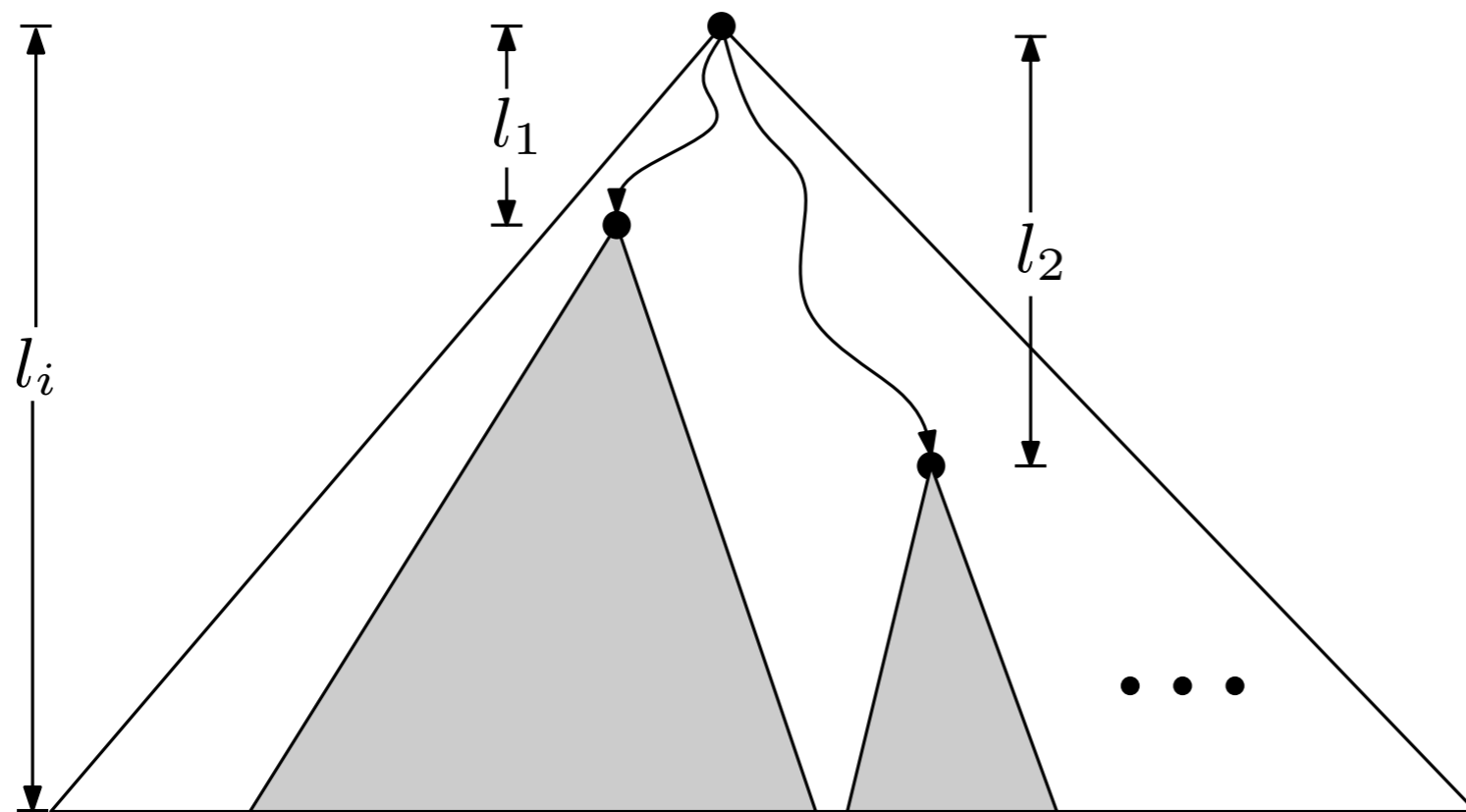
Grey regions are parts of infinite tree that are erased when node k on l_k becomes leaf.

Node on l_k has $L(l_i - l_k)$ descendants on l_i

Proof of the Lemma

- Let $L(n)$ be the number of nodes on level n of the infinite tree corresponding to \mathcal{C}

Can show $\exists t_1, t_2$ s.t., $t_1 \phi^n \leq L(n) \leq t_2 \phi^n$.



Grey regions are parts of infinite tree that are erased when node k on l_k becomes leaf.

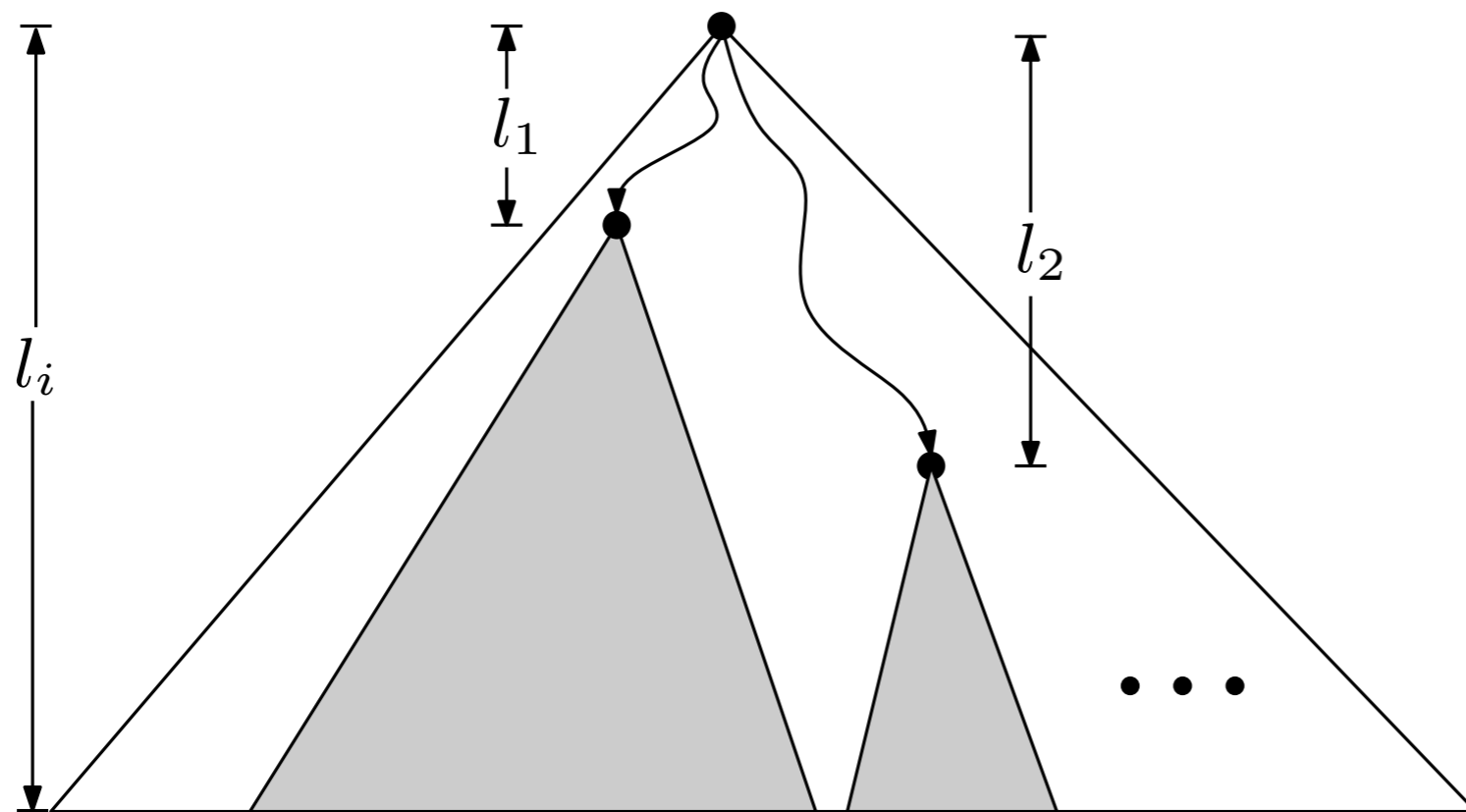
Node on l_k has $L(l_i - l_k)$ descendants on l_i

Node on l_i can become leaf iff grey regions do not cover all nodes on level l_i

Proof of the Lemma

- Let $L(n)$ be the number of nodes on level n of the infinite tree corresponding to \mathcal{C}

Can show $\exists t_1, t_2$ s.t., $t_1 \phi^n \leq L(n) \leq t_2 \phi^n$.



Grey regions are parts of infinite tree that are erased when node k on l_k becomes leaf.

Node on l_k has $L(l_i - l_k)$ descendants on l_i

Node on l_i can become leaf iff grey regions do not cover all nodes on level l_i

$$\sum_{k=1}^{i-1} L(l_i - l_k) < L(l_i)$$



Proof of the Lemma

- Just need to show that $0 < L(\ell_i) - \sum_{k=1}^{i-1} L(\ell - \ell_k)$.

Proof of the Lemma

□ Just need to show that $0 < L(\ell_i) - \sum_{k=1}^{i-1} L(\ell - \ell_k)$.

□

$$\begin{aligned} L(\ell_i) - \sum_{k=1}^{i-1} L(\ell - \ell_k) &\geq t_1 \phi^\ell - t_2 \sum_{k=1}^{i-1} \phi^{\ell - \ell_k} \\ &\geq \phi^\ell \left(t_1 - t_2 \sum_{k=1}^{i-1} \phi^{-\ell_k} \right) \\ &\geq \phi^\ell (t_1 - t_2 \beta) \end{aligned}$$

Proof of the Lemma

□ Just need to show that $0 < L(\ell_i) - \sum_{k=1}^{i-1} L(\ell - \ell_k)$.

□

$$\begin{aligned} L(\ell_i) - \sum_{k=1}^{i-1} L(\ell - \ell_k) &\geq t_1 \phi^\ell - t_2 \sum_{k=1}^{i-1} \phi^{\ell - \ell_k} \\ &\geq \phi^\ell \left(t_1 - t_2 \sum_{k=1}^{i-1} \phi^{-\ell_k} \right) \\ &\geq \phi^\ell (t_1 - t_2 \beta) \end{aligned}$$

□ Choose $\beta < \frac{t_1}{t_2}$

Proof of the Lemma

□ Just need to show that $0 < L(\ell_i) - \sum_{k=1}^{i-1} L(\ell - \ell_k)$.

□

$$\begin{aligned} L(\ell_i) - \sum_{k=1}^{i-1} L(\ell - \ell_k) &\geq t_1 \phi^\ell - t_2 \sum_{k=1}^{i-1} \phi^{\ell - \ell_k} \\ &\geq \phi^\ell \left(t_1 - t_2 \sum_{k=1}^{i-1} \phi^{-\ell_k} \right) \\ &\geq \phi^\ell (t_1 - t_2 \beta) \end{aligned}$$

□ Choose $\beta < \frac{t_1}{t_2} > 0$

Proof of the Main Theorem

- Set $K = -\log_{\phi} \beta$. (Recall $l_i \geq K + \lceil -\log_{\phi} p_i \rceil$)
Then

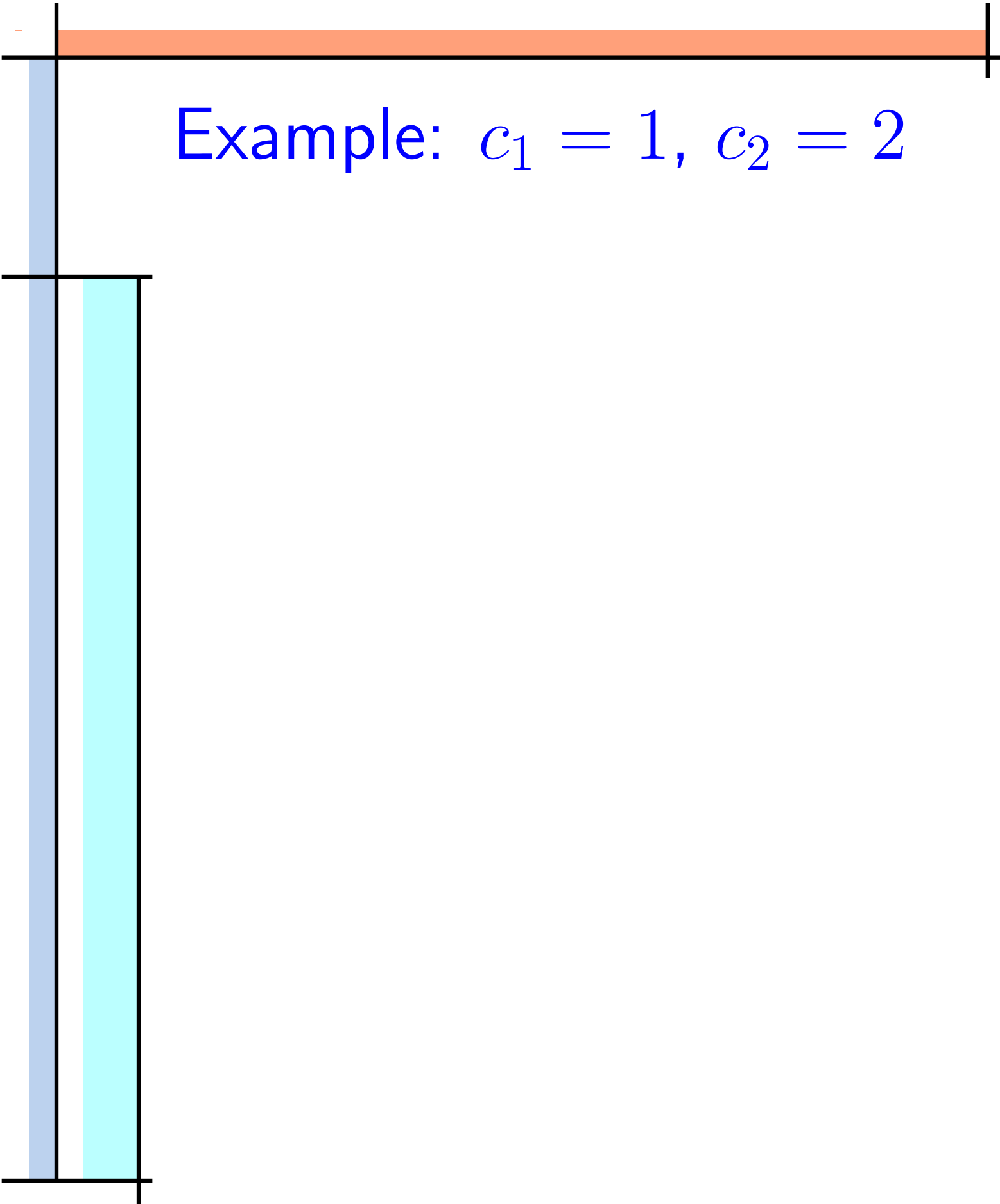
$$\begin{aligned} \sum_{i=1}^n \phi^{-l_i} &\leq \sum_{i=1}^n \phi^{-K - \lceil -\log_{\phi} p_i \rceil} \\ &\leq \beta \sum_{i=1}^n \phi^{\log_{\phi} p_i} = \beta \sum_{i=1}^n p_i = \beta \end{aligned}$$

Proof of the Main Theorem

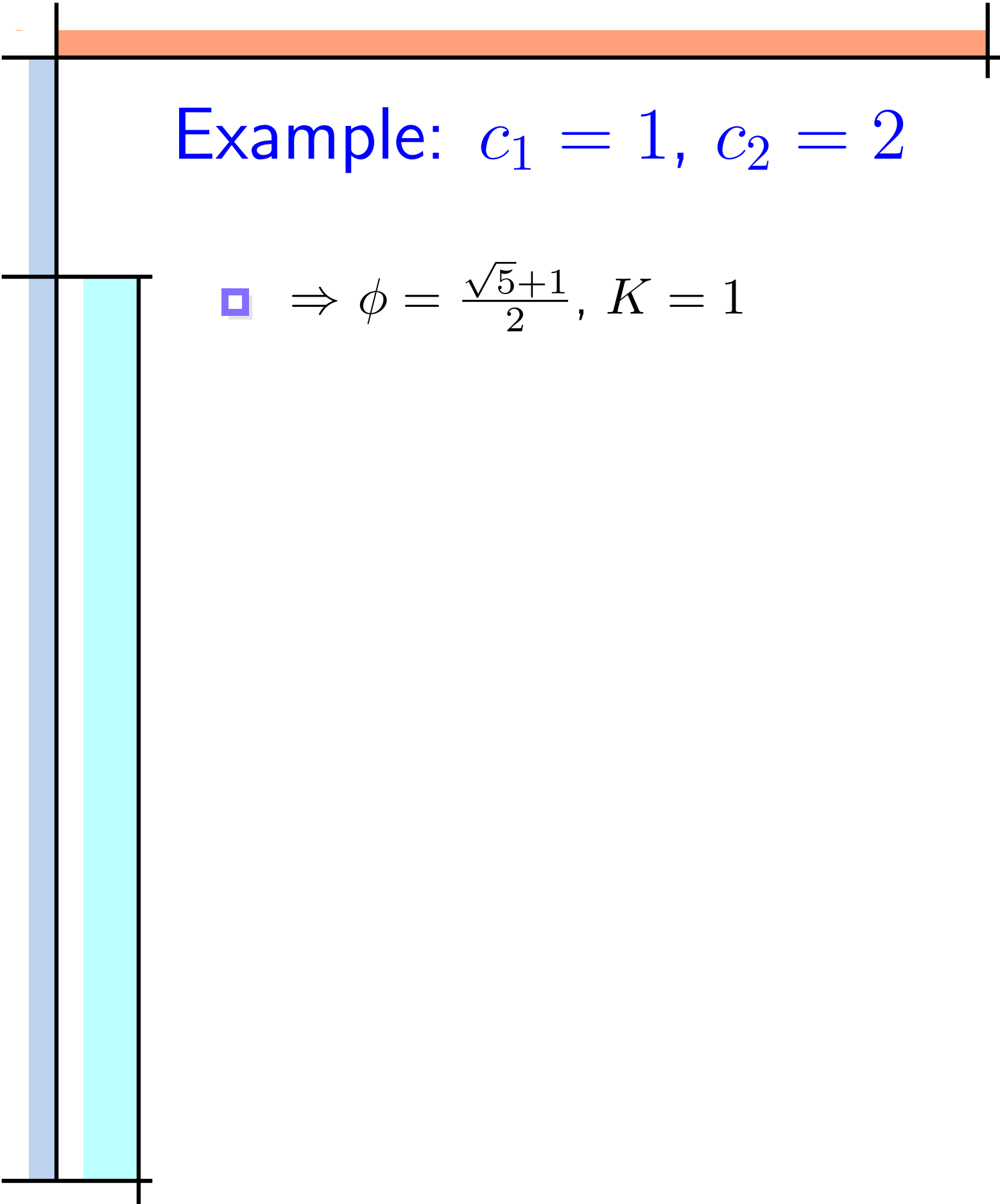
- Set $K = -\log_{\phi} \beta$. (Recall $l_i \geq K + \lceil -\log_{\phi} p_i \rceil$)
Then

$$\begin{aligned} \sum_{i=1}^n \phi^{-l_i} &\leq \sum_{i=1}^n \phi^{-K - \lceil -\log_{\phi} p_i \rceil} \\ &\leq \beta \sum_{i=1}^n \phi^{\log_{\phi} p_i} = \beta \sum_{i=1}^n p_i = \beta \end{aligned}$$

- From previous lemma, a prefix free code with those word lengths l_1, l_2, \dots, l_n exists, and we are done

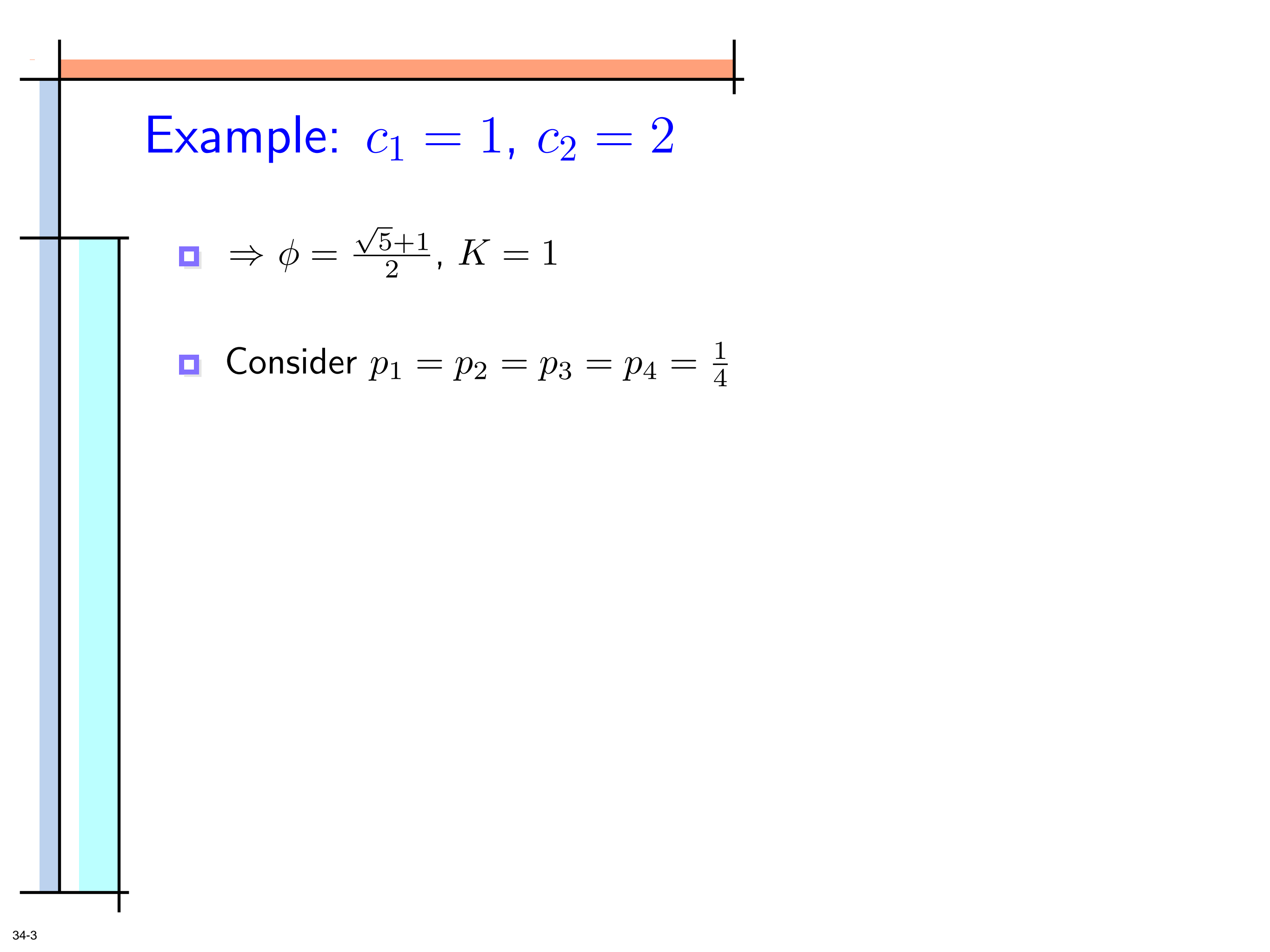


Example: $c_1 = 1, c_2 = 2$



Example: $c_1 = 1, c_2 = 2$

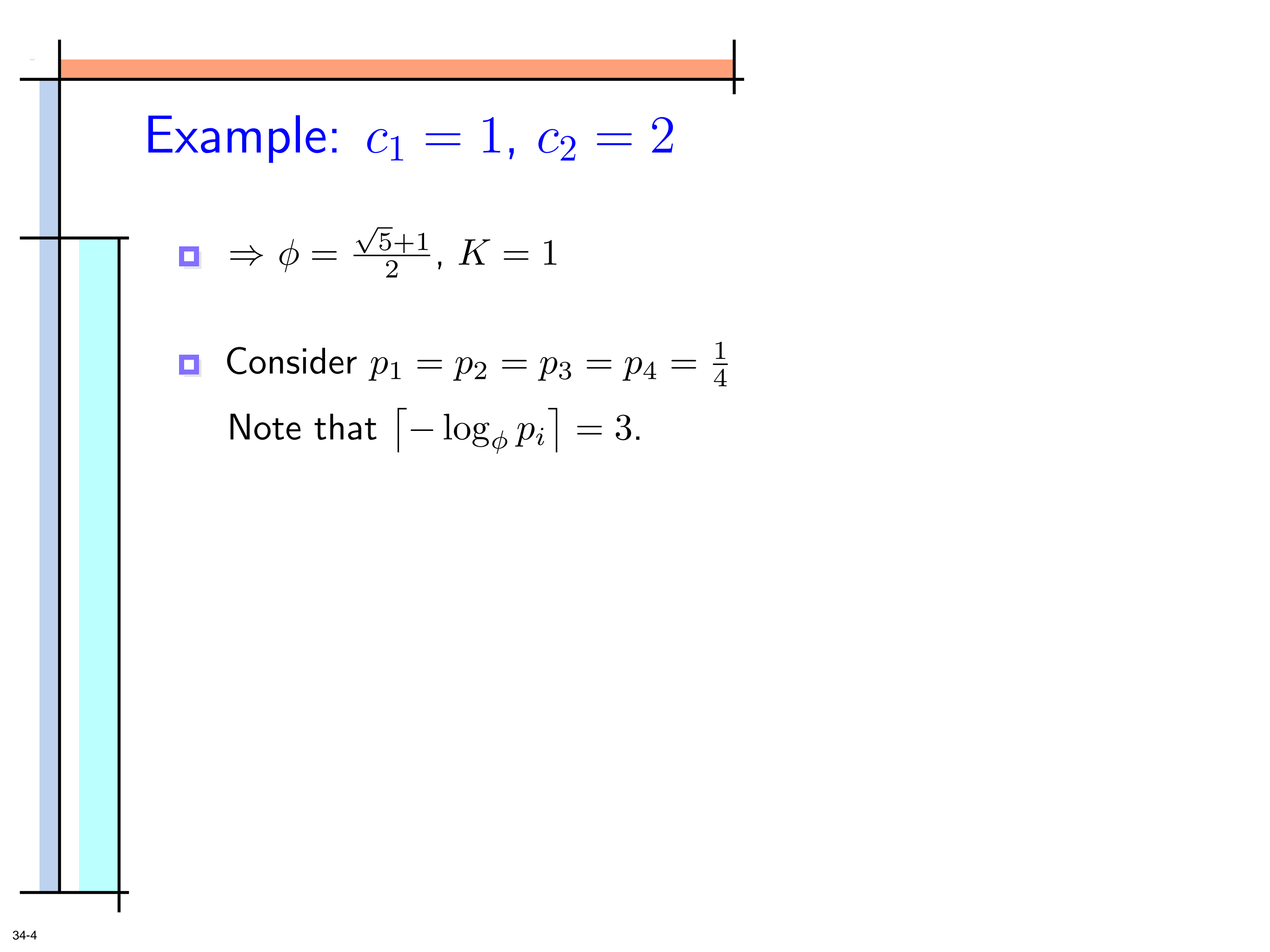
□ $\Rightarrow \phi = \frac{\sqrt{5}+1}{2}, K = 1$



Example: $c_1 = 1, c_2 = 2$

□ $\Rightarrow \phi = \frac{\sqrt{5}+1}{2}, K = 1$

□ Consider $p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$



Example: $c_1 = 1, c_2 = 2$

□ $\Rightarrow \phi = \frac{\sqrt{5}+1}{2}, K = 1$

□ Consider $p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$

Note that $\lceil -\log_{\phi} p_i \rceil = 3$.

Example: $c_1 = 1, c_2 = 2$

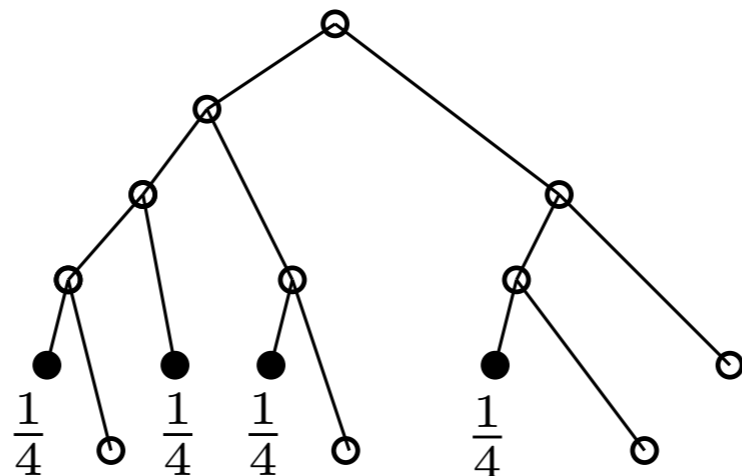
- $\Rightarrow \phi = \frac{\sqrt{5}+1}{2}, K = 1$

- Consider $p_1 = p_2 = p_3 = p_4 = \frac{1}{4}$

Note that $\lceil -\log_{\phi} p_i \rceil = 3$.

- No tree with $l_i = 3$ exists.

But, a tree with $l_i = \lceil -\log_{\phi} p_i \rceil + 1 = 4$ does!





The Algorithm

- A valid K could be found by working through the proof of Theorem. Technically, $O(1)$ but, practically, this would require some complicated operations on reals.

The Algorithm

- A valid K could be found by working through the proof of Theorem. Technically, $O(1)$ but, practically, this would require some complicated operations on reals.
- Alternatively, perform **doubling search for K** , the smallest K for which theorem is valid.

Set $\bar{K} = 1, 2, 2^2, 2^3 \dots$

Test if $l_i = \bar{K} + \lceil -\log_{\phi} p_i \rceil$ has valid code (can be done efficiently)

until \bar{K} is good but $\bar{K}/2$ is not.

The Algorithm

- A valid K could be found by working through the proof of Theorem. Technically, $O(1)$ but, practically, this would require some complicated operations on reals.
- Alternatively, perform **doubling search for K** , the smallest K for which theorem is valid.

Set $\bar{K} = 1, 2, 2^2, 2^3 \dots$

Test if $l_i = \bar{K} + \lceil -\log_{\phi} p_i \rceil$ has valid code (can be done efficiently)

until \bar{K} is good but $\bar{K}/2$ is not.

Note that $\bar{K}/2 < K \leq \bar{K}$

The Algorithm

- A valid K could be found by working through the proof of Theorem. Technically, $O(1)$ but, practically, this would require some complicated operations on reals.
- Alternatively, perform **doubling search for K** , the smallest K for which theorem is valid.

Set $\bar{K} = 1, 2, 2^2, 2^3 \dots$

Test if $\ell_i = \bar{K} + \lceil -\log_{\phi} p_i \rceil$ has valid code (can be done efficiently)

until \bar{K} is good but $\bar{K}/2$ is not.

Note that $\bar{K}/2 < K \leq \bar{K}$

Now set $a = \bar{K}/2, b = \bar{K}$, and binary search for K in $[a, b]$.

The Algorithm

- ▣ A valid K could be found by working through the proof of Theorem. Technically, $O(1)$ but, practically, this would require some complicated operations on reals.
- ▣ Alternatively, perform **doubling search for K** , the smallest K for which theorem is valid.

Set $\bar{K} = 1, 2, 2^2, 2^3 \dots$

Test if $l_i = \bar{K} + \lceil -\log_{\phi} p_i \rceil$ has valid code (can be done efficiently)
until \bar{K} is good but $\bar{K}/2$ is not.

Note that $\bar{K}/2 < K \leq \bar{K}$

Now set $a = \bar{K}/2, b = \bar{K}$, and binary search for K in $[a,b]$.

Subtle point: Search will find $K' \leq K$ for which code exists.

The Algorithm

- ▣ A valid K could be found by working through the proof of Theorem. Technically, $O(1)$ but, practically, this would require some complicated operations on reals.
- ▣ Alternatively, perform **doubling search for K** , the smallest K for which theorem is valid.

Set $\bar{K} = 1, 2, 2^2, 2^3 \dots$

Test if $l_i = \bar{K} + \lceil -\log_{\phi} p_i \rceil$ has valid code (can be done efficiently)

until \bar{K} is good but $\bar{K}/2$ is not.

Note that $\bar{K}/2 < K \leq \bar{K}$

Now set $a = \bar{K}/2, b = \bar{K}$, and binary search for K in $[a,b]$.

Subtle point: Search will find $K' \leq K$ for which code exists.

- ▣ **Time complexity $O(n \cdot \log K)$.**

The Algorithm for infinite encoding alphabets

□ Proof assumed two things.

(i) Root of $\sum \phi^{-c_i} = 1$ exists

(ii) $\exists t_1, t_2$ s.t., $t_1 \phi^n \leq L(n) \leq t_2 \phi^n$

$L(n)$ is number of nodes on level n of infinite tree

The Algorithm for infinite encoding alphabets

- Proof assumed two things.
 - (i) Root of $\sum \phi^{-c_i} = 1$ exists
 - (ii) $\exists t_1, t_2$ s.t., $t_1 \phi^n \leq L(n) \leq t_2 \phi^n$
 $L(n)$ is number of nodes on level n of infinite tree
- This is **always** true for finite encoding alphabet

The Algorithm for infinite encoding alphabets

- Proof assumed two things.
 - (i) Root of $\sum \phi^{-c_i} = 1$ exists
 - (ii) $\exists t_1, t_2$ s.t., $t_1 \phi^n \leq L(n) \leq t_2 \phi^n$
 $L(n)$ is number of nodes on level n of infinite tree
- This is **always** true for finite encoding alphabet
- Not necessarily true for infinite encoding alphabets
Will see simple example in next section

The Algorithm for infinite encoding alphabets

- Proof assumed two things.
 - (i) Root of $\sum \phi^{-c_i} = 1$ exists
 - (ii) $\exists t_1, t_2$ s.t., $t_1 \phi^n \leq L(n) \leq t_2 \phi^n$
 $L(n)$ is number of nodes on level n of infinite tree

- This is **always** true for finite encoding alphabet

- Not necessarily true for infinite encoding alphabets
Will see simple example in next section

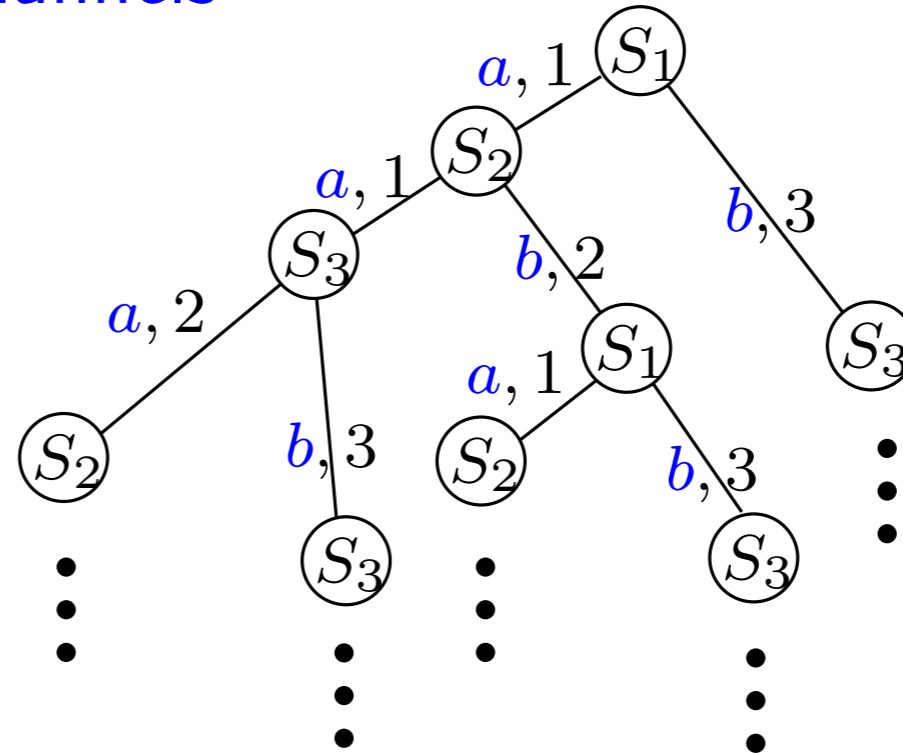
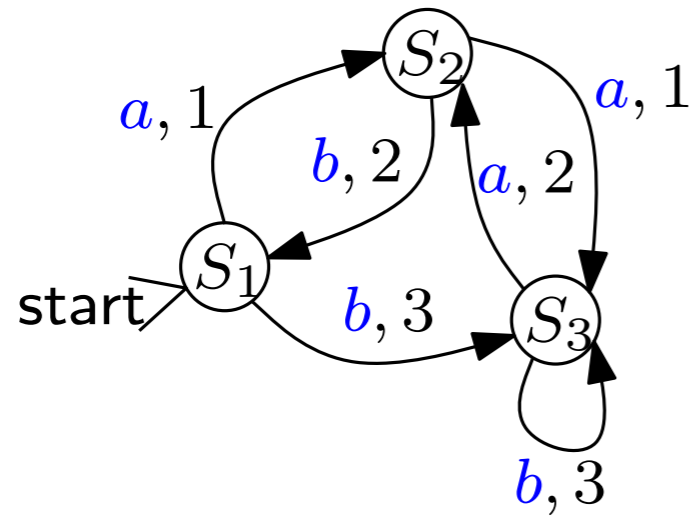
- But, if (i) and (ii) are true for an infinite alphabet
 \Rightarrow Theorem/algorithm hold

The Algorithm for infinite encoding alphabets

- Proof assumed two things.
 - (i) Root of $\sum \phi^{-c_i} = 1$ exists
 - (ii) $\exists t_1, t_2$ s.t., $t_1 \phi^n \leq L(n) \leq t_2 \phi^n$
 $L(n)$ is number of nodes on level n of infinite tree
- This is **always** true for finite encoding alphabet
- Not necessarily true for infinite encoding alphabets
Will see simple example in next section
- But, if (i) and (ii) are true for an infinite alphabet
 \Rightarrow Theorem/algorithm hold
- Example: '1'-Ended codes. $c_i = i$.
 $\Rightarrow \phi = \frac{1}{2}$ and (ii) is true \Rightarrow Theorem/algorithm hold

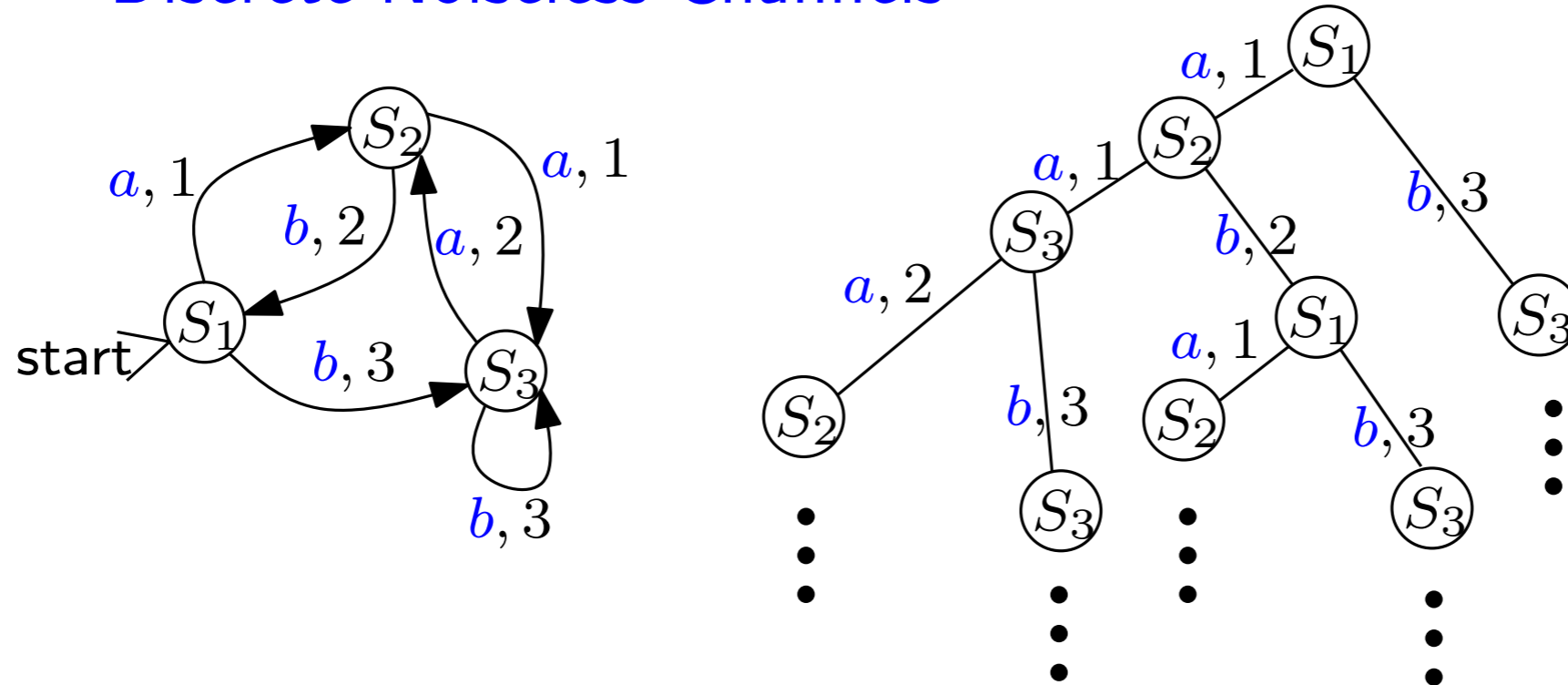
Extensions to DNC and Regular Language Restrictions

Discrete Noiseless Channels



Extensions to DNC and Regular Language Restrictions

Discrete Noiseless Channels



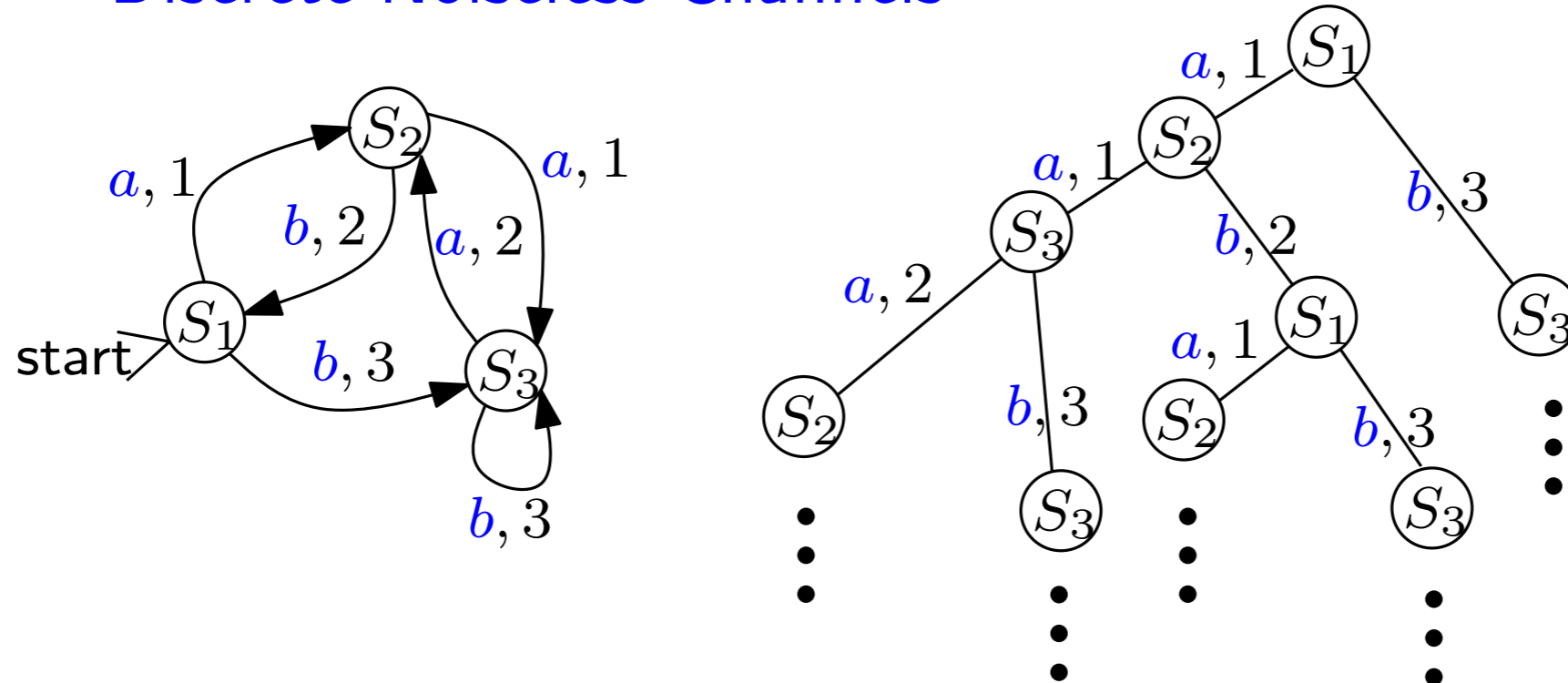
Let $L(n)$ be number of nodes on level n of infinite tree

Fact that graph is biconnected and “aperiodic” implies that

$$\exists, \phi, t_1, t_2 \text{ s.t., } t_1 \phi^n \leq L(n) \leq t_2 \phi^n$$

Extensions to DNC and Regular Language Restrictions

Discrete Noiseless Channels



Let $L(n)$ be number of nodes on level n of infinite tree

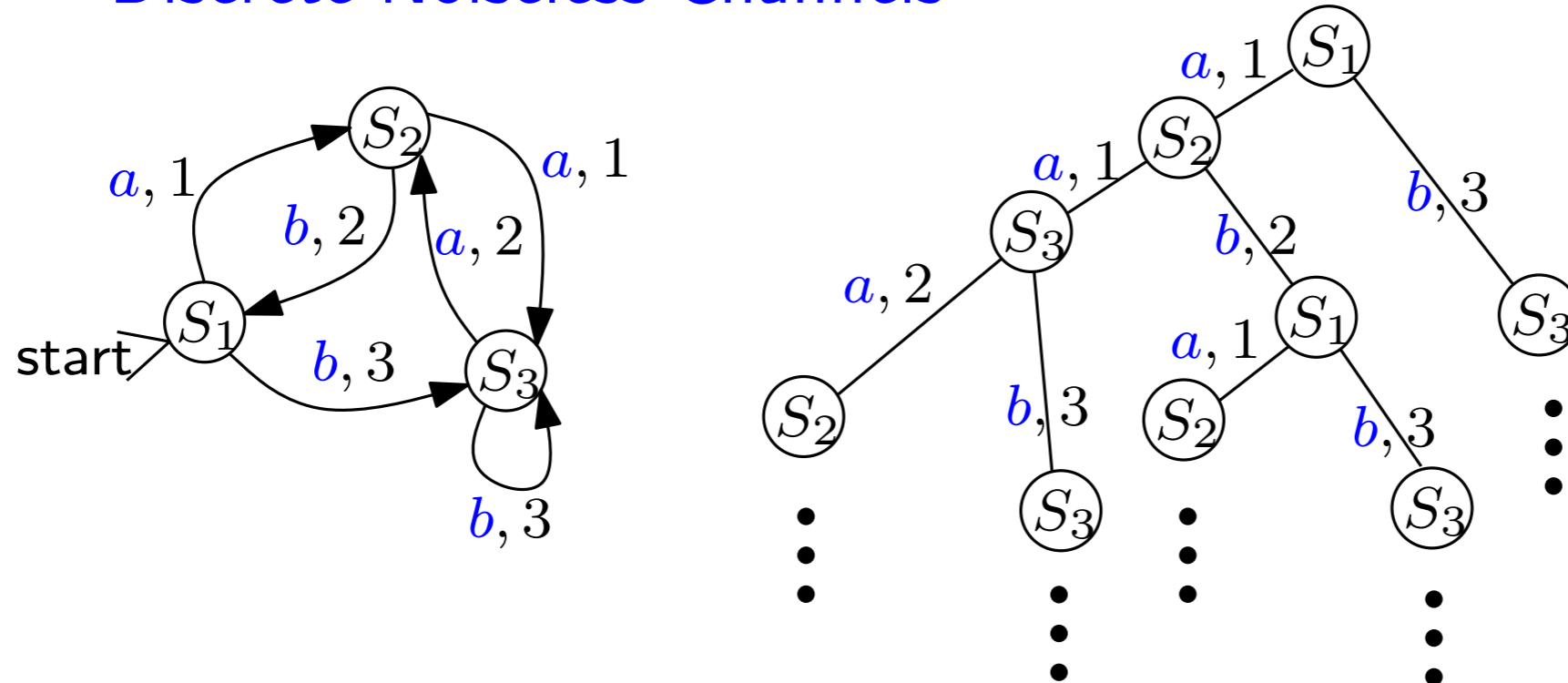
Fact that graph is biconnected and “aperiodic” implies that

$$\exists, \phi, t_1, t_2 \text{ s.t., } t_1 \phi^n \leq L(n) \leq t_2 \phi^n$$

Algorithm will still work for $l_i \geq K + \lceil -\log_\phi p_i \rceil$,

Extensions to DNC and Regular Language Restrictions

Discrete Noiseless Channels



Let $L(n)$ be number of nodes on level n of infinite tree

Fact that graph is biconnected and “aperiodic” implies that

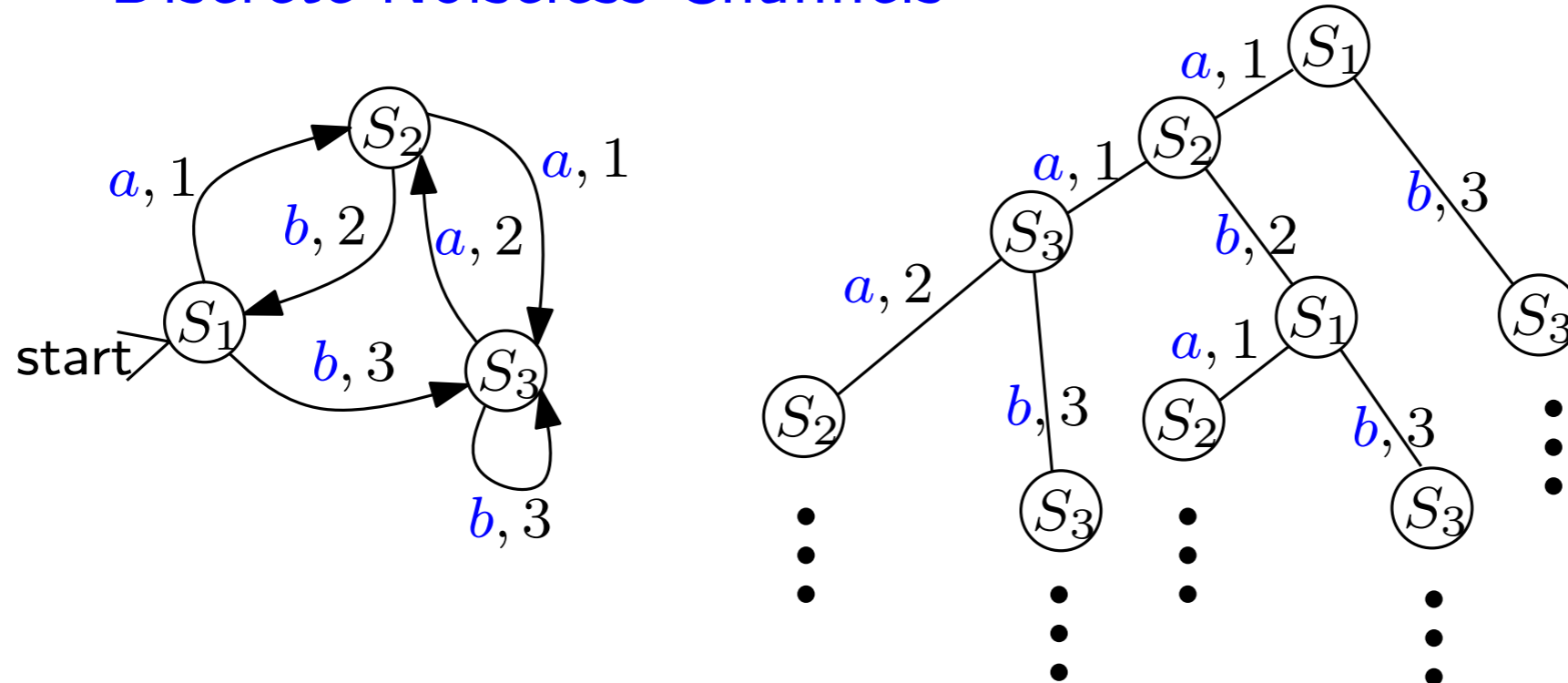
$$\exists, \phi, t_1, t_2 \text{ s.t., } t_1 \phi^n \leq L(n) \leq t_2 \phi^n$$

Algorithm will still work for $l_i \geq K + \lceil -\log_\phi p_i \rceil$,

Note: Algorithm must construct k different coding trees. One for each state (tree root).

Extensions to DNC and Regular Language Restrictions

Discrete Noiseless Channels



Let $L(n)$ be number of nodes on level n of infinite tree

Fact that graph is biconnected and “aperiodic” implies that

$$\exists, \phi, t_1, t_2 \text{ s.t., } t_1 \phi^n \leq L(n) \leq t_2 \phi^n$$

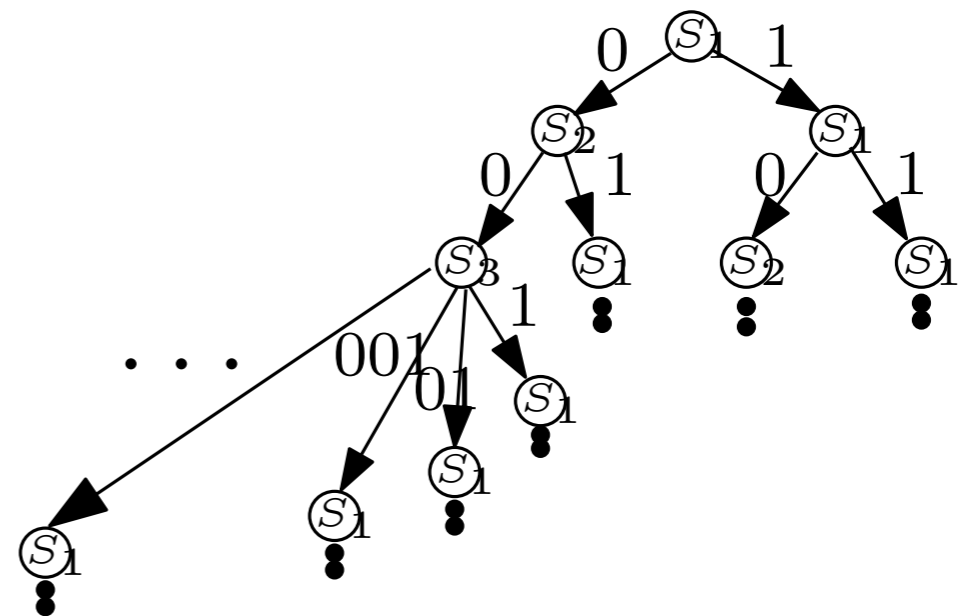
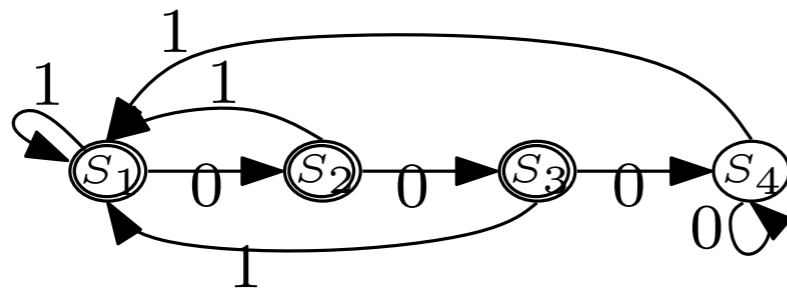
Algorithm will still work for $l_i \geq K + \lceil -\log_\phi p_i \rceil$,

Subtle point is that *any* node on level l_i can be chosen for p_i , independent of its state! Algorithm still works.

Extensions to DNC and Regular Language Restrictions

Regular Language Restrictions

Assumption: Language is 'aperiodic', i.e., $\exists N$, such that $\forall n > N$ there is at least one word of length n



Let $L(n)$ be number of nodes on level n of infinite tree
 Fact that language is "aperiodic" implies that

$$\exists, \phi, t_1, t_2 \text{ s.t., } t_1 \phi^n \leq L(n) \leq t_2 \phi^n$$

ϕ is largest dominant 'eigenvalue' of a conn component of the DFA.

Algorithm will still work for $l_i \geq K + \lceil -\log_\phi p_i \rceil$,

Again, any node at level l_i can be labelled with p_i , independent of state



Outline

- Huffman Coding and Generalizations
- Previous Work & Background
- New Work
- A “Counterexample”
- Conclusion and Open Problems

A “Counterexample”

- Let \mathcal{C} be the countably infinite set defined by

$$|\{j \mid c_j = i\}| = 2C_{i-1}$$

where $C_i = \frac{1}{i+1} \binom{2i}{i}$ is the i -th Catalan number.

Constructing prefix-free codes with these \mathcal{C} can be shown to be **equivalent** to constructing **balanced binary prefix-free codes** in which, for every word, the number of ‘0’s equals the number of ‘1’s.

A “Counterexample”

- Let \mathcal{C} be the countably infinite set defined by

$$|\{j \mid c_j = i\}| = 2C_{i-1}$$

where $C_i = \frac{1}{i+1} \binom{2i}{i}$ is the i -th Catalan number.

Constructing prefix-free codes with these \mathcal{C} can be shown to be **equivalent** to constructing **balanced binary prefix-free codes** in which, for every word, the number of ‘0’s equals the number of ‘1’s.

- No efficient additive-error approximation known.

A “Counterexample”

- Let \mathcal{C} be the countably infinite set defined by

$$|\{j \mid c_j = i\}| = 2C_{i-1}$$

where $C_i = \frac{1}{i+1} \binom{2i}{i}$ is the i -th Catalan number.

Constructing prefix-free codes with these \mathcal{C} can be shown to be **equivalent** to constructing **balanced binary prefix-free codes** in which, for every word, the number of ‘0’s equals the number of ‘1’s.

- No efficient additive-error approximation known.
- For this problem, the length of a balanced word = # of ‘0’s in word.*
e.g., $|10| = 1$, $|001110| = 3$.

A “Counterexample”

Let \mathcal{L} be the set of all balanced binary words.

Set $Q = \{01, 10, 0011, 1100, 000111, \dots\}$,

the language of all balanced binary words without a balanced prefix.

Then $\mathcal{L} = Q^*$ and every word in \mathcal{L} can be uniquely decomposed into concatenation of words in Q .

A “Counterexample”

Let \mathcal{L} be the set of all balanced binary words.

Set $Q = \{01, 10, 0011, 1100, 000111, \dots\}$,

the language of all balanced binary words without a balanced prefix.

Then $\mathcal{L} = Q^*$ and every word in \mathcal{L} can be uniquely decomposed into concatenation of words in Q .

words of length i in Q is $2C_{i-1}$.

A “Counterexample”

Let \mathcal{L} be the set of all balanced binary words.

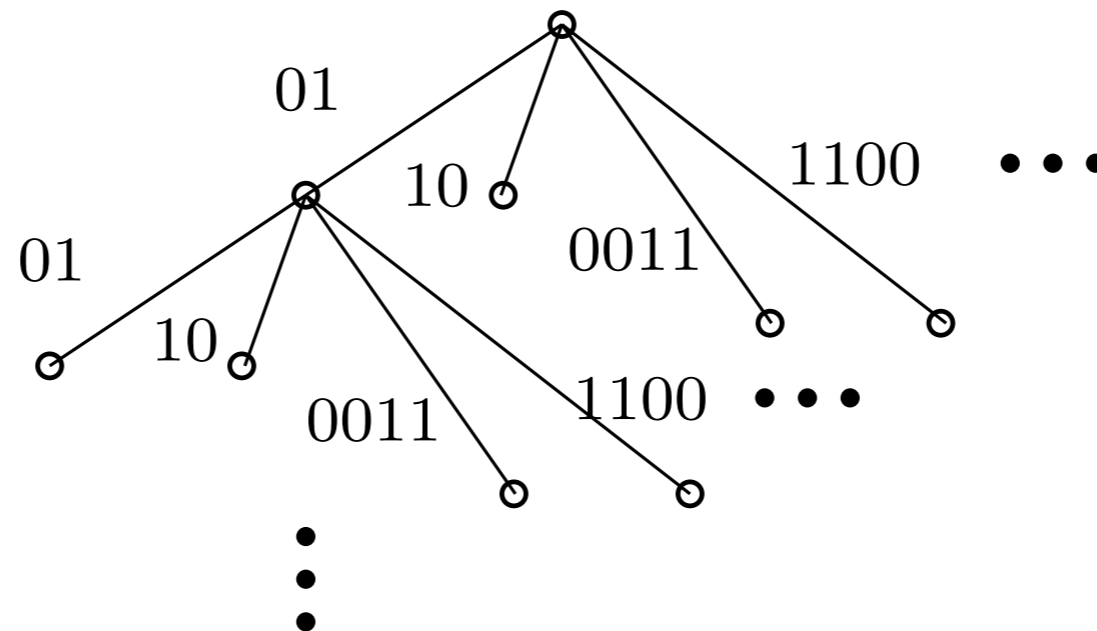
Set $Q = \{01, 10, 0011, 1100, 000111, \dots\}$,

the language of all balanced binary words without a balanced prefix.

Then $\mathcal{L} = Q^*$ and every word in \mathcal{L} can be uniquely decomposed into concatenation of words in Q .

words of length i in Q is $2C_{i-1}$.

Prefix coding in \mathcal{L} is equivalent to prefix coding with infinite alphabet Q .



A “Counterexample”

- Note: the characteristic equation is

$$g(z) = 1 - \sum_j \phi^{-c_j} = 1 - \sum_i 2C_{i-1} \phi^{-i} = \sqrt{1 - 4/\phi}$$

for which **root does not exist** ($\phi = 4$ is an algebraic singularity).

- Can prove that for $\forall \psi, K$, we can always find p_1, p_2, \dots, p_n s.t. there is **no** prefix code with length

$$l_i = K + \lceil \log_{\psi} p_i \rceil$$

A “Counterexample”

- $\phi = 4$ is algebraic singularity of characteristic equation

A “Counterexample”

- $\phi = 4$ is algebraic singularity of characteristic equation
- Can prove that for $\forall \psi \geq 4, K$, we can always find p_1, p_2, \dots, p_n s.t. there is **no** prefix code with length

$$l_i = K + \lceil \log_{\psi} p_i \rceil$$

A “Counterexample”

- $\phi = 4$ is algebraic singularity of characteristic equation
- Can prove that for $\forall \psi \geq 4, K$, we can always find p_1, p_2, \dots, p_n s.t. there is **no** prefix code with length

$$l_i = K + \lceil \log_{\psi} p_i \rceil$$

- Can also prove that for $\forall \psi < 4, K, \Delta$, we can always find p_1, p_2, \dots, p_n s.t. if prefix code with lengths $l_i \geq K + \lceil \log_{\psi} p_i \rceil$ exists, then

$$\sum_i l_i p_i - OPT > \Delta.$$

A “Counterexample”

- $\phi = 4$ is algebraic singularity of characteristic equation
- Can prove that for $\forall \psi \geq 4, K$, we can always find p_1, p_2, \dots, p_n s.t. there is **no** prefix code with length

$$l_i = K + \lceil \log_{\psi} p_i \rceil$$

- Can also prove that for $\forall \psi < 4, K, \Delta$, we can always find p_1, p_2, \dots, p_n s.t. if prefix code with lengths $l_i \geq K + \lceil \log_{\psi} p_i \rceil$ exists, then

$$\sum_i l_i p_i - OPT > \Delta.$$

- \Rightarrow No Shannon-Coding type algorithm can guarantee an additive-error approximation for a balanced prefix code.



Outline

- Huffman Coding and Generalizations
- Previous Work & Background
- New Work
- A “Counterexample”
- Conclusion and Open Problems

Conclusion and Open Problems

- We saw how to use Shannon Coding to develop efficient approximation algorithms for prefix-coding variants, e.g., unequal cost cost coding, coding in the Discrete Noiseless Channel and coding with regular language constraints.

Conclusion and Open Problems

- We saw how to use Shannon Coding to develop efficient approximation algorithms for prefix-coding variants, e.g., unequal cost cost coding, coding in the Discrete Noiseless Channel and coding with regular language constraints.
- Old Open Question: “is unequal-cost coding NP-complete?”

Conclusion and Open Problems

- We saw how to use Shannon Coding to develop efficient approximation algorithms for prefix-coding variants, e.g., unequal cost cost coding, coding in the Discrete Noiseless Channel and coding with regular language constraints.
- Old Open Question: “is unequal-cost coding NP-complete?”
- New Open Question: “is there an additive-error approximation algorithm for prefix coding using balanced strings?”

We just saw that Shannon Coding doesn't work.

G. & Li (2007) proved that (variant of) Shannon-Fano doesn't work. Perhaps no such algorithm exists.



The End

THANK YOU

Q and A