Sublinear Algorithms for Approximating String Compressibility

Sofya Raskhodnikova

Penn State University

Joint work with

Dana Ron (Tel Aviv University), Ronitt Rubinfeld (MIT)

and Adam Smith (Penn State University)

Question

bla bla bla bla bla bla bla bla bla bla bla bla

compressible?

Question

bla bla bla bla bla bla bla bla bla bla bla bla

compressible?

Fundamental question in

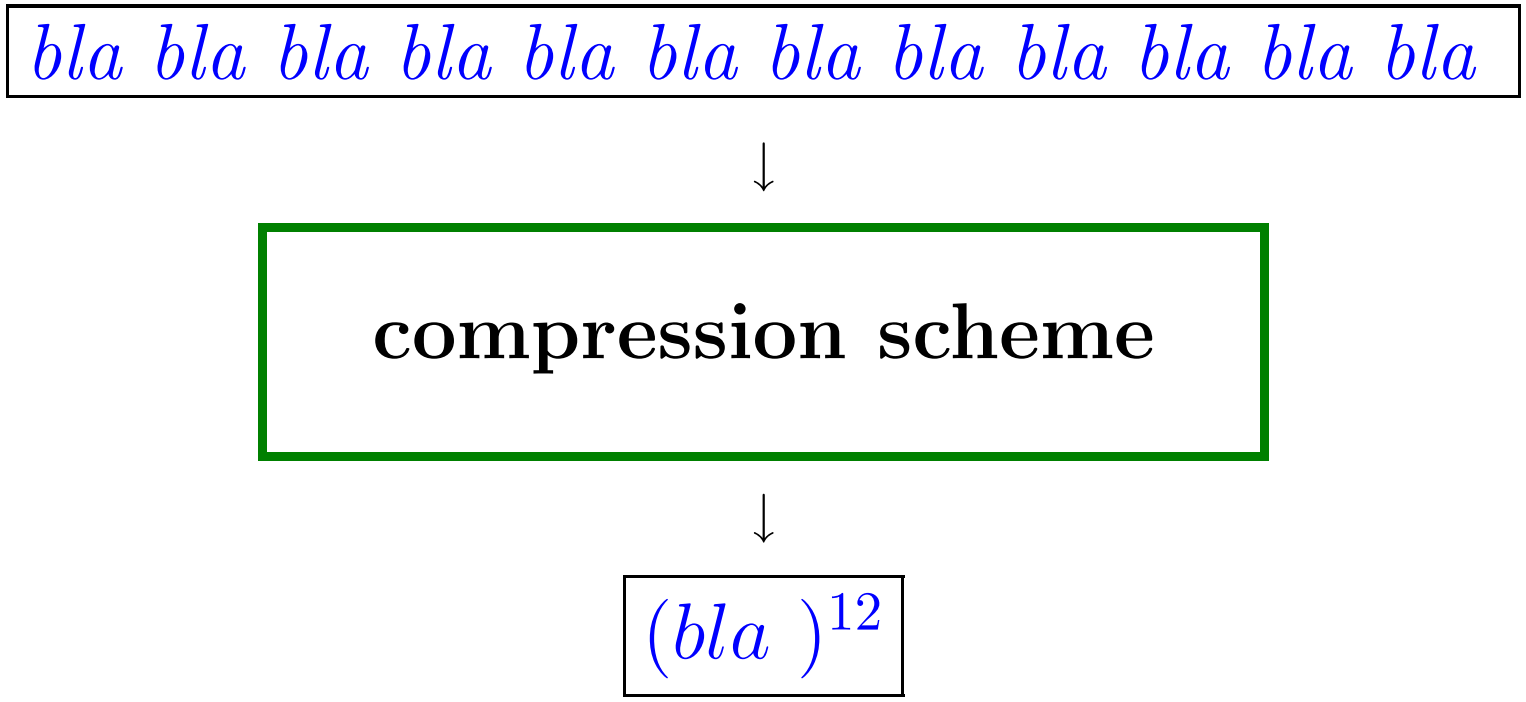
- communications
- storage
- complexity theory
- learning

Question

Can we determine (approximate) compressibility in sublinear time?

- In general (Kolmogorov complexity)
  - not computable
  - hard to approximate in poly time even with restrictions that make it computable
- We focus on specific compression schemes
  - Run-Length Encoding
  - Lempel-Ziv

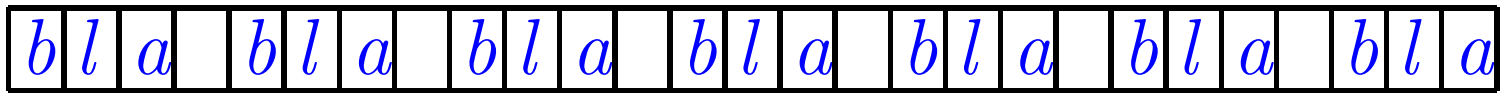
Approximating compressibility



- Compression algorithms take almost linear time.
- Can we tell how well a file compresses faster?

Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.

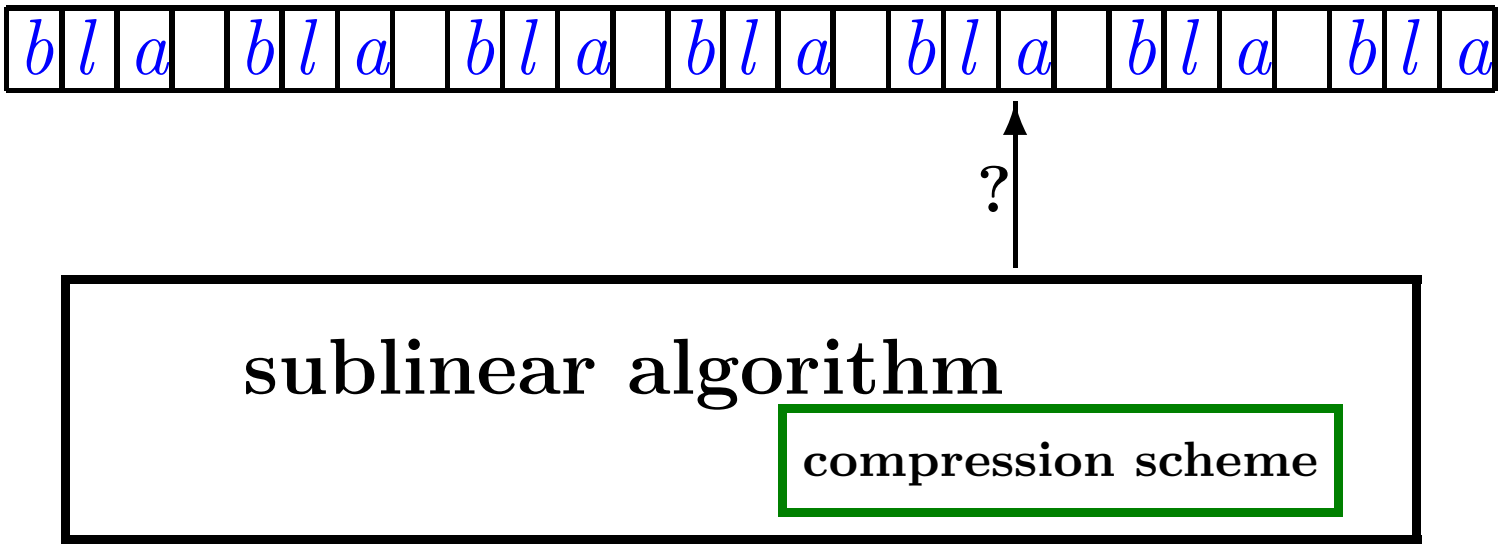


sublinear algorithm

compression scheme

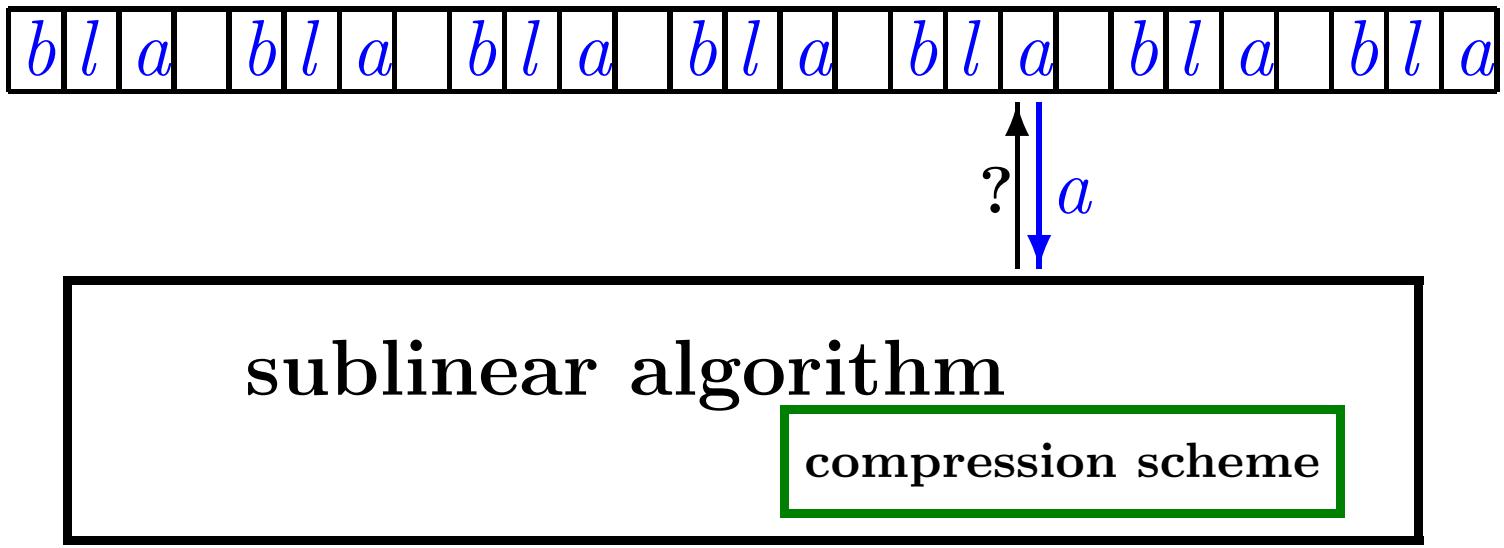
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



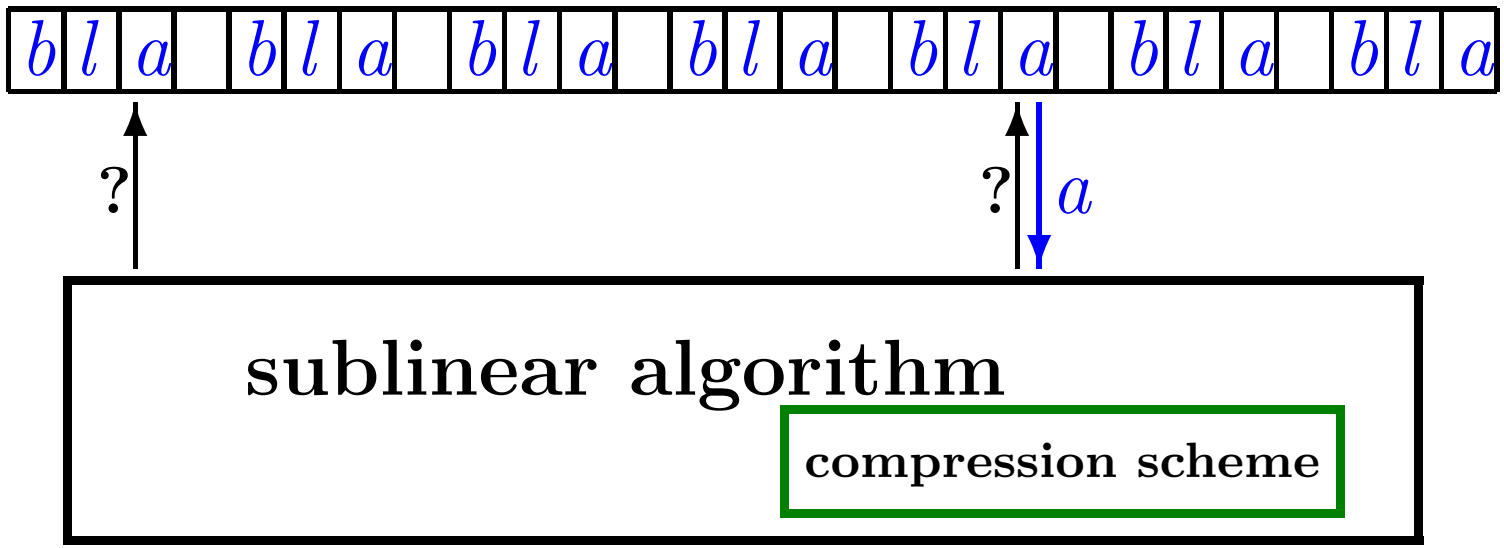
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



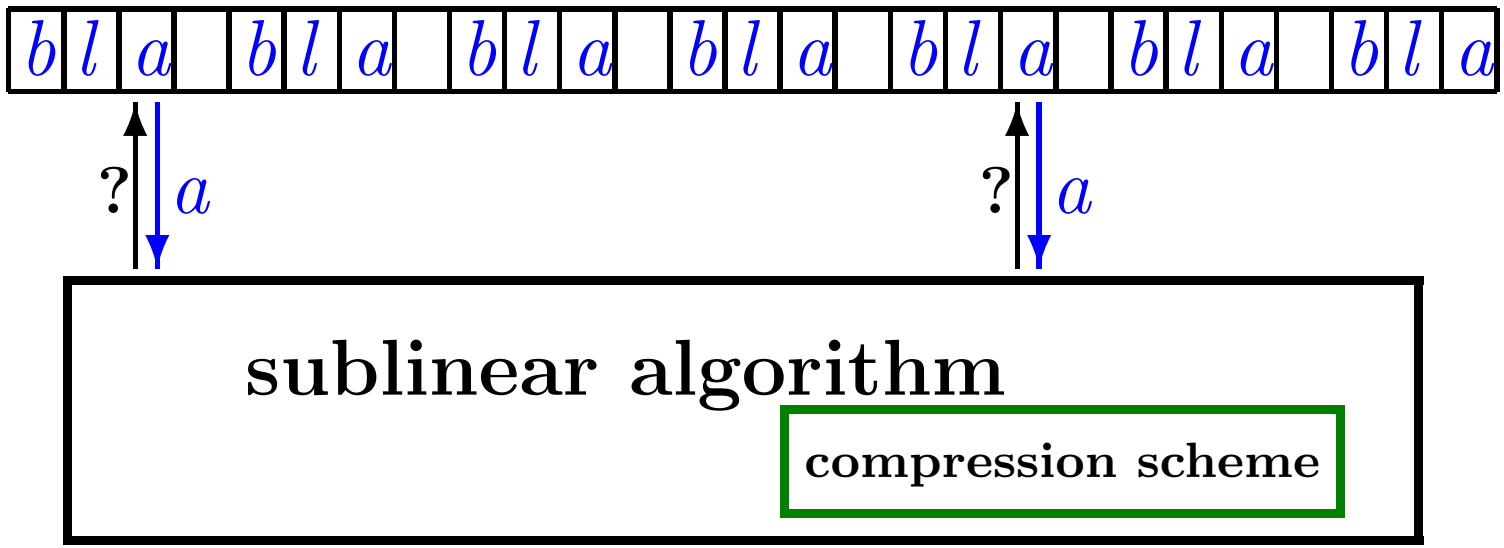
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



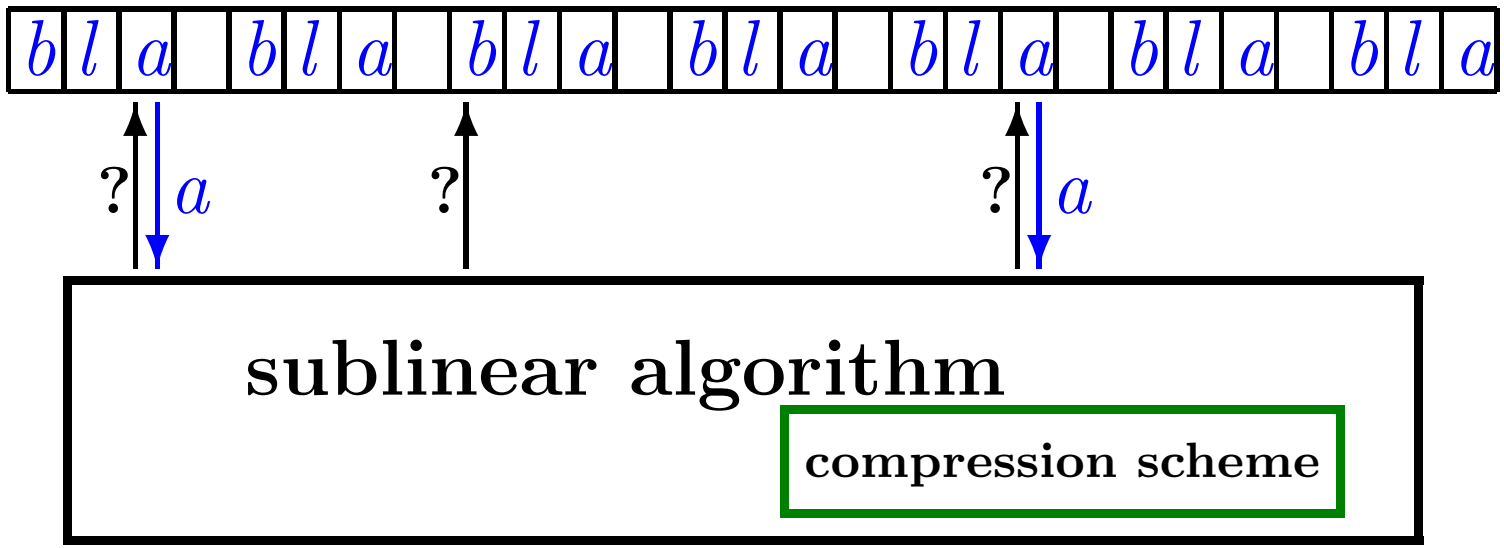
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



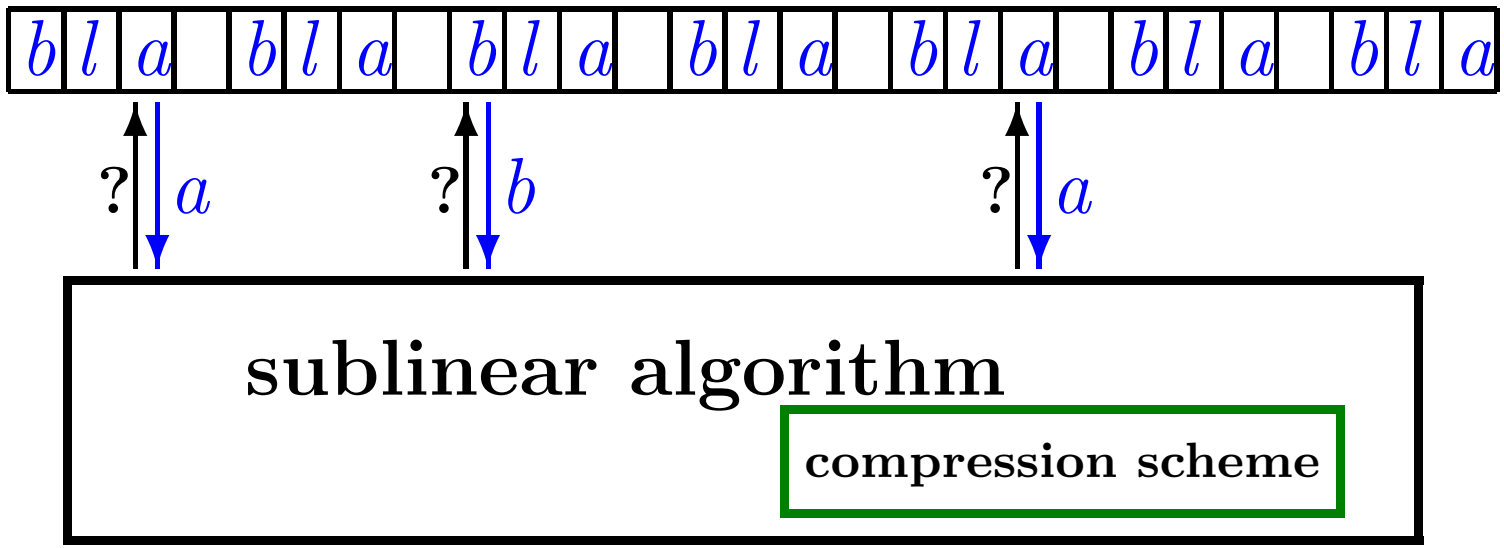
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



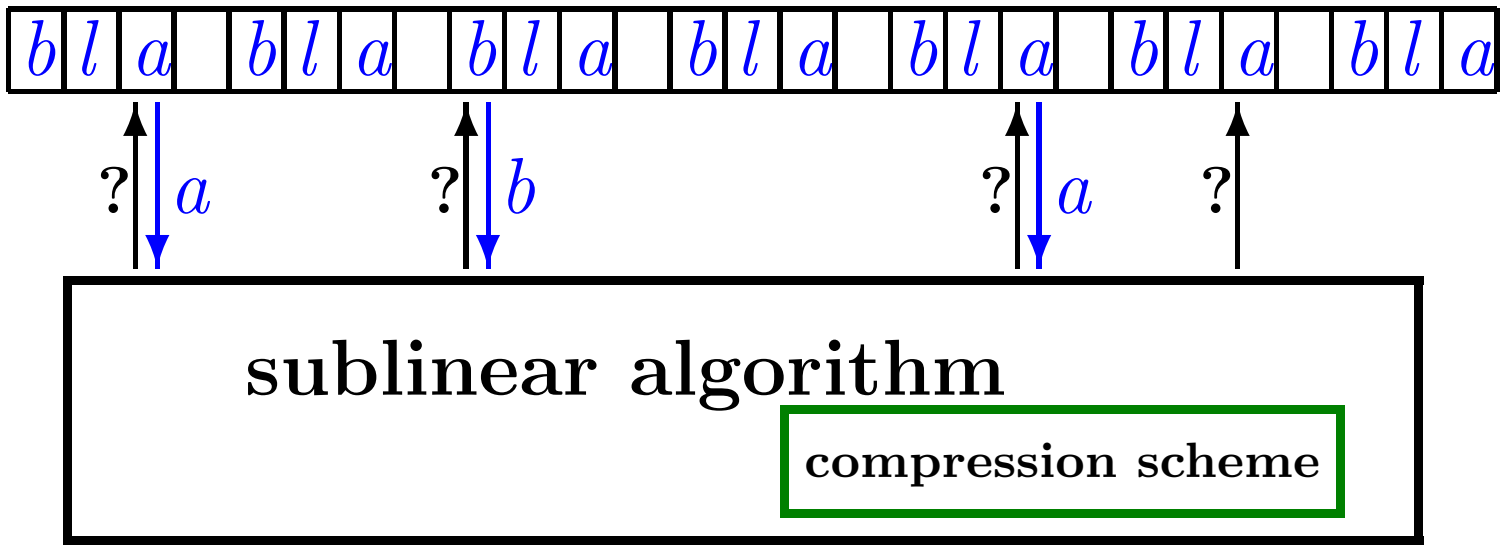
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



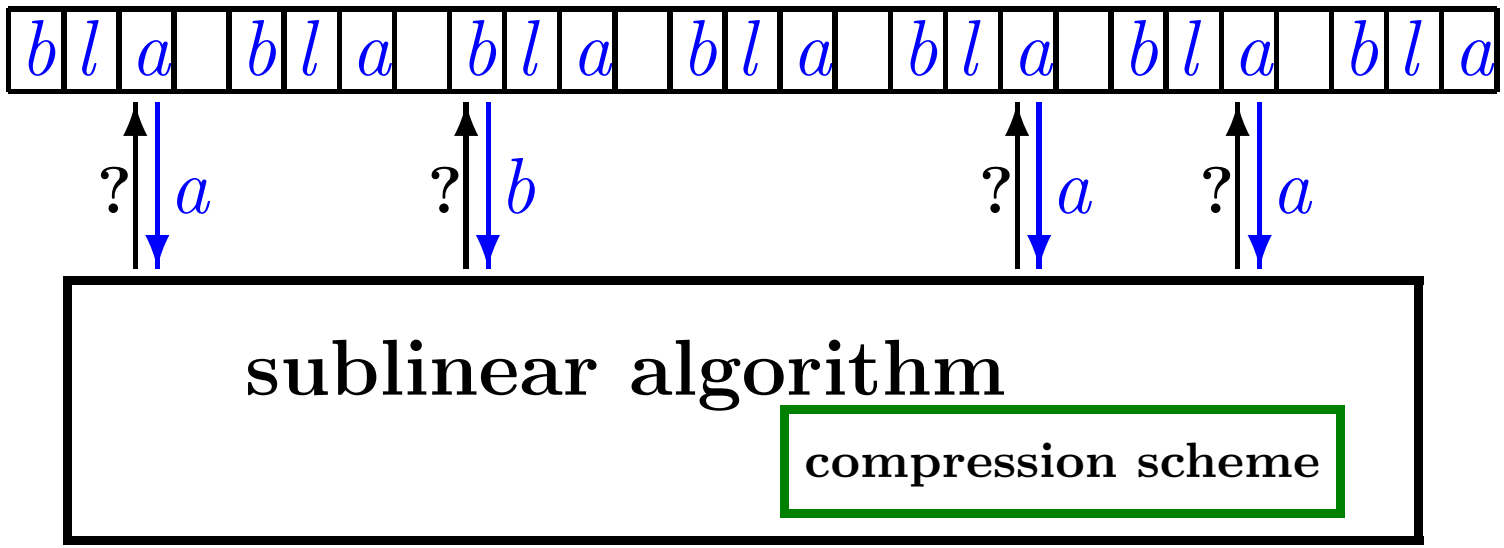
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



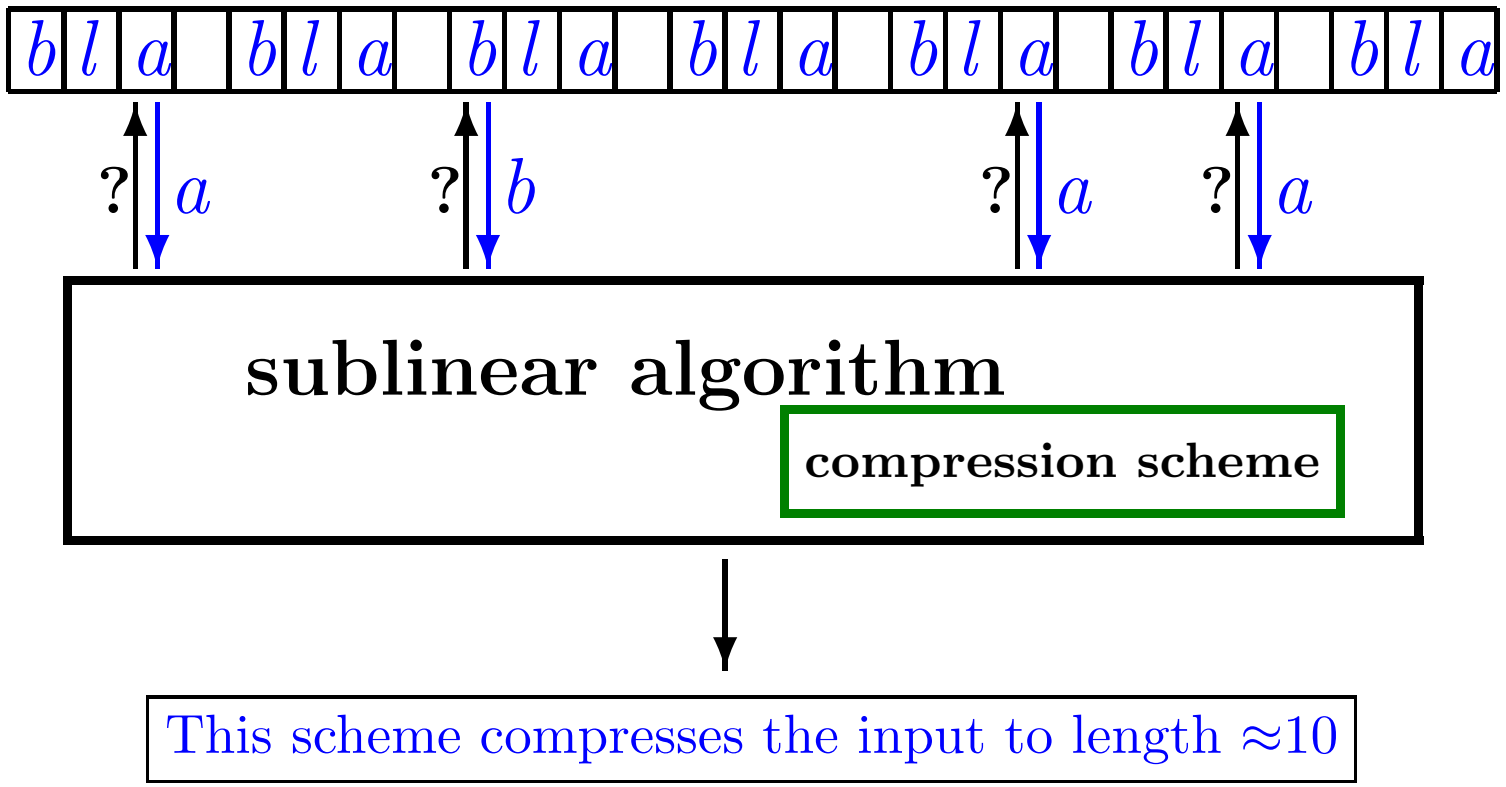
Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



Approximating compressibility

Problem: Approximate compressibility of a string w.r.t. a fixed compression scheme in *sublinear time*.



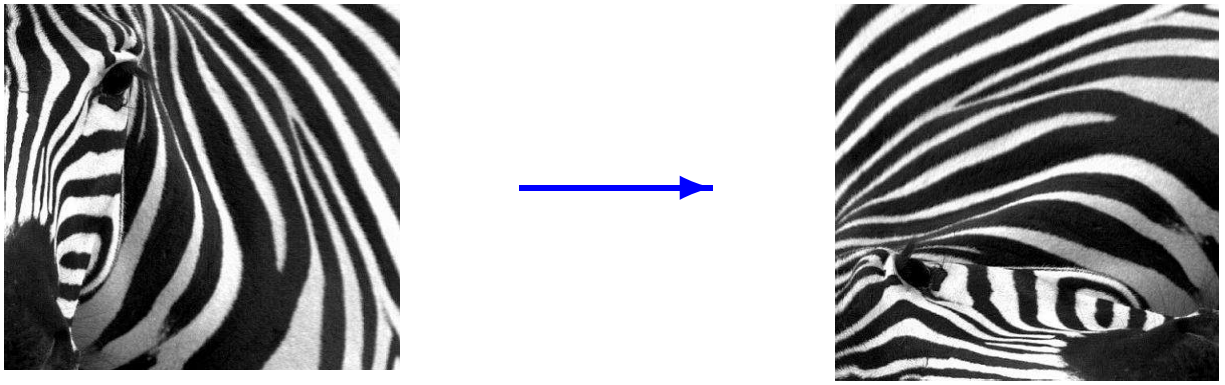
Motivation

- Given a large file, which scheme should one use to compress it?

Motivation

- Given a large file, which scheme should one use to compress it?

- Will a simple change to a file (e.g., a rotation of an image) make it more compressible?



Motivation

Compressibility is used to measure similarity/entropy in

- data mining [Li Chen Li Ma Vitányi 03, ...]
- natural language processing

  [Kukushkina Polikarpov Khmelev 00, ...]
- genomics [Loewenstern Hirsh Noordewier Yianilos 95, ...]

Need only the length of the compressed version of a file

Motivation

Compressibility is used to measure similarity/entropy in

- data mining [Li Chen Li Ma Vitányi 03, ...]
- natural language processing
  [Kukushkina Polikarpov Khmelev 00, ...]
- genomics [Loewenstern Hirsh Noordewier Yianilos 95, ...]

Need only the length of the compressed version of a file

Example: clustering t objects [Cilibrasi Vitáni 05]

- distance(x, y) = length of compressed concatenation
- run gzip $\Theta(t^2)$ times

Even a weak approximation algorithm would help!

Results overview

Compression schemes

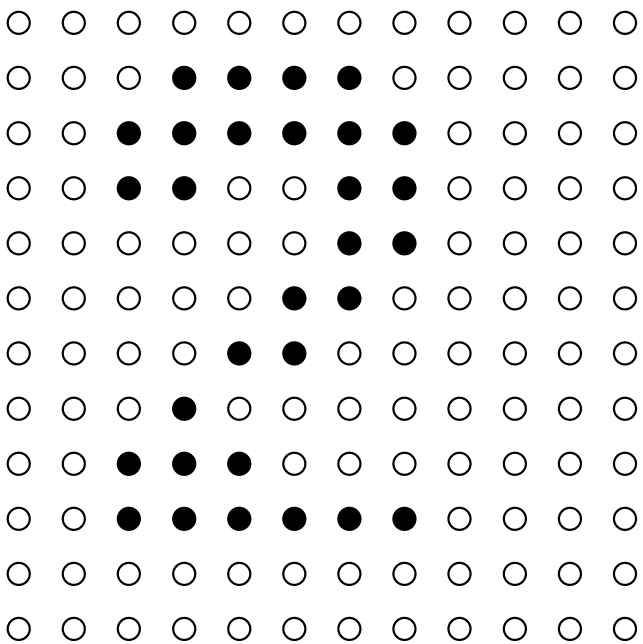
- Run-length encoding (RLE)
- Lempel-Ziv (LZ)

Error	RLE	LZ
A-Multiplicative	bad: $\Omega\left(\frac{n}{A^2}\right)$	bad: $\Omega\left(\frac{n}{A^2}\right)$
εn-Additive	excellent: $\tilde{O}\left(\frac{1}{\varepsilon^3}\right), \Omega\left(\frac{1}{\varepsilon^2}\right)$	bad: $n^{1-o(1)}$
Both	excellent: several algs.	non-trivial: $\tilde{O}\left(\frac{n}{A^3\varepsilon}\right)$

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



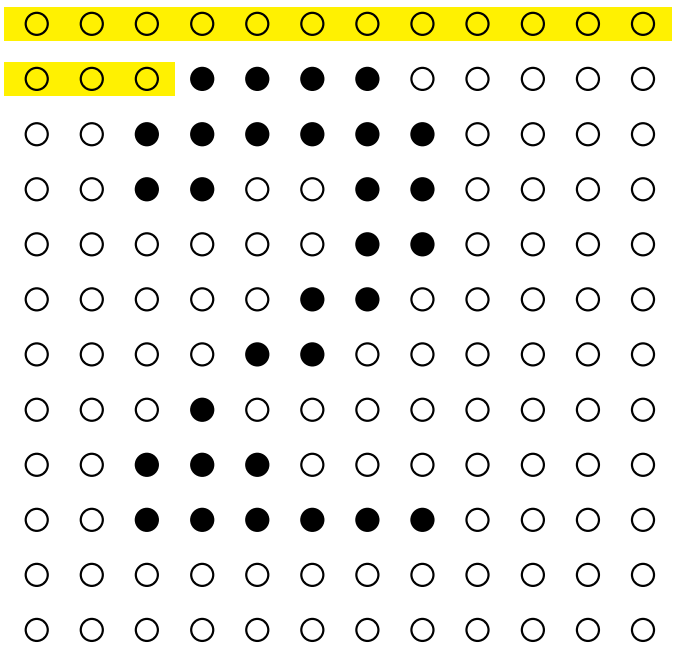
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



15,

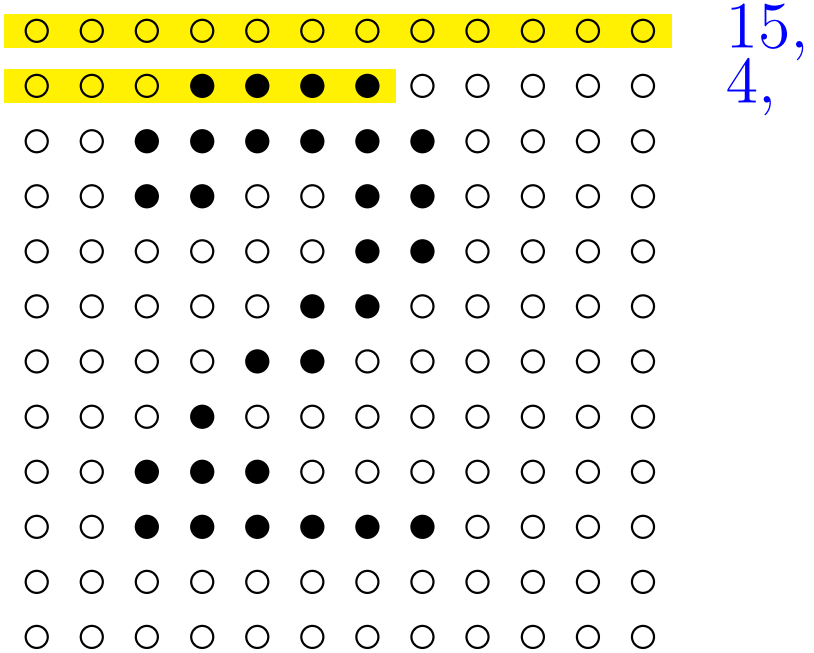
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



15,
4,

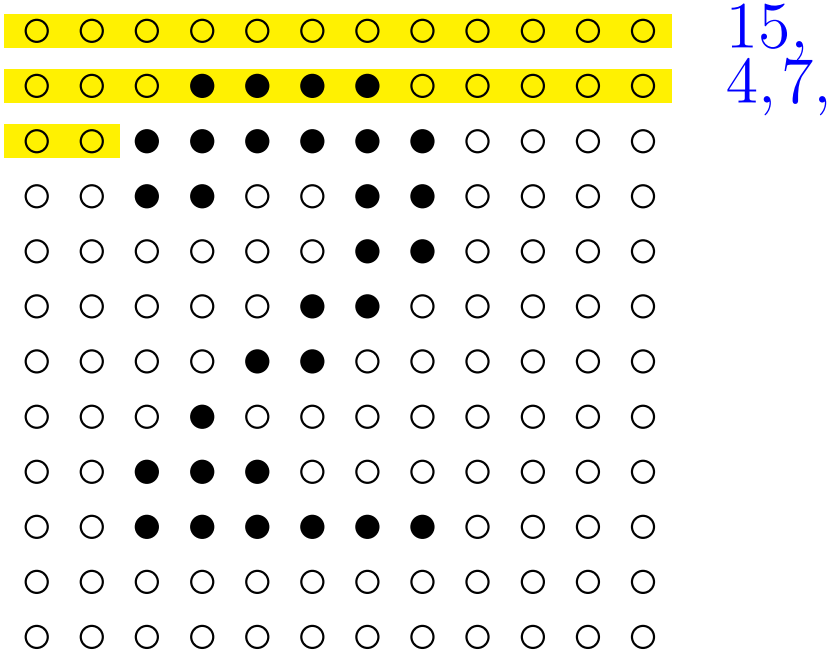
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



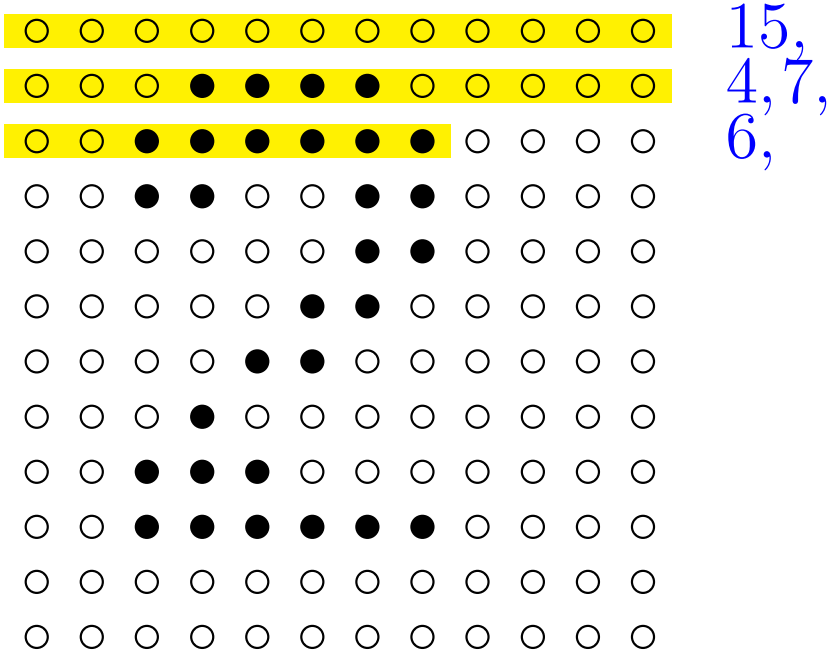
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



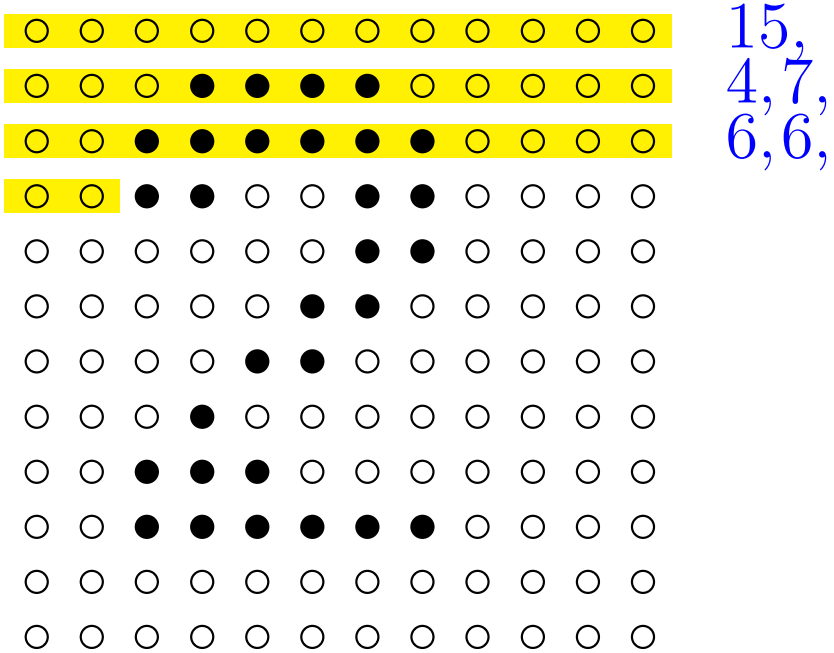
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



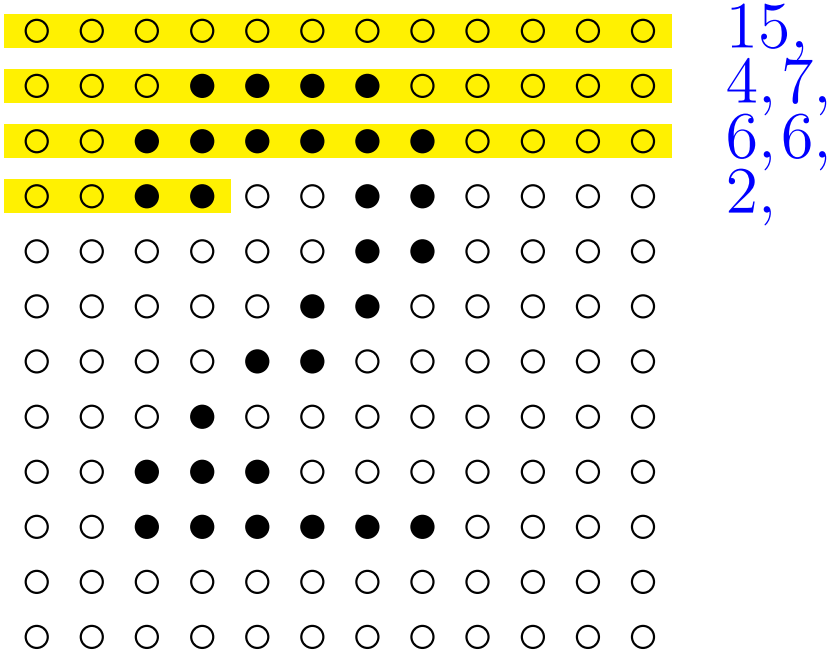
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



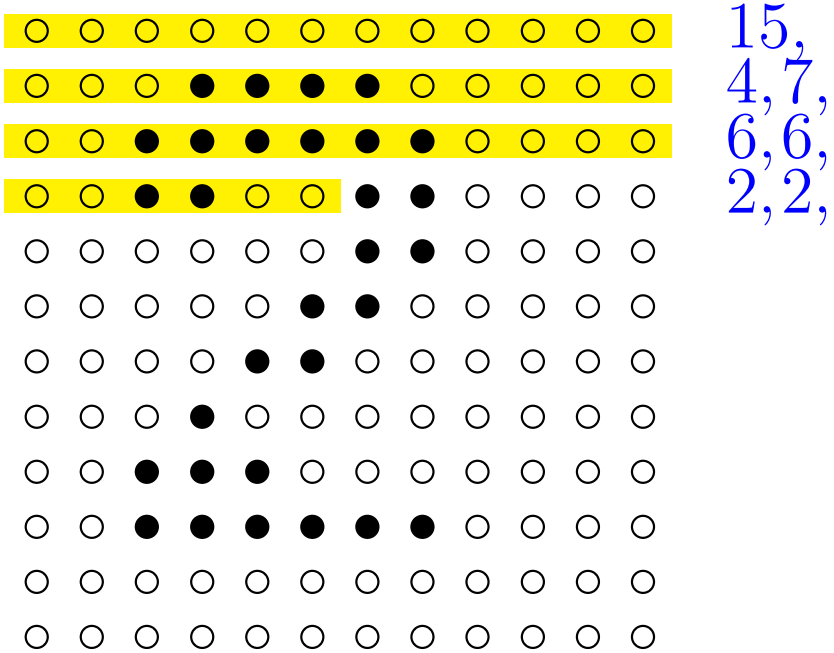
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



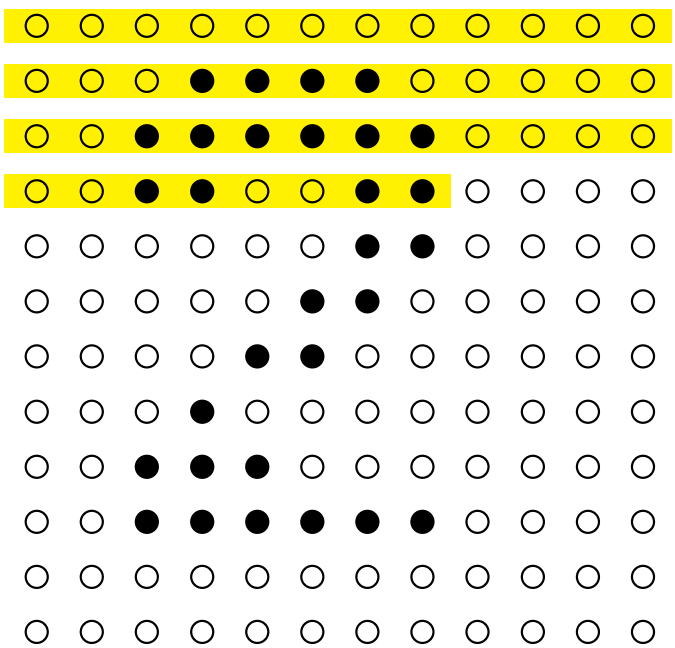
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding

Input: $\underbrace{000\ldots0}_{\ell_1}\underbrace{111\ldots1}_{\ell_2}\underbrace{000\ldots0}_{\ell_3}\underbrace{111\ldots1}_{\ell_4}\ldots$

Encoding: $\ell_1, \ell_2, \ell_3, \ell_4, \ldots$



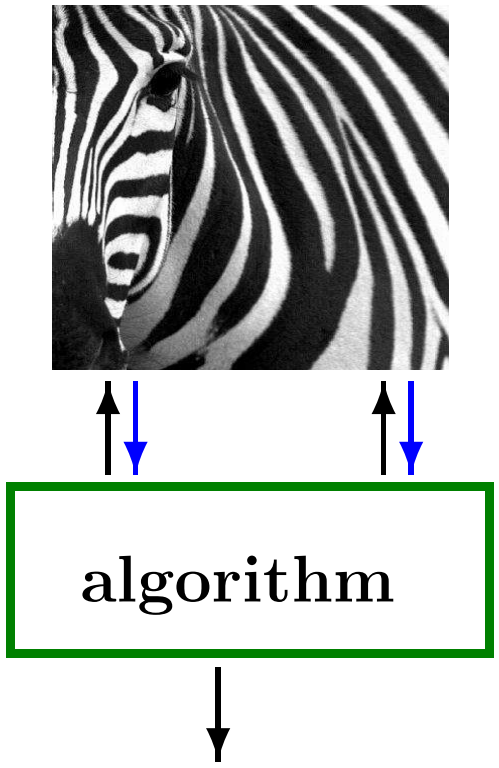
15,
4,7,
6,6,
2,2,2,

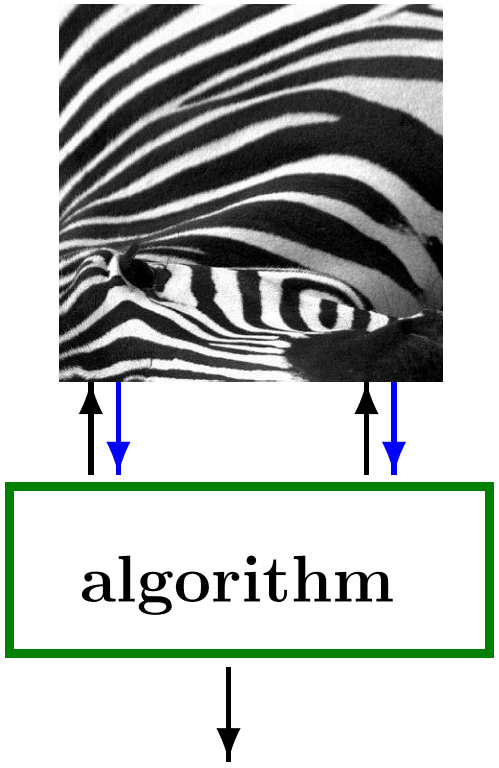
Encoding **cost**: $\sum_j \log \ell_j$

- Used for black-and-white images, faxes, icons

Run-length encoding: goal

Goal: Approximate RLE **cost**, in sublinear time.





RLE: multiplicative error

Claim

An A-multiplicative approximation of **cost**
requires $\Omega\left(\frac{n}{A^2}\right)$ *queries.*

Examples

constant approximation $\Longrightarrow$ linear number of queries

n^{α}-factor approximation $\Longrightarrow$ $\Omega(n^{1-2\alpha})$ queries

A "needle in a haystack" argument

Claim

An A-multiplicative approximation of **cost**
requires $\Omega\left(\frac{n}{A^2}\right)$ *queries.*

Intuition

Any algorithm has to make $\Omega\left(\frac{n}{A^2}\right)$ queries to distinguish

(a) 0^n

(b) 0^n with A^2 random bits flipped

A "needle in a haystack" argument

Claim

An A-multiplicative approximation of **cost**
requires $\Omega\left(\frac{n}{A^2}\right)$ *queries.*

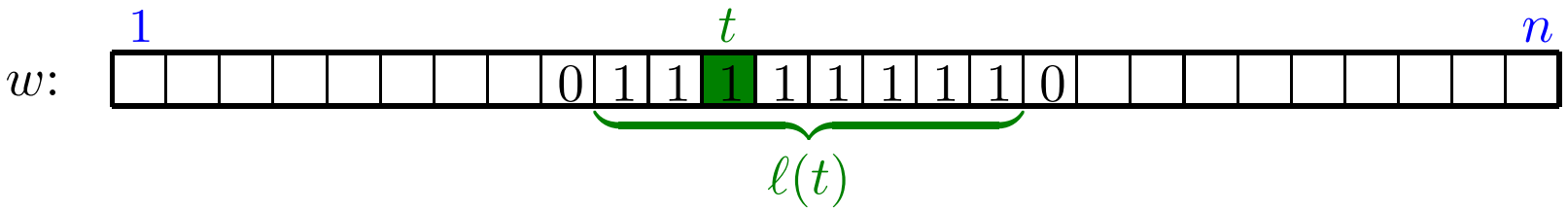
Intuition

Any algorithm has to make $\Omega\left(\frac{n}{A^2}\right)$ queries to distinguish

(a) 0^n (**cost** = $\log n$)

(b) 0^n with A^2 random bits flipped (**cost** = $A^2 \log n$)

RLE: additive approximation



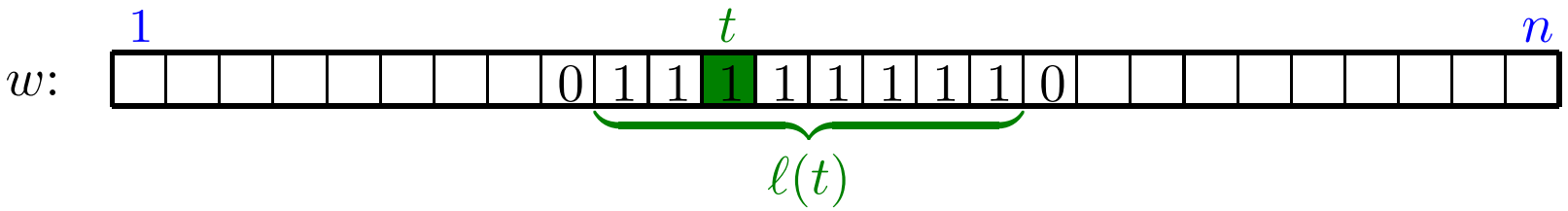
Contribution of position t is $c(t) = \frac{\log \ell(t)}{\ell(t)}$.

$$\textbf{cost} = \sum_{t=1}^{n} c(t) = n \cdot \text{Average}_t\, c(t)$$

Algorithm

1. *Sample positions at random.*
2. *Determine the length of the runs to which they belong.*
3. *For positions with $\ell(t) \geq \ell_0$ estimate $c(t)$ by 0.*
4. *Output $n \cdot$ (average contribution of samples).*

RLE: additive approximation



Contribution of position t is $c(t) = \frac{\log \ell(t)}{\ell(t)}$.

$$\textbf{cost} = \sum_{t=1}^{n} c(t) = n \cdot \text{Average}_t\, c(t)$$

Algorithm

1. *Sample* $\Theta\left(\frac{1}{\varepsilon^2}\right)$ *positions at random.*
2. *Determine the length of the runs to which they belong.*
3. *For positions with* $\ell(t) \geq \ell_0 = \Theta\left(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ *estimate* $c(t)$ *by 0.*
   Actual contribution is $\frac{\log \ell(t)}{\ell(t)} \leq \frac{\log \ell_0}{\ell_0} \leq \frac{\varepsilon}{2}$.
4. *Output* $n \cdot$ (*average contribution of samples*).

By Chernoff Bound, sampling introduces $\leq \frac{\varepsilon}{2}$ additive error.

Road map

Approximating compressibility w.r.t. fixed schemes

- Run-length encoding
  - A "needle in a haystack" lower bound
  - εn-additive approximation in time independent of n
- Lempel-Ziv
  - A multiplicative+additive approximation algorithm
  - A nearly linear lower bound for additive approximation

A variant of LZ77

Position in the string:	1	2	3	4	5	6	7	8	9	10	11	12	13
Original string:	a	b	a	b	b	a	b	c	b	a	b	b	a
Compressed sequence:													

Encoding **cost** is the number of symbols
(characters and pointer-length pairs) in the output.

- Variants used in many archivers (zip, zoo, arc, gzip)

A variant of LZ77

Position in the string:	1	2	3	4	5	6	7	8	9	10	11	12	13
Original string:	a	b	a	b	b	a	b	c	b	a	b	b	a
Compressed sequence:	a												

Encoding **cost** is the number of symbols
(characters and pointer-length pairs) in the output.

- Variants used in many archivers (zip, zoo, arc, gzip)

A variant of LZ77

Position in the string:	1	2	3	4	5	6	7	8	9	10	11	12	13
Original string:	a	b	a	b	b	a	b	c	b	a	b	b	a
Compressed sequence:	a	b											

Encoding **cost** is the number of symbols
(characters and pointer-length pairs) in the output.

- Variants used in many archivers (zip, zoo, arc, gzip)

A variant of LZ77

Position in the string:	1	2	3	4	5	6	7	8	9	10	11	12	13
Original string:	a	b	a	b	b	a	b	c	b	a	b	b	a
Compressed sequence:	a	b	$(1,2)$										

pointer length

Encoding **cost** is the number of symbols
(characters and pointer-length pairs) in the output.

- Variants used in many archivers (zip, zoo, arc, gzip)

A variant of LZ77

Position in the string:	1	2	3	4	5	6	7	8	9	10	11	12	13
Original string:	a	b	a	b	b	a	b	c	b	a	b	b	a
Compressed sequence:	a	b	$(1,2)$		$(2,3)$								

pointer length

Encoding **cost** is the number of symbols
(characters and pointer-length pairs) in the output.

- Variants used in many archivers (zip, zoo, arc, gzip)

A variant of LZ77

Position in the string:	1	2	3	4	5	6	7	8	9	10	11	12	13
Original string:	a	b	a	b	b	a	b	c	b	a	b	b	a
Compressed sequence:	a	b	$(1,2)$			$(2,3)$		c					

pointer length

Encoding **cost** is the number of symbols
(characters and pointer-length pairs) in the output.

- Variants used in many archivers (zip, zoo, arc, gzip)

A variant of LZ77

Position in the string:	1	2	3	4	5	6	7	8	9	10	11	12	13
Original string:	a	b	a	b	b	a	b	c	b	a	b	b	a
Compressed sequence:	a	b	$(1,2)$		$(2,3)$			c			$(2,5)$		

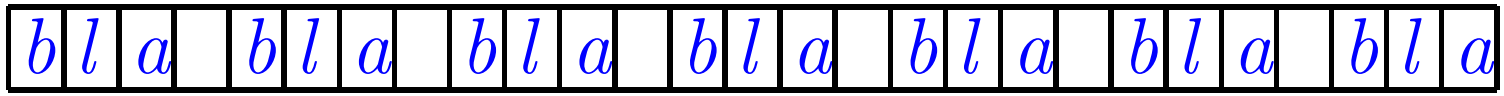
pointer length

Encoding **cost** is the number of symbols
(characters and pointer-length pairs) in the output.

- Variants used in many archivers (zip, zoo, arc, gzip)

Main idea for approximating LZ77 cost

compressible $\Longleftrightarrow$ few distinct substrings



4 distinct substrings of length 1

4 distinct substrings of length 2

$\vdots$

LZ77 cost *vs. # of distinct substrings*

Structural lemmas

1. *If* **cost** $\leq m$ *then*

   for all ℓ, *# of distinct length-*ℓ *substrings is* $\leq \ell m$.

2. *(Converse) If for all* ℓ

   # of distinct length-ℓ *substrings is* $\leq \ell m$ *then*

   $$\textbf{cost} \leq 4m \log n$$

[Lehman Shelat 02]: a lemma for grammars similar to 1.

LZ77 cost *vs. # of distinct substrings*

Structural lemmas

1. *If* **cost** $\leq m$ *then*

   for all ℓ, *# of distinct length-*ℓ *substrings is* $\leq \ell m$.

2. *(Converse) If for all* ℓ

   # of distinct length-ℓ *substrings is* $\leq \ell m$ *then*

   $$\textbf{cost} \leq 4m \log n$$

Estimate # of distinct length-ℓ substrings

[Lehman Shelat 02]: a lemma for grammars similar to 1.

LZ77 cost *vs. # of distinct substrings*

Structural lemmas

1. *If* **cost** $\leq m$ *then*

   for all ℓ, *# of distinct length-*ℓ *substrings is* $\leq \ell m$.

2. *(Converse) If for all* ℓ

   # of distinct length-ℓ *substrings is* $\leq \ell m$ *then*

   $$\textbf{cost} \leq 4m \log n$$

Estimate # of distinct length-ℓ substrings ($\ell = 1, \ldots, \ell_0$).

[Lehman Shelat 02]: a lemma for grammars similar to 1.

LZ77 cost *vs. # of distinct substrings*

Structural lemmas

1. If **cost** $\leq m$ *then*

for all ℓ*, # of distinct length-*ℓ *substrings is* $\leq \ell m$*.*

2.′ (Converse) If for all ℓ *up to* ℓ_0*,*

# of distinct length-ℓ *substrings is* $\leq \ell m$ *then*

cost $\leq$ ~~$4m \log n$~~ $4(m \log \ell_0 + \frac{n}{\ell_0})$.

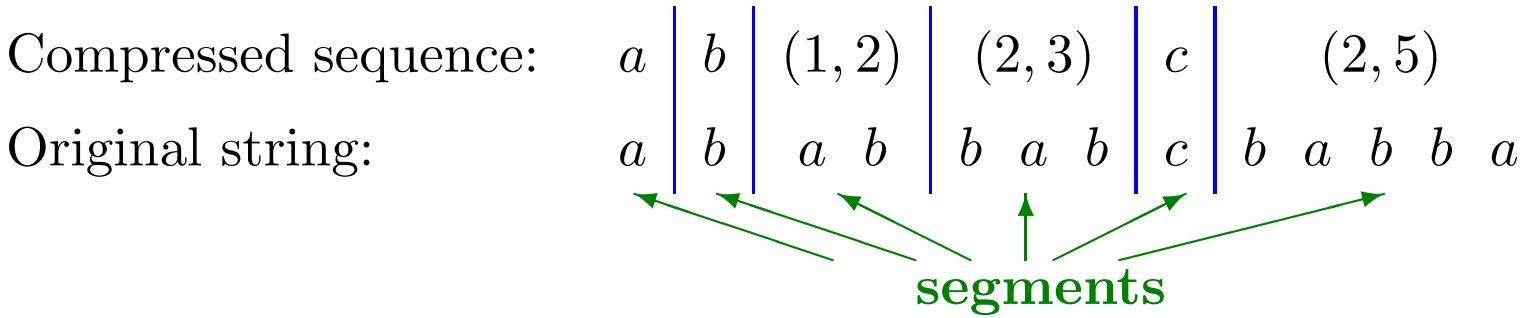
Estimate # of distinct length-ℓ substrings ($\ell = 1, \ldots, \ell_0$).

[Lehman Shelat 02]: a lemma for grammars similar to 1.

Proof of lemma 1

Lemma 1

If **cost** $\leq m$ *then for all* ℓ, *# of distinct length-*ℓ *substrings is* $\leq \ell m$.



Call a substring **new** if it hasn't appeared before.

$$\#\text{ distinct substrings} = \#\ \textbf{new}\ \text{substrings}$$

Key observation: substrings within segments are **old**.

\# of distinct length-ℓ substrings $\leq$
\# of length-ℓ substring that start and end in different segments $\leq$ $(\ell - 1)(m - 1)$.

Proof of lemma $2'$

Lemma 2′

If for all ℓ *up to* ℓ_0*, # of distinct length-*ℓ *substrings is* $\leq \ell m$ *then* **cost** $\leq 4(m \log \ell_0 + \frac{n}{\ell_0})$.

- Segments longer than ℓ_0 contribute at most $\frac{n}{\ell_0}$ to **cost**.
- Short segments add $\sum_{i=1}^{\ell_0} r_i$, where $r_i = \#$ of length-i segments.
- Enough to bound the sum.

Proof of lemma $2'$

Lemma 2′

If for all ℓ *up to* ℓ_0*, # of distinct length-*ℓ *substrings is* $\leq \ell m$ *then* **cost** $\leq 4(m \log \ell_0 + \frac{n}{\ell_0})$.

- Segments longer than ℓ_0 contribute at most $\frac{n}{\ell_0}$ to **cost**.
- Short segments add $\sum_{i=1}^{\ell_0} r_i$, where r_i = # of length-i segments.
- Enough to bound the sum.

Attempt 1: length-i **segment**

$\downarrow$

length-$(i+1)$ substring starting with the **segment**

Proof of lemma $2'$

Lemma 2′

If for all ℓ *up to* ℓ_0*, # of distinct length-*ℓ *substrings is* $\leq \ell m$ *then* **cost** $\leq 4(m \log \ell_0 + \frac{n}{\ell_0})$.

- Segments longer than ℓ_0 contribute at most $\frac{n}{\ell_0}$ to **cost**.
- Short segments add $\sum_{i=1}^{\ell_0} r_i$, where r_i = # of length-i segments.
- Enough to bound the sum.

Attempt 1: length-i **segment**

$\downarrow$

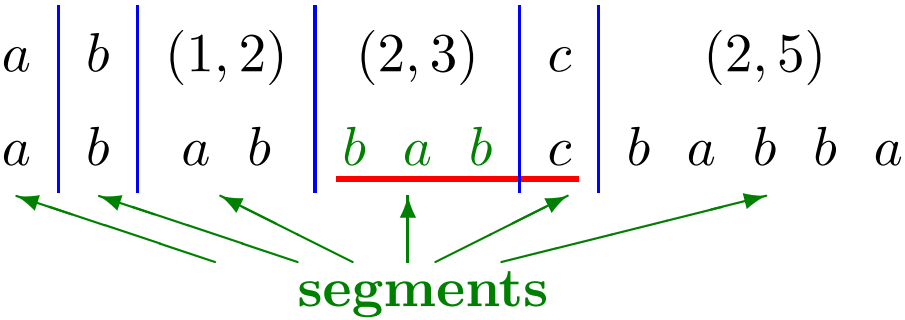
new length-$(i+1)$ substring starting with the **segment**

Proof of lemma $2'$

Lemma 2′

If for all ℓ *up to* ℓ_0*, # of distinct length-*ℓ *substrings is* $\leq \ell m$ *then* **cost** $\leq 4(m \log \ell_0 + \frac{n}{\ell_0})$.

- Segments longer than ℓ_0 contribute at most $\frac{n}{\ell_0}$ to **cost**.
- Short segments add $\sum_{i=1}^{\ell_0} r_i$, where r_i = # of length-i segments.
- Enough to bound the sum.



Attempt 1: length-i **segment**

↓

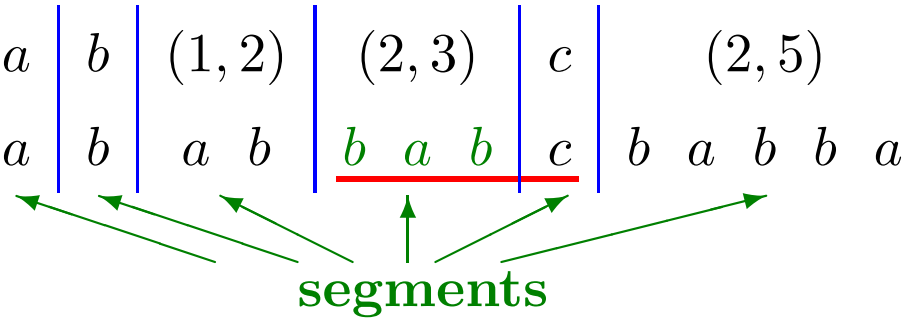
new length-(i+1) substring starting with the **segment**

Proof of lemma $2'$

Lemma 2′

If for all ℓ *up to* ℓ_0*, # of distinct length-*ℓ *substrings is* $\leq \ell m$ *then* **cost** $\leq 4(m \log \ell_0 + \frac{n}{\ell_0})$.

- Segments longer than ℓ_0 contribute at most $\frac{n}{\ell_0}$ to **cost**.
- Short segments add $\sum_{i=1}^{\ell_0} r_i$, where r_i = # of length-i segments.
- Enough to bound the sum.



Attempt 1: length-i **segment**

↓

new length-$(i+1)$ substring starting with the **segment**

$$r_i \leq \#\text{ of distinct length-}(i+1)\text{ substrings} \leq (i+1)m$$

$$\sum_{i=1}^{\ell_0} r_i \leq m \sum_{i=1}^{\ell_0} (i+1) = O\left(m\ell_0^2\right)$$

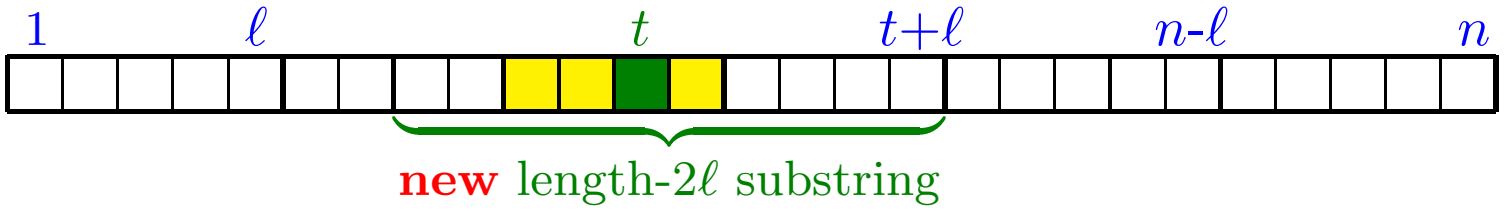
Real proof of lemma $2'$

A better bound, $\sum_{i=1}^{\ell_0} r_i = O(m \log \ell_0)$, follows from:

Claim

$$\sum_{i=1}^{\ell} i \cdot r_i \leq 2\ell(m+1).$$

I.e., # of positions in segments of length up to ℓ *is* $\leq 2\ell(m+1)$.



position t in a segment of size up to ℓ

$\downarrow$

length-2ℓ substring that ends at position $t + \ell$

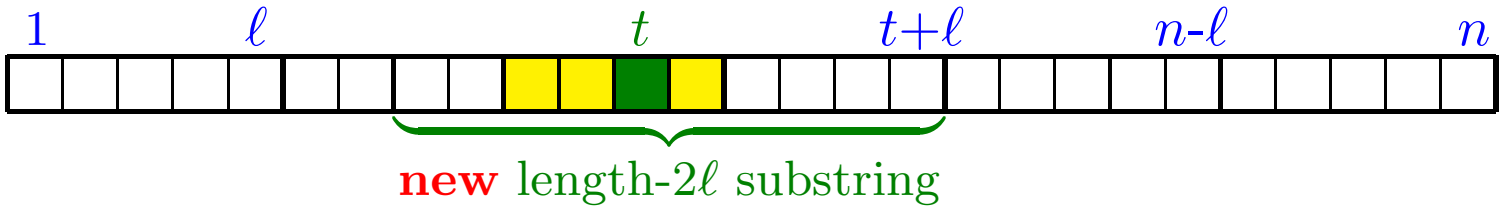
Real proof of lemma $2'$

A better bound, $\sum_{i=1}^{\ell_0} r_i = O(m \log \ell_0)$, follows from:

Claim

$$\sum_{i=1}^{\ell} i \cdot r_i \leq 2\ell(m+1).$$

I.e., # of positions in segments of length up to ℓ *is* $\leq 2\ell(m+1)$.



position t in a segment of size up to ℓ

$\downarrow$

new length-2ℓ substring that ends at position $t + \ell$

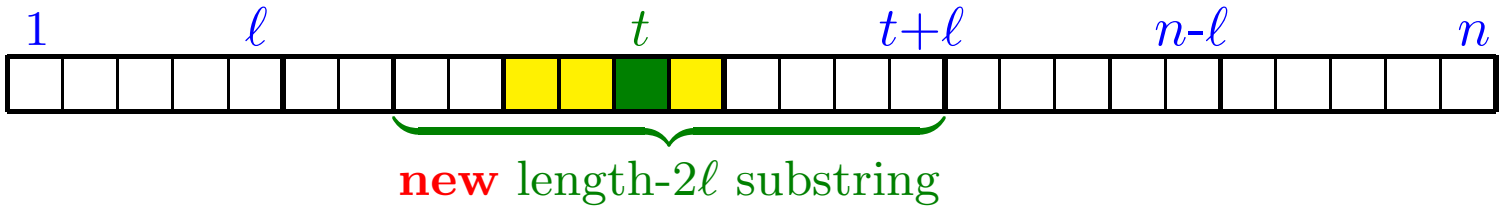
Real proof of lemma $2'$

A better bound, $\sum_{i=1}^{\ell_0} r_i = O(m \log \ell_0)$, follows from:

Claim

$$\sum_{i=1}^{\ell} i \cdot r_i \leq 2\ell(m+1).$$

I.e., # of positions in segments of length up to ℓ *is* $\leq 2\ell(m+1)$.



position t in a segment of size up to ℓ

$\downarrow$

new length-2ℓ substring that ends at position $t + \ell$

There are $\leq 2\ell m$ distinct length-2ℓ substrings.

Approximating LZ77 cost

Summary of structural lemmas

compressible

$\Updownarrow$

few distinct substrings of length ℓ $(\ell = 1, \ldots, \ell_0)$

Algorithm for approximating LZ77 **cost**

1. Estimate # of distinct length-ℓ substrings $(\ell = 1, \ldots, \ell_0)$.
2. Combine the estimates to get an estimate for **cost**.

Need: A procedure that estimates # of distinct length-ℓ substrings.

Counting distinct substrings

Distinct Elements Problem

Given n colored balls, approximate the number of distinct colors.

Or: Approximate the support size of a distribution where each element appears with probability $\geq \frac{1}{n}$.

Counting distinct substrings

Distinct Elements Problem

Given n colored balls, approximate the number of distinct colors.

Or: Approximate the support size of a distribution where each element appears with probability $\geq \frac{1}{n}$.

- In databases Approximating distinct values.
  [Charikar Chaudhuri Motwani Narasayya 00, Olken Rotem 95]
- In statistics Estimating the number of species in a population.
  822 references on www.stat.cornell.edu/∼bunge/bibliography.htm
- In streaming Approximating the frequency moment F_0.
  [Alon Matias Szegedy 99, Bar-Yossef Kumar Sivakumar 01]
  Counting distinct elements.
  [Bar-Yossef Jayram Kumar Sivakumar Trevisan 02, ...]

A-multiplicative approximation

Distinct Elements

[Charikar Chaudhuri Motwani Narasayya 00]

- $O(\frac{n}{A^2})$ time algorithm
- A matching "needle in a haystack" lower bound

With structural lemmas, for LZ77 **cost** it implies:

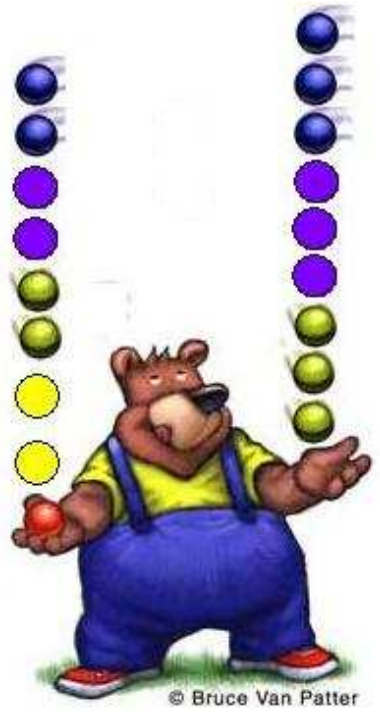
- A-multiplicative + εn-additive error in $\tilde{O}\left(\frac{n}{A^2\varepsilon}\right)$ time
  - can distinguish strings with **cost** $O(n^\alpha)$ from strings with **cost** $\Omega(n)$ in $\tilde{O}(n^\alpha)$ time.

Additive approximation

Distinct Elements

[Raskhodnikova Ron Shpilka Smith 07]

- $n^{1-o(1)}$ queries required



We give a reduction from Distinct Elements to LZ77.

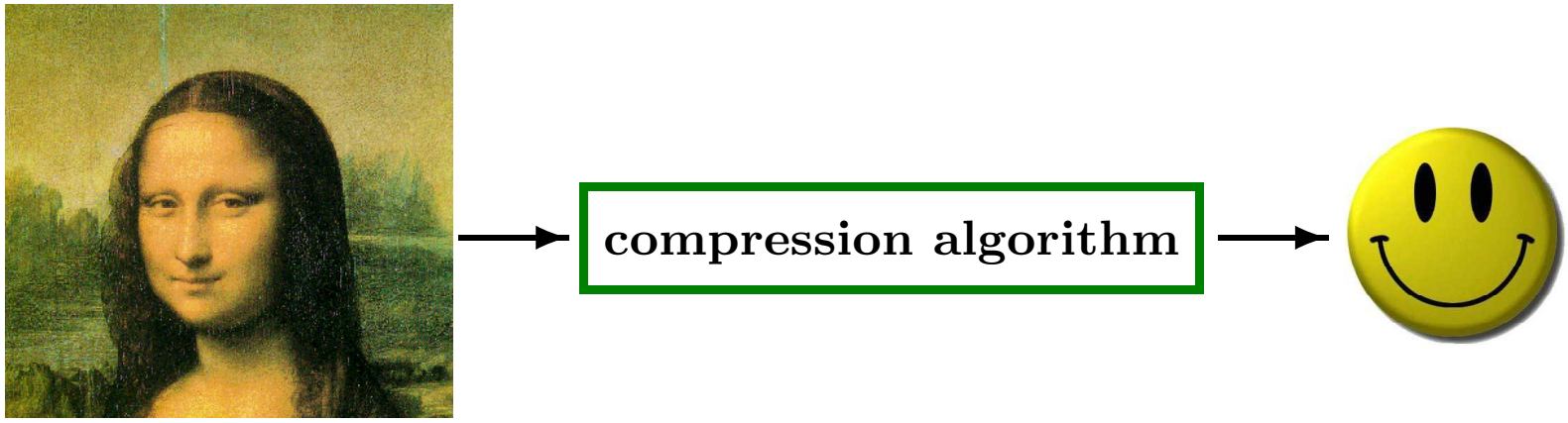
- The same lower bound holds for LZ77 **cost**.

Compressibility: summary

- Run-length encoding
  - additive approximation in constant time
- Lempel-Ziv
  - multiplicative+additive approximation
  - no sublinear additive approximation
- Huffman encoding
  - compression cost is equal to entropy
  - approximating entropy has been studied
    [Batu Dasgupta Kumar Rubinfeld 05,
    Brautbar Samorodnitsky 07]

Main open question

- approximating compressibility for other compression schemes?

Main open question

- approximating compressibility for other compression schemes?
  - for lossy compression schemes?