

Text Indexing, Suffix Sorting & Data Compression: Common Problems and Techniques

Roberto Grossi

Dipartimento di Informatica

Università di Pisa

Roadmap

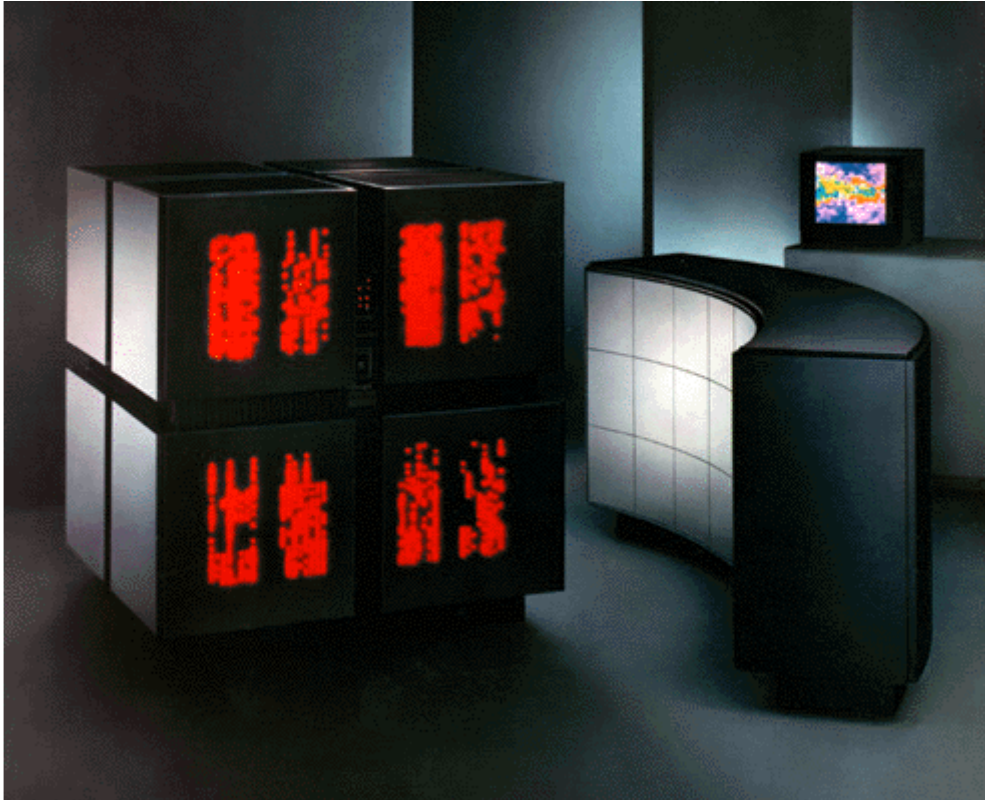
Space efficiency issues

Short stories:

- suffix array permutations
 - succinct data structures
 - compressed text indexing

Goal: give flavor, no technicalities

My first space issues...

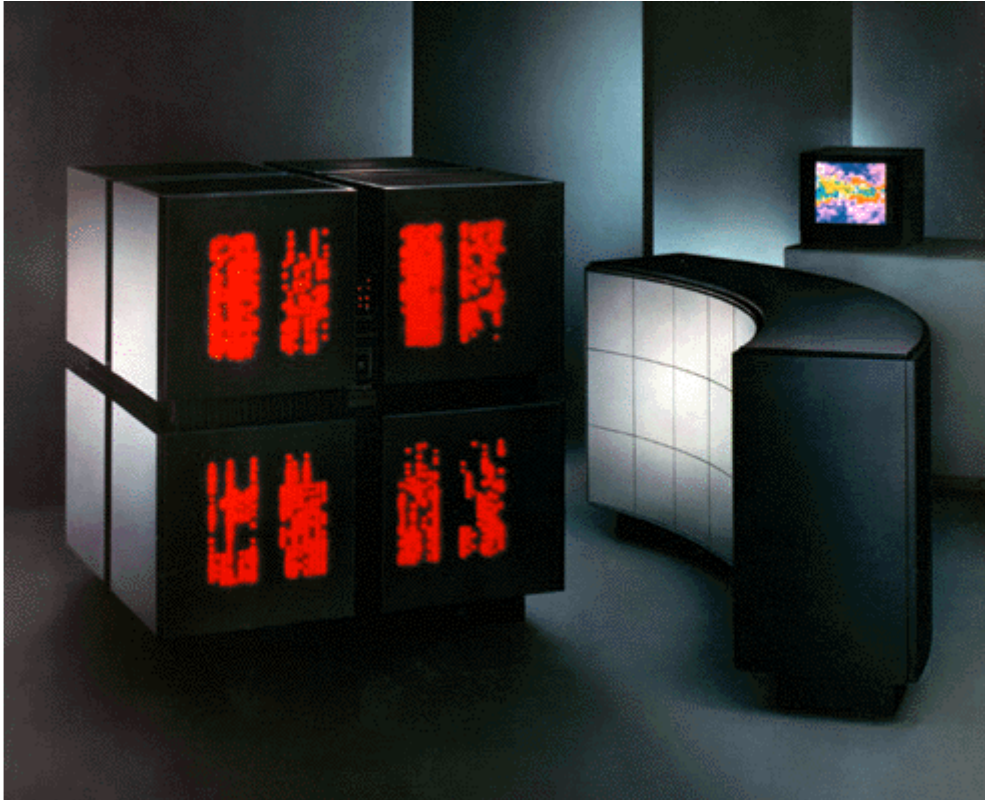


Connection Machine CM-2

- 4096 single-bit processors
- hypercube layout
- 512 MB RAM
- 25 GB Data Vault (RAID)
- cost ~ 500Keuro in 1990

AILS_V parallel
algorithm for
suffix tree
construction in
 $O(\log n)$ time

My first space issues...



Connection Machine CM-2

- 4096 single-bit processors
- hypercube layout
- 512 MB RAM
- 25 GB Data Vault (RAID)
- cost ~ 500Keuro in 1990

max text length:
just $n = 256$ chars ...

AILSV parallel
algorithm for
suffix tree
construction in
 $O(\log n)$ time

Few years later at AT&T Bell Labs

Visiting Ken Church (the “gigabyte” guy) with Paolo Ferragina



AMEX phone bill

- **1 CD** as bill couldn't be printed!
- **12 CD** for full text indexing...

Q: Just algorithm engineering
or some theory behind that?

Situation in the 90s...

zgrep: searching on compressed files
but needed to decompress them on the fly

New trend: keep the file in compressed format!

Amir, Benson, Farach '94: text scanning algorithm

Karkkainen, Sutinen, Ukkonen '96: text indexing algorithms

...more and more papers after the above ones

NOW: compressed text indexing (text + index)

~ space (**bzip**|**gzip**) text alone

Google Scholar search “suffix array”

Web Images Video News Maps more »

Google scholar Search [Advanced Scholar Search](#) [Scholar Preferences](#) [Scholar Help](#)

Scholar All articles - [Recent articles](#) Results 1 - 10 of about 34,200 for [suffix array](#). (0.12 seconds)

[Suffix arrays: A new method for on-line string searches](#) - [webglimpse.net](#) [PDF]

U Manber, G Myers - [Proceedings of the first annual ACM-SIAM symposium on ...](#), 1990 - [portal.acm.org](#)
... list of all the **suffixes** of A. When coupled with information about the longest common prefixes (lcp) of adjacent elements in the **suffix array**, string searches ...
[Cited by 934](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 51 versions](#)

} original paper

[Compressed suffix arrays and suffix trees with applications to text indexing and string matching \(...](#) - [psu.edu](#) [PDF]

R Grossi, JS Vitter - [Proceedings of the thirty-second annual ACM symposium on ...](#), 2000 - [portal.acm.org](#)
... The **suffix array** stores the pointers to the n **suffixes** in lexicographic order. ... In this paper we refer to the **suffix array** as the plain **array** of pointers. ...
[Cited by 245](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 21 versions](#)

}
←

[Linear work suffix array construction](#) - [psu.edu](#) [PDF]

J Kärkkäinen, P Sanders, S Burkhardt - 2006 - [portal.acm.org](#)
Page 1. Linear Work **Suffix Array** Construction JUHA KARKKAINEN ... time using $O(n/\sqrt{v})$ space in addition to the input string and the **suffix array**. ...
[Cited by 221](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 26 versions](#)

[Space efficient linear time construction of suffix arrays](#) - [psu.edu](#) [PDF]

P Ko, S Aluru - [Journal of Discrete Algorithms](#), 2005 - Elsevier
... The sorted order of **suffixes** of a string is also called **suffix array**, a data structure introduced by Manber and Myers that has numerous applications in pattern ...
[Cited by 164](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 16 versions](#)

} ← space issues

[Linear-time longest-common-prefix computation in suffix arrays and its applications](#) - [iastate.edu](#) [PDF]

T Kasai, G Lee, H Arimura, S Arikawa, K Park - [Lecture Notes in Computer Science](#), 2001 - Springer
... of block-sorting compression, and we present a linear-time algorithm to simulate the bottom-up traversal of a **suffix tree** with a **suffix array** combined with ...
[Cited by 135](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 9 versions](#)

[Linear-time construction of suffix arrays](#) - [psu.edu](#) [PDF]

DK Kim, JS Sim, H Park, K Park - [Lecture Notes in Computer Science](#), 2003 - Springer
Page 1. Linear-Time Construction of **Suffix Arrays** (Extended Abstract) ... The **suffix array** due to Manber and Myers [18] and independently due to Gonnet et al. ...
[Cited by 124](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 6 versions](#)

[Compressed text databases with efficient query algorithms based on the compressed suffix array](#) - [psu.edu](#) [PDF]

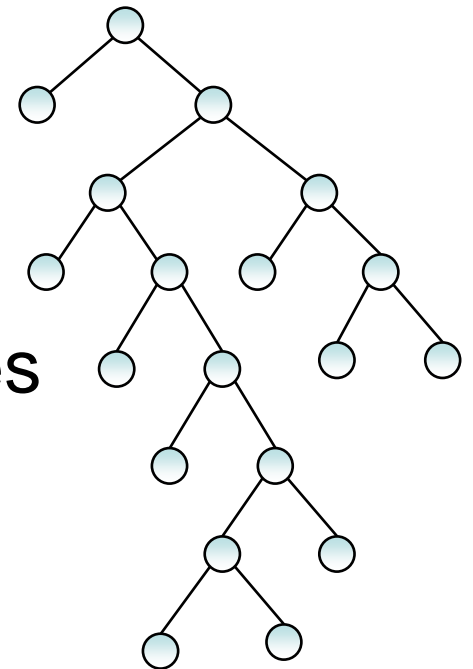
K Sadakane - [Lecture Notes in Computer Science](#), 2000 - Springer
... Based on the Compressed **Suffix Array** ... For such texts a kind of full-text indices is used, for example **suffix arrays** [14] and **suffix trees** [15]. ...
[Cited by 91](#) - [Related articles](#) - [Web Search](#) - [BL Direct](#) - [All 12 versions](#)

←
}

First story: Suffix Array (SA)

THE PERMUTATION in Stringology
[Gonnet '87, Manber, Myers '90]

More than just a simplification of suffix trees
[Weiner '73, Mc Creight '76]



text T =

S	E	N	S	E	L	E	S	S	N	E	S	S	_
---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9	10	11	12	13	14
---	---	---	---	---	---	---	---	---	----	----	----	----	----

i	SA[i]	T[SA[i] ... n]
1	14	_
2	5	E L E S S N E S S _
3	2	E N S E L E S S N E S S _
4	11	E S S _
5	7	E S S N E S S _
6	6	L E S S N E S S _
7	10	N E S S _
8	3	N S E L E S S N E S S _
9	13	S _
10	4	S E L E S S N E S S _
11	1	S E N S E L E S S N E S S _
12	9	S N E S S _
13	12	S S _
14	8	S S N E S S _

text T =

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

LCP[i]	i	SA[i]	T[SA[i] ... n]
0	1	14	_
1	2	5	E L E S S N E S S _
1	3	2	E N S E L E S S N E S S _
3	4	11	E S S _
0	5	7	E S S N E S S _
0	6	6	L E S S N E S S _
1	7	10	N E S S _
0	8	3	N S E L E S S N E S S _
1	9	13	S _
2	10	4	S E L E S S N E S S _
1	11	1	S E N S E L E S S N E S S _
1	12	9	S N E S S _
2	13	12	S S _
	14	8	S S N E S S _

RMQ(i,j-1) = LCP(i,j)
 e.g. Bender et al. 00

Kasai et al. 01

Disproving two beliefs on LCP info

When searching for a pattern P of length m into text T of length n :

- $O(m \log n)$ time is the **best** without LCP info (?)
- $O(m + \log n)$ time using **extra** space for LCP info (?)

What if using a plain SA permutation?

Implicit search in lexicographically sorted array
[Andersson et al. '94, '95, '01]

$$\Theta \left(\frac{m \log \log n}{\log \log \left(4 + \frac{m \log \log n}{\log n} \right)} + m + \log n \right)$$

Upper bound can be better than what we say $O(m \log n)$

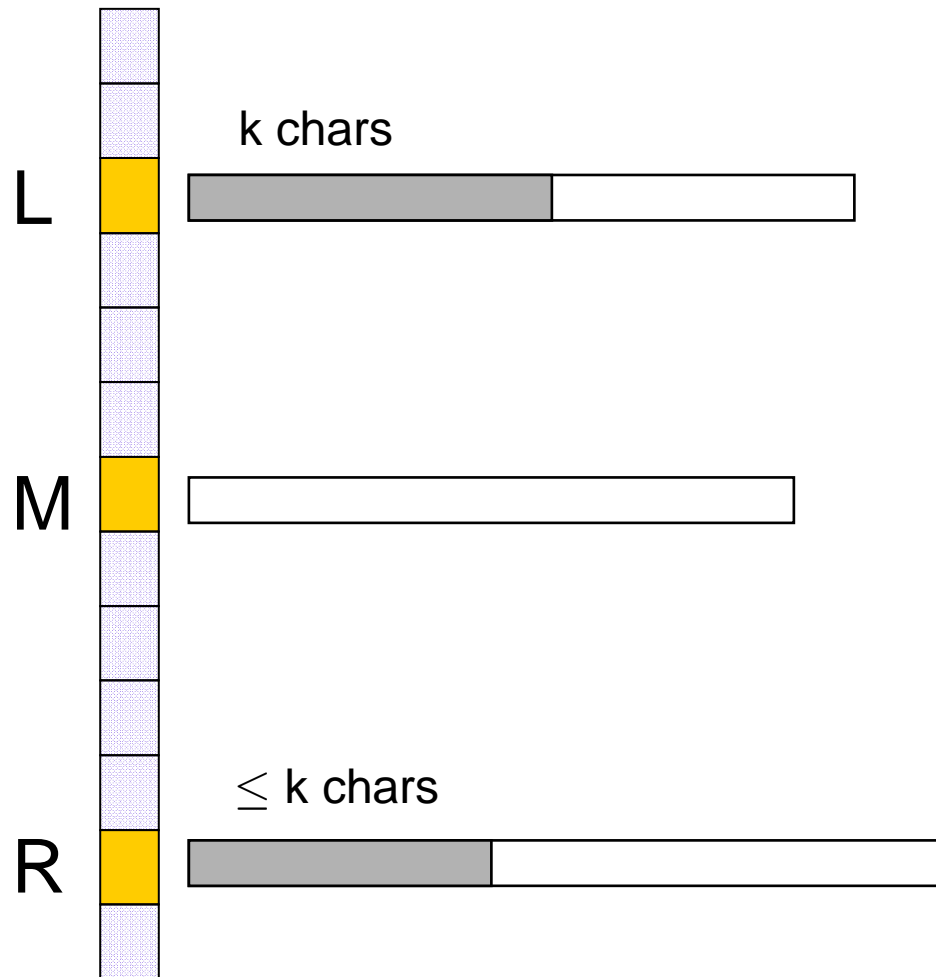
Lower bound is for n independent strings of length m

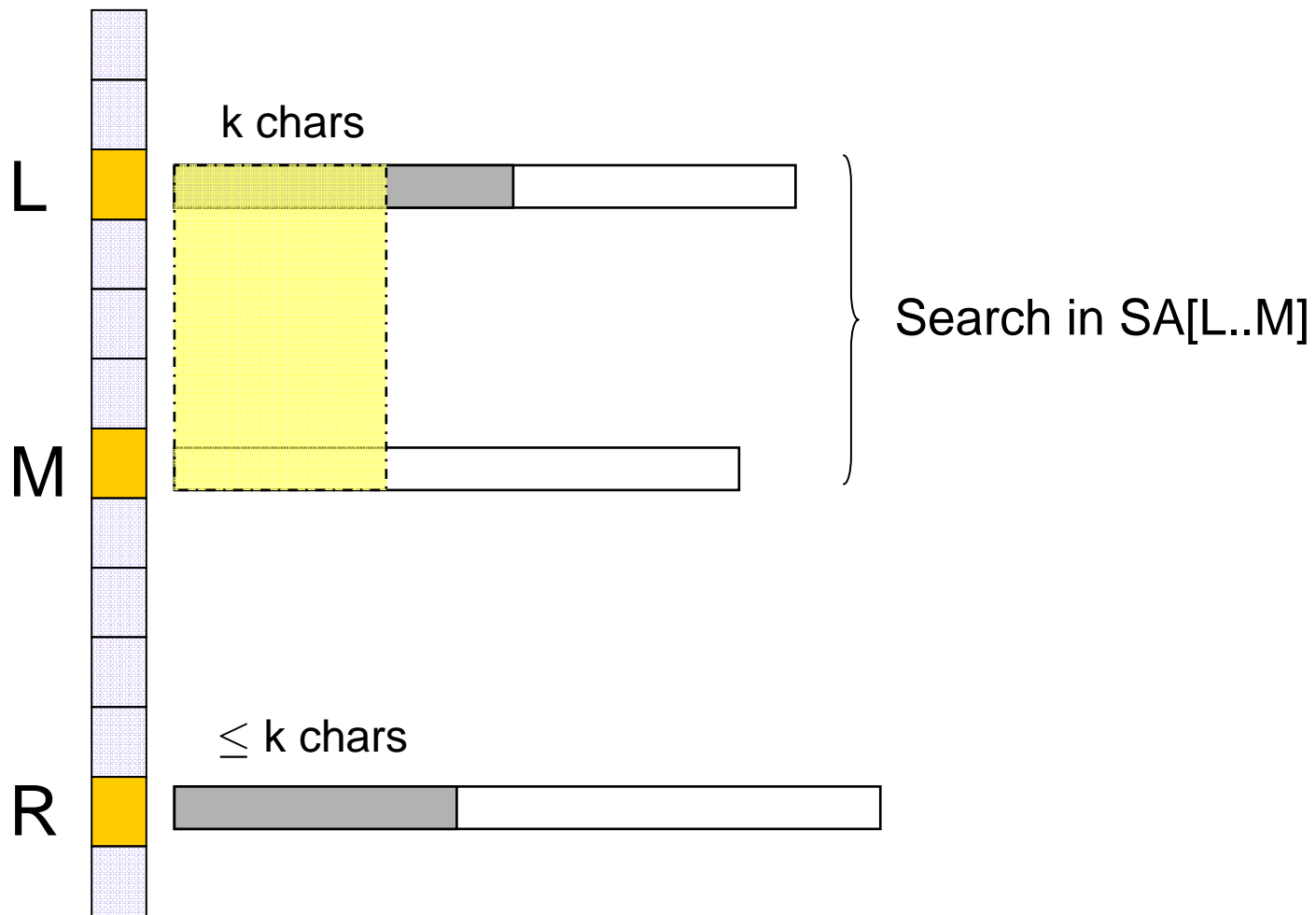
Need extra space for LCP info?

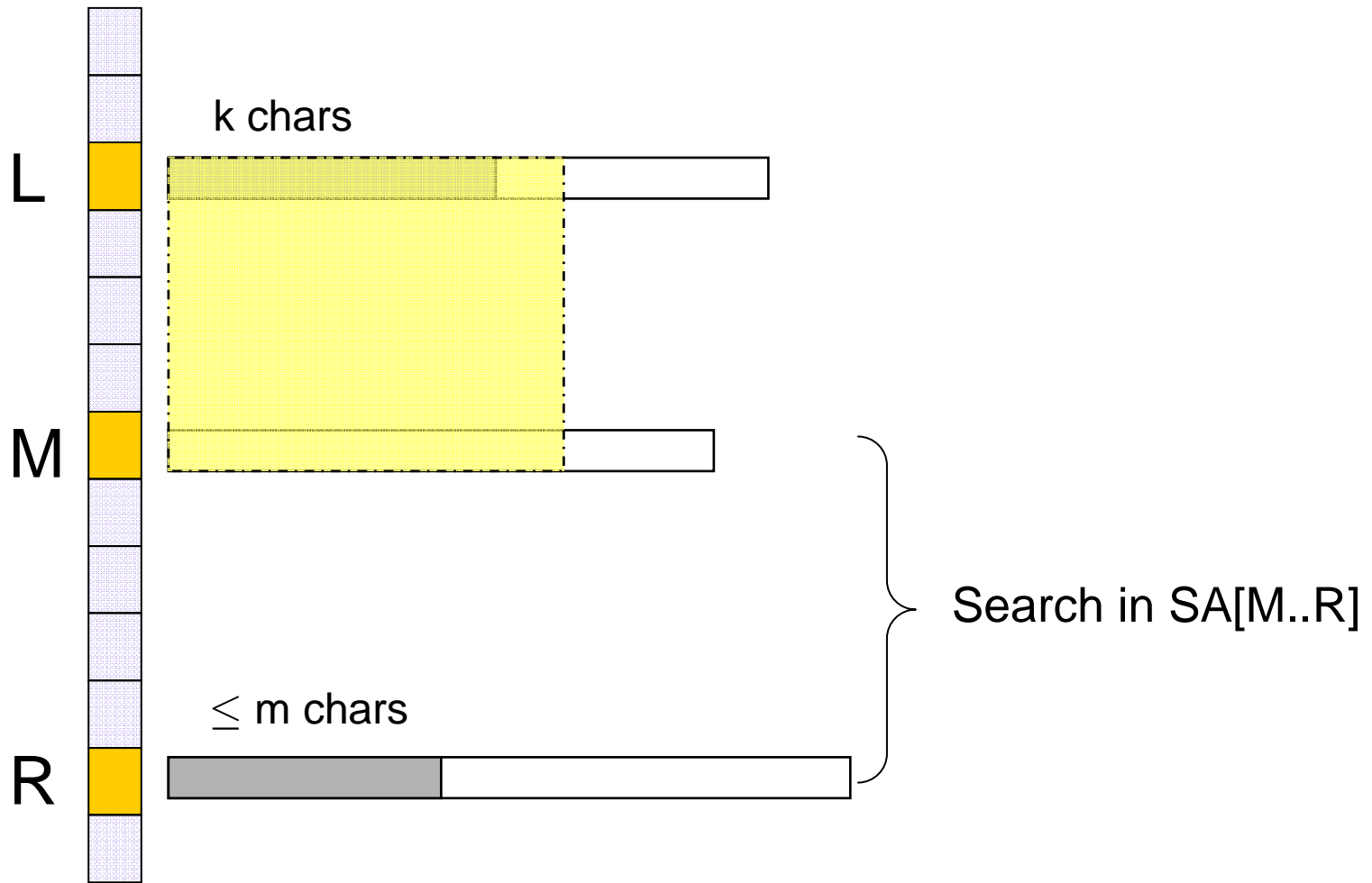
Let's first quickly review SA search [Manber, Myers '09]

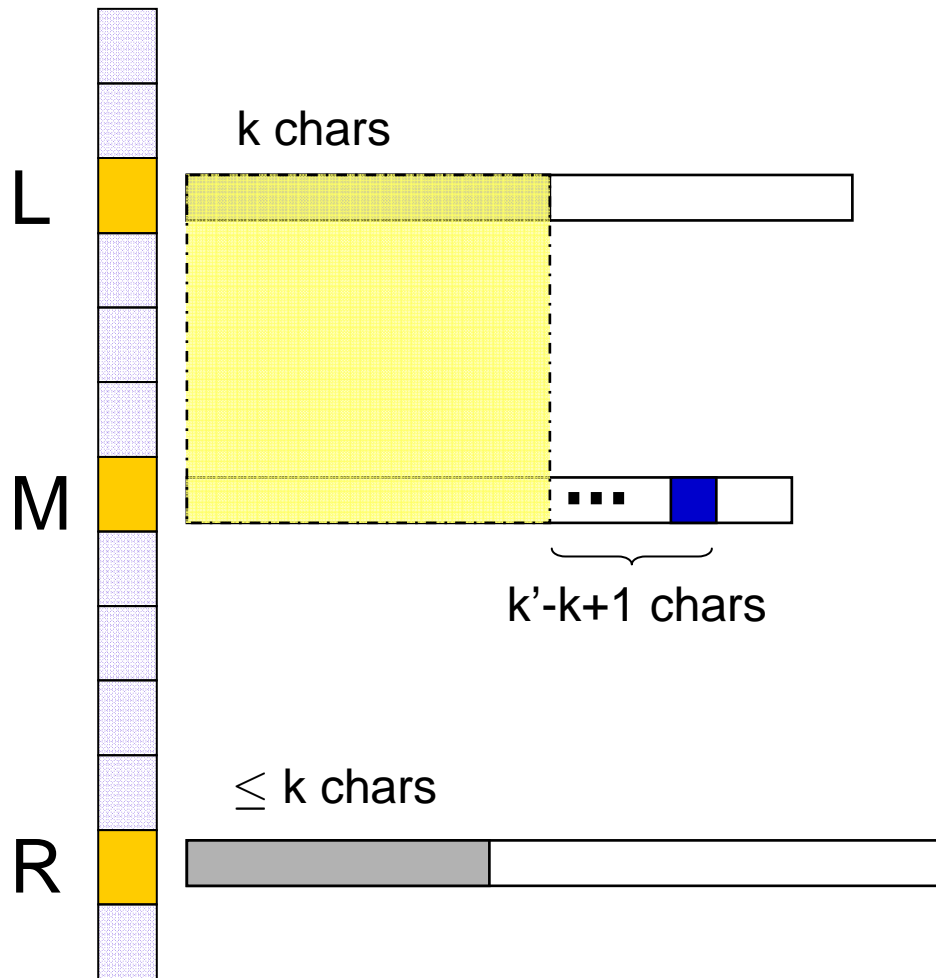
- Invariant for binary search step in SA[L..R]
 - *Pre*: matched $k < m$ chars
 - *Post*: matched k' chars ($k \leq k' \leq m$)
- Proceed by visual case analysis

“Visual” analysis of one case





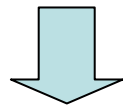




Match $k'-k$ chars and
 decide $SA[L..M]$ or $SA[M..R]$
 according to the **mismatch**

Making SA search “parsimonious”

It is possible to store an lcp value in $O(\log^2 n)$ bits, so that it can be compared by fetching just $O(1)$ of its bits



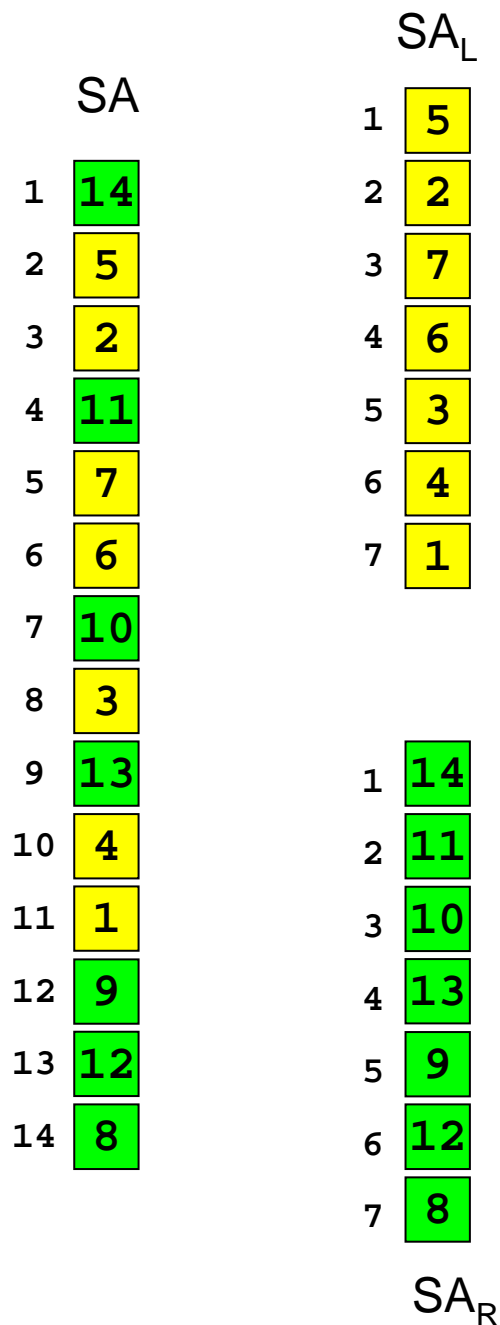
SA search can be modified so that it accesses $O(1)$ bits of LCP info per step

$O(m + \log n)$ search on a “perturbed” SA
without extra space for LCP info
[Francheschini & G. '04]

Rules of the game:

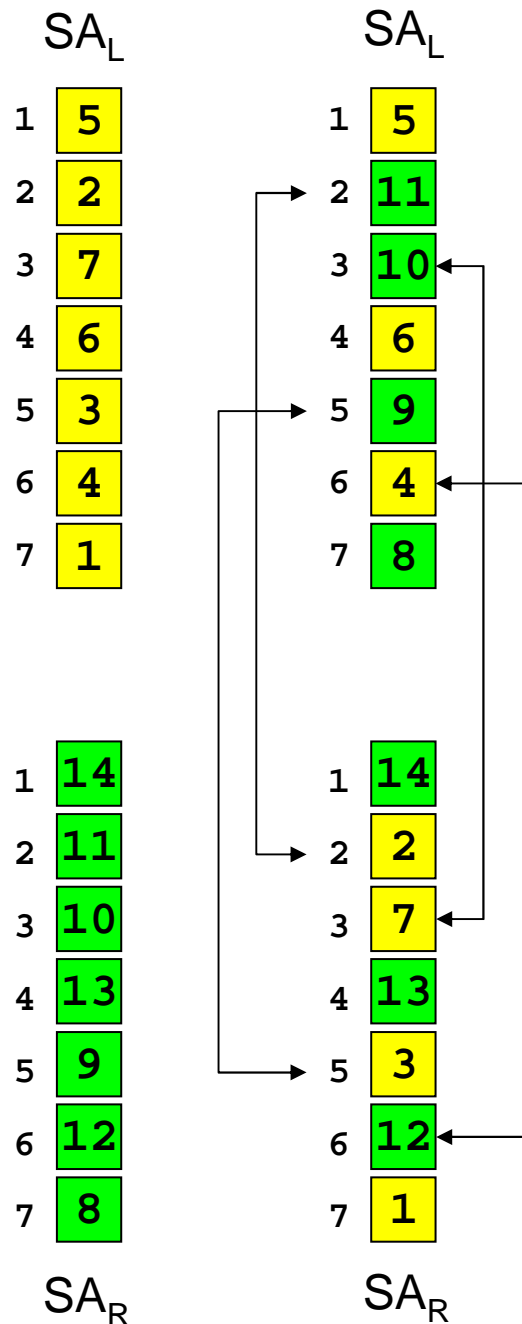
- given a read-only text T of n characters
- given a permutation of $[1\dots n]$, each entry of $\lceil \log n \rceil$ bits
- workspace of just $O(1)$ additional cells [$O(\log n)$ bits]
- how to find P in T using the above permutation

Bit stealing idea



- Partition SA permutation into SA_L and SA_R:
 - SA_L[i] ≤ n/2 and SA_R[i] > n/2
- One search in SA ⇒ one in SA_L and another in SA_R

Bit stealing idea



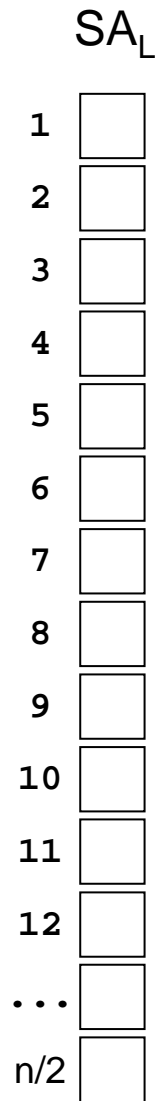
Define twin pair to encode one bit

- $0 \equiv (SA_L[i], SA_R[i])$
- $1 \equiv (SA_R[i], SA_L[i])$

So we have $n/2$ bits for “free”

Example: encoding **0110101**

Can still retrieve any original entry
from SA_L or SA_R in $O(1)$ time



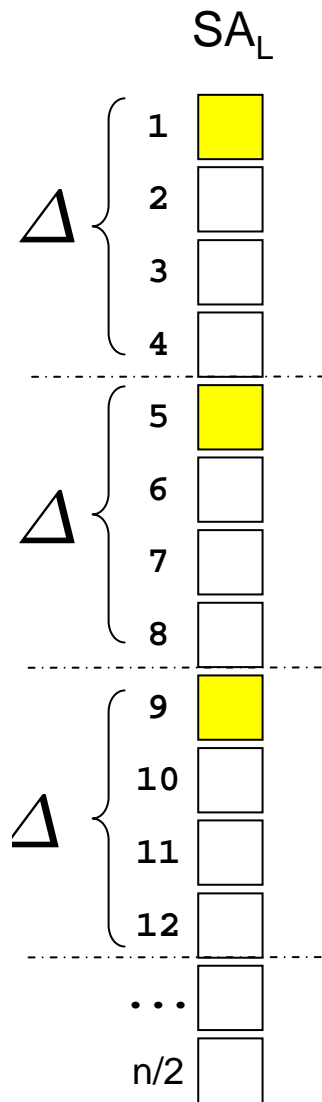
New game:

search SA_L (or SA_R) using $n/4$ stolen bits for LCP

recall parsomonus SA search retrieves

$O(1)$ bits per step

Goal: $O(m + \log n)$ search time without extra space for LCP info



1. fix $\Delta = \Theta(\log^2 n)$
2. sample every other Δ th suffix in SA_L
3. encode one LCP value per sample (bit stealing)
4. run parsimonious SA search and locate a run of Δ consecutive suffixes in SA_L
5. use Hirschberg's linear search twice in $O(m + \Delta^{1/2})$ time

Total time = $O(m + \log n)$

Space-efficient computation of SA

Same game rules as implicit searching

In-place suffix sorting:

$O(n \log n)$ time and $O(1)$ additional space

[Franceschini, Muthu '07]

based on Ko and Aluru's suffix sorting

Useful to compute our SA permutation

Checkpoint

We started from
text T , suffix array SA and LCP info



We saved storage for LCP info



Goal: can we also drop text T ?

A Dedication to French thinkers

*Cette denrée
[savoir/informaiton], donc, se
préparera sous des formes de
plus en plus maniables ou
comestibles ; elle se
distribuera à une clientèle de
plus en plus nombreuse ; elle
deviendra chose du
Commerce, chose enfin qui
s'imité et se **produit un peu
partout**.*

Paul Valéry, La Crise de l'esprit,
1919,

*This commodity
[knowledge/information], as a
result,
will be prepared in more and
more convenient or comestible
forms; it will be **distributed** to a
larger and larger circle of
buyers: it will be transformed
into something **commercial**,
something which is **imitated**
and **produced almost
everywhere**.*

Paul Valéry The intellectual crisis,
1919 (translated by Malcolm Cowley)

Checkpoint

We started from
text T , suffix array SA and LCP info



We saved storage for LCP info

Goal: can we also drop text T ?

Are all $[1..n]$ -perms SA permutations?

1	2	3
A	A	A
A	A	B
A	B	A
A	B	B
B	A	A
B	A	B
B	B	A
B	B	B

1	2	3
3	2	1
1	2	3
3	1	2
1	3	2
3	2	1
2	3	1
3	2	1
3	2	1

➤ Some permutations do **not** occur (e.g. 213)

➤ Some permutations occur **more than once** (e.g. 321)

$$\Sigma = \{A, B\}, n = 3$$

Getting itchy about combinatorics of SA perms?

	SA[i]	$\Phi(i)$
1	14	0/11
2	5	6
3	2	8
4	11	13
5	7	14
6	6	5
7	10	4
8	3	10
9	13	1
10	4	2
11	1	3
12	9	7
13	12	9
14	8	12

“suffix link” for SA:
 Φ function [G.&Vitter '00]:

$$\Phi(i) = k \text{ iff } SA[k] = SA[i] + 1$$

rank of the next suffix in T

$\Phi(i)=0$ when $SA[i]=n$ or

$\Phi(i)=z$ where $SA[z]=1$

Necessary and SUFFICIENT conditions

Duval et al '02, Bannai et al '03, Burkhardt et al '03,
Hohlweg et al '03, Crochemore et al '05, He et al '05,
Kopelowitz et al '05, Schurman et al '05

SA is the suffix array of T **iff** for each i

$$-T[SA[i]] \leq T[SA[i+1]]$$

$$-T[SA[i]] = T[SA[i+1]] \Rightarrow \Phi(i) < \Phi(i+1)$$

SA is the suffix array of T iff for each i

$$-T[SA[i]] \leq T[SA[i+1]]$$

$$-T[SA[i]] = T[SA[i+1]] \Rightarrow \Phi(i) < \Phi(i+1)$$

	SA[i]	$\Phi(i)$	
1	14	0/11	↓
2	5	6	E L
3	2	8	E N S E L E S S N E S S _
4	11	13	E S S _
5	7	14	E S S N E S S _
6	6	5	L E S S N E S S _
7	10	4	N E S S _
8	3	10	N S E L E S S N E S S _
9	13	1	S _
10	4	2	S E L E S S N E S S _
11	1	3	S E N S E L E S S N E S S _
12	9	7	S N E S S _
13	12	9	S S _
14	8	12	S S N E S S _

SA is the suffix array of T iff for each i

$$-T[SA[i]] \leq T[SA[i+1]]$$

$$-T[SA[i]] = T[SA[i+1]] \Rightarrow \Phi(i) < \Phi(i+1)$$

	SA[i]	$\Phi(i)$	
1	14	0/11	
2	5	6	E L
3	2	8	E N S E L E S S N E S S _
4	11	13	E S S _
5	7	14	E S S N E S S _
6	6	5	L E S S N E S S _
7	10	4	N E S S _
8	3	10	N S E L E S S N E S S _
9	13	1	S _
10	4	2	S E L E S S N E S S _
11	1	3	S E N S E L E S S N E S S _
12	9	7	S N E S S _
13	12	9	S S _
14	8	12	S S N E S S _

order dictated by the rank
of their next suffix

How many SA permutations?

[Schurmann, Stoye '05]

	SA[i]	$\Phi(i)$
1	14	0/11
2	5	6
3	2	8
4	11	13
5	7	14
6	6	5
7	10	4
8	3	10
9	13	1
10	4	2
11	1	3
12	9	7
13	12	9
14	8	12

Position i is a **descent**
if $\Phi(i) > \Phi(i+1)$

d descents \Rightarrow # strings
having the same SA is

$$\binom{n + \sigma - d - 1}{\sigma - d - 1}$$

Are all $[1..n]$ -perms SA permutations?

1	2	3
A	A	A
A	A	B
A	B	A
A	B	B
B	A	A
B	A	B
B	B	A
B	B	B

1	2	3
3	2	1
1	2	3
3	1	2
1	3	2
3	2	1
2	3	1
3	2	1
3	2	1

$$\Sigma = \{A, B\}, n=3, d=0$$

$$\binom{n+\sigma-d-1}{\sigma-d-1} = \binom{3+2-0-1}{2-0-1} = \binom{4}{1} = 4$$

How many distinct SAs for all strings in Σ^n ?

[Schurmann , Stoye '05]

$\left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle$ = Eulerian number = #perms with d descents

$$\# \text{ SAs} = \sum_{d=0}^{\sigma-1} \left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle \quad \text{and} \quad \lim_{n \rightarrow \infty} \frac{\sum_{d=0}^{\sigma-1} \left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle}{\sigma^n} = 1$$

constant σ

Are all $[1..n]$ -perms SA permutations?

1	2	3
A	A	A
A	A	B
A	B	A
A	B	B
B	A	A
B	A	B
B	B	A
B	B	B

1	2	3
3	2	1
1	2	3
3	1	2
1	3	2
3	2	1
2	3	1
3	2	1
3	2	1

$$\Sigma = \{A, B\}, n=3, d=0$$

In our example: $\left\langle \begin{matrix} 3 \\ 0 \end{matrix} \right\rangle + \left\langle \begin{matrix} 3 \\ 1 \end{matrix} \right\rangle = 1 + 4 = 5$

321

123, 312, 132, 231

Information theoretical lower bound

$$\log \left(\sum_{d=0}^{\sigma-1} \binom{n}{d} \right) \sim n \log \sigma \text{ bits}$$

to be compared with $n \lceil \log n \rceil$ bits for the plain SA permutation (e.g. when σ is constant)

Checkpoint

We started from
text T, suffix array SA and LCP info



We saved storage for LCP info

Let's drop text T

Text endmarkers make life easier

$$\Sigma' = \{A, \#, B\}, n' = n + 1$$

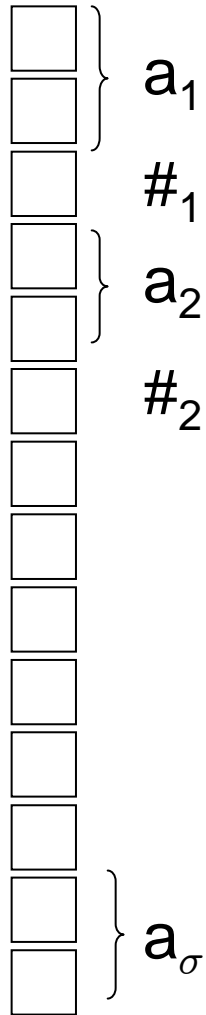
1	2	3	4
A	A	A	#
A	A	B	#
A	B	A	#
A	B	B	#
B	A	A	#
B	A	B	#
B	B	A	#
B	B	B	#

1	2	3	4
1	2	3	4
1	2	4	3
3	1	4	2
1	4	3	2
2	3	4	1
2	4	1	3
3	4	2	1
4	3	2	1

all distinct perms

Conceptual transformation

SA'



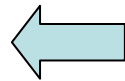
Alphabet: $\Sigma = \{ a_1 < a_2 < \dots < a_\sigma \} \rightarrow$
 $\Sigma = \{ a_1 < \#_1 < a_2 < \#_2 < \dots < a_\sigma \}$

Text: $T \rightarrow T' = T \#_1 \#_2 \dots \#_{\sigma-1}$

Resulting SA' can reconstruct text T
($\#_1, \#_2, \dots$ are not actually needed)

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

bv[i]		SA[i]	$T[SA[i]]$	$\Phi(i)$
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12

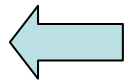


Using Φ and a bitvector we can avoid storing SA

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

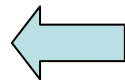
bv[i]		SA[i]	T[SA[i]]	$\Phi(i)$
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
1	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12

Using Φ and a bitvector we can avoid storing SA



S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

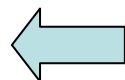
bv[i]		SA[i]	T[SA[i]]	$\Phi(i)$
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12



Using Φ and a bitvector we can avoid storing SA

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

bv[i]		SA[i]	$T[SA[i]]$	$\Phi(i)$
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12

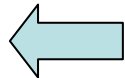


Using Φ and a bitvector we can avoid storing SA

S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

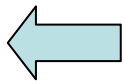
bv[i]		SA[i]	$T[SA[i]]$	$\Phi(i)$
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12

Using Φ and a bitvector we can avoid storing SA



S	E	N	S	E	L	E	S	S	N	E	S	S	_
1	2	3	4	5	6	7	8	9	10	11	12	13	14

bv[i]		SA[i]	T[SA[i]]	$\Phi(i)$
_	1	14	_	0/11
1	2	5	E	6
	3	2	E	8
	4	11	E	13
	5	7	E	14
1	6	6	L	5
1	7	10	N	4
	8	3	N	10
1	9	13	S	1
	10	4	S	2
	11	1	S	3
	12	9	S	7
	13	12	S	9
	14	8	S	12



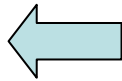
Using Φ and a bitvector we can avoid storing SA

S	E	N	S	E	L	E	S	S	N	E	S	S	_
2	1	3	4	5	6	7	8	9	10	11	12	13	14

	bv[i]	SA[i]	T[SA[i]]	$\Phi(i)$	
	—	1	14	—	0/11
	1	2	5	E	6
		3	2	E	8
		4	11	E	13
		5	7	E	14
	1	6	6	L	5
	1	7	10	N	4
		8	3	N	10
Interval _c	1	9	13	S	1
		10	4	S	2
		11	1	S	3
		12	9	S	7
		13	12	S	9
		14	8	S	12

Replacing the SA

- Record which SA[j]=1
- T[i] = c iff $\Phi^{(i)}(j) \in \text{Interval}_c$



Entering compression stuff...

A first simple approach

- Encode the bitvector $\sigma \log (n/\sigma) + o(n)$ bits
- Encode Φ in $n \log \sigma + o(n \log \sigma)$ bits
- Compressed representation in
 $n \log \sigma + \text{lower order bits}$
[matches the lower bound seen before]

Another story is coming: succinctness

- Succinct data structures [Jacobson '87]
- space $\log(\# \text{ configurations}) + \text{lower order bits}$
- time $O(1)$ per supported operation (if possible)
- Example:
 - sets of n elements out of universe $[1\dots m]$

$$B(n,m) = \lceil \log \binom{m}{n} \rceil \text{ bits}$$

Fully indexable dictionaries (FID)

[Brodnik, Munro '99, Raman et al '02]

- Bitvector BV with n 1s and $m-n$ 0s:

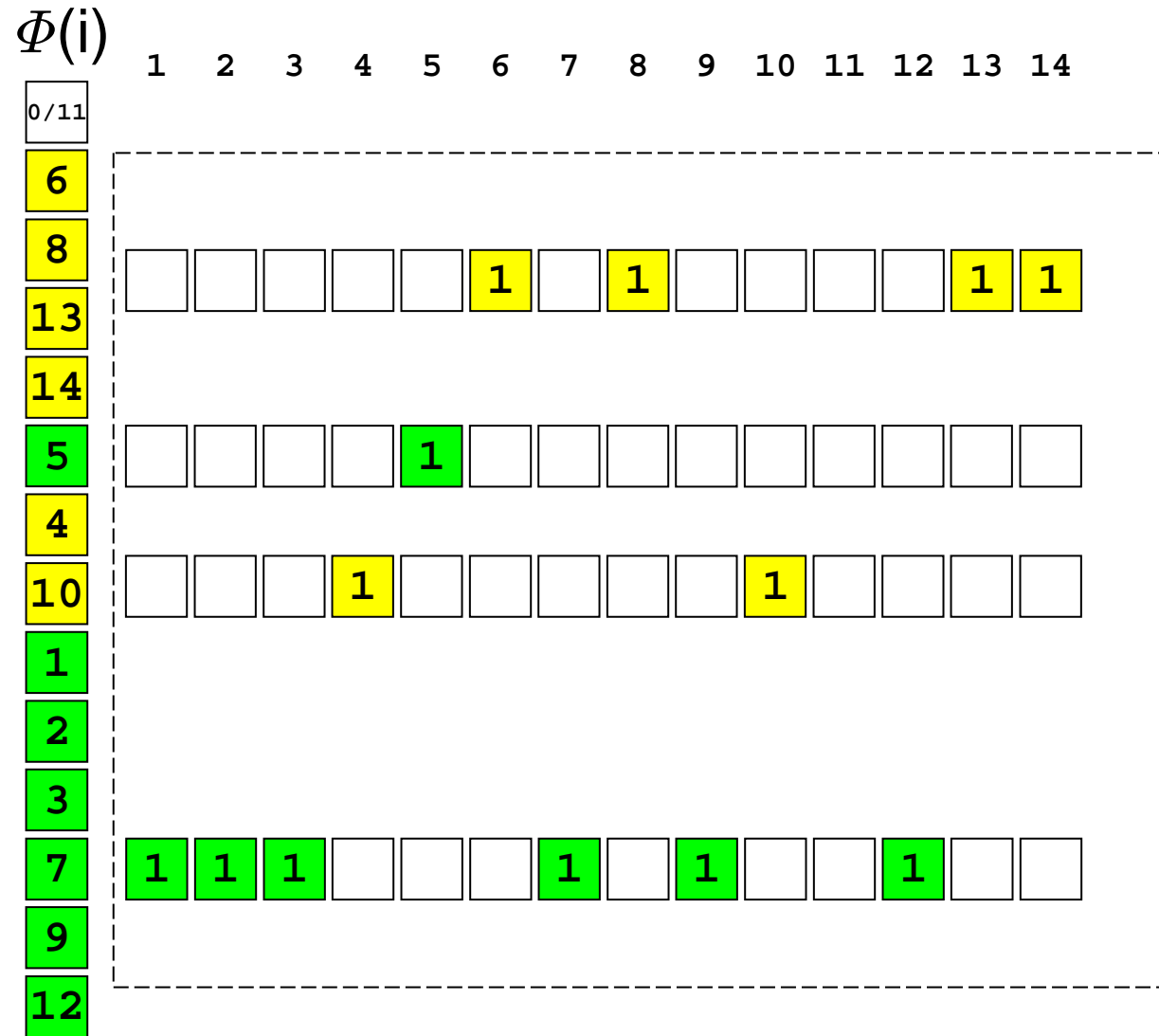
- $\text{rank}_1(i) = \#1\text{s in } BV[1\dots i]$
- $\text{select}_1(j) = \text{position in BV of the } j\text{th } 1$
- same operations for 0s

- Can solve also the predecessor problem

- space: $B(n,m) + o(m) \sim n \log(m/n) + o(m)$ bits

- time: $O(1)$

Using FIDs to store Φ function: Σ lists



Back to Φ

- Observation made when counting SA perms:
 $T[SA[i]] = T[SA[i+1]] \Rightarrow \Phi(i) < \Phi(i+1)$
- Monotonicity: a maximal run of equal symbols
 $T[SA[i]] = \dots = T[SA[j]] \Rightarrow 1 \leq \Phi(i) < \dots < \Phi(j) \leq n$
- At most σ runs, called Σ lists

Storing Φ : first idea

- Concatenate all the σ bitvectors
- Build a single FID using

$$B(n, n \sigma) + o(n\sigma) \sim n \log \sigma + o(n \sigma) \text{ bits}$$

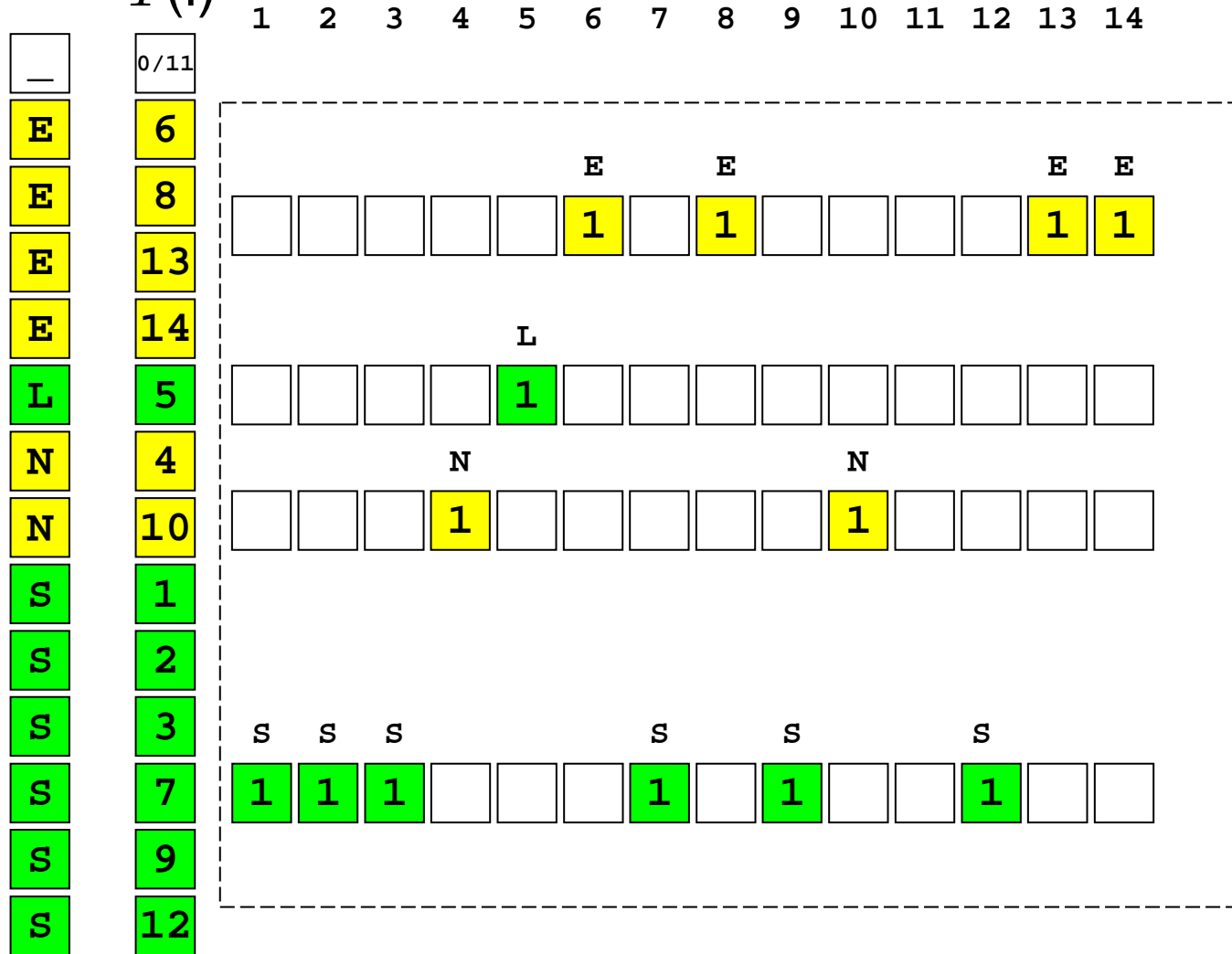
- Problem: $o(n \sigma)$ can be $\gg n \log \sigma$...

Better idea: Wavelet tree

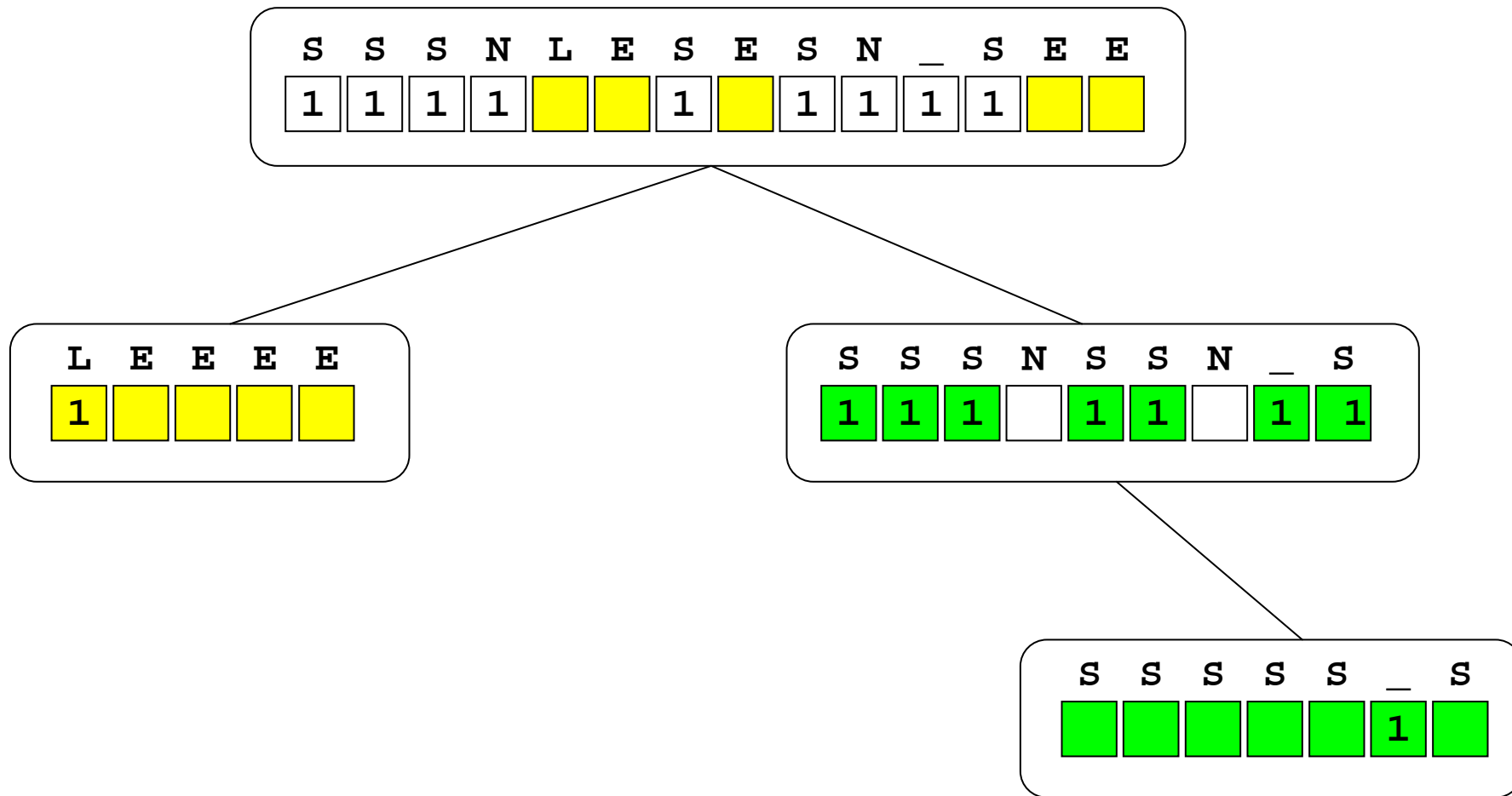
[G., Gupta, Vitter '03]

- Extend a FID from bitvectors to strings $S \in \Sigma^n$:
 - $\text{rank}_c(i) = \#cs \text{ in } S[1\dots i]$
 - $\text{select}_c(j) = \text{position in } V \text{ of the } j\text{th } c \text{ in } S$
 - for any $c \in \Sigma$
- Space $n \log \sigma + o(n \log \sigma)$ bits
- Time $O(\log \sigma)$

T[SA[i]] $\Phi(i)$



S S S N L E S E S N _ S E E



Wavelet tree

Compressed suffix array (CSA)

G. & Vitter '00, Sadakane '00

- Using additional $(n/k) \log n$ bits, we can obtain $SA[i]$ in $O(k)$ time using Φ . If $k = \omega(\log_{\sigma} n)$, this is $o(n \log \sigma)$ bits.

G., Gupta, Vitter '03

- High order entropy

The whole story is long...

- Φ function and CSA [G., Vitter '00]
- Avoiding to store the text [Sadakane '00]
- High order entropy and link to BWT– FM-index [Ferragina, Manzini'00]
- Wavelet tree [G., Gupta, Vitter, '03]
- Space-efficient construction [Hon, Sadakane, Wong '03]
- Alphabet-friendly FM-index [Ferragina, Makinen, Manzini, Navarro '04]
- ... many other results.....nice survey paper [Makinen, Navarro]

Unsolved problems: 10 years old open problems

- CSA using $n H_k + o(n \log \sigma)$ bits and $O(\log^\epsilon)$ decompress time $\rightarrow O(1)$ time?
- As of now, CSA makes random access to memory:
 - can we retrieve p pointers from SA with $O(p/B)$ transfers ($B =$ blocksize)?
 - can we decompress t consecutive symbols from T with $O(p/B)$ transfers?
- Can we add/remove a text to CSA in $O(n)$ time?
- Technological transfer? [pizza-chili web site by Ferragina and Navarro]

THANKS!

THE END

Eulerian numbers

Recursive enumeration of Eulerian numbers

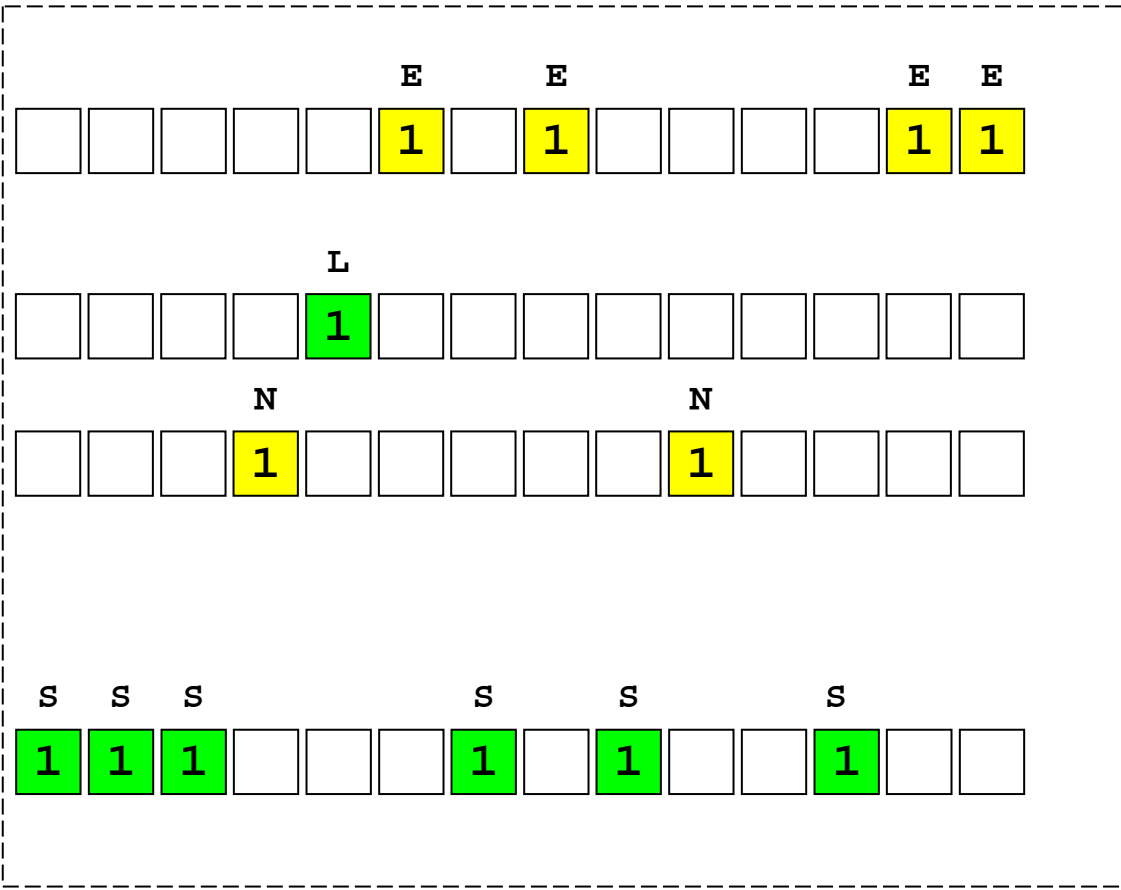
$$\text{a) } \left\langle \begin{matrix} n \\ 0 \end{matrix} \right\rangle = 1$$

$$\text{b) } \left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle = 0 \text{ for } n \leq d, \text{ and}$$

$$\text{c) } \left\langle \begin{matrix} n \\ d \end{matrix} \right\rangle = (d + 1) \left\langle \begin{matrix} n - 1 \\ d \end{matrix} \right\rangle + (n - d) \left\langle \begin{matrix} n - 1 \\ d - 1 \end{matrix} \right\rangle$$

bv[i]	SA[i]	T[SA[i]]	$\Phi(i)$
S	14	_	0/11
S	5	E	6
S	2	E	8
N	11	E	13
L	7	E	14
E	6	L	5
S	10	N	4
E	3	N	10
S	13	S	1
N	4	S	2
_	1	S	3
S	9	S	7
E	12	S	9
E	8	S	12

1 2 3 4 5 6 7 8 9 10 11 12 13 14



S S S N L E S E S N _ S E E