On a Technique for Parsing a String

Joint with S.C. Sahinalp

Alternative title:
An Idea Which Transcended
Its Original Application

Why Strings

- An elementary way for presenting data.

- Applications: text processing, computational biology, voice(?)...

Initial Talk Plan

- Highlight quest for new ideas as a research goal

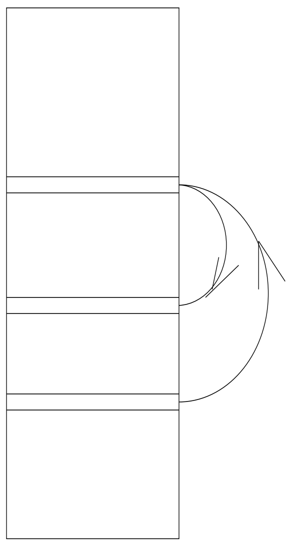
- Research Methodology:
1. Advance to the simplest problem you don't know to solve.
2. Think...

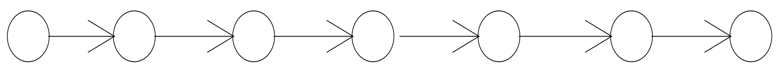
- Demonstrate methodology, and power of new ideas.

Original Simplest Problem Chain

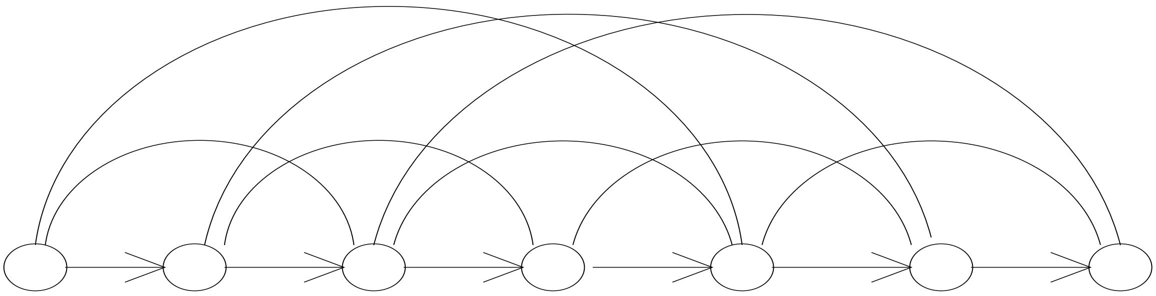
Motivation: "thinking in parallel". Existence of algorithm theory: "benchmark" for parallel system concepts.
Parallel algorithms for textbook problems: graph and tree problems.

New Problem: Parallel List Ranking.
Input: Linked list in an array. For each element, find distance from end of list.





Standard solution.



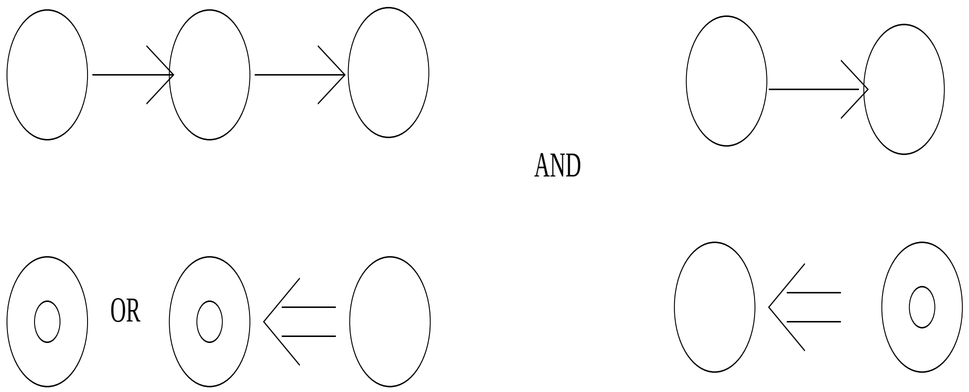
Inherent $O(nlogn)$ work.
Current simplest question: Is $O(n)$ possible?

Intuition for algorithmic attempt.
Problem is: Long line of people. Each one can communicate only with the next person. Where am I in line?

Attempt is: Have every second person raise his/her hand. Difficulty: how to do that without full ranking?

New simplest problem: 2-ruling set

Input: Unchanged (linked list).
Find a 2-ruling set. Definition:



• Deterministic coin tossing technique [Cole-V].
My problem: if I tell you now, you will go home.

Note: relevant to list ranking.

Applications

• Parallel Algorithms.
List problems: 2-ruling set, list ranking.
Tree problems: traversals, arithmetic.
Graph problems: k-connectivity $(k = 1, 2, 3, ..)$, spanning forests, path problems, planar graphs, coloring.

• Symmetry breaking: distributed computing.

• Comparing dynamic sets [Mehlhorn–Sundar-Uhrig].

New: serial(!) and parallel

- Suffix trees.
- String matching with errors.
- Dynamic data structures for string algorithms:

(i) dynamic alternative to suffix trees/ dynamic text indexing
(ii) dynamic (Aho-Corasick type) dictionary matching

- Data compression (in progress).

Points

• Demonstrates: Ideas transcend prescribed applications. Could not have been predicted.

• Structure of parallel algorithmics. Unusual for combinatorial related theories. See ICALP91.

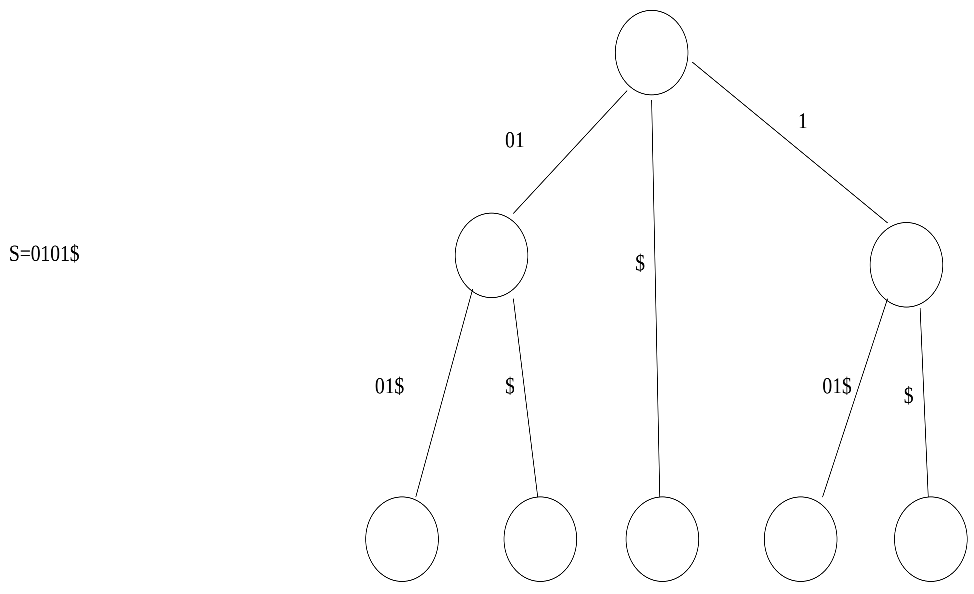
• Talk is a depth cut of the structure, rather than a bird-eye's view (breadth cut), emphasizing ideas over concrete complexity results.

Suffix trees

Input: String $S = s_1, s_2, .., s_n$, where $s_n = \$$.

• Rooted tree. Excluding root and leaves, each node must have at least 2 children.
• Each edge labeled with substring of S. First characters in sibling edges are different.

• Path from root to leaf "one-to-one" suffix of S.

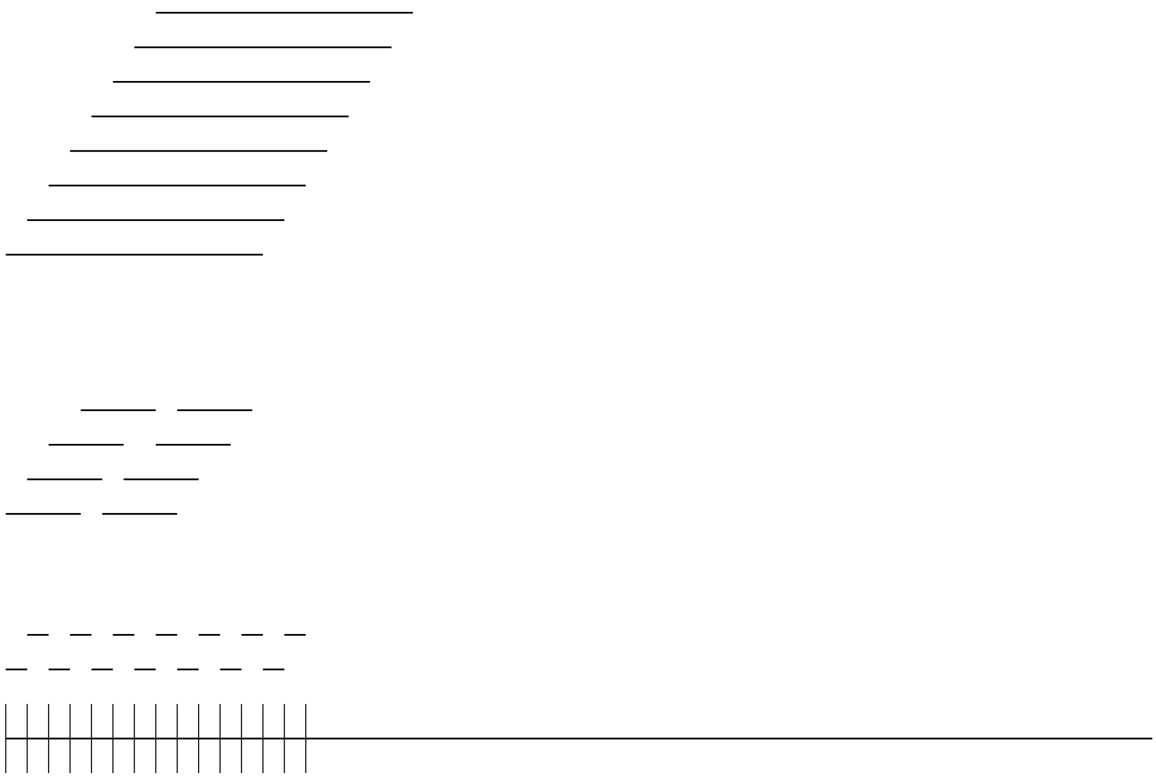


Note: lowest common ancestor (LCA) of two leaves gives their (longest common) prefix.

Preprocess in $O(n)$ time, then query in $O(1)$ time:
Harel-Tarjan, Schieber-V, Berkman-V.

Suffix tree algorithms

Karp, Miller, Rosenberg. $O(nlogn)$ time.
Apostolico, Iliopoulos, Landau, Schieber, V. $O(nlogn)$ work, $O(logn)$ time.
Key step:



Similarity: $O(nlogn)$ work. Is $O(n)$ possible?

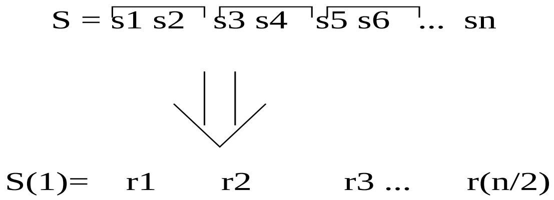
“Motivational” problem

Use only $O(n)$ labels.

(Motivational only since one can construct suffix trees by other approaches.)

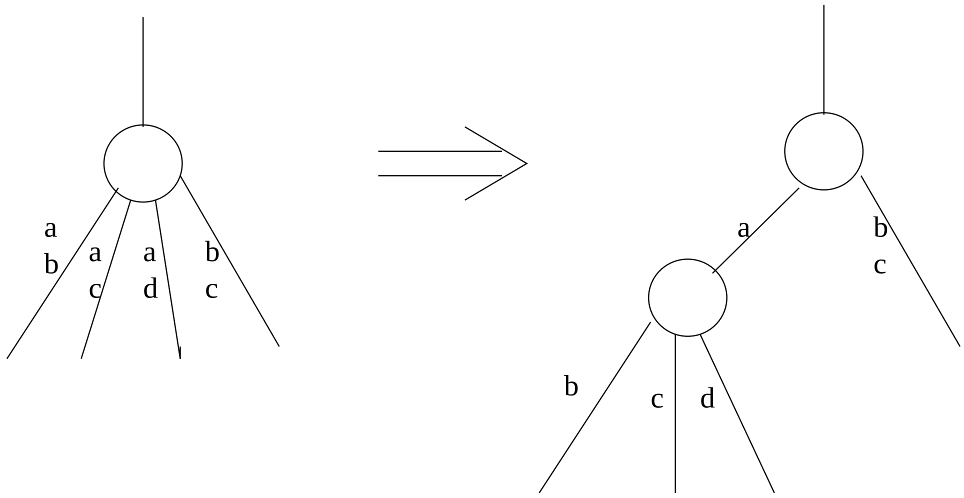
Wishful thought algorithm

1. Label each pair starting at every odd index

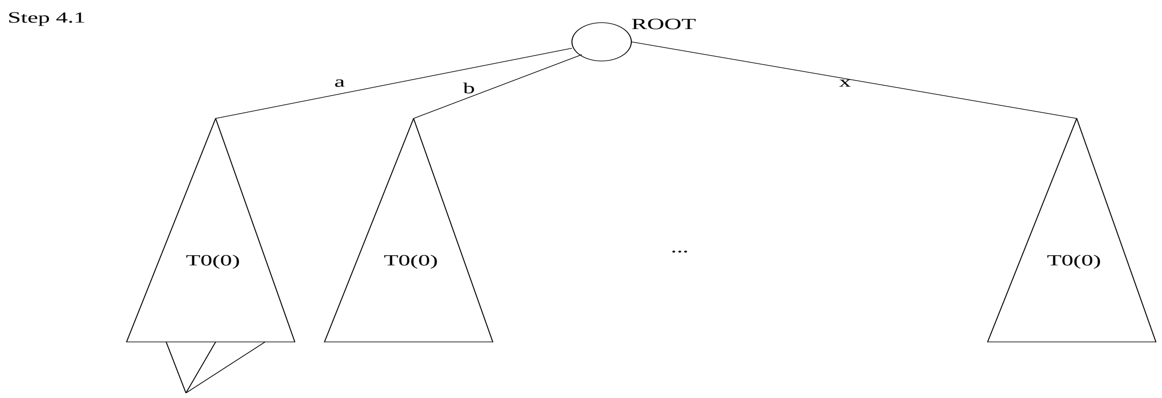


2. Recursively find the suffix tree $T(1)$ for $S(1)$.

3. Get $T(0)_0$ - a suffix tree for all odd indices of S.
Use procedure REFINE which looks for



4. Get $T(0)_1$ - a suffix tree for all even indices of S.



Mark all leaves of an “even-suffix” starting with a.

4.2 preprocess $T(0)_0$ for LCA queries.

4.3 procedure CONTRACT extracts from each copy of $T(0)_0$ the needed part for $T(0)_1$.

5. MERGE $T(0)_1$ and $T(0)_0$ into $T(0)$.

However ...

Design labeling of substrings (including which substrings to label) so that MERGE will be easy.

Our design results in a call for MERGE so that suffixes from different tree can have only “short” common prefixes.

High-level Description

Partition S into blocks of size 2 or 3.
Use the same label for the same block.

Partition consistency property:

$$S \quad |1\,\overbrace{0|2\,1|3\,0|4\,1}^{\text{Xi}}|3\,2\,\overbrace{0|2\,1|3\,0|4\,1\,2}^{\text{Xj}}|$$

$$L \quad 1 \quad 3 \quad 5 \quad 7 \quad 9 \quad 3 \quad 5 \quad 16$$

Observe: $X_i = X_j$.
Property is: same partition and labels.
Exception: some margins.

An iteration

$$\text{INPUT: } R = \ldots\underline{33}\,\overbrace{8\,4\,1\,2\,1\,2\,4\,8\,4\,8\,4}^{R'}\,\underline{88}\ldots$$

1. Separate repeating characters and apply different treatment to them.

2. Use **deterministic coin tossing** for breaking R' into blocks of length 2 or 3.

R’		8	4		2	1	2	4		8	4		8	4	
Binary		1000	0100		0010	0001	0010	0100		1000	0100		1000	0100	
Tags		3	2		1	1	2	3		3	3		3	0	
Dividers	|			|		reapply			|			|		|
Step#	1			2					4			2			1

Installing dividers:

(Caveat: never create a singleton.)

1. At ends.
2. 2 from ends.
3. At strictly local maxima.
4. At weakly local maxima with a 1 bit.

Consistency lemma

Given $X_i = X_j$, all but:
(1) $\log^* n + 1$ blocks in the right margin, and
(2) $\log^* n + 1$ blocks in the left margin
must be identical.

Implies short common prefixes...

Advertized Problem Was Another "Simplest"

Informally, "ideal parsing" of a string S provides
(i) (Consistency in partition:) Partition S into substrings of length 2 or 3 each; identical substrings in different parts of S are partitioned in the same way into short substrings.
(ii) (Consistency in labeling:) Assign unique labels to substrings; identical substrings get the same label; thereby, obtain a new string out of S.
(iii) Apply items (i) and (ii) (recursively) to the new string.

Conclusion

• Suffix trees are the single most important data structure for string algorithms. Efficient algorithm completely different than the main family of existing algorithm: [Weiner-73], [McCreight-76].

• **Postscript** A recent algorithm where the wishful thought Sahinalp-V algorithm (the one that first works recursively on the odd indices) is directly used is by Farach-FOCS97. The only difference is with respect to Merge. For integer input, subtle linear time SERIAL execution of Merge, implies $T(n) = T(n/2) + O(n)$.

• New ideas: hard to come by, but worth while the effort.