Weiner's Repetition Finder

(with simplifications suggested by the referee)

Vaughan Pratt

Computer Science Department
Stanford University

Combinatorial Pattern Matching 2013

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973
2. Referee's Report
   - Conditional accept
   - Condition: make more understandable

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973

2. Referee's Report
   - Conditional accept
   - Condition: make more understandable

3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973
2. Referee's Report
   - Conditional accept
   - Condition: make more understandable
3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

Weiner's starting point: Knuth's 1970 conjecture

- Knuth conjectured in 1970 that the longest common substring of two strings could not be found in linear time [KMP].
- Weiner set out to find a linear time algorithm for this problem.
- Morphed into a data structures paper: bi-trees, prefix trees, and associated algorithms.
- These solved a more general problem: build the suffix tree of a string in linear time, along with other applications.

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973
2. Referee's Report
   - Conditional accept
   - Condition: make more understandable
3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

SWAT paper

- Paper presented at SWAT'73.
- (Switching and Automata Theory, renamed Foundations of Computer Science, FOCS, in 1975.)

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973
2. Referee's Report
   - Conditional accept
   - Condition: make more understandable
3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

Journal submission

- Submitted for journal publication in CACM in 1973.
- Referee: VP

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973
2. Referee's Report
   - Conditional accept
   - Condition: make more understandable
3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

Accept or Reject?

- I attended SWAT'73 and listened to Peter's talk with great interest. The universal reaction seemed to be that the arguments were very intricate, and the question of correctness arose.
- When John Hopcroft asked me to referee the paper for CACM, I had no preconceptions about its correctness.
- On the one hand, after two weeks I was unable to find any serious error.
- On the other I was also unable to "grok" the method.
- Perhaps TOC is not my thing after all, and I should go back to NLP...

Conditional Accept

- Two more weeks and I was able to convince myself that *something like* Peter's bi-trees could be used to build tries in linear time.
- So even if there *were* any errors, it was no longer necessary to infer that they must be *fatal* errors.
- But why ask each of the paper's potential n readers ($n = 10$? 100? 1000?) to duplicate my effort if the author could somehow reduce their burden?
- Decision: Conditional accept.

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973
2. Referee's Report
   - Conditional accept
   - Condition: make more understandable
3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

Condition: make more understandable

- In its present form, even if the paper did contain errors, fixing them wasn't going to solve anything, as that would do nothing to clarify the paper.
- What was needed was to make it clearer why the paper was correct.
- My report offered one way of doing this, not as something the author should follow however but merely as an existence proof that much greater clarity was possible.

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973

2. Referee's Report
   - Conditional accept
   - Condition: make more understandable

3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

Example

- The string BANANAS has 7 symbols numbered 1 to 7 and 8 between-character positions numbered 0 to 7: 0B1A2N3A4N5A6S7.
- At each between-character position *i*, denote the left-longest repetition *so far* (i.e. ignoring text yet to come) by] at *i* and a matching [at $j \leq i$.

0. []BANANAS 0..0
1. B[]ANANAS 1..1
2. BA[]NANAS 2..2
3. BAN[]ANAS 3..3
4. BAN[A]NAS 3..4
5. BAN[AN]AS 3..5
6. BAN[ANA]S 3..6
7. BANANAS[] 7..7

4 repetitions: ε, A, AN, ANA
occurrences: 8, 3, 2, 2
Left longest repetitions are one character short of being a form of suffix identifier.

"[" advances monotonically in this example.

Basic algorithm

while picture is ...[w]a... do
    if wa is a repetition Move "]"
    else Move "["   (and "]" in context "[]")

Theorem

"[" always advances monotonically.

Hence $O(n)$ running time assuming $O(1)$ time steps.

Data structure

Realize the movements of [and] as movements along eges of a graph G constructed as we go along.

Vertices denote the left-longest repetitions.

They are created as soon as the repetition is first bracketed.

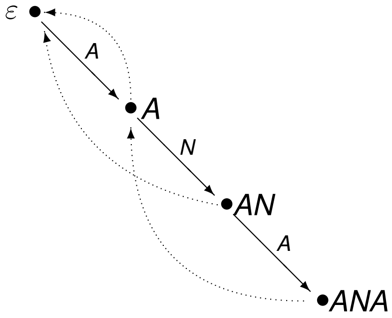
The initial vertex is ε, being considered a repetition from the very beginning.

Edges:

- Link w to wa via an edge labeled a. Notation: $w{:}a = wa$. These edges support] movement.
- Link wa to its longest proper suffix u in G. Notation: $S(w) = u$. These edges support [movement.

Data structure (cont.)

[]BANANAS
B[]ANANAS
BA[]NANAS
BAN[]ANAS
BAN[A]NAS
BAN[AN]AS
BAN[ANA]S
BANAN[A]S
BANANA[]S
BANANAS[]



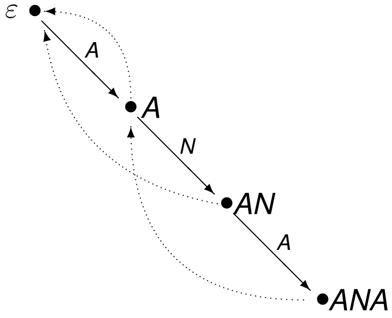
Algorithm
Start at ε: []BANANAS
While picture is ...[w]a...
- if *wa* is a repetition follow the *a* edge, creating *wa* if necessary
- else if the dotted (*S*) edge exists follow it
- else at ε: move [] (i.e. get next character)

Outline

1. Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973
2. Referee's Report
   - Conditional accept
   - Condition: make more understandable
3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication

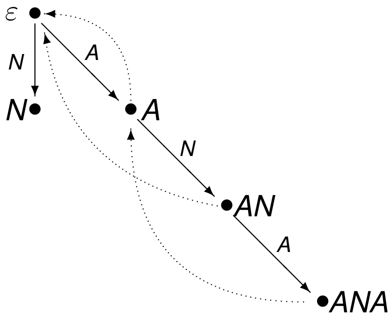
Suffix splitting: BANANASNA

BANANAS[]NA



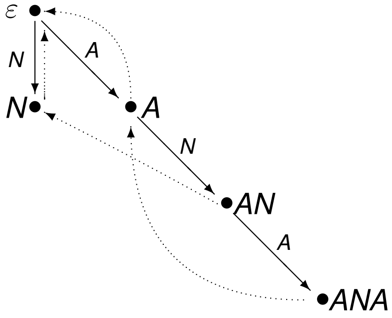
Suffix splitting: Create node N

BANANAS[]NA
BANANAS[N]A



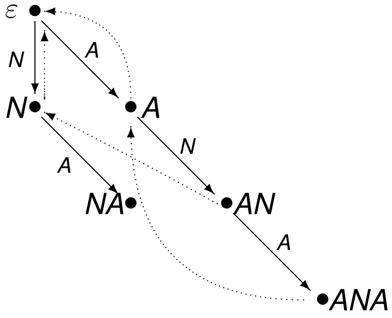
Suffix splitting: Split AN $\rightarrow \varepsilon$

BANANAS[]NA
BANANAS[N]A



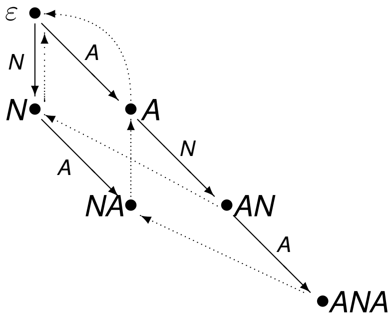
Suffix splitting: Create node NA

BANANAS[]NA
BANANAS[N]A
BANANAS[NA]



Suffix splitting: Split ANA $\to$ A

BANANAS[]NA
BANANAS[N]A
BANANAS[NA]



Suffix splitting: METHOD (outline)

Goal: Split $x \longrightarrow wa \longrightarrow u$
where u is the longest proper suffix of wa in G (always exists)
and x is the word in G if any such that $S(x) = u$ before wa enters G
and $S(x) = wa$ afterwards (x need not exist).

Theorem

If such an x exists it is unique, and is determined by the symbol c in $wa = vcu$.

Hence every suffix link $x \rightarrow u$ can be equipped with an inverse link $u \rightarrow x$ determined by u and the c such that $x = v'cu$. This c can in turn be determined as $c = A[loc(wa) - len(u)]$ since $S(x) = wa$.
Method:

- Find u using w and a.
- Find x using u and c as in the theorem.

Suffix splitting: Finding u and x

We must find $u = S(wa)$ at the creation of each new node wa, whether or not x exists. Do so as follows.

Before linking w to wa, set $t = w$ and then repeatedly set $t = S(t)$ (i.e. follow dotted suffix links) until either ta exists or $t = \varepsilon$. Take u to be ta if it exists, else ε.

This is still $O(n)$ because the next $w = S(w)$ skips over all the steps taken by $t = S(w)$ in a single step.

To find x we furnish every suffix edge $x \to u$ of the graph with its inverse $u \to x$. Although there may be multiple x satisfying $S(x) = u$, only one can "factor through" wa. (Connection with Weiner's algorithm: these are the edges of a compacted suffix trie.)

To find $x = v'cu$, determine c as $A[loc(wa) - len(u)]$.

All fields of a vertex of G

We can now list all 6 fields of a vertex of G.

- loc(w) location of 1st occurrence of w
- len(w) length of w
- $S(w)$ longest proper suffix of w (as a vertex in G)

The above three fields are fixed at the time w is created as $w = v : b$.
loc(w) $= v.b$, len(w) = len(v) $+ 1$, and $S(w) = u$ ($t : vb$ or *varepsilon*).

The remaining 3 fields are Σ-indexed sparse arrays.
For each symbol $a \in \Sigma$:

- $w.a$ Location of the first occurrence of wa (right end). Set at the later of creating w or 1st occurrence of wa (next slide).
- $w{:}a$ Vertex of G denoting wa.
  Set when wa is created (see next slide).
- $*a{:}w$ Inverse suffix link.
  Set when the corresponding suffix link is created (always paired).

Detecting repetitions

In the context ...[w]a..., wa is a repetition when $w.a$ is defined, namely as the location of the first occurrence of wa.

$w.a$ is stored at node w in G either at creation of w or later.

- *At creation:* When w is created in G, record for each symbol a the location of the first occurrence of wa in node a. Notation: $w.a$. To do this, either copy all $x.a$ to $w.a$ when x exists, otherwise set $w.a$ to $\mathrm{loc}(w) + 1$ where a is the letter at that location.
- *Later:* Whenever the repetition test for ...[w]a... fails, set $w.a$ to be the current position in the string.

Main theorem:

Theorem

For all $w \in G$, if wa occurs in the string then $w.a$ = the location of the first occurrence of w.

Application: pattern matching

To find patterns in a string A.

Dumb method (how my referee's report envisaged doing this):

- Apply the algorithm to A.
- For each pattern P *continue* the algorithm with input $\$P$, where $\$$ is a new symbol not in Σ. This produces a graph $G_{A\$P}$ as though having processed $A\$P$ in one pass.
- When done with each P restore $G_{A\$P}$ to G_A (routine).

Relation to Weiner's algorithm

Take VP* to be my variant reversed so both scan right to left.

Essential common feature of PW and VP*: both find prefix identifiers of position.

The edges of PW's compacted suffix trie (i.e. suffix tree) are VP*'s $*a; w$ edges.

Corollary 1: Instead of the above dumb method, VP can do pattern matching without modifying G, namely by scanning the *patterns* right to left and navigating in G via the inverse suffix links $*a : w$ instead of the $w : a$ links.

My report did not make that connection with PW and hence overlooked that possibility.

Corollary 2: VP and PW differ only in implementation details. (This was not clear to me until this morning.)