

+

+

Algorithms and Data Structures for Molecular Biology Problems

Ross McConnell
Computer Science
Colorado State University

+

1

+

+

The problem: The “find” operation available in most word processors

Formally: Given a text T and a search string X , find all the places in T where X occurs as a substring.

+

2

+

+

n : the length of T

m : the length of X

k : the number of occurrences of X in T .

It's easy to see how to solve this problem in time proportional to nm in the worst case.

T:	a	a	b	a	b	c	a	b
X:	a	^x b	c					

+

3

+

+

n : the length of T

m : the length of X

k : the number of occurrences of X in T .

It's easy to see how to solve this problem in time proportional to nm in the worst case.

T: a a b a b c a b
 | | ^X
X: a b c

+

4

+

+

n : the length of T

m : the length of X

k : the number of occurrences of X in T .

It's easy to see how to solve this problem in time proportional to nm in the worst case.

T: a a b a b c a b
 X
X: a b c

+

+

+

n : the length of T

m : the length of X

k : the number of occurrences of X in T .

It's easy to see how to solve this problem in time proportional to nm in the worst case.

T: a a b a b c a b
 | | |
X: a b c

Doing it in time proportional to n and independent of m is possible (Knuth-Morris-Pratt).

+

6

What if T is a large text, such as a chunk of the human genome?

It must be searched repeatedly by algorithms that spawn enormous numbers of search strings.

Time proportional to n for each search is not practical, since n is large.

Today's talk: how to do it in time proportional to $m + k$

This is better because n is enormous and m is short.

The Trick: A preprocessing step to create a network to facilitate searches on T .

The number of substrings of a text

What about storing a dictionary of the substrings?

Text = *abcdab*

Substrings: $\{\lambda, a, b, c, d, ab, bc, cd, da, abc, bcd, cda, dab, abcd, bcda, cdab, abcda, bcdab, abcdab\}$.

The number of substrings grows in proportion to n^2 .

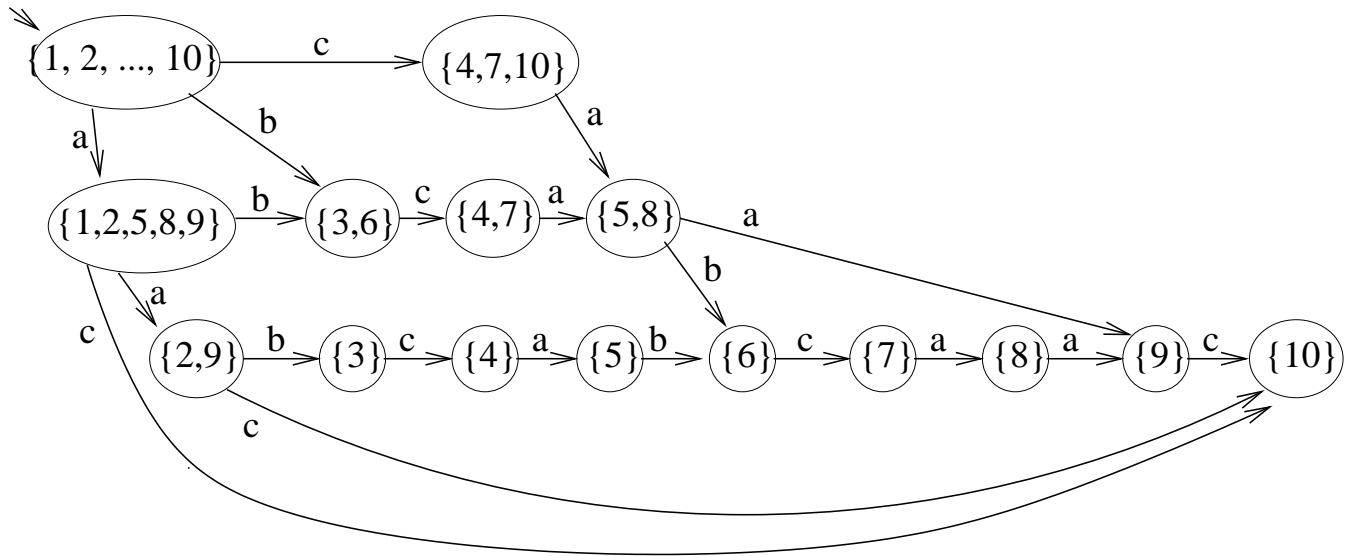
Not practical even for moderately-sized texts.

+

+

A better idea: use a network (DAWG)

(Joint work with Blumer, Blumer, Ehrenfeucht, Haussler)



T: a a b c a b c a a c
 Positions: 1 2 3 4 5 6 7 8 9 10

The numbers are **ending** positions of occurrences.

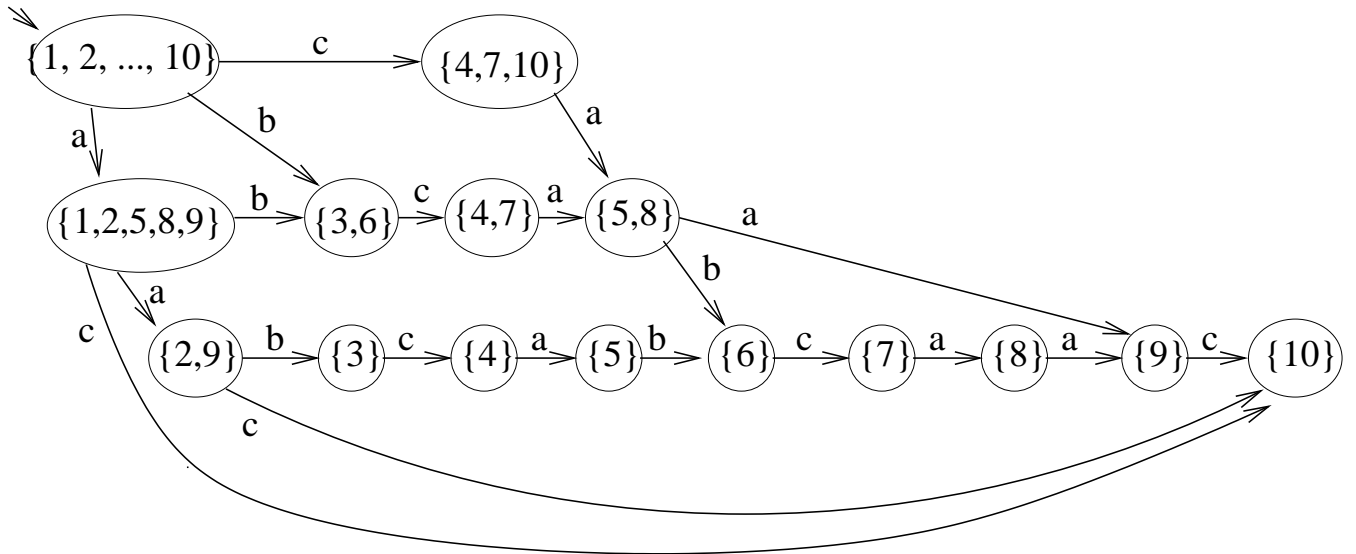
A set of positions is a node of the network if it is the set of ending positions of occurrences of some substring.

Savings: Most nodes are the set of ending positions of **many** substrings.

+

Lemma: If two strings share a position, then one is a suffix of the other.

Corollary 1: If two strings lead to the same node, then one is a suffix of the other.

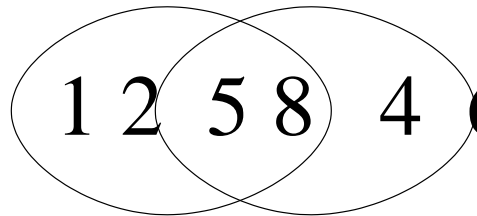


T: a a b c a b c a a c
 Positions: 1 2 3 4 5 6 7 8 9 10

+

+

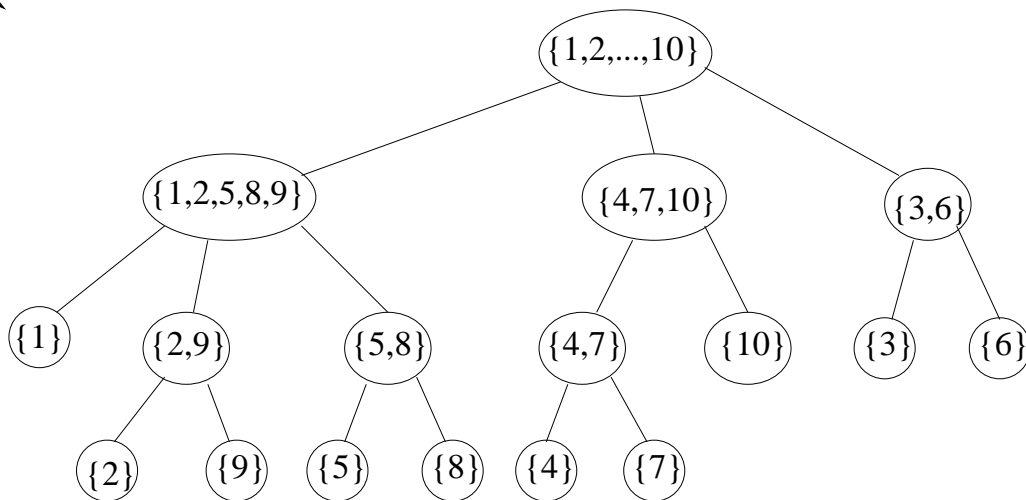
Corollary 2: If Y and Z are the position sets in two nodes, then either one of Y and Z is a subset of the other or they are disjoint.



Theorem: The number of nodes of the DAWG is at most $2n - 1$.

Proof: The subset relation organizes the nodes into a tree with n leaves, where each internal node has at least two children:

⇒



T: a a b c a b c a a c
Positions: 1 2 3 4 5 6 7 8 9 10

+

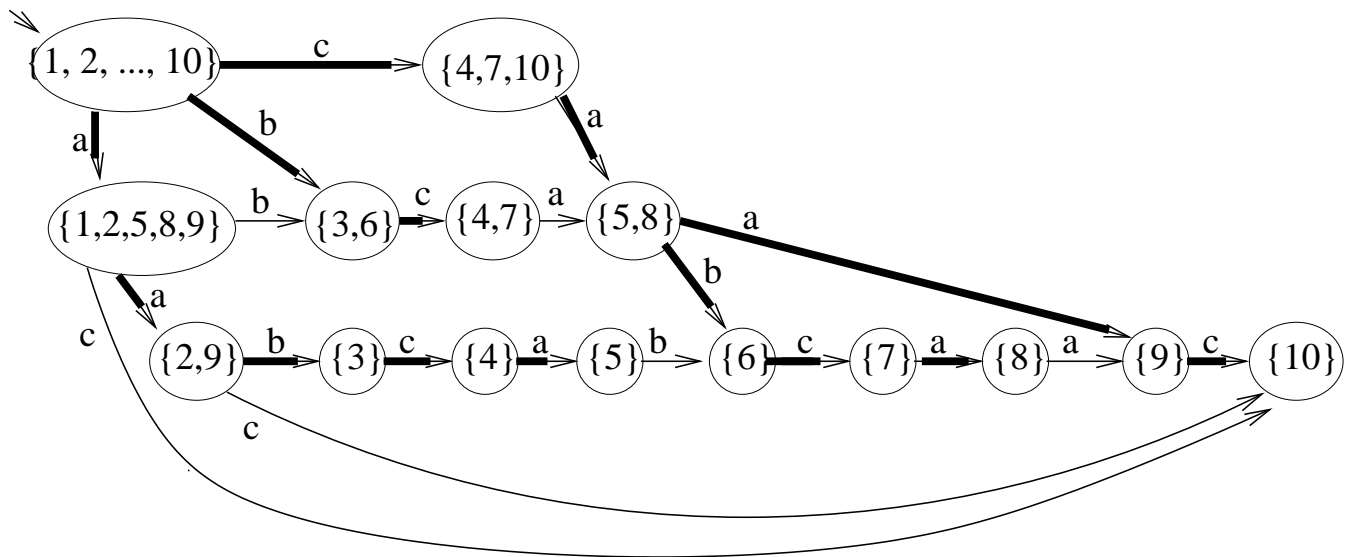
11

+

+

Theorem: The number of edges of the DAWG as at most $3n - 3$.

Proof: Find a rooted spanning tree of the DAWG.

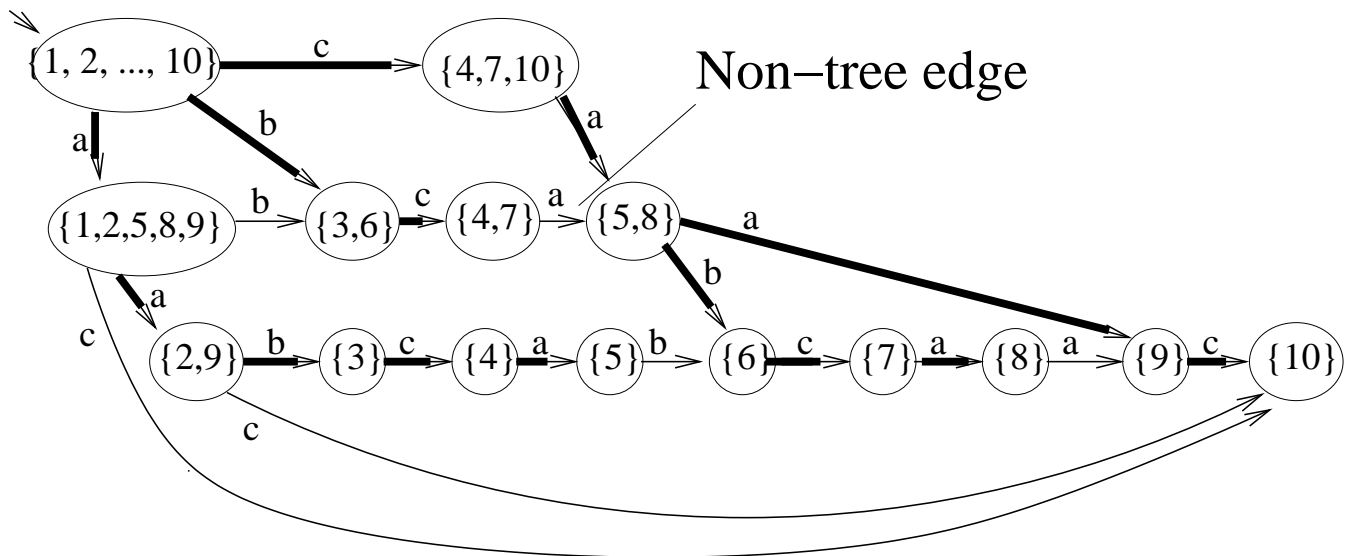


T: a a b c a b c a a c
 Positions: 1 2 3 4 5 6 7 8 9 10

The tree has at most $2n - 3$ edges since the DAWG has at most $2n - 2$ nodes.

+

We must also bound the number of non-tree edges:



T: a a b c a b c a a c
 Positions: 1 2 3 4 5 6 7 8 9 10

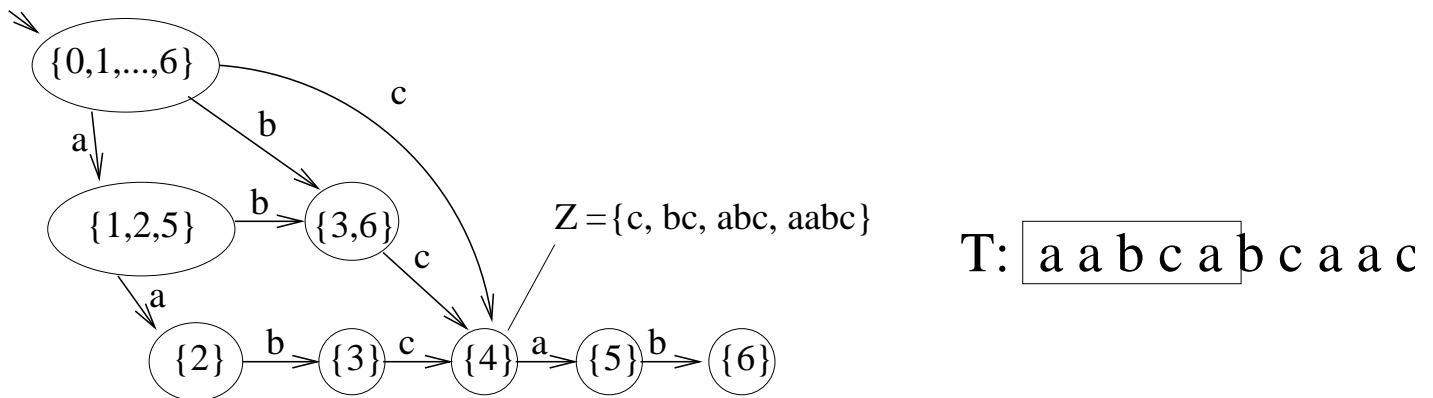
1. Follow tree edges to the tail of e (every node is reachable on tree edges).
2. Traverse the edge and follow any path to the last node, which contains position n .
3. The letters of the path must be a suffix of T . (Why?) Each non-tree edge is associated with a different suffix. (Why?)

+

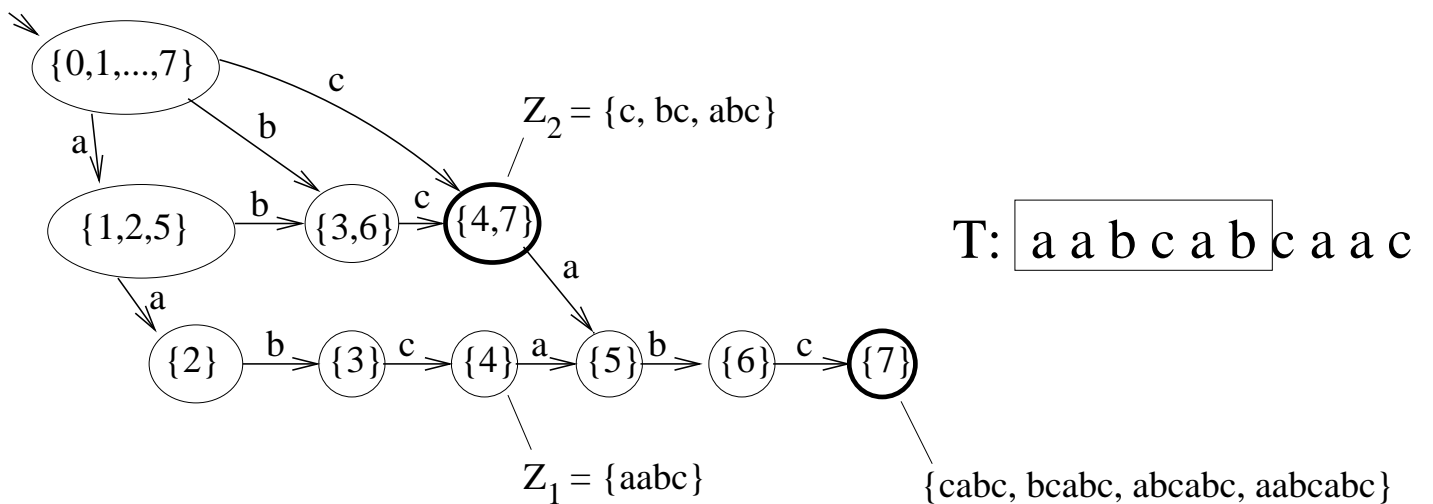
+

Building the DAWG

Build the dawg for larger and larger prefixes of T by adding a new letter, then letting the DAWG mutate to reflect the letters seen so far:



T: a a b c a b c a a c



T: a a b c a b c a a c

Takes time proportional to n , but the algorithm requires sophisticated programming to achieve this.

+

+

+

Sequence Landscapes

+

+

+

Inexact Sequence Landscapes

(Joint with CSU's Asa Ben-Hur, Andy Curtis,
Artem Sokolov, Mark Rogers)

+

Drawback of DAWGS: memory requirements and difficult construction algorithm.

Alternative Structure for Search Problem: Position Heaps

(Joint work with Andrzej Ehrenfeucht)

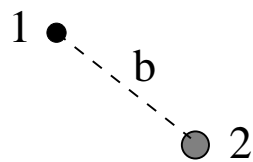
```
10 9 8 7 6 5 4 3 2 1  
b b a b b b a a b a
```

1 ●

Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

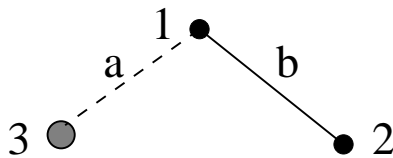
```
10 9 8 7 6 5 4 3 2 1  
b b a b b b a a b a
```



Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

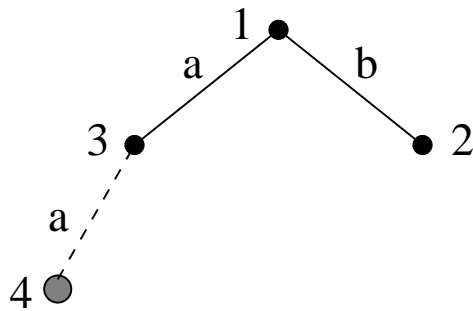
```
10 9 8 7 6 5 4 3 2 1
b b a b b b a a b a
           □
```



Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

10	9	8	7	6	5	4	3	2	1
b	b	a	b	b	b	<u>a</u>	<u>a</u>	b	a

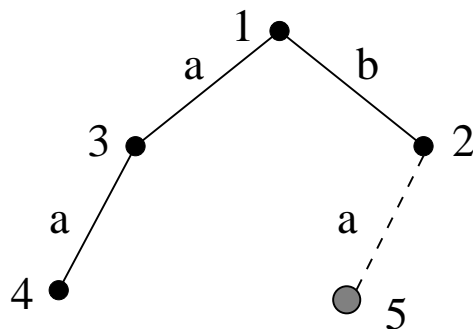


Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

```

10 9 8 7 6 5 4 3 2 1
  b b a b b b a a b a
  
```

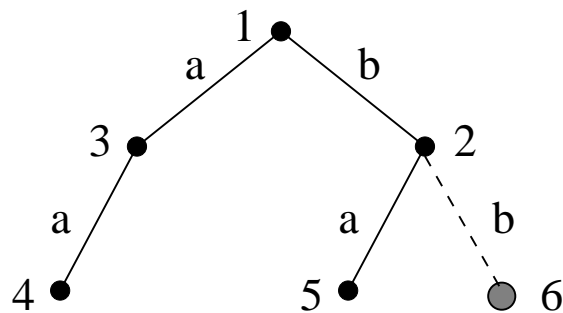


Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

```

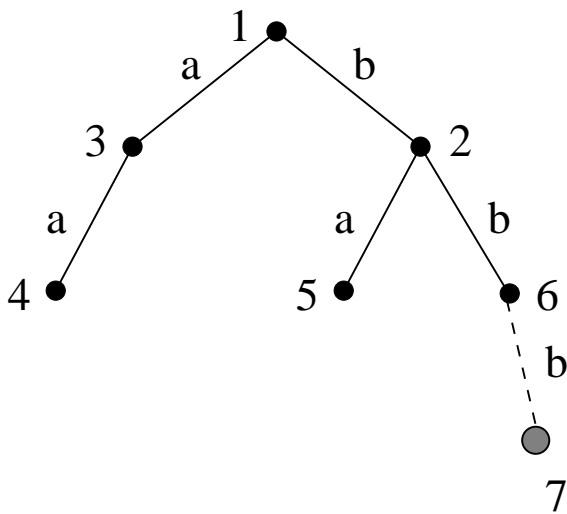
10 9 8 7 6 5 4 3 2 1
  b b a b b b a a b a
  
```



Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

10 9 8 7 6 5 4 3 2 1
b b a b b b a a b a



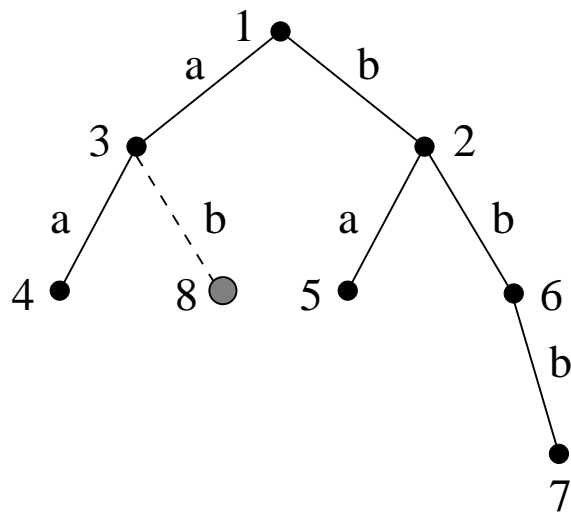
+

+

Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

10 9 8 7 6 5 4 3 2 1
b b a b b b a a b a



+

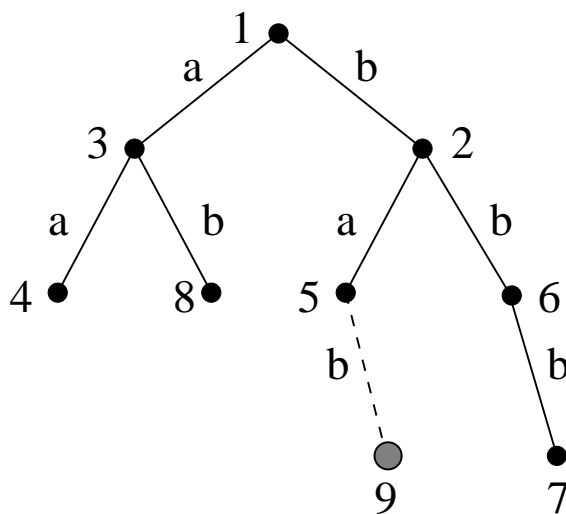
+

+

Drawback of DAWGS: memory requirements and difficult construction algorithm.

Newer Structure to Search Problem: Position Heaps

10 9 8 7 6 5 4 3 2 1
b b a b b b a a b a



+

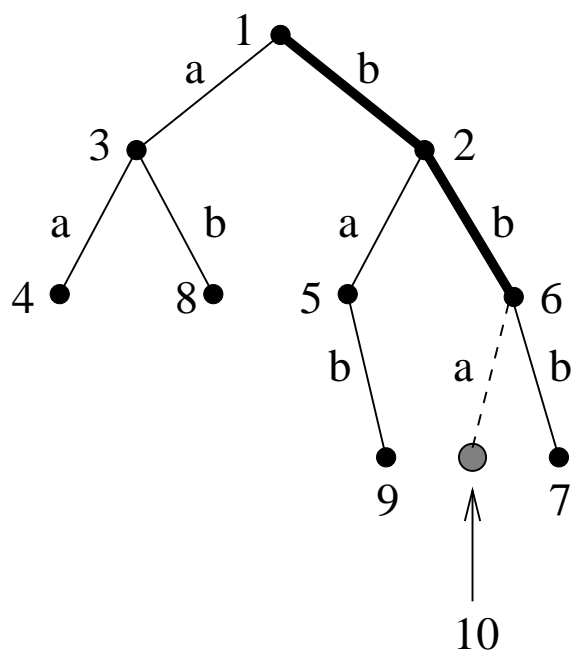
+

+

Drawback of DAWGS: memory requirements, difficult construction algorithm, no efficient algorithm to update it if the text is edited.

Newer Structure to Search Problem: Position Heaps

10 9 8 7 6 5 4 3 2 1
b b a b b b a a b a



+

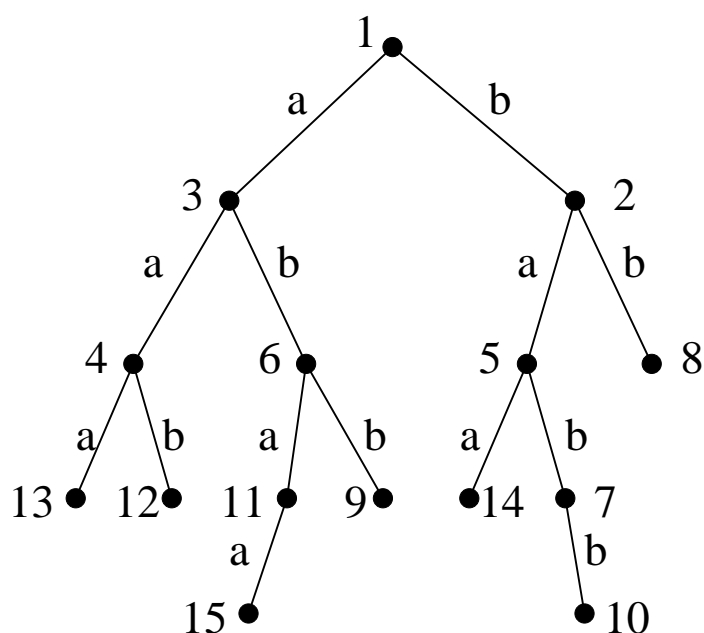
+

+

Searching for X in T using the position heap

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
 T: a b a a a b a b b a b a a b a

Search strings aa and ab



Time of search: proportional to $m^2 + k$. Why?

Ongoing work: modifying the position heap when T is edited. (Joint with Sung-Whan Woo, CSU.)

+