

Tries and suffixes trees

Alon Efrat
Computer Science Department
University of Arizona

Trie: A data-structure for a set of words

All words over the alphabet $\Sigma=\{a,b,\dots,z\}$.
In the slides, the alphabet is only $\{a,b,c,d\}$.
 S – set of words = $\{a,aba, a, aca, addd\}$.
Need to support the operations

- $insert(w)$ – add a new word w into S .
- $delete(w)$ – delete the word w from S .
- $find(w)$ is w in S ?

•Future operation:
•Given text (many words) where is w in the text.

•The time for each operation should be $O(k)$, where k is the number of letters in w

•Usually each word is associated with addition info – not discussed here.

Trie (Tree+Retrieve) for S

- A tree where each node is a struct consist
- Struct node {
 - `char[4] *ar;`
 - `char flag ; /* 1 if a word ends at this node. Otherwise 0 */`

Rule:
Each node corresponds to a word w .
 $w \in S$ iff $flag=1$

A trie - example

$S=\{a,b,dbb\}$

Finding if word w is in the tree

```

p=root; i=0
While(1){
  • If  $w[i] == '\0'$  // we scanned all letters of  $w$ 
    • then return the flag of  $p$ ;
  • If the entry of  $p$  correspond to  $w[i]$  is NULL
    return false;
  • Set  $p$  to be the node pointed by this entry, and set  $i++$ ;
}

```

Inserting a word w

- Try to perform $find(w)$.
 - If runs into a NULL pointers, create new nodes along the path.
 - The $flag$ fields of all new nodes is 0.
- Set the flag of the last node to 1


Deleting a word w

- Find the node p corresponding to w (using 'find' operation).
- Set the flag field of p to 0.
- If p is dead (I.e. $flag=0$ and all pointers are NULL) then $free(p)$, set $p=parent(p)$ and repeat this check.

7

Heuristics for saving space

- The space required is $\Theta(|\Sigma| |S|)$.
- To save some space, if Σ is larger, there are a few heuristics we can use. Assume $\Sigma=\{a,b..z\}$.
- We use two types of nodes
 - Type "A", which is used when the number of children of a node is more than 3

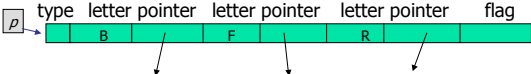


Note – the letters are not stored explicitly

8

Heuristics for space saving

- Type "B" is used if there are 3 or less children:
- The "letter" of the child is also stored:

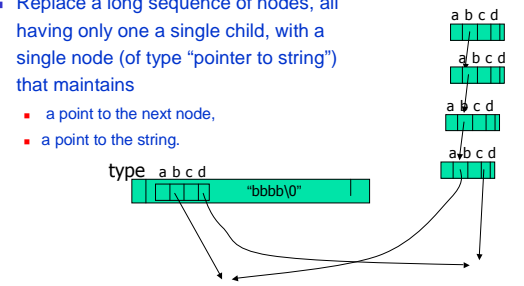


- The rule of the flag is the same as in type "A" nodes.
- We only store the 3 pointers, but we need to know to which letters they corresponds to.

9

Another Heuristics – path compression

- Replace a long sequence of nodes, all having only one a single child, with a single node (of type "pointer to string") that maintains
 - a point to the next node,
 - a point to the string.



10

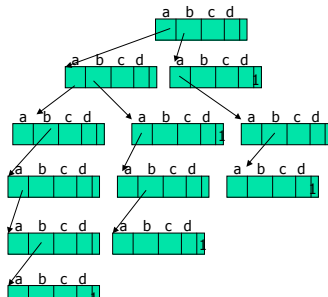
Suffix tree.

- Assume B (for book) is a long text.
- Want to preprocess B , so when a word w is given, we can quickly find if it is in B .
- We can find it in $O(|w|)$.
- Idea:
 - Consider B as a long string.
 - Create a trie T of all suffixes of B .
 - In addition to the flag (specifying if a word ends at node), we also stored the index in B where this word begins.
 - Example $B="aabab"$
 $S={"aabab", "abab", "bab", "ab", "b"}$

11

Suffix tree.

Example $B="aabab"$ $S={"aabab", "abab", "bab", "ab", "b"}$



To know where a word appear in B , we store with each node the index of the beginning of the suffix in B .

(we can store only the first appearance of the word in the text)

12

Size of suffix tree

Example $B = \text{"aabab"}$ $S = \{\text{"aabab"}, \text{"abab"}, \text{"bab"}, \text{"ab"}, \text{"b"}\}$

Assume $n = |B|$.
 Total length of all string $\Theta(n^2)$
 Size of a node is $|\Sigma|$
 So size of the tree is $\Theta(n^2 |\Sigma|)$.

Time to construct the tree $\Theta(n^2)$

We can save some space.

Example $B = \text{"aabab"}$
 $S = \{\text{"aabab"}, \text{"abab"}, \text{"bab"}, \text{"ab"}, \text{"b"}\}$

13

Suffix tries on a diet

Def: a *shred* is a path from node u to node v in the trie, consisting of nodes of outdegree 1 (except maybe the last one) and $flag=0$.

Obs: There is a contiguous part of B , identical to the string the shred represents. We call this part the shred-string

We store B itself as an array.

We use a new type of nodes, called shred-nodes, maintain the first ($id1$) and last ($id2$) indexes of the shred-string in B .

type	a	b	c	d	id1	id2	flag
					7	10	

$B = \text{"cabd}^1\text{da}^7\text{adb}^10\text{d}^14$

Suffix tries on a diet - cont

Algorithm for constructing a "thin" trie:

Given B – create an empty trie T , and insert all n suffixes of B into T -- generating a trie of size $\Theta(n^2)$.

Traverse the tries, and each time that a shred is seen, replace all nodes of the shred with a single shred-node.

15

Suffix tries on a diet - cont

Clearly the use of shred nodes saves some-but can we prove something ?

Observations: Every the number of leaves of T is at most n (every leaf is the end of one prefix)

16

Suffix tries on a diet - cont

Lemma: Let T be a tree where each internal node has outdegree 2 or more, and m leafs. Then T has at most m internal nodes.

Back to thin suffix tries: T does not have **exactly** this property, but it is very close (no long shreds), so a "massaged" lemma still works, so

$\#internal_nodes \leq \#leafs_nodes$,

But $\#leafs_nodes \leq \#suffixes_of_B = n$

So the size of the trie is only a constant more than the size of the book.

17