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 $\Sigma = \{a,b,..z\}$.
In the slides, let say that the alphabet is only $\{a,b,c,d\}$
$S$ – set of words = $\{a,aba, a, aca, adds\}$
Need to support the operations
• $\text{insert}(w)$ – add a new word $w$ into $S$.
• $\text{delete}(w)$ – delete the word $w$ from $S$.
• $\text{find}(w)$ 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.
(which is in $S$ if the flag is 1)
A trie - example

Finding if word \( w \) is in the tree

Inserting a word \( w \)
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.

Heuristics for space saving

- The space required is $\Theta(|\Sigma||S|)$.
- To save some space, if $\Sigma$ 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.
  - Type "B" is used if there are 3 or less children: The "letter" of the child is also stored:
    
    
    | type | a   | b   | letter pointer | flag |
    |------|-----|-----|----------------|------|
    | A    |     |     |                |      |
    | B    |     |     | letter pointer | flag |

    Note – the letters are not stores explicitly.

- 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 correspond to.
Another Heuristics – path compression

- Replace a long sequence of nodes that happens to have only one child, with a single node (of type “pointer to string”) that keeps a point to the next node, and a point to a string.

Suffix tree.

- Assume \( B \) (for book) is a long text.
- Want to preprocess \( B \), so when a word \( w \) is given, we could quickly find if it is in \( B \).
  - (as well as locations, how many etc)
- 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 \( B=\text{"aabab"} \)
  - \( S=\{\text{"aabab"}, \text{"abab"}, \text{"bab"}, \text{"ab"}, \text{"b"}\} \)
  - 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)
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 \( \Theta(n^2) \)

Rather than a flag, we store the first index where the suffix appear

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 \( \Theta(n^2) \)

In addition to the flag, we store the first index (in the book) where the suffix starts (in red)

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 stores \( B \) itself as an array.
We use a new type of nodes, called shred-nodes, that maintain only the indexes of the first (\( id_1 \)) and last (\( id_2 \)) letters of the shred-string in \( B \).
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.

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

Observations: The number of leaves of $T$ is at most $\leq n$ (every leaf is the end of one prefix).
In addition there are nodes have a single child, but their flag=1 (a suffixed have ended). We call them special nodes.

Observations: There are $\leq n$ special nodes.

Lemma: Let $T$ be a tree where each internal node has outdegree 2 or more, and $m$ leaves. 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

$\#\text{internal}\_\text{nodes} \leq \#\text{leafs}\_\text{nodes} + \#\text{special}\_\text{nodes}$

But $\#\text{leafs}\_\text{nodes} + \#\text{special}\_\text{nodes} \leq \#\text{suffixes}\_\text{of}\_B = n$

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

Lemma: Let $T$ be a tree where each internal node has outdegree 2 or more, and $m$ leaves and $k$ internal nodes. Then $k \leq m$.

Proof: Assume true for all trees with strictly less than $m$ leaves, and assume $T$ has $m$ leaves.

Find a leaf $u$ whose distance from root is maximum. Assume it has exactly one sibling $v$. Note that $v$ is a leaf (why?). Let $w$ be their common parent.

Remove both $u$ and $v$ from $T$. Let $T'$ be the resulting tree. Let $k'$, $m'$ denote # internal nodes and leaves in $T'$. Now in $T'$:

1. $w$ is a leaf.
2. $m' = m - 2 + 1 = m - 1$.
3. $k' = k - 1$.
4. The outdegree of every internal node $\geq 2$.

From induction, $k' \leq m'$. Hence $k \leq m$. 

\[ \text{Proof of lemma} \]