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, 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 additional 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

The label of an edge is the label of the cell from which this edge exits

Corresponding to $w=“d”$

The dictionary contains $S=\{a,b,dbb\}$

Corr. To $w=“dbb”$ (not in $S$, flag=0)

Corr. to $w=“dbb”$
Finding if word \textit{w} is in the tree

\begin{itemize}
\item \texttt{p=root; i=0} // remember - each string ends with `\0'
\item While(1){
  \begin{itemize}
  \item If \texttt{w[i] == \0} // we have scanned all letters of \textit{w}
  \quad then return the flag of \texttt{p}; \textbf{else}
  \item If \texttt{(p.a[w[i] - 'a']) == NULL} // the entry of \texttt{p} correspond to \texttt{w[i]} is NULL
    \quad return \textbf{false};
  \item \texttt{p = (p.a[w[i] - 'a'])} // Set \texttt{p} to be the node pointed by this entry
  \item \texttt{i++;}
  \end{itemize}
}\end{itemize}

Inserting a word \textit{w}

\begin{itemize}
\item Try to perform find(\textit{w}).
\item If runs into a NULL pointers, create new nodes along the path.
\item The flag fields of all new nodes is 0.
\item Set the flag of the last node to 1
\end{itemize}

Deleting a word \textit{w}

\begin{itemize}
\item Find the node \texttt{p} corresponding to \textit{w} (using `find' operation).
\item Set the flag field of \texttt{p} to 0.
\item If \texttt{p} is dead (I.e. flag==0 and all pointers are NULL) then
  \texttt{free(p)}, set \texttt{p=parent(p)} and repeat this check.
\end{itemize}

Heuristics for saving space

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

\begin{itemize}
\item \texttt{p}
\item Type \texttt{a b}
\item Flag
\end{itemize}

\textbf{Note –} the letters are not stores explicitly
Heuristics for space saving

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

```
P
```

- 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, 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.

Suffix tree.

- Assume B (for book) is a very 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”}

Observation: w appears in B ⇔ w is the prefix of a suffix of B. Example: B=“helloniceword”, w=“nice”.

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 \( B = \text{“aabab”} \) \( S = \{ \text{“aabab”, “abab”, “bab”, “ab”, “b”} \} \)

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 \( B = \text{“aabab”} \) \( S = \{ \text{“aabab”, “abab”, “bab”, “ab”, “b”} \} \)

Suffix tries on a diet

Def: a thread 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 contagious part of \( B \), identical to the string the shred represents. We call this part the shred-string.

We store the book \( B \) itself as an array.
We use a new type of nodes, called thread-nodes, maintain the first (\( id1 \)) and last (\( id2 \)) indexes 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.

Suffix tries on a diet - cont

- Clearly the use of thread-nodes saves some—but can we prove something?
- Observations: Every leaf of \( T \) must be the end of some prefix of \( B \). So the number of number of leaves of \( T \) is \( \leq n \).
- \( n = |B| \)
- To bound the size of \( T \), we will need to bound the number of internal nodes.
- Observations:
  - \( T \) might contain special nodes whose flag=1 (a suffix terminates at these nodes).
  - The number of special nodes is \( \leq n \) (since this is the number of suffixes).
- What about other internal nodes of \( T \)?
Lemma: Let $T'$ be a rooted tree with $m$ leaves, where each internal node has \( \geq 2 \) children. Then $T'$ has \( \leq m \) internal nodes. (proof - easy induction. Homework)

Back to thin suffix tries $T$:

- $T$ has \( \leq n \) special nodes (with flag=1) and
- $T$ has \( \leq n \) leaves.
- Every other nodes has \( \geq 2 \) children. (with flag=1). Applying the Lemma in this case, implies that the total number of internal nodes \( \leq 2n \).

**Conclusion:** The number of nodes in $T$ is \( \leq 3n \) (much better than the uncompressed version that could have $\Theta(n^2)$ nodes.

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