Tries

Yufei Tao
KAIST

April 17, 2013
In this lecture, we will discuss the following exact matching problem on strings.

**Problem**

Let $S$ be a set of strings, each of which has a unique integer id. Given a query string $q$, a query reports:

- the id of $q$ if it exists in $S$
- nothing otherwise.

**Example**

Suppose that $S = \{aaabb, aab, aabaa, aabab, aba, abbb, abbbba, abbbbb\}$. Let the ids of these strings be (from left to right) 1, 2, ..., 8, respectively. Given $q = aabaa$, a query returns id 3, whereas given $q = abab$, it returns nothing.
Think

How is this problem related to inverted indexes and search engines?
Let

- $A$ be the alphabet (i.e., every character of any string must come from $A$).
- $|s|$ be the length of a string $s$, i.e., the number of characters in $s$.
- $m = |S|$, i.e., the number of strings in $S$.
- $n = \text{the total length of the strings in } S$, i.e., $n = \sum_{s \in S} |s|$.

When $|A|$ is small and all strings in $S$ are short (e.g., $|s| \leq 10$ for all $s \in S$), the exact matching problem on strings can be reduced to exact matching on integers. For example, consider that each string $s$ represents an English word, and that every $s$ has length at most 10. We can map $s$ to an integer from 0 to $26^{10} - 1$.

**Think**

Why does the method no longer work if $|A|$ is large or strings can be arbitrarily long?
Next, we will describe another solution based on a data structure called trie. First, let us define the concept of prefix. Let $s$ be a string of length $t$. We can write its characters (from left to right) as $s[1], s[2], ..., s[t]$, respectively. Then, for any $i \in [1, t]$, the string formed by the sequence $s[1], ..., s[i]$ is called a prefix of $s$. Specially, an empty string $\emptyset$ is also a prefix of $s$.

**Example**

$s = \text{aabaa}$ has 6 prefixes: $\emptyset$, $a$, $aa$, $aab$, $aaba$, and $aabaa$.

Let $S$ be a set of strings. We say that a string $s$ is a possible prefix of $S$ if $s$ is a prefix of at least one string in $S$. 
A set $S$ of strings is called **prefix-free** if no string in $S$ is a prefix of any other string in $S$. Every set of strings can be made prefix-free by appending a special “termination symbol” to each string in $S$.

**Example**

Let $S = \{\text{aaabb, aab, aabaa, aabab, aba, abbb, abbbba, abbbbb}\}$. We can convert $S$ to $S' = \{\text{aaabb }\bot, \text{aab }\bot, \text{aabaa }\bot, \text{aabab }\bot, \text{aba }\bot, \text{abbb }\bot, \text{abbbba }\bot, \text{abbbbb }\bot\}$, which is prefix-free.

From now on, we will consider that $S$ is prefix-free, and that every string in $S$ ends with $\bot$. 
The trie on $S$ is a tree $T$ defined as follows:

- Each node $u$ of $T$ corresponds to a distinct possible prefix of $S$. Let $P(u)$ be the prefix that $u$ represents.

- Let $u$ be a node, and $v$ a child node of $u$. Then:
  - $P(u)$ is a prefix of $P(v)$.
  - $|P(v)| = |P(u)| + 1$.

- Each node $u$ is labeled with a character $c$, which is the last character of $P(u)$. 
Example: Let \( S = \{aaabb \perp, aab \perp, aaba \perp, aabab \perp, aba \perp, abbb \perp, abba \perp, abbb \perp\} \). The trie is:

\[
\begin{array}{c}
\emptyset \\
\downarrow \\
a \\
\downarrow \\
a \\
\downarrow \\
a \\
\downarrow \\
a \\
\downarrow \\
b \\
\downarrow \\
b \\
\downarrow \\
b \\
\downarrow \\
a \\
\downarrow \\
b \\
\downarrow \\
\perp \\
\perp \end{array}
\]

Note that every \( \perp \)-node \( u \) corresponds to a distinct string \( s \in S \). We therefore store the id of \( s \) at \( u \).
Lemma

The trie on $S$ has at most $n + 1$ nodes.
How do we answer an exact matching query with $q = aabaa$? How about $q = abab$?
How to delete the string $aaabb\perp$? How about inserting $ababb\perp$?
Notice that the efficiency of queries, insertions and deletions depends on how well we can solve the following problem:

Given a node $u$ and a character $\sigma \in A \cup \{\bot\}$, how to find the child of $v$ of $u$ that corresponds to $\sigma$?

Different tradeoffs exist:

- By organizing the child nodes of $u$ in an array, we can find $v$ in $O(1)$ time, but the array occupies $O(|A|)$ space.

- By organizing the child nodes of $u$ in a binary search tree (BST), we can find $v$ in $O(\log |A|)$ time, and the tree occupies $O(|f|)$ space, where $f$ is the number of child nodes of $u$. 
Theorem

- By using the array implementation, a trie occupies $O(|A|n)$ space, answers a query with string $q$ in $O(|q|)$ time, and supports the insertion and deletion of a string $s$ in $O(|A||s|)$ time.

- By using the BST implementation, a trie occupies $O(n)$ space, answers a query with string $q$ in $O(|q|\log |A|)$ time, and supports the insertion and deletion of a string $s$ in $O(|s|\log |A|)$ time.
Next, we will describe another trie variant, called balanced trie, which occupies $O(n)$ space, and answers a query with string $q$ in $O(\log m + |q|)$ time. The trie, however, is static, namely, it does not support insertions and deletions.
From now on, we consider that $S$ is sorted alphabetically (placing $\perp$ before all characters of $A$). In general, given a set $S'$ of $x$ sorted strings, we refer to the one in $S'$ whose rank is $\lceil x/2 \rceil$ as the median of $S'$.

**Example**

The median of \{aaabb$\perp$, aab$\perp$, aabaa$\perp$, aabab$\perp$, aba$\perp$, abbb$\perp$, abbba$\perp$, abbbb$\perp$\} is aabab$\perp$.

Furthermore, given a prefix $p$, denote by $S(p)$ the set of strings in $S$ with prefix $p$.

**Example**

Let $S = \{aaabb\perp$, aab$\perp$, aabaa$\perp$, aabab$\perp$, aba$\perp$, abbb$\perp$, abbba$\perp$, abbbb$\perp$\}. Then $S(aab) = \{aab\perp$, aabaa$\perp$, aabab$\perp$\}.
We also need to define what it means by \textit{concatenation}. The concatenation of two strings \(s_1\) and \(s_2\) forms a string by appending the characters of \(s_2\) at the end of \(s_1\).

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
\textbf{Example} \\
\hline
\textbf{If } \(s_1 = ab\text{ and } s_2 = bba\), then concatenation gives \(abbba\). \textbf{If } \(s_1 = \emptyset\text{ and } s_2 = bba\), then concatenation gives \(bba\). \textbf{Similarly, if } \(s_1 = ab\text{ and } s_2 = \emptyset\), concatenation gives \(ab\). \\
\hline
\end{tabular}
\end{table}
Let $S$ be a set of strings. The balanced trie on $S$ is a tree $T$ defined as follows:

- Every node $u$ in $T$ corresponds to a set $S(u)$ of strings, and carries a label $L(u)$ and a positional index $I(u)$, which will be formally defined below.
- If $\sigma$ is the median of $S(u)$, then $L(u) = \sigma[i]$, where $i = I(u)$. Denote by $p$ the length-$i$ prefix of $\sigma$.
- If $u$ is the root, $S(u) = S$, and $I(u) = 1$.
- $u$ is a leaf if $|S(u)| = 1$ and $I(u) = |s|$, where $s$ is the (only) string in $S(u)$.
- An internal $u$ has at most 3 child nodes $u_<$, $u_\equiv$, and $u_>$ such that:
  - $S(u_<)$ is the set of strings in $S(u)$ alphabetically less than $p$. $I(u_<) = I(u)$.
  - $S(u_\equiv)$ is the set of strings in $S(u)$ that have $p$ as a prefix. $I(u_\equiv) = I(u) + 1$.
  - $S(u_>)$ is the set of remaining strings in $S(u)$. $I(u_>) = I(u)$. 
Example: Let \( S = \{aaabb\bot, aab\bot, aabaa\bot, aabab\bot, aba\bot, abbb\bot, abbb\bot, \} \). The balanced trie is:

Each node \( u \) is denoted in the form \((L(u), I(u))\).
How do we answer an exact matching query with $q = aabaa$? How about $q = abab$?
Theorem

A balanced trie occupies $O(n)$ space, and answers a query with string $q$ in $O(\log m + |q|)$ time.