Multiset representation of objects in information retrieval systems

Mikita Akulich1
89172019@student.upr.si

Matjaž Krnc1,2
matjaz.krnc@sbg.ac.at

Iztok Savnik1
itzok.savnik@famnit.upr.si

Riste Škrekovski1,3,4
riste.skrekovski@famnit.upr.si

1 University of Primorska, FAMNIT, Koper, Slovenia
2 University of Salzburg, Department of Efficient Algorithms, Salzburg, Austria
3 Faculty of Information Studies, Novo Mesto, Slovenia
4 University of Ljubljana, FMF, Ljubljana, Slovenia

ABSTRACT
In this paper we present multiset-trie – a novel data structure which operates on objects represented as multisets. The multiset-trie is a search-tree-based data structure with properties similar to those of a trie. In particular, we efficiently implement the standard search tree operations together with the special set containment operations, i.e., subset and superset queries in the context of multisets. These are called submultiset and supermultiset, respectively, and are used for implementation of various queries that can be performed on multisets in a multiset-trie. The corresponding running times of the developed functions are mathematically and experimentally analyzed. One of the most important queries is the search of the nearest neighbor given an input object. The nearest neighbor search of a multiset-trie makes it a good alternative for the index data structures that are used in information retrieval systems. In particular, our research is focused on the application of the multiset-trie to full-text search systems.

Keywords
multiset, trie, multiset-trie, information retrieval, inverted index, full-text search

1. INTRODUCTION
Information retrieval (IR) systems are most commonly used for searching a text-based content such as text documents in a database. Such IR systems are called full-text search systems. The application of the full-text search techniques on the database almost always requires an index for a good performance of the queries. Indexes narrow down the search using pre-generated meta data constructed from the data in the database.

In IR most systems use the concept of an inverted index to achieve full-text indexing of a database. The so-called dictionary of the inverted index can be organized in different ways in order to meet the required types of queries and specification of data. In our research we will be focused on search trees [2, 4, 5, 7].

The proposed data structure multiset-trie can be used as an alternative implementation of the dictionary in an inverted index. It is an extension of the set-trie data structure proposed by Savnik [6]. Set-trie is used for storing and fast retrieval of objects represented as sets and provides the nearest-neighbor search by implementing methods that perform set-containment queries. Multiset-trie extends the abilities of set-trie and provides support for storing and retrieving objects that can be represented as both sets and multisets.

2. MULTISET-TREIE DATA STRUCTURE
2.1 Description
Let \( \Sigma \) be a set of distinct symbols that define an alphabet and let \( \sigma \) be the cardinality of \( \Sigma \). Define \( \phi \) to be a bijective mapping \( \phi : \Sigma \rightarrow I \), where \( I = \{1, 2, 3, \ldots, \sigma\} \). In this way, we obtain an indexing of elements from the alphabet \( \Sigma \), so we can work directly with integers rather then with specific symbols from \( \Sigma \).

The multiset-trie \( \mathcal{M} \) is an \( n \)-ary tree based data structure with the properties of trie. A node in multiset-trie always has out-degree \( n \). Some of the child nodes may be Null. Some of the child nodes may be Null, but at most \( n-1 \) of them. All the children of a node, including the Null children, are uniquely labeled from left to right with labels \( c_j \), where \( j \in \{0, 1, \ldots, n-1\} \).

The height of a multiset-trie is always \( \sigma + 1 \) if the structure is not empty. Levels in multiset-trie are enumerated by their height, i.e., a level \( L_i \) has height \( i \), where \( L_1 \) is the root level. The levels \( L_i \) and symbols from \( \Sigma \) are related as follows. For \( i \in \{1, 2, \ldots, \sigma\} \), a level \( L_i \) represents a symbol \( s \in \Sigma \), such that \( \phi^{-1}(i) = s \). The last level \( L_{\sigma+1} \) does not represent any symbol and is named leaf level (LL for short).

Consider two nodes \( u, v \) in a multiset-trie at levels \( L_i, L_{i+1} \), respectively.
respectively. Let a node \( u \) be a parent node of a node \( v \). Suppose that a child node \( v \) is not \( \text{Null} \) and has a label \( c_j \). Then the path \( u \rightarrow v \) represents a symbol \( \phi^{-1}(i) \in \Sigma \) with multiplicity \( j \). Such a transition \( u \rightarrow v \) is called a path of length \( 1 \) and is allowed if and only if a node \( v \) is not \( \text{Null} \) and \( u \) is a parent node of a node \( v \).

We define a complete path to be the path of length \( \sigma \) in a multiset-trie with the end points at root node (the 1st level) and \( LL \). Thus, a multiset \( m \) is inserted into a multiset-trie if and only if there exists a complete path in a multiset-trie that corresponds to \( m \). Note that every complete path in a multiset-trie is unique. Therefore, the multisets that share a common prefix in a multiset-trie can have a common path of length at most \( \sigma - 1 \). Thus, any multiset \( m \) that is composed of symbols from \( \Sigma \) with maximum multiplicity not greater than \( n - 1 \) can be represented by a complete path in a multiset-trie.

Let us have an example of a multiset-trie data structure, where \( \text{Null} \) children are omitted. Let a pair

![Multiset-Trie Diagram](image)

**Figure 1:** Example of multiset-trie structure.

\((L_i, c_j)\) represents a node with label \( c_j \) at a level \( L_i \). The pair \((L_i, c_j)\) is equivalent to \((L_1, \text{root})\), for every \( j \). According to the figure 1 we can extract inserted multisets as follows:

\[
\begin{align*}
(L_1, \text{root}) &\rightarrow (L_2, c_0) \rightarrow (LL, c_0) \equiv \{1^0, 2^0\} = \emptyset \\
(L_1, \text{root}) &\rightarrow (L_2, c_0) \rightarrow (LL, c_2) \equiv \{1^0, 2^2\} = \{2, 2\} \\
(L_1, \text{root}) &\rightarrow (L_2, c_1) \rightarrow (LL, c_2) \equiv \{1^1, 2^3\} = \{1, 2, 2\} \\
(L_1, \text{root}) &\rightarrow (L_2, c_2) \rightarrow (LL, c_2) \equiv \{1^2, 2^2\} = \{1, 1, 2, 2\}
\end{align*}
\]

where \( e^k \) represents an element \( e \) with multiplicity \( k \).

### 2.2 Operations

**Basic tree operations.** The procedure \( \text{insert}(M, m) \) inserts a new instance \( m \) of type Multiset into multiset-trie \( M \). If the complete path already exists, then procedure leaves the structure unchanged. Otherwise it extends partially existing or creates a new complete path. The procedure does not return any result.

The function \( \text{search}(M, m) \) checks if the complete path corresponding to a given multiset \( m \) exists in the structure \( M \). The function returns true if the multiset \( m \) exists in \( M \), and returns false otherwise.

The function \( \text{delete}(M, m) \) searches for the complete path that corresponds to \( m \) in order to remove it. If the path can not be found, the function immediately returns false. During search, the function collects additional data about common prefixes. When search is completed, the function removes the branch preserving common prefixes, and returns true.

**Sub- and super-multiset existence.** The \( \text{submsetExistence}(M, m) \) function checks if there exists a multiset \( x \) in \( M \), that satisfies the condition \( x \subseteq m \). The function starts with searching for an exact match \( x = m \) in \( M \), since \( m \subseteq m \) by definition of submultiset inclusion. If an exact match is not found in \( M \), the function uses multiset-trie to find the closest (the largest) submultiset of \( m \) in \( M \) by decreasing multiplicity of elements in \( m \). At every level the function tries to proceed with the largest possible multiplicity of an element that is provided by \( m \). However, when the function reaches some level where it meets a \( \text{Null} \) node and can not go further using path provided by \( m \), it decreases the multiplicity of an element that corresponds to a current level. Thus, the function can decrease multiplicity of an element or eventually skip it in order to find the closest \( x \subseteq m \).

The function \( \text{supermsetExistence}(M, m) \) checks if there exists supermultiset \( x \) of a given multiset \( m \) in \( M \). By analogy to the function \( \text{submsetExistence} \), the function \( \text{supermsetExistence} \) starts by searching for an exact match \( x = m \) in \( M \). If an exact match is not found in \( M \), the function searches for the closest (the smallest) supermultiset \( x \) of \( m \) in \( M \), in this case, by increasing multiplicity of elements in \( m \).

**Get all sub- and super-multisets.** The algorithms for functions \( \text{getALLSubmsets} \) and \( \text{getALLSupermsets} \) are based entirely on algorithms for \( \text{submsetExistence} \) and \( \text{supermsetExistence} \) functions that do not terminate on the first existing sub/supermultiset, but store the results and continue procedure until all existing sub/supermultisets in \( M \) are found and stored.

**2.3 Analysis of the multiset-trie**

By the design of the multiset-trie, it is easy to see that the functions \( \text{search}, \text{delete} \) and the procedure \( \text{insert} \) have complexity of \( O(\sigma) \). Because \( \sigma \) is defined when the structure is initialized and does not depend on the user input afterwards, the asymptotic complexity of the functions \( \text{search}, \text{delete} \) and the procedure \( \text{insert} \) is \( O(1) \). Nonetheless, in the general case the complexity is \( O(\sigma) \).

In what follows, we focus on analysis of the more involved functions which correspond to a multiset-containment queries: \( \text{submsetExistence}, \text{supermsetExistence}, \text{getALLSubmsets} \) and \( \text{getALLSupermsets} \).

**Mathematical model.** We start with the basics of our mathematical model. Let \( \Sigma \) be an alphabet of cardinality \( \sigma \), such that \( \Sigma = \{1, 2, \ldots, \sigma\} \). Define \( N \) to be the power multiset of \( \Sigma \) with respect to \( n \), where \( n \) is the maximal degree of a node in multiset-trie. Thus, the number of multisets in a complete multiset-trie is \( |N| = m^n \). Let \( M \) be a collection of multisets inserted into multiset-trie \( M \). Our model assumes that all the inserted multisets from \( M \) are chosen with the same probability \( p \) from \( N \), for some \( p \in (0, 1) \).
Let $\xi_1, \xi_2, \ldots, \xi_{i+1}$ be random variables such that $\xi_i$ represents the number of nodes in a multiset-trie on $i$-th level. For every node $j$ on $i$-th level we assign a random variable $\xi_{ij}$ to be the number of its children, such that $j \in [1, \xi_i]$. Then $\forall i \in [1, \sigma]$ the following holds:

$$\xi_{i+1} = \sum_{j=1}^{\xi_i} \xi_{ij}. \quad (1)$$

The random variable $\xi_{ij}$ is modeled by a zero-truncated binomial distribution on parameters $n$ and $p_{i+1}$, where $p_i = 1 - (1-p)^{\xi_{i+1} - 1}$. The distribution is zero-truncated, because any internal node has at least 1 descendant if $M$ is non-empty, and is binomial, since our model assumes a random input.

Applying the tools from Galton-Watson process [3] and using the fact that $\xi_1 = 1$, we calculate the expected number of nodes in a multiset-trie given parameters $n$, $\sigma$ and $p$

$$E(|M|) = \sum_{i=1}^{\sigma-1} n^{i-1} \frac{1 - (1-p)^{n^i - 1}}{1 - (1-p)^n}. \quad (2)$$

With the expected number of nodes in a multiset-trie $M$ obtained from (2), we can now generalize the result for a subtree in $M$ parameterized by an input multiset $m$. The subtrees that we are interested in are the ones that contain all the submultisets or all the supermultisets of $m$. In order to calculate the expected cardinality of such subtrees we need the following definition.

**Definition 1.** Let $m = \{ k_1, k_2, \ldots, k_m \}$, where $e^{k_i}$ is an element $e$ with multiplicity $k_i$. Let $M_1, M_2$ be the subsets of the set $M$, such that $M_1 = \{ x \in M : x \subseteq m \}$ and $M_2 = \{ x \in M : x \supseteq m \}$. Define $\alpha_i$ and $\beta_i$ as follows

$$\alpha_i = \begin{cases} 1, & i = 0 \\ \prod_{j=1}^{i} (k_j + 1), & 1 \leq i \leq \sigma \end{cases}$$

and

$$\beta_i = \begin{cases} 1, & i = 0 \\ \prod_{j=1}^{i} (n - k_j - 1), & 1 \leq i \leq \sigma \end{cases}$$

The expected cardinality of a multiset-trie subtree $M_{M_1}$ that contains all the multisets from the set $M_1$ is equal to

$$E(|M_{M_1}|) = \sum_{i=1}^{\sigma} \alpha_i - 1 \frac{1 - (1-p)^{n^i - 1}}{1 - (1-p)^n}. \quad (3)$$

The expected cardinality of a multiset-trie subtree $M_{M_2}$ that contains all the multisets from the set $M_2$ is equal to

$$E(|M_{M_2}|) = \sum_{i=1}^{\sigma} \beta_i - 1 \frac{1 - (1-p)^{n^i - 1}}{1 - (1-p)^n}. \quad (4)$$

The worst case time complexity of the multiset-trie operations is presented in the table below.

<table>
<thead>
<tr>
<th>Function</th>
<th>Worst case complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT, SEARCH, DELETE</td>
<td>$O(\sigma)$</td>
</tr>
<tr>
<td>GETALLSUBSETS</td>
<td>$O(</td>
</tr>
<tr>
<td>GETALLSUPERSETS</td>
<td>$O(</td>
</tr>
<tr>
<td>SUBMSETEXISTENCE</td>
<td>$O(</td>
</tr>
<tr>
<td>SUPERMSETEXISTENCE</td>
<td>$O(</td>
</tr>
</tbody>
</table>

2.4 Experiments

The implementation of multiset-trie is done in the C++ programming language and uses only the standard library of C++14 version of the standard.

In our experiments we will test the following functions: SUBMSETEXISTENCE, SUPERMSETEXISTENCE, GETALLSUBSETS and GETALLSUPERSETS. Performance of the functions will be measured by the number of visited nodes in multiset-trie by the particular function. In particular the performance is inversely proportional to the number of visited nodes. The selected parameters of the data structure that will be varied in experiments are as follows:

- $\sigma$ – the cardinality of the alphabet $\Sigma$
- $n$ – the maximal degree of a node
- $\phi$ – mapping $\phi : \Sigma \rightarrow \{1, 2, \ldots, \sigma\}$
- $d$ – density of a multiset-trie, where $d$ is the function $d(|M|) = |M| / |M|$

The artificially generated multisets are constructed with respect to parameters $\Sigma$, $\sigma$ and $n$. The collection $M$ is sampled from $N$ with uniform distribution which simulates a random user input.

Experiments 1 and 2. The Experiments 1 and 2 demonstrate the performance of multiset-trie depending on the density. In both experiments the data is artificially generated at random, meaning that $\phi$ has no influence on results, and therefore is set to an identity mapping.

In the first experiment the multiset-trie is used for storing and retrieving sets and the parameters $n$ and $\sigma$ are set to 2 and 25 respectively. The parameter $|M|$ varies from 10000 sets up to 320000 sets.

In the second experiment the multisets are used and the parameters $n$ and $\sigma$ are set to 6 and 25 respectively. The parameter $|M|$ varies from 40000 multisets up to 640000 sets.

The performance of the “existence” functions is different in Experiments 1 and 2. With increase of density it increases in the first experiment (see figure 2) and decreases in the second (see figure 3). It happens, because in the first case the multiset-trie is dense enough, so the increase of the density increases the probability of finding submultiset or supermultiset in multiset-trie, which lowers the number of visited nodes. However, in the second case the multiset-trie is of a lower density so the increase of the density does not affect the performance.
The behavior is as expected, because the number of multisets in a sparse multiset-trie increases with increase of density, which means that any multiset in the data structure will have more sub- and supermultisets.

Experiment 3. This experiment, we empirically find the extremum of density for functions submsetExistence and supermsetExistence. The parameters $\sigma$ and $n$ are set to 12 and 5 respectively. The density varies from $1.0 \times 10^{-4}\%$ to $1.0\%$. The number of visited nodes was chosen to be maximal for each value of particular density.

The dependence of the number of visited nodes on density appears to be a non linear function. Yet the function is continuous on the interval $[0, 1]$, and hence, there exists a global maximum. As we see on figure 4 both functions submsetExistence and supermsetExistence indeed have the maximum around $d \approx 7.0 \times 10^{-3}\%$.

Experiment 4. This experiment shows the influence of the mapping $\phi$ from alphabet $\Sigma$ to a set of consecutive integers.

The test data is taken from English dictionary which contains 235883 different words. Those words are mapped to multisets of integers according to the $\phi$. In particular, we are interested in cases when $\phi(\Sigma)$ enumerates letters by their relative frequency in English language. We say that $\phi(\Sigma)$ maps letters in ascending order if the most frequent letter is mapped to number $\sigma$. Conversely, in descending order this letter is mapped to number 1. The parameters $\sigma$ is set to 26 and $n$ is set to 10. The results on figure 5 are more balanced when letters are ordered by frequency in ascending order. Letters that have the least frequencies are now located at the top of multiset-trie according to ascending order of letters by frequency. This means that the search becomes narrower, because a lot of invalid paths will be discarded on top most levels. Thus, multiset-trie can be traversed faster.

3. CONCLUSIONS AND FUTURE WORK

Our studies show that multiset-trie is an input sensitive data structure. However, pre-processing of data, letter frequency analysis in particular, determines the best encoding of input data and ensures the best performance of the multiset-trie.

The performance of multiset-trie observed in our research opens even more interesting questions for the future investigation. Further steps in our research will be to extend the functionality of the multiset-trie. We are interested in more flexible multiset containment queries, where the types of sub and supermultisets can be specified.

4. REFERENCES