Д.А. Полушкин, К. В. Загретдинова, А.В. Бабич
Тюменский государственный университет, г.Тюмень
УДК 004.021

ИЗУЧЕНИЕ И СРАВНЕНИЕ ВОЗМОЖНЫХ СТРУКТУР ХРАНЕНИЯ СЛОВАРЯ ГОЛОСОВОГО ВВОДА И АЛГОРИТМЫ ПОИСКА ПО СЛОВАРЮ С УЧЁТОМ КОРРЕКЦИИ ОШИБОК

Аннотация. В статье описаны возможные структуры словарей голосового ввода и алгоритмы поиска по ним с возможностью коррекции ошибок.

Ключевые слова. Голосовой ввод, хранение словарей, поиск по словарю, ввод с коррекцией ошибок, поиск по словарю с коррекцией ошибок.

Введение. При реализации голосового ввода возникает необходимость распознания вводимых слов, что, в свою очередь, приводит к необходимости хранения словаря, а также поиска по нему, в том числе с использованием алгоритмов, обеспечивающих возможность нахождения слова даже в случае ошибки: как на стадии произнесения, так и на стадии распознания фонем.

Задача реализации голосового ввода разбивается на два основных этапа: выбор структуры для хранения словаря и алгоритм поиска по словарю с учётом коррекции ошибок. На первом этапе производится анализ способов построения словаря и их эффективности относительно занимаемой памяти, скорости поиска и добавления слов в словаре. На втором этапе рассматриваются алгоритмы поиска слова по вводимым поочерёдно фонемам (префиксу) с учётом возможных допущенных ошибок и выбирается наиболее оптимальный способ их исправления без привлечения пользователя.

Структура словаря. Исходя из поставленных задач, те или иные структуры хранения словарей могут быть более или менее эффективны [1]. На данном этапе будут рассмотрены основные подходы к организации словаря, исходя из занимаемой им памяти, скорости поиска и скорости добавления новых слов. При этом при расчёте временной сложности алгоритма общее количество слов в словаре будет обозначено за n, количество букв в алфавите за m, а количество букв в слове за k.

Вектор. Простейшим способом представления списка всех известных системе слов является несортированный одномерный массив слов - вектор.

... ...
выезд выход
выходка карета
каретка каток
поезд поездка
поток чай
... ...

Рис. 1. Визуальное представление словаря-вектора

Занимаемая память: являясь самым неэффективным способом хранения вектор хранит целиком все слова вне зависимости от их схожести.

Время на поиск слова: полный перебор всего массива со сравнением полного слова на каждом шаге, временная сложность O(n).

Время на добавление нового слова: так как словарь не отсортирован, слово просто добавляется в конец за O(1).

Двоичное дерево поиска. Дерево составляется, следуя следующим правилам:

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.
- У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равно, нежели значение ключа данных самого узла X.

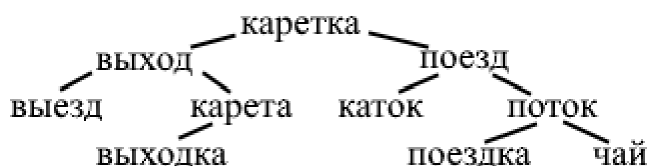

Рис. 2. Визуальное представление двоичного дерева поиска

Занимаемая память: так же, как и в случае с вектором хранятся абсолютно все существующие в словаре слова, и хранятся целиком.

Время на поиск слова: в лучшем случае – O(1), в худшем O(n), среднее время будет равно высоте дерева – O(log n).

Время на добавление нового слова: так как для добавления слова нужно фактически найти место, в котором это слово должно было стоять, то добавление нового слова занимает столько же, сколько и поиск – O(log n).

Префиксное дерево. Префиксное дерево представляет способ хранения информации, при котором в каждом узле хранится один символ, флаг завершения слова из цепочки символов и группа ссылок (рёбер), по количеству совпадающая с мощностью алфавита m. По своей сути является конечным автоматом.

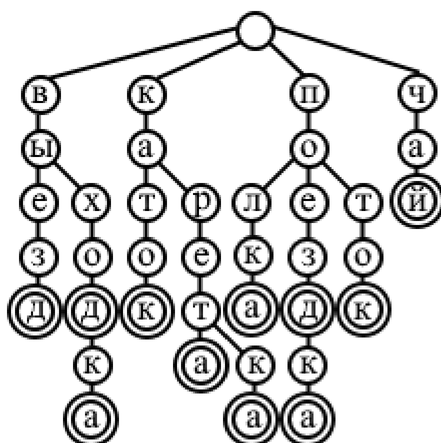

Рис. 3. Визуальное представление префиксного дерева

Занимаемая память: сокращает количество памяти, необходимое для хранения за счёт объединения общих префиксов, однако вынуждает хранить все пустые ссылки.

Время на поиск слова: для нахождения слова нужно затратить всего k итераций, так как при каждом переходе мы будем сразу переходить к следующей букве в слове. Итоговая временная сложность O(k) – самая быстрая среди всех рассмотренных вариантов.

Время на добавление нового слова: добавление сводится к нахождению позиции, в которой должен находится отсутствующий узел и его добавление, и

так, пока слово не будет целиком найдено в дереве, а значит, будет затрачено всего O(k) времени.

Префиксное альтернативное дерево. Префиксное альтернативное дерево отличается тем, что в каждом узле теперь хранится буква и из каждого узла ведёт ровно два ребра. Одно ведёт к следующей букве в слове, и переход по нему осуществляется, если текущая буква удовлетворяет поиску. Второе ведёт в альтернативное значение текущей буквы. Это позволяет не хранить множество пустых узлов дерева, что даёт выигрыш в памяти, но при этом сложность поиска слова возрастает.

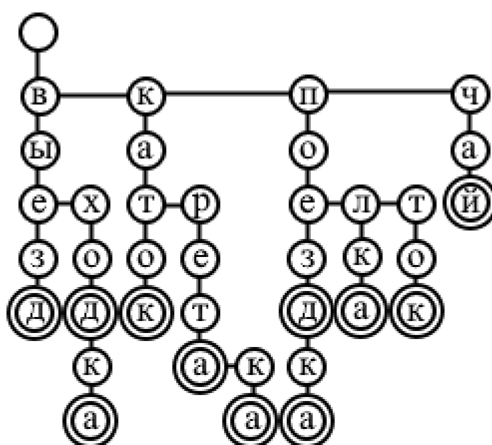

Рис. 4. Визуальное представление префиксного альтернативного дерева

Занимаемая память: сокращается количество хранимых пустых ссылок, что позволяет хранить ветвящиеся деревья в более компактном виде. Самое эффективно по памяти решение среди необработанных деревьев.

Время на поиск слова: так как теперь в худшем случае нужно будет перебрать для k букв в слове m-1 переходов по альтернативным ссылкам, итоговая временная сложность уже будет близка к O(m).

Время на добавление нового слова: аналогично поиску будет близка к O(m).

Тернарное дерево. Тернарное дерево является комбинацией свойств бинарного дерева и префиксного альтернативного дерева (также является конечным автоматом) [2]. У каждого узла ровно три потомка, так называемые меньший, больший и следующий. При продвижении по дереву, если буква в

данном узле нас удовлетворяет, мы переходим к следующему. Если же буква не подходит, то нужно выполнить сдвиг на альтернативную позицию, но, в отличие от префиксного альтернативного дерева, выбор перехода на букву, что больше текущей или на букву, что меньше текущей, позволяет производить поиск быстрее.

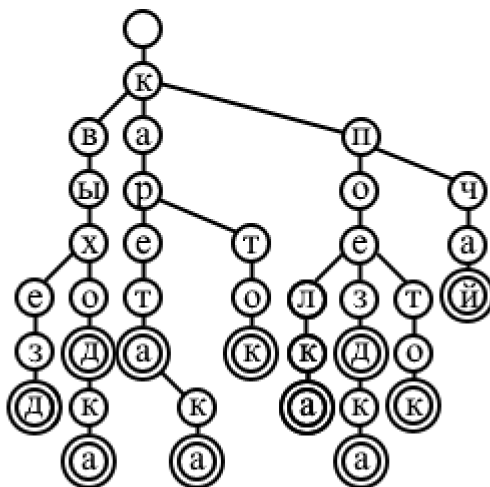

Рис. 5. Визуализация тернарного дерева

Занимаемая память: так как при хранении каждый узел обязательно имеет три ссылки, занимаемая память незначительно возрастает по сравнению с префиксным альтернативным деревом.

Время на поиск слова: при несбалансированном дереве может понадобиться до O(n) времени, однако при сбалансированном в среднем будет затрачиваться O(log n).

Время на добавление нового слова: новое слово добавляется по принципу поиска места, где должен стоять следующий символ, так что занимает, как и поиск, O(log n) времени. Исключением может послужить дополнительная затрата времени на балансировку дерева в случае неудачного добавления узлов.

Минимизация дерева. Так как тернарное и префиксное деревья являются конечными автоматами к ним можно применить операцию минимизации [3]. Она сводится к объединению общих суффиксов. При этом суффиксы объединяются с конца в начало.

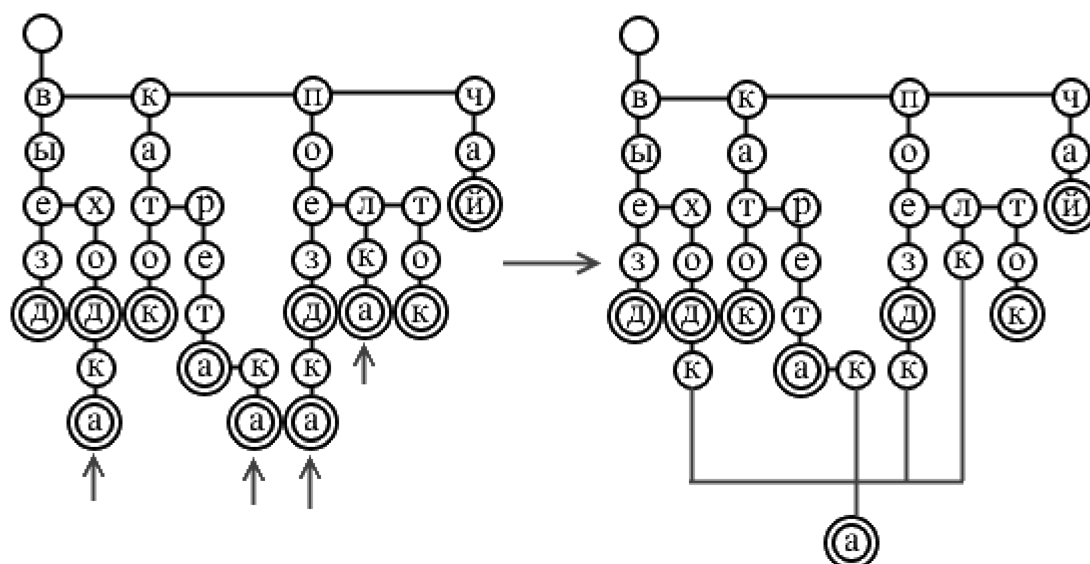

Рис. 6. Один шаг минимизации префиксного альтернативного дерева

Занимаемая память: уменьшение необходимости хранить одинаковые окончания отдельных слов словаря приводит к существенному снижению затрачиваемой памяти, необходимой для хранения.

Время на поиск слова: так как все ссылки остаются на своих местах и просто начинают указывать на другие ячейки памяти, скорость поиска никак не изменяется.

Время на добавление нового слова: из-за угрозы возникновения аномалий в минимальное дерево нельзя добавлять новые слова. В связи с этим для добавления слов необходимо построить изначальное дерево, а после добавления слова его нужно снова минимизировать, что существенно сказывается на затрачиваемом времени.

Сжатие дерева. Так как частой бывает ситуация, когда в дереве у узла есть только одна активная ссылка, можно применить операцию сжатия дерева и объединить узел с одной ссылкой с узлом, на который он ссылается. Может быть применено к минимальному дереву.

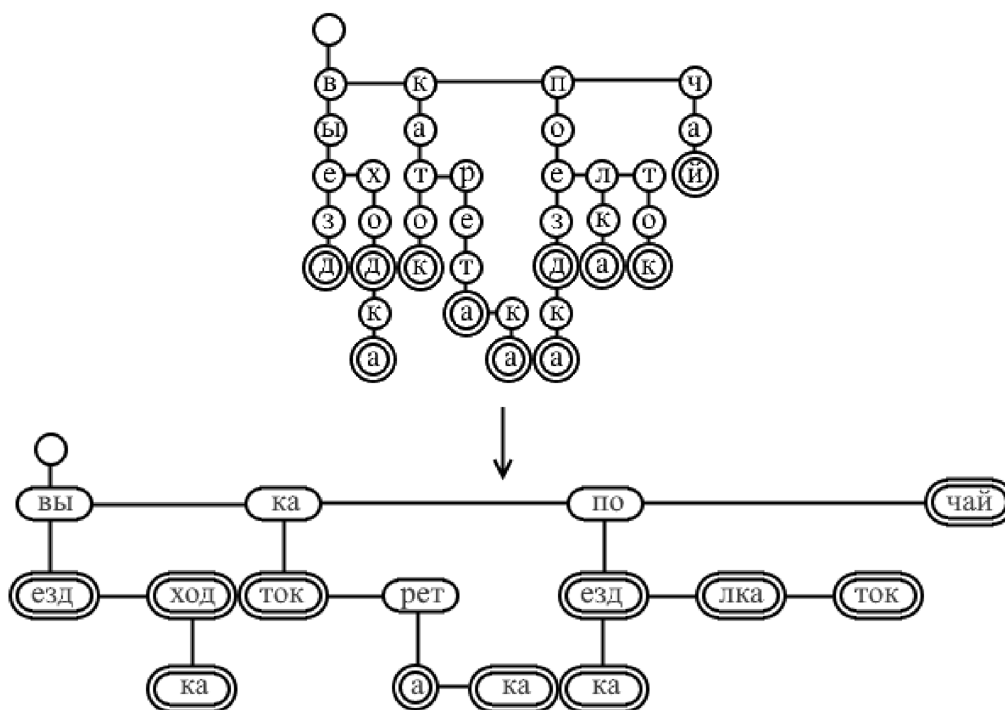

Рис. 7. Сжатие префиксного альтернативного дерева

Занимаемая память: занимаемая память при этом будет уменьшаться, так как кроме очевидного уменьшения узлов будут так же уменьшаться расходы памяти на хранение всех пустых ссылок этих узлов.

Время на поиск слова: поиск слова в сжатом дереве ничем не отличается от поиска в дереве, к которому не было применено сжатие.

Время на добавление нового слова: так как в некоторых ситуациях может потребоваться вставка слова в ссылки, которые были утрачены при объединении узлов может потребоваться разбитие этих узлов, что вызывает дополнительные затраты по времени.

Итоговая структура. При голосовом наборе необходимо обеспечивать высокую скорость работы. При этом также важно уменьшить объем хранимой информации для успешного хранения больших словарей. Время добавления новых слов не так значимо, поскольку большая часть словаря составляется один раз и в дальнейшем изменения минимальны.

Исходя из этого, можно сделать вывод, что оптимальным будет использование тернарного минимального сжатого дерева, так как оно

обеспечивает скорость поиска за O(log n) времени, имея при этом почти минимальный объём занимаемой памяти.

Поиск с учётом ошибок. Кроме очевидного поиска слова по словарю может возникнуть ситуация, при которой пользователь ошибся в произношении или библиотека вернула слово с ошибкой. При этом можно рассматривать четыре отдельных ситуации: лишний символ, недостающий символ, неправильный символ или неправильный порядок пары соседних символов [4].

Расстояние между словами. Под расстоянием между словами чаще всего подразумевается понятие расстояния Левенштейна (Levenshtein distance), также известное как редакционное расстояние или дистанция редактирования и выражается оно в количестве необходимых изменений (добавления или удаления букв, их перестановок или замен) для перехода от одного слова к другому.

Так как при голосовом вводе пользователь ожидает получить результат, программному комплексу необходимо самостоятельно выбирать алгоритм решения возникших ошибок. Множество алгоритмов, применимых в других областях, при этом не могут быть использованы тут, потому как требуют дополнительного участия пользователя. Исходить он при этом может из нескольких базовых подходов, рассмотрим каждый из них отдельно.

Наивный подход. Простейшим способом исправления ошибки будет поиск расстояния от текущего состояния, в котором у нас есть набор символов, не соответствующий ни одному слову в словаре, до всех слов, что есть в словаре. Тогда первое среди слов с минимальным расстоянием можно будет считать за искомое слово. Этот способ отнимает много времени, так как будет необходимо считать расстояние до каждого из слов в словаре.

Расширяющийся поиск. Для текущей последовательности символов постепенно увеличивая расстояние начинают создаваться все возможные комбинации слов, которые далее ищутся в словаре. При этом для расстояния Левенштейна, равного одному будет создано k слов удалением одного символа, k-1 перестановкой соседних символов, m*k заменой одного символа и m*(k+1)

слов добавлением одного символа. Такой подход, несмотря на огромное количество переборов является более оптимальным, чем наивный, как по скорости работы, так и по точности, так как позволяет быстро находить слова с минимальным расстоянием.

Дополнительно увеличить точность можно выбирая среди найденных слов минимального расстояния слова, у которых отличие подходят под одно из следующих требований:

- В замене участвовал звук, заменяемый на схожий с ним (б = п) [5];
- В удалении или добавлении участвовал звук, стоящий на позиции непроизносимого (требуется дополнительное хранение флага, указывающего на непроизносимость данного звука при хранении в словаре);
- В замене участвовали пары звуков, обозначающих составные гласные (йо = ё);
- В результате замены получилось слово, подходящее под контекст предложения (требуется более глубокий анализ контекста).

Заключение. В результате проведённой работы, были изучены и сравнены между собой возможные структуры хранения словаря голосового ввода, среди структур выбрана оптимальная – тернарное минимальное сжатое дерево. Изучены возможные алгоритмы коррекции ошибок поиска и дополнительные факторы, позволяющие без участия пользователя получить после корректировки наиболее подходящие результаты.

СПИСОК ЛИТЕРАТУРЫ

1. Никлаус Вирт. Алгоритмы и структуры данных. Новая версия для Оберона + CD / Пер. с англ. Ткачев Ф. В. – М.: ДМК Пресс, 2010. – 191с.
2. Bentley J., Sedgewick B. Ternary Search Trees: [Электронный ресурс] — Режим доступа. — URL: http://www.drdobbs.com/database/ternary-search-trees/184410528 (Дата обращения 27.02.2017)

3. Шишков Д. Б. Минимизация конечных автоматов. – М.: KYBERNETIKA, 1972. ТОМ 8. – 297с.
4. Вычисление редакционного расстояния: [Электронный ресурс] — Режим доступа. — URL: https://habrahabr.ru/post/117063/ (Дата обращения 15.03.2017)
5. Захаров Л.М. Транскрипция текстов при синтезе и анализе русской речи. АРСО-96.