Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра программного обеспечения
информационных технологий

Л.В. СЕРЕБРЯНАЯ

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ФИНАНСОВЫХ СТРУКТУР

Методическое пособие
к лабораторным работам
для студентов специальности
«Программное обеспечение информационных технологий»
всех форм обучения

Минск БГУИР 2011

УДК 004.02+004.04:330(076)
ББК 32.973.26-018.2+65я73
С32

Р е ц е н з е н т:
доцент кафедры «Интеллектуальные системы»
Белорусского национального технического университета,
кандидат технических наук, доцент Г. Э. Романюк

Серебряная, Л.В.

С 32 Информационное обеспечение финансовых структур : метод. пособие к лаб. работам для студ. спец. «Программное обеспечение информационных технологий» всех форм обуч. / Л. В. Серебряная. – Минск : БГУИР, 2011. – 43 с. : ил.

ISBN 978-985-488-636-7.

В пособии представлены семь лабораторных работ, в которых рассмотрены теоретические основы работы с текстовой информацией. Приведены статистические методы и алгоритмы ее обработки.

УДК 004.02+004.04:330(076)
ББК 32.973.26-018.2+65я73

ISBN 978-985-488-636-7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Предметом рассмотрения данного методического пособия является понятие *информация*, а также методы и алгоритмы ее обработки. Информацию можно определить как совокупность сведений, уменьшающих степень неопределенности знания о конкретных событиях, процессах, явлениях и т. п. В зависимости от сферы использования информация может быть самой разной: экономической, технической, генетической и т. д. Информация в системе управления рассматривается как *ресурс управления*, имеющий важное стратегическое значение. И хотя информационные ресурсы в значительной степени являются взаимозаменяемыми по отношению к материальным, финансовым или трудовым ресурсам, организационная форма инфоресурсов, объем и качество информации напрямую влияют на эффективность управления и качество решений, принимаемых в информационной системе.

Информационная система *(*ИС*)* представляет собой коммуникационную систему по сбору, передаче, обработке информации об объекте, снабжающую работников различных рангов информацией для реализации функции управления. Внедрение информационных систем выполняется с целью повышения эффективности производственно-хозяйственной деятельности объекта за счет обработки и хранения информации, автоматизации операций, а также за счет принципиально новых методов управления. Неотъемлемой частью ИС является *информационное обеспечение* (ИО).

ИО – это совокупность методов и средств по размещению и организации информации, включающих в себя системы классификации и кодирования, унифицированные системы документации, рационализации документооборота и форм документов, методов создания внутримашинной информационной базы ИС. Основное назначение ИО состоит в создании динамической информационной модели объекта, отражающей его состояние в текущий и предшествующий момент времени.

Хранение больших объемов информации практически оправдано только при условии, если ее поиск и обработка осуществляется быстро и выдается она в доступной для понимания форме. Предназначенные для этого программные средства должны учитывать структуру информационного фонда и физические свойства запоминающей среды.

Возможности электронно-вычислительной техники в последние десятилетия значительно расширяются: при ощутимом увеличении ее способности хранить большие объемы информации стоимость хранения постоянно сокращается. При этом развитие программных средств, связанных с организацией и поиском данных в «электронных хранилищах» происходит не такими быстрыми темпами. Поэтому специалисты, разрабатывающие программы для информаци- онных систем, вынуждены все время поспевать за новыми возможностями вычислительной техники.

Лабораторная работа №1

ПРИМЕНЕНИЕ ЗАКОНОВ ЗИПФА ДЛЯ ОБРАБОТКИ ТЕКСТОВОЙ ИНФОРМАЦИИ

Цель работы: ознакомиться с законами Зипфа и научиться их применять для определения заданных характеристик текстовой информации.

Порядок выполнения работы:
1 Изучить теоретическую часть работы.
2 Реализовать алгоритм проверки первого закона Зипфа.
3 Реализовать алгоритм проверки второго закона Зипфа.
4 Оформить отчет по лабораторной работе.

Исходные данные: тексты размером 1-2 страницы на русском, белорусском, английском языках.
Выходные данные: вычисленные характеристики Зипфа и наборы ключевых слов для исходных текстов.

Большинство существующих подходов к анализу текстов можно разбить на два класса. К первому классу относятся простые, быстрые, но не очень точные механизмы анализа. Чаще всего эти подходы используют формальные статистические методы, основанные на частоте появления в тексте слов различных тематик. Второй класс формируют достаточно изощренные, дающие хороший результат, но сравнительно медленные подходы, основанные на лингвистических методах. Эффективным же можно считать такой подход, который сочетал бы в себе «простоту» статистических алгоритмов с достаточно высоким качеством обработки лингвистических методов.

В то же время, как показала практика, для достижения приемлемого качества в решении практических задач компьютерного анализа текстовой информации (автоматическое аннотирование, тематическая классификация и т. д.) не требуется полный грамматический анализ фразы. Достаточно выделить наиболее информативные единицы текста: ключевые слова, словосочетания, предложения и фрагменты. При этом в качестве характеристики информативности удобно выбрать частоту повторения слов в тексте.

Для выделения понятий текста, представляющих слова и связные словосочетания, применяется статистический алгоритм, основанный на анализе частоты встречаемости цепочек слов различной длины и их вхождения друг в друга. Во всех созданных человеком текстах можно выделить статистические закономерности: независимо от текста и языка написания внутренняя структура текста остается неизменной. Она

описывается законами Дж. Зипфа, который предположил, что слова с большим количеством букв встречаются в тексте реже коротких слов. Основываясь на этом постулате, Зипф вывел два универсальных закона.

1.1 Первый закон Зипфа «Ранг - частота»

Если измерить количество вхождений каждого слова в текст и взять только одно значение из каждой группы, имеющей одинаковую частоту, расположить частоты по мере их убывания и пронумеровать (порядковый номер частоты называется рангом частоты – r), то наиболее часто встречающиеся слова будут иметь ранг 1, следующие за ними – 2 и т. д. Вероятность p_i встретить произвольно выбранное слово равна отношению количества вхождений этого слова n_i к общему числу слов n в тексте, т. е.

$$p_i = n_i / n. \qquad 1.1$$

Зипф обнаружил следующую закономерность: произведение вероятности обнаружения слова в тексте на ранг частоты r – константа (C):

$$C = p_i \cdot r \quad или \quad C = (n_i \cdot r)/n. \qquad 1.2$$

Это функция типа $y = k/x$, а её график – равносторонняя гипербола. Следовательно, по первому закону Зипфа, если самое распространенное слово встречается в тексте, например, 100 раз, то следующее по частоте слово с высокой долей вероятности окажется на уровне 50.

Значение константы C в разных языках различно, но внутри одной языковой группы остается неизменным, в не зависимости от текста. Так, например, для английских текстов константа Зипфа равна приблизительно 0,1.

1.2 Второй закон Зипф «Количество - частота»

В первом законе не учтён тот факт, что разные слова могут входить в текст с одинаковой частотой. Зипф установил, что частота и количество слов, входящих в текст с одинаковой частотой, тоже связаны между собой. Если построить график, отложив по оси X частоту вхождения слова, а по оси Y – количество слов с данной частотой, то получившаяся кривая будет сохранять свои параметры для всех без исключения созданных человеком текстов. Как и в предыдущем случае, это утверждение верно в пределах одного языка. Однако и межъязыковые различия невелики. На каком бы языке текст не был написан,

форма кривой Зипфа останется неизменной (рисунок 1.1). Могут немного отличаться лишь коэффициенты, отвечающие за наклон кривой.

Законы Зипфа универсальны. В принципе они применимы не только к текстам. Аналогично определяется, например, зависимость количества городов от числа проживающих в них жителей. Воспользуемся законами Зипфа для извлечения из текста слов, отражающих его смысл, т. е. ключевых слов. На рисунке 1.2 показан график зависимости ранга частоты слов от частоты их вхождения в текст.

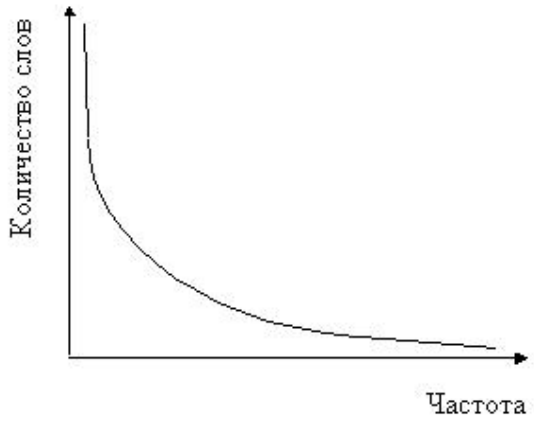


Рисунок 1.1 – График зависимости частоты вхождения слова от количества слов с данной частотой

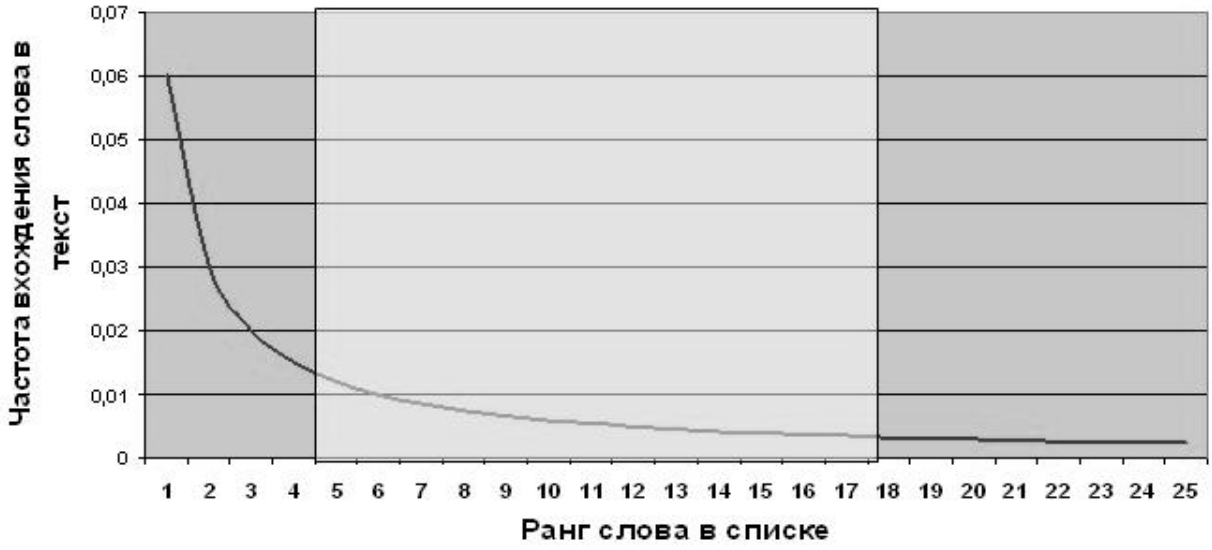


Рисунок 1.2 – График зависимости ранга частоты от частоты вхождения слова

Исследования показали, что наиболее значимые слова лежат в средней части диаграммы (на рисунке 1.2 это слова, имеющие ранг от 4 до 17). Это объясняется тем, что слова, которые встречаются слишком часто, в основном оказываются предлогами и местоимениями, в английском языке – артиклями и т. п. Редко встречающиеся слова, как правило, тоже не имеют решающего смыслового значения.

От того как будет определен диапазон значимых слов, зависит многое. Если диапазон широкий, то ключевыми словами будут вспомогательные слова; если установить узкий диапазон, то можно потерять смысловые термины.

Выделить ключевые слова помогает предварительное исключение из исследуемого текста некоторых слов, которые априори не являются значимыми, а, поэтому являются «шумом». Такие слова называются нейтральными или стоповыми (стоп-словами). Словарь стоп-слов называют стоп-листом. Например, для английского текста стоп-словами считаются термины: the, a, an, in, to, of, and, that… и т. д.

1.3 Весовые коэффициенты

Каждый из рассматриваемых отдельно взятых документов на практике чаще всего входит в базу данных (коллекцию) наряду с множеством других документов. Если представить всю базу данных как единый документ, к ней можно применить те же законы, что и к единичному документу.

В ходе назначения терминов (ключевых слов) коллекции стремятся избавиться от лишних слов (шума) и в то же время поднять рейтинг значимых слов, для чего вводят инверсную частоту термина. *Значение этого параметра тем меньше, чем чаще слово встречается в документах базы данных.* Вычисляют его по формуле

$$\lambda_i = \log(N / N_i), \tag{1.3}$$

где λ_i – инверсная частота термина i;

N – количество документов в базе данных;

N_i – количество документов с термином i.

Теперь каждому термину можно присвоить весовой коэффициент, отражающий его значимость

$$\kappa_{ij} = n_{ij} \cdot \lambda_i, \tag{1.4}$$

где κ_{ij} – вес термина i в документе j;

n_{ij} – частота термина i в документе j;

λ_i – инверсная частота термина i.

В качестве ключевых слов будут выбираться слова, имеющие высокий вес.

Современные способы индексирования не ограничиваются анализом перечисленных параметров текста. Поисковая машина может строить весовые коэффициенты с учетом местоположения термина внутри документа, взаимного расположения терминов, частей речи, морфологических особенностей и т. п. В качестве терминов могут выступать не только отдельные слова, но и словосочетания. Без этих законов Зипфа сегодня не обходится ни

одна система автоматического поиска информации. Это обусловлено тем, что математический анализ позволяет машине с хорошей точностью без участия человека распознать суть текста.

Лабораторная работа №2

ПОИСК ТЕКСТОВОЙ ИНФОРМАЦИИ ПО ЗАДАННОМУ НАБОРУ КЛЮЧЕВЫХ СЛОВ

Цель работы: реализовать поиск текстовой информации по документу-образцу.

Порядок выполнения работы:

1 Ознакомиться с теоретической частью работы.

2 Реализовать алгоритм поиска текстовой информации по документу-образцу.

3 Оформить отчет по лабораторной работе.

Исходные данные: коллекция документов из 10-12 текстов на русском и английском языках. Результат первой лабораторной работы: вектор ключевых слов, построенный по документу-образцу.

Выходные данные: документы, найденные в результате работы поисковой машины и соответствующие запросу поиска**.**

2.1 Особенности поиска информации

Накопленные к настоящему времени объемы информации в совокупности с непрерывно увеличивающимися темпами ее роста определяют актуальность и значимость исследований в области информационного поиска. Быстрое развитие сетевых технологий, в том числе и Интернета, способствует значительному увеличению доступных информационных ресурсов и объемов передаваемой информации. Зачастую это разнородная, слабо структурированная и избыточная информация, обладающая высокой динамикой обновления. При сегодняшних объемах доступной информации решение задач информационного поиска становится не только приоритетным, но и элементарно необходимым для обеспечения своевременного доступа к интересующей информации.

Информационный поиск – самостоятельное направление исследований, изучающее вопросы поиска документов, обработки результатов поиска, а также целый ряд смежных вопросов: моделирования, классификации, кластеризации и фильтрации документов, проектирования архитектур поисковых систем и пользовательских интерфейсов, языки запросов, и т. д.

Способы поиска можно разделить на две большие группы.

1. Библиографический поиск или поиск «по каталогу».

Такой вариант поиска обеспечивает нахождение документов по их выходным данным, например: по названию документа, его тематике, именам авторов, датам публикаций и т. д. Эти выходные данные составляют реквизиты документа. Основой каталога является предварительно заданная модель представления реквизитов, реализованная в виде базы данных, в соответствии с которой обеспечивается запись отдельных элементов реквизитов и последующий поиск по ним.

Основная проблема и недостаток такого варианта поиска – это необходимость выполнения значительного объема работ по предварительной организации, наполнению каталога. Как правило, это ручная классификация на основе привлечения экспертов. Подобный подход позволяет организовать лишь самую малую толику доступных информационных ресурсов.

2. Тематический поиск или поиск «по тексту».

Этот вариант поиска ориентирован на нахождение документов по их содержанию. Сюда же относится так называемый полнотекстовый поиск. Общая схема такого поиска заключается в формулировании некоторого запроса пользователем относительно содержания документа и отборе из множества доступных документов тех, которые удовлетворяют запросу. Такой вариант поиска удобен прежде всего, тем, что нет необходимости в предварительном разделении документов по различным категориям. Особенно это актуально при значительном объеме доступных документов, высокой динамике их обновления или отсутствии некоторых реквизитов. Такая ситуация характерна для Интернета. Основная проблема такого поиска – это сложность однозначной автоматической интерпретации содержания текстов документов и формулировок информационных потребностей пользователей. Сложность интерпретации затрудняет определение соответствия рассматриваемого документа информационным потребностям пользователя. Эти проблемы обусловлены отсутствием какой-либо регулярной структуры у текстовых документов на естественном языке. Такие информационные ресурсы принято называть неструктурированными или слабоструктурированными. Разработка методов анализа слабоструктурированных информационных ресурсов представляется весьма перспективным и многообещающим направлением исследований в области информационного поиска.

В соответствии с вышеприведенной классификацией способов поиска принято выделять два основных класса информационно-поисковых систем:

– поисковые каталоги;

– поисковые системы.

Поисковые каталоги в большей степени ориентированы на структурную организацию тематических коллекций с удобной системой ссылок и иерархией документов по тематическим коллекциям. Это

позволяет пользователю самостоятельно находить требуемый документ, просматривая структуру каталога, либо использовать механизмы поиска, ориентированные на данный каталог. Поисковые системы работают со слабоструктурированной информацией. Как правило, они используются для поиска документов в больших и динамичных информационных коллекциях, например в Интернете. Особенностью таких коллекций является отсутствие четко выраженной структурной организации, позволяющей упорядочить и однозначно классифицировать хранящиеся в них документы по тематической направленности.

В рамках данного методического пособия наибольший интерес представляют поисковые системы, а точнее используемые в них методы анализа документов. Процесс поиска текстовой информации, реализуемый типичной поисковой системой, включает в себя следующие этапы:

– формализация пользователем поискового запроса (представление пользователем в том или ином виде своих информационных потребностей);

– предварительный отбор документов по формальным признакам наличия интересующей информации (например, наличие в тексте документа одного из слов запроса, если запрос формулируется на естественном языке);

– анализ отобранных документов (лингвистический, статистический);

– оценка соответствия смыслового содержания найденной информации требованиям поискового запроса (ранжирование).

Одним из ключевых понятий, характеризующих выбор того или иного метода анализа текстовой информации, а также реализацию конкретного варианта поиска, является модель поиска. Модель поиска – это сочетание следующих составляющих:

– способа представления документов;

– способ представления поисковых запросов;

– вида критерия релевантности документов.

Всю совокупность представленных на сегодняшний день методов тематического анализа текста можно разделить на две группы:

– лингвистический анализ;

– статистический анализ.

Первый ориентирован на извлечение смысла текста по его семантической структуре, второй – по частотному распределению слов в тексте. Однако говорить о принадлежности какого-либо из подходов к конкретной группе можно лишь условно, как правило, в реальных задачах обработки текста приходится использовать сочетание методик из обеих групп.

Лингвистический анализ можно разделить на четыре взаимодополняющих анализа: лексический, морфологический, синтаксический, семантический.

Статистический анализ – это, как правило, частотный анализ в тех или иных его вариациях. Суть такого анализа заключается в подсчете

количества повторений слов в тексте и использовании результатов подсчета для конкретных целей, например вычисление весовых коэффициентов ключевых слов. Одним из эффективных статистических подходов является способ поиска по документу-образцу.

Документ-образец выступает в качестве одной из форм представления информационных потребностей пользователя. Целью поиска является обнаружение тематически близких документов. Самым простым подходом к решению задачи поиска документов по образцу является использование всех слов документа-образца в качестве запроса. Однако длина такого запроса может оказаться очень большой, что отрицательно скажется на качестве поиска, т. к. результатом поиска будут все документы, в которых присутствовали данные слова, и таких документов может быть очень много. Это отрицательно скажется как на самой поисковой системе – вычислительные ресурсы и трафик не безграничны и система может оказаться перегруженной, так и на человеке – просмотр и анализ найденных документов может занять значительное время, редкий пользователь готов к этому. Приемлемым вариантом в данном случае является выделение тематики документа. Под тематикой понимается множество ключевых слов, описывающих с некоторой степенью адекватности содержание документа. Тематика – это приближенное представление документа. Для повышения точности и адекватности описания содержания документа ключевые слова используются с некоторыми весовыми коэффициентами, которые соотносятся с частотой повторений этих слов в тексте. Вопросы выделения тематики и вычисления тематической близости документов по их тематическому представлению во многом и определяют возможность и эффективность поиска по документу-образцу.

2.2 Алгоритм поиска

Всю коллекцию документов необходимо организовать так, чтобы можно было легко отыскать в ней нужный материал. База данных должна взаимодействовать с пользовательским запросом. Запросы могут быть простыми, состоящими из одного слова, и сложными – из нескольких слов, связанных логическими операторами. Простой запрос оправдывает свое название. Пользователь вводит слово, машина ищет его в списке терминов и выдает все связанные с термином ссылки. Структура такой базы данных проста. Взаимодействие со сложными запросами требует более сложной организации.

Рассмотрим последовательность действий для организации поиска.

1 Подбирается текст-источник. Чем четче описаны проблемы в тексте-источнике, тем качественнее и точнее окажется результат.

2 Удаляются из текста стоп-слова.

3 Без учета морфологии слов вычисляется частота вхождения каждого термина.
4 Ранжируются термины в порядке убывания их частоты вхождения.
5. Выбирается диапазон частот из середины упорядоченного списка, построенного по определенному закону. Достаточно взять 10 – 20 терминов.
6 Составляется запрос из отобранных слов в порядке их следования в списке терминов. Запрос должен восприниматься машиной как слова, связанные логическим оператором ИЛИ.
7 Запрос отправляется поисковой системе.

Лабораторная работа №3

СПОСОБЫ ОРГАНИЗАЦИИ И ОБРАБОТКИ ИНФОРМАЦИИ ВО ВНЕШНЕЙ ПАМЯТИ

Цель работы: выполнить обработку информации во внешней памяти с помощью хеширования и B-деревьев.

Порядок выполнения работы:

1 Ознакомиться с теоретической частью работы.

2 Реализовать алгоритмы организации информации с помощью хеширования и *B*-деревьев. Сравнить эффективность двух способов.

3 Оформить отчет по лабораторной работе.

Исходные данные: файл с данными, в котором требуется выполнить поиск, вставку и удаление записей на базе хеширования и *B*-деревьев.

Выходные данные: результаты выполнения заданных операций, а также сравнение временных характеристик выполненных операций.

3.1 Хеширование данных

На сегодняшний день существует множество способов повышения эффективности работы с большими объемами информации. Например, для ускорения доступа к данным в таблицах можно использовать предварительное упорядочивание таблицы в соответствии со значениями ключей. При этом могут быть использованы методы поиска в упорядоченных структурах данных, что существенно сокращает время поиска данных по значению ключа. Однако при добавлении новой записи требуется переупорядочить таблицу. Потери времени на повторное упорядочивание справочника часто превышают выигрыш от сокращения времени поиска. Рассмотрим способы организации данных, лишенные указанного недостатка. Это различные формы хеширования данных.

На рисунке 3.1 показана базовая структура данных при открытом хешировании. Основная идея метода заключается в том, что множество данных разбивается на конечное число классов. Для B классов, пронумерованных от 0 до B-1, строится хеш-функция h, которая для любого элемента x исходного множества функция $h(x)$ принимает целочисленное значение из интервала 0, …, B-1, соответствующее классу, которому принадлежит элемент x. Элемент x называют ключом, $h(x)$ – хеш-значением x, а классы – сегментами. Массив (таблица сегментов), проиндексированный номерами сегментов 0, 1, ... B-1, содержит заголовки

для B списков. Элемент x i-го списка – это элемент исходного множества, для которого $h(x)=i$.

Если сегменты приблизительно равны по размеру, то в этом случае списки всех сегментов должны быть наиболее короткими при данном числе сегментов. Если исходное множество состоит из N элементов, тогда средняя длина списков будет N/B элементов. Если удается оценить величину N и выбрать B как можно ближе к этой величине, то в каждом списке будет один-два элемента. Тогда время выполнения операций с данными будет малой постоянной величиной, зависящей от N или от B. Однако не всегда ясно, как выбрать хеш-функцию h так, чтобы она примерно поровну распределяла элементы исходного множества по всем сегментам.

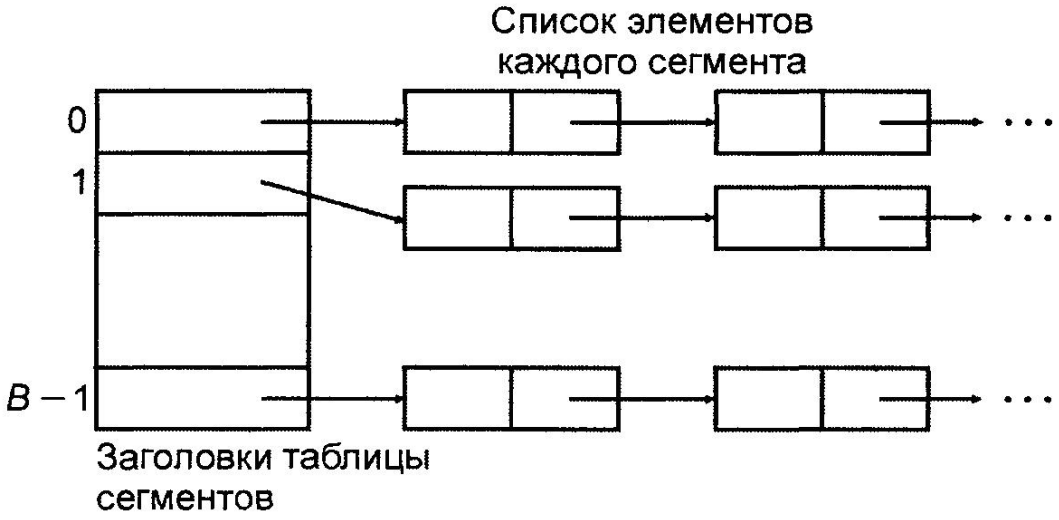


Рисунок 3.1 – Организация данных при открытом хешировании

Идеальной хеш-функцией является такая, которая для любых двух неодинаковых ключей выдает неодинаковые адреса, т. е.

$$k_1 \neq k_2 \Rightarrow h(k_1) \neq h(k_2). \qquad (3.1)$$

Однако подобрать такую функцию можно в случае, если все возможные значения ключей известны заранее. Такая организация данных носит название «совершенное хеширование». Если заранее не определено множество значений ключей и длина таблицы ограничена, подбор совершенной функции затруднителен. Поэтому часто используют хеш-функции, которые не гарантируют выполнение условия (3.1).

Хеширование – распространенный метод обеспечения быстрого доступа к информации, хранящейся во вторичной памяти. Записи файла распределяются между *сегментами*, каждый из которых состоит из связного списка одного или нескольких блоков внешней памяти.

Имеется таблица сегментов, содержащая B указателей, – по одному на каждый сегмент. Каждый указатель в таблице сегментов представляет собой физический адрес первого блока связного списка блоков для соответствующего сегмента. Сегменты пронумерованы от 1 до B. Хеш-функция h отображает каждое значение ключа в одно из целых чисел от 1 до B. Если x – ключ, то $h(x)$ является номером сегмента, который содержит запись с ключом x, если такая запись вообще существует. Блоки, составляющие каждый сегмент, образуют связный список. Таким образом, заголовок i-го блока содержит указатель на физический адрес (i+1)-го блока. Последний блок сегмента содержит в своем заголовке *nil*-указатель. Такой способ организации показан на рисунке 3.2. При этом в данном случае элементы, хранящиеся в одном блоке сегмента, не требуется связывать друг с другом с помощью указателей. Связывать между собой нужно только блоки.

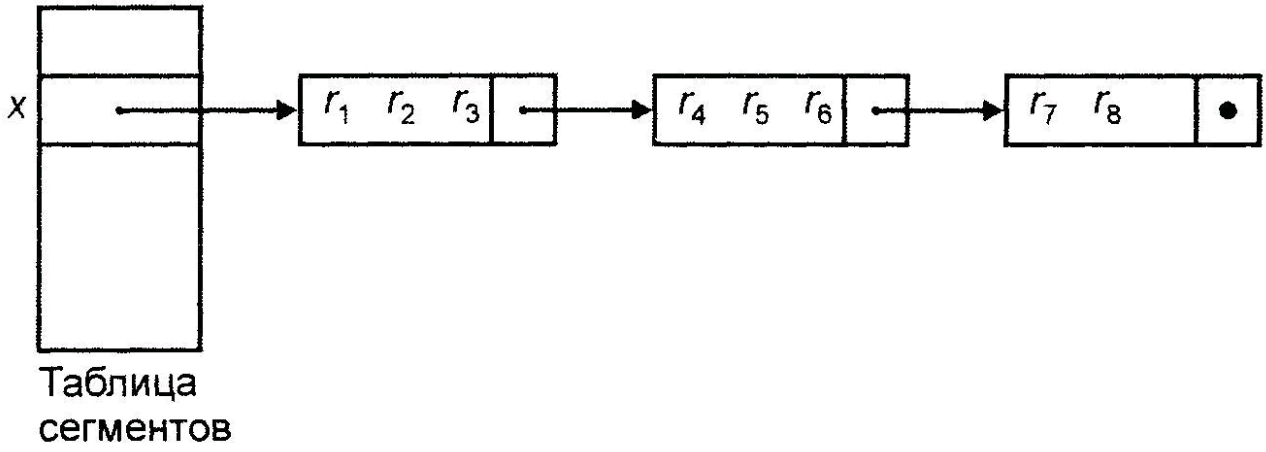


Рисунок 3.2 – Сегменты, состоящие из связанных блоков

Если размер таблицы сегментов невелик, ее можно хранить в основной памяти, иначе ее можно хранить последовательным способом в отдельных блоках внешней памяти. Если требуется найти запись с ключом x, вычисляется $h(x)$ и находится блок таблицы сегментов, содержащий указатель на первый блок сегмента $h(x)$. Затем последовательно считываются блоки сегмента $h(x)$, пока не обнаружится блок, содержащий запись с ключом x. Если исчерпаны все блоки в связном списке для сегмента $h(x)$, делается вывод, что x не является ключом ни одной из записей.

Такая структура оказывается эффективной, если в выполняемом операторе указываются значения ключевых полей. Среднее количество обращений к блокам, требующееся для выполнения оператора, в котором указан ключ записи, приблизительно равняется среднему количеству блоков в сегменте, которое равно n/bk, если n – количество записей, блок содержит b записей, а k соответствует количеству сегментов. В результате при такой организации данных операторы, использующие значения

ключей, выполняются в среднем в $к$ раз быстрее, чем в случае неорганизованного файла.

Чтобы вставить запись с ключом, значение которого равняется x, нужно сначала проверить, нет ли в файле записи с таким значением ключа. Если такая запись есть, то выдается сообщение об ошибке, поскольку предполагается, что ключ уникальным образом идентифицирует каждую запись. Если записи с ключом x нет, новая запись вставляется в первый блок цепочки для сегмента $h(x)$, в который ее удается вставить. Если запись не удается вставить ни в один из существующих блоков сегмента $h(x)$, файловой системе выдается команда – найти новый блок, в который будет помещена эта запись. Затем новый блок добавляется в конец цепочки блоков сегмента $h(x)$.

Для удаления записи с ключом x требуется сначала найти эту запись, а затем установить ее бит удаления.

Удачная организация файлов с хешированным доступом требует лишь незначительного числа обращений к блокам при выполнении каждой операции с файлами. Если выбрана удачная функция хеширования, а количество сегментов приблизительно равно количеству записей в файле, деленному на количество записей, которые могут поместиться в одном блоке, тогда средний сегмент состоит из одного блока. Если не учитывать обращения к блокам, которые требуются для просмотра таблицы сегментов, типичная операция поиска данных, основанного на ключах, потребует одного обращения к блоку, а операция вставки, удаления или изменения потребуют двух обращений к блокам. Если среднее количество записей в сегменте намного превосходит количество записей, которые могут поместиться в одном блоке, можно периодически реорганизовывать таблицу сегментов, удваивая количество сегментов и деля каждый сегмент на две части.

3.2 Внешние деревья поиска

Для представления внешних файлов удобно использовать древовидные структуры данных. B-дерево, являющееся обобщением бинарных деревьев, удачно подходят для представления внешней памяти. Поэтому оно стало стандартным способом организации индексов в системах баз данных.

Обобщением дерева двоичного поиска является m-арное дерево, в котором каждый узел имеет не более m сыновей. Так же, как и для деревьев бинарного поиска, для m-арного дерева считается, что выполняется следующее условие: если n_1 и n_2 являются двумя сыновьями одного узла и n_1 находится слева от n_2, тогда все элементы, исходящие вниз от n_1, оказываются меньше элементов, исходящих вниз от n_2.

Операции поиска, вставки и удаления элементов для *m*-арного дерева поиска реализуются путем обобщения аналогичных операций для деревьев бинарного поиска.

Однако для внешних деревьев поиска важна проблема хранения записей в файлах, когда файлы хранятся в виде блоков внешней памяти. Правильным применением идеи разветвленного дерева является представление об узлах как о физических блоках. Внутренний узел содержит указатели на своих *m* сыновей, а также *m*-1 ключевых значений, которые разделяют потомков этих сыновей. Листья также являются блоками, содержащими записи основного файла.

Если использовать дерево двоичного поиска из *n* узлов для представления файла, хранящегося во внешней памяти, то для поиска записи в таком файле потребуется в среднем $\log_2 n$ обращений к блокам. Если вместо бинарного дерева поиска использовать для представления файла *m*-арное дерево поиска, то для поиска записи в таком файле потребуется $\log_m n$ обращений к блокам. В случае n = 1 000 000 дерево бинарного поиска потребовало бы примерно 20 обращений к блокам, тогда как 128-арное дерево поиска потребовало бы лишь трех обращений к блокам. Однако нельзя сделать *m* очень большим, поскольку чем больше *m*, тем больше должен быть размер блока. Кроме того, считывание и обработка более крупного блока занимают больше времени, поэтому существует оптимальное значение *m*, которое позволяет минимизировать время, требующееся для просмотра внешнего *m*-арного дерева поиска.

B-дерево – это особый вид сбалансированного *m*-арного дерева, который позволяет выполнять операции поиска, вставки и удаления записей из внешнего файла с гарантированной производительностью для самой неблагоприятной ситуации.

B-дерево порядка m представляет собой *m*-арное дерево поиска, характеризующееся следующими свойствами.

1 Корень либо является листом, либо имеет хотя бы двух сыновей.

2 Каждый узел, кроме корня и листьев, имеет от *m*/2 до *m* сыновей.

3 Все пути от корня до любого листа имеют одинаковую длину.

На рисунке 3.3 показано *B*-дерево порядка 5; предполагается, что в блоке листа помещается не более трех записей.

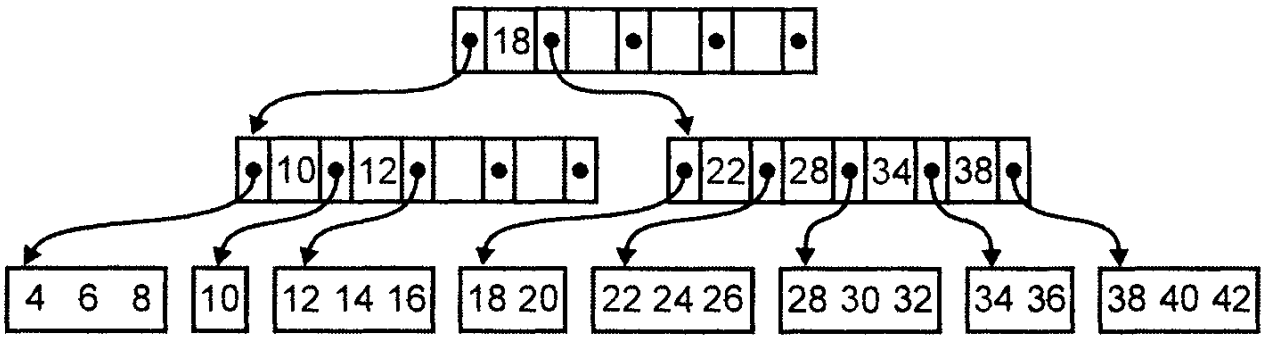


Рисунок 3.3 – *B*-дерево порядка 5

B-дерево можно рассматривать как иерархический индекс, каждый узел в котором занимает блок во внешней памяти. Корень *B*-дерева является индексом первого уровня. Каждый нелистовой узел на *B*-дереве имеет форму $(p_0, k_1, p_1, k_2, p_2, ...k_n, p_n)$, где p_i является указателем на *i*-ого сына, $0 \le i \le n$, а k_i – ключ, $1 \le i \le n$. Ключи в узле упорядочены, поэтому $k_1 < k_2 < ... < k_n$. Все ключи в поддереве, на которое указывает p_0, меньше, чем k_1. В случае $1 \le i < n$ все ключи в поддереве, на которое указывает p_i, имеют значения, не меньшие, чем k_i, и меньшие, чем k_{i+1}. Все ключи в поддереве, на которое указывает p_n, имеют значения, не меньшие, чем k_n.

Существует несколько способов организации листьев. В данном случае предполагается, что записи основного файла хранятся только в листьях и каждый лист занимает один блок.

3.3 Операторы на *B*-дереве

Поиск записей. Если требуется найти запись *r* со значением ключа *x*, нужно проследить путь от корня до листа, который содержит *r*, если эта запись вообще существует в файле. При прохождении указанного пути последовательно считываются из внешней памяти в основную внутренние узлы $(p_0, k_1, p_1, k_2, p_2, ...k_n, p_n)$ и вычисляется положение *x* относительно ключей $k_1, k_2, ...k_n$. Если $k_i \le x < k_{i+1}$, тогда в основную память считывается узел, на который указывает p_i, и повторяется описанный процесс. Если $x < k_1$, для считывания в основную память используется указатель p_0. Если $x \ge k_n$, тогда используется p_n.

Когда в результате изложенного процесса попадают на какой-либо лист, то пытаются найти запись со значением ключа *x*. Если количество входов в узле невелико, то в этом узле можно использовать линейный поиск, иначе лучше воспользоваться двоичным поиском.

Вставка записей. Если требуется вставить в *B*-дерево запись *r* со значением ключа *x*, нужно сначала воспользоваться процедурой поиска, чтобы найти лист *L*, которому должна принадлежать запись *r*. Если в *L* есть место для новой записи, то она вставляется в требуемом порядке в *L*. В этом случае не требуется внесений каких-либо изменений в предков листа *L*.

Если в блоке листа *L* нет места для записи *r*, у файловой системы запрашивается новый блок *L`* и из *L* в *L`* перемещается последняя половина записей. При этом *r* вставляется в требуемом порядке в *L* или *L`*. Допустим, узел *P* является родителем узла *L*. *P* известен, поскольку процедура поиска отследила путь от корня к листу *L* через узел *P*. Теперь можно рекурсивно применить процедуру вставки, чтобы разместить в *P* ключ *k`* и указатель *l`* на *L`*. *k`* и *l`* вставляются сразу же после ключа и указателя для листа *L*. Значение *k`* является наименьшим значением ключа в *L`*.

Если *P* уже имеет *m* указателей, вставка *k`* и *l`* в *P* приведет к расщеплению P и потребует вставки ключа и указателя в узел родителя *P*. Эта вставка может произвести эффект домино, распространяясь на предков узла *L* в направлении корня, вдоль пути, который уже был пройден процедурой поиска. Это может привести даже к тому, что понадобится расщепить корень, тогда создается новый корень, причем две половины старого корня выступают в роли двух его сыновей. Это единственная ситуация, когда узел может иметь менее *m*/2 потомков.

Удаление записей. Если требуется удалить запись *r* со значением ключа *x*, нужно сначала найти лист *L*, содержащий запись *r*. Затем, если такая запись существует, она удаляется из *L*. Если *r* является первой записью в *L*, после этого выполняется переход в узел *P* – родителя листа *L*, чтобы установить новое значение первого ключа для *L*. Если *L* является первым сыном узла *P*, то первый ключ *L* не зафиксирован в *P*, а появляется в одном из предков *P*. Таким образом, надо распространить изменение в наименьшем значении ключа *L* в обратном направлении вдоль пути от *L* к корню.

Если блок листа *L* после удаления записи оказывается пустым, он отдается файловой системе. После этого корректируются ключи и указатели в *P*, чтобы отразить факт удаления листа *L*. Если количество сыновей узла *P* оказывается теперь меньшим, чем *m*/2, проверяется узел *P`*, расположенный в дереве на том же уровне непосредственно слева или справа от *P*. Если узел *P`* имеет хотя бы *m*/2+2 сыновей, ключи и указатели распределяются поровну между *P* и *P`* так, чтобы оба эти узла имели хотя бы по *m*/2 потомков, сохраняя упорядоченность записей. Затем необходимо изменить значения ключей для *P* и *P`* в родителе *P* и, если необходимо, рекурсивно распространить воздействие внесенного изменения на всех предков узла *P*, на которых это изменение отразилось.

Если у $P`$ имеется ровно $m/2$ сыновей, $P$ и $P`$ объединяют в один узел с $2(m/2)–1$ сыновьями. Затем необходимо удалить ключ и указатель на $P`$ из родителя для $P`$. Это удаление можно выполнить с помощью рекурсивного применения процедуры удаления.

Если обратная волна воздействия удаления докатывается до самого корня, возможно, придется объединить только двух сыновей корня. В этом случае новым корнем становится результирующий объединенный узел, а старый корень можно вернуть файловой системе. Высота B-дерева уменьшается при этом на единицу.

Рассмотрим выполнение описанных операторов на примере B-дерева, изображенного на рисунке 3.4. Вставка записи со значением ключа 23 порождает B-дерево, показанное на рисунке. 3.4. Чтобы вставить эту запись, надо расщепить блок, содержащий записи с ключами 22, 23, 24 и 26, поскольку предполагается, что в один блок помещается не более трех записей. Два меньших остаются в этом блоке, а два больших помещаются в новый блок. Пару указатель-ключ для нового узла нужно вставить в родителя, который в таком случае расщепляется, поскольку не может содержать шесть указателей. Корень принимает пару указатель-ключ для нового узла, однако корень не расщепляется, т. к. он располагает достаточной емкостью.

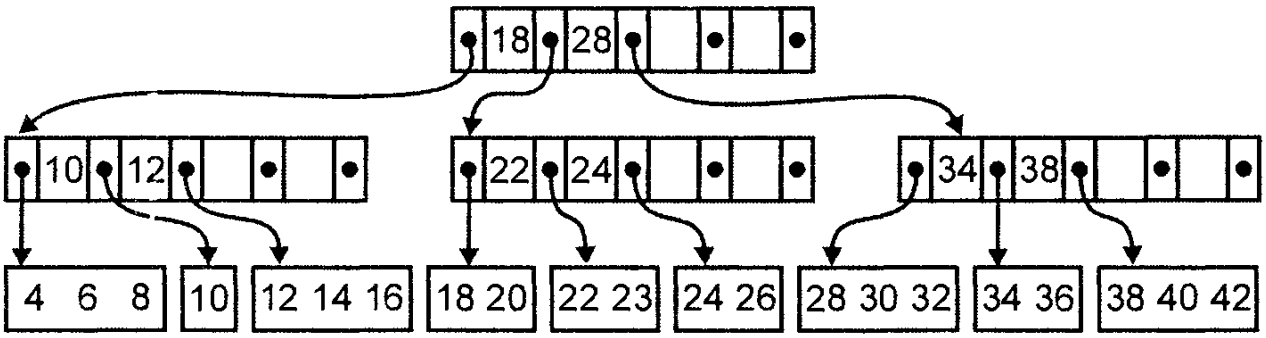


Рисунок 3.4 – B-дерево после вставки в него записи

Удаление записи с ключом 10 из B-дерева, показанного на рисунке 3.4, приводит к B-дереву, изображенному на рисунке 3.5. В этом случае блок, содержащий запись с ключом 10, отбрасывается. У его родителя теперь оказывается только два сына, а у правого брата этого родителя имеется минимальное количество сыновей – три. Таким образом, в результате объединения родителя и его брата получается один узел с пятью сыновьями.

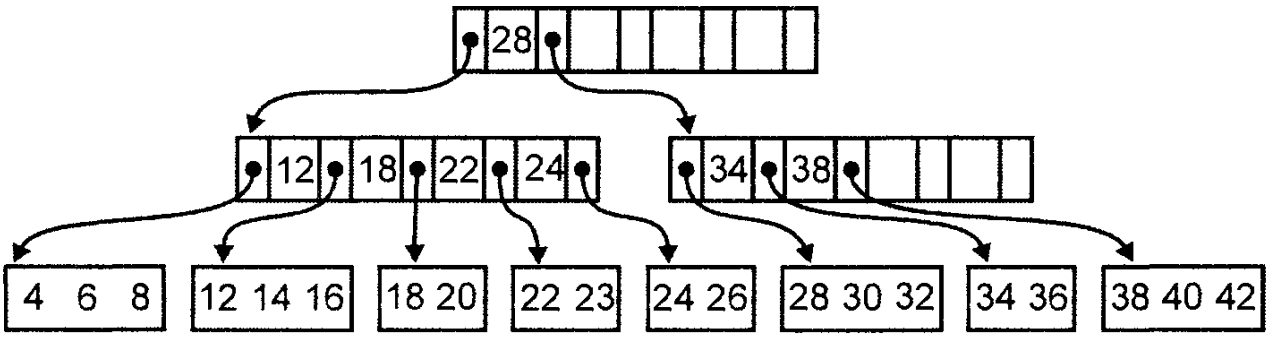


Рисунок 3.5 – *B*-дерево послед удаления записи

3.4 Сравнение методов

На практике сравнивают количество обращений к блокам для разных методов, связанное с выполнением той или иной операции с файлами.

Хеширование зачастую является самым быстрым из методов. Оно требует в среднем двух обращений к блокам по каждой операции, не считая обращений к блокам, которые требуются для просмотра самой таблицы сегментов, если количество сегментов достаточно велико для того, чтобы типичный сегмент использовал только один блок. Однако в случае хеширования сложно обращаться к записям в отсортированной последовательности.

B-деревья получили большую популярность как средство доступа к файлам в системах баз данных. Причина этой популярности частично заключается в их способности обрабатывать запросы, запрашивая записи с ключами, относящимися к определенному диапазону. Кроме того, *B*-деревья удачно использовать в качестве вторичных указателей, когда ключи в действительности не определяют ту или иную уникальную запись. Все перечисленные методы намного эффективнее обычного последовательного просмотра файла. Временные различия между ними невелики и не поддаются точной аналитической оценки, особенно с учетом того, что соответствующие параметры такие, как ожидаемая длина файла и коэффициенты заполненности блоков, трудно прогнозировать заранее.

Лабораторная работа №4

ПРЕДСТАВЛЕНИЕ СООБЩЕНИЙ КОДАМИ ХАФФМАНА

Цель работы: выполнить кодирование и декодирование информации с помощью кодов Хаффмана.

Порядок выполнения работы:

1 Ознакомиться с теоретической частью работы.

2 Реализовать алгоритм конструирования кодов Хаффмана и выполнить кодирование/декодирование текстовой информации.

3 Оформить отчет по лабораторной работе.

Исходные данные: символьные сообщения с заданными вероятностями появления каждого символа.

Выходные данные: кодированные и декодированные сообщения.

4.1 Метод Хаффмана

Рассмотрим задачу конструирования *кодов Хаффмана*, используемых для сжатия информации. Предположим, есть сообщения, состоящие из последовательности символов. В каждом сообщении символы независимы и появляются с известной вероятностью, не зависящей от позиции в сообщении. Например, имеются сообщения, состоящие из пяти символов *a*, *b*, *c*, *d*, *e,* которые появляются в сообщениях с вероятностями 0,12; 0,4; 0,15; 0,08 и 0,25 соответственно.

Необходимо закодировать каждый символ последовательностью из нулей и единиц так, чтобы код любого символа являлся префиксом кода сообщения, состоящего из последующих символов. Это *префиксное свойство* позволяет декодировать строку из нулей и единиц последовательным удалением префиксов (т. е. кодов символов) из этой строки.

В таблице 4.1показаны две возможные кодировки для указанных пяти символов.

Таблица 4.1 Два двоичных кода

Символ	Вероятность	Код 1	Код 2
a	0,12	000	000
b	0,40	001	11
c	0,15	010	01
d	0,08	011	001
e	0,25	100	10

Ясно, что *код* 1 обладает префиксным свойством, поскольку любая последовательность из трех битов будет префиксом для другой последовательности из трех битов, т. е. любая префиксная последовательность однозначно идентифицируется символом. Алгоритм декодирования для этого кода следующий: надо поочередно брать по три бита и преобразовывать каждую группу битов в соответствующие символы. Например, последовательность 001010011 соответствует исходному сообщению *bcd.*

Несложно проверить, что *код* 2 также обладает префиксным свойством. Процесс декодирования здесь не отличается от аналогичного процесса для первого кода. Единственная сложность для второго кода заключается в том, что нельзя сразу всю последовательность битов разбить на отдельные сегменты, соответствующие символам, т. к. символы могут кодироваться и двумя, и тремя битами. Для примера рассмотрим двоичную последовательность 1101001, которая опять представляет символы *bcd.* Первые два бита 11 однозначно соответствуют символу *b*, поэтому их можно удалить, тогда получится 01001. Здесь 01 также однозначно определяет символ *c* и т. д.

Задача конструирования кодов Хаффмана заключается в следующем: имея множество символов и значения вероятностей их появления в сообщениях, построить такой код с префиксным свойством, чтобы средняя длина кода (в вероятностном смысле) последовательности символов была минимальной. Минимизировать среднюю длину кода необходимо для того, чтобы уменьшить длину вероятного сообщения, т. е. чтобы сжать сообщение. Чем короче среднее значение длины кода символов, тем короче закодированное сообщение. В частности, первый код из нашего примера имеет среднюю длину кода 3. Это число получается в результате умножения длины кода каждого символа на вероятность появления этого символа. Второй код имеет среднюю длину 2,2, поскольку символы *a* и *d* имеют суммарную вероятность появления 0,20 и длина их кода составляет три бита, тогда как другие символы имеют код длиной 2.

Однако существует код с префиксным свойством, средняя длина которого равна 2,15. Это наилучший возможный код с теми же вероятностями появления символов. Способ нахождения оптимального префиксного кода называется *алгоритмом Хаффмана.* В этом алгоритме находятся два символа *a* и *b* с наименьшими вероятностями появления, заменяемыми одним фиктивным символом, например *x*, который имеет вероятность появления, равную сумме вероятностей появления символов *a* и *b.* Затем, используя эту процедуру рекурсивно, находим оптимальный префиксный код для меньшего множества символов (где символы *a* и *b* заменены одним символом *x*). Код для исходного множества символов получается из кодов замещающих символов путем добавления 0 и 1 перед кодом замещающего символа, и эти два новых кода принимаются как коды

заменяемых символов. Например, код символа *a* будет соответствовать коду символа *x* с добавленным нулем перед этим кодом, а для кода символа *b* перед кодом символа *x* будет добавлена единица.

Можно рассматривать префиксные коды как пути на двоичном дереве: прохождение от узла к его левому сыну соответствует 0 в коде, а к правому сыну – 1. Если пометить листья дерева кодируемыми символами, то получим представление префиксного кода в виде двоичного дерева. Префиксное свойство гарантирует, что нет символов, которые были бы метками внутренних узлов дерева (не листьев), и наоборот, помечая кодируемыми символами только листья дерева, мы обеспечиваем префиксное свойство кода этих символов.

Двоичные деревья для кодов 1 и 2 из таблицы 4.1 показаны на рисунке 4.1 (дерево слева соответствует коду 1, а дерево справа – коду 2).

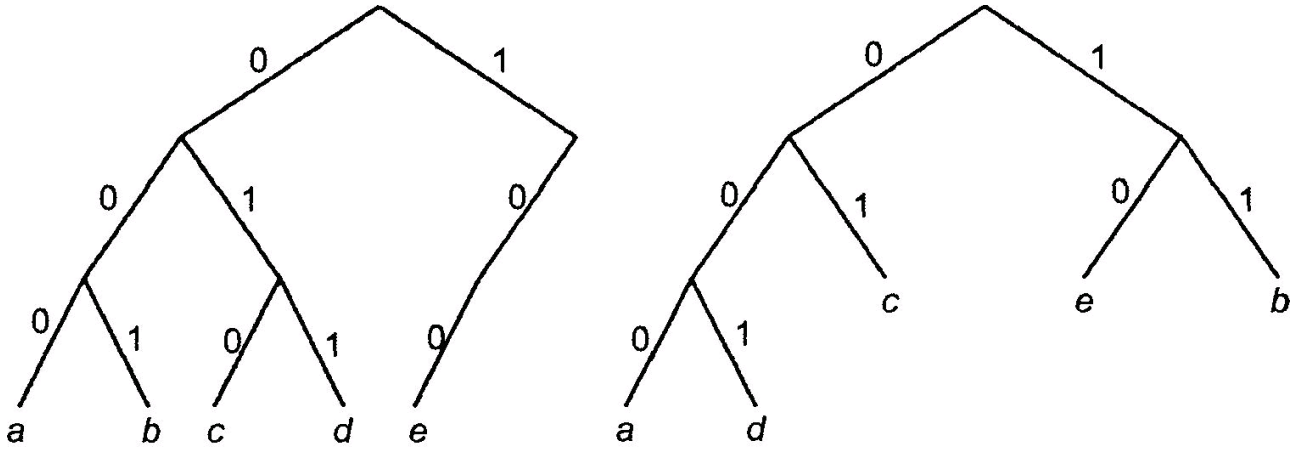


Рисунок 4.1 – Коды двоичных деревьев с префиксным свойством

Для реализации алгоритма Хаффмана используется *лес,* т. е. совокупность деревьев, чьи листья помечаются символами, для которых разрабатывается кодировка, а корни помечаются суммой вероятностей всех символов, соответствующих листьям дерева. Эти суммарные вероятности называются *весом* дерева. Вначале каждому символу соответствует дерево, состоящее из одного узла, в конце работы алгоритма получается одно дерево, все листья которого будут помечены кодируемыми символами. В результирующем дереве путь от корня к любому листу представляет код для символа-метки этого листа, составленный по схеме, согласно которой левый сын узла соответствует 0, а правый – 1.

Важным этапом в работе алгоритма является выбор из леса двух деревьев с наименьшими весами. Эти два дерева комбинируются в одно с весом, равным сумме весов составляющих деревьев. При слиянии деревьев создается новый узел, который становится корнем объединенного дерева и имеет в качестве левого и правого сыновей корни старых деревьев. Этот процесс продолжается до тех пор, пока не получится только одно дерево.

Это дерево соответствует коду, который при заданных вероятностях имеет минимальную среднюю длину.

Рассмотрим на примере последовательные шаги выполнения алгоритма Хаффмана. Кодируемые символы и их вероятности заданы в таблице 4.1. Этапы построения дерева Хаффмана показаны на рисунке 4.2

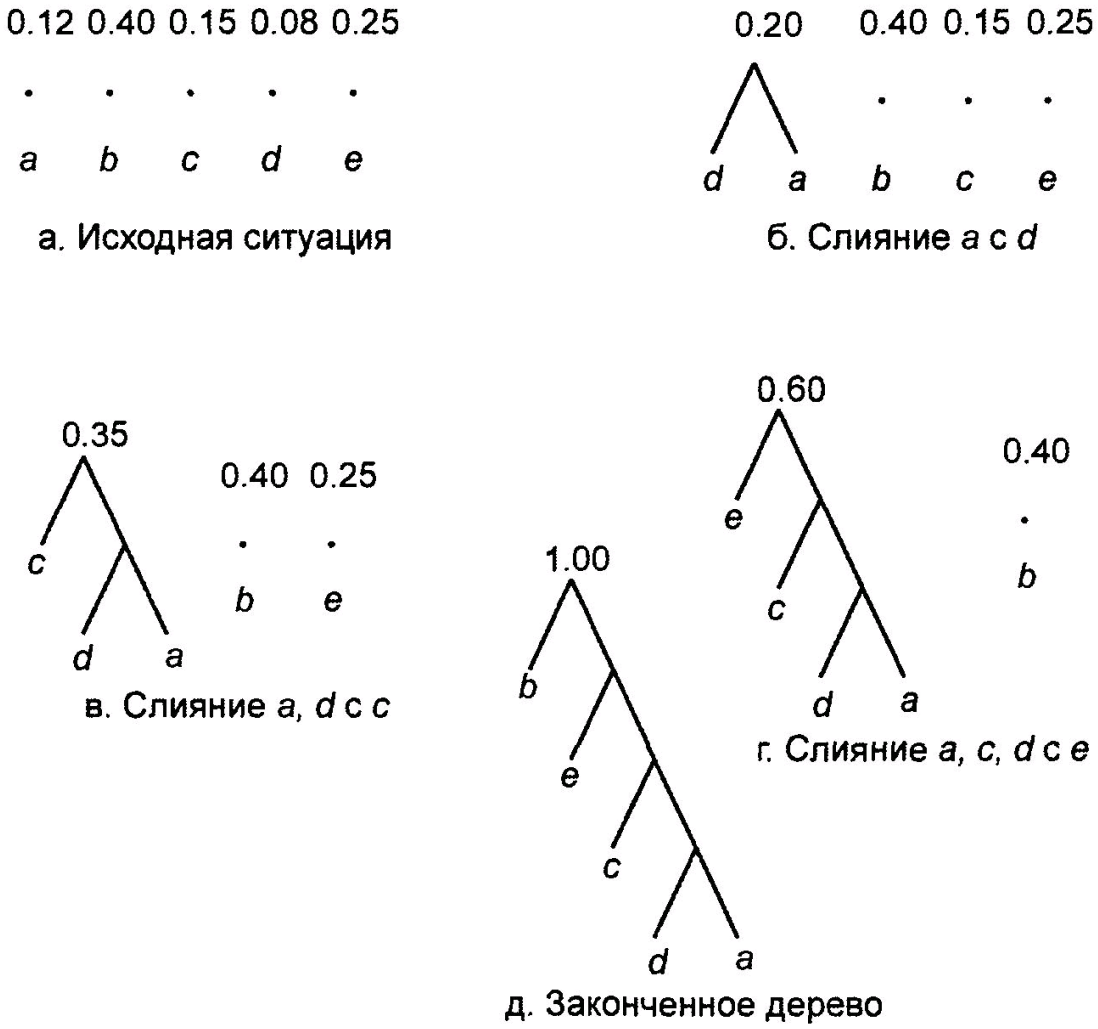


Рисунок 4.2 – Этапы построения дерева Хаффмана

На рисунке видно, что символы *a*, *b*, *c*, *d* и *e* получили соответственно коды 1111, 0, 110, 1110 и 10. В этом примере существует только одно нетривиальное дерево, соответствующее оптимальному коду, но в общем случае их может быть несколько. Например, если бы символы *b* и *e* имели вероятности соответственно 0,33 и 0,32, то после шага алгоритма, показанного на рисунке 4.2, в, можно было бы комбинировать *b* и *e,* а не присоединять *e* к большому дереву, как это сделано на рисунке 4.2 г.

Для представления бинарных деревьев используется массив *TREE*, состоящий из записей следующего типа:

```
record
    leftchild: integer;
    rightchild: integer;
    parent: integer
end
```

Также используется массив *ALPHABET*, состоящий из записей, имеющих следующий тип:

```
record
    symbol: char;
    probability: real;
    leaf: integer { курсор }
end
```

Для представления непосредственно деревьев необходим массив *FOREST*, состоящих из записей, имеющих тип

```
record
    weight: real;
    root: integer
end
```

Исходное состояние трех массивов показано на рисунке 4.3.

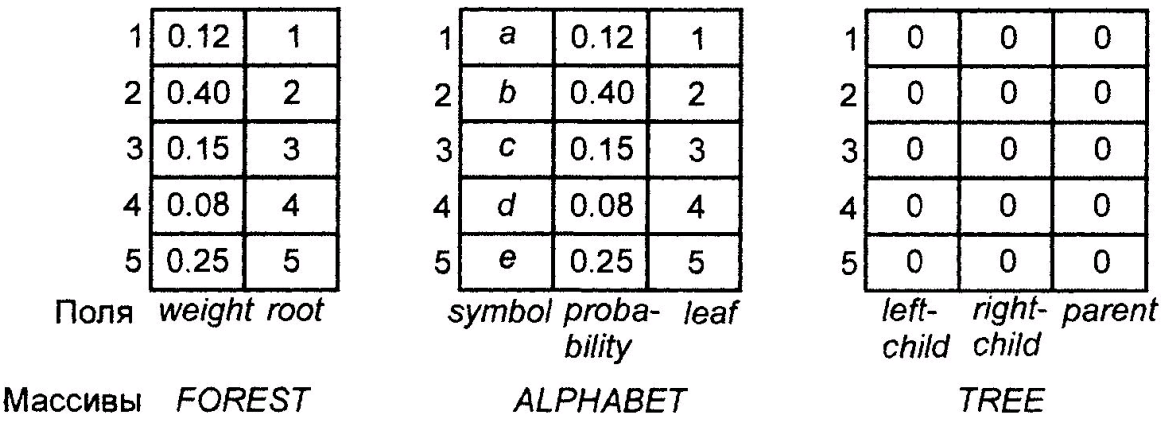


Рисунок 4.3 – Исходное состояние массивов

На рисунке 4.4 показано состояние трех массивов, соответствующее лесу, представленному на рисунке 4.2, в.

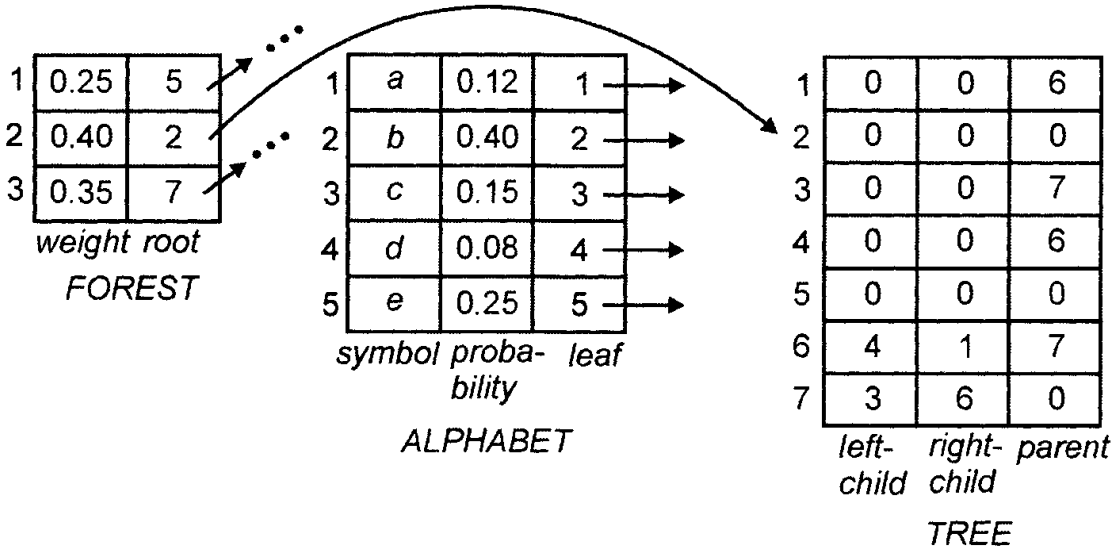


Рисунок 4.4 – Промежуточное состояние массивов

После завершения работы алгоритма код каждого символа можно определить следующим образом. Найти в массиве *ALPHABET* запись с нужным символом в поле *symbol*. Затем по значению поля *leaf* этой же записи определить местоположение записи в массиве *TREE*, которая соответствует листу, помеченному рассматриваемым символом. Далее нужно последовательно переходить по указателю *parent* от текущей записи, например, соответствующей узлу *n*, к записи в массиве *TREE*, соответствующей его родителю *p*. По родителю *p* определяют, в каком его поле, *leftchild* или *rightchild*, находится указатель на узел *n*, т. е. является ли узел *n* левым или правым сыном, и в соответствии с этим печатаются 0 (для левого сына) или 1(для правого сына). Затем выполняют переход к родителю узла *p* и определяют, является ли его сын *p* правым или левым, и в соответствии с этим печатают следующую 1 или 0. Таким образом продолжают до корня дерева. В результате код символа будет напечатан в виде последовательности битов, но в обратном порядке. Чтобы распечатать полученную последовательность в прямом порядке, нужно каждый очередной бит помещать в стек, а затем распечатать содержимое стека в обычном порядке.

Лабораторная работа №5

МЕТОД АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ТЕКСТОВОЙ ИНФОРМАЦИИ

Цель работы: научиться синтезировать текстовую информацию с помощью синтаксических методов.

Порядок выполнения работы:

1 Ознакомиться с теоретической частью работы.

2 Реализовать алгоритм автоматического синтеза текстовой информации с помощью синтаксического метода.

3 Оформить отчет по лабораторной работе.

Исходные данные: исходное множество терминальных цепочек.

Выходные данные: грамматика, способная автоматически порождать заданные цепочки, а также бесконечное множество цепочек, схожих по структуре с исходными.

5.1 Формальные грамматики

Синтаксические методы синтеза и распознавания основаны на восприятии основных элементов языка – *примитивов*. Они делятся на еще более мелкие составляющие – *символы*, являющиеся наименьшими элементами языка. Множество используемых символов называется *алфавитом* или *словарем.* Язык создается не только с помощью алфавита символов. Правила построения, преобразования и взаимодействия слов определяются *грамматикой.* Она представляет собой множество правил, по которым строятся фразы, а следовательно, и сам язык.

Формально грамматика может быть задана следующей записью:

$$G =< V_n, V_t, P, S >,$$

где V_n – нетерминальный словарь;

V_t – терминальный словарь;

P – множество правил подстановки;

S – начальная аксиома $(S \in V_n)$.

Для грамматики характерны следующие соотношения:

$V = V_n \cup V_t$ – словарь, $V_n \cap V_t = 0$;

$$P = \{\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, ..., \alpha_m \to \beta_m\},$$

где $\alpha_i \in V^* - \{\lambda\}$, $\beta_i \in V^*$, $\{\lambda\}$ – пустая строка.

Здесь V^* – множество всех возможных последовательностей, которые удается построить с помощью итерационных процедур на основе данного словаря.

Процесс создания языка начинается с аксиомы S, к которой применяются одно за другим правила подстановок. Основным вопросом после определения грамматики является разработка процедуры грамматического разбора, устанавливающей, является или нет рассматриваемый объект предложением языка, созданного на основе грамматики.

5.2 Грамматический вывод

Используя лингвистическую терминологию, процедуру получения решений с помощью обучающей выборки легко интерпретировать как задачу получения грамматики из множества выборочных предложений. Эта процедура называется грамматическим выводом и играет важную роль в изучении синтаксического распознавания образов в связи с ее значением для реализации автоматического обучения. Тем не менее область грамматического вывода находится еще в начальной стадии развития. На рисунке 5.1 представлена модель вывода цепочечных грамматик.

Рисунок 5.1 – Модель вывода цепочечных грамматик

Задача, показанная на рисунке 5.1, заключается в том, что множество выборочных цепочек подвергается обработке с помощью адаптивного обучающего алгоритма, представленного блоком. На выходе этого блока в конечном счете воспроизводится грамматика G, согласованная с данными цепочками, т. е. множество цепочек $\{x_i\}$ является подмножеством языка $L(G)$. Пока ни одна из известных схем не в состоянии решить эту задачу в общем виде. Вместо этого предлагаются многочисленные алгоритмы для вывода ограниченных грамматик. Рассмотрим один из алгоритмов, в котором сначала строят нерекурсивную грамматику, порождающую в точности заданные цепочки, а затем, сращивая нетерминальные элементы, получают более простую рекурсивную грамматику, порождающую бесконечное число цепочек. Алгоритм можно разделить на три части. Первая часть формирует нерекурсивную грамматику. Вторая часть

преобразует ее в рекурсивную грамматику. В третьей части происходит упрощение этой грамматики.

Рассмотрим выборочное множество терминальных цепочек (*caaab, bbaab, caab, bbab, cab, bbb, cb*). Требуется получить грамматику, способную автоматически порождать эти цепочки. Алгоритм построения грамматики состоит из следующих этапов.

1 *часть*. Строится нерекурсивная грамматика, порождающая в точности заданное множество выборочных цепочек. Они обрабатываются в порядке уменьшения длины. Правила подстановки строятся и прибавляются к грамматике по мере того, как они становятся нужны для построения соответствующей цепочки из выборки. Заключительное правило подстановки, используемое для порождения самой длинной выборочной цепочки, называется *остаточным правилом*, а длина его правой части равна 2 (это значение выбрано для удобства алгоритма). Остаточное правило дины *n* имеет вид

$$A->a_1a_{2...}a_n,$$

где A – нетерминальный символ, а $a_1a_{2...}a_n$ – терминальные элементы. Предполагается, что остаток каждой цепочки максимальной длины является суффиксом (хвостом) некоторой более короткой цепочки. Если какой-либо остаток не отвечает этому условию, цепочка, равная остатку, добавляется к обучающей выборке.

В нашем примере первой цепочкой максимальной длины в обучающей выборке является *caaab*. Для ее порождения строятся следующие правила подстановки:

$$S->cA_1,\ A_1->aA_2,\ A_2->aA_3,\ A_3->ab,$$

где A_3 – правило остатка.

Вторая цепочка – *bbaab*. Для ее порождения к грамматике добавляются следующие правила: $S->bA_4,\ A_4->bA_5,\ A_5->aA_6,\ A_6->ab$. Поскольку цепочки *bbaab* и *caaab* имеют одинаковую длину, требуется остаточное правило длины 2. Работа первой части алгоритма приводит к некоторой избыточности правил подстановки. Например, вторая цепочка может быть также получена введением следующих правил подстановки: $S->bA_4,\ A_4->bA_2$. Но первая часть алгоритма занимается лишь определением множества правил постановки, которое способно в точности порождать обучающую выборку, и не касается вопроса избыточности. Устранение избыточности выполняется в третьей части алгоритма. Для порождения третьей цепочки *caab* требуется добавление к грамматике только одного правила $A_3->b$. Рассмотрев остальные цепочки из обучающей выборки, устанавливаем, что окончательно множество правил подстановки для порождения выборки выглядит так:

$$S->cA_1,\ S->bA_4,\ A_1->aA_2,\ A_1->b,\ A_2->aA_3,\ A_2->b,\ A_3->ab,\ A_3->b,\ A_4->bA_5,$$
$$A_5->aA_6,\ A_5->b,\ A_6->ab,\ A_6->b.$$

2 *часть*. Здесь, соединяя каждое правило остатка длиной 2 с другим (неостаточным) правилом грамматики, получаем рекурсивную автоматную

грамматику. Это происходит в результате слияния каждого нетерминального элемента правила остатка с нетерминальным элементом неостаточного правила, который может порождать остаток. Например, если A_r – остаточный нетерминал вида A_r->a_1a_2, и A_n – неостаточный нетерминал вида A_n ->a_1A_m, где A_m->a_2, все встречающиеся A_r заменяются на A_n, а правило подстановки A_r->a_1a_2 отбрасывается. Так создается автоматная грамматика, способная порождать данную обучающую выборку, а также обладающая общностью, достаточной для порождения бесконечного множества других цепочек. В рассматриваемом примере A_6 может сливаться с A_5, а A_3 может сливаться с A_2, образуя следующие правила подстановки:

S->cA_1, S->bA_4, A_1->aA_2, A_1->b, A_2 ->aA_2, A_2->b, A_2->b, A_4->bA_5, A_5->aA_5, A_5->b, A_5->b.

Рекурсивными правилами являются A_2 ->aA_2 *и* A_5->aA_5.

3 *часть*. Грамматика, полученная во 2 части, упрощается объединением эквивалентных правил подстановки. Два правила с левыми частями A_i и A_j эквивалентны, если выполняются следующие условия. Предположим, что, начиная с A_i, можно породить множество цепочек $\{x\}_i$ и, начиная с A_j, можно породить множество цепочек $\{x\}_j$. Если $\{x\}_i = \{x\}_j$, то два правила подстановки считаются эквивалентными, и каждый символ A_j может быть заменен на A_i без ущерба для языка, порождаемого этой грамматикой, т.е. два правила эквивалентны, если они порождают тождественные цепочки языка.

В рассматриваемом примере эквивалентны правила с левыми частями A_1 и A_2. После слияния A_1 и A_2 получаем

S->cA_1, S->bA_4, A_1->aA_1, A_1->b, A_4->bA_5, A_5->aA_5, A_5->b, где исключены многократные повторения одного и того же правила. Теперь ясно, что эквиваленты A_1 и A_5, выполним преобразования для них и получим

S->cA_1, S->bA_4, A_1->aA_1, A_1->b, A_4->bA_4.

Дальнейшее слияние правил невозможно, поэтому алгоритм в процессе обучения строит следующую автоматную грамматику:

$$G=(V_N, V_T, P, S).\ V_N=(S,A,B),\ V_T=(a,b,c),$$

$$P:\ S\text{->}cA,\ S\text{->}bB,\ A\text{->}aA,\ B\text{->}bA,\ A\text{->}b.$$

Можно легко проверить, что данная грамматика порождает обучающую выборку, использованную в процессе построения грамматики.

Лабораторная работа №6

АВТОМАТИЧЕСКАЯ РУБРИКАЦИЯ ТЕКСТОВОЙ ИНФОРМАЦИИ ПО ОБРАЗЦУ

Цель работы: научиться классифицировать по темам текстовую информацию, используя документы-образцы.

Порядок выполнения работы:

1 Ознакомиться с теоретической частью работы.

2 Реализовать алгоритм классификации текстов по рубрикам на основе документов-образцов.

3 Оформить отчет по лабораторной работе.

Исходные данные: тексты-образцы, о которых известно, к каким рубрикам они относятся, и тексты, требующие тематической классификации по заданным рубрикам.

Выходные данные: список рубрик с принадлежащими им текстовыми документами.

6.1 Алгоритм автоматической рубрикации текстов по образцу

1 Применение законов Зипфа для построения векторов признаков (ключевых слов) текстов-образцов, задающих необходимые рубрики.

2 Обучение системы классификации с помощью текстов-образцов. Применение метода персептрона для построения p линейных функций (по числу классов), отделяющих в r-мерном пространстве признаков каждый класс от всех остальных.

3 Построение по законам Зипфа наборов ключевых слов для текстов, требующих рубрикации.

4 Рубрикация текстов с помощью разделяющих функций, построенных на втором шаге алгоритма.

Рассмотрим более подробно каждый из шагов алгоритма.

1 Для построения векторов с признаками текстов-образцов создадим словарь терминов. Каждая рубрика может быть задана произвольным количеством текстов-образцов. Пусть векторы признаков всех рубрик, чье количество обозначим a, имеют равное количество ключевых слов – b. Тогда словарь ключевых слов можно представить следующей динамической структурой (рисунок 6.1). Где в вертикальной таблице расположены заголовки классов: от 1 до a. Каждый элемент таблицы содержит ссылку на список ключевых слов данного класса. Списки могут

быть разной длины в зависимости от количества текстов-образцов, имеющихся для каждой рубрики.

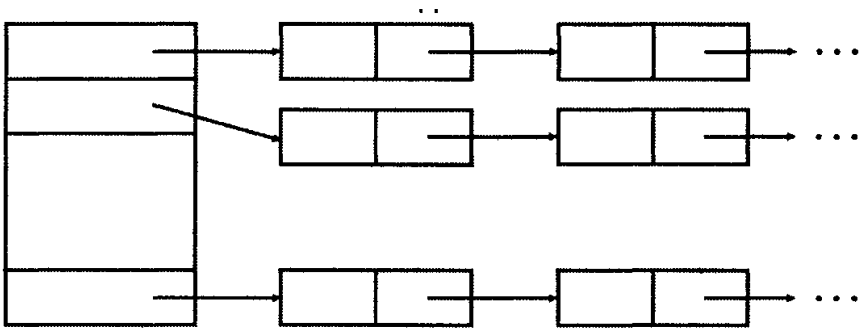
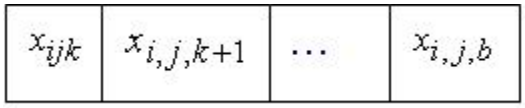


Рисунок 6.1 – Структура словаря терминов рубрик

Вектор признаков каждого класса представим следующей структурой.

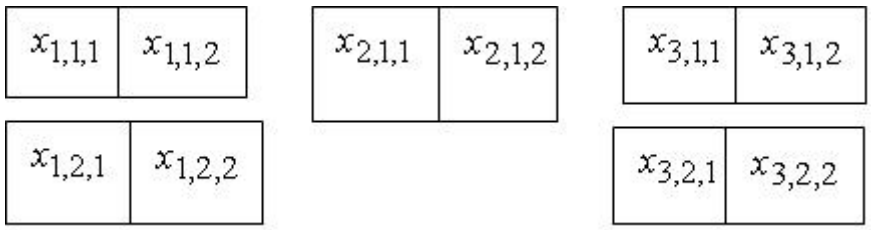


Здесь x – это ключевые слова j-го текста-образца i-й рубрики.

Индекс i изменяется от 1 до a, индекс j зависит от представленного количества образцов для данной рубрики. Индекс k – это номер ключевого слова в векторе, поэтому он изменяется от 1 до b.

На основе векторов признаков всех образцов для всех рубрик определим структуру обобщенного вектора признаков, на базе которого будем строить разделяющие функции. В обобщенный вектор войдут все ключевые слова из всех текстов-образцов. Рассмотрим на примере построение обобщенного вектора ключевых слов.

Пусть имеются следующие исходные данные: $a = 3$, $b = 2$. Рубрики заданы следующими векторами признаков.



Для первого класса предложены два образца, для второго – один и для третьего – два образца. При этом второй признак образца второго класса совпадает с первым признаком второго образца третьего класса, т. е. $x_{2,1,2} = x_{3,2,1}$.

Тогда структура обобщенного вектора признаков будет следующей:

$x_{1,1,1}$	$x_{1,1,2}$	$x_{1,2,1}$	$x_{1,2,2}$	$x_{2,1,1}$	$x_{2,1,2}$	$x_{3,1,1}$	$x_{3,1,2}$	$x_{3,2,1}$	$x_{3,2,2}$

Теперь каждый из пяти имеющихся текстов-образцов будет также описан вектором из десяти элементов. В позиции вектора, относящиеся к ключевым словам, имеющимся в данном образце, занесем 1 (позже там будем стоять частота встречаемости слова в тексте, определенная по законам Зипфа), в остальные позиции запишем нули. Получим следующие векторы признаков для имеющихся пяти текстов-образцов (рисунок 6.2).

1	1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0	1	0
0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	1	0	0	1	1

Рисунок 6.2 – Векторы признаков для построения разделяющих функций

2 После того как для каждого текста-образца составлен вектор признаков (рисунок 6.2), применим метод персептрона для построения разделяющих функций по числу заданных рубрик (метод персептрона описан ниже).

3 Вектор признаков незнакомого текста имеет такую же структуру и размер, как и вектор признаков текста-образца. Сначала определим вектор признаков, состоящий из b элементов, незнакомого текста. Затем проверим, есть ли среди признаков ключевые слова, входящие в обобщенный вектор. Пусть первоначально вектор признаков незнакомого текста имеет вид

y_1	y_2

При этом $y_1 = x_{1,1,2;}\ y_2 = x_{2,1,2} = x_{3,2,1}$. Тогда вектор признаков незнакомого текста примет следующий вид (рисунок 6.3):

0	1	0	0	0	1	0	0	1	0

Рисунок 6.3 – Вектор признаков незнакомого текста

Вектор признаков незнакомого текста подставляется в каждую из разделяющих функций, и текст относится к той рубрике, чья функция показала максимальный результат.

6.2 Разделение объектов на *N* классов методом персептрона

В настоящее время существует большое множество методов классификации, в том числе и текстовой информации. Однако все методы распознавания можно разделить на две группы. Первая основана на понятии пространства признаков и их обработки в этом пространстве. Вторая – на исследовании конструкции рассматриваемых образов (синтаксическое распознавание).

Для решения поставленной задачи рассмотрим особенности первой группы методов. Для них в качестве основополагающей принята гипотеза о возможности представления образа в виде вектора, принадлежащего множеству V. Множество образов представляется в виде множества векторов, состоящего из таких N подмножеств, что каждый вектор, отнесенный в результате классификации к j-му классу, принадлежит подмножеству E_j.

Свойства множества V могут быть записаны в виде

$$\bigcup_{i=1}^{N} E_i = V, E_i \cap E_j = 0 (\forall i = j).$$

Задача классификации состоит в отыскании функции f, обеспечивающей разделение пространства V на требуемые классы:

$$f: V \text{->} \Pi(V).$$

Процедура классификации состоит в том, чтобы для каждой области R_l найти такую решающую функцию $g_l(x)$, что если

$$g_i(\bar{x}) > g_j(\bar{x}),\ mo\ \bar{x} \in R_i\ \forall j = 1,2...N,$$

где N – общее количество областей.

Разделяющую функцию часто представляют в виде линейной суммы

$$g(\bar{x}) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n,$$

где ω_i – весовые коэффициенты, каждый из которых относится к определенной составляющей.

Для удобства записи вводится весовой коэффициент с нулевым индексом ω_0. Это позволяет записать решающую функцию в более компактной форме:

$$g(\bar{x}_a) = \bar{\omega}\bar{x}_a,$$

где $\bar{x}_a = \{1, x_1, x_2 ... x_n\}$ – вектор, в число составляющих которого входит дополнительно одна вещественная константа, чью величину обычно принимают равной единице.

Решающее правило *d* для случая *N* сепарабельных классов *(N>2)* можно записать следующим образом:

$$d = \begin{cases} c_i, \text{если } g_i(\bar{x}) = \bar{\omega}_i \bar{x}_a \geq 0, \\ \bar{c}_i, \text{если } g_i(\bar{x}) < 0 \end{cases},$$

где *C* – множество, состоящее из *N* классов. $C = \{c_1, c_2, ... c_N\}, c_i + \bar{c}_i = C.$

В процессе построения разделяющей функции основная задача заключается в том, чтобы найти весовые коэффициенты вида $\bar{\omega}_i = \{\omega_{0i}, \omega_{1i} ...\}$ для каждого конкретного применения.

Рассмотрим один из вариантов применения линейных разделяющих функций для разбиения объектов на *N* классов.

Существует *M* решающих функций $d_k(x) = w_k x, k = 1, 2, ... M,$ таких, что если образ *x* принадлежит классу ω_i, то $d_i(x) > d_j(x)$ для всех $j \neq i$.

Граница между классами ω_i и ω_j определяется теми значениями вектора *x*, при которых имеет место равенство $d_i(x) = d_j(x)$. Поэтому при выводе уравнения разделяющей границы для классов ω_i и ω_j значения решающих функций $d_i(x)$ и $d_j(x)$ используются совместно.

Пример расположения разделяющих функций приведен на рисунке 6.4.

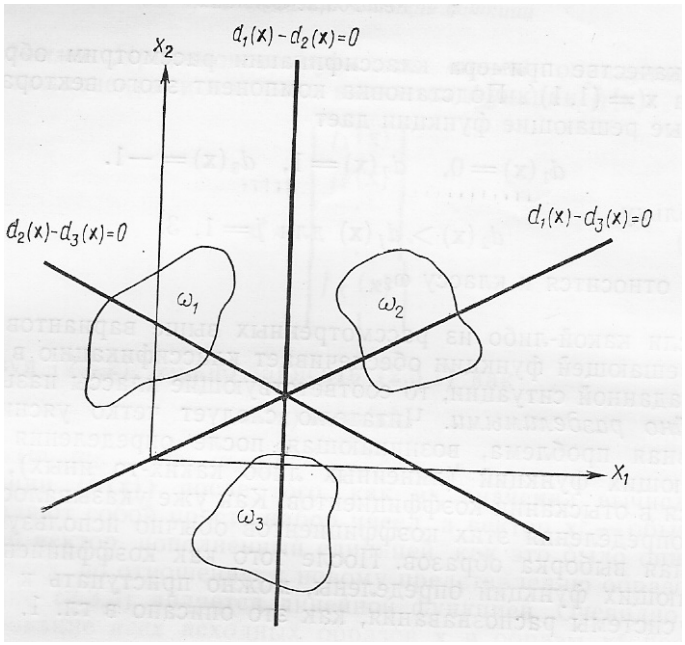


Рисунок 6.4 – Случай разделения образов на три класса

Для образов, принадлежащих классу ω_1, должны выполняться условия $d_1(x) > d_2(x),\ d_1(x) > d_3(x)$. В общем случае требуется, чтобы входящие в класс ω_i образы располагались в положительных зонах поверхностей $d_i(x) - d_j(x) = 0, j = 1,2,...M, i \neq j$. Положительная зона границы $d_i(x) - d_j(x) = 0$ совпадает с отрицательной зоной границы $d_j(x) - d_i(x) = 0$.

Пусть в качестве решающих функций выбраны следующие:

$d_1(x) = -x_1 + x_2,\ d_2(x) = x_1 + x_2 - 1,\ d_3(x) = -x_2$. Разделяющие границы для трех классов выглядят при этом так:

$$d_1(x) - d_2(x) = -2x_1 + 1 = 0,$$
$$d_1(x) - d_3(x) = -x_1 + 2x_2 = 0,$$
$$d_2(x) - d_3(x) = x_1 + 2x_2 - 1 = 0.$$

Для того чтобы определить область решений, соответствующую классу ω_1, необходимо выделить область, в которой выполняются неравенства $d_1(x) > d_2(x),\ d_1(x) > d_3(x)$. Эта область совпадает с положительными зонами для прямых $-2x_1 + 1 = 0$ и $-x_1 + 2x_2 = 0$.

Область принятия решения о принадлежности образа классу ω_2 совпадает с положительными зонами для прямых $2x_1 - 1 = 0$ и $x_1 + 2x_2 - 1 = 0$. Область, отвечающая классу ω_3, определяется положительными зонами для прямых $x_1 - 2x_2 = 0$ и $-x_1 - 2x_2 + 1 = 0$ (рисунок 6.5).

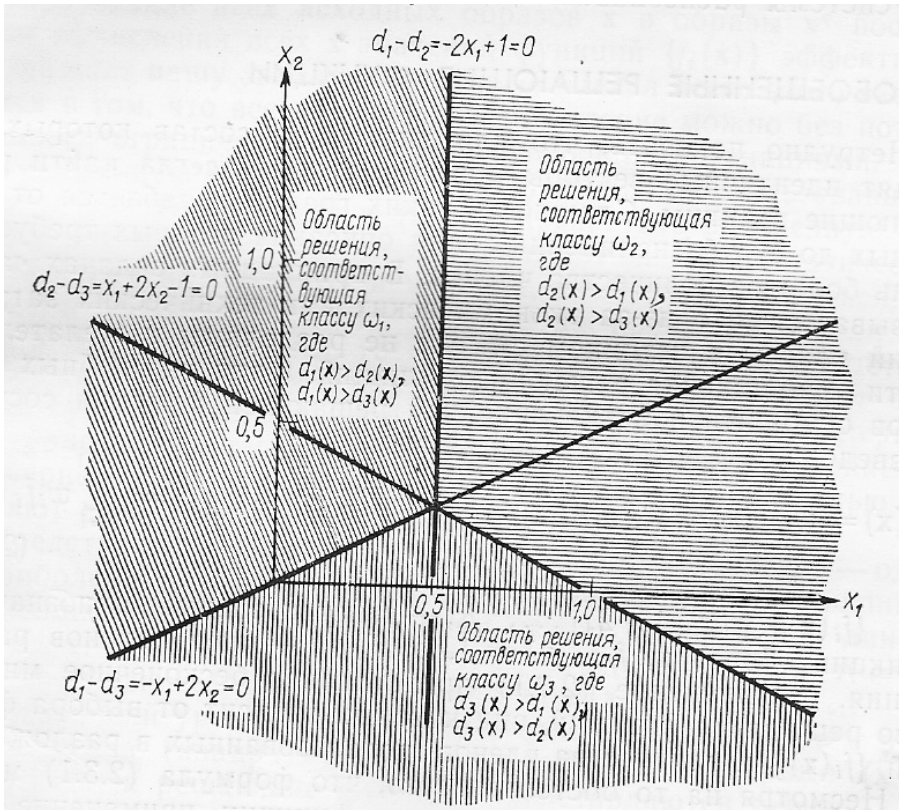


Рисунок 6.5 – Иллюстрация разделения объектов на несколько классов

В качестве примера классификации рассмотрим обработку образа x = (1,1). Подстановка признаков образа в выбранные решающие функции дает следующие значения:

$$d_1(x)=0,\ d_2(x)=1,\ d_3(x)=-1.$$

Поскольку $d_2(x)>d_j(x), j=1,3,$ образ относится к классу ω_2.

Если какой-либо из рассмотренных вариантов линейной решающей функции обеспечивает классификацию в некоторой заданной ситуации, то соответствующие классы называются *линейно разделимыми.* Основная проблема, возникающая после определения набора решающих функций, заключается в отыскании коэффициентов. Для их определения обычно используется доступная выборка образов. После того как коэффициенты всех решающих функций определены, можно приступать к построению системы распознавания.

Рассмотрим алгоритм – *метод персептрона*, который можно применить для определения решающих функций, когда допускается существование *M* решающих функций, характеризующихся тем свойством, что при $x\in\omega_i$, где x – объект, ω_i – класс $d_i(x)>d_j(x)$ для всех $i\neq j$.

Рассмотрим *M* классов $\omega_1,\omega_2,...\omega_M$. Пусть на *к*-м шаге процедуры обучения системе предъявляется образ *x(к)*, принадлежащий классу ω_i. Вычисляются значения *M* решающих функций $d_j[x(k)]=w_j(k)x(k), j=1,2...,M$. Затем, если выполняются условия

$d_i[x(k)] > d_j[x(k)]$, $j = 1,2,...,M; j \neq i,$ то векторы весов не изменяются, т.е. $w_j(k+1) = w_j(k), j = 1,2,...M.$

С другой стороны, допустим, что для некоторого l $d_i[x(k)] \leq d_l[x(k)]$. В этом случае выполняются следующие коррекции весов:

$$\begin{aligned} w_i(k+1) &= w_i(k) + cx(k), \\ w_l(k+1) &= w_l(k) - cx(k), \\ w_j(k+1) &= w_j(k), j = 1,2, ...M; j \neq i, j \neq l, \end{aligned} \qquad (6.1)$$

где с – положительная константа.

Если классы разделимы, то доказано, что этот алгоритм сходится за конечное число итераций при произвольных начальных векторах.

Рассмотрим это на примере.

Даны классы, причем каждый из них содержит один образ: ω_1: {(0, 0)}, ω_2: {(1,1}, ω_3: {(-1, 1)}. Дополним заданные образы: (0, 0, 1), (1, 1, 1), (-1, 1, 1). Выберем в качестве начальных векторов весов $w_1(1) = w_2(1) = w_3(1) = (0,0,0),$ положим c = 1 и, предъявляя образы в указанном порядке, получим следующее:

$$\begin{aligned} d_1[x(1)] &= w_1(1)x(1) = 0, \\ d_2[x(1)] &= w_2(1)x(1) = 0, \\ d_3[x(1)] &= w_3(1)x(1) = 0. \end{aligned}$$

Поскольку $x(1) \in \omega_1$ и $d_2[x(1)] = d_3[x(1)] = d_1[x(1)],$ первый весовой вектор увеличивается, а два других уменьшаются согласно соотношениям (6.1), т.е.

$$w_1(2) = w_1(1) + x(1) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$

$$w_2(2) = w_2(1) - x(1) = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix},$$

$$w_3(2) = w_3(1) - x(1) = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}.$$

Следующий предъявляемый образ $x(2)$ = (1, 1, 1) принадлежит классу ω_2.

Для него получаем $w_1(2)x(2) = 1, w_2(2)x(2) = -1, w_3(2)x(2) = -1.$ Поскольку все произведения больше либо равны $w_2(2)x(2)$, вводятся коррекции

$$w_1(3) = w_1(2) - x(2) = \begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix},$$

$$w_2(3) = w_2(2) + x(2) = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix},$$

$$w_3(3) = w_3(2) - x(2) = \begin{pmatrix} -1 \\ -1 \\ -2 \end{pmatrix}.$$

Следующий предъявленный образ $x(3)$ = (-1, 1, 1) принадлежит классу ω_3. Для него получаем $w_1(3)x(3) = 0,\ w_2(3)x(3) = 0, w_3(3)x(3) = -2$. Все эти произведения опять требуют корректировки:

$$w_1(4) = w_1(3) - x(3) = \begin{pmatrix} 0 \\ -2 \\ -1 \end{pmatrix},$$

$$w_2(4) = w_2(3) - x(3) = \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix},$$

$$w_3(4) = w_3(3) + x(3) = \begin{pmatrix} -2 \\ 0 \\ -1 \end{pmatrix}.$$

Поскольку в данном цикле итерации присутствовали ошибки, следует провести новый цикл. Положив $x(4)=x(1)$, $x(5)=x(2)$, $x(6)=x(3)$, получим $w_1(4)x(4) = -1,\ w_2(4)x(4) = -1, w_3(4)x(4) = -1$. Так как образ $x(4)$ принадлежит классу ω_1, то все произведения «неверны». Поэтому

$$w_1(5) = w_1(4) + x(4) = \begin{pmatrix} 0 \\ -2 \\ 0 \end{pmatrix},$$

$$w_2(5) = w_2(4) - x(4) = \begin{pmatrix} 2 \\ 0 \\ -2 \end{pmatrix},$$

$$w_3(5) = w_3(4) - x(4) = \begin{pmatrix} -2 \\ 0 \\ -2 \end{pmatrix}.$$

Следующий предъявленный образ $x(5) = (1, 1, 1)$ принадлежит классу ω_2. Соответствующие скалярные произведения равны $w_1(5)x(5) = -2$, $w_2(5)x(5) = 0$, $w_3(5)x(5) = -4$. Образ $x(5)$ классифицирован правильно. Поэтому

$$w_1(6) = w_1(5) = \begin{pmatrix} 0 \\ -2 \\ 0 \end{pmatrix},$$

$$w_2(6) = w_2(5) = \begin{pmatrix} 2 \\ 0 \\ -2 \end{pmatrix},$$

$$w_3(6) = w_3(5) = \begin{pmatrix} -2 \\ 0 \\ -2 \end{pmatrix}.$$

Следующий образ $x(6)$=(-1, 1, 1) принадлежит классу ω_3, для него получаем $w_1(6)x(6) = -2$, $w_2(6)x(6) = -4, w_3(6)x(6) = -0$. Этот образ также классифицирован правильно, так что коррекции не нужны, т. е.

$$w_1(7) = w_1(6) = \begin{pmatrix} 0 \\ -2 \\ 0 \end{pmatrix},$$

$$w_2(7) = w_2(6) = \begin{pmatrix} 2 \\ 0 \\ -2 \end{pmatrix},$$

$$w_3(7) = w_3(6) = \begin{pmatrix} -2 \\ 0 \\ -2 \end{pmatrix}.$$

Если продолжить процедуру обучения, рассматривая образы $x(7)$, $x(8)$, $x(9)$, можно убедиться, что в следующем полном цикле никакие коррекции не производятся. Поэтому искомые решающие функции имеют следующий вид:

$$d_1(x) = 0 \cdot x_1 - 2x_2 + 0 = -2x_2,$$
$$d_2(x) = 2x_1 - 0 \cdot x_2 - 2 = 2x_1 - 2,$$
$$d_3(x) = -2x_1 + 0 \cdot x_2 - 2 = -2x_1 - 2.$$

Теперь, получив объект, требующий классификации, необходимо его признаки подставить в каждую из решающих функций и в качестве результата выбрать функцию (класс), на которой будет достигнуто максимальное значение.

Лабораторная работа №7

МЕТОД АВТОМАТИЧЕСКОГО РЕФЕРИРОВАНИЯ ТЕКСТОВОЙ ИНФОРМАЦИИ

Цель работы: научиться строить реферат на основе предложенного текстового документа.

Порядок выполнения работы:
1 Ознакомиться с теоретической частью работы.
2 Реализовать алгоритм автоматического синтеза реферата.
3 Оформить отчет по лабораторной работе.

Исходные данные: текстовый документ и величина реферата, указанная в процентах относительно исходного текста.
Выходные данные: автоматически построенный реферат.

7.1 Автоматическое реферирование и аннотирование текстов

Аннотирование текста заключается в формировании краткого описания его основных тем. Существует два разных подхода к аннотированию. В первом случае выявляется небольшое количество предложений, существующих в тексте, которые наиболее полно отражают основные темы текста. Дополнительно часто выделяются ключевые слова. Во втором случае основные темы текста выявляются как смыслы, и уже эти смыслы выражаются новыми предложениями, новым текстом. Второй вариант в большинстве случаев значительно более предпочтителен, но он и значительно сложнее. Все современные системы аннотирования/реферирования основаны на первом варианте. SemLP-технология позволяет реализовать второй вариант в ограниченном виде: автоматический синтез коротких (в несколько слов) простых фраз или предложений. В целом задача аннотирования включает определение тематики документов, выделение ключевых (по темам) слов и фраз с учетом смысла, поиск предложений, содержащих ключевые слова и фразы, и синтез на этой основе фраз и предложений, отражающих основные темы текста.

В рамках первого подхода выделяют три основных направления, которые в современных системах применяются совместно:
– статистические методы, основанные на оценке информативности разных элементов текста по частоте появления, которая служит основным критерием информативности слов, предложений или фраз;
– позиционные методы, которые опираются на предположение о том, что информативность элемента текста зависит от его позиции в документе;
– индикаторные методы, основанные на оценке элементов текста, исходя из наличия в них специальных слов и словосочетаний – маркеров важности, которые характеризуют их содержательную значимость.

После выявления набора ключевых слов по ним строится реферат. Преимущество методов первого подхода заключается в простоте их реализации. Однако выделение текстовых блоков, не учитывающее взаимоотношений между ними, часто приводит к формированию бессвязных рефератов. Некоторые предложения могут оказаться пропущены, либо в них могут встречаться слова или фразы, которые невозможно понять без предшествующего пропущенного текста. Попытки решить эту проблему в основном сводятся к исключению таких предложений из рефератов. Реже делаются попытки разрешения ссылок с помощью методов лингвистического анализа.

Краткое изложение содержания первичных документов основывается на выделении из текстов наиболее важной информации и порождении новых текстов, содержательно обобщающие первичные документы. В отличие от частотно-лингвистических методов подход, основанный на базах знаний, опирается на автоматизированный качественный контент-анализ, состоящий, как правило, из трех основных стадий. Первая – сведение исходной текстовой информации к заданному числу фрагментов, которыми являются категории, последовательности и темы. На второй стадии производится поиск регулярных связей между фрагментами, после чего начинается третья стадия – формирование выводов и обобщений. На этой стадии создается структурная аннотация, представляющая содержание текста в виде совокупности концептуально связанных смысловых единиц.

Семантические методы формирования рефератов-изложений предполагают два основных подхода: метод синтаксического разбора предложений и методы, опирающиеся на понимание естественного языка. В первом случае используются деревья разбора текста. Процедуры автоматического реферирования манипулируют непосредственно деревьями, выполняя перегруппировку и сокращение ветвей на основании заданных критериев. Такое упрощение обеспечивает построение реферата, т. е. структурную «выжимку» исходного текста.

Второй подход основывается на системах искусственного интеллекта, в которых также на этапе анализа выполняется синтаксический разбор текста, но синтаксические деревья не порождаются. В этом случае

формируются семантические структуры, которые накапливаются в виде концептуальных подграфов в базе знаний. В частности, известны модели, позволяющие производить реферирование текстов на основе психологических ассоциаций сходства и контраста. В базах знаний избыточная и не имеющая прямого отношения к тексту информация устраняется путем отсечения некоторых подграфов. Затем информация подвергается агрегированию методом слияния оставшихся графов или их обобщения. Для выполнения этих преобразований выполняются манипуляции логическими предположениями, выделяются определяющие шаблоны в текстовой базе знаний. В результате преобразования формируется концептуальная структура текста – аннотация, т. е. концептуальные «выжимки» из текста.

Многоуровневое структурирование текста с использованием семантических методов позволяет подходить к решению задачи реферирования путем выполнения следующих операций.

Удаление малозначащих смысловых единиц. Преимуществом метода является гарантированное сохранение значащей информации, недостатком – низкая степень сжатия, т. е. сокращения объема реферата по сравнению с первичными документами.

Сокращения смысловых единиц – замена их основной лексической единицей, выражающей основной смысл.

Гибридный способ – уточнение реферата с помощью статистических методов, с использованием семантических классов, особенностей контекста и синонимических связей.

Алгоритм реферирования текстов на базе статистического подхода.

1 Согласно законам Зипфа определяются характеристики исходного текста, и строится строка ключевых слов, т. е. терминов текста.

2 Фиксируется первое ключевое слово, и в реферат добавляются предложения, содержащие данное ключевое слово. При этом сохраняется порядок предложений относительно исходного текста.

3 Если достигнут требуемый размер реферата, алгоритм закончен, иначе выбирается следующее ключевое слово и повторяются шаги 2 и 3.

4 Если обработана вся строка ключевых слов, а заданный размер реферата не достигнут, из области значимых слов (рисунок 1.2) выбирается первое необработанное слово, а затем выполняется переход на шаг 2.

ЛИТЕРАТУРА

1 Вирт , К . Алгоритмы + Структуры данных = Программы / К . Вирт . – М . : Вильямс , 2002 .

2 Грешилов, А . А . Математические методы принятия решений : учеб . пособие для студ . вузов , обучающихся по машиностроит . спец . / А . А . Грешилов – М . : МГТУ им . Н . Э . Баумана , 2006 .

3 Евгенев , Г . Б . Системология инженерных знаний / Г . Б . Евгенев – М . : Машиностроение , 2001 .

4 Вендров , А . М . Проектирование ПО экономических информационных систем / А . М . Вендров . – М . : Наука , 2000 .

5 О'Лири , Д . ERP системы : выбор , внедрение , эксплуатация . Современное планирование и управление ресурсами предприятия / Д . О'Лири . – М . : Вершина , 2004 .

Св.план 2011, поз. 50

Учебное издание

Серебряная Лия Валентиновна

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ФИНАНСОВЫХ СТРУКТУР

Методическое пособие к лабораторным работам
для студентов специальности
«Программное обеспечение информационных технологий»
всех форм обучения

Редактор Н. В. Гриневич

Подписано в печать	Формат 60x84 1/16	Бумага офсетная
Гарнитура «Таймс»	Отпечатано на ризографе	Усл. печ.
Уч.-изд.л. 2,5.	Тираж 100 экз.	Заказ 790.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П.Бровки,6