11.2. ИСПОЛЬЗОВАНИЕ АЛГОРИТМОВ НЕЧЕТКОГО ПОИСКА ПРИ РЕШЕНИИ ЗАДАЧ ОБРАБОТКИ МАССИВОВ ДАННЫХ В ИНТЕРЕСАХ КРЕДИТНЫХ ОРГАНИЗАЦИЙ

Карахтанов Д.С., аспирант

Университет Российской академии образования

В статье описан алгоритм устранения дубликатов записей в базе данных при наличии нескольких источников информации и ошибок операторского ввода.

Предложен алгоритм вычисления функции релевантности на основании метода ***N-gram***, а также способ ускорения его работы с использованием построения частотного префиксного кода.

Полученные решения по использованию алгоритма нечеткого поиска на базе построения ***N-gram*** могут быть применены в качестве типовых при построении автоматизированных информационных систем финансовых структур.

ВВЕДЕНИЕ

Администратор баз данных рано или поздно сталкивается с проблемой управления качеством данных, то есть необходимостью приведения информации к состоянию, которое удовлетворяет требованиям по критериям достоверности, актуальности, логической полноты и непротиворечивости.

Для выполнения всех перечисленных требований понадобится целый комплекс мер, одной из составляющих которого будет обеспечение отсутствия дубликатов.

В любой базе данных можно выделить два основных типа дублирования информации:

- дублирование атрибутов, имеющих жестко заданную структуру (формат) содержания;
- дублирование атрибутов, не имеющих жестко заданной структуры (формата) содержания, т.е. слабоструктурированных.

В первом случае речь идет о различных кодах из различных справочников и классификаторов, идентификаторах сущностей, используемых в качестве ключевых атрибутов поиска (коды организаций, номера документов, счетов, телефонов, ПИН-коды и т.д.).

Во втором случае рассматривается вариант работы с разнообразными именами собственными, используемыми для идентификации людей:

- фамилий, имен, отчеств;
- адресов;
- наименований государственных учреждений;
- названий организаций, фирм и т.д.

Проблема дублирования жестко структурированных атрибутов решается ограничением на ввод данных в соответствующие поля. Например, можно позволить ввод только разрешенных символов в заданном формате либо выбирать все допустимые значения атрибута из справочника. Поиск дубликатов в этом случае ведется по точному совпадению и не вызывает сложностей.

Ситуация со слабоструктурированными полями несколько сложнее, так как нет возможности использовать ограничения формата, а также нельзя применять словари-справочники, поскольку их требуемый объем может выйти за допустимые пределы и многократно превысить объем основной информации базы данных.

Для устранения ошибок операторского ввода и проверки дублирования слабоструктурированной информации предложен алгоритм нечеткого поиска, позволяющий находить дубликаты на основании неполного совпадения и оценки их релевантности – количественного критерия схожести.

Следует учитывать, что данный алгоритм не дает стопроцентной гарантии от ошибок, т.е. сохраняется вероятность того, что будут пропущены дублирующие данные и (или) данные будут распознаны как дубликаты, не являясь таковыми. Поэтому для принятия окончательного решения необходимо участие человека.

Метод ***N-gram*** представляет модель последовательностей (в частности природного языка) с использованием статистических свойств ***N-gram***. Идея данного метода была описана К. Шенноном в работе «Теория информации» и заключалась в том, чтобы, учитывая последовательность букв, рассчитать вероятность появления в последовательности каждой буквы.

Например, для последовательности for ex можно получить распределение:

A = 0,4; B = 0,00001; C = 0; …..,

где вероятности появления всех возможных последующих букв в сумме дают 1,0.

Другими словами, ***N-gram*** модель предсказывает ***x(i)*** на основе значений ***x(i-1), x (i-2), …, x (i-n).*** С вероятностной точки зрения это не что иное, как:

P (x(i) | x(I – 1), x (I – 2), …, x (I – n)).

N-gram модели широко используются в статистической обработке естественного языка. В распознавании речи, фонем и последовательности фонем моделируются распределения с помощью ***N-gram***. Для распознавания языка, последовательности букв моделируются по-разному для каждого языка.

Например, для последовательности слов the dog smelled like a skunk триграммы выглядят так: #the dog, the dog smelled, dog smelled like, smelled like a, like a skunk и a skunk#. Для последовательностей символов good morning триграммами являются goo, ood, od, dm, mo, mor и т.д. Некоторые строки полезно предварительной очистить от пробелов. Пунктуация также обычно удаляется препроцессором.

Рассмотрим сначала классические алгоритмы n-грамм. Уже более 30 лет [1] ***N-gram*** индексация используется в области информационного поиска. Словарная ***N-gram*** индексация основана на следующем свойстве: если слово ***u*** получается из слова ***w*** в результате не более чем ***k*** элементарных операций редактирования (за исключением перестановок символов), то при любом представлении ***u*** в виде конкатенации из ***k + 1-й*** строки, одна из строк такого представления будет точной подстрокой ***w***.

Это свойство можно усилить, заметив, что среди подстрок представления существует такая, что разность между ее позицией в строках ***w*** и ***u*** не больше ***k***. Таким образом, задача поиска сводится к задаче выборки всех слов, содержащих заданную подстроку.

АЛГОРИТМ СРАВНЕНИЯ ПОДСТРОК

Функция нечеткого сравнения использует в качестве аргументов две строки и параметр сравнения – максимальную длину сравниваемых подстрок. Подстроки содержат буквы кириллического алфавита и пробел. Результатом работы функции является число, лежащее в пределах от нуля до единицы, где ноль соответствует полному несовпадению двух строк, а единица – полной

их идентичности. Сравнение строк происходит по следующей схеме.

Функция сравнения составляет все возможные комбинации подстрок с длиной вплоть до указанной и подсчитывает их совпадения. Количество совпадений, разделенное на число вариантов, объявляется коэффициентом схожести строк для фиксированного ***N*** и выдается в качестве результата работы функции, далее берется среднее значение для всех коэффициентом. Формула релевантности будет выглядеть следующем образом:

$$R = \frac{\sum_{i=1}^{N} r(i)}{N}; \quad (1)$$

$$r(i) = \frac{Match(Str1, Str2, i) + Match(Str2, Str1, i)}{Count(Str1, i) + Count(Str2, i)}, \quad (2)$$

где

Count(Str,i) = (len(Str) - i + 1);

len(S) – длина строки ***S***;

Match(S_1, S_2, i) – сумма совпадений всех подстрок длиной ***i*** из ***S1*** в строке ***S2***.

Пусть, например, в качестве аргументов заданы две строки «Привет» и «Превед» и некоторая максимальная длина подстрок, скажем, четыре.

Таблица 1

ПОДСЧЕТ РЕЛЕВАНТНОСТИ СТРОКИ «ПРИВЕТ» И «ПРЕВЕД»

Сравниваемая подстрока	Подстроки второй строки	Есть совпадение?	Количество совпадений	Количество вариантов	Коэфф. похожести
Сравниваем подстрокам при *n* = 1					
П	п,р,е,в,е,д	Да	4	6	9/12
Р	п,р,е,в,е,д	Да			
И	п,р,е,в,е,д	Нет			
В	п,р,е,в,е,д	Да			
Е	п,р,е,в,е,д	Да			
Т	п,р,е,в,е,д	Нет			
П	п,р,и,в,е,т	Да	5	6	
Р	п,р,и,в,е,т	Да			
Е	п,р,и,в,е,т	Да			
В	п,р,и,в,е,т	Да			
Е	п,р,и,в,е,т	Да			
Д	п,р,и,в,е,т	Нет			
Сравниваем по подстрокам при *n* = 2					
пр	пр,ре,ев,ве,ед	Да	2	5	4/10
ри	пр,ре,ев,ве,ед	Нет			
ив	пр,ре,ев,ве,ед	Нет			
ве	пр,ре,ев,ве,ед	Да			
ет	пр,ре,ев,ве,ед	Нет			
пр	Пр,ри,ив,ве,ет	Да	2	5	
ре	Пр,ри,ив,ве,ет	Нет			
ев	Пр,ри,ив,ве,ет	Нет			
ве	Пр,ри,ив,ве,ет	Да			
ед	Пр,ри,ив,ве,ет	Нет			
Сравниваем по подстрокам при *n* = 3					
при	Пре,рев,еве,вед	Нет	0	4	0
рив	Пре,рев,еве,вед	Нет			
иве	Пре,рев,еве,вед	Нет			
вет	Пре,рев,еве,вед	Нет			
пре	при,рив,иве,вет	Нет	0	4	
рев	при,рив,иве,вет	Нет			
еве	при,рив,иве,вет	Нет			
вед	при,рив,иве,вет	Нет			
Сравниваем по подстрокам при *n* = 4					
прив	прев,реве,евед	Нет	0	3	0
риве	прев,реве,евед	Нет			
ивет	прев,реве,евед	Нет			
прев	прив,риве,ивет	Нет	0	3	
реве	прив,риве,ивет	Нет			
евед	прив,риве,ивет	Нет			

Приведенная табл. 1 иллюстрирует алгоритм подсчета коэффициента схожести двух строк. Для строк «Привет» и «Превед» и длины максимальной подстроки, равной четырем, были получены значения коэффициента, равные 0,75, при ***n = 1; 0,4*** при ***n = 2; 0*** при ***n = 3 и n = 4.***

Таким образом, общая релевантность двух строк:

R = (0,75 + 0,4 + 0 + 0) / 4 = 0,2875 = 28,75%

Увеличение длины максимальной подстроки приводит к увеличению времени работы функции. С другой стороны, поиск становится более четким.

УСТРАНЕНИЕ ДУБЛИКАТОВ

Общий принцип применения функции для поиска дубликатов следующий:

1. Производится вычисление релевантности.
2. Данный показатель приводится к относительной шкале соответствия в интервале от нуля до единицы (ноль – полное несовпадение, единица – полное совпадение). В дальнейшем эта шкала легко может быть приведена к процентному виду, удобному для восприятия человеком.
3. Экспериментальным путем для тестового массива данных определяется нижний порог автоматической обработки $П_а$, за которым количество ошибок распознавания дубликатов становится неприемлемым. Определяется также нижний порог ручной обработки $П_р$, за которым поиск выдает практически одни ошибки.
4. $П_а$ может быть использован для дальнейшей уточняющей обработки дубликатов в автоматическом режиме, оставляя найденные элементы со значениями соответствия ниже $П_а$, но выше $П_р$ для обработки человеком.

Основными способами внесения и изменения информации в базу данных являются:

- непосредственный ввод пользователями;
- импорт данных из внешних источников.

При ручном вводе данных требуется обеспечить минимальное время отклика системы, поэтому используемый на этом этапе алгоритм должен работать не столько точно, сколько предельно быстро. При этом $П_а$ для данной операции может быть изменен в соответствии с требованиями по скорости поиска. Так как система распознавания не может предоставить стопроцентную точность, пользователь должен также иметь возможность игнорировать подсказку системы и ввести данные. При таких условиях в базу данных неизбежно будет попадать часть некачественной информации, которая должна быть обнаружена в дальнейшем. Таким образом, задачу выявления и устранения дубликатов можно разбить на три этапа.

1. Выявление дубликатов на уровне ввода информации и их отклонение.
2. Выявление дубликатов путем сравнения и анализа уже введенных данных в соответствии с заданным $П_а$.
3. Автоматическое удаление дублирующей информации.
4. Анализ и обработка человеком результатов п. 2, которые не могут быть обработаны автоматически (показатель соответствия ниже $П_а$, но выше $П_р$).

Алгоритм работы блока представлен на рис. 1.

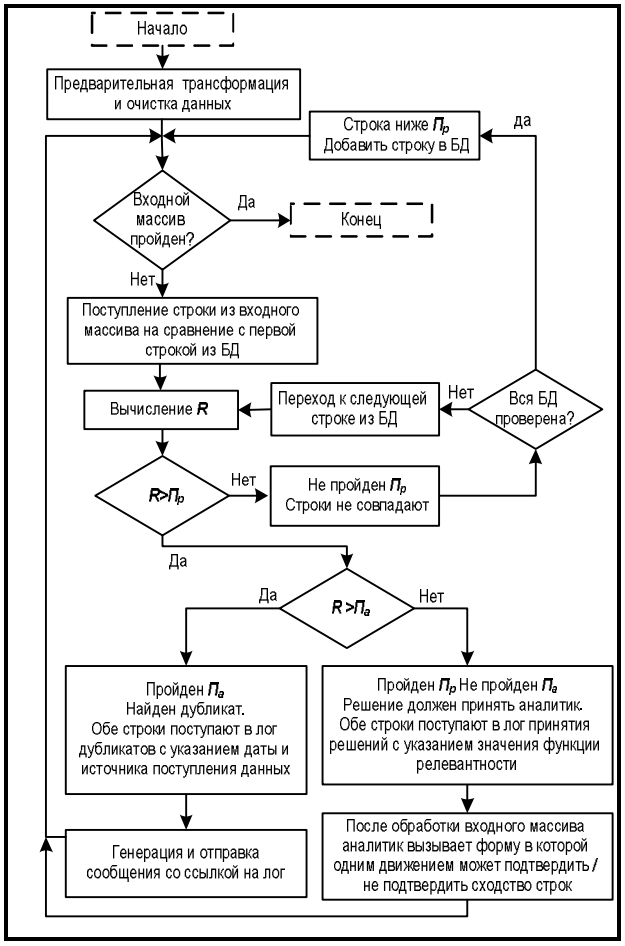

Рис. 1. Алгоритм проверки на дубликаты

РЕАЛИЗАЦИЯ

На этапе 1 все атрибуты поиска склеиваются в одну строку. Например, фамилия, имя и отчество будут обрабатываться как одна строка «ФИО». Данный способ формирования строк для сравнения дает недостаточно точный, но быстрый результат при поиске.

На этапе 2 результаты поиска этапа 1 уточняются, проходя дополнительную проверку путем вычисления релевантности и расстояний уже для отдельных атрибутов поиска с учетом различных весовых коэффициентов, подбираемых экспериментально.

Например, если две строки с реквизитами фирмы совпали с релевантностью 90%, но при сравнении названий фирм получена релевантность 60%, то несмотря на 100% имени владельца и адреса можно отвергнуть результат первого этапа, как ошибочный.

Следует упомянуть о недостатке метода ***N-gram***, который в некоторых случаях может оказаться существенным: большой размер производного множества подстрок (***N-grams***) относительно количества исходных строк. Так, если $N = 3$, то для каждой строки будет сформировано $M = L - N + 1$ словарных элементов, где L – длина строки ≥ N (при $L < N$, $M = 0$ и поиск невозможен).

Например, если имеется база данных на 50 тыс. клиентов, средняя длина ФИО равна 24 символов, то мощность множества подстрок при $n = 4$ будет составлять примерно $50\,000 * (24 - 3) = 1\,050\,000$ элементов. При этом количество уникальных элементов, т.е. словаря, которое напрямую зависит от длины подстроки N, будет существенно ниже (экспериментально была получена разница примерно в два порядка), и, следовательно, будет затруднительно организовать быстрый поиск данных по такому множеству используя стандартные средства реляционной системы управления базами данных (СУБД) в виде индексов. Избирательность (selectivity) такого индекса будет мала (один к ста), что может привести к отказу от его использования оптимизатором запросов.

Решением данной проблемы при большом количестве данных для словаря является увеличение длины N. В качестве нестрогого доказательства применимости данного решения можно рассмотреть функцию максимального количества элементов словаря (уникальных элементов множества) в зависимости от размеров подстрок и алфавита. Эта функция будет давать верхнее значение в предположении, что ***N-gram*** равномерно распределены в строках. Чем меньше равномерность, тем меньше будет количество уникальных элементов.

На практике равномерность при небольших N далека от идеальной, но она увеличивается с ростом N, и в предельном случае, когда N равно длине строки, при отсутствии точных дублей строк, будет получено полностью равномерное распределение.

Пусть N – длина подстроки, m – количество символов в алфавите, включая пробел. Тогда количество возможных размещений с повторениями из m символов алфавита по N вычисляется в соответствии с известной формулой комбинаторики и равно m^N. Это значение и будет являться самой верхней и грубой оценкой (пренебрегаем еще и тем, что далеко не все размещения встречаются и допустимы в языке словаря) количества уникальных элементов словаря. Таким образом, при увеличении N количество уникальных элементов растет, как степенная функция.

Рассмотрим более подробно пример реализации системы распознавания дубликатов для списка фирм с использованием СУБД MS SQL 2008. Фирма характеризуется набором атрибутов, показанных в табл. 2.

Таблица 2

ОПИСАНИЕ АТРИБУТОВ ТАБЛИЦЫ DBO.COMPANY

Атрибут	Тип	Максимальная длина в экспериментальной базе	Длина в среднем, символов
Название фирмы	Varchar (500)	370	15
Почтовый индекс	Varchar (6)	6	6
Населенный пункт (город)	Varchar (50)	20	10
Адрес	Varchar (100)	71	30

Длина в среднем равна сумме длин значений полей нужного атрибута, деленной на количество строк в таблице. Хоть 15 и не такая большая длина, не стоит думать, что задача упростится. Встречаются и вот такие сложные наименования, из-за которых тип данных увеличен до 500:

- садовое товарищество первичная профсоюзная организация путевой машинной станции №31 ст.лянгасово кировского отделения горьковской железной дороги общественной

организации-российский профессиональный союз железнодорожников и транспортных строителей (роспрофжел);
- профсоюзная организация работников государственных учреждений и общественного обслуживания верхнеуслонского района республики татарстан татарстанской республиканской организации общественной общероссийской организации профессиональный союз работников государственных учреждений и общественного обслуживания Российской Федерации;
- объединенная первичная профсоюзная организация работников филиала фгп во ждт россии на гжд межрегиональной общественной организации-объединенной первичной профсоюзной организации работников фгп ведомственная охрана железнодорожного транспорта российской федерации общественной организации-российский профессиональный союз железнодорожников и транспортных строителей.

Для распознавания дубликатов на этапе 1 будет использована строка, составленная из всех перечисленных атрибутов, общая длина которой может достигать 466 символов. Предположим, что в базе данных имеется 20 тыс. записей со средней длиной строки 61 символ. При таких параметрах выбираем длину подстроки ***N = 4.*** Скрипт создания таблиц для хранения списка фирм и словаря будет выглядеть как показано на рис. 2.

```
CREATE TABLE dbo.Company
 ( OID     int          not null identity(1, 1),
   Name    varchar(500) not null,
   ZIPCode varchar(6)  not null,
   City    varchar(50)  not null,
   Address varchar(100)  not null,
CONSTRAINT PK_Company PRIMARY KEY (OID) )
GO
CREATE TABLE CompanyNGrams
(Name    varchar(500) not null,
   NGram      char(4)  not null,
   NGramCount  smallint not null)
GO
```

Рис. 2. Скрипт создания таблица dbo.Company

```
CREATE FUNCTION [dbo].[GetNGram](@str varchar(8000))
  RETURNS @NGramTable TABLE(N int,s varchar(8000))
  AS
  begin
  declare @k int,@l int
  select @k=1,@l=1
  while @k<5
   begin
   set @l=1
   while @l<LEN(@str)+1
                    begin
                    IF @k = len(substring(  replace(@str,' ','0')  ,@l,@k))
                              BEGIN
                              insert @NGramTable(N,s)
                              select @k,substring(@str,@l,@k  )
                              END
                      set @l=@l+1
                      end
  set @k=@k+1
   end
  RETURN
end
      go
DECLARE @idblank varchar(80)
DECLARE #InqTab CURSOR FOR
  select name from dbo.Company
OPEN #InqTab
FETCH NEXT FROM #InqTab INTO @idblank
WHILE @@FETCH_STATUS = 0
BEGIN
          insert  into  dbo.CompanyNGrams (Name, NGram, NGramCount )
          select @idblank ,* from dbo.GetNGram(@idblank)
 FETCH NEXT FROM #InqTab INTO @idblank
END
CLOSE #InqTab
      DEALLOCATE #InqTab
```

Рис. 3. Код функции разбивки на подстроки

Для первоначального заполнения множества подстрок (таблицы CompanyNGrams) и последующего ее поддержания в согласованном со списком фирм состоянии (например, используя триггер), нам потребуется функция разбиения строки на подстроки (грамы).

На рис. 3 представлен код функции, возвращающей множество подстрок (грамов) исходной строки в специально созданную таблицу.

Для поиска дубликата следует сформировать строку для поиска, разбить ее на грамы и вычислить релевантность по отношению к другим строкам, хранящимся в базах данных (БД) и которые были предварительно разделены на подстроки. Релевантность двух строк Str_1 и Str_2 при конкретном значении N вычисляется по формуле (2), для N, заданного диапазоном по формуле (1).

Тогда для поиска похожих строк можно использовать следующую функцию, представленную на рис. 4.

```
CREATE FUNCTION Company_Relevance
(@n int, @k int,
@str1 varchar(8000),
@str2 varchar(8000))
RETURNS  @str_return numeric(10,3)
AS
Begin
Declare @l int
set @str99=0, @l=@n

while @n<@k
 begin

set @str99=@str99+
 ( SELECT
   (
   (SELECT COUNT( t1.s) FROM dbo.CompanyNGrams t1
     inner join (select distinct * from dbo.StrToGrams where N=@n) t2 on t1.s=t2.s
     WHERE t1.n=@n  )
    +
   (SELECT COUNT( t2.s) FROM dbo.StrToGrams t2
     inner join (select distinct * from dbo.CompanyNGrams where N=@n) t1 on
     t1.s=t2.s
      WHERE t2.n=@n )
    )       / cast(len(@str1)+ len(@str2) -2*@n+2 as numeric(10,3) )
)
set @n=@n+1
end
select @str99/(@k +@l)
RETURN
end
```

Рис. 4. Код функции релевантности

Система расширяема. Для добавления нового типа поиска дубликатов, например, в географический справочник или каталог товаров вам потребуется только внести в систему таблицу хранения граммов. Реализация с помощью хранимых процедур и функций заданной сигнатуры позволяет «включить» класс в подсистему нечеткого поиска без дополнительных издержек. Разумеется, за универсальность нужно платить. Например, необходимо писать код проверки семантики классов объектов, добавляемых в таблицу дубликатов. На этапе 1 при грубом сравнении без учета семантики следующие записи будут распознаны как дубликаты:

- «Черноморское отделение Арбатской конторы по заготовке рогов и копыт», Россия, Черноморск, Остап Бендер, Рябиновая улица д. 1.
- «Черноморское отделение Арбатской конторы по заготовке рогов и копыт» Россия, г. Черноморск, Бендер Остап, особняк на Рябиновой улице.

Однако и такие вот записи тоже попадут в число «подозрительных»:

- «Jeff Peters&Andy Tucker Ltd.», Mr. Tucker, UK, London, Baker street 226.
- «J. Peters, A. Tucker and Co», Mr. Peters, UK, London, Baker street 226.

Эти записи попадут в так называемый лог принятия решений, специально созданный для тех записей, значение функции релевантности которых принадлежит промежутку $\Pi_p < R < \Pi_a$.

На этапе 2 при анализе с учетом семантики аналитик скорее всего примет решение о добавлении строки в базу со статусом «Новые данные» вследствие низкой релевантности названий компаний и фамилий. Также на этапе 2 аналитику доступен анализ по отдельным полям, но теперь ***N*** будет браться в диапазоне от единицы до четырех. При этом будет использована общая формула релевантности (1).

Т.е. аналитик получит следующие цифры (табл. 3).

Таблица 3

ПРИМЕР ДАННЫХ, ПЕРЕДАВАЕМЫХ В ФОРМУ ПРИНЯТИЯ РЕШЕНИЙ

Название фирмы	Владелец	Страна	Населенный пункт (город)	Адрес
Общая релевантность 65,7%				
Jeff Peters & Andy Tucker Ltd.	Mr. Tucker	UK	London	Baker street 226
J. Peters, A. Tucker and C°	Mr. Peters	UK	London	Baker street 226
Частная релевантность				
55%	39%	100%	100%	100%

Если строк больше одной, то форма будет выглядеть как табл. 4.

Таблица 4

ОБРАЗЕЦ ФОРМЫ ПРИНЯТИЯ РЕШЕНИЙ

R, %	Название фирмы	Владелец	Страна	Город	Адрес	Дубликат
-	1. Jeff Peters & Andy Tucker Ltd.	Mr. Tucker	UK	London	Baker street 226	-
65,7	J. Peters, A. Tucker and C°	Mr. Peters	UK	London	Baker street 226	Нет
-	57%	39%	100%	100%	100%	
58,7	J. Peters Ltd.	Mr. Peters	UK	London	Baker street 220	Нет
-	51%	39%	100%	100%	72%	
-	2. Прима	Иванов	Россия	Москва	2-я улица Строителей д. 3	-
83,7	Рима	Иванова	Россия	Москва	3-я улица Строителей д. 2	Нет
-	80%	93,6%	100%	100%	93%	
66	Марина	Иванько	Россия	Москва	1-я улица Строителей д. 9	Нет
-	31,5%	53%	100%	100%	91%	
55,8	Прима	Иванов И.Ю.	РФ	Москва	2-я ул. Строителей д. 3	Да
-	100%	64,8%	6%	100%	68%	

УСКОРЕНИЕ РАБОТЫ ФУНКЦИИ РЕЛЕВАНТНОСТИ

Ускорить работу функции можно на этапе поиска одинаковых грам. Максимальная длина хранимой подстроки равна четырем, следовательно, тип данных, хранящий этой строке, будет присвоен varchar (4) длиной 4 байта. Для того чтобы ускорить работу сравнения, требуется придумать способ хранить строки меньше чем в 4 байтах. Идея состоит в следующем:

- вычислить частоты появления символов в БД;
- построить префиксный код на основе частотных характеристик символов в БД;
- перекодировать текстовые данные в цифровые.

Для словарей, содержащих не более 2-3 миллионов слов, частоты легко подсчитать. Взяв таблицы с данными о клиентах и просчитав частоту появления каждого символа, получаем табл. 5.

Таблица 5

ЧАСТОТА ПОЯВЛЕНИЯ БУКВ

Буква	Частота	Буква	Частота	Буква	Частота	Буква	Частота
Пробел	0,175	о	0,090	е, ё	0,072	а	0,062
и	0,062	т	0,053	н	0,053	с	0,045
р	0,040	в	0,038	л	0,035	к	0,028
м	0,026	д	0,025	п	0,023	у	0,021
я	0,018	ы	0,016	з	0,016	ъ, ь	0,014
б	0,014	г	0,013	ч	0,012	й	0,010
х	0,009	ж	0,007	ю	0,006	ш	0,006
ц	0,004	щ	0,002	э	0,003	ф	0,002

На основе частотной информации может быть сформирован код с помощью простого «жадного» алгоритма [2]. Данный способ хорошо подходит для так называемого дискового поиска, когда страницы индекса выбираются непосредственно с диска, потому что в процессе поиска считывается относительно небольшое число списков, занимающих несколько последовательных дисковых страниц.

Алгоритм построения префиксного кода довольно прост (рис. 5), но в рамках поставленной задачи будет неудобно хранить двоичное представление слова, как показано в табл. 6, так как храниться они будут в MS SQL, а он в свою очередь хранит данные в полях фиксированной длины. Это значит, что полю, в котором будет храниться двоичный код буквы, должен будет присвоен тип данных binary [3]. При этом нужно заранее указать длину этого поля, в нашем случае это девять – максимальная длина кода букв Ц, Щ, Э, Ф. Следовательно, если код короче девяти, то слева он добивается нулями до нужной длины.

Таблица 6

ХРАНЕНИЕ БИНАРНЫХ СТРОК

Буква	Код
К	000000100
С	000000100
…	…
Ц	110000000
Щ	110000001
…	…
О	000000011
Й	001100001

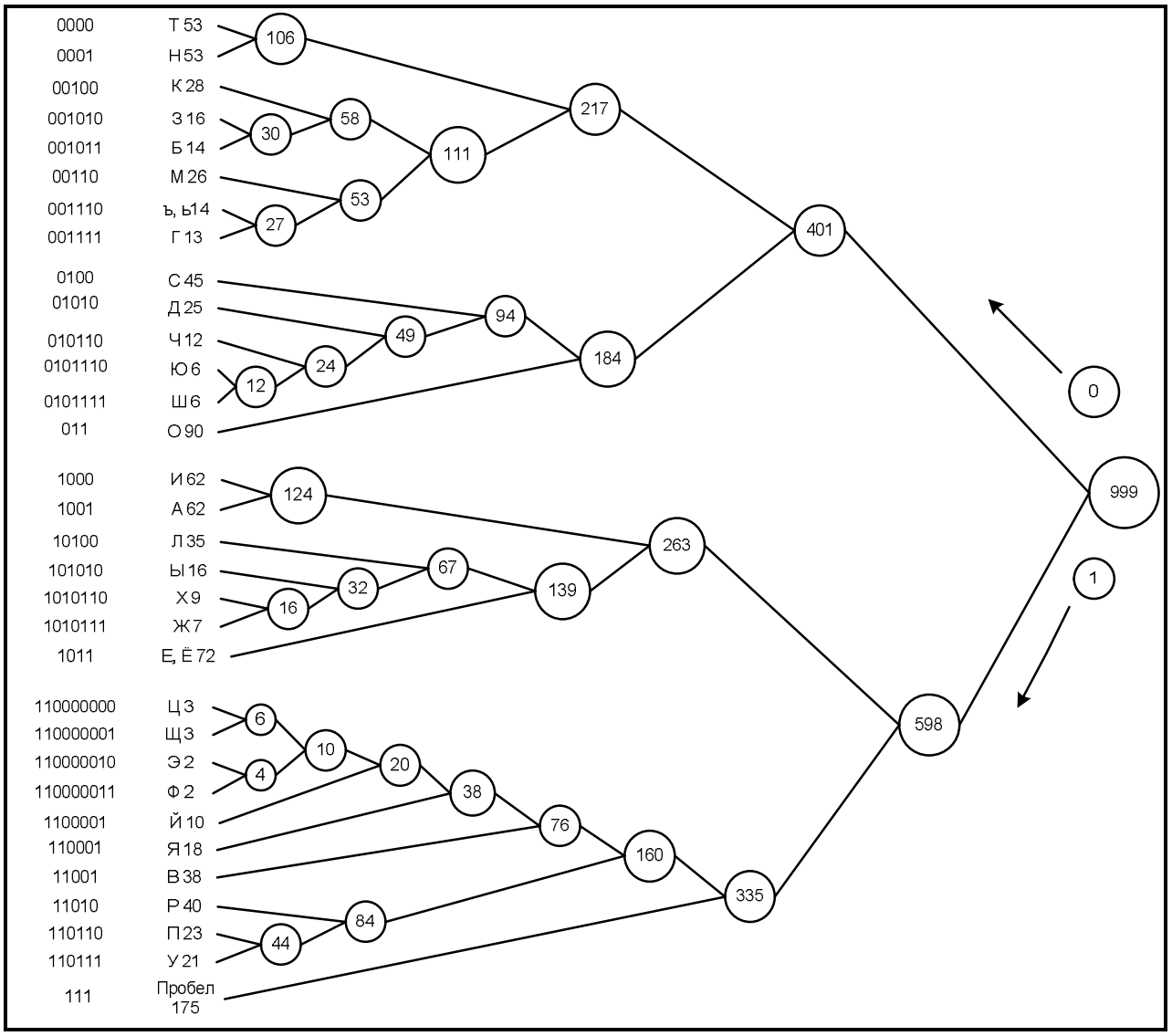

Рис. 5. Построение кода на основе частотной информации

При этом теряется правило префиксности, некоторые коды начинают сливаться (буквы К и С), следовательно одному коду не однозначно соответствует слово. В принципе это решается путем добавления ко всем кодам единицы слева. Но и это еще не все: при сравнении строк в MS SQL 2008 нужно будет писать функцию, которая в цикле будет побитово сравнивать строки, а это уже нерационально.

Выход из ситуации предлагается следующий – создать не двоичный префиксный код, а десятичный. При этом оптимизатор MS SQL данные десятичные коды все равно будет хранить в двоичном представлении, но теперь поиск будет осуществляться не самописной функцией, а с помощью годами отлаженных программистами Microsoft алгоритмов поиска, которые в сочетании с индексом должны дать существенный прирост производительности. В табл. 7 показан пример построения префиксного кода.

Теперь самый длинный код составляет 6 бит (34_{dec} = $100\ 010_{bin}$, т.е. шесть двоичных порядков), и в самом худшем случае занимает: $ффффф = 34343434_{dec} = 24$ бита против 32 при хранении в формате varchar.

Таблица 7

ПРЕФИКСНЫЙ ДЕСЯТИЧНЫЙ КОД

Буква	Частота	Код	Буква	Частота	Код
Пробел	0,175	0	я	0,018	19
о	0,09	4	ы	0,016	20
е, ё	0,072	5	з	0,016	21
и	0,062	6	б	0,014	22
а	0,062	7	ъ, ь	0,014	23
т	0,053	8	г	0,013	24
н	0,053	9	ч	0,012	25
с	0,045	10	й	0,01	26
р	0,04	11	х	0,009	27
в	0,038	12	ж	0,007	28
л	0,035	13	ю	0,006	29
к	0,028	14	ш	0,006	30
м	0,026	15	ц	0,004	31
д	0,025	16	э	0,003	32
п	0,023	17	щ	0,002	33
у	0,021	18	ф	0,002	34

Таблица 8

РАСЧЕТ СРЕДНЕЙ ДЛИНЫ КОДА

Буква	Частота	$Код_{dec}$	$Код_{bit}$	Длина битовой строки	Частота длина кода	Мат. ожидание
Пробел	0,175	0	0	1	0,175	0,175
о	0,09	4	100	3	0,27	0,858
е, е	0,072	5	101	3	0,216	
и	0,062	6	110	3	0,186	
а	0,062	7	111	3	0,186	
т	0,053	8	1 000	4	0,212	1,272
н	0,053	9	1 001	4	0,212	
с	0,045	10	1 010	4	0,18	
р	0,04	11	1 011	4	0,16	
в	0,038	12	1 100	4	0,152	
л	0,035	13	1 101	4	0,14	
к	0,028	14	1 110	4	0,112	
м	0,026	15	1 111	4	0,104	
д	0,025	16	10 000	5	0,125	1,07
п	0,023	17	10 001	5	0,115	
у	0,021	18	10 010	5	0,105	
я	0,018	19	10 011	5	0,09	
ы	0,016	20	10 100	5	0,08	
з	0,016	21	10 101	5	0,08	
б	0,014	22	10 110	5	0,07	
ъ, ь	0,014	23	10 111	5	0,07	
г	0,013	24	11 000	5	0,065	
ч	0,012	25	11 001	5	0,06	
й	0,01	26	11 010	5	0,05	
х	0,009	27	11 011	5	0,045	
ж	0,007	28	11 100	5	0,035	
ю	0,006	29	11 101	5	0,03	
ш	0,006	30	11 110	5	0,03	
ц	0,004	31	11 111	5	0,02	
э	0,003	32	100 000	6	0,018	0,042
щ	0,002	33	100 001	6	0,012	
ф	0,002	34	100 010	6	0,012	
Итого						3,417

Проведем тестирование работы блока с применением кода и без (табл. 9).

Таблица 9

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Сек

Операция	Без кодирования (тыс. строк)				С кодированием (тыс. строк)			
	10	20	30	50	10	20	30	50
Перекодирование ***N-Gram*** исходного массива	0	0	0	0	6	10	14	23
Перекодирование ***N-Gram*** входного массива	0	0	0	0	6,5	10,5	14,7	24
Работа функции релевантности при ***N*** = 4	39,5	78	116	207	22	47,5	69	117
Работа функции релевантности при ***N = (1, …, 4)***	77,5	154	242	378	49	98,8	162	263
Итого	117	232	358	585	83,5	167	259	427

Эксперименты проводились на компьютере с двумя процессорами Dual Core AMD Opteron(tm) 2216 HE 2,4 Ггц и 4 Гб оперативной памяти. Несмотря на возможные погрешности реализации: тестирование проводилось на рабочем сервере параллельно с выполнением текущих фоновых задач, которые могли повлиять на скорость тестирования.

Вместе с тем полученные результаты позволяют дать приближенную оценку вычислительной сложности предложенного алгоритма. С увеличением объема БД время выполнения всегда будет увеличиваться линейно – ***O(n)***. Простота и быстродействие данного алгоритма является его конкурентным преимуществом в случае обработки больших массивов данных.

Заключение

Полученные в ходе тестирования алгоритма результаты позволяют сделать следующие выводы:

- Предложенный алгоритм построения функции релевантности на основании алгоритма ***N-Gram*** целесообразно использовать в условиях больших объемов данных.
- Характерной особенностью всех полученных результатов является относительная стабильность в точности.
- Устойчивость в точности дает возможность создания различных модификаций алгоритма с целью увеличения логической полноты.

Для дальнейших исследований представляет интерес проведение тестирования, с включением в алфавит спецсимволов. В рамках данного тестирования необходимо провести сравнительный анализ эффективности нахождения дубликатов по одной и по нескольким базам данных. Кроме того, становится актуальной задача создания тестовой коллекции, содержащей наборы «идеальных» пар дубликатов по отдельным полям, что позволит проводить исследования эффективности алгоритмов по различным критериям.

Литература

1. Левитин А.В. Алгоритмы: введение в разработку и анализ [Текст] / А.В. Левитин. – М. : Вильямс, 2006. – С. 392-398.
2. Майкрософт [Электронный ресурс] : официальный сайт. – Режим доступа: http://www.microsoft.com/rus/
3. Тарасов С. Как избавиться от дубликатов в базе данных [Электронный ресурс] / Сергей Тарасов. URL: http://www.osp.ru/pcworld/2007/11/4656322/.
4. Shang H., Merret T.H.. Tries for approximate string matching. In IEEE transactions on knowledge and data engineering. 1996. Vol. 8(4). P. 540-547.
5. Ukkonen E. Algorithms for approximate string matching // In information and control. 1985. Vol. (64). P. 100-118.
6. Ukkonen E. Approximate string matching with q-Grams and maximal matches // In theoretical computer science. 1992. Vol. 92(1). P. 191-211.
7. Ukkonen E. Finding approximate patterns in strings, O(k * n) time // In journal of algorithms. 1985. Vol. 6. P. 132-137.
8. Web site of the Computer science department of Maryland University. Research on N-Grams in Information Retrieval. http://www.cs.umbc.edu/ngram//

Ключевые слова

Нечеткий поиск; ***N-gram***; релевантность строк; поиск данных; приблизительное сравнения строк; порог идентификации; база данных; код Хаффмана; сжатие данных без потерь; код переменной длины; теория информации; префиксный код.

Карахтанов Дмитрий Сергеевич

РЕЦЕНЗИЯ

Проблема организации поиска в больших массивах данных является одной из важнейших проблем развития современных информационных систем. В статье Карахтанова Д.С. раскрыты особенности нечеткого поиска данных с применением комбинации базовых алгоритмов, адаптированных для специализированных задач ООО «Кредитное бюро Русский Стандарт».

Актуальность выбранной темы обуславливается необходимостью разработки и внедрения специального математического и программного обеспечения в интересах кредитных организаций с целью устранения искажений информации, обусловленных ошибками операторского ввода данных.

Предложенные в статье решения по использованию алгоритмов нечеткого поиска могут использоваться в качестве типовых при построении автоматизированных информационных систем бюро кредитных историй.

Статья рекомендована к публикации.

Антоненко В.Д., к.э.н., профессор Университета Российской академии образования

11.2. USING OF FUZZY SEARCH ALGORITHM IN PROCESSING OF DATA FOR CREDIT INSTITUTIONS

D.S. Karakhtanov, Post-graduate Student

University of the Russian Academy of Education

Author described the algorithm of duplicate records elimination under condition of several data sources and errors of operator input. The algorithm based on relevance function using N-gramme method, and also a way of acceleration of its performance thought construction frequency prefics code usage. The received solutions can be applied as template for building information systems in financial organizations.

Literature

1. Web site of the Computer science department of Maryland University. Research on N-Grams in Information Retrieval. http://www.cs.umbc.edu/ngram/
2. A.V. Levitin. Introduction to The Design and Analysis of Algorithms. «Williams», 2006. p. 392-398.
3. http://msdn.microsoft.com/en-us/library/aa258271%28SQL.80%29.aspx
4. http://www.osp.ru/pcworld/2007/11/4656322/
5. H. Shang, T.H. Merret. Tries for Approximate String Matching. In IEEE Transactions on Knowledge and Data Engineering, volume 8(4), p. 540-547, 1996.
6. E. Ukkonen. Algorithms for approximate string matching. In Information and Control, volume (64), pages 100-118, 1985.
7. E. Ukkonen. Finding approximate patterns in strings, O (k * n) time. In Journal of Algorithms volume 6, pages 132-137, 1985.
8. E. Ukkonen. Approximate String Matching with q- Grams and maximal matches. In Theoretical Computer Science, volume 92(1), pages 191-211, 1992.

Keyword

Fuzzy search; N-gram; string relevance; data search; approximate string matching; identification limit; database; Huffman coding; lossless data compression; variable-length code; Information theory; prefix code.