17 Epp V.V. Kachestvo i testirovanie programmnogo obespecheniya. Uchebnoe posobie.// Penza: Izddatel'stvo: PGU, 2016.-129 s.

18 Gvozdeva V. A., Lavrent'eva I. YU. Osnovy postroeniya avtomatizirovannyh informacionnyh sistem; sinteg - Moskva, 2016. - 320 s.

19 Vodyaho A.I., Vygovskii L.S., Arhitekturnye resheniya informacionnyh sistem: uchebnik. – Lan', 2017.-356 s.

20 Sovetov B.YA. Arhitektura informacionnyh sistem; Akademiya (Academia) - M., 2017. - 934 s.

РЕЗЮМЕ

В настоящее время многие компании, такие как Netflix, Apple, Instagram и Pinterest, меняют свои программы и системы на микросервисы, поскольку эта архитектура позволяет компаниям масштабировать вычислительные ресурсы в соответствии с ними, используя их.

Для полноценной работы современных веб-приложений должны быть такие работы, как возможность предоставления интерфейса программного обеспечения, обработка большого количества запросов, масштабирование, обеспечение, высокая скорость доступа к данным, обеспечение высокой надежности, стабильная бесперебойная работа. Различные проблемы указанных гигантов решались путем перехода на микросервисы. Эти компании являются хорошим примером использования гибких микросервисов для внедрения инноваций и оптимизации издержек. Тысячи других брендов по всему миру также используют новые возможности микросервисной архитектуры.

В данной статье изучено понятие масштабируемой веб-архитектуры, ее виды, принципы проектирования и инфраструктурные требования. Подробно обсуждался вопрос перехода информационных систем от монолита к микросервисной архитектуре, анализировались сложность и специфика. Проведен сравнительный анализ различных типов архитектуры - монолитных, SOA и типов микросервисов, их слабых и сильных сторон, состояния проекта и команды с учетом основных критериев проектирования системы. Статья направлена прежде всего на обобщение результатов, проведенных по сравнительной оценке двух архитектур. Во-вторых, он фокусируется на определении производительности и взаимосвязей между различными переменными, такими как процессор, потребление памяти, производительность сети, дисковые операции и время разработки, интеграция, которая переходит от монолитной структуры к микросервисной.

УДК 004.021
МРНТИ 20.23.19

Жаксыбаев Д. О., магистр педагогических наук, **основной автор,** https://orcid.org/0000-0001-6355-5431
НАО «Западно-Казахстанский аграрно-технический университет имени Жангир хана», г. Уральск, ул. Жангир хана 51, 090009, Казахстан, darhan.03.92@mail.ru

Zhaxybayev D. O., Master of Pedagogical Sciences, **the main author,** https://orcid.org/0000-0001-6355-5431
NJSC «West Kazakhstan Agrarian and Technical University named after Zhangir khan», Uralsk, st. Zhangir khan 51, 090009, Kazakhstan, darhan.03.92@mail.ru

АЛГОРИТМЫ ПОИСКА ИНФОРМАЦИОННЫХ РЕСУРСОВ ДЛЯ ПОВЫШЕНИЯ ЭФФЕКТИВНОСТИ РАБОТЫ С ДОКУМЕНТАМИ
ALGORITHMS FOR FINDING INFORMATION RESOURCES TO IMPROVE THE EFFICIENCY OF PAPERWORK MANAGEMENT

Аннотация

В данной статье описывается основа хранения, восстановления и обновления информации с особым акцентом на алгоритм поиска и структуру данных, необходимую для

максимальной производительности программы. Эффективность программы гарантируется, в частности, в тех случаях, когда в процессе поиска используется естественный или символический язык. Максимальная эффективность программы достигается за счет конкретного алгоритма поиска и структуры данных, и эта эффективность вычисляется простым термином «среднее количество запросов», необходимых для поиска ~ p объекта, вместо того чтобы тестировать более высокий коэффициент вызова и точность. Для поиска объекта, даже если он впервые найден, требуется хотя бы один поиск. При использовании хэш- адреса ключа или ключевого слова, в сочетании со структурированной таблицей списка и широким списком пробелов, среднее число поисков для поиска определенного объекта составляет 1,25, независимо от размера файла. Это противоположно поиску в бинарном формате 15,6 в файле с 50 точками и поиску в таблице с 5,8 буквами безотносительно размера файла. Программное обеспечение может использовать одну и ту же систему для хранения и обновления информации, а также обеспечить полную эффективность. Это устраняет все проблемы, связанные с неэффективностью при создании файла и обновлении файла. Все собранные данные и последующие заключения в этой работе были основаны на существующих концепциях и теориях различных авторов по данной тематике.

ANNOTATION

This study describes the basics of storing, restoring, and updating information, with particular emphasis on the search algorithm and data structure required for maximum software performance. The effectiveness of the software is guaranteed, in particular, in cases where natural or symbolic language is used in the search process. The maximum efficiency of the software is achieved through a specific search algorithm and data structure, and this efficiency is calculated in the simple term “average number of queries” required to find ~ p of an object, as opposed to testing for higher call rate and accuracy. Finding an object, even if it is first found, requires at least one search. When a key or keyword hash is used, combined with a structured list table and a wide list of spaces, the average number of searches to find a specific object is 1.25, regardless of file size. This is the opposite of a 15.6 binary search on a 50-dot file and a 5.8 letter table search regardless of file size. The software can use the same system to store and update information, and be fully efficient. This eliminates all inefficiency issues when creating a file and updating a file. All the data collected and subsequent conclusions in this study were based on existing concepts and theories of various authors on this subject.

Ключевые слова: *линейный поиск, поисковый каталог, таблица писем, хранение информации.*

Key words: *linear search, search catalog, table of letters, information storage.*

Введение. Интерес к вопросу о поиске информации в интернете не ослабевает на протяжении всего времени существования сети. Поиск может вестись как пользователем-любителем, так и профессионалом. При проведении поиска информации, удовлетворяющего информационным потребностям пользователя, необходимо знать, от чего зависит успешный поиск, и какие проблемы возникают при работе с информацией. Некоторые поисковые системы вообще не имеют алгоритма как такового. Их работа сводится в основном к очистке текста сайта от программного кода и выстраивания слов, встречающихся на сайте, по их частоте. Такой подход имеет под собой определенные основания: чем сложнее алгоритм работы поисковой системы, тем, с одной стороны, больше вероятность получения наиболее точных и полных результатов, но, с другой стороны, больше вероятность ошибок в работе самого алгоритма. Иными словами, усложняя алгоритм работы поисковой системы, можно как достичь более полных и точных результатов, так и, наоборот, получить менее точные и полные результаты.

Работа по поиску информации в любой поисковой системе примерно одинакова и сводится к работе нескольких агентов. Суть работы поисковых агентов заключается: в отслеживании существующих ссылок; анализе страниц на наличие ссылок на другие страницы; поиске информации по новым ссылкам, полученным при анализе текстов; просмотре новых страниц, которые регистрирует хозяин нового ресурса. Если рассматривать поиск информации

на основе работы метапоисковых систем, то он будет более простым: они работают с поисковыми системами и у них остаются только агенты, занятые опросом последних, и, возможно, проверкой существования выдаваемых ссылок.

В данной статье описывается основа хранения, восстановления и обновления информации, в которой уделяется пристальное внимание алгоритму поиска и структуре данных, необходимых для оптимальной работы программы. Эффективность программы гарантируется, в частности, в тех случаях, когда в процессе поиска используется естественный или символический язык. Этот метод является важнейшей основой для создания эффективной информационной системы. Он может быть реализован для обработки текстовых и документальных данных, для извлечения числовых данных, а также для обработки больших файлов, таких как словари, каталоги и личные записи, и графической информации. В настоящее время в пакетном режиме реализовано восемь координат хранить, извлечь, добавить, удалить [1]. Дальнейшим усовершенствованием стало бы использование консоли телетайпа, терминала электронно-лучевые трубки (ЭЛТ) и плоттера для немедленного реагирования в условиях совместного использования времени.

Максимальная эффективность программного обеспечения достигается с помощью конкретного алгоритма поиска и структуры базы данных, которая вычисляется простым термином «среднее число поисков», необходимых для просмотра коэффициента поиска и коэффициента точности на более высоком этапе. Требуется хотя бы один поиск, даже если он был впервые найден для идентификации объекта [2]. Однако, используя хэш-адрес ключа или ключевого слова вместе с таблицей, структурированной по косвенным цепочкам, и широким списком пробелов, средний номер поиска, необходимый для поиска определенного объекта, составляет 1,25, независимо от качества рассматриваемого файла. Это сравнивается с бинарным поиском 15,6 в файле 50,000 элементов и поиском 5,8 в таблице с буквами, независимо от размера файла [3]. Лучше всего то, что поскольку программное обеспечение может использовать одну и ту же технологию хранения и обновления, оно также одинаково легко применять полную производительность. Это устраняет все проблемы, которые могут привести к неэффективности, связанной с созданием и обновлением файла.

Сравнительный анализ различных алгоритмов поиска

Шесть из них проверяются и сравниваются с точки зрения эффективности поиска, чтобы получить общее представление о различных алгоритмах поиска. Линейный поиск списка или файла без порядка - самый простой, но не эффективный, так как средний номер поиска в файле N-ввода составляет N/2. Например, при N = 50,000 средний номер поиска для данной записи составляет 25,000.[4] Вероятность наличия конкретной записи в файле считается равной единице. Среднее число поиска при линейном поиске равно:

$$S = \frac{N+1}{2} \quad (1)$$

$$S = \frac{N}{2} \quad (2)$$

если N большое число.

Линейный поиск должен выполняться в последовательной области хранения, и это происходит, когда соответствующая область хранения очень широка. Недостатка можно избежать с помощью последнего машинного слова (или некоторых его битов для индексирования положения следующего сегмента используемого поля запаса, таким образом, создавая единую цепочку поиска). Этот вариант называется единой цепной формой линейного поиска, он отличается от линейного поиска универсальности хранения, но в остальном производительность одинаковая [5].

Поисковый каталог, его также часто называют цепным поиском. Лучший результат можно получить при поддержке директории, содержащей адреса каждого Bth-входа упорядоченного файла, так как среднее количество запросов значительно уменьшается. В приведенном выше примере для получения наилучшего результата выбирается коэффициент блокировки B = 220, ответ - 223,6 поиска, при этом [6]:

$$S = \frac{N+B^2}{2B} \quad (3)$$

$$S = \frac{N}{2B} + \frac{B}{2} \quad (4)$$

Следующий алгоритм - бинарный поиск, который предлагает более удовлетворительный результат. Поиск начинается с середины файла и попадает в середину оставшейся половины, если поиск не удается. Аналогично принципу Диктаты половина материала может быть следующей попыткой. Следующий этап - сравнение, этот процесс повторяется до тех пор, пока не будет найдено совпадение. Среднее число поисков в примере оценивается в 15,6 поисков по следующей формуле [7]: (5) если H - большое число.

$$S = \frac{N+1}{N}\log_2(N+1) - 1 \quad (5)$$

Хиббардский метод двойной цепочки и метод распределенных ключей Суссенгута совместимы с бинарным поиском в эффективности поиска и гораздо лучше модифицированы процессом структурированной адресной цепочки.[8] Соответствующие примеры вычислений: Хиббард: S = 1.4 $\log_2$N = 21.9, и Суссенгут: S = 1.24 I$\log_2$N = 19.4.

Метод буквенной таблицы появился как предложение в 1961 году для поиска словарей в системе машинного перевода, эта заманчивая техника, как показали Ягнен и Якобсен, не получила широкой огласки для своих осуществимых программ в общей информационной системе [9]. Пояснения могут быть немедленным ответом после второго уровня на различные буквенные таблицы, что говорит о неэффективности его хранения и отсутствии последовательной эффективности поиска и обновления.

Предположим, включено только 33 буквы в русском языке, потенциально есть 33^1 таблицы первого уровня, 33^2 таблицы второго уровня, 33^3 таблицы третьего уровня и т.д. На практике количество таблиц с буквами после второго этапа будет значительно сокращено за счет реального ограничения комбинации букв [10]. Однако для оценки потребности в ее загрузке никакие исследования такого рода не могут опровергнуть потерю ее загрузки. Средний номер поиска или предполагаемое время поиска при таком подходе не может быть измерено в виде файла или функции размера словаря. Это просто среднее количество букв или символов определенного языка плюс символ пробела или какой-то другой предел. Для русского благоприятны 5,8 поиска (S = 4,8 + 1) без учета размера файла [11]. Результат его обновления соответствует производительности поиска и может быть вычислен менее чем в два раза по сравнению со средним поисковым номером.

Для вышеуказанного качества таблицы с буквами должны быть расположены в алфавитном порядке на каждом уровне, и каждая буква должна быть переведена в числовое значение, такое как А = 1, Б = 2, в = 3, ... Я = 33 и лимит пробела = 0 или 34 быстрым просмотром таблиц. Затем эти преобразованные значения будут использоваться для каждого подмножества букв алфавита в качестве адреса прямого доступа на каждом уровне буквенной таблицы. Это устраняет необходимость проверки двоичных файлов в подмножестве «братья». [12]

Метод открытого обращения был изобретен Петерсоном для хранения данных со случайным доступом в 1957 году. Этот подход также известен как линейное тестирование. Он предполагает, что ключ или ключевое слово входа является числовым значением в диапазоне размера thq-таблицы, которое по умолчанию равно 2^M для любого целочисленного значения M. Размер таблицы должен быть достаточно широким для обработки всех записей файла [13]. Как и в других методах, такой подход часто предполагает возможность того, что запись содержится в файле. В обоих этих предположениях открытый подход к адресации решит ситуацию, когда на определенном слоте в таблице отображены более одного ключа, и приведет к очень привлекательному среднему числу поисков в большинстве ситуаций. Алгоритм лучше всего представлен в предложениях Морриса [14].

Первый подход к созданию последовательных определенных адресов, предложенный в литературе, буквально, в следующем контексте, заключался в том, чтобы приблизить сталкивающиеся друг с другом записи к их номинально распределенному местоположению. При столкновении следует смотреть вперед от номинального местоположения (вычисленный начальный адрес) до тех пор, пока не будет найдена либо желаемая запись, либо, по возможности, не появится пустое место для поиска по кругу, мимо конца таблицы [15]. Когда

пустое место найдено, это место становится домом для нового входа. Результаты среднего количества поисков из девяти различных раундов по сравнению с результатами (Таблица 1), полученными из рецептуры Morris или рецептуры Salton (L - коэффициент загрузки или процент таблицы полноты при проведении поиска [16]): Петерсон выполнил несколько вычислений открытой адресации, создав случайные числа и сохранив их в таблице на 500 входов - формула Морриса:

$$S = \frac{1-\frac{L}{2}}{1+L^2} \quad (6)$$

Формула Солтона:

$$S = \frac{L}{2(1-L)} + 1 \quad (7)$$

Таблица 1 – Вычисления открытой адресации Питерсона

Коэффициент загрузки	Питерсон	Моррис/Салтон
0,1	1,053	1,056
0,2	1,137	1,125
0,3	1,23	1,214
0,4	1,366	1,333
0,5	1,541	1,5
0,6	1,823	1,75
0,7	2,26	2,167
0,8	3,223	3,0
0,9	5,526	5,5
1,0	16,914	У

Очевидно также, что если таблица почти завершена, то средние поисковые номера удивительно малы. Например, если коэффициент загрузки 0,9 или ниже, то среднее число поиска составит 1,965. Этого можно добиться, включив дополнительные 10% от шкалы таблицы [17]. Качество его хранения менее желательно в этой ситуации. Эффективность поиска и обновления данных превосходна, однако, благодаря крайне низкому среднему показателю поиска.

Метод непрямой адресации базируется в основном на хэш-адресе и делает те же два предположения, что и открытый метод адресации, то для этого метода предлагается более описательное имя в виде хэш-адресованного косвенного цепного поиска (HAICS) [18]. Таблицы индекса рассеяния Косвенная цепочка (сдвиг в цепных структурах), закрытая адресация (прямая и косвенная цепочка) и виртуальные таблицы рассеяния - это другие имена, встречающиеся в литературе (соответствующие дополнительным хэшированным битам). Система HAICS использует стандартизированную таблицу с четырьмя полями, добавленную не адресуемую таблицу переполнения или отдельную таблицу переполнения, а также свободную область хранения, называемую списком пробелов. Перед использованием области переполнения и свободной области хранения она намерена полностью использовать все помещения для таблицы. При таком подходе область адресной таблицы считается полной, т.е. после последнего адреса будет учитываться первый адрес таблицы [19]. В случае переполнения расширенная область таблицы создает не адресную область переполнения. Это планируется так, что эффективность хранения повышается и в большинстве случаев дополнительная область переполнения не нужна, а значит, может быть удалена или вставлена в начале при необходимости.

В таблице цепочек HAICS есть четыре области: ключевое слово, индекс, ссылка и указатель. Поле ключевого слова обычно является машинным словом для символов, обозначающих запись. Поле индекса должно иметь достаточное количество битов для указания наибольшего среднего адреса в списке высших баллов, чтобы эта таблица могла индексировать запись переменной длины, содержащуюся в доступном списке пробелов. Поле связи используется для обозначения связи со следующей таблицей, в которой можно найти

информацию о записях с одинаковым значением хэширования. Поле указателя обычно содержит адрес самой первой записи с таким же хэшированным значением. И поле связи, и поле указателя должны иметь длину поля, соответствующую наибольшему относительному адресу адресной области таблицы, которая должна храниться в битах, т.е. размер таблицы, к которой можно применить привязку.

Записи вводятся сначала по их хэшированным адресам, а затем по следующим или окруженным) пустым адресам при столкновении. Создаются их указатели и ссылки для правильной цепочки. Когда запись проверяется, первым шагом является проверка указателя входа на адрес хэширования, а затем переход к указанному адресу и начало поиска с этой первоначальной записи. Если запись не найдена, то до тех пор, пока она не будет найдена, будет проверяться запись, обозначенная текущим соединением. Последний случай показывает, что этот поиск не найден. Если запись найдена по ключевому слову ID, то адрес в поле i~dex приведет фактическое хранилище записей к списку пробелов. Индексное поле и доступный список пробелов нужны только в том случае, если записи имеют разные номера, чтобы сохранить пространство для хранения [20]. Для записей фиксированной длины доступное пространство больше не требуется, и индексная камера в таблице может быть преобразована в камеру для записей фиксированной длины

Основным преимуществом хэш-адреса является то, что для изменения записей в файле не требуется никакой сортировки или сортировки. Удаление записи в методе HAICS заключается в том, чтобы следовать алгоритму до тех пор, пока запись не будет найдена, а затем перехватить- до предыдущей записи в цепочке. Все предварительно занятые хранилища этой записи освобождаются для дальнейшего использования. Чтобы добавить вход, используйте тот же алгоритм, чтобы получить последний вход в определенной цепи, а затем создать соединение со следующим вакуумом в таблице и сохранить входную информацию. Сама вставленная запись находится в некоторых неограниченных данных и индексируется в таблице цепочки в списке доступных пробелов.

Этот метод был впервые предложен Джонсоном в 1961 году и просто рассчитывается как среднее число поисков [21]:

$$S = 1 + \frac{L}{2} \quad (8)$$

Эти формулы также более интересны, если коэффициент загрузки L больше единицы, то есть количество записей превышает размер выделенной таблицы, а данные о переполнении сохраняются в области переполнения, в то время как сами записи помещаются обратно в список свободного места. Стоимость переполнения линейно возрастает всего на 0,9 поиска за 100-процентный рост переполнения. Эта мера практически снимает страх перед переполнением, что иногда приводит к почти неуправляемым трудностям и очень высоким затратам. Перед тем, как таблица будет завершена, как обычно, среднее число поисков методом косвенной цепочки составит 1,25 с максимальным числом 1,5, когда таблица будет почти завершена. Автор считает, что некоторую некомпетентность в хранении и сложность программирования следует признать безболезненно как с приведенными выше цифрами, так и с приведенной выше эффективностью обновления и приростом переполнения [22] (Таблица 2, 3).

Понятие хэш-функции для эффективного поиска. Из предыдущего сравнения многих алгоритмов поиска можно понять, что поворотным моментом для отличной эффективности поиска и обновления является хэш-адресация, которая, по сути, является базовой процедурой, использующей определенную хэш-функцию на ключе поиска или ключевом слове. Поскольку одно и то же ключевое слово всегда вставляется в одно и то же табличное хэш-значение адреса, уникальность характеристики является критерием для выбора или создания ключевого слова для записи [23]. А для того, чтобы избежать непреднамеренного столкновения с хэшированными адресами, следует выбрать эффективную хэш-функцию, которая обеспечит сбалансированное распределение значений хэш-функции в диапазоне размеров таблицы. Принимая во внимание программирование и эффективность вычислений, а также производительность и эффективность хранения, ключевое слово одного размера машинного слова, как правило, является более подходящим, например, 48-битное ключевое слово.

Таблица 2 – Среднее число поисков Джонсона

	Johnson
0,1	1,05
0,2	1,1
0,3	1,15
0,4	1,2
0,5	1,25
0,6	1,3
0,7	1,35
0,8	1,4
0,9	1,45
1,0	1,5
1,5	1,75
2,0	2,0
3,0	2,5
4,0	3,0

Таблица 3– Сравнение методов поиск

Метод поиска	Среднее количество поисков	Образец S(N=50,000)	Эффективность поиска	Эффективность обновления	Эффективность хранения
Линейный поиск	$\frac{N}{2}$	25.000	Низкая	Хорошая	Отличная
Поисковый каталог	$\frac{N}{2B}+\frac{B}{2}$	223,6 (B=220)	Средняя	Средняя	Отличная
Бинарный поиск	$\log_2 N$	15.6	Хорошая	Низкая	
Метод буквенной таблицы (Ламб и Якобсен)	Среднее количество букв в слове плюс один пробел	5,8 (Английский)	Отличная	Хорошая	Низкая
Метод открытой адресации (Петерсон)	$\frac{L}{2(t-L)}+1$	1,965 (L≤0.9)	Отличная	Отличная	Хорошая
Метод непрямой адресации (Джонсон)	$1+\frac{L}{2}$	1,25 (L≤1.0)	Отличная	Отличная	Средняя

Ключевое слово почти всегда доступно для взлома в компьютерных файлах, таких как определения словарей, тезаурусы, индексы ключевых слов и товарные каталоги. Если длина ключевого слова превышает допустимое количество символов, можно использовать простую конкатенацию слов в правом конце или другие системы сжатия слов, чтобы свести размер слова к соответствующему количеству символов [24]. Если желательны титульные указатели, каталоги, предметные указатели и каталоги, бизнес-телефонные книги, научно-технические словари, лексиконы, словари языка и словосочетаний и другие описательные сведения о нескольких словах, то первая особенность каждого нетривиального слова может быть использована для формирования ключевого слова в оригинальном словарном ряду. Например, такое ключевое слово, как SADSIRS, может быть очень длинным заголовком данной работы. Некоторые известные системы информации называются SIR (Semantic Info~retrieval system Raphael), SADSAM (Sentence Auditor and Semantic Analyzer Computer Lindsay), BIRS

(Vinsonhalers simple indexing and retrieval system) и CGC (Klein and Simmons' Computational Grammar Coder) [25].

Альтернативой, но с потенциальным изменением сингулярности соответствующего хэш- значения, является выполнение арифметических и логических манипуляций с бинарной схемой из нескольких слов. Когда многословие хранится в последовательных компьютерных словах, двоичное представление каждого компьютерного слова будет рассматриваться как отдельная константа. Любое из этих машинных слов должно затем выполняться арифметической операцией, такой как «добавить», «умножить» и «удалить», или логической операцией (например, «И», «ИЛИ»), чтобы они могли распадаться на одно ключевое слово компьютерного термина. Полученное в результате ключевое слово этой формы принуждения не является гуманным, а выполняет свою функцию.

Различные функции для случайно генерируемых запросов также могут служить в качестве хэш-функции, если может быть сформировано вероятное соотношение «один к одному» между ключевыми словами и результирующим случайным числом. Для этого также необходимо, чтобы в диапазоне размеров таблицы находились только числа [26]. Этот метод также не обеспечивает сбалансированного распределения адресов в таблицах и, таким образом, влияет на производительность поиска и обновления. Вышеописанная арифметика или логическая обработка многословных объектов также может быть использована в качестве алгоритма хэширования. Маурер предлагает рассматривать двоичное представление ключевого слова как целое число и разделить его на размер таблицы. Остальная часть этого отрезка также находится в диапазоне размеров таблицы и используется в качестве значения хэша. Как обнаружил Маурер, недостатком такого подхода является то, что он часто не генерирует равномерно распределенные подсказки.

Общая структура данных HAICS очень ориентирована на список, но упакована в массивы, чтобы сделать процесс индексирования и поиска более эффективным. Исследовательская программа для адаптации к другим компьютерам была написана в CDC 3600 Fortran (версия Fortran IV) [27]. Следующие темы также относятся к терминологии Fortran и оформлению списка для большей ясности. Четырехпольную таблицу можно удобно настроить в виде четырехмерного массива или четырехмерного массива за несколько потраченных впустую битов машинного слова для хранения индекса в высший список оценок, ссылки на адрес следующей таблицы или указателя на начало цепочечного под-списка. Преимущество заключается в меньшем количестве слов при упаковке и распаковке.

По отношению к первому пункту, т.е. к адресам хэшей, местоположения внутри каждого пункта таблицы с четырьмя полями могут рассматриваться как основной перечень таблицы. Подключаемые записи одного и того же хэш-адреса известны как подписка. Так как идентифицируется относительное местоположение каждого элемента таблицы, включая его хэш-адрес, то список адресов для каждого элемента не требуется. Более того, поскольку на каждую запись можно ответить хэшем со средним числом поисковых номеров до 1,25, а большинство цепочек не намного короче одного или двух участников, обратное звено в цепочке не требуется [28].

Выводы. Подход HAICS является фундаментальной парадигмой для улучшения общей эффективности информационной системы. Он может быть запрограммирован на различные языки, от базового языка системы или ассемблера конкретного компьютерного семейства, до высокопроцедурных языков, таких как Fortran и Algol, приемлемых для большинства компьютеров. При среднем значении 1,25 поиска на запись, этот подход определенно не делает естественную обработку языка хуже, чем n - нерациональные вычисления. Он подходит для работы с текстом и поиска документов; для поиска числовых данных; и для работы с большими файлами, такими как словари, каталоги и личные записи, а также с графическим материалом. Как уже упоминалось ранее, на CDC 3600 реализовано восемь команд, которые работают в пакетном режиме в тестовой программе, закодированной на языке Fortran и машинном языке COMPASS.

Дополнительной разработкой является использование консоли телетайпа, ЭЛТ- терминала и плоттера для получения немедленных ответов в среде совместного использования времени. Это делается для того, чтобы под рукой у вас была самая подробная энциклопедия или персонализированный указатель ссылок. В частности, операция поиска по

словарю как основная операция информационной системы больше не представляет собой длительную и болезненную процедуру и не является барьером для естественной обработки языка. Лингвистический анализ может быть предоставлен полной свободой для того, чтобы отсылать туда и обратно каждую запись в словаре и грамматике, а собранная информация может храниться и извлекаться таким же образом на любом этапе анализа.

Поиск документов может углубляться в контент-анализ и включать синонимичный словарь для лучшего преобразования и подгонки функций дескриптора вопроса. Статья предназначена для поддержки аргументации Шоффнера в предоставлении обзора современных методов поиска и подробного объяснения процесса HAICS, который является возможным применением для большинства информационных систем. Исследование Шоффнера сосредоточено на анализе современных методов поиска и подробном описании.

СПИСОК ЛИТЕРАТУРЫ

1 Jiang X., Lin Z., He T., Ma X., Ma S., Li S. Optimal Path Finding with Beetle Antennae Search Algorithm by Using Ant Colony Optimization Initialization and Different Searching Strategies // IEEE Access. - 2020. - No. 8. - P. 15459-15471.

2 Jiang Y., Hao K., Cai X., Ding Y. An improved reinforcement-immune algorithm for agricultural resource allocation optimization // Journal of Computational Science. - 2018. - No. - P. 320-128.

3 Wei J., Zeng X.-F. Optimal computing resource allocation algorithm in cloud computing based on hybrid differential parallel scheduling // Cluster Computing. - 2019. - No. 22. - P. 75777583.

4 Nesterenko A.V., Netesin I.E. Cybersecurity graph model of information resources// Journal of Automation and Information Sciences. - 2020. - Vol. 52, No. 8. - P. 14-31.

5 Dahiru G., Oladokun O., Grand B., Mutshewa A. Exploring the Application of Information and Communication Technologies in the Acquisition of Information Resources in Three Academic Libraries in North-West Nigeria: Preliminary Findings // Collection Management. - 2019. - Vol. 45, No. 3. - P. 252-272.

6 Saastamoinen M., Jarvelin K. Relationships between work task types, complexity and dwell time of information resources // Journal of Information Science. - 2018. - Vol. 44, No. 2, P. 265-284.

7 Матренин П.В. Использование принципов объектно-ориентированного программирования при реализации оптимизационных алгоритмов// Объектные системы. - 2015. - № 10. - С. 25-28.

8 Захаров В.П. Информационно-поисковые системы. - Санкт-Петербург: СПбГУ, 2005.

9 Iqbal M., Siti Astuti E., Trialih R., Wilopoa Arifin, Z., Alief Aprilian, Y. The influences of information technology resources on Knowledge Management Capabilities: Organizational culture as mediator variable // Human Systems Management. - 2020. - Vol. 39, No. 2. - P. 129-139.

10 Cai S., Zhang X. Pure MaxSAT and Its Applications to Combinatorial Optimization via Linear Local Search // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). - 2020. - No. 12333 LNCS. - P. 90-106.

11 Опанасенко М. Описание алгоритмов сортировки и сравнение их производительности [Электронный ресурс]. - Режим доступа: https://habr.com/ru/post/335920/ (дата обращения: 11.11.2020)

12 Wang S., Yu Z., Yu X. Real-time online action detection and segmentation using improved efficient linear search // International Journal of Computing Science and Mathematics. - 2019. - Vol. 10, No. 2. - P. 129-139.

13 El-Hadidy M.A.A. Generalised linear search plan for a D-dimensional random walk target. International Journal of Mathematics in Operational Research. - 2019. - Vol. 15, No. 2. - P. 211-241.

14 El-Hadidy, M.A.A. On the existence of a finite linear search plan with random distances and velocities for a one-dimensional Brownian target // International Journal of Operational Research. - 2020. - Vol. 37, No. 2. - P. 245-258.

15 Salton G., Fox E., Wu H. Extended Boolean information retrieval. - New York: Cornell University, 1982.

16 Oviedo H., Lara H., Dalmau O. A non-monotone linear search algorithm with mixed direction on Stiefel manifold // Optimization Methods and Software. - 2019. - Vol. 34, No. 2.

- P. 437457.

17 Sharma M., Wagh S.J., Sar A. Identifying cuts by linear search method // International Journal of Engineering and Advanced Technology. - 2019. - Vol. 8, No. 4C. - P. 5-11.

18 Czyzowicz J., Kranakis E., Krizanc D., Narayanan L., Opatrny J., Shende S. Linear search with terrain-dependent speeds // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). - 2017. - No. 10236 LNCS. - P. 430-441.

19 Бондарев В.М. Основы программирования. - Ростов-на-Дону: Феникс, 1997.

20 Левин В.И. Задача Беллмана-Джонсона для конвейерных систем с переменным порядком работ // Вестник Тамбовского государственного технического университета. - 2003. - Т. 9, №3. - С. 457-463.

21 Захаров Д.Е. Разработка интеллектуальной нейросетевой поисковой системы «Нейропоиск» // Тезисы Молодежной научно-технической конференции «Наукоемкие технологии и интеллектуальные системы». - 2002. - №7. - С. 32-38.

22 Миллер Б., Рэнум Д. Хэширование [Электронный ресурс]. - Режим доступа: http://aliev.me/runestone/SortSearch/Hashing.html (дата обращения: 11.11.2020)

23 Озкарахан Э. Машины баз данных и управление базами данных. - Москва: Мир, 1989.

24 Некрестьянов И.С Тематико-ориентированные методы информационного поиска. - Санкт-Петербург: Санкт-Петербургский государственный университет, 2000.

25 Аграновский А.В., Арутюнян Р.Э., Хади Р.А. Современные аспекты проблемы поиска в текстовых базах данных // Телекоммуникации. - 2003. - №3. - С. 25-30.

26 Maurer W.D. An improved hash code for scatter storage // Communications of the ACM. - 1983. - Vol. 26, No. 1. - P. 36-38.

27 Аграновский А.В., Арутюнян Р.Э. Способы индексации и поиска документов в интернет- порталах // Труды X Всероссийской научно-методической конференции «Телематика- 2003». - 2003. - №1. - С. 204-206.

28 Толстобров А.А., Хромых В.Г. Полнотекстовый поиск в электронных библиотеках // Четвертая Всероссийская научная конференция «Электронные библиотеки: перспективные методы и технологии, электронные коллекции». - Дубна: Объединенный институт ядерных исследований, 2002.

REFERENCES

1 Jiang, X., Lin, Z., He, T., Ma, X., Ma, S., Li, S. 2020. Optimal Path Finding with Beetle Antennae Search Algorithm by Using Ant Colony Optimization Initialization and Different Searching Strategies. IEEE Access, 8, 15459-15471.

2 Jiang, Y., Hao, K., Cai, X., Ding, Y. 2018. An improved reinforcement-immune algorithm for agricultural resource allocation optimization. Journal of Computational Science, 27, 320-128.

3 Wei, J., Zeng, X.-F. 2019. Optimal computing resource allocation algorithm in cloud computing based on hybrid differential parallel scheduling. Cluster Computing, 22, 7577-7583.

4 Nesterenko, A.V., Netesin, I.E. 2020. Cybersecurity graph model of information resources. Journal of Automation and Information Sciences, 52(8), 14-31.

5 Dahiru, G., Oladokun, O., Grand, B., Mutshewa, A. 2019. Exploring the Application of Information and Communication Technologies in the Acquisition of Information Resources in Three Academic Libraries in North-West Nigeria: Preliminary Findings. Collection Management, 45(3), 252-272.

6 Saastamoinen, M., Jarvelin, K. 2018. Relationships between work task types, complexity and dwell time of information resources. Journal of Information Science, 44(2), 265-284.

7 Matrenin, P.V. 2015. Using the principles of object-oriented programming in the implementation of optimization algorithms. Object Systems, 10, 25-28.

8 Zakharov, V.P. 2005. Information retrieval systems. Saint Petersburg: SPbSU.

9 Iqbal, M., Siti Astuti, E., Trialih, R., Wilopoa, Arifin, Z., Alief Aprilian, Y. 2020. The influences of information technology resources on Knowledge Management Capabilities: Organizational culture as mediator variable. Human Systems Management, 39(2), 129-139.

10 Cai, S., Zhang, X. 2020. Pure MaxSAT and Its Applications to Combinatorial Optimization via Linear Local Search. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12333 LNCS, 90-106.

11 Opanasenko, M. 2017. Description of sorting algorithms and comparison of their performance. https://habr.com/ru/post/335920/

12 Wang, S., Yu, Z., Yu, X. 2019. Real-time online action detection and segmentation using improved efficient linear search. International Journal of Computing Science and Mathematics, 10(2), 129-139.

13 El-Hadidy, M.A.A. 2019. Generalised linear search plan for a D-dimensional random walk target. International Journal of Mathematics in Operational Research, 15(2), 211-241.

14 El-Hadidy, M.A.A. 2020. On the existence of a finite linear search plan with random distances and velocities for a one-dimensional Brownian target. International Journal of Operational Research, 37(2), 245-258.

15 Salton, G., Fox, E., Wu, H. 1982. Extended Boolean information retrieval. New York: Cornell University.

16 Oviedo, H., Lara, H., Dalmau, O. 2019. A non-monotone linear search algorithm with mixed direction on Stiefel manifold. Optimization Methods and Software, 34(2), 437-457.

17 Sharma, M., Wagh, S.J., Sar, A. 2019. Identifying cuts by linear search method. International Journal of Engineering and Advanced Technology, 8(4C), 5-11.

18 Czyzowicz, J., Kranakis, E., Krizanc, D., Narayanan, L., Opatrny, J., Shende, S. 2017. Linear search with terrain-dependent speeds. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10236 LNCS, 430-441.

19 Bondarev, V.M. 1997. Basics of programming. Rostov-on-Don: Phoenix.

20 Levin, V.I. 2003. Bellman-Johnson problem for conveyor systems with variable work order. Bulletin of the Tambov State Technical University, 9(3), 457-463.

21 Zakharov, D.E. 2002. Development of an intelligent neural network search engine “Neuropoisk”. Abstracts of the Youth Scientific and Technical Conference “Science-Intensive Technologies and Intelligent Systems”, 7, 32-38.

22 Miller, B., Renum, D. 2015. Hashing. http: //aliev.me/runestone/SortSearch/Hashing.html

23 Ozkarakhan, E. 1989. Database machines and database management. Moscow: Mir.

24 Nekrestyanov, I.S. 2000. Topic-oriented information retrieval methods. Saint Petersburg: Saint Petersburg State University.

25 Agranovsky, A.V., Harutyunyan R.E., Khadi R.A. 2003. Modern aspects of the problem of search in text databases. Telecommunications, 3, 25-30.

26 Maurer, W.D. 1983. An improved hash code for scatter storage. Communications of the ACM, 26(1), 36-38.

27 Agranovskiy, A.V., Harutyunyan, R.E. 2003. Methods for indexing and searching documents in Internet portals. Proceedings of the X All-Russian Scientific and Methodological Conference “Telematics-2003”, 1, 204-206.

28 Tolstobrov, A.A., Khromykh, V.G. 2002. Full-text search in electronic libraries. In: Fourth AllRussian Scientific Conference “Digital Libraries: Advanced Methods and Technologies, Digital Collections”. Dubna: Joint Institute for Nuclear Research.

ТҮЙІН

Бұл мақалада бағдарламаның максималды өнімділігі үшін қажетті іздеу алгоритмі мен деректер құрылымына ерекше назар аудара отырып, ақпаратты сақтау, қалпына келтіру және жаңарту негізі сипатталған. Бағдарламаның тиімділігіне, атап айтқанда, іздеу процесінде табиғи немесе символдық тіл қолданылған жағдайларда кепілдік беріледі. Бағдарламаның максималды тиімділігіне белгілі бір іздеу алгоритмі мен деректер құрылымы арқылы қол жеткізіледі және бұл тиімділік қоңырау жылдамдығы мен дәлдігін тексерудің орнына ~ p нысанын іздеуге қажетті Қарапайым «орташа сұраулар» терминімен есептеледі. Нысанды табу үшін, егер ол бірінші рет табылса да, кем дегенде бір іздеу қажет. Кілттің немесе кілт сөздің хэш мекенжайын құрылымдық тізім кестесімен және кең бос орындармен бірге қолданған кезде, белгілі бір нысанды іздеуге арналған іздеулердің орташа саны файл өлшеміне қарамастан 1,25

құрайды. Бұл 50 нүктелі файлда 15,6 екілік форматта іздеуге және файл өлшеміне қарамастан 5,8 әріптен тұратын кестеде іздеуге қарама-қарсы. Бағдарламалық жасақтама ақпаратты сақтау және жаңарту үшін бірдей жүйені қолдана алады, сонымен қатар толық тиімділікті қамтамасыз етеді. Бұл файлды құру және файлды жаңарту кезінде тиімсіздікпен байланысты барлық мәселелерді жояды. Осы жұмыста жиналған барлық мәліметтер мен кейінгі тұжырымдар осы тақырып бойынша әртүрлі авторлардың қолданыстағы тұжырымдамалары мен теорияларына негізделген.

УДК 004.657
МРНТИ 20.53.19

Кубегенова А.Д., техника ғылымдарының магистрі, **негізгі автор,** https://orcid.org/0000-0003-0156-7757
«Жәңгір хан атындағы Батыс Қазақстан аграрлық-техникалық университеті» КеАҚ, Орал қ., Жәнгір хан көшесі 51,090009,Қазақстан, aigul-03@mail.ru
Кубегенов Е.С., оқытушы, https://orcid.org/0000-0001-7424-2641
«М. Өтемісов атындағы Батыс Қазақстан университеті»КеАҚ, Орал қ., Н.Назарбаева данғылы., 162, 090000, Қазақстан, erlando78@mail.ru

Kubegenova A.D., Master of Technical Sciences, **the main author,** https://orcid.org/0000-0003-0156-7757
NJSC «West Kazakhstan Agrarian and Technical University named after Zhangir khan», Uralsk, st. Zhangir khan 51, 090009, Kazakhstan, aigul-03@mail.ru
Kubegenov Y.S., teacher,https://orcid.org/0000-0001-7424-2641
NJSC "M. Utemisov West Kazakhstan University", Uralsk, N. Nazarbayev Avenue, 162, 090000, Kazakhstan, erlando78@mail.ru

БІЛІМ БЕРУ ПРОЦЕСІНДЕ БІЛІМ АЛУ МІНДЕТТЕРІН ШЕШУ ҮШІН ДЕРЕКТЕРДІҢ ИНТЕЛЛЕКТУАЛДЫ ТАЛДАУ МЕХАНИЗМДЕРІН ЗЕРТТЕУ
THE STUDY OF DATA MINING MECHANISMS FOR SOLVING EDUCATIONAL PROBLEMS IN THE EDUCATIONAL PROCESS

Аннотация

Цифрландыру қазіргі заманғы білім беру жүйесін дамытудың аса маңызды бағыттарының бірі болып табылады. Білім беру процесіне цифрлық технологияларды енгізу білім беру деректерін талдаумен айналысатын ақпараттың үлкен көлемінің пайда болуына әкеледі.

Мақалада білім беру процесін басқару туралы шешім қабылдауға қолдау көрсету үшін білім беру процесінде туындайтын деректерді іздеу әдістерін қолданудың өзектілігі негізделген.

Білім беру саласындағы деректерді зияткерлік талдаудың негізгі міндеттері мен әдістері баяндалған, сондай-ақ қарастырылып отырған бағытқа арналған негізгі жұмыстарға шолу берілген. Күрделі алгоритмдерді қолдана отырып, деректерді талдауға мүмкіндік беретін бағдарламалық қосымша ұсынылған. Қосымшаның сипаттамасы және оны қолданудың мүмкін әдістері келтірілген. Бұл әдістер білім беру процесін басқарудың барлық деңгейлерінде шешім қабылдауда қолдау жүйелерінде кеңінен қолдануға болады. Білім беру процесінің деректерін интеллектуалды талдаудың негізгі әдістері мен сипатамалары берілген.Сипаттамада түрлі салалардағы деректерді талдауға қатысты data mining, білім беру домендеріндегі тәсілдер қолданылатыны анықталды.

Білім беру доменінің деректерін интеллектуалды талдауды қолданудың негізгі бағыттары, сондай-ақ шешімі табылған немесе зерттеу жүргізіліп жатқан міндеттер қарастырылды. Сипатталған қолдану салаларын, зерттеу арқылы алынған нәтижені, деректерді іздеу арқылы шешуге болатын маңызды міндеттері көрсетілген.