NOVATEUR PUBLICATIONS
JournalNX- A Multidisciplinary Peer Reviewed Journal
ISSN No: 2581 - 4230
VOLUME 6, ISSUE 6, June -2020

RESEARCH OF SEARCH ALGORITHM METHODS

MENGATOVA XURSHIDA TOSHMUXAMATOVNA
Teacher, Termez branch of Tashkent State Technical University named after Islam Karimov,
xurshidamengatova7288@gmail.com

BOZOROV ASQAR KHAITMUROTOVICH
Teacher, Termez State University,
asqarjon1990@umail.uz

ABSTRACT:

This article discusses the methods of search algorithms used in modern search engines, which are currently being developed and have a wide range of capabilities. The performance mechanisms of existing modern search engines, binary, interpolation and sequential search methods were analyzed. The results of the analysis can be used to develop general functional requirements for a database of electronic documents when creating an advanced search system.

KEYWORDS: Sequential Search Method, Interpolation Search Method, Binary Search Method, Array, Index, PostgreSQL.

INTRODUCTION:

The growing volume of electronic documents causes problems with finding the necessary information quickly and accurately. As a solution to this problem, many large companies working in the field of software development, presented their products. Work is also underway to further improve them [1].

In this regard, the study of search algorithms used in advanced search engines is one of the relevant studies.

This article discusses the search algorithms used in modern search engines, which are currently being developed and have a wide range of capabilities.

Search is one of the main operations in data processing, and its task is to determine whether to find the data corresponding to this argument in the database by the given argument (key).

MAIN PART:

All search algorithms are divided into static and dynamic algorithms. When performing a static search, the running time of the algorithm does not change. However, during a dynamic search, the size or content of the data to be searched may change. In addition, search algorithms are divided into algorithms that work with disordered data, and algorithms that work with pre-sorted data. To get the sorted data, first all the elements are sorted in a specific order to increase the search speed. An example of this method is a dictionary search in which all information is sorted alphabetically [2].

Based on this, we will look at the search methods listed in Figure 1 below.

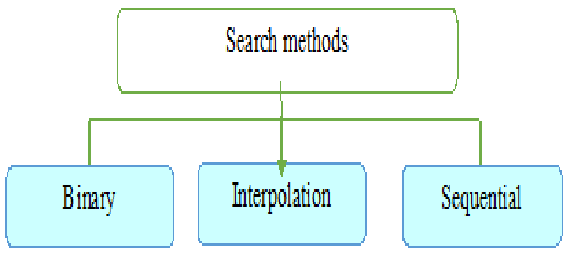


Figure 1. Search methods.

Objects are made up of many elements. These elements are called keys and are the index letter $\mathbf{k}_i$, which indicates the number of the element. The function of the search system is to search for the key $\mathbf{k_i}$ from the key arrays $\{k_1, k_2, k_3, k_4, \ldots, k_n\}$ and display the result. The search result in two cases: the key you are looking for is available or not available [3]. Consider the above methods separately with examples.

SEQUENTIAL SEARCH METHOD:

When the elements of the array are stored as a list, we can say that there is a linear or serial connection between them. Each element is stored in a specific position, that is, in the index. If several elements are sorted, we can go through them sequentially. This process leads to a series of searches. The array specified in the sequential search method is not required to be sorted. From the first item to the last search item, each item is compared with a search key until a key to search is found. The sequential search method consists of the following steps: where **A[1...n]** is an array, **k** is the key to search.

Step 1: we get the first element of the array **i = 1**;

Step 2: if **i<n**, go to step 3, otherwise go to step 5.

Step 3: if **A[i]=k** is equal, the search was completed successfully, if **A[i]≠ k**, go to step 4.

Step 4: we get the next element of the array **i + 1** and go to step 2.

Step 5: the search failed, that is, the key k was not found.

The sequential search method can search through ordered and unordered arrays. In an unordered array, elements can be located in any serial number, and the key to search for each element is compared n times (n – array length) [4]. In this case, the result is in two different situations, that is, the key that is searched in the array is present or absent.

Example: an array - {23,70,36,9,58,25,22,50,15,45} is specified, and a key search of k = 25 is required. Initially, the array is the first, i.e. we take the element at position 0 and compare it with the key 23 ≠ 25. Thus, the obtained value is not equal, so we get the next element 70 ≠ 25, this element is also not equal, we get the next element again until the search key is found and the process continues. 6-қадамда 7-позициядаги элемент қидирилаётган калитга тенг 25 = 25 ва қидирув муваффақиятли якунланди. If the key you are looking for is k = 26, that is, it does not exist in the array, the search is performed to the last element, and the search ends unsuccessfully.

If the array is sorted and the elements of the array are in ascending order, the search is performed by comparing each element from the first element with the key that you are looking for until the key is found, if the key you are looking for is in the array. If the array does not contain a key to search, this further reduces the algorithm. Let A [1 ... n] be an array, k = 26 is the key to search.

Массив - {10,14,15,19,22,25,31,46,50,55}

We search from the first element to a value of 31, because the values after the number 31 are greater than the key we are looking for. If the desired key k was not found during the search, the array is not compared with the last element, and the search stops in the middle of the array. Thus, the application of this method is effective in small arrays.

INTERPOLATION SEARCH METHOD:

This search method is designed to search by sorted data. The essence of this algorithm is to separate the required element for comparing keys. In the interpolation search algorithm, the index of the element that is sought for part of the array is selected by specifying it.

As mentioned earlier, the interpolation search engine considers an element that is not located in the middle of the array and is determined by the following formula at each step **d** [5].

$$d = \left[\frac{(j-i)(K-Ki)}{(Kj-Ki)}\right] \qquad (1)$$

- Here **i** is the sequence number of the first element of the array under consideration;
- **j** – serial number of the last element of the considered array;
- **K** – is the key to search;
- **K_i , K_j** - key values located at positions **i** and **j**.
- **[ ] –** get the whole part of the result.

If the source array is sorted as arithmetic progression or reduction, multiplication, this search method is successful, otherwise the algorithm may not perform the search. We will consider the principle of the method in the following example.

Array - {4, 5, 10, 23, 24, 30, 47, 50, 59, 60, 64, 65, 77, 90, 95, 98, 102}.

Let this key search K = 90 be required.

In the first step, we define the element in step **d**. Here i=1, j=17, K_i=4 and K_j=102.

$$d = \left[\frac{(17-1)(90-4)}{(102-4)}\right]=[14.04]=14$$

Therefore, the desired key is compared with the number in the serial number K_{15}, 95> 90 [3]. Thus, the value we found is larger than the key we are looking for, so we shorten the search field and define **d** again. i=1, j=15, K_i=4 and K_j=95.

Array - {4, 5, 10, 23, 24, 30, 47, 50, 59, 60, 64, 65, 77, 90, 95}

$$d = \left[\frac{(15-1)(90-4)}{(95-4)}\right]=[13.23]=13$$

K_{14} is compared with the number in the sequence number whose key is being searched, 90 = 90. This means that the key you are looking for is in the 14th digit.

When developing this method, a number of problems may arise. For example, let an array be given in the following order:

Array - {2,4,5,9,10,12,18,20,280,1000}

If the desired key is K = 10, finding d by formula (1) is as follows.

$$d = \left[\frac{(10-1)(10-2)}{(1000-2)}\right]=[0.07]=0$$

First, we compare the element with the number 10> 2 so that it is larger, and continue the search on the right side of the array. You can see that the denominator of formula (1) is a sufficiently large number, and if K is a small value, and the array consists of thousands of elements, the value d will appear in increments of one hundred or one hundred thousand steps. In this case, we need to bring d closer to a larger value. If the array is in an unordered state, using this method is not considered effective.

BINARY SEARCH METHOD:

The most efficient search method from sorted files in an array view is the binary search method. The algorithm was proposed by J.V. Mauchly, and the essence of the algorithm is as follows. The search argument is compared with the value located in the center of the array, if they are equal, the search is completed successfully. If the values are not equal, continue searching in the right or left side of the file. When comparing each argument, the length of the array is halved [6]. The algorithm searches by sorting arrays and comparing aggregates. The algorithm consists of the following sequences. Here K is the required argument.

Step 1: The upper **u** and lower **l** of the search boundary are set: **l = 1, u = n**.

Step 2: If u <1, the search is not performed; otherwise, the value i corresponding to the average value of the array under consideration is determined, then

$$\mathbf{i} = (\mathbf{n}/2) + 1 \qquad (2)$$

if **n** is even, then

$$\mathbf{i} = (\mathbf{n}/2). \qquad (3)$$

Step 3: If **K<K_i**, go to step 4, if **K>K_i**, go to step 5, if **K=K_i**, the algorithm is completed successfully.

Step 4: We change the upper limit of the array **u = i-1** and go to step 2.

Step 5: We change the upper limit of the array $\mathbf{l = i + 1}$ and go to step 2.

To better understand this algorithm, let's look at the following example. The length of the array is n = 16, and the searched argument is k = 26.

Array - [2, 5, 9, 11, 12, 15, 18, **20**, 23, 25, 26, 30, 36, 37, 39, 40]

Determine the value at the center of the array **i = (16/2) = 8**. The received argument is compared with the received key **26 <20**, since the argument is large, the search continues to the right of the array, and we change the lower limit.

Array: 2, 5, 9, 11, 12, 15, 18, 20, [23, 25, 26, **30**, 36, 37, 39, 40]

Here, the key obtained is also compared with the aggregate **26 <30**, since the aggregate is small, it moves to the left of the array, and the upper boundary changes.

Array: 2, 5, 9, 11, 12, 15, 18, 20, [23, **25**, 26], 30, 36, 37, 39, 40

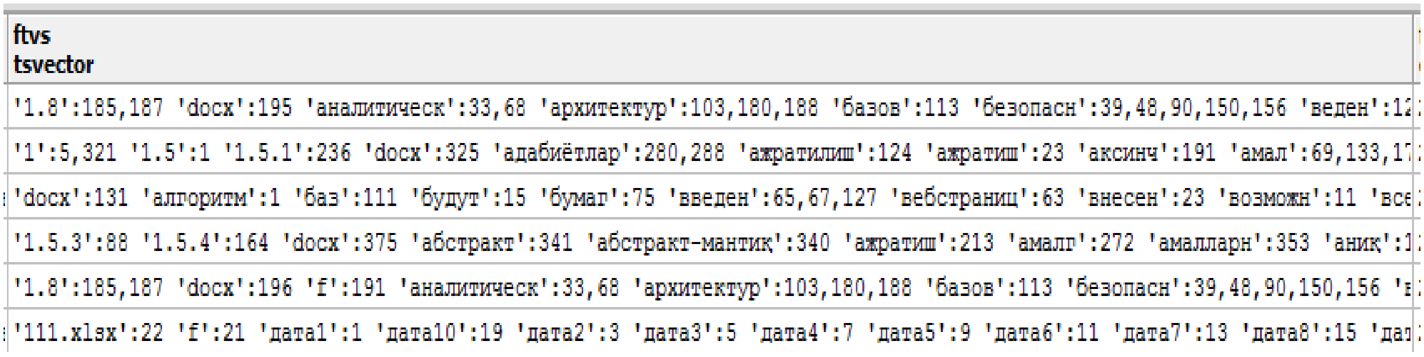
In the next step, the above processes are also repeated, that is, the arguments are compared, 26> 25 the argument is large, so it moves to the right of the array, and the lower border changes.

Array: 2, 5, 9, 11, 12, 15, 18, 20, 23, 25, [**26**], 30, 36, 37, 39, 40

At the last step, the length of the array is 1, and comparison 26 = 26 is performed. The tool found is equal to the key and the result is obtained. If the argument that is searched in the last step is not equal to the key, it is concluded that the argument to be sought does not exist in the array. In our example, the key we are looking for is K = 26 and is in the tenth serial number of the array, we take this value as an answer.

The above analysis methods were considered in the absence of an index and in small arrays. If we use these methods in this case in large arrays, this is not considered effective, since the search time increases and the speed decreases. In such cases, indexes are created to prevent deficiencies.

We will see the application of these methods in PostgreSQL. PostgreSQL uses a number of functions to quickly and accurately search for text files. One of these functions is the to_tsvector () function, which converts the words in the text into an array of lexically significant words according to their position. Figure 2 below shows an array of lexically significant words (in russian and uzbek languages, cyrillic alphabet).

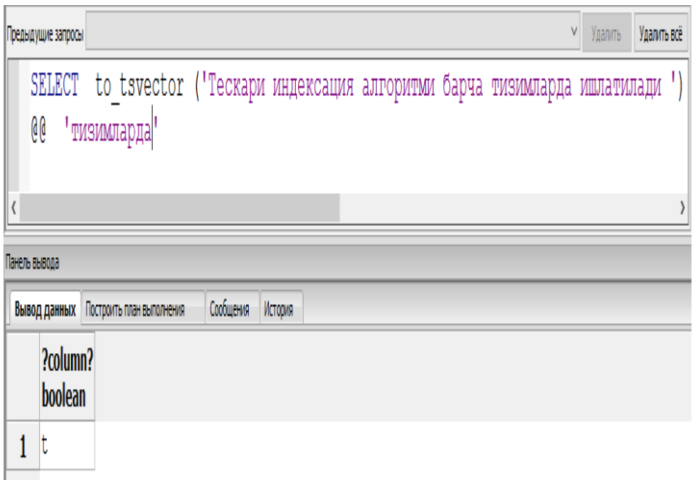


```
ftvs
tsvector
'1.8':185,187 'docx':195 'аналитическ':33,68 'архитектур':103,180,188 'базов':113 'безопасн':39,48,90,150,156 'веден':12
'1':5,321 '1.5':1 '1.5.1':236 'docx':325 'адабиётлар':280,288 'ажратилиш':124 'ажратиш':23 'аксинч':191 'амал':69,133,17
'docx':131 'алгоритм':1 'баз':111 'будут':15 'бумаг':75 'введен':65,67,127 'вебстраниц':63 'внесен':23 'возможн':11 'все
'1.5.3':88 '1.5.4':164 'docx':375 'абстракт':341 'абстракт-мантиқ':340 'ажратиш':213 'амалг':272 'амалларн':353 'аниқ':1
'1.8':185,187 'docx':196 'f':191 'аналитическ':33,68 'архитектур':103,180,188 'базов':113 'безопасн':39,48,90,150,156 'в
'111.xlsx':22 'f':21 'дата1':1 'дата10':19 'дата2':3 'дата3':5 'дата4':7 'дата5':9 'дата6':11 'дата7':13 'дата8':15 'дат
```

Figure 2. Array of lexically significant words

The vector values in this array are arranged in alphabetical order. In this case, the sequential search method cannot be used, because if you need to search for the word “ажратиш” in an array, comparing along the serial path takes a lot of time. In this case, the binary search method is used. It divides the array into two equal parts and determines whether the search is performed on the right or left. The reason is that the array is in alphabetical order, so it continues to search among words starting with the letter "A". This in turn leads to an increase in search speed.

The "to_tsvector ()" function above stores lexical words and information about their location in an array built into the tsvector type. This lexical position of the keyword is used to calculate the number of search results and other information, the participation of the keyword in the search process. The function also supports logical operations. An example is Figure 3. All queries were executed in PostgreSQL.



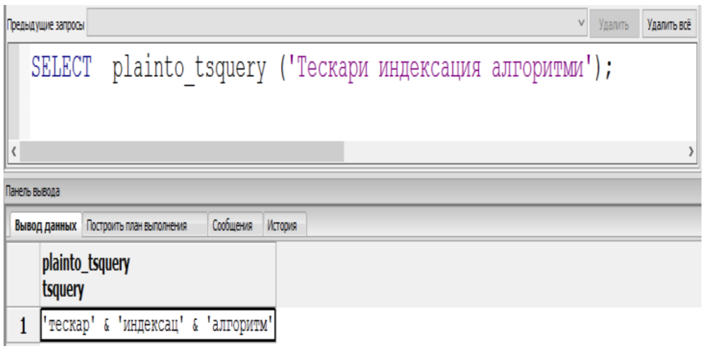
```
SELECT to_tsvector ('Тескари индексация алгоритми барча тизимларда ишлатилади ')
@@ 'тизимларда'
```

	?column? boolean
1	t

Figure 3.

In Figure 3, if the query "тизимларида" is contained in the text, the result returns a true value. If another word is added because the request is not of a text type, the result will return an error. For this, PostgreSQL has two functions that convert tsquery queries to text. These are plainto_tsquery() and to_tsquery().

The plainto_tsquery() function converts a query into a text representation. The function uses the logical operation AND (&) to combine words. As an example, we give Figure 4.

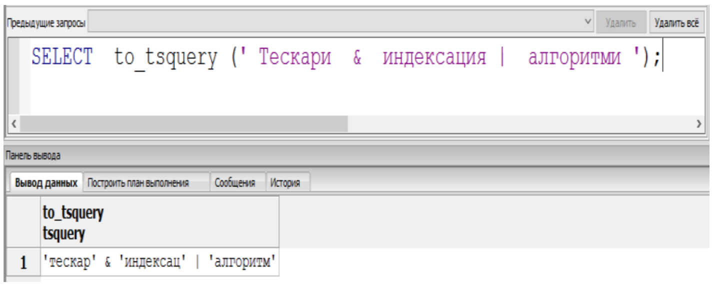


```
SELECT plainto_tsquery ('Тескари индексация алгоритми');
```

	plainto_tsquery tsquery
1	'тескар' & 'индексац' & 'алгоритм'

Figure 4.

The to_tsquery() function supports logical operations in addition to the AND element (&) or the OR element (|). An example is shown in Figure 5.

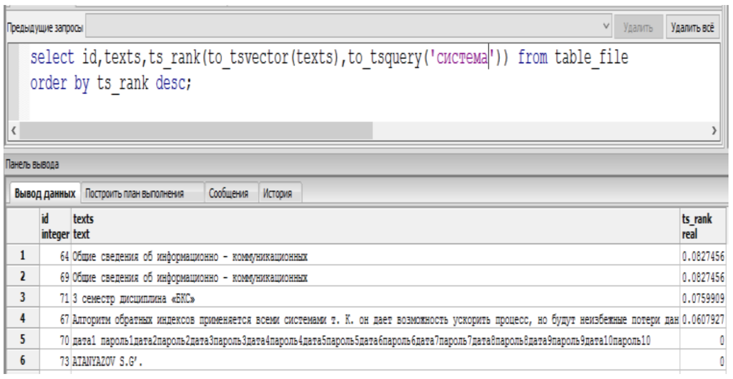


```
SELECT to_tsquery (' Тескари & индексация | алгоритми ');
```

	to_tsquery tsquery
1	'тескар' & 'индексац' | 'алгоритм'

Figure 5.

One of the functions ts_rank() or ts_rank_cd() is used to calculate the number of complete requests for participation in search results. Figure 6 below shows an example of this function.



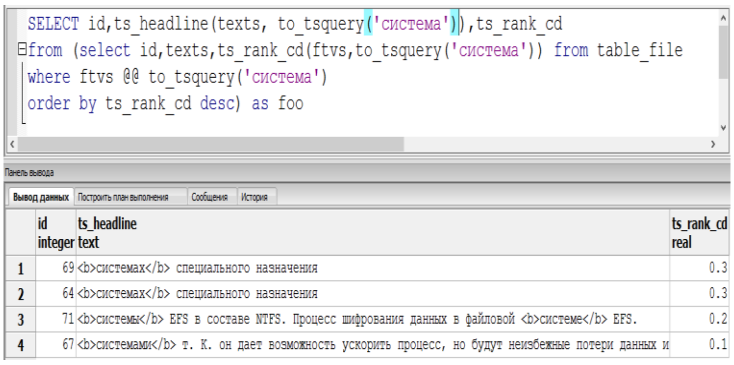
```
select id,texts,ts_rank(to_tsvector(texts),to_tsquery('система')) from table_file
order by ts_rank desc;
```

	id integer	texts text	ts_rank real
1	64	Общие сведения об информационно - коммуникационных	0.0827456
2	69	Общие сведения об информационно - коммуникационных	0.0827456
3	71	3 семестр дисциплина «БКС»	0.0759909
4	67	Алгоритм обратных индексов применяется всеми системами т. К. он дает возможность ускорить процесс, но будут неизбежные потери дан	0.0607927
5	70	дата1 пароль1дата2пароль2дата3пароль3дата4пароль4дата5пароль5дата6пароль6дата7пароль7дата8пароль8дата9пароль9дата10пароль10	0
6	73	ATANYAZOV S.G'.	0

Figure 6.

Therefore, the main purpose of this function is to calculate the amount of participation in the search query according to this query and sort them in descending order.

Another important feature used in full-text search in PostgreSQL is pg_headline(), which allows you to select a keyword from the contents of a search result. An example is shown in Figure 7:



```
SELECT id,ts_headline(texts, to_tsquery('система')),ts_rank_cd
from (select id,texts,ts_rank_cd(ftvs,to_tsquery('система')) from table_file
where ftvs @@ to_tsquery('система')
order by ts_rank_cd desc) as foo
```

	id integer	ts_headline text	ts_rank_cd real
1	69	<b>системах</b> специального назначения	0.3
2	64	<b>системах</b> специального назначения	0.3
3	71	<b>системы</b> EFS в составе NTFS. Процесс шифрования данных в файловой <b>системе</b> EFS.	0.2
4	67	<b>системами</b> т. К. он дает возможность ускорить процесс, но будут неизбежные потери данных и	0.1

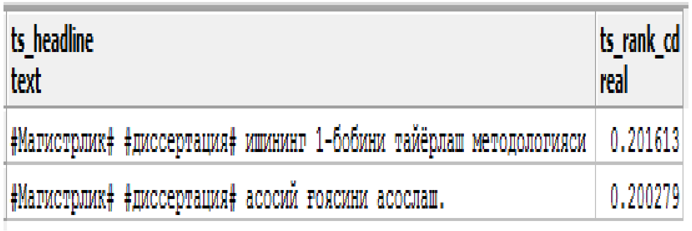
Figure 7.

When a search query is entered using the text search functions listed above, the binary search algorithm quickly and accurately finds information and returns a result matching the query.

Using the above functions, we give the following example:

```
SELECT ts_headline(texts, q, ' startsel = #, stopsel = # '),
ts_rank_cd(ftvs, q)
FROM table_file,
plainto_tsquery(' магистрлик диссертация ') q
WHERE q @@ ftvs
```

The result is as follows:



ts_headline text	ts_rank_cd real
#Магистрлик# #диссертация# ишининг 1-бобини тайёрлаш методологияси	0.201613
#Магистрлик# #диссертация# асосий ғоясини асослаш.	0.200279

Figure 8. Search results.

The search process performs the following sequence of steps.:

- The plainto_tsquery() function converts the search query into text format and separates them into a separate representation of keywords.
- Keywords are searched from an array of lexical words generated by the to_tsvector() function. If keywords are present, the desired file is selected, otherwise it does not return a value.
- The ts_rank() function calculates the number of keywords in the text using an array of keywords and lexical words.
- The pg_headline() function separates the keywords that appear in the text with the symbol attached to them and, finally, returns all the results together.
-

CONCLUSION:

The article analyzes the working mechanisms of existing modern search engines, binary, interpolation and sequential search methods. The results of the analysis can be used to develop general functional requirements for a database of electronic documents when creating an advanced search system.

Based on the foregoing, further studies will study the use of indexing databases and search engines[1,6].

REFERENCES:

1) N. A. Gaydamakin. Automated information systems, databases and banks. M.: Gelios ARV, 2002. 259-260p.
2) Search Algorithms in Arrays http://school-collection.lyceum62.ru/ecor/storage/3ab4a160-cf85-4876-b8da-9bbb1099ac8f/[INF10_04_12_TI_1C].html
3) Search Algorithms http://algol.adept-proekt.ru/algoritms_poiska
4) New Google product contributes to data leakage. http://www.securitylab.ru/favicon.ico
5) A.V. Kirillov. Features of Google Desktop Search. http://hostinfo.ru/favicon.ico
6) Snippet, search back algorithm, page indexing and working features of Yandex http://joomla-s.ru/interesnye-stati/51-seo/uchimsya-nravitsya-yandeksu-i-google/158-cnippet-algoritm-obratnogo-poiska-indeksatsiya-stranits-i-osobennosti-raboty-yandeksa