A Novel Representation and Searching Algorithm for Opening Hours

Teodora Husar
Department of Computer Science and Information Technology,
University of Oradea
Oradea, Romania

Robert Győrödi
Department of Computer Science and Information Technology,
University of Oradea
Oradea, Romania

Cornelia Győrödi
Department of Computer Science and Information Technology,
University of Oradea
Oradea, Romania

Sorin Sarca
Department of Informatics, Faculty of Sciences
University of Oradea
Oradea,
Romania

***Abstract*—Opening Hours can be considered a data type having a human representation; this means that it can be easily understood by human beings and hardly understood by computers because the lack of a standard structured representation. In essence, the opening hours gives us a simple information: opening state at a certain date and time, and this is our focus in this paper. So far, this kind of functionality does not exists in today's database management systems because there are no algorithms developed in this way. The purpose of this paper is to presents a novel and easy to implement algorithm for encoding opening hours in order to quickly search and get the opening state for records.**

Keywords—Opening Hours; Java; optimizations

I. Introduction

A few years ago an application normally only used to have thousands of users to tens of thousands of users in the most extreme cases, and currently there are applications that have millions of users and amount of data is increasing, so is very important to perform efficient search to get relevant information from data [7]. It is important to use search engines to access information, that has grown tremendously based on the needs of users [2]. The goal of information extraction methods is the extraction of specific information from data documents [1].

Opening Hours are becoming increasingly significant in online environment, because it is very important to know when a restaurant, store or other business is open, this information can help you avoid unnecessary roads, or in late hours you can search for open places where you can fix urgent problems: dentist, auto service and so on. Businesses is usually listed in directories where you can perform a search to get relevant results, but these results are not relevant to current time, nor the results will not be relevant for a future date.

If there would be a way to filter the results only get the ones that are open now or in a future date, then you will save a lot of time, otherwise you have to loop over the result list and check each opening hour. So far, there is no efficient algorithm to solve this problem.

This paper proposes an encoding of the opening hours, which can show in a very short time if it is opened or closed at a certain date and time. There are many representations for opening hours but this encoding is not assuming any, because there is not a standard representation. Instead it is a byte sequence, not easily readable by humans, which encapsulates opening hours info, so that if given a date and hour it can determine very fast if it is open or closed. This paper makes a first attempt to describe a new algorithm for searching through opening hours. To perform comparisons tests, we also achieved an implementation of algorithm written in Java language [4][5] for Apache Solr [6].

II. The Algorithm Representation

Opening Hours can be composed from many entities (day of week, date, time) and to easily represent them we must choose some notations. We consider the following primitives:

- Z - represents the day of the week using one digit, 1 (for Monday) through 7 (for Sunday)
- H - represents hour of the day using four digits, 0000 (for 00:00) to 2359 (for 23:59)
- D - represents the date of year (only month and day) using four digits, 0101 (for January 1) to 1231 (for December 31).

Having these three primitives, we can move forward and create some types representing time intervals.

A. ZHH type

With this format, we can encode one interval of opening hours in a single day. Therefore, encoding “Monday from 8:00 to 16:00” will produce 108001600. We can see this representation in Table 1 that indicates every primitive used:

TABLE. I. THE REPRESENTATION OF THE ZHH TYPE

Z	H	H
1	0800	1600
Monday	8:00	16:00

B. DHH type

Similar to ZHH but specifies an exact date instead of day of week. This is just great if you have some special hours in a date, because it can encode something like "Opened on December 25 from 9:00 to 12:00" to 122509001200, and the Christmas day is saved. We can see this representation in Table 2.

TABLE. II. THE REPRESENTATION OF THE DHH TYPE

D		H	H
1225		0900	1200
12	25	9:00	12:00
Dec	25		

C. ZDDHH type

This type extends ZHH and DHH types by adding a date interval. More precisely, it can encode something like "Opened every Sunday from 9:00 to 18:00 between February 12 and March 16" to 70212031609001800, so you can easily represent opening hours for seasons. The date interval must be greater than a week, otherwise you can use ZHH or DHH types. This representation is shown in Table 3.

TABLE. III. THE REPRESENTATION OF THE ZDDHH TYPE

Z	D		D		H	H
7	0212		0316		0900	1800
Sunday	02	12	03	16	9:00	18:00
	Feb	12	Mar	16		

D. -D

We can use this type to represent a closed date. Therefore, if we close on December 25 the encoding will be -1225. This representation is shown in Table 4.

TABLE. IV. THE REPRESENTATION OF THE -D TYPE

-	D	
-	1225	
Closed	12	25
	Dec	25

E. -DD

This type is just like -D type but specifies a date interval instead of a single date. Can encode a range of closed dates, "Closed between January 1 and January 3" to -01010103. We show this representation in Table 5.

TABLE. V. THE REPRESENTATION OF THE -DD TYPE

-	D		D	
-	0101		0103	
Closed	01	01	01	03
	Jan	1	Jan	3

Examples of encoded hours

Now that we have our types, we can go ahead and try to encode some opening hours. For example:

- Monday – Friday from 8:00 to 16:00

For each day of week, we must add a separate record of type ZHH: 108001600, 208001600, 308001600, 408001600, 508001600.

- Saturday from 9:00 to 13:00

Simply use a ZHH record: 609001300.

- Sunday - closed

We do not have to do anything, if there is no other data for opening hours we assume it is closed.

- Monday – Friday from 8:00 to 12:00 and from 14:00 to 18:00

Because each day contains two intervals of opening time we must add a ZHH record for each interval: (108001200, 114001800), (208001200, 214001800), …, (508001200, 514001800).

- Sunday from 9:00 to 12:00 – only during summer season

Considering the summer season as being between June 1 and August 31, we could just use a ZDDHH record: 70601083109001200.

There are some special hours:

December 24 from 9:00 to 13:00

December 25 - closed

January 1 & 2 - closed

For December 24 we must use a DHH record: 122409001300, for December 25 a -D record: -1225 and for January 1 & 2 a -DD record: -01010102.

We consider specifying a closed date record (-D or -DD) implies that you also use the other types to specify opening hours, otherwise it is pointless.

Special cases are:

- Opened 24/7

We must use seven ZHH records to specify the opening interval as 00:00 – 23:59 for each day of week: 100002359, 200002359, …, 700002359. Please note that specifying interval 00:00 – 00:00 is like saying that it is open only at 00:00 (exactly one minute, until 00:01).

- Monday – Friday from 18:00 to 02:00

In this case, the opening interval is past midnight, so it means we must split them in ZHH records to mark as open the first two hours of next day: (118002359, 200000200), (218002359, 300000200), …, (518002359, 600000200).

If Saturday or Sunday is closed, we must not specify a -D record for it.

F. Evaluation order algorithm

So far, we defined representation types and we know how to represent opening hours using our types. However, to know for a specified date and hour if it is opened or not, we consider that each opening hours contains a list of encoded representations in different types. We also set the evaluation order from the most specific to most general type and compare our date and hour to specified representation. For our representation types, the priorities are (from most important to less important): -D, -DD, DHH, ZDDHH, ZHH. We define further the related algorithm for determining opening state:

Let X = searched date (month and day)

Let Y = searched time (hour and

minute)

Step 0

Let Q = day of week for X

Step 1

Get next -D, if no -D goto Step 2. If X equals D return CLOSED else repeat Step 1.

Step 2

Get next -DD, if no -DD goto Step 3. If X between DD return CLOSED else repeat Step 2.

Step 3

Get next DHH, if no DHH goto Step 4. If X equals D and between HH return OPEN else repeat Step 3.

Step 4

Get next ZDDHH, if no ZDDHH goto Step 5. If Q equals Z and X between DD and Y between HH return OPEN else repeat Step 4.

Step 5

Get next ZHH, if no ZHH goto Step 6. If Q equals Z and Y between HH return OPEN else repeat Step 5.

Step 6

Return CLOSED.

III. Sorting and Searching Algorithm

In this section, we describe in pseudocode the logic for algorithm. We are not making any assumptions for the original format of opening hours because there is not a standard to represent them, so there could be many implementations. We just consider that there exists a list of records encoded using our representation types.

A. Sorting the list of records

Each type has different length (except closing ones, which will be put first) so we will use that to sort the records.

```
set Z to 1
set H to 4
set D to 4
set _D to D + 1
set _DD to _D to D
set DHH to D + H + H
set ZDDHH to Z + D + D + H + H
set ZHH to Z + H + H
set list[0] to  empty array
set list[1] to empty array
set list[2] to empty array
set list[3] to empty array

loop
 read record
 set len as record length
 if first char of record is '-' then
        if len is _D or _DD then
                index is 0
 else
        if len is DHH then index is 1
        if len is ZDDHH then index is 2
        if len is ZHH then index is 3

 push into list[len] the record

  concatenate list into one array:
 final list is list[0] concatenated with list[1]
concatenated with list[2] concatenated with list[3]
```

B. Searching through records

We consider that we have a sorted array of records. In the code below, we return if the records contains open day based on a specific date, hour and year.

```
set Z to 1
set H to 4
set D to 4
set _D to D + 1
set _DD to _D to D
set DHH to D + H + H
set ZDDHH to Z + D + D + H + H
set ZHH to Z + H + H
set delimiter to ';'
set close to false
```

```
set firstDigit to 0 // used for the day of the week
set length to 0 // used for type
set interval to 0
set cType to 0

read list that is a sorted records delimited by
semicolon
read date, hour, year
set currentD as week number for the corresponding
read data

loop through list as record
   if record is '-' then
        set close to true and continue
   if record is a number then
        if firstDigit is 0 then set to record
        set interval with interval * 10 + record
        else if record is delimiter then
         if close then
            if date is the interval or is between
interval then
            return false
            set close to false and continue
            if cType is more than 0 and  not the
same as previous type then
               return false
            if length is DHH then
               if date, hour and year is between the
interval then
               return true
               set cType to DHH
               if length is ZDDHH then
                if firstDigit is currentD and date,
hour and year is between the interval then
                return true
               set cType to ZDDHH
               if length is ZHH then
                 if firstDigit is currentD and date,
hour and year is between the interval then
                return true
               set cType to ZHH
               set inverval,fistDigit and length to
0
return false
```

IV. BENCHMARKS

When comparing the performance of two search algorithms or two sorting algorithms, we consider two types of operations: data movements, or swaps and comparisons [3].

Because the test results depend on the computer on which these tests are carried out, it is important to note that all the results presented below in Table 6, were obtained from studies conducted on a computer with the following characteristics: processor Intel Core i7, 4 GB RAM memory and 320 GB SSD.

TABLE. VI. THE RESULTS OF TESTS

Implementation	Test	Time
Java	1.000.000 iterations	930ms
Java	12.000.000 iterations	10s
Java*	1.000.000 iterations	450ms
Java*	25.000.000 iterations	10s
Apache Solr no caching	100.000 documents	100ms full search

The results are based on the code written in the Java language by the authors [8].

In Java* implementation we can improve the speed of *isOpen()* function by removing the code that generates the *currentDay* integer (by creating a new Calendar instance), and pass it as an argument.

From results tests presented in Table 6 we can say that the resulted times are excellent: we can search through 1.000.000 database records/documents in less than one second and with optimization in less than half a second.

For *Apache Solr* the search through 100.000 documents took 100 milliseconds, which is great. In a real life case there will often be some enforced search criteria such as a limit or a category, city and so on; because it does not make sense to return 100.000 documents to end-user. In other words, the algorithm can handle very well millions of records if other search criteria are used.

Without taking into account the database management system where this algorithm is implemented, for the best results one must consider to:

- use a separate field for the encoded opening hours. Even if you can reverse the encoding, it is not a good idea because the format is intended for search only, so use your original format of opening hours if you want to show it to your users
- do the sorting once (when you insert or update), doing it on every search request will slow-down the process
- do not return the value of encoded field (to save bandwidth and speed-up the search)
- make sure that the check is done only when necessary. Remember that if you have A && B, B is not evaluated if A evaluates to false
- remove redundant records from opening hours (for example having a -D between a -DD is useless

V. CONCLUSIONS

To sum-up, the advantages of the algorithm are:

- O(n) complexity for search (in the worst case)
- covers all cases for opening hours, including special hours or season hours

- easy to extend (for example adding time zone support) and implement since it operates on a byte sequence

The only downside of algorithm is the size of needed byte sequence. However, when it comes to search, we will not have to worry about disk or ram space because speed comes first. However, there are some possible workarounds:

- add another types, such as ZZHH (which extends ZHH, using ZZ as interval) and a special one to represent always open
- use a fast method to pack and unpack the sequence (you can reduce the size to half by using something similar to BCD – binary coded decimal)

In the end we can say that our method is a good proof of concept for encoding opening hours in order to determine if is open or closed at certain date and time considering the low complexity and high speed of search.

REFERENCES

[1] Jadhav Bhushan G, Warke Pushkar U, Kuchekar Shivaji P, Kadam Nikhil, "Searching Research Papers Using Clustering and Text Mining", International Journal of Emerging Technology and Advanced Engineering, Volume 4, Issue 4, April 2014, Available: http://ijetae.com/files/Volume4Issue4/IJETAE_0414_135.pdf, accessed jan 2016.

[2] E.A. Calvillo, A. Padilla, J. Munoz, J.T. Fernandez, "Searching research papers using clustering and text mining", International Conference on Electronics, Communications and Computing (CONIELECOMP), 11-13 March 2013, pp. 78 – 81, ISBN 978-1-4673-6156-9, Available:
https://www.researchgate.net/publication/261036444_Searching_research_papers_using_clustering_and_text_mining, accessed jan 2016.

[3] Amy Csizmar Dalal, "Searching and Sorting Algorithms", Supplementary Lecture Notes, 2004, Available: http://www.cs.carleton.edu/faculty/adalal/teaching/f04/117/notes/searchSort.pdf, accessed jan 2016.

[4] S. Wild and M. E. Nebel, *"*Analysis of Yaroslavskiy's dual-pivot quicksort used in Java 7*"*. In Proceedings of the 20th European Symposium on Algorithms, 2012.

[5] Java programming language, https://docs.oracle.com/en/java/

[6] Apache Solr, Available: http://lucene.apache.org/solr/

[7] Cornelia Győrödi, Robert Győrödi, George Pecherle, Andrada Olah, "A comparative study: MongoDB vs. MySQL", IEEE - 13th International Conference on Engineering of Modern Electric Systems (EMES), 2015, Oradea, Romania, 11-12 June 2015, ISBN 978-1-4799-7649-2, pag. 1-6.

[8] Solr Opening Hours solutions, Available: https://github.com/husart/SolrOpeningHours