A Binary Search For Non-Unique Keys

Derek Drummond, ARC/Professional Services Group

Abstract

This paper will describe a binary search algorithm that allows users to specify a range of key values. Most common binary search algorithms require a single key value and will search for only one observation. In fact, these algorithms assume that each key value in the file to be searched can be found on only one observation (i.e., key values are unique).

For our modified binary search algorithm, the key values do not have to be unique. The algorithm finds the first and last observations of a group of observations having the same key value. Also, the algorithm can search for a range of key values, finding the first and last observations that satisfy the range requested. Even if the file to be searched contains only unique key values, this modified algorithm can be used.

This paper will be divided into three major sections. The Introduction section describes the purpose of binary search algorithms in general, some advantages and disadvantages of our modified algorithm, and major concerns related to a range search. The Procedures section describes details of the algorithm and includes a SAS® coded example of the algorithm. The Application-section describes an actual problem that was resolved using this type of algorithm.

Introduction

Recently, I was faced with a need to search a large range of observations interactively. The range was (and is being) updated daily and must continue to be available to other applications with minimal modifications. Also, the range contained numerous records with the same (non-unique) key values.

I considered using hashing, indexing, or binary search algorithms to search the range. Each of these algorithms had "drawbacks". I chose to modify a standard (unique-key value) binary search algorithm so that it could be used to search ranges of non-unique key values.

A binary search is an algorithm that successively divides the range to be searched in half until the key value being sought is found. During each iteration, the algorithm determines which half of the range must contain the key value being sought and discards the other half of the range. This is done by comparing the value of the data found at midpoint of the range against the value being sought. If the value of the data found at the midpoint is greater than the value being sought, the top half of the range is discarded. The bottom half of the range is discarded where the data found at the midpoint is less than the value being sought. After only 20 iterations of a binary search, a range of 16 million observations could be reduced to less than 20 observations.

Standard binary search algorithms access ranges that have a different (unique) key value for each observation. Such searches terminate when the value being sought is found. However, to solve my problem of having to search ranges that contain groups of observations with same (non-unique) key value, the search should continue until the first and last observations of the group are found. Notice that the standard search algorithm can be viewed as being a special version of this algorithm; a version that assumes that the first and last observations are the same observation.

I rejected hashing because of anticipated problems with the 'collision' resolution steps of such algorithms. Collision resolution is the process performed by hashing algorithms when two or more entries have the same 'hashed' (key) value. When two or more entries have the same key value, a collision has occurred. The first entry is stored on the observation pointed to by the hashing function and the second and later entries are stored in an overflow area. Often a significant amount of a range is required for the overflow area; possibly as much as 50%. I barely had enough disk storage space to store the anticipated 4 million observations. Also, hashing works best when collisions are minimized. Non-Unique key values infer numerous collisions. For large groups of non-unique keys, hashing could spend a significant amount of time performing collision resolution.

I rejected indexing because it requires the maintenance of a separate index range. Other applications that use this file might change the file without rebuilding the index range. Also, efficient indexing often requires that the range be sorted prior to indexing. Once the range is sorted, a binary search algorithm could be used without the time consuming process of creating the index range. In fact, while using a binary search algorithm, you can change from one key variable to another by simply resorting the range by the new key variable(s).

When can a binary search for non-unique key values be used? It should be used any time the user
requires that all observations with a specific key value be made available. One example is to search a very large range for all observations that contains a specific date.

Procedures

A standard binary search procedure is comprised of two major sections: initialization and search loop. In the initialization section, the TOP and BOTTOM of the range are determined. Normally, BOTTOM is assumed to be 1 and TOP is usually assumed to be the last observation in the data set being searched. Some implementations allow users to specify BOTTOM and TOP values as parameters.

The search loop determines which observation is the midpoint (MID) of the range. MID is calculated by adding the value of TOP to the value of BOTTOM and the result is divided by 2. Then, the data on that observation is compared against the value being sought (key value). A search statement with the POINT option is used to access that observation. If the data on that observation equals the key value, the loop stops and the algorithm assumes that the key value was found at MID. If the key value is not on that observation, the half of the range that could not contain the key value is discarded and the algorithm continues to search the other half of the range. The algorithm determines which half to keep and which half to discard by further comparing the key value and the data found on the observation at MID. If the data at MID is less than the key value, TOP is reassigned as the value of MID minus 1 and the upper half of the range is ignored (discarded) during the rest of the search. If the data at MID is greater than key value, BOTTOM is reassigned as the value of MID plus 1 and the bottom half is discarded. Note, it is possible that the key value will not be found in the range. This is the case whenever BOTTOM has a value that is greater than the value of TOP. A typical search loop may look like the following:

```sas
found=0;
_stop=0;
do while(not _stop);
  _mid=int(( _bottom+_top)/2);
  set sasdata point= _mid;
  if value < _keyval then
    _bottom= _mid + 1;
  else if _mid= _bottom then
    _stop=1;
  else do;
    _top= _mid - 1;
    set sasdata point= _top;
    if value < _keyval then
      _stop=1;
  end;

/* _keyval is not in range*/
  if _bottom>_top then
    _stop=1;
end;
```

After the standard binary search algorithm is complete, two variables FOUND and MID are returned. FOUND indicates whether or not the key value was found.

A SET statement with the POINT option (POINT=MID) can be used to access the observation containing the key value.

A binary search for non-unique keys is simply a standard binary search algorithm with two additional search loops. These loops find the first and last observations that contain the key value. To find the first observation, the algorithm searches for an observation that has the key value and is preceded by an observation that does not.

```sas
/* look for start of group */
holdb= _mid; _holdt= _top;
_top= _mid;
_stop=0;
do while(not _stop);
  _mid=int(( _bottom+_top)/2);
  set test1 point= _mid;
/* note, all values are less than or equal to the value being sought */
  if value < _keyval then
    _bottom= _mid + 1;
  else if _mid= _bottom then
    _stop=1;
  else do;
    _top= _mid - 1;
    set test1 point= _top;
    if value < _keyval then
      _stop=1;
  end;
_end;
_first= _mid;
```

To find the last observation, the algorithm searches for an observation that has the key value and is followed by an observation that does not.
\[ \text{/* look for end of group */}\]
\[ \_bottom= \_holdb; \_top= \_holdt; \_stop=0; \]
\[ \text{do while(!\_stop);} \]
\[ \_mid=\text{int}((\_bottom+\_top)/2); \]
\[ \text{set test1 point= \_mid;} \]
\[ \text{/* note all values must be} \]
\[ \text{greater than or equal} \]
\[ \text{to the value being} \]
\[ \text{sought */}\]
\[ \text{if value> \_keyval then} \]
\[ \_top= \_mid - 1; \]
\[ \text{else if \_mid=\_top then} \]
\[ \_stop=1; \]
\[ \text{else} \]
\[ \text{do;} \]
\[ \_bottom= \_mid + 1; \]
\[ \text{set test1 point= \_bottom;} \]
\[ \text{if value< \_keyval then} \]
\[ \_stop=1; \]
\[ \text{end;} \]
\[ \text{end;} \]
\[ \_last= \_mid; \]

After the binary search for non-unique keys is complete, three variables are returned: FOUND, FIRST and LAST. FOUND indicates whether or not any observation with the key value was found. FIRST points to the first observation with the key value. LAST points to the last observation with the key value. A SET statement with the POINT option can be used in conjunction with a series of DO group statements to access each observation in the group.

With a few modifications, both the standard and non-unique algorithms could be used to search ranges that are sorted in descending order. Ascending is assumed to be the sort order. Simply change the "greater than" comparison statements to "less than" and the "less than" comparisons to "greater than".

Applications

This algorithm was developed because I was faced with having to search a file of over four million observations. The file contained transactions from over 100 sites across the nation. Approximately 1-2 million transactions were added to the file daily. The file was sorted by the date and time of the transactions. However, I needed to find all transactions for one site that occurred within a specific time frame. Also, the search had to be performed interactively.

The first thing I did was to change the sorting step to sort the data by site and then by date and time. This step had no negative impact on any of the other applications that accessed this data. It did have a positive impact on one application that was resoring the data by site, date, and time. That sort step can now be discarded.

Next, I tried to use a standard binary algorithm to locate an observation from the specified site, date and time frame. It took only 5 iterations of the search loop to find an observation with the desired value. Then, I performed two sequential searches; one to find the first observation of the group and the second to find the last. It took 589 iterations to find the first observation in the group and 917 to find the last of the group. There were over 1500 observations in the group.

I decided to add the two additional search loops to the standard algorithm I was using and searched again for the group of 1500+ observations. Only 17 iterations of the first loop were required to locate the first observation of the group. Also, 17 iterations of second loop were required to locate the last observation of the group.

The range being searched had 4,005,298 observations. The desired group was found to start at observation 124,576, end at observation 126,083 and contain 1,508 observations (size=128,03-124,526+1). The following tables contains the values of the control variables used in the algorithm after each iteration:

<table>
<thead>
<tr>
<th>Method 1. (Sequential search starting with observation 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>124,576</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method 2. (Binary search followed by sequential searches)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find any key value:</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
</tr>
<tr>
<td>Iteration</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

The following tables contains the values of the control variables used in the algorithm after each iteration:

Method 1. (Sequential search starting with observation 1)

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Bottom</th>
<th>Top</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4,005,298</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4,005,298</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>124,576</td>
<td>124,576</td>
<td>4,005,298</td>
</tr>
</tbody>
</table>

Method 2. (Binary search followed by sequential searches)

<table>
<thead>
<tr>
<th>Find any key value:</th>
<th>Find first:</th>
<th>Find Last:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>Bottom</td>
<td>Top</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4,005,298</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2,002,648</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>594</td>
<td>124,576</td>
<td>1512</td>
</tr>
</tbody>
</table>
Method 3. (Binary search followed by two binary searches)

Find any key value:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Bottom</th>
<th>Top</th>
<th>Mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4,005,298</td>
<td>2,002,649</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2,002,648</td>
<td>1,001,324</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1,001,323</td>
<td>500,662</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>500,661</td>
<td>250,331</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>250,330</td>
<td>125,165</td>
</tr>
</tbody>
</table>

Find first:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Bottom</th>
<th>Top</th>
<th>Mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>125,165</td>
<td>62,583</td>
</tr>
<tr>
<td>7</td>
<td>62,584</td>
<td>125,165</td>
<td>93,874</td>
</tr>
<tr>
<td>8</td>
<td>93,875</td>
<td>125,165</td>
<td>109,520</td>
</tr>
<tr>
<td>9</td>
<td>109,521</td>
<td>125,165</td>
<td>117,343</td>
</tr>
<tr>
<td>10</td>
<td>117,344</td>
<td>125,165</td>
<td>121,254</td>
</tr>
<tr>
<td>11</td>
<td>121,255</td>
<td>125,165</td>
<td>123,210</td>
</tr>
<tr>
<td>12</td>
<td>123,211</td>
<td>125,165</td>
<td>124,188</td>
</tr>
<tr>
<td>13</td>
<td>124,189</td>
<td>125,165</td>
<td>124,677</td>
</tr>
<tr>
<td>14</td>
<td>124,189</td>
<td>124,676</td>
<td>124,432</td>
</tr>
<tr>
<td>15</td>
<td>124,433</td>
<td>124,676</td>
<td>124,554</td>
</tr>
<tr>
<td>16</td>
<td>124,555</td>
<td>124,676</td>
<td>124,615</td>
</tr>
<tr>
<td>17</td>
<td>124,555</td>
<td>124,614</td>
<td>124,585</td>
</tr>
<tr>
<td>18</td>
<td>124,555</td>
<td>124,584</td>
<td>124,569</td>
</tr>
<tr>
<td>19</td>
<td>124,570</td>
<td>124,584</td>
<td>124,577</td>
</tr>
<tr>
<td>20</td>
<td>124,570</td>
<td>124,576</td>
<td>124,573</td>
</tr>
<tr>
<td>21</td>
<td>124,574</td>
<td>124,576</td>
<td>124,575</td>
</tr>
<tr>
<td>22</td>
<td>124,574</td>
<td>124,576</td>
<td>124,576</td>
</tr>
</tbody>
</table>

Find last:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Bottom</th>
<th>Top</th>
<th>Mid</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>125,165</td>
<td>250,330</td>
<td>107,747</td>
</tr>
<tr>
<td>24</td>
<td>125,165</td>
<td>197,746</td>
<td>156,455</td>
</tr>
<tr>
<td>25</td>
<td>125,165</td>
<td>156,454</td>
<td>140,809</td>
</tr>
<tr>
<td>26</td>
<td>125,165</td>
<td>140,808</td>
<td>132,986</td>
</tr>
<tr>
<td>27</td>
<td>125,165</td>
<td>132,985</td>
<td>129,075</td>
</tr>
<tr>
<td>28</td>
<td>125,165</td>
<td>129,074</td>
<td>127,119</td>
</tr>
<tr>
<td>29</td>
<td>125,165</td>
<td>127,118</td>
<td>126,141</td>
</tr>
<tr>
<td>30</td>
<td>125,165</td>
<td>126,140</td>
<td>125,652</td>
</tr>
<tr>
<td>31</td>
<td>125,653</td>
<td>126,140</td>
<td>125,896</td>
</tr>
<tr>
<td>32</td>
<td>125,897</td>
<td>126,140</td>
<td>126,018</td>
</tr>
<tr>
<td>33</td>
<td>126,019</td>
<td>126,140</td>
<td>126,079</td>
</tr>
<tr>
<td>34</td>
<td>126,080</td>
<td>126,140</td>
<td>126,110</td>
</tr>
<tr>
<td>35</td>
<td>126,080</td>
<td>126,109</td>
<td>126,094</td>
</tr>
<tr>
<td>36</td>
<td>126,080</td>
<td>126,093</td>
<td>126,086</td>
</tr>
<tr>
<td>37</td>
<td>126,080</td>
<td>126,085</td>
<td>126,082</td>
</tr>
<tr>
<td>38</td>
<td>126,083</td>
<td>126,085</td>
<td>126,084</td>
</tr>
<tr>
<td>39</td>
<td>126,083</td>
<td>126,083</td>
<td>126,083</td>
</tr>
</tbody>
</table>

Summary

In summary, our modified binary search algorithm is able to quickly locate the first and last observations of groups containing the same key value, regardless of the size of the groups. Such algorithms are useful when searching for groups of unknown size.

The author can be contacted at:
Atlantic Research Corporation,
Information Systems Division
1301 Piccard Drive, 2nd Floor
Rockville, MD 20850
(301) 258-5300

SAS® is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

References
Ray, Craig, 'A Comparison of Table Lookup Techniques',
SUGI '87
The following is an example of a macro that contains a binary search for non-unique keys. In addition, the macro allows users to select:

1. whether to use an unique (standard) or non-unique search algorithm;
2. a range of key values (for non-unique searches only);
3. whether range is sorted in ascending or descending order.

Due to space restrictions some comments, spacing and parameter validation have been removed from the following macro. Also, indentation has been kept to a minimum. Note, the 'keys' parameter is included for applications that require the key value to be calculated.

%macro bsearch(table=, keys=, unique=YES, lowvalue=, highvalue=, order=A, first=t, last=);
%let unique = %upcase(&unique);
%let order = %upcase(&order);
_found = 0;
_bottom = &first;
%if &last ::: %then
%do;
_top = &last;
%end;
%if &order eq D %then
%do;
%let gt = gt;%let firstval = &highvalue;
%let lastval = &lowvalue;
%end;
%else
%do;
%let lt = lt;%let firstval = &lowvalue;
%let lastval = &highvalue;
%end;
_length_text $ 80;
do while (not _stop);
   _mid = int(Lbottom + _top) / 2);
set &table point = _mid;
_key = &keys;
if _key &lt &firstval then
   _bottom = _mid + 1;
else if _key &gt &lastval then
   _top = _mid - 1;
else
do;
   _found = 1;
   _stop = 1;
end;
if _top < _bottom then
   _stop = 1;
end;
%if &unique ne NO %then
   goto exit;
if not _found then
go to endsrch;
_holdb = _mid;
_holdt = _top;
_top = _mid;
_stop = 0;
do while (not _stop);
   _mid = int(_bottom + _top) / 2);
set &table point = _mid;
_key = &keys;
if _key &lt &lt &firstval then
   _bottom = _mid + 1;
else if _mid eq _bottom then
   _stop = 1;
else
do;
   _top = _mid - 1;
   set &table point = _top;
   _key = &keys;
if _key &gt &lt &firstval then
   _stop = 1;
else
end;
end;
_first = _mid;
_bottom = _holdb;
_top = _holdt;
_stop = 0;
do while (not _stop);
   _mid = int(_bottom + _top) / 2);
set &table point = _mid;
_key = &keys;
if _key &gt &lt &lastval then
   _top = _mid - 1;
else if _mid eq _top then
   _stop = 1;
else
   _bottom = _mid + 1;
   set &table point = _bottom;
   _key = &keys;
if _key &lt &gt &lastval then
   _stop = 1;
else
   _last = _mid;
endsrch:_stop = 1;
%exit;%put end of bsearch macro;
%mend bsearch;