Paper 031-2007

Quick Record Lookup without an Index

Paul A. Choate, California State Developmental Services

ABSTRACT

Would you like to find one record out of a billion in the time it takes to read thirty records? Does your sorted data not have an index? This paper demonstrates how to code a simple, yet extremely powerful binary tree search that can be used on any sorted SAS® dataset.

INTRODUCTION

As with most database software the SAS System allows data sets to be indexed on key variables. An index is a secondary file that stores the unique values of the key and their positions on the data set. Once a data set is indexed SAS will directly lookup a record when a WHERE clause is used. Up to a certain percentage of records, the index will retrieve data much more quickly than a full pass over the data using a subsetting IF statement, which does not take advantage of the presence of the index. The cost of this utility is the construction, storage, and maintenance of appropriate indexes.

If SAS data files are sorted but not indexed, a similar lookup using a binary tree search can be constructed within a DATA step. This method starts at the middle record of a file, compares it to the lookup value and then branches either up or down one quarter of the number of records of the file and compares again. This process is repeated until the record is found, with the maximum number of reads to find any record being log2(nobs)+1, where nobs is the number of observations in the data set being searched. For example, to find one record out of ten million takes at most 24 reads, and to find one out of a billion takes at most 30 reads.

SEQUENTIAL VERSUS INDEXED LOOKUP

To demonstrate the advantage of a standard indexed lookup, consider a data set of ten million odd numbers, in sort order and also indexed:

```
DATA BigData(INDEX=(id));
     DO id=1 TO 2e7 BY 2;
          OUTPUT;
     END;
RUN;
```

Without the use of the index SAS must look across all ten million records to find the matches, in this case taking 1.68 seconds of real time:

```
DATA findit;
     SET BigData;
     IF id in (05555555, 00015562, 16555557, 19933980, 00000001);
RUN;
```

Using the index, SAS uses a binary tree search of the index table to find the matching records and then reads them directly, in this case requiring 0.01 seconds, or over one hundred times more quickly:

```
DATA findit;
     SET BigData;
     WHERE id in (05555555, 00015562, 16555557, 19933980, 00000001);
RUN;
```

If you have data that are sorted but not indexed, a similar savings can be achieved with a binary tree search coded in a DATA step.

CODING A SORTED LOOKUP WITH A DATA STEP

To code a binary tree lookup the lookup targets should be placed in a SAS data set. The lookup data does not need to be sorted.

```
DATA IDData;
ID=05555555; OUTPUT;
ID=00015562; OUTPUT;
ID=16555557; OUTPUT;
```

```
ID=19933980; OUTPUT;
ID=00000001; OUTPUT;
RUN;
```

The binary tree search for looking up unique keys in the BIGDATA dataset can then be coded as such:

```
DATA Found(DROP=_:) Not_Found(KEEP=ID);
     SET IDData;

     _low=1; _high=nobs; _direction=0; _tries=0;

     DO WHILE(_high>_low AND ID NE _target AND _tries<(LOG2(nobs)+2));
          _tries+1;

          IF _direction<0 THEN _high=(_mid-1)<>1; ELSE
          IF _direction>0 THEN _low=(_mid+1)><nobs;
          _mid=int((_low+_high)/2);

          SET BigData(KEEP=ID RENAME=(ID=_target))
              point=_mid nobs=nobs;
          _direction= SUM(((ID<_target)*(-1)),((ID>_target)*1));
     END;

     SET BigData(RENAME=(ID=_target)) POINT=_mid;

     IF ID=_target THEN OUTPUT Found;
                   ELSE OUTPUT Not_Found;
RUN;
```

The data step finds the same records in 0.03 seconds real time. This demonstrates that when SAS data files are stored as sorted on a key variable but are not indexed, this method finds records with similar efficiency as an index. This datastep lookup can easily be wrapped in a macro, or separate versions maintained for different systems and stored for inclusion with an %INCLUDE statement. A simple extension of this method is shown below that finds ranges of non-unique keys by reading up and then down the file until all matching records are found.

With the SORTEDBY option, PROC SQL can offer similar savings on a sorted-but-not-indexed dataset lookup, but the SQL method is restricted by available memory.

COMPARISON TO AN SQL JOIN MERGE OF SORTED DATA

PROC SQL performs enhanced lookups for sorted but non-indexed datasets when memory is sufficient. The SQL equivalent is coded like this:

```
PROC SQL;
  CREATE TABLE found AS
    SELECT b.*
    FROM BigData (SORTEDBY=ID) b,
         IDData i
    WHERE b.ID = i.id ;
QUIT;
```

For the example of finding five out of ten million records above, the SQL procedure takes 3.01 seconds where the binary tree search takes 0.03 seconds. In other situations SQL may run in a similar speed as the data step binary lookup. Extending the above example, finding five out of one hundred million records the SQL results are 43.35 seconds and the data step 0.31 seconds. The advantage is that the datastep lookup is not limited by available memory.

So far we have considered data sets with unique keys. The binary tree lookup can be extended to work with non-unique keys.

CODING LOOKUPS WITH A NON-UNIQUE KEY VARIABLE

The above example works for a database with unique keys, to find non-unique keys the method must be expanded. After a matching record is found using the same initial lookup, preceding records are examined until the first record matching the ID is found. Then the following records are output until the ID changes:

```
/*  18 Million Odd Numbered Records */
/*  with 5n+1 vaules repeated 5 times */
data BigData;
     do ID=1 to 2e7 by 2;
          output;
          if mod(id,5)=1 then do; output; output; output; output; end;
     end;
run;

/* Five unsorted targets - all in data */
data IDData;
ID=05555555; output;
ID=00044561; output;
ID=16555555; output;
ID=19933911; output;
ID=00000001; output;
RUN;

/* Unduplicate targets to remove possible duplicates */
PROC SORT DATA=IDData FORCE NODUPKEY;
     BY ID;
RUN;

DATA Found(DROP=_:) Not_Found(KEEP=ID);
     SET IDData;

     _low=1; _high=nobs; _direction=0; _tries=0;

     DO WHILE(_high>_low AND ID NE _target AND _tries<(LOG2(nobs)+2));
          _tries+1;

          IF _direction<0 THEN _high=(_mid-1)<>1; ELSE
          IF _direction>0 THEN _low=(_mid+1)><nobs;
          _mid=int((_low+_high)/2);

          SET BigData(KEEP=ID RENAME=(ID=_target))
              point=_mid nobs=nobs;
          _direction= SUM(((ID<_target)*(-1)),((ID>_target)*1));
     END;

     SET BigData(KEEP=ID RENAME=(ID=_target)) POINT=_mid;
     IF ID NE _target THEN OUTPUT Not_Found;

     DO WHILE(ID=_target AND _mid>0);
          _mid=_mid-1;
          IF _mid>0 THEN
            SET BigData(KEEP=ID RENAME=(ID=_target)) POINT=_mid;
     END;
     _target=ID;
     DO WHILE(ID=_target);
          _mid=_mid+1;
          SET BigData(RENAME=(ID=_target)) POINT=_mid;
          IF ID=_target THEN OUTPUT Found; ELSE LEAVE;
     END;
RUN;
```

This example lookup takes 0.09 seconds on an XP system, the equivalent SQL takes 5.73 seconds.

CONCLUSION

Using a data step binary tree search has been shown to quickly lookup a limited number of target records from a sorted dataset, without the need for an index. PROC SQL and DATA step merge in some cases achieve similar gains, but the DATA step binary tree lookup method is not limited by system memory constraints. This binary tree search could be easily generalizing into a macro or called from a program library with %INCLUDE. For environments where large sorted but unindexed files are regularly accessed, the savings in processing time and cost could be significant.

REFERENCES

SAS 9.1.3 XP Platform
SAS Institute Inc., Cary, NC

SAS OnlineDoc 9.1.3 for the Web
SAS Institute Inc., Cary, NC

SAS-L Discussion Group Thread Subject: *OT: Binary Tree Lookups for Dummies*
http://groups.google.com/group/comp.soft-sys.sas/tree/browse_frm/thread/2562af8d298a9fc2/

TS553 *SQL Joins -- The Long and The Short of It*, by Paul Kent available on the SAS web site

ACKNOWELDGMENTS

The author would like to thank Richard A. DeVenezia for his valuable comments on optimization of this method, and Paul Dorfman for his erudite explanation of the effects of key distribution, and explaining the maximal probes made by the search routine. Ken Borowiak graciously volunteered editorial assistance and made important comments on SQL merges using the SORTEDBY option.

RECOMMENDED READING

SAS-L@LISTSERV.UGA.EDU
http://listserv.uga.edu/archives/sas-l.html

Documentation for SAS Products and Solutions
http://support.sas.com/documentation/onlinedoc/index.html

CONTACT INFORMATION

Any errors or omissions are the fault of the author. Your comments and questions are valued and encouraged. Feel free to contact the author at:

Paul Choate, Senior Programmer Analyst
California Department of Developmental Services
1600 9th Street, Sacramento, CA 95818
Work Phone: (916) 654-2160
E-mail: pchoate@dds.ca.gov

Join the SAS-L @ LISTSERV.UGA.EDU or Google Groups!