Transaction Tracing

Liang Xie
Marketing Analytics
Reliant Energy

October 24, 2008

Abstract

In this paper, we show a very interesting case in data manipulation I came across when I was in Countrywide Home Loans.

In mortgage industry, a customer may keep refinancing every several years, taking advantage of bank discounts, mortgage interest rate cut as well as other significant changes in their lives. Since they refinancing from existing account with the same lender, they are called portfolio accounts. In one project, we want to trace all mortgages that are refinanced from a portfolio account all the way back to the very first one. Sometimes a portfolio account refinance into two liens, and those derived second lien account will be excluded from tracing since in the reporting system all subsequent refinancing activities will be credited to the first Lien only. This reporting system hence keep two identical records of refinancing accounts pairs. However, stand alone Second Lien won't be excluded. This impose no further difficulty here because only derived Second Lien is flagged in our system. The whole purpose is to make sure we only trace the unique household in our portfolio.

For instance, Account 333 is a refinanced 1st lien mortgage from account 222 which again is a refinancing result from joint account 111 and 112 where 112 is a Second Lien. Hence in our system we will see Account 222 corresponds to two accounts: 111 and 112. Tracing for reported pair 112–222 is excluded. But more complicate, Account 111 is a refinanced Account from previously existing account 100 which is the very original account, usually a Purchase mortgage. The tracing will stop at this account since it is the true start of all subsequent Refinanced accounts.

This data manipulation can be easily done via recursion, but SAS doesn't support recursion in data step, hence making this problem tricky. In this paper, I demonstrate how to handle this sort of problem by two means. The first approach take advantage of Hash table featuring in SAS® v9, while the second one utilized disk random access that is available in all SAS® version.

†

1 Introduction

In one project at Countrywide Home Loans to address " Frequency, average Refi Timing and Discount Points" question from Executive Management Team, we need to trace all mortgage refinancing transactions at Account level. The senior managers want to know how many refinancing transactions a customer did with Countrywide Home Loans (CHL) since his first business relationship with us, as well as the timing and monetary results. Once a mortgage account is established with CHL, it will be called Portfolio Account since it forms the base of the asset portfolio CHL holds. Each refinancing transaction from an existing portfolio account will have an transaction record in our database system with both accounts' numbers are linked together. For some particular purpose, some special transactions will have two almost identical records in our system differing in only one field that indicats it is a special transaction. Specifically, the senior managers want:

1. Trace all refinancing transactions associated with protfolio accounts all the way back to the very first one which is flagged in the system;
2. Special transactions need not be traced but need to be listed in the final table;
3. Assign a unique identifier to the group of linked transactions so other measurements pertained to each group can be easily calculated.

A typical data structure can be seen from the Appendix, which a simplified hypothetical version of the data on server. We maintain a database for our direct sales division that records all refinancing history for each pair of new loan and old loan. When a portfolio account does refinancing, there are various scenarios to be considered:

1. An existing first lien account refinances into a new first lien account. This is the simplest case;
2. A portfolio account refinances into two accounts, one conforming first lien and one piggy heloc. This is common case during refinancing boom from 2003 to 2005 when customers' houses had experienced enough property value appreciation and they did a Cash Out or Rate and Term refinancing, making the new principle balance surpassed conforming loan limit, which was 417000 till this fall;
3. Two portfolio accounts consolidated into one lien. Customers usually take advantage of lower interest rate using this method.
4. Two portfolio accounts refinance into two accounts of the same lien.

Despite various refinancing cases, we usually link only the primary first lien account to the new account. But in the last case, each lien will be linked to the corresponding new lien.

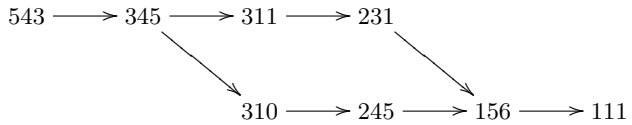
There are tricky difficulties associated with this data manipulation request.

1. Complete Refinancing history with specific business divisions is required;
2. We don't know *a prior* how many refinacing an account has done during a specific time period, therefore the program needs to repeatedly scan the transaction table until no match is found;
3. In case two liens consolidate into one lien, and the two liens were refinanced liens from a previous refinancing, we need to track the two refinancing routes separately and combined them together so we have a complete history of all relavant measurements;
4. SAS doesn't support recursion in DATA STEP, hence making this data manipulation more tricky;

†

2 Formalized Problem

In order to map a formal structure for this problem, we consider the following example. In case two liens consolidate into one line or two liens refinance separately into two new liens, we keep two records respectively. A sample route can be found below.



The complete transaction history for Loan Num 543 includes loans from 345 all the way to loan 111 which is the first transaction, which consists two routes:

1. $543 -> 345 -> 311 -> 231 -> 156 -> 111$;
2. $543 -> 345 -> 310 -> 245 -> 156 -> 111$;
3. Complete history $543 -> 345 -> 311 -> 231 -> 156 -> 111; -> 310 -> 245 -> 156 -> 111$. We could set a flag at the end of each route so the in further analysis, we are able to separate this history into individual ones.

This is exactly an example of Traversing Tree where we explore every node on the tree that connected to the root. In this particular case, we are going to reach as far as possible along each branch before backtracking to the root, which is some chosen node on the branch, and this falls into the application of Deep First Search (DFS) on a Directed Graph.

A DFS is a search algorithm that first expand its first child node and go deeper and deeper until rearch the end node or the node being searched for. Then it tracks back along the branch to the immediate upper level node where it hasn't explored all the child node of that node, and so on so forth. This makes recursion a natural solution to this type of problem. Unfortunately, SAS doesn't support recursion in DATA STEP, so we have to set up a stack that is Last In Last Out to keep information for backtracking. The linked record just saved will be output to the final

table and cleared from the stack so that new linked record will be saved where the most recent upper level node is.

DFS has linear time complexity in the number of its nodes and edges it has to traverse, and space complexity soly depends on the length of one simple path, i.e the length of total backtracking, which makes it appealing. In this directed graph application, we have even a simpler DFS because.

A non-recursion psedo-code for DFS is as the following:

```
dfs(graph G)
{
  list L = empty
  tree T = empty
  choose a starting vertex x
  search(x)
  while(L is not empty)
  {
    remove edge (v, w) from end of L
    if w not yet visited
    {
      add (v, w) to T
      search(w)
    }
  }
}

search(vertex v)
{
  visit v
  for each edge (v, w)
    add edge (v, w) to the end of L
}
```

In the next two sections, we are going to show concrete SAS code following the logic of psedo-code, even though it is in some degree obvious.

†

3 Hash Table Solution

Hash table available to SAS® v9 fits naturally into the above psedo-code. We could literally map the psedo-code directly to the SAS code. The programmers are now being able to add and remove components in the table on the fly and no pre-determined table size is required. Corresponding to the psedo-code, we only need to set up one list using the hash table because the Tree will be directly output to the final data set sequentially. For the "choose a starting vertex" statement, we just need to set the transaction table because each account of interest is naturally a vertex, or say, node. The Search part need some pre-process of the master table. Because it is necessary to add all contiengent edges of some vertex x into the stack list, we need to keep a table for all the positions of the linked records in the master table. A simple and effective way is just sorting the

master table by the target nodes and output a table contains the position range of each unique node. So that each time we need to search for all edges of certain node, we first locate the position range of this node in the master table, and just use random access method "set table point=" to extract data from the master table.

The search part is coded within a macro to save coding lines:

```
%macro search(node);
      rcm=master.find(key:&node);
    if rcm=0 then do;
       do i=first to last;
          set main(drop=ID  rename=(FROMNODE=FN0  TONODE=TN0))  point=i;
      rcl=list.add();
      k+1;
     end;
    end;
%mend;
```

In above code, `master.find` locates the position ranges, if a node is found to be existing in the master table, all linked records were added to the stack list of hash "list".

Besides, in order to perform `'if w not yet visited'`, we set up another hash table "hist" to store all nodes output from this root. This hash table need to be cleaned after all branches and nodes are explored. We simply delete this hash table at the end because we set the keys of this hash table to be the output pair linked records, hence it is not easy to use remove method which relies on key to locate component in the hash table. Of course, we need to initiate this hash table for every starting node.

The main logic is implemented in the following code:

```
do while(k>0);                      .......................(1)
       rcl=list.find(key:k);
       FROMNODE=FN0; TONODE=TN0;
       rch=hist.check(key:FROMNODE, key:TONODE);  .........(2)
       if rch^=0 then do;           .......................(3)
          keep ID  FROMNODE  TONODE  STARTFLAG  BRANCH;
          output;
          FN1=FROMNODE;  TN1=TONODE;
          rch=hist.add();           .......................(4)
       end;
       rcl=list.remove();           .......................(5)
       k=k-1;
       %search(FROMNODE);           .......................(6)
end;
rch=hist.delete();
```

There needs some explanations for the steps.

1. Step 1 used a counter to track the stack list. We can alternatively use *object*.`num_items` method. There is no practical difference, but that this way can be directly mapped onto non-hash table solution.
2. Step 2 implements checking `'if w not yet visited'` . Just like the example above, we want to ouput only one record no matter

whichever the spanning route. Another role of this check is to avoid infinit looping in general case, but since in this case it is directed graph without close loop, we won't encount such problem therefore it is simply to eliminate redundency.

3. Step 3 outputs new record only if it is not yet output.
4. Step 4 adds the new output to the history list for check in further steps.
5. Step 5 excutes the Last In Last Out operation. This will also move the pointer in the hash table to the most recent parent node of the vertex just output.
6. Step 6 begins another round of search from this most recent parent node. Of course, after looping when we exhaust all branches, the history list will be deleted.

We test on two hypothetical datasets. The first one is the example shown above where we demonstrated the full path when starting at vertex 111. SAS log output shows:

```
FROMNODE=156
NOTE: The data set WORK.LIST has 2 observations and 2 variables.
k=2 rcm=0
list  k=2 FN0=245 TN0=156 rch=-2147450842
list  k=2 FN0=310 TN0=245 rch=-2147450842
list  k=2 FN0=345 TN0=310 rch=-2147450842
list  k=2 FN0=543 TN0=345 rch=-2147450842
list  k=1 FN0=231 TN0=156 rch=-2147450842
list  k=1 FN0=311 TN0=231 rch=-2147450842
list  k=1 FN0=345 TN0=311 rch=-2147450842
list  k=1 FN0=543 TN0=345 rch=0
list  k=0 FN0=543 TN0=345 rch=0
NOTE: There were 7 observations read from the data set WORK.MAIN_U.
NOTE: There were 1 observations read from the data set WORK.MAIN2.
NOTE: The data set WORK.TRACE has 8 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

The next example is more complicated in that it is longer, has more splits and includs all types of edges from spanning tree that are legible for this case, i.e. Tree Edge, Cross Edge, Forward Edge. The only one type that is not legible is Back Edge because every new loan will have a unique new account number so that the direction won't go back at all.

```
FROMNODE=773501327
NOTE: The data set WORK.LIST has 2 observations and 2 variables.
k=2 rcm=0
list  k=2 ID=T FN0=773500791 TN0=773501327 rch=-2147450842
list  k=1 ID=T FN0=773501293 TN0=773501327 rch=-2147450842
list  k=1 ID=T FN0=773500800 TN0=773501293 rch=-2147450842
list  k=2 ID=T FN0=773500814 TN0=773500800 rch=-2147450842
list  k=3 ID=T FN0=773500813 TN0=773500814 rch=-2147450842
```

```
list  k=3 ID=T FN0=773500830 TN0=773500813 rch=-2147450842
list  k=3 ID=T FN0=773500833 TN0=773500830 rch=-2147450842
list  k=4 ID=T FN0=773500845 TN0=773500833 rch=-2147450842
list  k=4 ID=T FN0=773500879 TN0=773500845 rch=-2147450842
list  k=5 ID=T FN0=773500885 TN0=773500879 rch=-2147450842
list  k=4 ID=T FN0=773500403 TN0=773500879 rch=-2147450842
list  k=3 ID=T FN0=773500845 TN0=773500833 rch=0
list  k=3 ID=T FN0=773500879 TN0=773500845 rch=0
list  k=4 ID=T FN0=773500885 TN0=773500879 rch=0
list  k=3 ID=T FN0=773500403 TN0=773500879 rch=0
list  k=2 ID=T FN0=773500813 TN0=773500814 rch=0
list  k=2 ID=T FN0=773500830 TN0=773500813 rch=0
list  k=2 ID=T FN0=773500833 TN0=773500830 rch=0
list  k=3 ID=T FN0=773500845 TN0=773500833 rch=0
list  k=3 ID=T FN0=773500879 TN0=773500845 rch=0
list  k=4 ID=T FN0=773500885 TN0=773500879 rch=0
list  k=3 ID=T FN0=773500403 TN0=773500879 rch=0
list  k=2 ID=T FN0=773500845 TN0=773500833 rch=0
list  k=2 ID=T FN0=773500879 TN0=773500845 rch=0
list  k=3 ID=T FN0=773500885 TN0=773500879 rch=0
list  k=2 ID=T FN0=773500403 TN0=773500879 rch=0
list  k=1 ID=T FN0=773500807 TN0=773500800 rch=-2147450842
list  k=1 ID=T FN0=773500830 TN0=773500807 rch=-2147450842
list  k=1 ID=T FN0=773500833 TN0=773500830 rch=0
list  k=2 ID=T FN0=773500845 TN0=773500833 rch=0
list  k=2 ID=T FN0=773500879 TN0=773500845 rch=0
list  k=3 ID=T FN0=773500885 TN0=773500879 rch=0
list  k=2 ID=T FN0=773500403 TN0=773500879 rch=0
list  k=1 ID=T FN0=773500845 TN0=773500833 rch=0
list  k=1 ID=T FN0=773500879 TN0=773500845 rch=0
list  k=2 ID=T FN0=773500885 TN0=773500879 rch=0
list  k=1 ID=T FN0=773500403 TN0=773500879 rch=0
FROMNODE=773500885
```

From the log we can clearly see how the program works along all the branches and only those tracking with `'rch=0'` will be output.

†

4 SAS® v8 solution

SAS® v8 and older versions don't have hash table capability, which makes programming more difficult and less efficint. The lack of hash table not only eliminate quick search but also modify on the fly capability, which are the major obstable to overcome. Think of a hash table as a data set in the memory rather than on the disk, there is a way to mimic the operation only with much less efficiency. The key is, instead of one overall DATA STEP as above, that using multiple steps to operate the 3 tables we need to modify on the fly, therefore a MACRO program is necessary.

Examining code in previous section, we figured that modification on the fly only takes place in two steps. The first one is to remove the last

record for stack list in the main logic part, and the other one is sweeping history list at the very end. Therefore we can break the program into 4 segments of DATA STEPs. The first one will read record from transaction data, and to facilitate further processing, we will transfer the data to a series MACRO variables. And the "search" modular consists of the next step. The "search" conduct two jobs, the first one is to locate the record in main data and the second is to add all affiliated edges to the stack list file. The third segment is composed of the main logic. In this segment, there are three major operations. The first one is search the stack list file to extract the last record and use the record value for further search, and the next operation is to check the history list to see if the found edge has ever been output for this root. If not found, then the program output the edge to the destiny data as well as add it to the history list. The last record from the stack list will be removed in the third segment and the program continue searching using new node value in the last segment.

The counter variable k in the hash table solution should be replaced by a MACRO variable so its value can be used across those segments involving several DATA STEPs.

And of course, at the very end, we simply NULLize the history list table using `modify hist; remove hist;` statements.

Since this one introduced no new ideas but some simple SAS techniques, we are not going through the programs in details here. See Appendix 7.2 on page 11 for the complete program. We, however, do emphasize that remember to add `stop;` statement when using random access method.

†

5 Reference

1. SAS Institute Inc., *SAS 9.1.3 Language Reference: Concepts*, Third Edition. Cary, NC 2005

2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 22.3: Depth-First Search.

†

6 Contact Information

Liang Xie
Reliant Energy
1000 Main Street
Houston, TX 77081

Work phone: 713-497-6908
E-mail: xie1978@yahoo.com
Web: www.linkedin.com/in/liangxie

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ®indicates USA registration. Other brand and product names are trademarks of their respective companies.

†

7 Appendix

7.1 Hash Table Solution Example

```
/************************************************/
data main;
    input  ID $   LENGTHKM   FROMNODE    TONODE   STARTFLAG  Branch;
datalines;
    T       0.43      773501327      773500764        0       0
    tt      0.53      773500885      773500764        1       0
    kk      1.19      773501293      773501327        0       1
    S       0.82      773500791      773501327        1       0
    R       4.53      773500796      773500769        0       0
    P       3.47      773500809      773500796        0       0
    O       1.02      773500821      773500809        0       0
    nn      0.86      773500836      773500821        0       1
    N       1.08      773500836      773500821        0       0
    M       1.14      773500218      773500836        1       0
    hh      0.89      773500807      773500809        0       1
    Q       0.29      773500778      773500796        0       0
    ll      0.18      773501344      773500778        0       1
    L       0.62      773501344      773500778        0       0
    K       0.09      773501293      773501344        0       0
    J       1.78      773500800      773501293        0       0
    H       0.43      773500807      773500800        0       0
    ff      0.40      773500830      773500807        0       1
    I       0.31      773500814      773500800        0       0
    gg      0.42      773500813      773500814        0       1
    G       0.56      773500813      773500814        0       0
    F       0.17      773500830      773500813        0       0
    E       2.10      773500833      773500830        0       0
    dd      0.14      773500845      773500833        0       1
    D       0.44      773500845      773500833        0       0
    C       1.36      773500879      773500845        0       0
    B       3.42      773500403      773500879        1       0
    A       3.79      773500885      773500879        1       0
;
run;

proc sort data=main; by TONODE; run;

data main_u;
        set main;  by TONODE;
```

```
retain first  last  0;
if first.TONODE then do;
  first=_n_;
end;
if last.TONODE then do;
          last=_n_; TN=TONODE;
  keep TN  first  last;
  output;
end;
run;

data main2;  set main; where TONODE=773500764; run;
/***********************************************/

%macro search(node);
       rcm=master.find(key:&node);
     if rcm=0 then do;
        do i=first to last;
          set main(drop=ID  rename=(FROMNODE=FN0  TONODE=TN0))  point=i;
            k+1;
            rcl=list.add();
      end;
     end;
%mend;

%macro init_hist;
       declare hash  hist();
       hist.defineKey('FN1', 'TN1');
       hist.defineData('FN1', 'TN1');
       hist.defineDone();
       call missing(FN1, TN1);
%mend;

proc printto log='c:\dfs_output2.txt'; run;
data trace;
     if _n_=1 then do;
        declare hash list();
        list.defineKey('k');
        list.defineData('FN0', 'TN0');
    list.defineDone();
    call missing(FN0, TN0);

    declare hash  master();
    master.defineKey('TN');
    master.defineData('TN', 'first', 'last');
    master.defineDone();
    do until (eof);
           set main_u  end=eof;
     rcm=master.add();
    end;
```

```
 end;
 %init_hist;
 set Main2;
 put FROMNODE=;
 k=0;
     /* %search(x) */
     %search(FROMNODE);
 /* search (x) finished */
     keep ID  FROMNODE  TONODE  STARTFLAG  BRANCH;
     output;
 rcl=list.output(dataset: "work.list");
   put k= rcm=  ;
   do while( k>0);    /* or use:  (list.num_items>0) */
   rcl=list.find(key:k);
   FROMNODE=FN0; TONODE=TN0;
   rch=hist.check(key:FROMNODE, key:TONODE);
       put "list  " k=  ID=  FN0=  TN0=  rch=;
   if rch^=0 then do;
          keep ID  FROMNODE  TONODE  STARTFLAG  BRANCH;
      output  trace;
          FN1=FROMNODE;  TN1=TONODE;
      rch=hist.add();
   end;
       rcl=list.remove();
       k=k-1;
       %search(FROMNODE);
   end;
 rch=hist.delete();
run;
proc printto; run;
```

7.2 File Random Access Solution

```
/*********************************************/
data main;
     input ID  FROMNODE  TONODE  STARTFLAG  BRANCH;
datalines;
4   543  345  0  0
2   345  311  0  0
1   311  231  0  0
3   345  310  0  0
5   231  156  0  0
6   245  156  0  0
7   156  111  1  0
8   310  245  0  0
;
run;
data main2;
     input ID  FROMNODE  TONODE  STARTFLAG  BRANCH;
```

```
datalines;
7   156  111  1  0
;
run;
proc sort data=main; by TONODE; run;
data main_u;
        set main;  by TONODE;
        retain first  last  0;
        if first.TONODE then do;
           first=_n_;
        end;
        if last.TONODE then do;
           last=_n_; TN=TONODE;
           keep TN  first  last;
           output;
        end;
run;

data main_u2(index=(TN)); set main_u; run;

/*****************************/

%macro main_job(dsn, main_dsn, main_idx);
/* initialize list & hist tables as well as output table*/
data list; FN0=.;  TN0=.; output; run;
data list; modify list; remove list; run;
data hist; FN1=.;  TN1=.; output; run;
data hist; modify hist; remove hist; run;

data trace;
       ID=.; FROMNODE=.; TONODE=.; STARTFLAG=.; BRANCH=.;
       output;
run;
data trace; modify trace; remove trace; run;

%global  k;
%let dsid=%sysfunc(open(&dsn));
%let ntotal=%sysfunc(attrn(&dsid, nobs));
%let dsid=%sysfunc(close(&dsid));

%put ------- &ntotal -----------;

%do obs=1 %to &ntotal;
    data _null_;
         j=&obs;
         %let jj=j;
         set &dsn  point=&jj;
         call symput('FROMNODE', FROMNODE); call symput('TONODE', TONODE);
         call symput('ID', ID);  call symput('STARTFLAG', STARTFLAG);
         call symput('Branch', Branch);
```

```
     stop;
run;

%let k=0;   %put FROMNODE= &FROMNODE   k= &k;

/* Search begin */
data tmp_list; FN0=.;  TN0=.; output; run;
data tmp_list; modify tmp_list; remove tmp_list; run;
%put  Begin Searching;
data tmp_list(keep=FN0  TN0);
     kk=0;
     TN=&FROMNODE;
     set &main_idx  key=TN;
     if _iorc_=%sysrc(_sok) then do;
        do i=first to last;
           set &main_dsn.(rename=(FROMNODE=FN0  TONODE=TN0) drop=ID)
               point=i;
           kk+1;
           output tmp_list;
        end;
     end;
     else do;
        _error_=0;
     end;
     call symput('kk', kk);
     stop;
 run;
 %let k=%eval(&k+&kk);
 proc append  base=list  data=tmp_list; run;
 %put End Searching  k= &k;
 /* Search ends */
 data tmp_trace;
      ID=&ID; FROMNODE=&FROMNODE; TONODE=&TONODE;
      STARTFLAG=&STARTFLAG; BRANCH=&BRANCH;
      output;
 run;
 %if &obs=1 %then %do;
 data trace;
      set tmp_trace;
 run;
 %end;
 %else %do;
      proc append base=trace  data=tmp_trace; run;
 %end;

 %do %while (&k>0);
     %put k= &k;
     data _null_;  set list;  put  _n_=  FN0=  TN0=; run;
     data _null_;
         set list  point=numobs  nobs=numobs;
```

```
    call symput('FN0', FN0);  call symput('TN0', TN0);
    stop;
run;
%put FN0=&FN0  TN0=&TN0;
data _null_;
     rch=-1;
     do while(^eof_hist);
        set hist  end=eof_hist;
        if  (FN1 eq &FN0 & TN1 eq &TN0) then rch=0;
     end;
     call symput('rch', rch);
run;
%put rch= &rch  ;
%if &rch^=0 %then  %do;
    data tmp_trace;
         FROMNODE=&FN0; TONODE=&TN0; ID=&ID;
         STARTFLAG=&STARTFLAG; BRANCH=&BRANCH;
         output; stop;
    run;
    proc append  base=trace  data=tmp_trace; run;
    data tmp_hist;
         FN1=&FN0; TN1=&TN0;
         output; stop;
    run;
    proc append  base=hist  data=tmp_hist; run;
    data _null_;  set hist;  put  'hist'  FN1=  TN1=; run;
%end;
data list;
     set list  end=eof_list;
     if eof_list then delete;
run;

%let k=%eval(&k-1);

/* Search begin */
data tmp_list; FN0=.;  TN0=.; output; run;
data tmp_list; modify tmp_list; remove tmp_list; run;
%put  Begin Searching;
data tmp_list(keep=FN0  TN0);
     kk=0;
     TN=&FN0;
     set &main_idx  key=TN;
     if _iorc_=%sysrc(_sok) then do;
        do i=first to last;
           set &main_dsn.(rename=(FROMNODE=FN0  TONODE=TN0) drop=ID)
               point=i;
           kk+1;
           output tmp_list;
        end;
     end;
```

```
            else do;
                 _error_=0;
            end;
            call symput('kk', kk);
            stop;
        run;
        %let k=%eval(&k+&kk);
        proc append  base=list  data=tmp_list; run;
        data _null_;  set list;  put  _n_=  FNO=  TNO=; run;
        %put End Searching  k= &k;
        /* Search ends */
     %end;
     %put  begin dumping history table at step  &obs;
     data hist;
         do until(eof);
            modify hist  end=eof;
            remove hist;
         end;
     run;
%end;
%mend;

options  nonotes;
proc printto  log='c:\dfs_v8.txt'; run;
%main_job(main2, main, main_u2);
proc printto log=log;  run;
options  notes;
```