Paper 951-30

Text on My Remote Control: An Experiment in Shortest Distance Using SAS® Software

Rick Langston, SAS Institute Inc., Cary, NC

ABSTRACT

My DVD remote control has an interesting approach to adding text, and I thought it would be interesting to design a program to give me the optimal paths for entering text with the least amount of button-pushing. This program is primarily an exercise in shortest-distance algorithms using the DATA step.

INTRODUCTION

My DVD recorder allows you to enter titles for individual programs that you're saving onto a DVD. For example, "Thanksgiving 2004" can be added to a menu of programs so that you can easily click the menu item to begin playing the video of that event.

The text entry is all done through a hand-held remote, not a computer keyboard. It's somewhat tedious to enter text using the remote, because a combination of buttons is necessary for entering any character.

Figure 1 shows the character mapping that is seen on the screen.

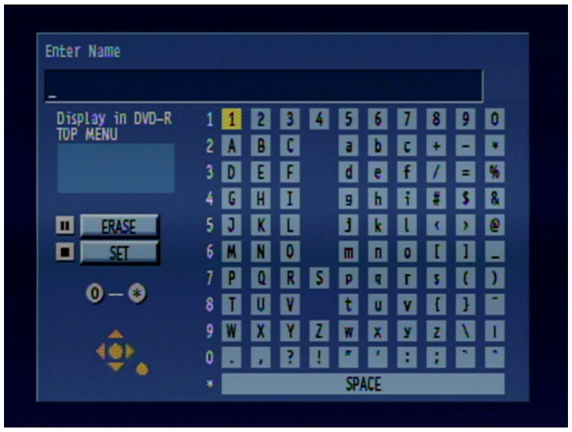

Figure 1. Character Mapping on the Screen

The possible buttons that you can press on the remote are 0 through 9, *, Up/Down/Left or Right arrow, and Enter. To get to a character, enter one of the digits 0 through 9 to get to the row for that character, then press the Right arrow to get to the character. If you press the Right arrow until you reach the last column and then press it again, that arrow will go to an alternative location, and the next Right arrow takes you back to the first column. The Left arrow works the same way. The Up and Down arrows also work this way in changing rows. Notice that rows 2, 3, 4, 5, 6, and 8 do not have a character in column 4, so that the Right arrow entered on column 3 will jump to column 5. A space is obtained by entering an asterisk (*).

I thought it would be interesting to write a SAS program that would determine the least number of keystrokes for entering the characters. This would be an exercise in short-distance algorithms using the DATA step.

First, I read in the layout so that ENTRYCHAR is the key character and DISPCHARS is the set of possible display characters.

```
data temp;
     input entrychar $char1. @3 dispchars $char10.;
     datalines4;
1 1234567890
2 ABC abc+-*
3 DEF def/=%
4 GHI ghi#$&
5 JKL jkl<>@
6 MNO mno[]_
7 PQRSpqrs()
8 TUV tuv{}
9 WXYZwxyz\|
0 .,?!"':;`^
;;;;
```

Next, create the more useful CHARS data set, which has one observation per possible character. DISP is the displayed character and ROW and COL specify the position. ENTRYCHAR is the character that's used in order to get to col 1 of the row.

```
data chars; set temp end=eof;
     row+1;
     length disp $1;
     do col=0 to 9;
        disp=substr(dispchars,col+1,1);
        if disp^=' ' then output;
        end;
     keep disp entrychar row col;
     run;
```

Now, create a list of transitions between each possible character.

```
data trans;
     do i=1 to nobs;
        set chars(rename=(disp=disp1 entrychar=entrychar1 row=row1 col=col1))
            point=i nobs=nobs;
        do j=1 to nobs;
           first=(j=1);
           set chars(rename=(disp=disp2 entrychar=entrychar2 row=row2 col=col2))
               point=j nobs=nobs;
           output;
           end;
        end;
     stop;
     keep disp1 entrychar1 row1 col1 disp2 entrychar2 row2 col2 first;
     run;
```

Note: The following program uses L for left arrow, R for right arrow, U for up arrow, D for down arrow, and E for enter.

Here's how this works. You find the necessary moves to get to the same row by using the TO_SAME_ROW link, which determines if there are fewer moves to go up or down to get to the correct row. After you're in the same row, use the SAME_ROW link to find out if it's shorter to move to the left or to the right to get to the next character. You might find that entering a digit to expedite moving to col 0 of a row, plus the left or right moves to get to the correct character, consists of fewer moves. For example, consider the string 'Pack'. You get to 'P' by 7E. To get to 'a', you can press D six times or press U five times. Either way, you are on col 0. Now, you can press L seven times or press R three times. When using the cursor, the fastest way to get from 'P' to 'a' is UUUUURRR (eight moves). However, if you use the fast-path 2 to get to row 2 and press R three times, the move is 2RRR, which is only four moves instead of eight. Blanks are somewhat special because to get a blank you always enter the asterisk (*) as a fast-path.

```
data moves; set trans;
     length moves $20;
     moves=' '; origcol1=col1;

     /*-----determine moves to same row as disp2-----*/
     if row1^=row2 then link to_same_row;

     /*-----now get the LEFT or RIGHT moves-----*/
     link same_row;

     /* If you change rows, you might find that you can get
        there faster by entering the "entry character"
        (1-9) to fast-path to the row. Test to see
        if the number of moves would be less using
        fast-path. If it is, use that instead. */

     if row1^=row2 then do;
        oneway=moves;
        moves=entrychar2;
```

```
         col1=0;
         link same_row;
         if lengthn(moves)>lengthn(oneway)
            then moves=oneway;
         end;

      /*-----emit our combination-----*/
      output;

      /* For the first occurrence of each disp1
         character, set disp2 to blank and
         indicate the move is simply a '*', and
         emit an observation for that. Also,
         produce an observation where disp1 is
         a blank and disp2 is the disp1 char.
         In this case, always use the fast-path
         approach to get to the row. */

      if first;
      disp2=' '; moves='*'; output;
      disp2=disp1; disp1=' ';
      row2=row1; col2=origcol1;
      moves=entrychar1;
      col1=0;
      link same_row;
      output;
      keep disp1 disp2 moves;
      return;

   /* link routine to get shortest number of moves to
      the same row */

   to_same_row:;
      length movesd movesu $11;

      /* adjust to not use column 3 because not all
         slots are occupied there */

      col3skip=(col1=3 and col2=3);
      if col1=3 and ^col3skip then do;
         length repos $1;
         if col2 lt col1 then do;
            repos='L';
            col1=col1-1;
            end;
         else do;
            repos='R';
            col1+1;
            end;
         moves=trimn(moves)||repos;
         end;
      rove=row1;
      movesd=' ';

      /*=====check first for move count using DOWN key=====*/

      /* Step down the rows. When you get to the last row,
         add an extra D move and then jump back to row 1.
         Stop when you get to the row2 value. */

      do while(rove^=row2);
         rove=rove+1;

         /*-----handle the special skipped chars-----*/
         if col3skip and (2 le rove le 5 or rove eq 8)
            then continue;

         movesd=trimn(movesd)||'D';
         if rove=11 then do;
            movesd=trimn(movesd)||'D';
            rove=1;
            end;
         end;

      /*=====check now for move count using UP key=====*/

      /* Step up the rows. When you get to the first row,
         add an extra U move and then jump back to the
         last row. Stop when you get to the row2 value. */

      rove=row1;
      movesu=' ';
      do while(rove^=row2);
         rove=rove-1;

         /*-----handle the special skipped chars-----*/
         if col3skip and (2<=rove<=5 or rove=8)
            then continue;

         movesu=trimn(movesu)||'U';
         if rove=0 then do;
            movesu=trimn(movesu)||'U';
            rove=11;
            end;
         end;

      /*-----select the smallest number of moves-----*/
      if lengthn(movesu) lt lengthn(movesd)
         then moves=trimn(moves)||movesu;
      else moves=trimn(moves)||movesd;

      return;
same_row:;
      length movesl movesr $9;
      movesl=' ';

      /*-----no character in col3 on certain rows-----*/
      if 2 le row2 le 6 or row2=8 then skip=3;
      else skip=.;

      /* collect moves to col2 using the
         LEFT key only */

      rove=col1;
      do while(rove^=col2);
         movesl=trimn(movesl)||'L';
         rove=rove-1;
         if rove=skip then rove=rove-1;
         if rove=-1 then do;
            movesl=trimn(movesl)||'L';
            rove=9;
            end;
         end;

      /* collect moves to col2 using the
         RIGHT key only */

      rove=col1;
```

```
        movesr=' ';
        do while(rove^=col2);
           movesr=trimn(movesr)||'R';
           rove+1;
           if rove=skip then rove+1;
           if rove=10 then do;
              movesr=trimn(movesr)||'R';
              rove=0;
              end;
           end;

        /*-----select the smallest number of moves to col2-----*/
        if lengthn(movesl) lt lengthn(movesr)
           then moves=trimn(moves)||movesl;
        else moves=trimn(moves)||movesr;
        return;
        run;
```

Now that you have all the moves, you can create a format where the start value is the **from** character and the **to** character. The label is the list of moves.

```
   data cntlin; set moves(keep=disp1 disp2 moves rename=(moves=label));
        retain fmtname '$MOVES';
        start=disp1||disp2;
        run;
   proc format cntlin=cntlin; run;
```

The following program tests the format by reading in test strings and producing the complete move set for each string.

```
   data _null_; length string $30;
        input @1 string &$;
        put string=;
        l=length(string);
        length c1 c2 $1;
        c1=' ';
        do i=1 to l;
           c2=substr(string,i,1);
           moves=put(c1||c2,$moves.);
           moveslen=lengthn(moves); /* moveslen may be 0  */
           c1=c2;
           put moves $varying200. moveslen 'E' @;
           end;
        datalines4;
   Hello There
   hello there
   Julie #1
   Julie #2
   S4m
   Smash
   Pack
   ;;;;
```

Here are the results.

```
string=Hello There
4REURRREDDREEDE*E8E4RRRREUEDDDDRE3RRRRE
string=hello there
4RRRREUEDDREEDE*E8RRRE4RRRREUEDDDDRE3RRRRE
string=Julie #1
5E8RRRREUUUREUEULE*E4LLLLE1E
string=Julie #2
5E8RRRREUUUREUEULE*E4LLLLE1RE
string=S4m
7RRREUUE6RRRE
string=Smash
7RRRERUE2RRRE7LLLLE4RRRRE
string=Pack
7E2RRRERREDDDLE
```

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author:

Rick Langston
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Phone: (919) 677-8000
Rick.Langston@sas.com