Subject: Re: how to sort data based on other sorted data
Posted by Tom McGlynn on Fri, 11 Jan 2008 17:53:21 GMT
View Forum Message <> Reply to Message

On Jan 10, 3:51 pm, placebo <willie.mad...@gmail.com> wrote:
>> try Craig Markwardt's multisort
>
>> http://astrog.physics.wisc.edu/~craigm/idl/arrays.html#MULTI SORT
>
> Brian,
>
> The multisort method works quite well.
>
...

Multisort works fine and knowing Craig will work very robustly
but I don't think you need to have any limit on
the number of columns. Below is a routine that should
be able to handle an arbitrary number of columns and rows...
I tried it on a 500000 row, 5 column structure -- with each field
a random int between 0 and 29. It ran a bit faster than multisort
(38 s versus 56 s) but gave identical results.
Accommodating different directions for sorting
the different columns should be straightforward. You just need to
invert the index at the appropriate point.

In this case I've had the user organize the input as a structure
where the columns are the sort fields, but they could
just as easily be separate arguments as in multisort. I believe
the columns can be any sortable type.

One thing it does is check if it needs to sort by the next column
or if the sort order is fully determined by the columns already
processed.

The bit I like is the second call to ndistinct which collapses
the maximum values that the key (fullIndex) can have from nrow^2 back
to nrow.
Without something like this the algorithm would either need to
use a big string to accumulate the index (I think that's what
multisort
does) or suffer exponential growth in the indices requiring
limiting nrow^ncol to < 2^64. In principle I think you can sort up
to 2^31.5 rows with this algorithm which is probably enough for most
of us.

Note that I've just put this together this morning, so I wouldn't
be surprised if I've missed some edge cases (e.g., I believe it will

fail with arrays of length 1).

Regards,
Tom McGlynn

```
; This function takes an array and
; returns the number of transitions
; (i.e., x[i] ne x[i-1]) before the current
; element.  It returns a array one shorter
; than the input.  If the original array
; is sorted it returns the number of distinct
; entries before the current entry.

function ndistinct, x
    n = n_elements(x)
    chng = long(x[0:n-2] ne x[1:n-1])
    return, long(total(chng, /cumulative))
end

;+ Main routine
;  Usage: sortIndex = bigsort(userStructure)
;-
function bigsort, x

    nrow = n_elements(x)
    ncol = n_tags(x)

    longlong  = nrow gt 40000 ; rougly sqrt(2^31)
    fullIndex = lonarr(nrow)

    fullIndex[*] = 0

    if (longlong) then begin
        fullIndex = long64(fullIndex)
    endif

    for i=0, ncol-1 do begin

        ; Get the column order for a column.
        index  = sort(x.(i))

 ; Now see if everything is distinguished.
```

```
 cum = ndistinct(x[index].(i))

 ; This creates the index for all columns so far
 if i gt 0 then begin
    fullIndex  = fullIndex*nrow
 endif
 fullIndex[index[1:*]] = fullIndex[index[1:*]] + cum

 ; Sort by the index so far.
       index = sort(fullIndex)
 cum   = ndistinct(fullIndex[index])
 fullIndex[index[0]]        = 0
 fullIndex[index[1:nrow-1]] = cum

 if fullIndex[index[nrow-1]] eq nrow-1 then begin
          ; All rows are distinct, so we're done.
   return, index
 endif
    endfor

    return, index
end
```