Subject: Re: Array intersections
Posted by David Foster on Mon, 09 Mar 1998 08:00:00 GMT
View Forum Message <> Reply to Message

J.D. Smith wrote:
>
> Andy Loughe wrote:
>>
>>> What is the most efficient way (using IDL, of course) to return
>>> the index at which two arrays intersect? e.g.
>>> <snip>
>>
>> I believe the response of David Fanning does not return the "index"
>> at which two arrays intersect, but the actual values themselves
>> (right?).
>> Here is one solution for what you have asked for...
>
> I made these comments about this method last year:
>
>> Check out the NASA library routine match(), which is array based. It uses a
>> flag array and an index array, so the memory overhead is roughly 3 times the
>> sum of the two arrays, but it's pretty fast. It's attached. Note that it takes
>> vectors, so you've go to flatten your array upon input (with reform).
>>
>
>> Just make sure you don't try and use [where_array] with big arrays -- it's an n^2 >algorithm (versus the order n algorithms posted prior). E.g., to compare two >floating 128x128 arrays for overlapping values, the program would create 3 arrays, >each of which takes 1 GB! The routine match() is likely much more efficient for >doing intersections on big arrays. (Unless you have some serious RAM on your >machine).
>
> JD

Some time ago someone from RSI posted these routines for doing
array computations. I have found them to be very fast, and memory
efficient as well. If you need a routine to return the VALUES of
the intersection, you can download FIND_ELEMENTS.PRO at:

ftp://bial8.ucsd.edu pub/software/idl/share/idl_share.tar.gz

This routine is quite fast! It returns the values, not the indices.

Enjoy!

Here are the routines posted by RSI:

---------- SNIP ----------
; _____ _____

```
;
; SETARRAY_UTILS.PRO  [RSI]  9-04-97
;
; Routines posted on newsgroup by RSI. SetIntersection() is much
; faster than Find_Elements(), but it returns the elements
; themselves, not the indices. Also, it ignores duplicate elements.
;
;
; Set operators.  Union, Intersection, and Difference (i.e. return
; members of A that are not in B.)
;
; These functions operate on arrays of positive integers, which need
; not be sorted.  Duplicate elements are ignored, as they have no
; effect on the result.
;
; The empty set is denoted by an array with the first element equal to
-1.
;
; These functions will not be efficient on sparse sets with wide
; ranges, as they trade memory for efficiency.  The HISTOGRAM function
; is used, which creates arrays of size equal to the range of the
; resulting set.

; For example:
;   a = [2,4,6,8]
;   b = [6,1,3,2]
; SetIntersection(a,b) = [ 2, 6]       ; Common elements
; SetUnion(a,b) = [ 1, 2, 3, 4, 6, 8]  ; Elements in either set
; SetDifference(a,b) = [ 4, 8]         ; Elements in A but not in B
; SetIntersection(a,[3,5,7]) = -1 = Null Set

 ;---------------------------------------------------------- --------
FUNCTION SetUnion, a, b
if a[0] lt 0 then return, b    ;A union NULL = a
if b[0] lt 0 then return, a    ;B union NULL = b
return, where(histogram([a,b], OMIN = omin)) + omin ;Return combined set
end


 ;---------------------------------------------------------- --------
FUNCTION SetIntersection, a, b
minab = min(a, MAX=maxa) > min(b, MAX=maxb) ;Only need intersection of
ranges
maxab = maxa < maxb

  ;If either set is empty, or their ranges don't intersect: result =
NULL.
if maxab lt minab or maxab lt 0 then return, -1
```

```
r = where((histogram(a, MIN=minab, MAX=maxab) ne 0) and  $
        (histogram(b, MIN=minab, MAX=maxab) ne 0), count)
if count eq 0 then return, -1 else return, r + minab
end


 ;-------------------------------------------------------- --------
FUNCTION SetDifference, a, b  ; = a and (not b) = elements in A but not
in B
mina = min(a, MAX=maxa)
minb = min(b, MAX=maxb)
if (minb gt maxa) or (maxb lt mina) then return, a ;No intersection...
r = where((histogram(a, MIN=mina, MAX=maxa) ne 0) and $
        (histogram(b, MIN=mina, MAX=maxa) eq 0), count)
if count eq 0 then return, -1 else return, r + mina
end


; -------- Message from RSI to NewsGroup
--------------------------------
;
; A somewhat belated reply to the numerous postings on finding the
; common elements of vectors:

; > Given vectors of the type...
; >
; > a = [1,2,3,4,5]
; > b = [3,4,5,6,7]
; >
; > What is the most efficient way to determine which values that occur
in
; > a also occur in b (i.e., the values [3,4,5] occur in both a and b).
; >

; Below appear three IDL functions that operate on sets represented by
; arrays of positive integers.  The SetIntersection(a,b) function
; returns the common elements, SetUnion(a,b) returns all unique elements
; in both arguments, and SetDifference(a,b) returns the elements
; (members) in a but not in b.

; It is faster than previously published functions, e.g. contain() and
; find_elements().
;
; Hope this helps,

; Research Systems, Inc.
-------------------------------- SNIP --------------------------------
```

--

~~~~~~ ~~~~~~~
David S. Foster Univ. of California, San Diego
Programmer/Analyst Brain Image Analysis Laboratory
foster@bial1.ucsd.edu Department of Psychiatry
(619) 622-5892 8950 Via La Jolla Drive, Suite 2240
La Jolla, CA 92037
~~~~~~ ~~~~~~~