Subject: Array searching efficiency
Posted by Matt Francis on Thu, 10 Feb 2011 23:47:08 GMT
View Forum Message <> Reply to Message

Hi All,

I'm going through and trying to squeeze every last bit of optimum
efficiency out of a code I've been working on. I have a small, very
simple, problem that I'd appreciate some experienced input on.

I have a time series of maps, but the time stamps for them are not
always regular (some maps get missed, some are late etc). I need to
(many millions of times..) find which two maps a given time sits
between, then interpolate between the relevant two maps at some given
location.

The first step is to establish, for a given time, which two maps I
need to interpolate between. I have the times for each map stored in a
single array, in time order. If I can work out the indices in that
array of the two map times then I'm done. This is a simple problem to
solve in any number of ways, but the question is which is the fastest?

I've come up with this:

```
iup = (WHERE(times-time_now GT 0))[0]
ilow = iup-1
```

I'm not sure how WHERE works 'under the hood', but assuming that at
some level it loops over the given array, then optimally once [times -
time_now] becomes positive you would stop searching. Implementing that
kind of algorithm in a WHILE loop is probably slower than using WHERE
though.

Any thoughts or suggestions?

Subject: Re: Array searching efficiency
Posted by David Fanning on Fri, 11 Feb 2011 00:03:15 GMT
View Forum Message <> Reply to Message

Matt Francis writes:

> I'm going through and trying to squeeze every last bit of optimum
> efficiency out of a code I've been working on. I have a small, very
> simple, problem that I'd appreciate some experienced input on.
>
> I have a time series of maps, but the time stamps for them are not
> always regular (some maps get missed, some are late etc). I need to

> (many millions of times..) find which two maps a given time sits
> between, then interpolate between the relevant two maps at some given
> location.
>
> The first step is to establish, for a given time, which two maps I
> need to interpolate between. I have the times for each map stored in a
> single array, in time order. If I can work out the indices in that
> array of the two map times then I'm done. This is a simple problem to
> solve in any number of ways, but the question is which is the fastest?
>
> I've come up with this:
>
> iup = (WHERE(times-time_now GT 0))[0]
> ilow = iup-1
>
> I'm not sure how WHERE works 'under the hood', but assuming that at
> some level it loops over the given array, then optimally once [times -
> time_now] becomes positive you would stop searching. Implementing that
> kind of algorithm in a WHILE loop is probably slower than using WHERE
> though.
>
> Any thoughts or suggestions?

I would shocked if it wasn't several orders of magnitude faster
(for this many iterations) to Histogram your times array with some
appropriate bin size and then ask "which bin" your time_now
was in with Value_Locate.

Cheers,

David

--
David Fanning, Ph.D.
Fanning Software Consulting, Inc.
Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
Sepore ma de ni thui. ("Perhaps thou speakest truth.")

---

## Subject: Re: Array searching efficiency
Posted by penteado on Fri, 11 Feb 2011 00:32:10 GMT
View Forum Message <> Reply to Message

On Feb 10, 9:47 pm, Matt Francis <mattjamesfran...@gmail.com> wrote:
> The first step is to establish, for a given time, which two maps I
> need to interpolate between. I have the times for each map stored in a
> single array, in time order. If I can work out the indices in that

> array of the two map times then I'm done. This is a simple problem to
> solve in any number of ways, but the question is which is the fastest?
>
> I've come up with this:
>
> iup = (WHERE(times-time_now GT 0))[0]
> ilow = iup-1
>
> I'm not sure how WHERE works 'under the hood', but assuming that at
> some level it loops over the given array, then optimally once [times -
> time_now] becomes positive you would stop searching. Implementing that
> kind of algorithm in a WHILE loop is probably slower than using WHERE
> though.

where() is certainly *not* optimal for this. It has no reason to stop
searching on the first occurrence. It will keep searching to the end,
as its job is to return every occurrence, and it cannot assume there
will be only one. Besides, you would be doing one call of where() for
each time you are searching for. A very wasteful repeat of searches.

A single call to value_locate does this. Something like

ind=value_locate(times,times_to_search)

will return an array with the index for each value in times_to_search.
Whether the returned index is below or above the value you search for
depends on the ordering of times. See the help on value_locate.

---

Subject: Re: Array searching efficiency
Posted by Matt Francis on Fri, 11 Feb 2011 00:44:44 GMT
View Forum Message <> Reply to Message

> I would shocked if it wasn't several orders of magnitude faster
> (for this many iterations) to Histogram your times array with some
> appropriate bin size and then ask "which bin" your time_now
> was in with Value_Locate.

Thanks David, VALUE_LOCATE is exactly the function I'm looking for.

---

Subject: Re: Array searching efficiency
Posted by Matt Francis on Fri, 11 Feb 2011 00:47:58 GMT
View Forum Message <> Reply to Message

On Feb 11, 11:44 am, Matt Francis <mattjamesfran...@gmail.com> wrote:
>> I would shocked if it wasn't several orders of magnitude faster

>>  (for this many iterations) to Histogram your times array with some
>>  appropriate bin size and then ask "which bin" your time_now
>>  was in with Value_Locate.
>
>  Thanks David, VALUE_LOCATE is exactly the function I'm looking for.

Thanks also to Paulo, who ninja'd my previous post.

---

## Subject: Re: Array searching efficiency
## Posted by pgrigis on Fri, 11 Feb 2011 15:35:00 GMT

View Forum Message <> Reply to Message

On Feb 10, 7:47 pm, Matt Francis <mattjamesfran...@gmail.com> wrote:
>  On Feb 11, 11:44 am, Matt Francis <mattjamesfran...@gmail.com> wrote:
>
>>>  I would shocked if it wasn't several orders of magnitude faster
>>>  (for this many iterations) to Histogram your times array with some
>>>  appropriate bin size and then ask "which bin" your time_now
>>>  was in with Value_Locate.
>
>>  Thanks David, VALUE_LOCATE is exactly the function I'm looking for.
>
>  Thanks also to Paulo, who ninja'd my previous post.

As a general comment, from a basic algorithmic point of view,
finding one element in a sorted array is a log(N) kind of problem.

An example of an algorithm that does this is bisection:
go to the middle of the array - check if the wanted item
is left or right, then go to the middle of that side, check
in which quarter the element is, rinse and repeat.

Ciao,
Paolo