Subject: Re: Locating sequence of bytes within binary file
Posted by JDS on Thu, 17 Jun 2010 17:48:57 GMT
View Forum Message <> Reply to Message

On Jun 17, 4:03 am, medd <med...@googlemail.com> wrote:

> I am not an expert, but when I explain IDL to newbies I always say
> that it is a "matrix-oriented language", with all possible operations
> you can imagine on arrays. But looking for more than one consecutive
> value within an array seems to be too hard...

As usual, we can (over-)use IDL's array strengths to brute-force this using REBIN and dimensional TOTAL:

```
;===============
nh=1000000L
nn=20b
haystack=byte(randomu(sd,nh)*256)
start=long(randomu(sd)*(nh-nn))
needle=haystack[start:start+nn-1]

targ=[nn,ceil(float(nh)/nn)]
search=make_array(/BYTE,targ,/NOZERO)
quill=rebin(needle,targ,/SAMPLE)

for off=0,nn-1 do begin
  search[0]=shift(haystack,-off)
  matches=where(total(search eq quill,1,/PRESERVE_TYPE) eq nn,cnt)
  if cnt gt 0 then break
endfor

match=cnt gt 0?matches[0]*nn+off:-1
;===============
```

If you are interested in more than just the first match, simply omit the break statement, and accumulate a list of match locations (or increment a match count). It's limited to 256 byte needles, but that could be fixed by substituting PRODUCT for TOTAL (at a very slight speed penalty). It's reasonably fast, though of course cannot touch the speed of a true Boyer-Moore string search. I tried it on the same data set using INDEX in perl and found it roughly 40x faster.

Now for the really disappointing news: as is often found, brute-forcing, while emphasizing IDL's strengths, often comes with a penalty compared to more efficient algorithms. I find that STRPOS in IDL is at least 100x faster, likely because it uses an efficient string search algorithm internally. But, as you notice, IDL won't allow null

characters (0b) in a string (probably as a questionable concession to 0-delimited C strings).

That motivates another deeply unsatisfying, but resoundingly faster (20-50x) option: simply replace all 0b's with 1b's in both input byte array and search array, and just double-check for spurious matches as you go:

```
;===============
function binpos,haystack,needle
  nn=n_elements(needle)

  sn=string(needle>1b)
  sh=string(haystack>1b)

  pos=-1
  repeat begin
     pos=strpos(sh,sn,pos+1)
     if pos ne -1 && array_equal(haystack[pos:pos+nn-1],needle) then
break
  endrep until pos eq -1
  return,pos
end
;===============
```

In random arrays I find false positives are quite rare for search array lengths greater than a few. Of course, your data probably isn't random.

We might also lobby ITT to let STRPOS and its sort accept byte arrays (since frankly there is very little difference between them internally).

JD