### Ruby master - Bug #7724

#### 6 bugs with Range#bsearch over floats

01/22/2013 06:47 PM - marcandre (Marc-Andre Lafortune)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>marcandre (Marc-Andre Lafortune)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.0.0</td>
</tr>
<tr>
<td>ruby -v:</td>
<td>r38825</td>
</tr>
<tr>
<td>Backport:</td>
<td></td>
</tr>
</tbody>
</table>

#### Description

Take the following code, with from, to and search all Integer or all Float:

```
(from...to).bsearch{|f| f >= search}
```

I expected it to:

1) never yield more than 65 times for floats or Math.log(to-from, 2)+1 for Integer (*"log test"")
2) return search if from <= search < to, nil otherwise (*"coverage test"")
3) never yield the same f (*"uniqueness test"")
4) if range is exclusive, never yield to nor return to; if range is inclusive and block returns false always yield to (*"end of range test"")
5) never yield a number < from or > to (*"out of range test"")

These conditions are all respected for Integers but all can be broken for Floats

Test 0, 1, 3, 4 & 5 fail even for small ordinary float values, while test 2 requires some larger floats, say 1e100 to fail. For example bsearch can yield over 2000 times (instead of at most 65).

These tests and a correct Ruby implementation of bsearch can be found here: [GitHub](https://github.com/marcandre/backports/compare/marcandre:master...marcandre:bsearch)

My implementation also passes the MRI tests.

#### Associated revisions

Revision 115a51f8 - 01/29/2013 10:00 PM - marcandre (Marc-Andre Lafortune)

- range.c: Fix Range#bsearch for floats [Bug #7724]

Revision 38978 - 01/29/2013 10:00 PM - marcandre (Marc-Andre Lafortune)

- range.c: Fix Range#bsearch for floats [Bug #7724]

Revision 38978 - 01/29/2013 10:00 PM - marcandre (Marc-Andre Lafortune)

- range.c: Fix Range#bsearch for floats [Bug #7724]

Revision 38978 - 01/29/2013 10:00 PM - marcandre (Marc-Andre Lafortune)

- range.c: Fix Range#bsearch for floats [Bug #7724]

Revision 38978 - 01/29/2013 10:00 PM - marcandre (Marc-Andre Lafortune)

- range.c: Fix Range#bsearch for floats [Bug #7724]
History

#1 - 01/23/2013 05:48 PM - naruse (Yui NARUSE)
- Status changed from Open to Assigned
- Assignee set to mame (Yusuke Endoh)

#2 - 01/23/2013 06:42 PM - mame (Yusuke Endoh)
- Assignee changed from mame (Yusuke Endoh) to marcandre (Marc-Andre Lafortune)

Thanks Marc-Andre!

But I'm very sorry that I'll have no enough time to pursue this issues.
Could you please create a patch for C implementation?

Notice that the current implementation does NOT assume that the internal representation of double is IEEE 754. Please do not depend on it but use C's constants about double, such as FLT_RADIX and DBL_MANT_DIG.

--
Yusuke Endoh mame@tsg.ne.jp

#3 - 01/24/2013 04:38 AM - marcandre (Marc-Andre Lafortune)

mame (Yusuke Endoh) wrote:

Thanks Marc-Andre!

But I'm very sorry that I'll have no enough time to pursue this issues.
Could you please create a patch for C implementation?

Sure, I'll do it.

Notice that the current implementation does NOT assume that the internal representation of double is IEEE 754. Please do not depend on it but use C's constants about double, such as FLT_RADIX and DBL_MANT_DIG.

Actually, I don't think there is need to know how many bits are used for the mantissa vs exponent; I only need to know that double is represented with exponent bits then by mantissa bits. Apart from VAX (which had strange byte ordering, but we don't support), I believe this to be true.

So floatings can be seen as [exp, mantissa], thus have the same ordering as the the corresponding integers (forgetting about the sign). I.e. (double)(++(int64_t)a_float) is the smallest float strictly greater than a_float (assuming 0 <= a_float < infinity)

Do we have a list of non IEEE architectures we're hoping to support? Are there systems where sizeof(double) != sizeof(int_64_t)?

#4 - 01/30/2013 07:00 AM - marcandre (Marc-Andre Lafortune)
- Status changed from Assigned to Closed
- % Done changed from 0 to 100

This issue was solved with changeset r38978.
Marc-Andre, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.