Ruby trunk - Feature #10730

Implement Array#bsearch_index

01/11/2015 10:05 PM - radan (Radan Skorić)

Status:	Closed	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

We currently have Array#bsearch but no Array#bsearch_index and to me it seems that violates the principle of least surprise, especially when we consider the other combinations of existing methods that find either an element or it's index.

For example: the method would be very useful when needing to slice out a part of a sorted array. It would allow very quick location of the indices of the beginning and end of the segment.

A quick google of “ruby bsearch_index” reveals that I am not the only one that needed and expected that function to exist:

http://stackoverflow.com/questions/23481725/using-bsearch-to-find-index-for-inserting-new-element-into-sorted-array
http://stackoverflow.com/questions/8672472/is-there-a-built-in-binary-search-in-ruby
https://github.com/tyler/binary_search#usage

The very good thing is that we can get that method almost for free since the current implementation of bsearch internally finds the index and then looks up the actual element.

I have opened a PR on the github mirror that adds bsearch_index in what seems to me the simplest way possible:
https://github.com/ruby/ruby/pull/813 .
I changed the bsearch implementation into bsearch_index and based the bsearch on it.
Please Note: The diff is deceptively large, if you look carefully you will notice that the change is actually small and the actual binary search algorithm remained completely intact.

I have kept the behaviour documentation on bsearch and simply referenced it from bsearch_index to minimize the documentation changes.

Associated revisions

Revision c64d283f - 06/12/2015 07:28 AM - nobu (Nobuyoshi Nakada)

array.c: bsearch_index

- array.c (rb_ary_bsearch_index): Implement Array#bsearch_index method, which is similar to bsearch and returns the index or nil. [Feature #10730]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@50839 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 50839 - 06/12/2015 07:28 AM - nobu (Nobuyoshi Nakada)

array.c: bsearch_index

- array.c (rb_ary_bsearch_index): Implement Array#bsearch_index method, which is similar to bsearch and returns the index or nil. [Feature #10730]

Revision 50839 - 06/12/2015 07:28 AM - nobu (Nobuyoshi Nakada)

array.c: bsearch_index

- array.c (rb_ary_bsearch_index): Implement Array#bsearch_index method, which is similar to bsearch and returns the index or nil. [Feature #10730]

Revision 50839 - 06/12/2015 07:28 AM - nobu (Nobuyoshi Nakada)

array.c: bsearch_index

- array.c (rb_ary_bsearch_index): Implement Array#bsearch_index method, which is similar to bsearch and returns the index or nil. [Feature #10730]

Revision 50839 - 06/12/2015 07:28 AM - nobu (Nobuyoshi Nakada)

array.c: bsearch_index

- array.c (rb_ary_bsearch_index): Implement Array#bsearch_index method, which is similar to bsearch and returns the index or nil. [Feature #10730]

Revision 50839 - 06/12/2015 07:28 AM - nobu (Nobuyoshi Nakada)

array.c: bsearch_index

- array.c (rb_ary_bsearch_index): Implement Array#bsearch_index method, which is similar to bsearch and returns the index or nil. [Feature #10730]

History

#1 - 01/12/2015 08:43 AM - mame (Yusuke Endoh)

You may want to use Range#bsearch for the case.

```
i = (0...ary.size).bsearch {|i| predicate(ary[i]) }
```

--
Yusuke Endoh mame@ruby-lang.org

#2 - 01/12/2015 09:53 AM - radan (Radan Skorić)

Yusuke Endoh wrote:

> You may want to use Range#bsearch for the case.
>
> ```
> i = (0...ary.size).bsearch {|i| predicate(ary[i]) }
> ```
>
> --
> Yusuke Endoh mame@ruby-lang.org

Yusuke, thanks, that is a very clever approach.

But if we do that then we are just one small step away from having Array#bsearch based on Range#bsearch :

```
el = ary.at((0...ary.size).bsearch {|i| predicate(ary[i]) })
```

So then it seems to me the question also becomes: Why do we have Array#bsearch?

It seemed to me that we should either have Array#bsearch AND Array#bsearch_index or neither of them. IMHO the case where we have only one looks surprising which is not the ruby way.

What do you think?

#3 - 01/12/2015 12:46 PM - mame (Yusuke Endoh)

> Why do we have Array#bsearch?

Just because matz wanted it: https://redmine.ruby-lang.org/issues/4766#note-2

Personally, I don't think Array#bsearch is necessarily required. Range#bsearch is more general and powerful. However, Array#bsearch is indeed useful for typical use case of binary search, and it provides a similar API to C's BSEARCH(3). In this sense, Array#bsearch is considered as a handy short cut for Range#bsearch.

I'm neutral to Array#bsearch_index, but how handy is it? Symmetry/consistency is not a good reason for Ruby design.

http://stackoverflow.com/questions/23481725/using-bsearch-to-find-index-for-inserting-new-element-into-sorted-array

shows a somewhat reasonable use case, but the insert takes O(n). Is it okay? When you have to modify the array that is used for bsearch, rbtree gem might be a better choice; both search and insert take O(log n).

--
Yusuke Endoh mame@ruby-lang.org

#4 - 01/12/2015 01:56 PM - radan (Radan Skorić)

> I'm neutral to Array#bsearch_index, but how handy is it? Symmetry/consistency is not a good reason for Ruby design.

Let me then tell you about my use case. There is a sparse array of dates and I want to slice out a part of it that falls within minimum and maximum date. It is then later used to retrieve same values associated with those dates for displaying a chart. In another use case, you have to find a first element that matches the criteria and then take next N elements to display them. Bsearch index would make both concise and also very fast oneliners:

```
ary.slice((ary.bsearch_index {|e| e>=min} || 0) .. (ary.bsearch_index {|e| e>=max} || ary.size))
```

```
ary.slice(ary.bsearch_index { |e| predicate(e) }, N)
```

When implementing it, I knew about bsearch and I actually just kind of assumed I also have bsearch_index and was surprised it's not there. That is what made me go check the ruby source and see if I can actually add the method in a simple way.

So the biggest reason why IMHO we need either BOTH or NONE is the principle of least surprise and not symmetry or consistency.

But then again, maybe it's just me and not many other people would get the same idea. I'm open to being wrong on that point :)

> http://stackoverflow.com/questions/23481725/using-bsearch-to-find-index-for-inserting-new-element-into-sorted-array
>
> shows a somewhat reasonable use case, but the insert takes O(n). Is it okay? When you have to modify the array that is used for bsearch, rbtree gem might be a better choice; both search and insert take O(log n).

Yes, I agree with you, that is not the best use case. However it shows how that person also thought that exact method will be there.

P.S. Bye the way, thanks for taking the time to discuss this issue. :)

#5 - 01/12/2015 02:32 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

> Let me then tell you about my use case.

Thank you, looks good to me.

Also, I just noticed that Array#bsearch is weaker than C's BSEARCH(3). BSEARCH(3) returns the pointer to the array, so it effectively returns not only the value but also the index. Array#bsearch_index will make up for the weakness.

I give +1 for this feature, but anyway, Matz will determine a feature proposal. Assigning to him.

--
Yusuke Endoh mame@ruby-lang.org

#6 - 01/25/2015 08:55 PM - radan (Radan Skorić)

I rebased on the latest trunk and resolved conflicts with the updates to bsearch implementation.

If there's anything I can do to improve this proposal, I'll be happy to do it, just let me know, thanks.

#7 - 06/12/2015 07:11 AM - matz (Yukihiro Matsumoto)

Sounds good. Go ahead, Nobu.

Matz.

#8 - 06/12/2015 07:28 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

Applied in changeset r50839.

---

array.c: bsearch_index

- array.c (rb_ary_bsearch_index): Implement Array#bsearch_index method, which is similar to bsearch and returns the index or nil. [Feature #10730]