Ruby trunk - Feature #2348

RBTree Should be Added to the Standard Library

11/09/2009 06:41 AM - JEG2 (James Gray)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.6	

Description

=begin
The merits of this library have been discussed on Ruby core, with the strengths best summarized by this post:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/26602

RBTree has now been fixed to run on Ruby 1.9:

http://github.com/skade/rbtree

I think we should now give serious consideration to bringing it into the standard library.
=end

Related issues:

Related to Ruby trunk - Bug #9121: [PATCH] Remove rbtree implementation of So...	**Assigned**	**11/18/2013**

History

#1 - 11/09/2009 07:38 AM - Skade (Florian Gilcher)

=begin

On Nov 8, 2009, at 6:11 PM, James Gray wrote:

> Feature #2348: RBTree Should be Added to the Standard Library
> http://redmine.ruby-lang.org/issues/show/2348
>
> Author: James Gray
> Status: Open, Priority: Normal
> Category: lib, Target version: 1.9.2
>
> The merits of this library have been discussed on Ruby core, with
>
> the strengths best summarized by this post:
>
> http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/26602
>
> RBTree has now been fixed to run on Ruby 1.9:
>
> http://github.com/skade/rbtree
>
> I think we should now give serious consideration to bringing it into
>
> the standard library.

To add to that: I also contacted the maintainer of RBTree to
inform him of my patches and to ask for his thoughts. As the
library is of good-quality and fitted with a good test suite,
I would also volunteer to maintain it, but I want to wait
for an answer first.

Regards,
Florian Gilcher

=end

#2 - 11/09/2009 12:12 PM - nobu (Nobuyoshi Nakada)

=begin

Hi,

At Mon, 9 Nov 2009 06:41:57 +0900,
James Gray wrote in [ruby-core:26635]:

> RBTree has now been fixed to run on Ruby 1.9:
>
> http://github.com/skade/rbtree

It can't compile with non-gcc, or with gcc and $DEBUG.

diff --git a/extconf.rb b/extconf.rb
index 272790b..02f2e8e 100644
--- a/extconf.rb
+++ b/extconf.rb
@@ -2,5 +2,7 @@ require 'mkmf'

if $DEBUG

- $CFLAGS << ' -std=c89 -pedantic -Wall -Wno-long-long'
- if CONFIG['GCC'] == 'yes'
- $CFLAGS << ' -std=c89 -pedantic -Wno-long-long'
- end $defs << ' -Dinline=__inline' else @@ -8,4 +10,4 @@ else end

-have_func('rb_enumeratorize')
+have_func('rb_exec_recursive', 'ruby.h')
create_makefile('rbtree')
diff --git a/rbtree.c b/rbtree.c
index 9f19613..08bde65 100644
--- a/rbtree.c
+++ b/rbtree.c
@@ -12,8 +12,15 @@
#define HASH_PROC_DEFAULT FL_USER2

-#ifndef HAVE_RB_ENUMERATORIZE
+#ifndef RETURN_ENUMERATOR
#define RETURN_ENUMERATOR(obj, argc, argv) ((void)0)
#endif

+#ifndef RHASH_TBL
+#define RHASH_TBL(h) RHASH(h)->tbl
+#endif
+#ifndef RHASH_IFNONE
+#define RHASH_IFNONE(h) RHASH(h)->ifnone
+#endif
+
VALUE RBTree;
VALUE MultiRBTree;
@@ -428,5 +435,5 @@ static int
value_eq(const void* key1, const void* key2)
{

- return rb_equal((VALUE)key1, (VALUE)key2);
- return rb_equal((VALUE)key1, (VALUE)key2) != 0; }

@@ -1077,5 +1084,5 @@ rbtree_to_hash(VALUE self)
hash = rb_hash_new();
rbtree_for_each(self, to_hash_i, (void*)hash);

- RHASH(hash)->ifnone = IFNONE(self);
- RHASH_IFNONE(hash) = IFNONE(self); if (FL_TEST(self, RBTREE_PROC_DEFAULT)) FL_SET(hash, HASH_PROC_DEFAULT); @@ -1097,7 +1104,6 @@ rbtree_begin_inspect(VALUE self) { const char* c = rb_class2name(CLASS_OF(self));
- char str [strlen(c) + 5];
- sprintf(str, "#<%s: ", c);
- VALUE rb_str = rb_str_new2(str);
- VALUE rb_str = rb_str_new(0, strlen(c) + 4);
- sprintf(RSTRING_PTR(rb_str), "#<%s: ", c); return rb_str; } @@ -1109,4 +1115,5 @@ to_s_rbtree(VALUE self, VALUE nil) }

+#ifdef HAVE_RB_EXEC_RECURSIVE
VALUE
rbtree_to_s_recursive(VALUE self, VALUE arg, int recursive)
@@ -1116,4 +1123,5 @@ rbtree_to_s_recursive(VALUE self, VALUE arg, int recursive)
return to_s_rbtree(self, Qnil);
}

+#endif

/*
@@ -1123,8 +1131,11 @@ VALUE
rbtree_to_s(VALUE self)
{
+#ifdef HAVE_RB_EXEC_RECURSIVE
return rb_exec_recursive(rbtree_to_s_recursive, self, Qnil);

- //if (rb_inspecting_p(self))
- //   return rb_str_cat2(rbtree_begin_inspect(self), "...>");
- //return rb_protect_inspect(to_s_rbtree, self, Qnil); +#else
- if (rb_inspecting_p(self))
- return rb_str_cat2(rbtree_begin_inspect(self), "...>");
- return rb_protect_inspect(to_s_rbtree, self, Qnil); +#endif }

@@ -1194,8 +1205,12 @@ VALUE
rbtree_inspect(VALUE self)
{

- /*VALUE str = rbtree_begin_inspect(self);
- if (rb_inspecting_p(self))
- return rb_str_cat2(str, "...>");*/ +#ifdef HAVE_RB_EXEC_RECURSIVE   return rb_exec_recursive(rbtree_inspect_recursive, self, Qnil); +#else
- VALUE str = rbtree_begin_inspect(self);
- if (rb_inspecting_p(self))
- return rb_str_cat2(str, "...>");
- return rb_protect_inspect(inspect_rbtree, self, str); +#endif }

diff --git a/test.rb b/test.rb
index 8c533b8..32fdd25 100644
--- a/test.rb
+++ b/test.rb
@@ -136,5 +136,5 @@ class RBTreeTest < Test::Unit::TestCase
assert_raises(ArgumentError) { rbtree.default("e", "f") }

- rbtree = RBTree.new {|rbtree, key| @rbtree[key || "c"] }
- rbtree = RBTree.new {|tree, key| @rbtree[key || "c"] }   assert_equal("C", rbtree.default(nil))   assert_equal("B", rbtree.default("b")) @@ -182,5 +182,5 @@ class RBTreeTest < Test::Unit::TestCase   a = RBTree.new   b = RBTree.new
- a.readjust {|a, b| a <=> b }
- a.readjust {|x, y| x <=> y }   assert_not_equal(a, b)   b.readjust(a.cmp_proc) @@ -198,5 +198,14 @@ class RBTreeTest < Test::Unit::TestCase   assert_equal("E", @rbtree.fetch("e", "E"))   assert_equal("E", @rbtree.fetch("e") { "E" })
- class << (stderr = "")
- alias write <<
- end
- $stderr, stderr, $VERBOSE, verbose = stderr, $stderr, false, $VERBOSE
- begin   assert_equal("E", @rbtree.fetch("e", "F") { "E" })
- ensure
- $stderr, stderr, $VERBOSE, verbose = stderr, $stderr, false, $VERBOSE
- end

- assert_match(/warning: block supersedes default value argument/, stderr)

  assert_raises(ArgumentError) { @rbtree.fetch }
  @@ -535,5 +544,5 @@ class RBTreeTest < Test::Unit::TestCase
  assert_equal(%({"a"=>"A", "b"=>"B", "c"=>"C", "d"=>"D"}), tree)
  assert_equal(%("e"), default)

- assert_match(/#Proc:w+(@test.rb:d+)?/, cmp_proc)

- assert_match(/#Proc:w+(@#{__FILE__}:d+)?/o, cmp_proc)

  rbtree = RBTree.new

--
Nobu Nakada

=end

#3 - 11/10/2009 12:21 AM - Skade (Florian Gilcher)

=begin
Hi,

thanks for the patch, I applied it with a minor addition (also matching on HAVE_RB_EXEC_RECURSIVE in pp functions to use the "old" behaviour instead).

I expect that the compiler problems are only because of the flags in extconf.rb or are there other thing I missed?

Concerning HAVE_RB_EXEC_RECURSIVE: i looked it up and rb_inspecting_p is gone since March 2005. Are there considerable chances of a modern ruby version still in support that does not have rb_exec_recursive? (1.8.6 perhaps?)

Both questions are more out of curiosity, as I said, I'm not that into Ruby internals and/or consider myself a good C developer ;).

Regards,
Florian
=end

#4 - 11/11/2009 09:22 AM - nobu (Nobuyoshi Nakada)

=begin
Hi,

At Tue, 10 Nov 2009 00:21:45 +0900,
Florian Gilcher wrote in [ruby-core:26654]:

> thanks for the patch, I applied it with a minor addition
> (also matching on HAVE_RB_EXEC_RECURSIVE in pp functions to
> use the "old" behaviour instead).

Indentation seems broken in rbtree_to_s().

> I expect that the compiler problems are only because of the
> flags in extconf.rb or are there other thing I missed?

And syntax errors in rbtree_begin_inspect():

```
char str [strlen(c) + 5];
sprintf(str, "#<%s: ", c);
VALUE rb_str = rb_str_new2(str);
```

Dynamic size array and local variable definition after
executable statements are C99 features but not allowed in C89.

Also C++ style one-line comment in rbtree_to_s():
//if (rb_inspecting_p(self))

> Concerning HAVE_RB_EXEC_RECURSIVE: i looked it up and
> rb_inspecting_p is gone since March 2005. Are there
> considerable chances of a modern ruby version still in
> support that does not have rb_exec_recursive? (1.8.6
> perhaps?)

Since there was the code using rb_inspecting_p(). I don't like
comment-out for such case.

--
Nobu Nakada

=end

#5 - 11/28/2009 11:54 AM - ujihisa (Tatsuhiro Ujihisa)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

=begin

=end

#6 - 03/22/2010 04:26 AM - JEG2 (James Gray)

=begin
Is there any chance we could get this incorporated before the 1.9.2 feature freeze?
=end

#7 - 03/22/2010 05:16 AM - naruse (Yui NARUSE)

=begin
This ticket doesn't have:

- Who maintain it
- Sufficient reason to bundle

If someone maintain it, the problem is, Is this worth to bundle?
So you should persuade, like:

- Gauche have RBTree http://practical-scheme.net/gauche/man/gauche-refe_166.html
- You should use RBTree when you want to use Array#assoc
- You should use RBTree when you want OrderedHash So RBTree is worth to bundle with Ruby =end

#8 - 03/22/2010 09:40 AM - mame (Yusuke Endoh)

=begin
Hi,

2010/3/22 James Gray redmine@ruby-lang.org:

> Is there any chance we could get this incorporated before the 1.9.2 feature freeze?

This ticket is not simple since this feature seems to be against "Large Class Principle". We need matz's approval.

My current personal opinion is that it is appropriate for the feature to be just a part of set.rb as a back-end, instead of a first class library.

Is there case where we want to use RBTree directly, instead of set.rb?

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#9 - 03/22/2010 10:59 AM - spatulasnout (B Kelly)

=begin
Hi,

Yusuke ENDOH wrote:

> Is there case where we want to use RBTree directly, instead of set.rb?

I'm sorry if I've misunderstood - but it would not have occurred to me to use 'set' to access RBTree's functionality.

RBTree and MultiRBTree (both provided by require 'rbtree') are akin to std::map and std::multimap in C++ STL.

It has long been a mystery to me that a Sorted Pair Associative Container with O(log N) insert, search, and delete complexity has not been a part of ruby's stdlib.

RBTree and MultiRBTree provide functionality which, with its worst-case O(log N) search, insert, and delete complexity for a sorted pair associative container can't be readily duplicated with Array, Hash, or Set. (As far as I know.)

RBTree and MultiRBTree are very useful container types when needed.

I do think the "RB" portion of the name is slightly unfortunate, as we don't generally care that it is implemented as a red-black tree internally; we just care about O(log N) complexity guarantees.

Anyway - I apologize if i've merely regurgitated a litany of obvious points into the conversation. I didn't really understand why RBTree/MultiRBTree would be considered a variant of Set?

Regards,

Bill

=end

#10 - 03/22/2010 11:27 AM - mame (Yusuke Endoh)

=begin
Hi,

2010/3/22 Bill Kelly billk@cts.com:

> RBTree and MultiRBTree provide functionality which, with its worst-case O(log N) search, insert, and delete complexity for a sorted pair associative container can't be readily duplicated with Array, Hash, or Set. ?(As far as I know.)

Hash has amortized O(1) search, insert, and delete complexity, I think. Indeed, it becomes O(N) at worst-case (when rehash occurs). Does anyone have a concrete problem due to rehash?

I think this feature request is very tough because it can be substituted by Hash in many cases. So I think you guys should appeal the difference. It would be good to show some real-world case where Hash cannot be used and RBTree is really needed.

> I do think the "RB" portion of the name is slightly unfortunate, as we don't generally care that it is implemented as a red-black tree internally; we just care about O(log N) complexity guarantees.

True.

> Anyway - I apologize if i've merely regurgitated a litany of obvious points into the conversation. ?I didn't really understand why RBTree/MultiRBTree would be considered a variant of Set?

There is no use case presented other than set, as far as I read.

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#11 - 03/22/2010 07:06 PM - spatulasnout (B Kelly)

=begin
Yusuke ENDOH wrote:

> 2010/3/22 Bill Kelly billk@cts.com:
>
> > RBTree and MultiRBTree provide functionality which, with its worst-case O(log N) search, insert, and delete complexity for a sorted pair associative container can't be readily duplicated with Array, Hash, or Set. ?(As far as I know.)
>
> Hash has amortized O(1) search, insert, and delete complexity, I think. Indeed, it becomes O(N) at worst-case (when rehash occurs). Does anyone have a concrete problem due to rehash?

Agreed: for Hash I would expect O(1) search, and amortized O(1) insert and delete complexity.

To avoid rehash, a Hash#reserve(size) method might be nice, but, for me, that is all separate from why I am interested in RBTree.

> I think this feature request is very tough because it can be substituted by Hash in many cases. So I think you guys should

> appeal the difference. It would be good to show some real-world case where Hash cannot be used and RBTree is really needed.

Some differences:

Hash is not maintained in key-sorted order.

Hash does not offer upper_bound(key) or lower_bound(key) or bound(key1, key2) in O(log N) time.

Hash doesn't provide fast search for partial string key.

An example, indexing words in documents, and doing partial keyword searches.

(Note: MultiRBTree#bound seems to be broken.)

```
require 'rbtree'

ful_ = %w(
fulcra
fulcrum
fulcrums
fulfil
fulfill
fulfilled
fulfilling
fulfillment
fulfills
fulfilment
fulfils
full
fullback
fullbacks
fulled
fuller
fullest
fulling
fullness
fulls
fully
fulminate
fulminated
fulminates
fulminating
fulmination
fulminations
fulsome
)

multi_ = %w(
multicolored
multicultural
multiculturalism
multidimensional
multifaceted
multifarious
multifariousness
multilateral
multilingual
multimedia
multimillionaire
multimillionaires
multinational
multinationals
multiple
multiples
multiplex
multiplexed
multiplexer
multiplexers
multiplexes
multiplexing
multiplicand
```

```
multiplicands
multiplication
multiplications
multiplicative
multiplicities
multiplicity
multiplied
multiplier
multipliers
multiplies
multiply
multiplying
multiprocessing
multipurpose
multiracial
multitasking
multitude
multitudes
multitudinous
multivariate
multivitamin
multivitamins
)

dis_ = %w(
distortions
distorts
distract
distracted
distracting
distraction
distractions
distracts
distrait
distraught
distress
distressed
distresses
distressful
distressing
distressingly
distribute
distributed
distributes
distributing
distribution
distributions
distributive
distributor
distributors
district
districts
distrust
distrusted
distrustful
distrustfully
distrusting
distrusts
disturb
disturbance
disturbances
disturbed
disturbing
disturbingly
disturbs
)


doc1 = [ "foo/doc1.txt", ful_   + multi_ ]
doc2 = [ "bar/doc2.txt", multi_ + dis_   ]
doc3 = [ "baz/doc3.txt", ful_   + dis_   ]


dict = MultiRBTree.new
```

```
[doc1, doc2, doc3].each do |docpath, words|
  words.each do |w|
    dict.store(w, docpath)
  end
end
```

```
puts dict.lower_bound("mult")  # => ["multicolored", "foo/doc1.txt"]
puts dict.upper_bound("mult")  # => ["fulsome", "baz/doc3.txt"]
puts dict.bound("mult")        # <-- broken
```

Note: The documentation for RBTree#bound reads:

- call-seq:
- rbtree.bound(key1, key2 = key1) => array
- rbtree.bound(key1, key2 = key1) {|key, value| block} => rbtree *
- Returns an array containing key-value pairs between the result of
- MultiRBTree#lower_bound and MultiRBTree#upper_bound. If a block is
- given it calls the block once for each pair.

So I expected dict.bound("mult") to return all elements from:

dict.lower_bound("mult") => ["multicolored", "foo/doc1.txt"]

through:

dict.upper_bound("mult") => ["fulsome", "baz/doc3.txt"]

However, #bound just retunrs [] :(

I consider this a bug.

A comment:

Even if MultiRBTree#bound worked as expected, I must concede a
significant liability of MultiRBTree's API compared to C++
std::multimap, is the lack of iterators.

With std::multimap, I can find dict.lower_bound("mult"), and
then iterate over as many or as few subsequent elements in sorted
order as I choose. (When building a typedown menu, for example,
I may only want the first 10 results.)

I suppose rbtree.bound(key1, key2, limit) would be one way to
provide equivalent functionality; or perhaps support for 1.9
Enumerable would be another.

Anyway, issues with MultiRBTree#bound aside, other ways I've
used a Sorted Pair Associative Container include implementing
various kinds of priority queues. (Sorted integer keys.)

I do think that for RBTree and MultiRBTree to be as generally
useful as C++ std::map and std::multimap, there should be
versions of methods like bound, lower_bound, upper_bound, that
return an enumerator.

Also, I think it should be possible to unambiguously delete
a specific element from a MultiRBTree.

Currently:

t = MultiRBTree.new
t.store "foo", "456"
t.store "foo", "123"
t.delete "foo" # <-- which is deleted? foo/123 or foo/456 ?

It appears that MultiRBTree#delete deletes the oldest key/value
pair matching the supplied key, so it would be foo/456.

As far as I can tell, there's no way to delete foo/123 from t
without first deleting foo/456. So that is another limitation
when compared to std::multimap.

Hmm.

So it seems that even though RBTree and MultiRBTree are internally equivalent to C++ std::map and std::multimap, the interface exposed to the programmer is less flexible than the C++ versions.

I think RBTree and MultiRBTree would be more useful it were possible to obtain enumerators from bound and lower_bound, and to be able to delete arbitrary elements in a MultiRBTree.

Sorry this email is so long. I didn't expect to encounter these issues.

Regards,

Bill

=end

#12 - 03/22/2010 08:06 PM - mame (Yusuke Endoh)

=begin
Hi,

Thank you for your detailed reply!

2010/3/22 Bill Kelly billk@cts.com:

> Hash is not maintained in key-sorted order.
>
> Hash does not offer upper_bound(key) or lower_bound(key) or bound(key1, key2) in O(log N) time.

Good. I start to want RBTree too :-)

> Hash doesn't provide fast search for partial string key.

You mean prefix search, right?
And, can partial *array* key be handled?

> puts dict.upper_bound("mult") ?# => ["fulsome", "baz/doc3.txt"]

Is this correct? I expect it to return ["multivitamins", "foo/doc1.txt"] or ["multivitamins", "bar/doc2.txt"].

> However, #bound just retunrs [] ?:(

I guess it is because upper_bound is broken.

> I do think that for RBTree and MultiRBTree to be as generally useful as C++ std::map and std::multimap, there should be versions of methods like bound, lower_bound, upper_bound, that return an enumerator.

Agreed. I think bound should return an enumerator instead of an array when block is not given.

You presented dictionary-like application and priority queue as use cases. I'm convinced at the explanation.
But RBTree seems to have some problems of not only simple bug but also API design. If so, it is slightly premature, so it may be better to defer its bundle to 1.9.3 or later.

--
Yusuke ENDOH mame@tsg.ne.jp

=end

#13 - 03/23/2010 07:31 AM - spatulasnout (B Kelly)

=begin
Tanaka Akira wrote:

> 2010/3/22 Bill Kelly billk@cts.com:
>
> > Hash doesn't provide fast search for partial string key.
>
> RBTree doesn't provide it.
> Because RBTree uses <=> for comparing elements.
> The result of <=> is not useful to test partial key match.

Ah. I meant via #lower_bound.

/*

- Look for the node corresponding to the lowest key that is equal to or
- greater than the given key. If there is no such node, return null. */

dnode_t *dict_lower_bound(dict_t *dict, const void *key)

Seems to me this should provide a fast search for a partial
string key. (?)

Regards,

Bill

=end

#14 - 03/23/2010 07:50 AM - spatulasnout (B Kelly)

=begin
Bill Kelly wrote:

> Tanaka Akira wrote:
>
> > 2010/3/22 Bill Kelly billk@cts.com:
> >
> > > Hash doesn't provide fast search for partial string key.
> > > RBTree doesn't provide it.
> > > Because RBTree uses <=> for comparing elements.
> > > The result of <=> is not useful to test partial key match.
>
> Ah. I meant via #lower_bound.
>
> /*
>
> - Look for the node corresponding to the lowest key that is equal to or
> - greater than the given key. If there is no such node, return null. */
>
> dnode_t *dict_lower_bound(dict_t *dict, const void *key)
>
> Seems to me this should provide a fast search for a partial
> string key. (?)

Sorry, I was imprecise. By partial I meant prefix, as
Yusuke ENDOH pointed out.

Regards,

Bill

=end

#15 - 03/23/2010 07:53 AM - headius (Charles Nutter)

=begin
Jumping in with JRuby perspective..

I suppose this would be easiest for us to implement by wrapping the built-in TreeMap from Java:

http://java.sun.com/j2se/1.5.0/docs/api/java/util/TreeMap.html

I have not looked over RBTree API, but hopefully there's nothing there we couldn't implement atop TreeMap/TreeSet.

- Charlie

=end

#16 - 03/23/2010 08:07 AM - spatulasnout (B Kelly)

=begin
Yusuke ENDOH wrote:

> Hash doesn't provide fast search for partial string key.

You mean prefix search, right?
And, can partial *array* key be handled?

Ah, yes. Thanks, I did mean prefix.

And indeed, based on experiments in irb with array-based keys, it does appear that lower_bound works with array key prefix search.

> puts dict.upper_bound("mult") ?# => ["fulsome", "baz/doc3.txt"]

Is this correct? I expect it to return ["multivitamins", "foo/doc1.txt"] or ["multivitamins", "bar/doc2.txt"].

I had assumed it worked like std::map upper_bound (http://www.cplusplus.com/reference/stl/map/upper_bound/) returning "first element in the container whose key compares greater than x."

However, in rbtree's dict.c, dict_upper_bound() is documented as:

/*

- Look for the node corresponding to the greatest key that is equal to or
- lower than the given key. If there is no such node, return null. */

So, its behavior does not seem to match the comment. (I don't know whether to consider the comment wrong, or the behavior wrong. :)

Regards,

Bill

=end

#17 - 04/04/2010 01:28 AM - znz (Kazuhiro NISHIYAMA)

- Target version changed from 1.9.2 to 2.0.0

=begin

=end

#18 - 10/07/2011 01:49 AM - dgraham (David Graham)

Is there a chance RBTree can be added to the standard library for Ruby 2.0? I've needed it to implement priority queues and key range scans, but the binary gem doesn't play well with JRuby or Rubinius. It would help if we could count on this data structure being included with Ruby.

Thanks!
David

#19 - 10/07/2011 03:03 AM - JEG2 (James Gray)

I still agree. We've literally been asking for NArray and RBTree in the standard library for years. Pretty please? :)

#20 - 10/07/2011 05:22 AM - spatulasnout (B Kelly)

I wholeheartedly agree about the usefulness of the data structure.

I'm hesitant to type this, because I don't want to impede RBTree's path toward first-class citizenship.

But last time I checked there appeared to be some API deficiencies that significantly limited RBTree's potential usefulness:

http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/28860
http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/28879

Although I suppose it's possible these could be addressed at a later date?

Regards,

Bill

#21 - 10/07/2011 07:23 AM - rkh (Konstantin Haase)

SortedSet could then depend on it properly instead of the voodoo code that ships with Ruby atm.

Konstantin

On Oct 6, 2011, at 13:22 , B Kelly wrote:

> Issue #2348 has been updated by B Kelly.
>
> I wholeheartedly agree about the usefulness of the data structure.
>
> I'm hesitant to type this, because I don't want to impede RBTree's path toward first-class citizenship.
>
> But last time I checked there appeared to be some API deficiencies that significantly limited RBTree's potential usefulness:
>
> http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/28860
> http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/28879
>
> Although I suppose it's possible these could be addressed at a later date?
>
> Regards,
>
> Bill
>
> ---
>
> Feature #2348: RBTree Should be Added to the Standard Library
> http://redmine.ruby-lang.org/issues/2348
>
> Author: James Gray
> Status: Assigned
> Priority: Normal
> Assignee: Yukihiro Matsumoto
> Category: lib
> Target version: 1.9.x
>
> ¾gin
> The merits of this library have been discussed on Ruby core, with the strengths best summarized by this post:
>
> http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/26602
>
> RBTree has now been fixed to run on Ruby 1.9:
>
> http://github.com/skade/rbtree
>
> I think we should now give serious consideration to bringing it into the standard library.

#22 - 10/07/2011 08:53 AM - ko1 (Koichi Sasada)

(2011/10/07 1:50), David Graham wrote:

> Is there a chance RBTree can be added to the standard library for Ruby 2.0? I've needed it to implement priority queues and key range scans, but the binary gem doesn't play well with JRuby or Rubinius. It would help if we could count on this data structure being included with Ruby.

Gem is not enough?

--
// SASADA Koichi at atdot dot net

#23 - 10/07/2011 09:53 AM - Anonymous

On Thu, Oct 6, 2011 at 6:34 PM, SASADA Koichi ko1@atdot.net wrote:

> (2011/10/07 1:50), David Graham wrote:
>
> > Is there a chance RBTree can be added to the standard library for Ruby 2.0? I've needed it to implement priority queues and key range scans, but the binary gem doesn't play well with JRuby or Rubinius. It would help if we could count on this data structure being included with Ruby.
>
> Gem is not enough?

I guess I just feel I would use RBTree and NArray a lot more than some
things we have in the standard library. It's about the same
usefulness as Set, in my opinion. Maybe even a little more.

James Edward Gray II

#24 - 10/07/2011 10:23 AM - ko1 (Koichi Sasada)

(2011/10/07 9:46), James Gray wrote:

> On Thu, Oct 6, 2011 at 6:34 PM, SASADA Koichi ko1@atdot.net wrote:
>
> > (2011/10/07 1:50), David Graham wrote:
> >
> > > Is there a chance RBTree can be added to the standard library for Ruby 2.0? I've needed it to implement priority queues and key range scans, but the binary gem doesn't play well with JRuby or Rubinius. It would help if we could count on this data structure being included with Ruby.
> >
> > Gem is not enough?
>
> I guess I just feel I would use RBTree and NArray a lot more than some
> things we have in the standard library. It's about the same
> usefulness as Set, in my opinion. Maybe even a little more.

Some people think most of standard libraries should be in gem. I think
you need to persuade them.

--
// SASADA Koichi at atdot dot net

#25 - 10/07/2011 10:23 AM - Anonymous

On Thu, Oct 6, 2011 at 8:07 PM, SASADA Koichi ko1@atdot.net wrote:

> Some people think most of standard libraries should be in gem. I think
> you need to persuade them.

I sympathize, but we are still adding new libraries as of Ruby 1.9.3
and people have literally been wanting these two for years. I'm not
clear on why some libraries make it but these don't.

James Edward Gray II

#26 - 10/07/2011 10:29 AM - Anonymous

On Oct 6, 2011, at 9:07 PM, SASADA Koichi wrote:

> Some people think most of standard libraries should be in gem. I think
> you need to persuade them.

I think the intent is for RBTree to be included with the Ruby distribution via the standard library or via 'standard gems'. That is to say, the inclusion of RBTree into the standard Ruby 'distribution' is orthogonal to whether the standard distribution packages the standard library as gems or not.

Gary Wright

#27 - 10/07/2011 11:23 AM - mrkn (Kenta Murata)

(2011.10.07 01:50), David Graham wrote:

> Is there a chance RBTree can be added to the standard library for Ruby 2.0?

I agree with you if the library name is changed.
The name of RBTree is too specific to its internal algorithm.
If we adopt RBTree, we must change the name of the library after
more better algorithms would be discovered.

--
Kenta Murata muraken@gmail.com
1D69 ADDE 081C 9CC2 2E54 98C1 CEFE 8AFB 6081 B062

#28 - 10/07/2011 03:23 PM - cjheath (Clifford Heath)

On 07/10/2011, at 1:16 PM, Kenta Murata wrote:

> (2011.10.07 01:50), David Graham wrote:
>
> > Is there a chance RBTree can be added to the standard library for Ruby 2.0?
> > I agree with you if the library name is changed.
> > The name of RBTree is too specific to its internal algorithm.
> > If we adopt RBTree, we must change the name of the library after
> > more better algorithms would be discovered.

I agree. Hash is not named after the hashing algorithm that's being used,
and Array is not named after its structure either.

For sorted structures, I've previously used the name Sequence. I think
this name would be suitable.

I also wish that Ruby had this container type available as a standard.

Clifford Heath.

#29 - 10/07/2011 11:23 PM - Anonymous

On Fri, Oct 7, 2011 at 1:20 AM, Clifford Heath clifford.heath@gmail.com wrote:

> On 07/10/2011, at 1:16 PM, Kenta Murata wrote:
>
> > (2011.10.07 01:50), David Graham wrote:
> >
> > > Is there a chance RBTree can be added to the standard library for Ruby 2.0?
> > > I agree with you if the library name is changed.
> > > The name of RBTree is too specific to its internal algorithm.
> > > If we adopt RBTree, we must change the name of the library after
> > > more better algorithms would be discovered.
>
> I agree. Hash is not named after the hashing algorithm that's being used,
> and Array is not named after its structure either.
>
> For sorted structures, I've previously used the name Sequence. I think
> this name would be suitable.
>
> I also wish that Ruby had this container type available as a standard.

I think Tree would be a fine name and closer to Hash.

James Edward Gray II

#30 - 10/08/2011 06:54 AM - cjheath (Clifford Heath)

On 08/10/2011, at 1:10 AM, James Gray wrote:

> On Fri, Oct 7, 2011 at 1:20 AM, Clifford Heath clifford.heath@gmail.com wrote:
>
> > On 07/10/2011, at 1:16 PM, Kenta Murata wrote:
> >
> > > (2011.10.07 01:50), David Graham wrote:
> > >
> > > > Is there a chance RBTree can be added to the standard library for Ruby 2.0?
> > > > I agree with you if the library name is changed.
> > > > The name of RBTree is too specific to its internal algorithm.
> > > > If we adopt RBTree, we must change the name of the library after
> > > > more better algorithms would be discovered.
> > >
> > > I agree. Hash is not named after the hashing algorithm that's being used,
> > > and Array is not named after its structure either.
> > >
> > > For sorted structures, I've previously used the name Sequence. I think
> > > this name would be suitable.
> > >
> > > I also wish that Ruby had this container type available as a standard.
> >
> > I think Tree would be a fine name and closer to Hash.
>
> Is there any part of the API which allows a user to know it's a Tree?
> If so, why?
>
> If it's not externally visible in the API, it should not appear in the name.
>
> My 2c.
>
> Clifford Heath.

#31 - 05/18/2012 10:33 AM - jvoorhis (Jeremy Voorhis)

I think that Ruby developers would definitely benefit from having a range of well-implemented data structures within reach. I don't understand why the implementation-revealing name is an issue when our most common options are already named Array [contiguous chunk of memory] and Hash[-table]. Renaming this library's classes to something SortedMap and SortedMultiMap seems incongruous.

#32 - 10/27/2012 05:08 AM - ko1 (Koichi Sasada)

ping. status?

#33 - 10/27/2012 08:03 AM - matz (Yukihiro Matsumoto)

- Target version changed from 2.0.0 to 2.6

I am not positive about adding a new library to the distribution while we are discussion moving toward gems.
I am not refusuig, however, so I label this "next minor".

Matz.

#34 - 01/21/2014 05:06 PM - zzak (Zachary Scott)

Theres a discussion going on about possibly removing dependency on RBTree, or SortedSet all together.

Please see #9121

#35 - 08/27/2014 03:29 AM - hsbt (Hiroshi SHIBATA)

- Related to Bug #9121: [PATCH] Remove rbtree implementation of SortedSet due to performance regression added

#36 - 10/22/2017 02:14 AM - mame (Yusuke Endoh)

Three points:

- If RBTree gem is bundled, we will do so by using the (recently-established) framework of bundled gems.
- The current framework of bundled gems does not support extension library (maybe). We need to improve the framework first.
- After that, we must decide if RBTree gem should be bundled or not.

#37 - 10/23/2017 04:26 AM - knu (Akinori MUSHA)

Honestly, I have no idea if this library is or can become popular.

SortedSet was originally meant to be an example implementation to show what it is like to implement a subclass of Set with an alternative data structure or an additional algorithm, because I designed Set with consideration so that it is easily extensible unlike stock container classes like Hash and Array. Actually I wrote two examples: SortedSet and RestrictedSet, and the latter was kept in the document as I was unsure if it was practically useful.

So, it was not my point to promote rbtree as standard library, but just to show Hash is not the only possible backend for Set.

#38 - 11/29/2017 07:37 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

Unlike the past, it's not smart to add the standard library. Use gem.

Matz.