Subject: what about VectorBiMap / ArrayBiMap ?
Posted by kohait00 on Fri, 30 Apr 2010 07:55:48 GMT
View Forum Message <> Reply to Message

if you have lets say VectorMap<String, String> which is kind of a dictionary map, than you can search in one direction using the Find method. but if you want to look for the key that corresponds to a value, than there is only FindIndex(), which is sort of plain compare loop. what about having a container class, that offers both directions?

```
const VectorMap<K, T>;
int id = VectorMap<K, T>::Find(const K & key);
int id = VectorMap<K, T>::Find(const T & t);

const K & VectorMap<K, T>::operator[] const;
K & VectorMap<K, T>::operator[];

//donno if thats possible to override like that
const K & VectorMap<K, T>::operator[] const;
K & VectorMap<K, T>::operator[];
```

Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by mirek on Fri, 30 Apr 2010 08:49:47 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 30 April 2010 03:55if you have lets say VectorMap<String, String> which is kind of a dictionary map, than you can search in one direction using the Find method.

What you mean by direction? If you want to search starting with the last element with the value, you still can - FindLast, FindPrev.

Quote:
 but if you want to look for the key that corresponds to a value, than there is only FindIndex(), which is sort of plain compare loop. what about having a container class, that offers both directions?

Just use pair of Indexes.

In fact, the Index is the ultimate mapping weapon. VectorMap is just its common use wrapper...

Mirek

## Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by Mindtraveller on Fri, 30 Apr 2010 11:13:10 GMT
View Forum Message <> Reply to Message

Over three years of using U++ containers approved them to be extremely quick and easy to use. Everything you put into ****Map, is searched with hash, which is extremely fast even for large number of records.

The only thing which IMO is needed is Index "const hash/key" flavour which means that container element doesn't change it's internal state far enough to change it's hash value. This flavour should make possible to return (T &) from operator[], not (const T &).

Above this thing, U++ containers are far more comfortable and fast than any STL.

## Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by kohait00 on Fri, 30 Apr 2010 12:45:11 GMT
View Forum Message <> Reply to Message

hi Mindtraveller

this is actually exactly what i was adressing in
 http://www.ultimatepp.org/forum/index.php?t=msg&th=5142& amp;start=0&

to return the responsibility of maintainig the hash/state of an object returned by T& Index::operator[] to the user.

say if a user modifies an element from Index (which is needed sometimes), he knows which element it was, and simply updates the internel hash ref of that object, calling SetKey(i) or sth. (analog to VectorMap::SetKey(i, key) )

add the follwing at
Index.h:127

```
 void     SetKey(int i)             { hash.Set(i,hashfn(key[i])); }

 T&       operator[](int i)         { return key[i]; }
```

should do it. i discussed it there with mirek, he was hesitating.. which is sort of understandable, doing so, you might end up breaking consistency of Index if not carefull.

## Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by kohait00 on Fri, 30 Apr 2010 14:32:12 GMT
View Forum Message <> Reply to Message

@Mirek:

direction is meant to be key -> value, or value -> key;

say i have a pair of Strings, in a capabale VectorBiMap,
and add "MyKey" | "MyValue" pair (here spoken as key/value), then i will be able to find location of "MyValue" *VIA HASH* of "MyKey", what is trivial and possible with VectorMap.

but what if i want to be able to find location of "MyKey" (which is same location of "MyValue") *VIA HASH* of "MyValue"?

this is currently not possible, except manually using 2 Indexes, separately driving their api together. But is quite cool.

usecase: (simple and stupid) a dictionary 2 languages with exactly a pair of words like "Buenos Dias" <=> "Hi" (stupid)
and a *HIGH* speed translation in *BOTH* directions needed.

another usecase is a communication protocol translator, which depending on strings sends special values, and answers with values, which map to strings again.

this would need *2* Indexes with same stuff in it but swaped K-T

i had some ideas like the following:

```
template<class K, class T, class HashFnK = StdHash<K>, class HashFnT = StdHash<T>>
class VectorBiMap
 : public Index<K, HashFnK>
 , public Index<T, HashFnT>
{

 //following the api
//find T with K hash
//find K with T hash
};

template<class K, class T, class HashFnK = StdHash<K>, class HashFnT = StdHash<T>>
class VectorBiMap2
 : public MoveableAndDeepCopyOption<VectorBiMap2<K, T, HashFnK, HashFnT> >
 , public AMap< K, T, Vector<T>, HashFnK >
{
 typedef AMap< K, T, Vector<T>, HashFnK > B;
 HashBase hash2;
 HashFnT hashfn2;

 //following the api
//find T with K hash
//find K with T hash
};
```

i took a look in to implementation and it looks like there is no way except for coding an additional Interface, AIndex is not suitable AMap neither, but its a mix of both

PITFALL: all hash containers of Upp allow hash collision (handled internally with linked list). for a bijectional relation, this is not allowed ! or to define this to be a right unique relation (surrection)

i'll try it..any hints/comments/evaluations on the idea welcome

PS: in case of using Index, this would imply to use non const operator[] there like dicsribed above i think

---

## Subject: Re: what about VectorBiMap / ArrayBiMap ?
## Posted by mirek on Fri, 30 Apr 2010 20:54:29 GMT

View Forum Message <> Reply to Message

kohait00 wrote on Fri, 30 April 2010 10:32@Mirek:

direction is meant to be key -> value, or value -> key;

say i have a pair of Strings, in a capabale VectorBiMap,
and add "MyKey" | "MyValue" pair (here spoken as key/value), then i will be able to find location of "MyValue" *VIA HASH* of "MyKey", what is trivial and possible with VectorMap.

but what if i want to be able to find location of "MyKey" (which is same location of "MyValue") *VIA HASH* of "MyValue"?

this is currently not possible, except manually using 2 Indexes, separately driving their api together. But is quite cool.

usecase: (simple and stupid) a dictionary 2 languages with exactly a pair of words like "Buenos Dias" <=> "Hi" (stupid)
and a *HIGH* speed translation in *BOTH* directions needed.

```
Index<String> spain;
Index<String> english;

void AddWord(const String& sp, const String& en)
{
  spain.Add(sp);
  english.Add(en);
}
```

```
String EnglishToSpain(const String& w)
{
  int q = english.Find(w);
  return q >= 0 ? spain[q] : String();
}

String SpainToEnglish(const String& w)
{
  int q = spain.Find(w);
  return q >= 0 ? english[q] : String();
}
```

Mirek

---

## Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by kohait00 on Mon, 03 May 2010 20:37:11 GMT

View Forum Message <> Reply to Message

hi mirek

thanks for the advice, its the fastest solution anyway, but thats exactly what i wanted to avoid
i'd prefer a monolithic solution, more of the kind of AMap or AIndex.. to have a template for later..

i'll try and if it makes sense, i might post it here.

BTW: is there a possibility in the containers to restrict the hash colision avoidance, something like make an ASSERT if next element would add or sth?. one could check it for one's self, but to have a container setup up like "you only accept one of them" is pretty fine.

---

## Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by mirek on Tue, 04 May 2010 14:29:06 GMT

View Forum Message <> Reply to Message

kohait00 wrote on Mon, 03 May 2010 16:37
BTW: is there a possibility in the containers to restrict the hash colision avoidance, something like make an ASSERT if next element would add or sth?. one could check it for one's self, but to have a container setup up like "you only accept one of them" is pretty fine.

Sorry, but I do not really understand what are you speaking about...

---

## Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by kohait00 on Thu, 06 May 2010 18:38:40 GMT

View Forum Message <> Reply to Message

sorry, i try to make it more clear..

in containers, by default, it is possible adding another element with same hash. this is called hash collision, because normaly a hash value should map to exactly one counterpart.

is there a possib in the containers internally, to restrict such behaviour besides manually checking before adding another element?

i know of FindAdd, but this handles it kind of differently..

Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by mirek on Fri, 07 May 2010 12:37:13 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 06 May 2010 14:38sorry, i try to make it more clear..

in containers, by default, it is possible adding another element with same hash. this is called hash collision, because normaly a hash value should map to exactly one counterpart.

is there a possib in the containers internally, to restrict such behaviour besides manually checking before adding another element?

i know of FindAdd, but this handles it kind of differently..

Uh, I am not quite sure what you want to check.

Generally, hashing is always a little bit stochastic. In theory, it is possible that collisions would slow down your code, but statistically, it is quite unlikely.

Or, if you want some analogy, some very special pattern of memory allocations and frees could put down to knees any memory allocator and/or any cache subsystem. But it is so unlikely to happen (unless engineered) that we are still using memory allocators and caches...

BTW, of course, U++ changes the size of hashing space dynamically based on number of elements...

Whatever, my advice is to forget about this. It is internal issue, do not think in terms of hash codes...

Subject: Re: what about VectorBiMap / ArrayBiMap ?
Posted by kohait00 on Fri, 07 May 2010 15:59:11 GMT
View Forum Message <> Reply to Message

thank you mirek, that was pretty clear