Exeray Big Data Enabler

Efficient Data Structures Specially Designed for

Big Data Analytics

Exeray Inc

Table of Contents

Introduction

Exeray is about evolution and revolution in big data. It is well known that data structures are the fundamental components in computer software engineering and play a critical role in developing complex computer devices and systems. However, popular legacy data structures such as linked list, binary search tree, hash maps already have existed for many decades. As smart devices, information sharing networks, and data mining tools lead us into big data age, simply superior data structures and algorithms are needed to build complex systems to handle massive datasets.

Exeray has creatively designed Big Data Enabler, a suite of innovative and data containers, to resolve the challenges of large datasets in real-time computing. Big Data Enabler provides consistent high performance. From the experiments with datasets of millions of data items, 300% ~ 1000% speed improvement has been achieved compared to conventional key-value containers.

Installation

Abax Big Data Enabler, also known as Abax, can be downloaded from www.exeray.com and installed on your development host, client hosts, or any server hosts. You can follow these simple steps to install:

Step 1: Download abaxN.N.tar.gz from www.exeray.com

Step 2: $ tar zxf abaxN.N.tar.gz; cd abaxN.N; make install (as user root)

Note that N.N denotes the current release version of Abax. Root account or any sudo user account should be used during the installation. Big Data Enabler C++ header files will be installed in the /usr/local/include directory, and library files will be installed in the /usr/local/lib directory. Make sure you include these search directories in compiler options. In free-evaluation packages, only a limited number of keys can be inserted into the containers, while commercial packages do not have the limit. Also make sure you use an up-to-date license for your commercial package. Invalid licenses will cause undefined behavior in Big Data Enabler.

How to Use Big Data Enabler

In your C++ programs, you must include file “abax.h” in your source programs and link your executale program with “–labax”. During the installation of Big Data Enabler, header files should have been copied to /usr/local/include directory. Static and dynamic library archives should be in /usr/local/lib directory.

main.cc:

```
#include <abax.h>
int main (int argc, char *argv[] )
{
    …
}
```

Compile:

```
g++ -I/usr/local/include  -o myprog main.cc –L/usr/local/lib –labax –lm
```


Programming Guide

Big Data Enabler will enable you to build powerful and fast programs when you have millions or billions of data to process in memory. Big Data Enabler consists of these container classes as building blocks: AbaxMap, AbaxMultiMap, AbaxSet, AbaxMultiSet, AbaxHashMap, AbaxHashSet, AbaxHashList, AbaxCounter, and AbaxGraph.

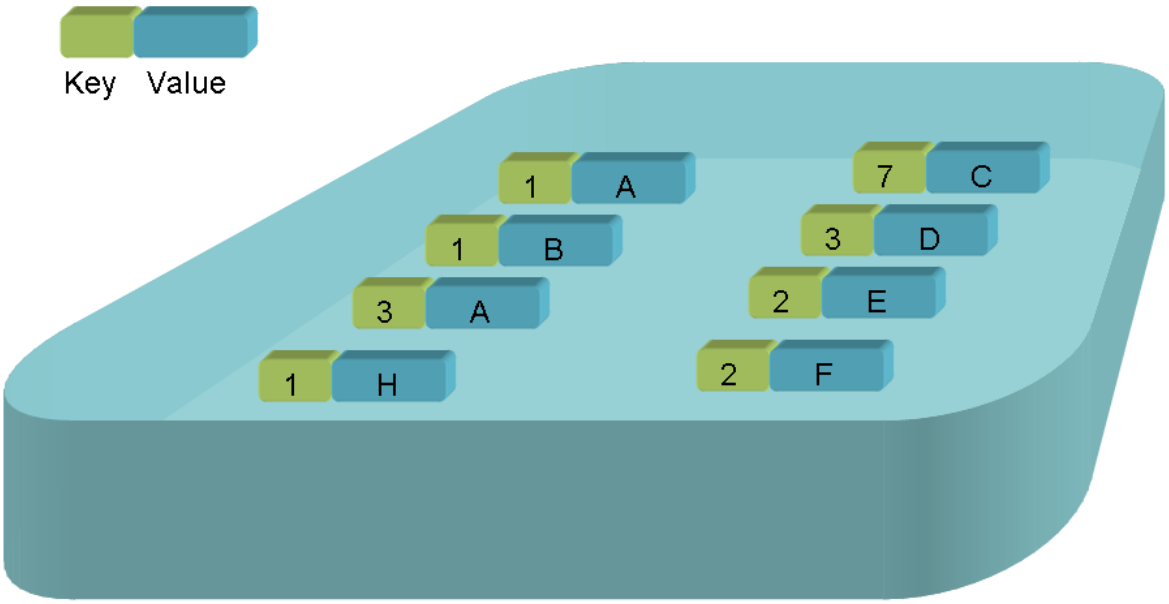
AbaxMap

From AbaxMap class you can create containers, or dictionaries, that manage key-value pairs as elements. AbaxMap does not allow duplicate keys. If a key is already stored in a map, adding a key-value pair with the same key has no effect and is ignored.

In the following examples, AbaxInt denotes the type of unsigned integer. The class type for keys is AbaxInt, so is the type for values.

```
#include <iostream>
#include <abax.h>
int main( int argc, char *argv[] )
{
    AbaxMap<AbaxInt, AbaxInt>  map;
```


// insert key value pairs into map



AbaxMap

AbaxInt key = 3839492;

AbaxInt value = 18383930;

map.addKeyValue(key, value);

key = 38384842;


```
value = 238383890;
map.addKeyValue( key, value );

key = 5009383;
value = 18384742;
map.addKeyValue( key, value );

// modify the value corresponding to a key
value = 28283832;
map.setValue( key, value );

// delete a key-value pair from map
map.removeKey( key );

// test whether a key-pair value exist in map
bool rc = map.keyExit( key );
if ( ! rc ) {
   cout << " Key " << key << " is not found in map" << endl;
}

// retrieve the value of a key from the map
if ( map.getValue( key, value ) ) {
   cout << " Key " << key << " has value: " << value << endl;
} else {
```



```
        cout << " Key " << key << " is not found in map" << endl;
    }

    // Get the total number of keys in the map
    unsigned long  len = map.size();
    cout << "There are total=" << len << " keys in the map" << endl;
    return 0;
}
```

AbaxMap Iterators

You can use basic or raw iterators and sorting iterators to traverse elements in an AbaxMap object. Basic iterators are fast, but it does not provide strict order for keys. It generates list of elements in only partial order. The keys in the list tend to increase or decrease in value, but consecutive keys are not guaranteed to be in correct order. It, however, has the advantage of best speed performance.

Forward Iterator

```
    // basic forward iterator
    AbaxMapIterator<AbaxInt,AbaxInt> iter( &map );
    while ( iter.hasNext() ) {
        AbaxKeyValue<AbaxInt,AbaxInt> pair = iter.next();
```



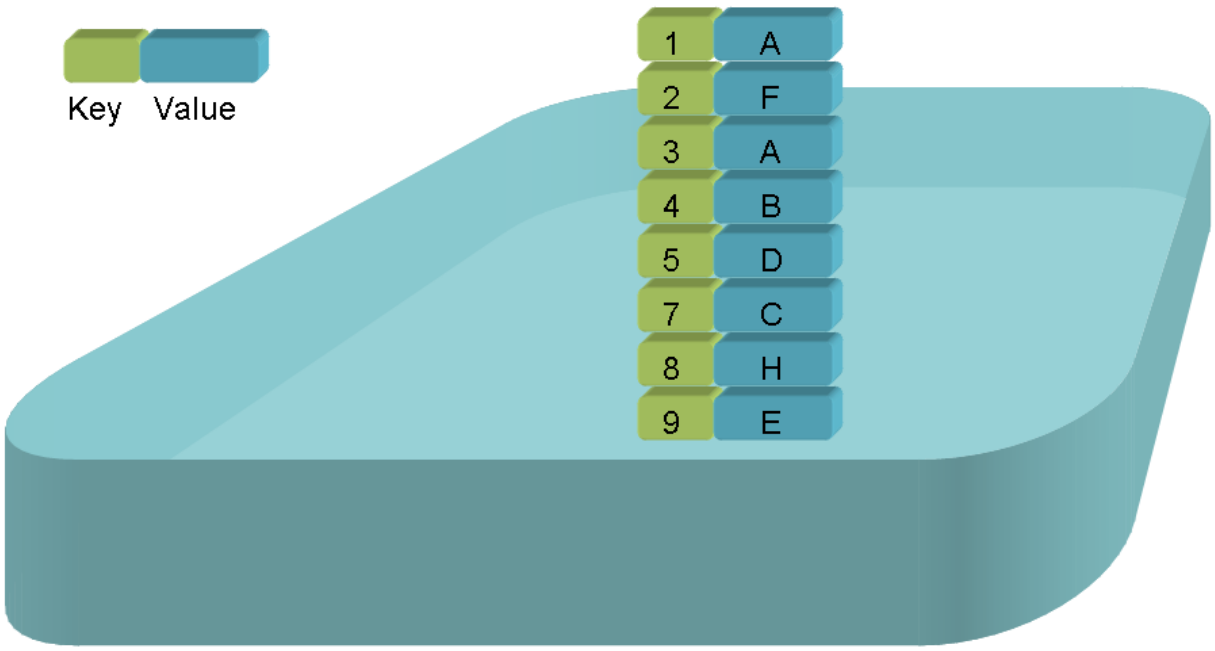
```
    abaxcout << "map iterator key=" << pair.key << " value=" << pair.value << abaxendl;
}

// iterate only 10 keys and then stop:
AbaxMapIterator<AbaxInt,AbaxInt> iter( &map, 10 );
while ( iter.hasNext() ) {
    AbaxKeyValue<AbaxInt,AbaxInt> pair = iter.next();
    cout << "map iterator key=" << pair.key << " value=" << pair.value << endl;
}
```

The iterator reads data in AbaxMap with sorted key order.

Reverse Iterator

```
// basic reverse iterator
AbaxMapReverseIterator<AbaxInt,AbaxInt> iter( &map );
while ( iter.hasNext() ) {
    AbaxKeyValue<AbaxInt,AbaxInt> pair = iter.next();
    cout << "reverse iter key=" << pair.key << " value=" << pair.value << endl;
}
```

When an iterator is active, the map should not be modified. That is, new elements should not be added and elements should not be removed. Otherwise you may see undefined behavior.



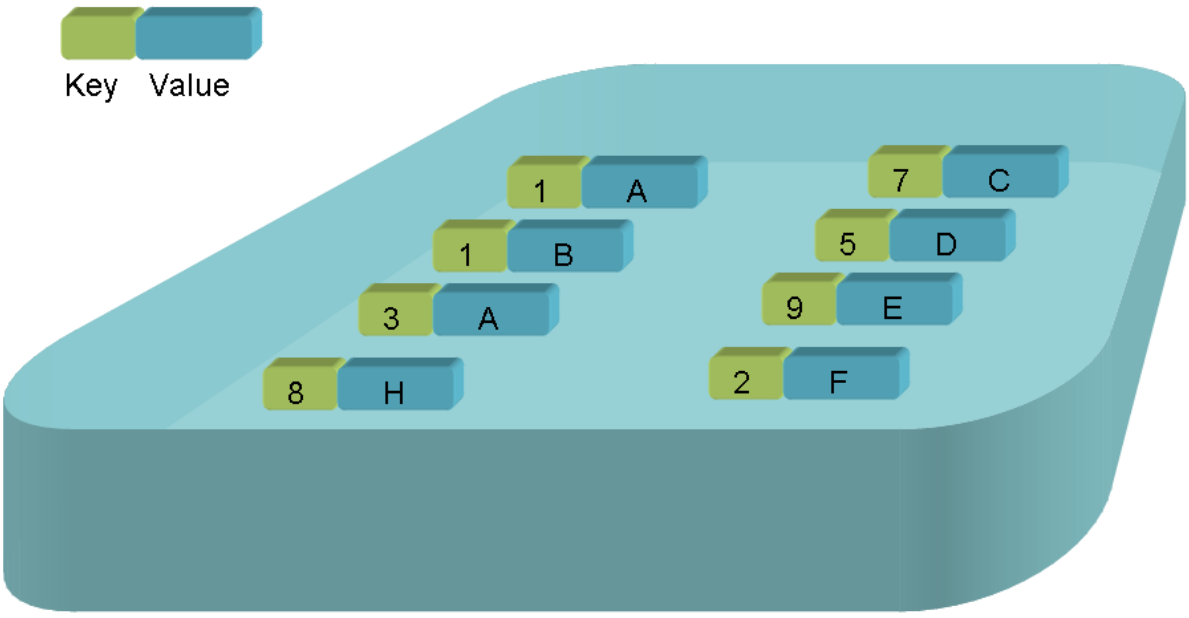
AbaxMap

AbaxMultiMap

AbaxMultiMap is a map object which can store multiple values associated with a given key. In other words, it allows duplicates unlike AbaxMap.

The following examples will show you how to program with AbaxMultiMap.

```
AbaxMultiMap<AbaxInt, AbaxInt>  mmap;
AbaxInt key = 1033940;
AbaxInt value = 10436400;
mmap.addKeyValue( key, value );

value = 2087600;
mmap.addKeyValue( key, value );

value = 302323200;
mmap.addKeyValue( key, value );
```

AbaxMultiMap

So far key 1033940 is associated with three values: 10436400, 2087600, and 302323200. You can replace one value associated with the key with a new value:

```
mmap.setKeyValue(key,  302323200, 402323200 );
```

Now key 1033940 is associated with values 10436400, 2087600, and 402323200

You can delete one value for the key:

```
value = 2087600;
mmap.removeKeyValue( key, value );
```

Now key 1033940 is associated with just two values: 10436400 and 402323200.

You can call this method to remove a key all its associated values from the map:

```
mmap.removeAllKeyValues( key );
```

To check whether a key-value pair exists in the map, you can use this method:

```
bool b = mmap.keyValueExist( key, 2300 );
```

Since multiple values can be associated with a key, how can you retrieve all its values? You can use AbaxList class to get the values:

```
mmap.addKeyValue( key, value );
mmap.addKeyValue( key, value );
mmap.addKeyValue( key, value );
mmap.addKeyValue( key, value );
```



```
AbaxList<AbaxInt> list;
if ( matw.getKeyValues( key, list ) ) {
    AbaxListIterator<AbaxInt> iter( &list );
    while ( iter.hasNext() ) {
        const AbaxInt &val = iter.next();
        cout << "multimap list iterator multivalue key=" << key;
        cout <<  " value=" << val << endl;
    }
}
```

Note that the class parameter to AbaxList is AbaxInt which is exactly the class type of values in the map.

AbaxMultiMap Iterators

You can find the following iterators of AbaxMultiMap similar to AbaxMap: basic forward iterator, basic reverse iterator, sorting forward iterator, and sorting reverse iterator.

AbaxMultiMap Iterator

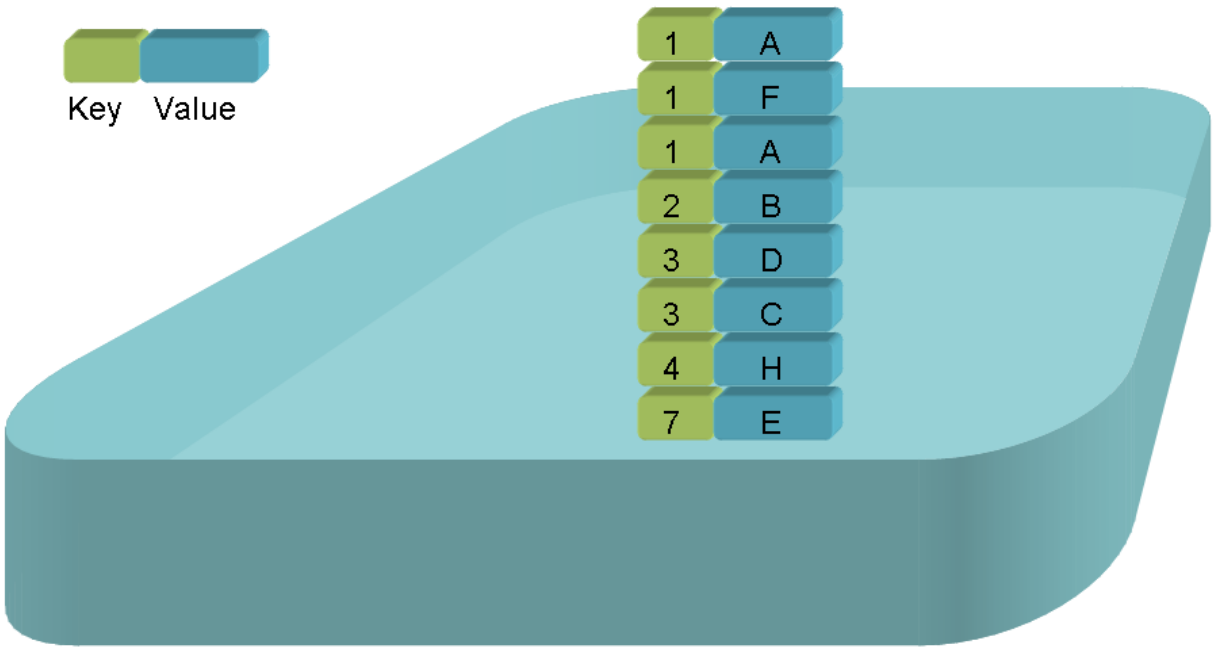
```
AbaxMultiMapIterator<AbaxInt,AbaxInt> iter( &mmap );
while ( iter.hasNext() ) {
```



```
    AbaxKeyValue<AbaxInt,AbaxInt> pair = iter.next();
    // you now have access to pair.key and pair.value
}
```

AbaxMultiMap Reverse Iterator

```
AbaxMultiMapReverseIterator<AbaxInt,AbaxInt> iter( &mmap );
while ( iter.hasNext() ) {
    AbaxKeyValue<AbaxInt,AbaxInt> pair = iter.next();
    // you now have access to pair.key and pair.value
}
```

AbaxMultiMap

```
AbaxInt key = 1323200;
AbaxInt value = 1007664540;
mmap.addKeyValue( key, value );

value = 20767600;
mmap.addKeyValue( key, value );
```



```
value = 305378600;
mmap.addKeyValue( key, value );
```

Method removeKeyValue allows you to remove a value for the key:

```
value = 20767600;
mmap.removeKeyValue( key, value );
```

The following example demonstrates delayed or asynchronous sorting when many key-values pairs are added to the map:

```
// batch insertion of key-value pairs
for ( int i = 0; i < 1000000; ++i ){
   key = someKey();
   value = someValue();
   mmap.addKeyValue( key, value );
   mmap.addKeyValue( key, value );
   mmap.addKeyValue( key, value  );
}
```

When a new element is inserted into the map container, order of the keys are maintained. At the same time, a hash table is built and maintained so that look up of data items is achieved with O(1) speed.

A key and all its values can be removed from the map:

```
mmap.removeAllKeyValues( key );
```

You can also check whether a key-value exists in the map:

```
bool b = mmap.keyValueExist( key, 23009391 );
```

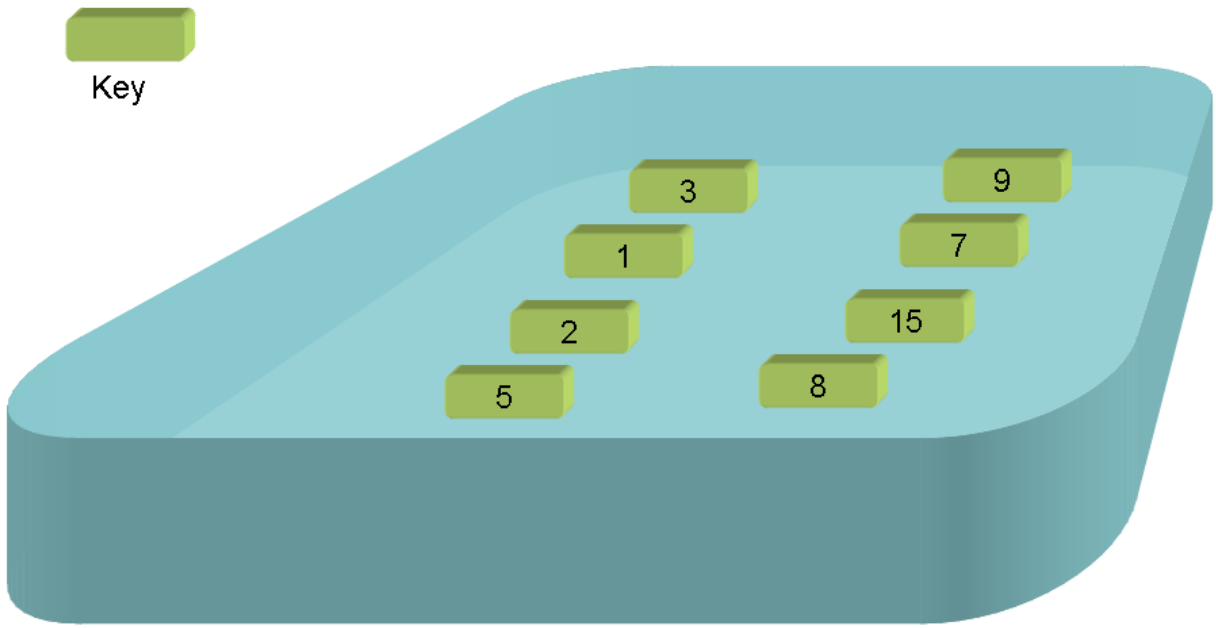
AbaxSet

AbaxSet and AbaxMultiSet store only keys as opposed to AbaxMap and AbaxMultiMap which store pairs of both a key and a value. The set classes use much less memory than the map classes. The following examples demonstrate how to use the set objects.

```
AbaxSet<AbaxString>  set;

AbaxString key = "Smart Phone";

// add the key to the set
```



```
set.addKey( key );
```



AbaxSet

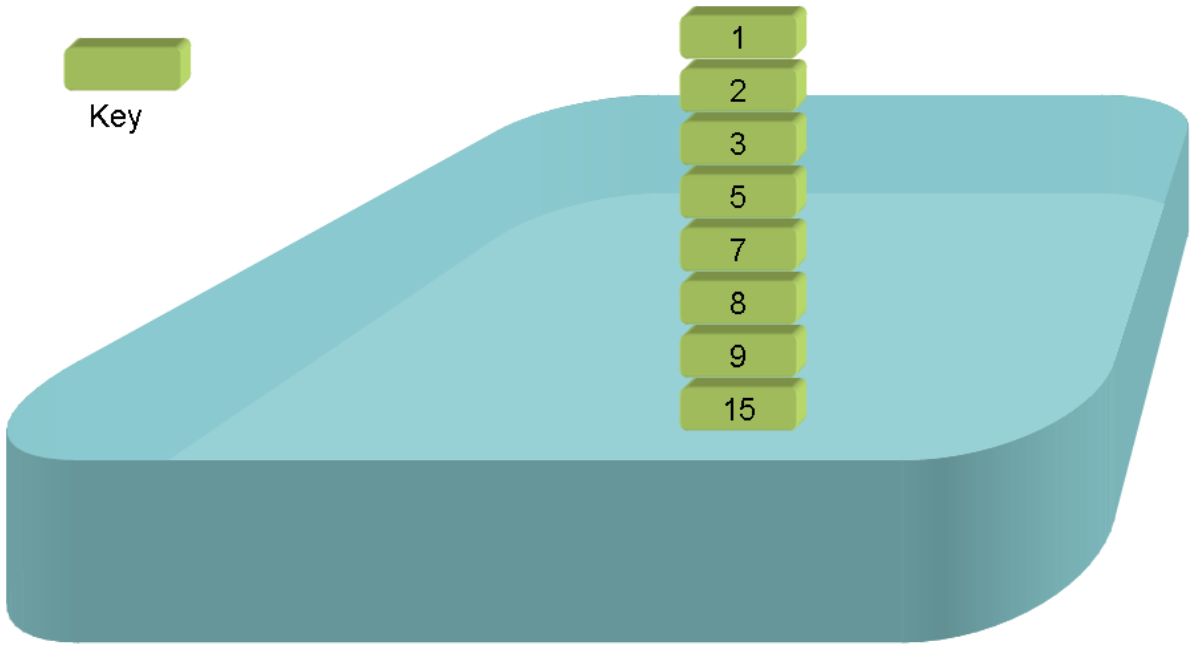
```
// adding twice has no effect, since AbaxSet takes unique keys only
set.addKey( key );

// delete the key from the set
```



```
set.removeKey( key );

bool b = set.keyExist( key );
```

AbaxSet Iterators

```
AbaxSetIterator<AbaxString> iter( &set );
while ( iter.hasNext() ) {
   const AbaxString &key = iter.next();
}

AbaxSetReverseIterator<AbaxString> iter( &set );
while ( iter.hasNext() ) {
   const AbaxString &key = iter.next();
}
```

AbaxSet

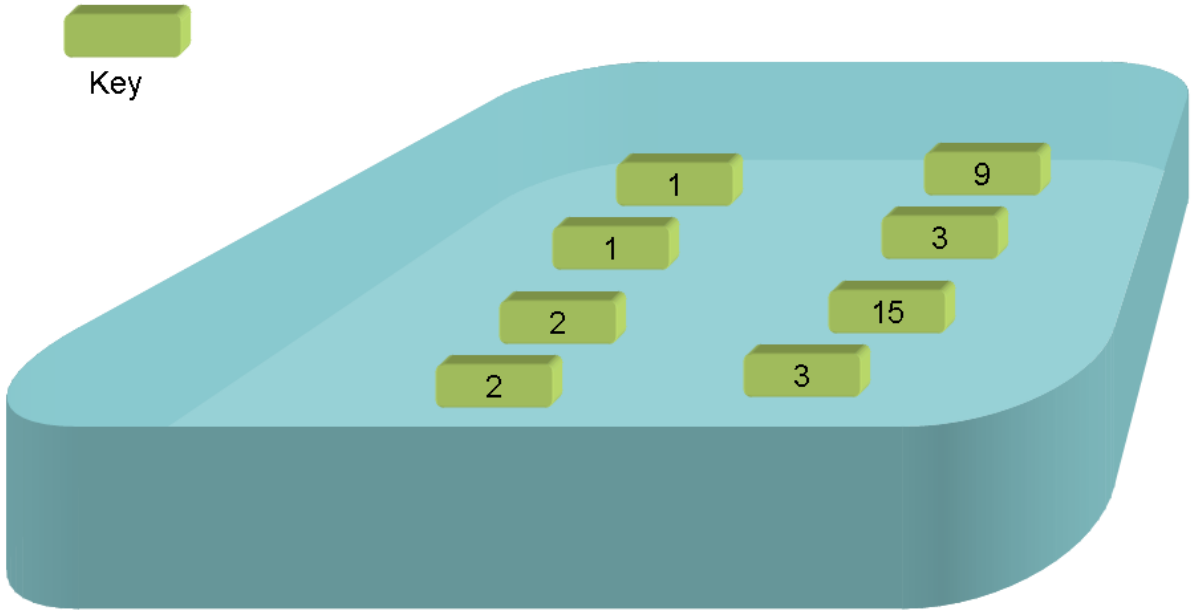
AbaxMultiSet

The difference between AbaxMultiSet and AbaxSet is that the former allows duplicates whereas the latter does not allow duplicates.

```
AbaxMultiSet<AbaxString>  mset;
AbaxString key = "Smart Phone";

// add the key to the set
mset.addKey( key );

// multiset allows two or more same keys in the set
mset.addKey( key );
mset.addKey( key );
mset.addKey( key );

// remove one copy of key from the set
mset.removeKey( key );   // this should be 3 copies left in the set
```



```
unsigned long count = mset.getKeyCount( key );
cout << "count=" << count << endl;

bool b = set.keyExist( key );
```



AbaxMultiSet

AbaxMultiSet Iterators

AbaxMultiSet Iterator

```
AbaxMultiSet<AbaxInt>  mset;

AbaxMultiSetIterator<AbaxInt> iter( &mset );
while ( iter.hasNext() ) {
   const AbaxInt &key = iter.next();
}

AbaxMultiSetIterator<AbaxInt> iter( &mset, 1000 );
while ( iter.hasNext() ) {
   const AbaxInt &key = iter.next();
}
```

AbaxMultiSet Reverse Iterator

```
AbaxMultiSetReverseIterator<AbaxInt> iter( &mset );
while ( iter.hasNext() ) {
   const AbaxInt &key = iter.next();
}
```



```
AbaxMultiSetReverseIterator<AbaxInt> iter( &mset, 30 );
while ( iter.hasNext() ) {
   const AbaxInt &key = iter.next();
}
```

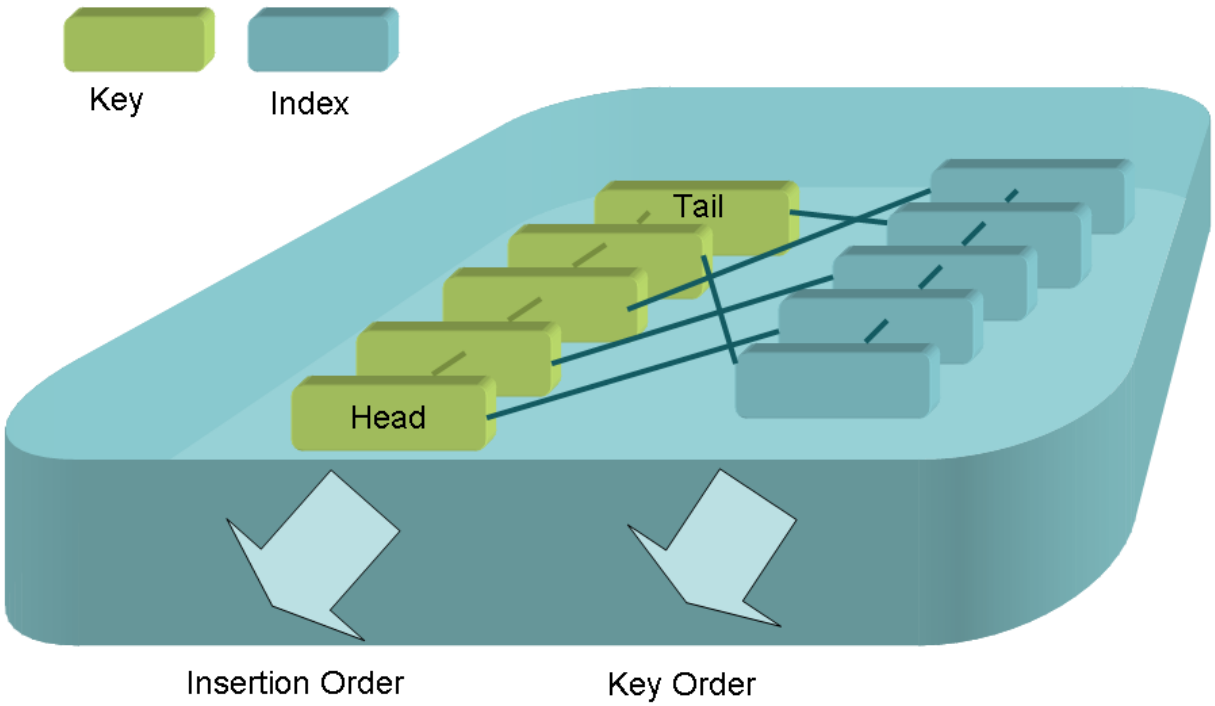
AbaxHashList

In some cases, you may want to maintain the order of insertion and the order of keys in a container. With AbaxHashList, you can add keys at the head or tail of a linked list. Then later, you can use iterators to traverse the list itself or obtain a sequence of keys sorted by either ascending order or descending order.

```
AbaxHashList<AbaxInt>  hashlist;

AbaxInt key = 3;
for ( int i=0; i < 1000000; ++i) {
   key = rand();
   hashlist.addHead( key );
}

// add a key at the tail of the linked list
```



```
key = rand();
hashlist.addTail( key );

// remove a key at the head of the linked list
hashlist.removeHead();

// remove a random key without traversing the whole list
hashlist.removeKey( key );

unsigned long sz = hashlist.size();
cout << "hashlist has " << sz << " keys=" << endl;
```

AbaxHashList

AbaxHashList Iterators

AbaxHashList Iterator

```
AbaxHashListIterator<AbaxInt> iter( &hashlist );
while ( iter.hasNext() ) {
```



```
    const AbaxInt &key = iter.next();
    cout << "hashlist iterator key=" << key << endl;
}
```

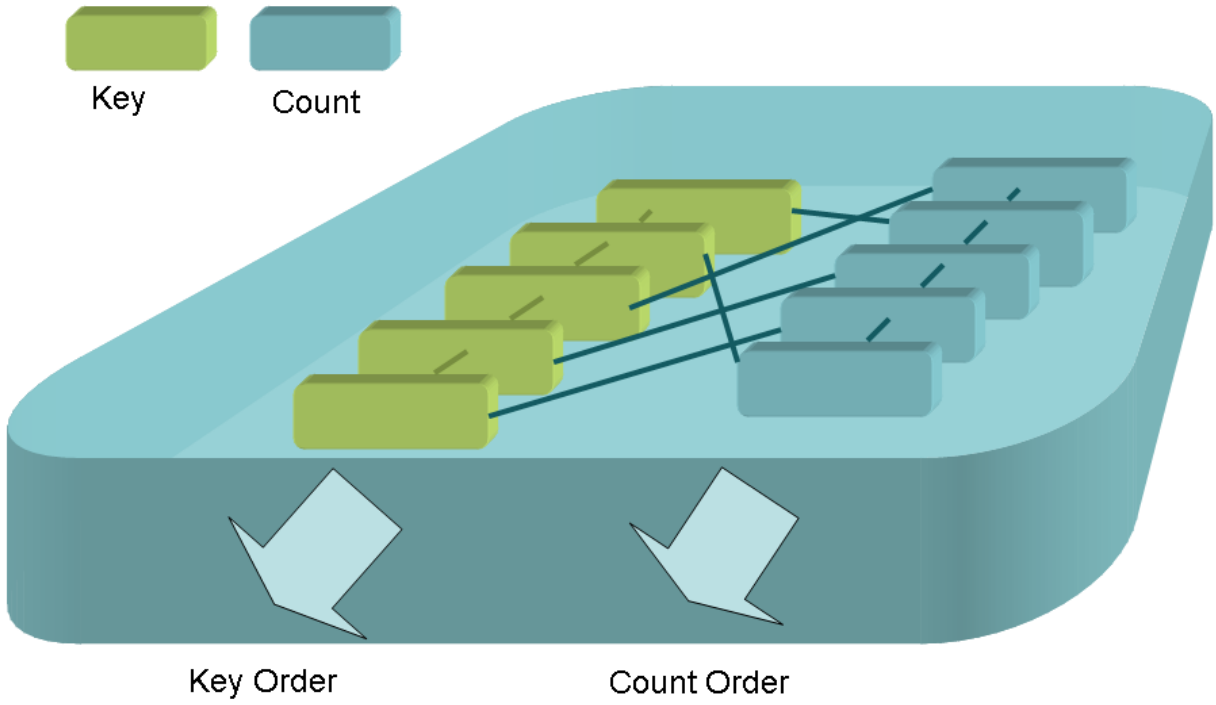
AbaxHashList Reverse Iterator

```
AbaxHashListReverseIterator<AbaxInt> iter( &hashlist, 5 );
while ( iter.hasNext() ) {
    const AbaxInt &key = iter.next();
    cout << "hashlist reverse iterator key=" << key << endl;
}
```

AbaxCounter

As keys or any data items are inserted into a container, you may want to count the total number of replicas of a key in the container. Also you may want to check the top-ranked keys ordered by their number of occurrences. AbaxCounter is such a class that allows for storing, counting, and ranking of keys. As keys are added into an AbaxCounter container, their counts are incremented automatically. You can use their iterators to quickly retrieve the keys by the order of frequency. The following examples demonstrate the abilities of AbaxCounter objects:


```
AbaxString key;
AbaxCounter<AbaxString> counter;

key = "Obama";
```



AbaxCounter


```
counter.addKey( key );
counter.addKey( key );
counter.addKey( key );
counter.addKey( key );

key = "Smart phone";
counter.addKey( key );
counter.addKey( key );
counter.addKey( key );
counter.addKey( key );
counter.addKey( key );

key = "Gas price";
counter.addKey( key );
counter.addKey( key );
counter.addKey( key );

key = "3D TV";
counter.addKey( key );
counter.addKey( key );

key = "Cheap tablet";
counter.addKey( key );
counter.addKey( key );
```



```
counter.addKey( key );
counter.addKey( key );

unsigned long keys = counter.size();
unsigned long allKeys = counter.sizeAll();
 cout << "Number of unique keys=" << keys;
cout << "  Total number of keys=" << allKeys << endl;
```

AbaxCounter Key Iterator

This iterator basically iterates over the keys in the counter object and lists the keys in sorted order.

```
unsigned long count;
AbaxCounterKeyIterator<AbaxString> iter( &counter );
while ( iter.hasNext() ) {
   AbaxString &s = iter.next();
   count = counter.getKeyCount( s );
   cout << "counter  key=" << s << " count=" << count << endl;
}
```


AbaxCounter Count Iterator

This iterator sorts the keys by their count and iterates over the sorted key counts.

```
AbaxCounterCountIterator<AbaxString> iter( &counter, 100 );
while ( iter.hasNext() ) {
    AbaxString &s = iter.next( count );
    cout << "100 least frequent keys  key=" << s
    cout << " count=" << count << endl;
}
```

AbaxCounter Count Reverse Iterator

This iterator starts from the key with the most counts and moves in reverse direction.

```
AbaxCounterCountReverseIterator<AbaxString> iter( &counter, 30 );
while ( iter.hasNext() ) {
    AbaxString &s = iter.next( cnt );
    cout << "30 most frequent keys  key=" << s;
   cout << " count=" << count << endl;
}

// Keys that have occurred at most 900 times and ensuing 30 elements:
AbaxCounterCountReverseIterator<AbaxString> iter( &counter, 900, 30 );
while ( iter.hasNext() ) {
```



```
    AbaxString &s = iter.next( count );
    cout << "counts in a range [900,300] key=" << s;
    cout << " count=" << count << endl;
}
```

AbaxGraph

AbaxGraph container implements the undirected graph and directed graph data structure. A graph contains a set of nodes, a set of edges (links), where an edge has a value and a weight associated with it. AbaxGraph is a dynamic container where nodes and links can be added and removed any time.

```
AbaxGraph<AbaxString, AbaxString, AbxString, AbaxInt>  graph;
AbaxString  key1, key2;
AbaxString  value;
AbaxString  edgeValue;
AbaxInt       edgeWeight;

key1 = “SFO”;
value = “San Francisco Intl Airport”;
graph.addNode( key1, value );
key2 = “PEK”;
value = “Beijing Intl Airport”;
```



```
graph.addNode( key2, value );

edgeValue = “air”;
edgeWeight= 3000;
graph.addLink( key1, key2, edgeValue, edgeWeight );
```

The above sample program demonstrates how nodes and links are added to a graph. AbaxGraph class includes methods for testing whether two nodes are adjacent, removing nodes and links, finding the minimum spanning tree, returning neighbors of a node, etc.

Thread Support

The container classes in Abax Enabler support multithreads by having a method concurrent() which uses mutex for object locking. A read mutex allows multiple concurrent readers and a write mutex blocks all readers and writers.

Reference

Basic Data Types

AbaxLong -- AbaxNumeric<long>

AbaxInt -- AbaxNumeric<int>

AbaxShort -- AbaxNumeric<short>

AbaxFloat -- AbaxNumeric<float>

AbaxDouble -- AbaxNumeric<double>

AbaxChar -- AbaxNumeric<char>

AbaxString -- string as internal data

AbaxBuffer -- void pointer (void*) as internal data

AbaxMap<K,V>

A. Map Class

A template class for maps. K is the class for keys, V is the class for values.

Constructor

AbaxMap( )

Methods

void addKeyValue(const K& key, const V& value)

-- insert a key of class K and a value of class V to the map object.

bool keyExist(const K& key)

-- test whether the key exists in the map object.

bool getValue(const K& key, V &value)

-- read the value corresponding to the key from the map. If the key does not exist in the map, it returns false. Otherwise it returns true, and value contains the value for the key.

bool setValue(const K& key, const V& value)

-- change the old value of key to new value. If key does not exist, it returns false.

bool removeKey(const K& key, AbaxDestroyAction action=ABAX_NOOP)

-- remove key and its value from the map. Parameter *action* specifies how the memory of value will be handled. It can take two values: ABAX_NOOP and ABAX_FREE. By default action is ABAX_NOOP, which means no action is taken. ABAX_FREE will invoke the free() function to free up the memory space used by the value field. ABAX_FREE

must be used when the value type is AbaxBuffer and its memory buffer is allocated by the malloc() function.

unsigned long size()
-- the total number of key-value pairs in the map

void setFormat(const char *format)
-- set the format of key if it is of type AbaxString. This allows the key to have multiple partial keys each of different type. For example, if the format is "%s,%d", then the key has a string part, and a digit part separated by the colon ",".

void concurrent(bool flag)
-- whether the container allows concurrent operations.

B. AbaxMap Iterator

AbaxMapIterator<K,V>
-- A template class for iterator of AbaxMap. K is the class name for the keys, V is the class name for values.

Constructors

AbaxMapIterator(AbaxMap<K,V> *map)

-- basic map iterator which can output all key-value pairs. Keys are not sorted.

AbaxMapIterator(AbaxMap<K,V> *map, unsigned long count)

-- basic map iterator which steps through only count number of key-value pairs and then terminates.

AbaxMapIterator(AbaxMap<K,V> *map, const K& start)

-- basic map iterator which starts at key start inclusive.

AbaxMapIterator(AbaxMap<K,V> *map, const K& start, unsigned long count);

-- basic map iterator which starts at key start and outputs count number of pairs.

AbaxMapIterator(AbaxMap<K,V> *map, const K& start, const K& end)

-- basic map iterator which starts at key start and ends at key specified by end.

Methods

void begin()

-- start the iterator from beginning. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to

rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to doing so by just once.

bool hasNext()
-- check whether iterator has more key-value pairs to retrieve. It checks the starting key, number of pairs, and ending key criteria specified in the iterator constructor.

AbaxKeyValue<K,V> next(AbaxStepAction action)
-- returns the pair, containing the key field and the value field, from the iterator. The key field is a const reference to the internal key data, and value is a regular reference to the value associated to the key. Thus the key field is immutable while the value field is mutable. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the pair and also steps to the next iterating position. If ABAX_NOMOVE is used, only the pair is returned and iterator does not move on to the next position. ABAX_NEXT is the default value for action.

C. AbaxMap Reverse Iterator

AbaxMapReverseIterator<K,V>
-- A template class for iterator of AbaxMap in reverse direction. K is the class name for the keys, V is the class name for values.

Constructors

AbaxMapReverseIterator(AbaxMap<K,V> *map)
-- basic map reverse iterator which can output all key-value pairs. Keys are not sorted and start at approximately high end.

AbaxMapReverseIterator(AbaxMap<K,V> *map, unsigned long count)
-- basic map reverse iterator which steps through only count number of key-value pairs and then terminates.

AbaxMapReverseIterator(AbaxMap<K,V> *map, const K& start)
-- basic map reverse iterator which starts at key start inclusive.

AbaxMapReverseIterator(AbaxMap<K,V> *map, const K& start, unsigned long count);
-- basic map reverse iterator which starts at key start and outputs count number of pairs.

AbaxMapReverseIterator(AbaxMap<K,V> *map, const K& start, const K& end)
-- basic map reverse iterator which starts at key start and ends at key specified by end. In general the value of start should be greater than that of end. However since output list is not sorted, the iterator my still find some pairs for any value of start and end.

Methods

void begin()

-- start the iterator from beginning. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to just one time.

bool hasNext()

-- check whether iterator has more key-value pairs to retrieve. It checks the starting key, number of pairs, and ending key conditions specified in the iterator constructor.

AbaxKeyValue<K,V> next(AbaxStepAction action)

-- returns the pair, containing the key field and the value field, from the iterator. The key field is a const reference to the internal key data, and value is a regular reference to the value associated to the key. Thus the key field is immutable while the value field is mutable. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the pair and also steps to the next iterating position. If ABAX_NOMOVE is used, only the pair is returned and iterator does not move on to the next. ABAX_NEXT is the default value for action.

AbaxMapReverseIterator& operator++()

-- iterator stepping operator. It is invoked by the preincrement operator "++iterator" to move the iterator to the position of next key-value pair. Note that postincrement ++ operator is not provided.

AbaxMultiMap

A. Map Class

A template class for maps. K is the class for keys, V is the class for values.
Multiple key-value pairs that have the same key can be stored in the map.
In other words, keys in the map are not unique.

Constructor

AbaxMultiMap( )

Methods

void addKeyValue(const K& key, const V& value)
-- insert a key of class K and a value of class V to the map object.

bool keyExist(const K& key);
-- check whether the key exists in the map object.

bool keyValueExist(const K& key, const V &value);
-- check whether the key and one of its associated values exists in the map object.

bool getKeyValues(const K& key, AbaxList<V> &list)
-- read all the values corresponding to the key from the map. If the key does not exist in the map, it returns false. Otherwise it returns true, and list will contain all the values associated with the key. You can use AbaxListIterator to iterate over all the values in the list.

bool setKeyValue(const K& key, const V& oldValue, const V& newValue)
-- change one of the old values of key to new value. If key or the oldValue does not exist, it returns false.

bool removeKeyValue(const K& key, const V& value, AbaxDestroyAction action=ABAX_NOOP)
-- remove one value for the key from the map. Parameter *action* specifies how the memory of value will be handled. It can take two values: ABAX_NOOP and ABAX_FREE. By default action is ABAX_NOOP, which means no action is taken. ABAX_FREE will invoke the free() function to free up the memory space used by the value field. ABAX_FREE must be used when the value type is AbaxBuffer and its memory buffer is allocated by the malloc() function. If no more values exist for the key, the key is also removed from the map.

bool removeAllKeyValues(const K& key, AbaxDestroyAction action=ABAX_NOOP)
-- remove the key and all its values from the map. Parameter *action* specifies how the memory of value will be handled. It can take two values: ABAX_NOOP and

ABAX_FREE. By default action is ABAX_NOOP, which means no action is taken. See removeKeyValue() for how to use the action option.

void setFormat(const char *format)
-- set the format of key if it is of type AbaxString. This allows the key to have multiple partial keys each of different type. For example, if the format is "%s,%d", then the key has a string part, and a digit part separated by the colon ",".

void concurrent(bool flag)
-- whether the container allows concurrent operations.

B. AbaxMultiMap Iterator

AbaxMultiMapIterator<K,V>
-- A template class for iterator of AbaxMultiMap. K is the class name for the keys, V is the class name for values.

Constructors

AbaxMultiMapIterator(AbaxMultiMap<K,V> *map)
-- basic map iterator which can output all key-value pairs. Keys are not sorted.

AbaxMultiMapIterator(AbaxMultiMap<K,V> *map, unsigned long count)

-- basic map iterator which steps through only count number of key-value pairs and then terminates.

AbaxMultiMapIterator(AbaxMultiMap<K,V> *map, const K& start)
-- basic map iterator which starts at key start inclusive.

AbaxMultiMapIterator(AbaxMultiMap<K,V> *map, const K& start, unsigned long count);
-- basic map iterator which starts at key start and outputs count number of pairs.

AbaxMultiMapIterator(AbaxMultiMap<K,V> *map, const K& start, const K& end)
-- basic map iterator which starts at key start and ends at key specified by end.

Methods

void begin()
-- start the iterator from beginning. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to doing so by just once.

bool hasNext()

-- check whether iterator has more key-value pairs to retrieve. It checks the starting key, number of pairs, and ending key criteria specified in the iterator constructor.

AbaxKeyValue<K,V> next(AbaxStepAction action)
-- returns the pair, containing the key field and the value field, from the iterator. The key field is a const reference to the internal key data, and value is a regular reference to the value associated to the key. Thus the key field is immutable while the value field is mutable. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the pair and also steps to the next iterating position. If ABAX_NOMOVE is used, only the pair is returned and iterator does not move on to the next position. ABAX_NEXT is the default value for action.

C. AbaxMultiMap Reverse Iterator

AbaxMultiMapReverseIterator<K,V>
-- A template class for iterator of AbaxMultiMap in reverse direction. K is the class name for the keys, V is the class name for values.

Constructors

AbaxMultiMapReverseIterator(AbaxMultiMap<K,V> *map)

-- basic map reverse iterator which can output all key-value pairs. Keys are not sorted and start at approximately high end.

AbaxMultiMapReverseIterator(AbaxMultiMap<K,V> *map, unsigned long count)
-- basic map reverse iterator which steps through only count number of key-value pairs and then terminates.

AbaxMultiMapReverseIterator(AbaxMultiMap<K,V> *map, const K& start)
-- basic map reverse iterator which starts at key start inclusive.

AbaxMultiMapReverseIterator(AbaxMultiMap<K,V> *map, const K& start, unsigned long count);
-- basic map reverse iterator which starts at key start and outputs count number of pairs.

AbaxMultiMapReverseIterator(AbaxMultiMap<K,V> *map, const K& start, const K& end)
-- basic map reverse iterator which starts at key start and ends at key specified by end. In general the value of start should be greater than that of end. However since output list is not sorted, the iterator my still find some pairs for any value of start and end.

Methods

void begin()

-- start the iterator from beginning. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to just one time.

bool hasNext()
-- check whether iterator has more key-value pairs to retrieve. It checks the starting key, number of pairs, and ending key conditions specified in the iterator constructor.

AbaxKeyValue<K,V> next(AbaxStepAction action)
-- returns the pair, containing the key field and the value field, from the iterator. The key field is a const reference to the internal key data, and value is a regular reference to the value associated to the key. Thus the key field is immutable while the value field is mutable. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the pair and also steps to the next iterating position. If ABAX_NOMOVE is used, only the pair is returned and iterator does not move on to the next position. ABAX_NEXT is the default value for action.

AbaxMultiMapReverseIterator& operator++()
-- iterator stepping operator. It is invoked by the preincrement operator "++iterator" to move the iterator to the position of next key-value pair. Note that postincrement ++ operator is not provided.

AbaxSet

This container is similar to AbaxMap except that it stores only the key field in each element.

AbaxMultiSet

This container is similar to AbaxMultiMap except that it stores only the key field in each element.

AbaxHashMap

This container is similar to AbaxMap except it functions as a hash table only. Keys are not maintained in sorted order.

AbaxHashSet

This container is similar to AbaxSet except that it functions as a hash table only. Keys are not maintained in sorted order.

AbaxHashList

A. AbaxHashList<K> Class

A template class for hashed linked list. K is the class for keys. Only unique keys can be stored in the hashed list. Keys can be added to the head or tail of the linked list. AbaxHashList objects have the properties of both an AbaxSet and a linked list. It preserves the order of insertion as well as the key order.

Constructor

AbaxHashList()
-- constructor of an AbaxHashList object

Methods

void addHead(const K& key)
-- add a key of class K at the head of the linked list.

void addTail(const K& key)
-- add a key of class K at the head of the linked list.

bool removeHead(const K& key, AbaxDestroyAction action)
-- remove a key of class K at the head of the linked list. It takes constant time. Flag action can be either ABAX_NOOP or ABAX_FREE. If the key type K is AbaxBuffer and the associated memory of K is required to be deallocated, then ABAX_FREE should be used. Otherwise, ABAX_NOOP should be used. By default, the *action* flag takes the value of ABAX_NOOP.

bool removeTail(const K& key, AbaxDestroyAction action)

-- remove a key of class K at the tail of the linked list. It takes constant time. Flag action can be either ABAX_NOOP or ABAX_FREE. If the key type K is AbaxBuffer and the associated memory of K is required to be deallocated, then ABAX_FREE should be used. Otherwise, ABAX_NOOP should be used. By default, the action flag is ABAX_NOOP.

bool removeKey(const K& key)

-- remove a key from the linked list. If the key is not found in the list, it returns false.

bool keyExist(const K& key);

-- test whether the key exists in the linked list.

unsigned long size()

-- number of keys in the hashlist.

B. AbaxHashList Iterator

AbaxHashListIterator<K>

-- A template class for iterator of AbaxHashList. K is the class name for the keys.

Constructors

AbaxHashListIterator(AbaxHashList<K> *hashlist)

-- hashlist iterator which steps through all the keys in the hashlist from head to tail. When the keys are added with addTail(), this iterators outputs the keys in insertion order.

AbaxHashListIterator(AbaxHashList<K> *hashlist, unsigned long count);
-- hashlist iterator similar to the iterator above except it outputs only count keys starting at the head.

Methods

void begin()
-- start the iterator from the head. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to doing so by just once.

bool hasNext()
-- check whether iterator has more keys to retrieve.

const K& next(AbaxStepAction action)
-- returns the next available key from the iterator. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only

the key is returned and iterator does not move to the next position. ABAX_NEXT is the default value for action.

C. AbaxHashList Reverse Iterator

AbaxHashListReverseIterator<K,V>

-- A template class for iterator of AbaxHashList in reverse direction. K is the class name for the keys, V is the class name for values.

Constructors

AbaxHashListIterator(AbaxHashList<K> *hashlist)

-- hashlist iterator which steps through all the keys in the hashlist from head to tail. When the keys are added with addHead(), then this iterators outputs the keys in insertion order.

AbaxHashListIterator(AbaxHashList<K> *hashlist, unsigned long count);

-- hashlist iterator similar to the iterator above except it outputs only count keys starting at the head.

Methods

void begin()

-- start the iterator from the tail. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to just one time.

bool hasNext()

-- check whether iterator has more keys to retrieve.

const K& next(AbaxStepAction action)

-- returns the next available key from the iterator. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only the key is returned and iterator does not move to the next position unless operator++() is called explicitly such as "++myiter;". ABAX_NEXT is the default value for action.

AbaxHashListIterator& operator++()

-- iterator stepping operator. It is invoked by the preincrement operator "++iterator" to move the iterator to the position of next key. Note that postincrement ++ operator is not provided because it is slower than the preincrement operator.

D. AbaxHashList Key Iterator

AbaxHashListKeyIterator<K>

-- A template class for iterator of AbaxHashList sorted by keys. K is the class name for the keys. Keys are sorted in ascending order.

Constructors

AbaxHashListKeyIterator(AbaxHashList<K> *hashlist)

-- hashlist iterator which outputs all keys in ascending order.

AbaxHashListKeyIterator(AbaxHashList<K> *hashlist, unsigned long count);

-- hashlist iterator similar to the iterator above except it outputs only count keys starting at the least key.

AbaxHashListKeyIterator(AbaxHashList<K> *hashlist, const K& start)

-- hashlist iterator which starts at key start inclusive.

AbaxHashListKeyIterator(AbaxHashList<K> *hashlist, const K& start, unsigned long count)

-- hashlist iterator which starts at key start and outputs count number of keys.

AbaxHashListKeyIterator(AbaxHashList<K> *hashlist, const K& start, const K& end)

-- hashlist iterator which starts at key start and ends at key specified by end. The end key must be greater than or equal to the start key.

Methods

void begin()
-- start the iterator from the least key. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to doing so by just once.

bool hasNext()
-- check whether iterator has more keys to retrieve.

const K& next(AbaxStepAction action)
-- returns the next available key from the iterator. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only the key is returned and iterator does not move to the next position. ABAX_NEXT is the default value for action.

E. AbaxHashList Key Reverse Iterator

AbaxHashListKeyReverseIterator<K>

-- A template class for iterator of AbaxHashList sorted by keys in reverse order. K is the class name for the keys.

Constructors

AbaxHashListKeyReverseIterator(AbaxHashList<K> *hashlist)

-- hashlist iterator which outputs all keys in descending order.

AbaxHashListKeyReverseIterator(AbaxHashList<K> *hashlist, unsigned long count);

-- hashlist iterator similar to the iterator above except it outputs only count keys starting at the greatest key.

AbaxHashListKeyReverseIterator(AbaxHashList<K> *hashlist, const K& start)

-- hashlist iterator which starts at key start inclusive.

AbaxHashListKeyReverseIterator(AbaxHashList<K> *hashlist, const K& start, unsigned long count)

-- hashlist iterator which starts at key start and outputs count number of keys.

AbaxHashListKeyReverseIterator(AbaxHashList<K> *hashlist, const K& start, const K& end)

-- hashlist iterator which starts at key start and ends at key specified by end. The start key must be greater than or equal to the end key.

Methods

void begin()
-- start the iterator from the greatest key. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to doing so by just once.

bool hasNext()
-- check whether iterator has more keys to retrieve.

const K& next(AbaxStepAction action)
-- returns the next available key from the iterator. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only the key is returned and iterator does not move to the next position. ABAX_NEXT is the default value for action.

AbaxCounter

A. AbaxCounter<K> Class

A template class for storing, counting, and ranking of multiple keys. It is primarily AbaxMultiSet coupled with auxiliary structure to easily count and sort plural keys. Multiple copies of a key can be stored in an AbaxCounter container.

Constructor

AbaxCounter()
-- constructor for an AbaxCounter object

Methods

void addKey(const K& key)
-- add a key of class K to the counter object.

void removeKey(const K& key)
-- remove an copy of the key from the counter object. It is possible that other copies of the same key still exist in the counter.

void removeAllKeys(const K& key)
-- remove all the copies of key from the counter object.

bool keyExist(const K& key);

-- test whether any key exists in the counter.

unsigned long getKeyCount(const K& key)

-- number count of key in the counter.

unsigned long size()

-- number of unique keys in the counter.

unsigned long sizeAll()

-- number of all key copies, including non-unique keys, in the counter.

void setFormat(const char *format)

-- set the format of key if it is of type AbaxString. This allows the key to have multiple partial keys each of different type. For example, if the format is "%s,%d", then the key has a string part, and a digit part separated by the colon ",".

void concurrent(bool flag)

-- whether the container allows concurrent operations.

B. AbaxCounter Key Iterator

AbaxCounterKeyIterator<K>

-- A basic template class for iterator of AbaxCounter. K is the class name for the keys.

Constructors

AbaxCounterKeyIterator(AbaxCounter<K> *counter)
-- iterator which steps through all the keys in the counter object. Keys are not sorted and start at approximately low end.

AbaxCounterKeyIterator(AbaxCounter<K> *counter, unsigned long count);
-- iterator which steps through only count keys from beginning.

Methods

void begin()
-- start the iterator from the beginning. All the constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to doing so by just once.

bool hasNext()
-- check whether the iterator has more keys to retrieve.

const K& next(AbaxStepAction action)

-- returns the next available key from the iterator. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only the key is returned and iterator does not move to the next position. ABAX_NEXT is the default value for action.

C. AbaxCounter Key Reverse Iterator

AbaxCounterKeyReverseIterator<K>

-- A basic template class for reverse iterator of AbaxCounter. K is the class name for the keys.

Constructors

AbaxCounterKeyReverseIterator(AbaxCounter<K> *counter)

-- iterator which steps through all the keys in the counter object. Keys are not sorted and start at approximately high end.

AbaxCounterKeyReverseIterator(AbaxCounter<K> *counter, unsigned long count);

-- iterator which steps through only count keys from beginning.

Methods

void begin()

-- start the iterator from the beginning. All the constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to doing so by just once.

bool hasNext()

-- check whether the iterator has more keys to retrieve.

const K& next(AbaxStepAction action)

-- returns the next available key from the iterator. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only the key is returned and iterator does not move to the next position. ABAX_NEXT is the default value for action.

D. AbaxCounter Count Iterator

AbaxCounterCountIterator<K>

-- A template class for iterator of AbaxCounter sorted by key counts, the counts for all the keys. K is the class name for the keys. The iterator sorts the counts for all keys and

generates a sequence of key counts. The sorted key count sequence is in ascending order.

Constructors

AbaxCounterCountIterator(AbaxCounter<K> *counter)
-- sorting iterator which steps through all key counts. Key counts are sorted in ascending order.

AbaxCounterCountIterator(AbaxCounter<K> *counter, unsigned long N)
-- sorting iterator which steps through only N number of key counts and then terminates.

AbaxCounterCountIterator(AbaxCounter<K> *counter, unsigned long startCount, unsigned long N)
-- sorting iterator which steps through keys that have counts startCount and ensuing N items.

Methods

void begin()
-- start the iterator from beginning. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to

rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to just one time.

bool hasNext()
-- check whether iterator has more keys to retrieve. It checks the number of keys, and range of count specified in the corresponding iterator constructors.

const K& next(unsigned long &count, AbaxStepAction action)
-- returns the next available key in the counter and the count for the key in the first parameter. The key returned here must not be modified. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only the key is returned and iterator does not move to the next position. ABAX_NEXT is the default value for action.

E. AbaxCounter Count Reverse Iterator

AbaxCounterCountReverseIterator<K>
-- A template class for iterator of AbaxCounter sorted by key counts, the counts for all the keys. K is the class name for the keys. The iterator sorts the counts for all keys and generates a sequence of key counts. The sorted key count sequence is in descending order.

Constructors

AbaxCounterCountReverseIterator(AbaxCounter<K> *counter)
-- sorting iterator which steps through all key counts. Key counts are sorted in descending order.

AbaxCounterCountReverseIterator(AbaxCounter<K> *counter, unsigned long N)
-- sorting iterator which steps through only N number of key counts and then terminates. Key counts are sorted in descending order. This is useful when you want to find the top N keys in the counter.

AbaxCounterCountReverseIterator(AbaxCounter<K> *counter, unsigned long startCount, unsigned long N)
-- sorting iterator which steps through keys that have counts startCount and ensuing N elements.

Methods

void begin()
-- start the iterator from beginning. All the above constructors include this operation, so begin() is not necessary after an iterator is constructed. However, it can be invoked to rewind the iterator back to the beginning by explicitly calling begin(). Invoking multiple times of begin() is equivalent to just one time.

bool hasNext()

-- check whether the iterator has more keys to retrieve. It checks the number of keys, and range of count specified in the corresponding iterator constructors.

const K& next(unsigned long &count, AbaxStepAction action)

-- returns the next available key in the counter and the count for the key in the first parameter. The key returned here must not be modified. The options of the action parameter are either ABAX_NEXT or ABAX_NOMOVE. If ABAX_NEXT is used, the method returns the key and also steps to the next key position. If ABAX_NOMOVE is used, only the key is returned and iterator does not move to the next position. ABAX_NEXT is the default value for action.

AbaxGraph<K,V, EV, EW>

Graph Class

A template class for graphs. K is the class for the key in a node. V is the class for value of the node. EV is the class for an edge in a graph. EW is the class for the weight of an edge in the graph.

Constructor

AbaxGraph(AbaxGraphType type=ABAX_UNDIRECTED)

An graph can be undirected (ABAX_UNDIRECTED) or directed (ABAX_DIRECTED).

Methods

void addNode(const K& key, const V& value)
-- insert a node with a key of class K and a value of class V to the graph object.

bool isAdjacent(const K& key1, const K& key2)
-- test whether node key1 is adjacent to node key2.

bool addLink(const K& k1, const K& k2, const EV &ev, const EW &ew)
--add an edge or link between node k1 and k2, with edge value ev, and edge weight ew.

bool removeNode(const K& key)
-- remove a node and all the links from it to its neighbors.

bool removeLink(const K& key1, const K& key2)
-- remove a link from node key1 to node key2.

long numberOfNodes()

-- the total number of nodes in the graph

long numberOfLinks()

-- the total number of links in the graph

long neighbors(const K& k, AbaxVector<K> &vec, AbaxLinkType type, bool dolock)

-- the list of neighbors of node k saved in vector vec, with link type, lock flag. If dolock is false, no locking is performed for concurrency.

long minSpanTreePrim(AbaxHashSet< AbaxKeyKeyPair<K,EW> > &links, EW &cost)

-- the minimum spanning tree of the graph. Result links are saved in the link hash set. Cost contains the minimum cost of the spanning tree. The methods returns the number of links in the minimum spanning tree. This method uses the Prim's algorithm.

