CPTS 223 Advanced Data Structure C/C++

Tree: Set and Map

Overview

- Tree data structure
- Binary search trees
  - Support O(lg(n)) operations
  - Balanced trees
- STL set and map classes
- B-trees for accessing secondary storage
- Applications of Tree

Balanced BSTs

- AVL trees
  - Guarantees O(log2 N) behavior
  - Requires maintaining height information
- Splay trees
  - Guarantees amortized O(log2 N) behavior
  - Moves frequently-accessed elements closer to root of tree
- Other self-balancing BSTs:
  - Red-black tree (used in STL)
  - Scapegoat tree
  - Treap
- All these trees assume N-node tree can fit in main memory

Balanced BSTs in STL

- vector and list STL classes inefficient for search
- STL set and map classes guarantee logarithmic insert, delete and search
- Internally use a red-black tree

STL set class

- STL set class is an ordered container that does not allow duplicates
- Like lists and vectors, sets provide iterators and related methods: begin, end, empty and size
- Sets also support insert, erase and find

STL set class

- insert adds an item to the set and returns an iterator to it
- Because a set does not allow duplicates, insert may fail
  - In this case, insert returns an iterator to the item causing the failure
  - (if you want duplicates → use multiset)
- To distinguish between success and failure, insert returns a pair of results
  - This pair structure consists of an iterator and a Boolean indicating success
- `pair<iterator,bool> insert (const Object & x);`

STL pair class

- pair<Type1,Type2>

```
#include <utility>

pair<iterator,bool> insert (const Object & x)
{
   iterator itr;
   bool found;
   …
   return pair<itr,found>;
}
```

STL set class

```
set<int> s;
//insert
for (int i = 0; i < 1000; i++) {
  s.insert(i);
}

// print

iterator<set<int>> it=s.begin();
for(it=s.begin(); it!=s.end();it++) {
  cout << *it << " " << endl;
}
```

What order will the elements get printed?

STL set class

```
set<int> s;
//insert
for (int i = 0; i < 1000; i++) {
  s.insert(i);
}

// print

iterator<set<int>> it=s.begin();
for(it=s.begin(); it!=s.end();it++) {
  cout << *it << " " << endl;
}
```

What order will the elements get printed?

Sorted order: iterator does an in-order traversal

Set insert

- Giving insert a hint

```
pair<iterator,bool> insert(iterator hint, const Object & x);
```

- For good hints, insert is O(1)
- Otherwise, reverts to one-parameter insert
- For example:

```
set<int> s;
for (int i = 0; i < 1000000; i++)
  s.insert(s.end(), i);
```

hint: represents the position where x should go

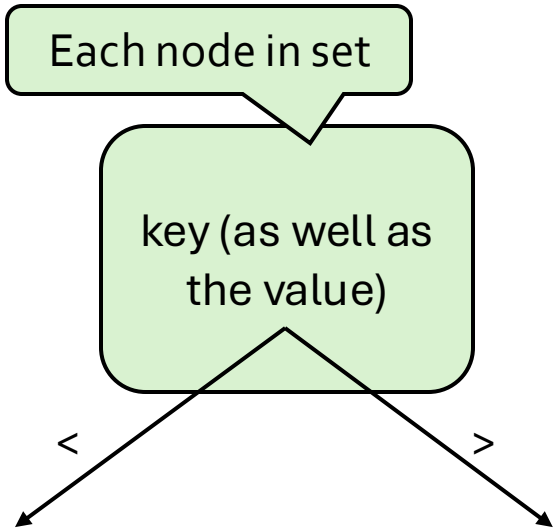
Set delete

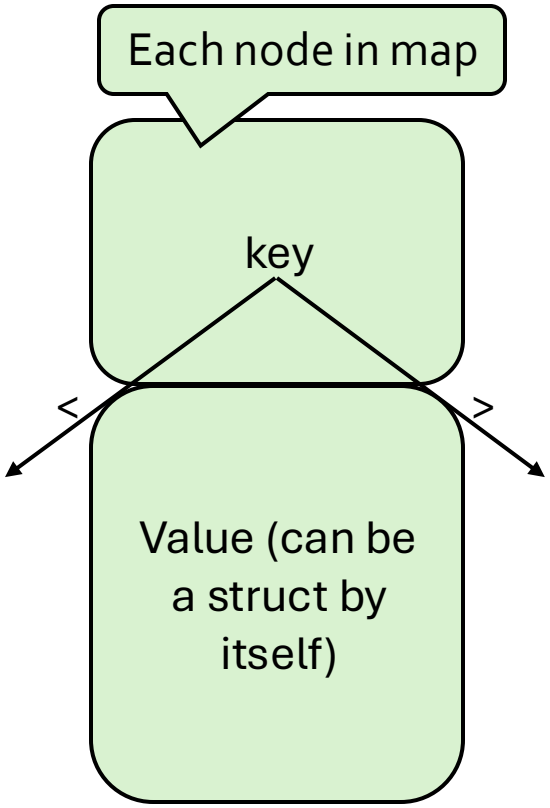
- int erase (const Object & x);
  - Remove x, if found
  - Return number of items deleted (0 or 1)
- iterator erase (iterator itr);
  - Remove object at position given by iterator
  - Return iterator for object after deleted object
- iterator erase (iterator start, iterator end);
  - Remove objects from start up to (but not including) end
  - Returns iterator for object after last deleted object
  - Again, iterator advances from start to end using in-order traversal

STL map class

- Associative container
  - Each item is 2-tuple: [Key, Value]
- STL map class stores items sorted by Key
- set vs. map:
  - set == map with no value (where key is the whole record)
- Keys must be unique (no duplicates)
  - If you want duplicates → mulitmap
- Different keys can map to the same value
- Key type and Value type can be totally different

STL set and map classes





STL map class

- Methods
  - begin, end, size, empty, insert, erase, find
- Iterators reference items of type

  `pair<KeyType,ValueType>`
- Inserted elements are also of type

  `pair<KeyType,ValueType>`

STL map class

- Main benefit: overloaded operator[]

```
ValueType & operator[] (const KeyType & key);
```

- If key is present in map
  - Returns reference to corresponding value
- If key is not present in map
  - Key is inserted into map with a default value
  - Returns reference to default value

```
map<string,double> salaries;
salaries["Pat"] = 75000.0;
```

STL map example

```
struct ltstr
{
  bool operator()(const char* s1, const char* s2) const {
    return strcmp(s1, s2) < 0;
  }
};

int main()
{
  map<const char*, int, ltstr> months;
  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
}
```

Key type

Value type

Comparator if key type not primitive

- You really do not have to call insert() explicitly.
- This syntax will do it for you.
- If element already exists, then value will be updated.

STL map example

```
...
months["may"] = 31;
months["june"] = 30;
...
months["december"] = 31;

cout << "february -> " << months["february"] << endl;
map<const char*, int, ltstr>::iterator cur = months.find("june");
map<const char*, int, ltstr>::iterator prev = cur;
map<const char*, int, ltstr>::iterator next = cur;
++next; --prev;
cout << "Previous (in alphabetical order) is " << (*prev).first << endl;
cout << "Next (in alphabetical order) is " << (*next).first << endl;

months["february"] = 29;
cout << "february -> " << months["february"] << endl;
```

What will these two lines do?

Implementation of set and map

- Support insertion, deletion and search in worst-case logarithmic time
- Use balanced binary search tree (a red-black tree)
- Support for iterator
  - Tree node points to its predecessor and successor
  - Which traversal order?

When to use set and map

- set
  - Whenever your entire record structure to be used as the Key
  - e.g., to maintain a searchable set of numbers
- map
  - Whenever your record structure has fields other than Key
  - e.g., employee record (search Key: ID, Value: all other info such as name, salary, etc.)

Summary: trees so far

- Trees are ubiquitous in software
- Search trees important for fast search
  - Support logarithmic searches
  - Must be kept balanced (AVL, Splay, B-tree to be seen soon)
- STL set and map classes use balanced trees to support logarithmic insert, delete and search
  - Implementation uses top-down red-black trees (not AVL) – Chapter 12 in the textbook