Chapter 21: Algorithms and Maps

Plan for today

- We will talk about:
  - Associative containers:
    - map,
    - set,
    - unordered_map
  - Standard algorithms
    - copy, sort, …
    - Input iterators and output iterators

Map (an associative array)

- For a vector, we subscript using an integer
- For a map, we can define the subscript to be (just about) any type

Map (an associative array)

- For a vector, we subscript using an integer
- For a map, we can define the subscript to be (just about) any type

Key type

Value type

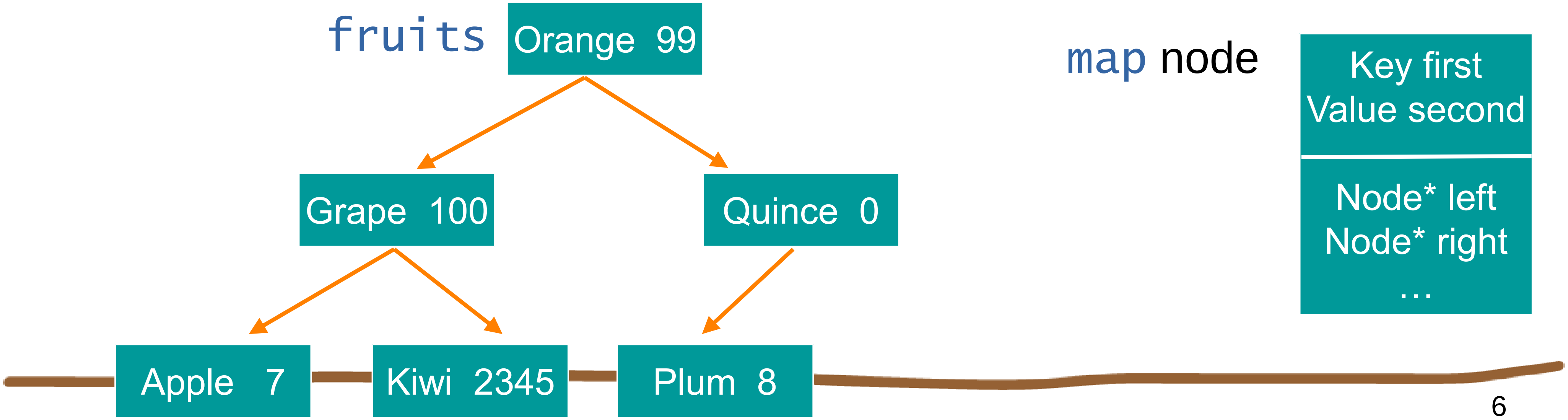
```
int main()
{
    map<string,int> words;          // keep (word,frequency) pairs
    for (string s; cin>>s; )
        ++words[s];                // note: words is subscripted by a string
                                   // words[s] returns an int&
                                   // the int values are initialized to 0
    for (const auto&  p : words)
            cout << p.first << ": " << p.second << "\n";
}
```

Map

- After `vector`, `map` is the most useful standard library container
  - Maps (and/or hash tables) are the backbone of scripting languages
- A `map` is really an *ordered balanced binary tree*
  - By default ordered by `<` (less than)

Map

- After `vector`, `map` is the most useful standard library container
  - Maps (and/or hash tables) are the backbone of scripting languages
- A `map` is really an *ordered balanced binary tree*
  - By default ordered by `<` (less than)
  - For example, `map<string,int> fruits;`



Map

// do you see some similarity to vector and list?

```
template<class Key, class Value> class map {
    // ...
    using value_type = pair<Key,Value>; // a map deals in (Key,Value) pairs

    using iterator = ???;                  // probably a pointer to a tree node
    using const_iterator = ???;

    iterator begin();    // points to first element
    iterator end();      // points to one beyond the last element

    Value& operator[ ](const Key&);   // get Value for Key
    iterator find(const Key& k);       // is there an entry for k?
    void erase(iterator p);      // remove element pointed to by p
    pair<iterator, bool> insert(const value_type&); // insert new
                        // (Key,Value) pair,  the bool is false if insert failed
    // …
};
```

Map

- Let’s see some work in mapExamples.cpp

Containers and “almost containers”

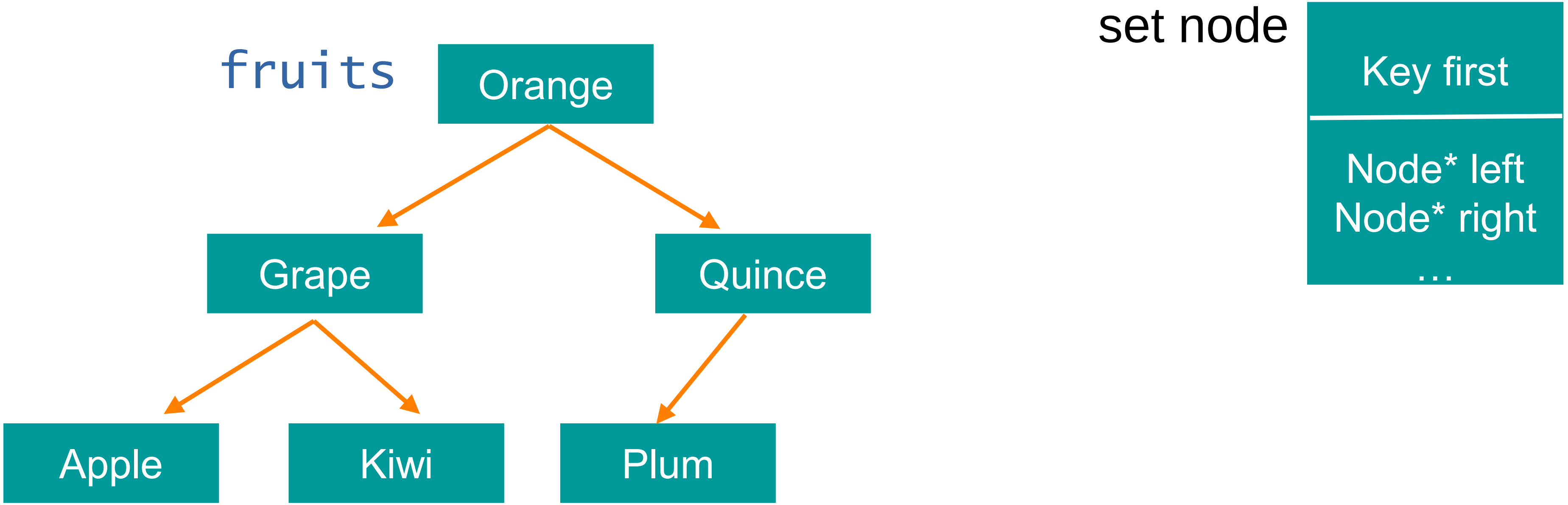
- Sequence containers
  - vector, list, deque
- Associative containers
  - map, set, multimap, multiset
- “almost containers”
  - array, string, stack, queue, priority_queue, bitset
- New C++11 standard containers
  - unordered_map (a hash table), unordered_set, …

Containers and “almost containers”

- For anything non-trivial, consult documentation
  - Online
    - SGI, RogueWave, Dinkumware
  - Other books
  - Stroustrup: The C++ Programming language 4th ed. (Chapters 30-33, 40.6)
  - Austern: Generic Programming and the STL
  - Josuttis: The C++ Standard Library

Set

- A `set` is really an *ordered balanced binary tree*
  - By default ordered by `<`
  - For example, `set<string> fruits;`



Set

- Sets are useful for checking if a value is present
  - e.g., keeping a track of which fruits are available
- Sets do not support subscripting (operator[ ]), nor push_back()
- Use “list operations”:
  - insert()
  - erase()
- We can use the value obtained from the iterator directly

  (since there is no <key,value> pair)
- See an example of its use in setExample.cpp

unordered_map

- `unordered_map` is using hash table to have fast access
  - (look up is O(1))
  - The elements are not ordered
- Very useful if a lot of lookup is projected in a large map, and we don’t need an ordered traversal
- The use is similar to that of `map`
- Python `dict` and C++ `unordered_map` are similar (modulo type of elements)

Some useful standard algorithms

- `r = find(b,e,v)`
  - r points to the first occurrence of v in [b,e)
- `r = find_if(b,e,p)`
  - r points to the first element x in [b,e) for which p(x)
- `x = count(b,e,v)`
  - x is the number of occurrences of v in [b,e)
- `x = count_if(b,e,p)`
  - x is the number of elements in [b,e) for which p(x)
- `sort(b,e)`
  - sort [b,e) using <
- `sort(b,e,p)`
  - sort [b,e) using p

Some useful standard algorithms (continues)

- `copy(b,e,b2)`
  - copy [b,e) to [b2,b2+(e-b)); there had better be enough space after b2
- `unique_copy(b,e,b2)`
  - copy [b,e) to [b2,b2+(e-b)), but don’t copy adjacent duplicates
- `merge(b,e,b2,e2,r)`
  - merge two sorted sequence [b2,e2) and [b,e) into [r,r+(e-b)+(e2-b2))
- `r = equal_range(b,e,v)`
  - r is the subsequence of [b,e) with the value v (basically a binary search for v)
- `equal(b,e,b2)`
  - do all elements of [b,e) and [b2,b2+(e-b)) compare equal?

copy_if()

// a very useful algorithm (missing from the standard library):

```
template<class In, class Out, class Pred>
Out copy_if(In first, In last, Out res, Pred p)
   // copy elements that fulfill the predicate
{
   while (first != last)
   {
      if (p(*first))
         *res++ = *first;
      ++first;
   }
   return res;
}
```

copy_if()

```
// example of copy_iff() use:

void f(const vector<int>& v)   // “typical use” of predicate with data
                               // copy all elements with a value less than 6
{
   vector<int> v2(v.size());

   copy_if(v.begin(), v.end(), v2.begin(),
         [](int x) { return x<6; } );
   // …
}
```

In-class work

Let’s write a program that given a vector of integer values, that are ages of people, will tell us how many people of each age mentioned in the vector there are.
We were told that the vector values are not sorted.
We are asked not to use the sorting procedure, as it is “too expensive” (usually O(n log n)).
Instead they want us to “walk through the values of the vector only once.

Resources used for these slides

- slides provided by B. Stroustrup at https://www.stroustrup.com/PPP2slides.html
- Class textbook