Searching, Maps, Tables

- Searching is a fundamentally important operation
  - We want to do these operations quickly
  - Consider searching using google.com, altavista.com, etc.,
  - In general we want to search in a collection for a key

- We’ve seen searching in context of the MultiSet class
  - Tree implementation was quick
  - Table implementation wasn’t bad, how to make it better?

- If we compare keys, we cannot do better than log n to search n elements
  - Lower bound is \( \Omega(\log n) \), provable
Log (google) is a big number

- Comparison based searches are too slow for lots of data
  - How many comparisons needed for a billion elements?
  - What if one billion web-pages indexed?

- Hashing is a search method that has average case $O(1)$ search
  - Worst case is very bad, but in practice hashing is good
  - Associate a number with every key, use the number to store the key
    - Like catalog in library, given book title, find the book
- A hash function generates the number from the key
  - Efficient to calculate
  - Distributes keys evenly in hash table
Hashing details

● There will be collisions, two keys will hash to the same value
  ➤ We must handle collisions, still have efficient search
  ➤ What about birthday “paradox”: using birthday as hash function, will there be collisions in a room of 25 people?

● Several ways to handle collisions, in general array/vector used
  ➤ Linear probing, look in next spot if not found
    • Hash to index h, try h+1, h+2, ..., wrap at end
    • Clustering problems, deletion problems, growing problems
  ➤ Quadratic probing
    • Has to index h, try h+1^2, h+2^2, h+3^2, ..., wrap at end
    • Fewer clustering problems
  ➤ Double hashing
    • Hash to index h, with another hash function to j
    • Try h, h+j, h+2j, ...
Chaining with hashing

- With n buckets each bucket stores linked list
  - Compute hash value h, look up key in linked list table[h]
  - Hopefully linked lists are short, searching is fast
  - Unsuccessful searches often faster than successful
    - Empty linked lists searched more quickly than non-empty
  - Potential problems?

- Hash table details
  - Size of hash table should be a prime number
  - Keep load factor small: number of keys/size of table
  - On average, with reasonable load factor, search is O(1)
  - What if load factor gets too high? Rehash or other method
Hashing problems

- Linear probing, hash(x) = x, (mod tablesize)
  - Insert 24, 12, 45, 14, delete 24, insert 23

  0   1   2   3   4   5   6   7   8   9  10

- Same numbers, use quadratic probing (clustering better?)

  0   1   2   3   4   5   6   7   8   9  10

- What about chaining, what happens?
What about hash functions

● Hashing often done on strings, consider two alternatives

```cpp
unsigned hash(const string& s)
{
    unsigned int k, total = 0;
    for(k=0; k < s.length(); k++)
    {
        total += s[k];
    }
    return total;
}
```

● What about \( \text{total} += k \times s[k] \), why might this be better?
  ➤ Other functions used, always mod result by table size

● What about hashing other objects?
  ➤ Sometimes address of value used to hash it
Implementation issues

- We want to consider map, table, dictionary (what’s in a name?)
  - STL uses map, Java uses map, we’ll use map
  - What operations should a map support: keys and values
    - •
    - •
    - •
    - •

- What is interface to individual key/value pairs, what are alternatives?
  - What about iterators, what about apply function?
  - If we get a value from the map, can we change it?
  - What about key from map, change it?
Iterating over a map

● Suppose we want to write this code

```cpp
Map<string, int> * m = new HashMap<string, int>();
m->insert("apple", 1);
if (m->contains("apple")) m->get("apple") += 1;
...
Iterator<string> * it = m->makeKeyIterator();
for (it->Init(); it->HasMore(); it->Next())
{
    cout << it->Current() << " "
         << m->get(it->Current()) << endl;
}
```

● What is all this about?
We need a simple map interface

- We map keys->values
  - Insert key/value
  - Get value associated with a key (by reference, why?)
  - Internally combine key/value into a pair
    - <pair.h> is part of STL, standard template library

```cpp
Map<string, int> map;       // this is pseudocode
string word;               // but very close to real
while (input >> word)      // could be map.insert(pair<word, 1>);
{
  if (map.contains(word))
  {
    map.get(word) += 1;
  }
  else
  {
    map.insert(word, 1);
  }
}
```
Some Map details

- As Map is used it’s templated with two parameters
  - tvector, tstack, tqueue, etc., have one template parameter
  - Templated classes aren’t hard to use, but can be hard to develop and debug
    - Errors with basic_string<.....>
    - Develop without templates, then change to templates

- A templated class is a generic class, can store many kinds of object, but some constraints on the object
  - tvector, object must have default constructor and must be assignable, e.g., a = b;
  - Functions and classes can be templated
Selection Sort: The Code (selectsort2.cpp)

void SelectSort(tvector<int> & a)
// pre: a contains a.size() elements
// post: elements of a are sorted in non-decreasing order
{
    int j,k,temp,minIndex,numElts = a.size();
    // invariant: a[0]..a[k-1] in final position
    for(k=0; k < numElts - 1; k++)
    {
        minIndex = k;  // minimal element index
        for(j=k+1; j < numElts; j++)
        {
            if (a[j] < a[minIndex])
            {
                minIndex = j;  // new min, store index
            }
        }
        temp = a[k];  // swap min and k-th elements
        a[k] = a[minIndex];
        a[minIndex] = temp;
    }
}
What changes if we sort strings?

- The parameter changes, the definition of **temp** changes
  - Nothing else changes, code independent of type
  - We can use features of language to capture independence

- We can have different versions of function for different array types, with same name but different parameter lists
  - Overloaded function: parameters different so compiler can determine which function to call
  - Still problems, duplicated code, new algorithm means ...?

- With function templates we replace duplicated code maintained by programmer with compiler generated code
Creating a function template

```cpp
template <class Type>
void SelectSort(tvector<Type> & a) // pre: a contains a.size() elements
   // post: elements of a are sorted in non-decreasing order
{
   int j,k,minIndex,numElts = a.size();
   Type temp;
   // invariant: a[0]...a[k-1] in final position
   for(k=0; k < numElts - 1; k++)
   {
      minIndex = k; // minimal element index
      for(j=k+1; j < numElts; j++)
      {
         if (a[j] < a[minIndex])
         {
            minIndex = j; // new min, store index
         }
      }
      temp = a[k]; // swap min and k-th elements
      a[k] = a[minIndex];
      a[minIndex] = temp;
   }
}

● When the user calls this code, different versions are compiled
```
Some template details

- Function templates permit us to write once, use several times for several different types of vector
  - Template function “stamps out” real function
  - Maintenance is saved, code still large (why?)

- What properties must hold for vector elements?
  - Comparable using < operator
  - Elements can be assigned to each other

- Template functions capture property requirements in code
  - Part of generic programming
  - Some languages support this better than others
Templated class, .h ok, .cpp ugly

- See tstack.h for example

```cpp
template <class Type>
class tstack
{
    public:
    tstack(); // construct empty stack
    const Type & top() const; // return top element
    bool isEmpty() const; // return true iff empty
    int size() const; // # elements

    void push( const Type & item ); // push item
}
```

- But look at part of stack.cpp, class is templated (ugly?)

```cpp
template <class Type> bool
tstack<Type>::isEmpty() const
{
    return myElements.size() == 0;
}
```
Using templated classes

- **Client code includes (typically) only .h file**
  - Why is this a good idea?
  - Is foo.h included in foo.cpp? Why?

- **Template .cpp file is NOT code, it’s a code generator/template**
  - When template is *instantiated* by client, code is generated
  - To instantiate, need access to source templated
  - Templated foo.h typically has `#include “foo.cpp”`
    - Why is this better in foo.h than in client program?