# CS35X: Competitive Programming

## Lecture 4: Dictionaries

Joshua Brody

# Problem debrief:  icoudlhavewon

# STL pair class

- **maintain two things at once!**

```
#include <pair>                          // include pair library
pair<string,int> foo;                    // create pair object
foo.first = “hello!”;                    // set first item to “hello!”
foo.second = 17;                         // set second item to 17;
pair<int, string> bar(44,”science”);  // direct initialization
```

# more C++ math

- `#include <cmath>` `// include math library`
- sample operations: `sqrt, pow, ceil, round`
- Big integers: use `unsigned long long` data type
- How big can `int/long/unsigned long long` be?
  - `#include <climits>`
  - `INT_MIN, INT_MAX` // smallest/largest int
  - `LONGLONG_MAX` // largest unsigned long long

# Dictionary ADT

- Maintain collection of (***key***, ***value***) pairs.
  - keys must be ***unique***.
- Support the following operations:
  - Initialize an empty dictionary.
  - **Insert** a new (***key***, ***value***) pair.
  - Given a key, update its value.
  - Given a key, **get** and return its value.
  - Check to see if a key is present in the dictionary.
  - **Remove** a (***key***, ***value***) pair from dictionary.
- CS35 implementations: **BST** *(weeks 8-9)*, **Hash Table** *(week 11)*

# Example Syntax

```
#include <map>
map<string,int> my_dict;          // create empty dictionary
my_dict[“a”] = 1;                 // insert (“a”,1) into my_dict
my_dict[“a”] = 2;                 // update value of “a” to 2
int a_val = my_dict[“a”];         // use array-like syntax to get values!
if(my_dict.count(“b”)) {          // returns 1 if “b” in dictionary; 0 otherwise.
    my_dict[“b”] +=2;
}
my_dict.erase(“a”);               // delete (key,value) pair associated with “a
for(const auto &mypair:my_dict) { // iterate over (key,value pairs)
   cout << my pair.first << end;
}
```

# Implementation Details

- Definitely use built-in dictionaries!
- Dictionaries can be implemented by a hash map or binary search tree
- **Hashmap** aka **hash table**:
  - Assigns each key to an array index based on a ***hash function***.
  - **O(1)** time operations in practice.
  - C++ STL class: `unordered_map`
- Binary Search Tree aka BST:
  - Arranges keys in binary tree according to some ordering
  - **O(log n)** time operations for balanced BSTs
  - C++ STL class: `map`
- BSTs support `predecessor` and `successor` ops.
- In practice both data structures are fast. For ICPC probs map sometimes faster

# Exercise: week4/colors

# Practice Contest