CS35X: Competitive Programming

Lecture 3: ADTs, vectors

Joshua Brody

Warmup Kattis Problem: cutthenegativity

Competitive Programming Skill:
Identify the easy problems

Problem debrief: conundrum

Abstract Data Types

An Abstract Data Type (ADT) is a specification for how a data structure should operate without specifying the implementation details.

Advantages:

- Different implementations might be better for different operations
- **Abstraction:** don't need to know implementation details to use data structure

Data Structures for CS35x

- Extensively use off-the-shelf STL implementations for ADTs
- Creating new data structures will be *rare*.

C++ Standard Template Library (STL)

- Collection of existing implementations for most ADTs.
- We will not implement our own data structures.
- CS35X applications: use STL data structures off the shelf.
- Our goal: ***understand portions of each ADT deeply.***
- Note: interfaces for most STL data structures not quite same as CS35 ADTs.

List ADT

- Maintain ordered collection of items L[0…n-1]
- Support insert, deletion, query of items.
- Core List operations:
  - `insertFirst, insertLast`
  - `getFirst, getLast, get(i)`
  - `getSize, isEmpty`
  - `removeFirst, removeLast`
- CS35 implementations: **ArrayList, LinkedList**

C++ vector

The C++ **vector** library provides a data type for templated **Lists**.

You should know how to:

- Create a vector.
- Access the vector documentation on cplusplus.com
- Add items to a vector.
- Iterate through items in a vector.
- Reverse elements of a vector
- Sort a vector

Core vector operations

The C++ **vector** library provides a data type for templated **Lists**.

You should know how to:

- Create a vector.
- Access the vector documentation on cplusplus.com
- Add items to a vector.
- **Iterate** through items in a vector.
- **Reverse** elements of a vector
- **Sort** a vector

C++ vector examples

```
#include <vector>
vector<int> foo;                // create empty vector
vector<int> bar(10);            // create vector of ten items all initialized to zero.
for(int i=0; i<n; i++) {
        foo.push_back(i);        // add i to end of vector
  }
cout << foo.size();             // check size of vector
if(foo.empty()) {…}             // is vector empty?
foo.back();                     // get last element of vector
foo.pop_back();                 // delete last element.  NOTE: does not return element!
for(int i=0; i<n; i++) {    // iterate over items
        sum+= foo[i];
}
for(vector<int>::iterator it = foo.begin(); it!= foo.end(); it++) {    also // iteration
        sum+= *it;
}
```

Exercise: reverse items in vector

C++ algorithm library

The C++ **algorithm** library provides common algorithms, including

- `#include <algorithm>    // import algorithm library`
- `vector<int> foo;`
- …
- `reverse(foo.begin(), foo.end());  //reverse items`
- `sort(foo.begin(), foo.end());     //sort items`

Kattis Problem: icouldhavewon