Competitive Programming
234900

Lesson 2 - Basic C++ and introduction to STL
Introduction

• Competitive programming makes intensive use of C++ and STL
• The goal of this presentation is to refresh your memory
• Each slide contains input and output examples:

| Input | Output |

• More information can be found in the C++ reference website:
  • Good knowledge of STL can very helpful! We encourage you to experiment with and play 😊
Basic Usage

• Programs that use STL need to include the appropriate headers:

```cpp
#include <vector>
```

• STL interfaces are implemented inside the `std` namespace:

```cpp
using namespace std;
```

• The data type is specified when creating an instance:

```cpp
vector<int> v;
```
Data types – Size in memory

- `cout << "int= " << sizeof(int) << endl;`
- `cout << "long= " << sizeof(long) << endl;`
- `cout << "long long= " << sizeof(long long) << endl;`
- `cout << "double= " << sizeof(double) << endl;`
- `cout << "long double= " << sizeof(long double) << endl;`

- int = 4
- long = 4
- long long = 8
- double = 8
- long double = 8
pair

- `pair<int, string> a = {2,"what"};`
- `cout << '(' << a.first << " , " << a.second << ')'
  << endl;`
- `pair<int, pair<int, int>> tuple;`
  // pairs can also be nested
- `(2 , what)`
C++ string

- `#include <string>`
- `// strings work like vectors`
- `string s = "hello";`
- `s.pop_back();`
- `cout << s << endl;`
- `s.push_back('l');`
- `cout << s << endl;`
- `s += " world";`
- `cout << s << endl;`

- `hell`
- `helll`
- `helll world`
vector

- vector<int> arr1(1e5, 0);  //init - o(n)
- arr1[0] = 1;
- vector<int> arr2 = arr1;   //copy - o(n)
- arr2[0] = 2;  //arr1[0]=1
- arr1.push_back(3);
- cout << "arr1[1e5]=" << arr1[arr1.size () - 1] << endl;
- arr1.pop_back();
- arr1[1e5]=3
unordered_map hash table

• unordered_map<string, int> ht;
• ht["key"] = 5;
• cout << "ht["key"]=" << ht["key"] << endl;
• if (ht.find("key") != ht.end())
  • cout << "key found"<<endl;
• else
  • cout << "key not found" << endl;
• cout << "second=" << ht["second"]<<endl; // Implicitly create an item (val=0) for non-existant keys.
• ht["third"]++; // Increase by 1 if the key already exists in the hash table. If not, insert the key and set the value as 1.
• for (auto& p : ht)//to over all pairs O(n)
  • cout << "key=" << p.first << " val=" << p.second << endl;

• ht[key]=5
• key found
• second=0
• key=second val=5
• key=third val=1
unordered_map - hash table

- unordered_map - hash table in O(1)-keys must have hash function so int, string, long long and so on is ok
- but!! pair or struct are not ok because they don't have hash function so..you can implement or use map.
priority_queue-min/max heap

- vector<int> arr = {5,3,6,3,2,1,1};
- priority_queue<int> max_heap(arr.begin(),arr.end()); // built in o(n)
- cout << "my max_heap" << endl;
- while (!max_heap.empty()) // goes over the heap
  - {
    - cout << max_heap.top() << endl;
    - max_heap.pop(); // no return val
  }
- priority_queue<int, vector<int>, comparator> min_heap;
- min_heap.push(6);min_heap.push(8);min_heap.push(1);min_heap.push(2); // log(n)
- cout << "my min_heap" << endl;
- while (!min_heap.empty())
  - {
    - cout << min_heap.top() << endl;
    - min_heap.pop();
  }

- my max_heap
  - 6
  - 5
  - 3
  - 3
  - 2
  - 1
- my min_heap
  - 1
  - 2
  - 6
  - 8
struct comparator {// this already implement-greater<int>()
    bool operator()(int i, int j) {
        return i > j;
    }
};
• set<int> g;
• g.insert(5); g.insert(2); g.insert(8);
g.insert(2); // log(n) each
• g.erase(2); // log(n)
• g.insert(1); g.insert(9);
• cout << "my set" << endl;
• for (auto& x : g)
  • cout << x << " ";
  • cout << endl;
• if (g.find(1) != g.end())//log(n)
  • cout << "1 has been found"<<endl;

• my set
• 1 5 8 9
• 1 has been found
Set-lower/upper bound

- auto itr = g.lower_bound(1);// find the first element that does not compare less than val, return itr
  - cout << *itr << endl;
  - itr++;// need to check itr!=g.end()
  - cout << *itr << endl << endl;
  - itr = g.lower_bound(2);
  - cout << *itr << endl;
  - itr++;
  - cout << *itr << endl << endl;
  - itr = g.upper_bound(1);// find the first element that greater than val, return itr
  - cout << *itr << endl;
  - itr++;
  - cout << *itr << endl << endl;
  - itr = g.lower_bound(2);
  - cout << *itr << endl;
  - itr++;
  - cout << *itr << endl << endl;
STL Algorithms

• STL includes useful algorithms that operate on iterable objects:
  • sort – Sort an array/vector in O(nlog(n)) time
  • unique – Remove consecutive repeating elements
  • merge – Merge two sorted arrays/vectors
  • binary_search – Search sorted array in O(logn) time
    • Also related: lower_bound, upper_bound
  • nth_element – Find the nth element in a given array by performing partition.
  • And many more. Most of them are trivial and relatively easy to implement (max, min, count, reverse...)
STL Algorithms – cont.

• Pairs are sorted by p.first, and then by p.second

• When dealing with non-premitive objects (structs, classes), STL algorithms can still be used by overloading the < operator:

```cpp
struct fraction {
    int n, d; // (n/d)
    bool operator < (const fraction& f) const {
        return n*f.d < f.n*d;
    }
};
vector<fraciton> v;
sort(v.begin(), v.end());
```
Algorithm - sort

```cpp
• arr.clear();
• arr.push_back(3); arr.push_back(1);
  arr.push_back(1); arr.push_back(7);
• sort(arr.begin(), arr.end()); // nlog(n)
• for (auto& x : arr)
  • cout << x << " ";
  • cout << endl;
• vector<pair<int, int>> vp;
• sort(vp.begin(), vp.end()); // compare by the first and then by the second
• Example-(1,7)(2,1)(2,2)(2,7)(3,1)
```

• 1 1 3 7
Algorithm – All permutations

• //All permutation
• int myints[] = { 1,2,3 };
• sort(myints, myints + 3); //must before next permutation
• cout << "The 3! possible permutations with 3 elements:\n";
• do {
';
• } while (next_permutation(myints, myints + 3));

• The 3! possible permutations with 3 elements:
• 1 2 3
• 1 3 2
• 2 1 3
• 2 3 1
• 3 1 2
• 3 2 1
• After loop: 1 2 3
General Tips
Input/Output

• Input and output is always done through stdin, stdout.

• Input can have several formats:
  • Read until EOF:
    ```cpp
    while (cin >> n) {...}
    ```
  • Number of test cases is given first:
    ```cpp
    cin >> T;
    for(int t=0; t<T; t++) {...}
    ```
  • Read until 0:
    ```cpp
    while (cin >> N) {
        if (N==0) break;
        ...
    }
    ```
Submission instructions

- We will be using two online judges:
  - Live Archive: https://icpcarchive.ecs.baylor.edu
  - UVa Online Judge: https://uva.onlinejudge.org/
- Submission instructions:
  - You should only submit the problems you managed to solve (== got accepted by the online judge using the C++11 compiler)
  - Your submission will be checked using an automated system. Please make sure to name your files according to the following pattern:
    - For Live Archive problems, the file name will be icpc_####.cpp, where #### is the problem ID.
    - For UVa problems, the file name will be uva_####.cpp, where #### is the problem ID.
- One team member can submit for each team.
  - Please include the IDs of your team members.
General Tips

• Read all the questions
• Make sure your solutions has the correct time complexity (Rule of thumb: ~10M operations)
• Have fun!