CS3233
Competitive Programming
Dr. Steven Halim
Week 02 – Data Structures & Libraries
Focus on Bit Manipulation & Binary Indexed Tree
Outline

• Mini Contest 1 + Break (discussion of A/B)
• Some Admins
• **Data Structures With Built-in Libraries**
  – Just a quick walkthrough
  • Read/experiment with the details on your own
  – Linear Data Structures (CS1010/1\(^{st}\) half of CS2020)
  – Non Linear Data Structures (CS2010/2\(^{nd}\) half of CS2020)
  • Focus on the **red highlights**
• **"Top Coder" Coding Style** (overview) + Break
• **Data Structures With Our-Own Libraries**
  – Focus on Binary Indexed (Fenwick) Tree
Basic knowledge that all ICPC/IOI-ers must have!

LINEAR DATA STRUCTURES
WITH BUILT-IN LIBRARIES

CS3233 - Competitive Programming,
Steven Halim, SoC, NUS
I am...

1. A pure C coder
2. A pure C++ coder
3. A mix between C/C++ coder
4. A pure Java coder
5. A multilingual coder: C/C++/Java
Linear DS + Built-In Libraries (1)

1. Static Array, built-in support in C/C++/Java
2. Resize-able: C++ STL `vector`, Java `Vector`
   - Both are very useful in ICPCs/IOIs

• There are 2 very common operations on Array:
  – Sorting
  – Searching
  – Let’s take a look at efficient ways to do them

CS3233 - Competitive Programming,
Steven Halim, SoC, NUS
Two “fundamental” CS problems

SORTING + SEARCHING INVOLVING ARRAY
Sorting (1)

• Definition:
  – Given unsorted stuffs, sort them... *

• Popular Sorting Algorithms
  – $O(n^2)$ algorithms: Bubble/Selection/Insertion Sort
  – $O(n \log n)$ algorithms: Merge/Quick^/Heap Sort
  – Special purpose: Counting/Radix/Bucket Sort

• Reference:
In ICPC, you can “forget” all these...

- In general, if you need to sort something..., just use the $O(n \log n)$ sorting library:
  - C++ STL `algorithm:: sort`
  - Java `Collections.sort`

In ICPC, sorting is either used as *preliminary step* for more complex algorithm or to *beautify output*

- Familiarity with sorting libraries is a must!
Sorting (3)

• Sorting routines in C++ STL algorithm
  – sort – a bug-free implementation of *introsort*
    • Fast, it runs in $O(n \log n)$
    • Can sort basic data types (ints, doubles, chars), Abstract Data Types (C++ class), multi-field sorting ($\geq 2$ criteria)
  – partial_sort – implementation of *heapsort*
    • Can do $O(k \log n)$ sorting, if we just need top-k sorted!
  – stable_sort
    • If you need to have the sorting ‘stable’, keys with same values appear in the same order as in input
Searching in Array

- Two variants:
  - When the array is sorted versus not sorted
- Must do $O(n)$ linear scan if not sorted - trivial
- Can use $O(\log n)$ binary search when sorted
  - PS: must run an $O(n \log n)$ sorting algorithm once
- Binary search is ‘tricky’ to code!
  - Instead, use C++ STL `algorithm::lower_bound`
Linear DS + Built-In Libraries (2)

3. Array of Boolean: C++ STL **bitset**
   - Faster than *array of bools* or *vector<bool>*!
   - No specific API in Java that is similar to this

4. Bitmask
   - a.k.a. lightweight set of Boolean or bit string
   - Explanation via:
Linear DS + Built-In Libraries (3)

5. Linked List, C++ STL list, Java LinkedList
   – Usually not used in ICPCs/IOIs
   – If you need a resizeable “list”, just use vector!

6. Stack, C++ STL stack, Java Stack
   – Used by default in Recursion, Postfix Calculation, Bracket Matching, etc

7. Queue, C++ STL queue, Java Queue
   – Used in Breadth First Search, Topological Sort, etc
   – PS: Deque, used in ‘Sliding Window’ algorithm
More efficient data structures

NON-LINEAR DATA STRUCTURES
WITH BUILT-IN LIBRARIES
Binary Search Tree (1)

- ADT Table (key → data)
- Binary Search Tree (BST)
  - Advertised $O(\log n)$ for insert, search, and delete
  - Requirement: the BST must be balanced!
    - AVL tree, Red-Black Tree, etc... *argh*
- Fret not, just use: C++ STL map (Java TreeMap)
  - UVa 10226 (Hardwood Species)*
Binary Search Tree (2)

- ADT Table (key exists or not)
- Set (Single Set)
  - C++ STL `set`, similar to C++ STL `map`
    - map stores a `(key, data)` pair
    - set stores just the `key`
  - In Java: `TreeSet`
- Example:
  - UVa [11849](http://www.uva.be/index.php) – CD
Heap

• Heap
  – C++ STL algorithm has some heap algorithms
    • partial_sort uses heapsort
  – C++ STL priority_queue (Java PriorityQueue) is heap
    • Prim’s and Dijkstra’s algorithms use priority queue

• But, we rarely see pure heap problems in ICPC
Hash Table

• Hash Table
  – Advertised O(1) for insert, search, and delete, but:
    • The hash function must be good!
    • There is no Hash Table in C++ STL (exists in Java API)
  – Nevertheless, O(log n) using map is usually ok

• Direct Addressing Table (DAT)
  – Rather than hashing, we more frequently use DAT
  – UVa 11340 (Newspaper)
Supplementary
Top Coder Coding Style (1)

• You may want to follow this coding style (C++)

1. Include **important** headers 😊

   - #include <algorithm>
   - #include <cmath>
   - #include <cstdio>
   - #include <cstring>
   - #include <iostream>
   - #include <map>
   - #include <queue>
   - #include <set>
   - #include <string>
   - #include <vector>
   - using namespace std;

Want More?

Add libraries that you frequently use into this template, e.g.:

ctype.h
bitset
etc
2. Use shortcuts for common data types
   - typedef long long ll;
   - typedef vector<int> vi;
   - typedef pair<int, int> ii;
   - typedef vector<ii> vii;

3. Simplify Repetitions/Loops!
   - #define REP(i, a, b) for (int i = int(a); i <= int(b); i++)
   - #define REPN(i, n) REP (i, 1, int(n))
   - #define REPD(i, a, b) for (int i = int(a); i >= int(b); i--)
   - #define TRvi(c, it) \
     for (vi::iterator it = (c).begin(); it != (c).end(); it++)
   - #define TRvii(c, it)
     for (vii::iterator it = (c).begin(); it != (c).end(); it++)

Define your own loops style and stick with it!
Top Coder Coding Style (3)

4. More shortcuts
   - `for (i = ans = 0; i < n; i++)`... // do variable assignment in for loop
   - `while (scanf("%d", n), n) { ... // read input + do value test together`
   - `while (scanf("%d", n) != EOF) { ... // read input and do EOF test`

5. STL/Libraries all the way!
   - `isalpha (ctype.h)`
     - `inline bool isletter(char c) {
       return (c>='A'&&c<='Z') || (c>='a'&&c<='z'); }
   - `abs (math.h)`
     - `inline int abs(int a) { return a >= 0 ? a : -a; }
   - `pow (math.h)`
     - `int power(int a, int b) {
       int res=1; for (; b>=1; b--) res*=a; return res; }
   - Use STL data structures: vector, stack, queue, priority_queue, map, set, etc
   - Use STL algorithms: sort, lower_bound, max, min, max_element, next_permutation, etc

CS3233 - Competitive Programming,
Steven Halim, SoC, NUS
6. Use I/O Redirection
   - `int main() {`
   -   `// freopen("input.txt", "r", stdin); // don't retype test cases!
   -   `// freopen("output.txt", "w", stdout);
   -   `scanf and printf as per normal; // I prefer scanf/printf than
   -     // cin/cout, C style is much easier

7. Use memset/assign/constructor effectively!
   - `memset(dist, 127, sizeof(dist));`
     // useful to initialize shortest path distances, set INF to 127!
   - `memset(dp_memo, -1, sizeof(dp_memo));`
     // useful to initialize DP memoization table
   - `memset(arr, 0, sizeof(arr));` // useful to clear array of integers
   - `vector<int> dist(v, 2000000000);
   - `dist.assign(v, -1);`
Top Coder Coding Style (5)

8. Declare (large) static DS as global variable
   – All input size is known, declare data structure size LARGER than needed to avoid silly bugs
   – Avoid dynamic data structures that involve pointers, etc
   – Use global variable to reduce “stack size” issue

• Now our coding tasks are much simpler 😊
• Typing less code = shorter coding time
  = better rank in programming contests 😊
Quick Check

1. I can cope with this pace...
2. I am lost with so many new information in the past few slides
5 Minutes Break

• One data structures *without* built-in libraries will be discussed in the last part...
  – Binary Indexed (Fenwick) Tree
  – Graph, Union-Find Disjoint Sets, and Segment Tree are not discussed in this year’s CS3233 Week02
    • Graph DS is covered in details in CS2010/CS2020
    • UFDS is covered briefly in CS2010/CS2020
    • Please study Segment Tree on your own
      – We try not set any contest problem involving Segment Tree
Graph (not discussed today, revisited in Week05/08)
Union-Find Disjoint Sets (not discussed today, read Ch2 on your own)
Segment Tree (not discussed today, read Ch2 on your own)
Fenwick Tree (discussed today)

DATA STRUCTURES
WITHOUT BUILT-IN LIBRARIES
Fenwick Tree (1)

- Cumulative Frequency Table
  - Example, \( s = \{2, 4, 5, 5, 6, 6, 6, 7, 7, 8\} \) (already sorted)

<table>
<thead>
<tr>
<th>Index/Score/Symbol</th>
<th>Frequency</th>
<th>Cumulative Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Fenwick Tree

- Fenwick Tree (inventor = Peter M. Fenwick)
  - Also known as “Binary Indexed Tree”, very aptly named
  - Implemented as an array, let call the array name as ft
- Each index of ft is responsible for certain range (see diagram)

<table>
<thead>
<tr>
<th>Key/Index</th>
<th>Binary</th>
<th>Range</th>
<th>F</th>
<th>CF</th>
<th>FT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>1..2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>1..4</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>5..6</td>
<td>3</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1..8</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Fenwick Tree (3)

- To get the cumulative frequency from index 1 to b, use \texttt{ft\_rsq(ft, b)}
  - The answer is the sum of sub-frequencies stored in array \texttt{ft} with indices related to \texttt{b} via this formula: \texttt{b'} = \texttt{b} - \texttt{LSOne(b)}
    - Recall that \texttt{LSOne(b)} = \texttt{b} \& (-\texttt{b})
      » That is, strip the least significant bit of \texttt{b}
  - Apply this formula iteratively until \texttt{b} is 0

**Analysis:**
- This is \texttt{O(log n)}

**Example:** \texttt{ft\_rsq(ft, 6)}
  - \texttt{b} = 6 = \texttt{0110}, \texttt{b'} = \texttt{b} - \texttt{LSOne(b)} = \texttt{0110} - \texttt{0010}, \texttt{b'} = 4 = \texttt{0100}
  - \texttt{b'} = 4 = \texttt{0100}, \texttt{b''} = \texttt{b'} - \texttt{LSOne(b')} = \texttt{0100} - \texttt{0100}, \texttt{b''} = 0, \text{stop}

- Sum \texttt{ft[6] + ft[4]} = 5 + 2 = 7
  - (see the blue area that covers range \texttt{[1..4] + [5..6] = [1..6]})
To get the cumulative frequency from index $a$ to $b$, use $\text{ft}_\text{rsq}(ft, a, b)$.

- If $a$ is not one, we can use:
  $$\text{ft}_\text{rsq}(ft, b) - \text{ft}_\text{rsq}(ft, a - 1)$$
  to get the answer.

**Analysis:**

- Example: $\text{ft}_\text{rsq}(ft, 3, 6) = \\
  \text{ft}_\text{rsq}(ft, 6) - \text{ft}_\text{rsq}(ft, 3 - 1) = \\
  \text{ft}_\text{rsq}(ft, 6) - \text{ft}_\text{rsq}(ft, 2) = \\
  \text{blue area minus green area} = \\
  (5 + 2) - (0 + 1) = \\
  7 - 1 = 6$

**Why?**
Fenwick Tree (5)

- To update the frequency of an key/index $k$, by $v$ (either positive or negative), use $\text{ft\_adjust}(ft, k, v)$
  
  • Indices that are related to $k$ via $k' = k + \text{LSOne}(k)$ will be updated by $v$ when $k < \text{ft\_size}()$
    
    - Example: $\text{ft\_adjust}(ft, 5, 2)$
      
      » $k = 5 = 0101$, $k' = k + \text{LSOne}(k) = 0101 + 0001$, $k' = 6 = 0110$
      
      » $k' = 6 = 0110$, $k'' = k' + \text{LSOne}(k') = 0110 + 0010$, $k'' = 8 = 1000$
      
      » And so on while $k < \text{ft\_size}()$

Analysis:
- This is also $O(\log n)$

Why?
- Observe that the dotted red line in the figure below stabs through the ranges that are under the responsibility of indices 5, 6, and 8
  
  - $\text{ft}[5]$, 2 updated to 4
  - $\text{ft}[6]$, 5 updated to 7
  - $\text{ft}[8]$, 10 updated to 12
typedef vector<int> vi;
define LSONe(S) (S & (-S))

void ft_create(vi &ft, int n) { ft.assign(n + 1, 0); } // init: n+1 zeroes

int ft_rsq(const vi &ft, int b) { // returns RSQ(1, b)
    int sum = 0; for (; b; b -= LSONe(b)) sum += ft[b];
    return sum;
}

int ft_rsq(const vi &t, int a, int b) { // returns RSQ(a, b)
    return ft_rsq(t, b) - (a == 1 ? 0 : ft_rsq(t, a - 1));
}

// adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
void ft_adjust(vi &ft, int k, int v) {
    for (; k < (int)ft.size(); k += LSONe(k)) ft[k] += v;
}
Fenwick Tree (7) – Application

• Fenwick Tree is very suitable for *dynamic* RSQs (cumulative frequency table) where each update occurs on a certain index only

• Now, think of potential real-life applications!
  – [http://uhunt.felix-halim.net/id/32900](http://uhunt.felix-halim.net/id/32900)
  – Consider code running time of [0.000 - 9.999] for a particular UVa problem
    • There are up to 9+ million submissions/codes
      – About thousands submissions per problem
    • If your code runs in 0.342 secs, what is your rank?

• How to use Fenwick Tree to deal with this problem?
Quick Check

1. I am lost with Fenwick Tree
2. I understand the basics of Fenwick Tree, but since this is new for me, I may/may not be able to recognize problems solvable with FT
3. I have solved several FT-related problems before
Summary

• There are a lot of great Data Structures out there
  – We need the most efficient one for our problem
    • Different DS suits different problem!

• Many of them have built-in libraries
  – For some others, we have to build our own (focus on FT)
    • Study these libraries! Do not rebuild them during contests!

• From Week03 onwards and future ICPCs/IOIs, use C++ STL and/or Java API and our built-in libraries!
  – Now, your team should be in rank 30-45 (from 60)
    (still solving ~1-2 problems out of 10, but faster)