GB21802 - Programming Challenges

Week 1 - Data Structures

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2019-04-19,22

Last updated April 18, 2019

Results for the Previous Week

Results on Wednesday:

Week 0: Introduction and Problem Solving

Deadline: 4/18/2019, 11:59:59 PM (1 day, 07:07 hours from now)
Problems Solved -- 0P:16, 1P:5, 2P:2, 3P:2, 4P:13, 5P:2, 6P:3, 7P:1, 8P:4,

#	Name	Sol/Sub/Total	My Status
1	The 3n + 1 problem	27/31/48	
2	Cost Cutting	28/29/48	
3	Event Planning	21/22/48	
4	Horror Dash	26/26/48	
5	Y3K Problem	11/12/48	
6	Stack 'em Up	6/7/48	
7	Traffic Lights	7/7/48	
8	Jolly Jumpers	8/10/48	

click to show/hide

Hint: Take a quick look at all problems before solving!

Messages to Students:

1. Be careful with the input: Worst cases and TLE
2. Use **pipes** to avoid typing cases by hand!

```
$ java Main < input.in
1 10 20
10 1 20
10 10 7
```

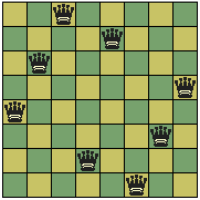
Data Structures
CP Book Chapter 2

Data Structures and Programming Challenges

- Using the correct data structure makes the program faster.
- Using the correct data structure makes the problem easier.
- Every program needs some data structure. Learn it!

In this lecture, we review some common data structures, and we also look a bit at common library functions

Example 0: 8 Queen Problem (UVA 750)



For a board of size $n \times n$, find how many safe configurations of n queens exist.

Because we need to find how many configurations exist, we need to test "all" configurations.

```
for i = 0 to #configurations do
  sum = testIfConfigurationIsSafe(i)
```

Example 0: 8 Queen Problem (UVA 750)

How can we represent a configuration?

(1) Position x, y of every queen (size: n^{n^2})

```
conf[0] = {{a,1}, {a,1}, {a,1}, ... {a,1}, {a,1}}
conf[1] = {{a,1}, {a,1}, {a,1}, ... {a,1}, {a,2}}
conf[2] = {{a,1}, {a,1}, {a,1}, ... {a,1}, {a,3}}
```

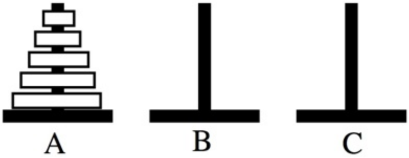
(2) Row r of every queen (size: n^n)

```
conf[0] = {0,0,0,0,0,0,0,0}
conf[1] = {0,0,0,0,0,0,0,1}
conf[2] = {0,0,0,0,0,0,0,2}
```

(3) Permutation of row positions (size: $n!$)

```
conf[0] = {0,1,2,3,4,5,6,7}
conf[1] = {0,1,2,3,4,5,7,6}
conf[2] = {0,1,2,3,4,6,5,7}
```

Example 1: The Towers of Hanoi



- You have N disks and K poles. Each disk has unique size s_i.
- A disk i can be moved from one pole to another.
- A move of disk i to pole k is only valid if k has no disks smaller than i
- Find the list of moves to move all disks from pole 1 to pole K.

How do you represent the data in this problem?

Example 1: The Towers of Hanoi

A string with “n” disks, from smaller to larger.

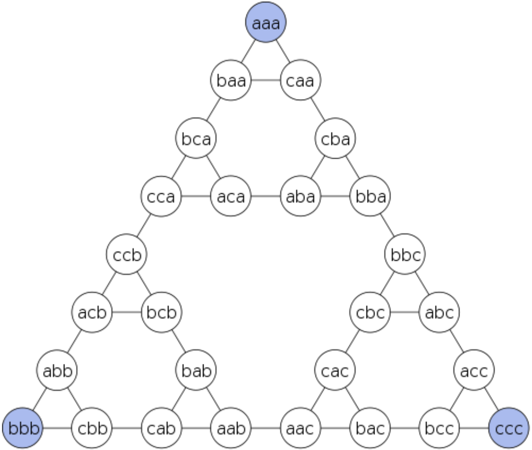


Image created by nonenmac

Example 2: Army Buddies (UVA 12356)

Problem Description

- There is a line of S soldiers: $0, 1, 2, 3, 4, ..., S$
- There are Q queries that remove soldiers from i to j:

```
Q1: 2,4          (removes soldiers 2, 3, 4)
Q2: 6,7          (removes soldiers 6, 7)
Q3: 1,1          (removes soldier 1)
```

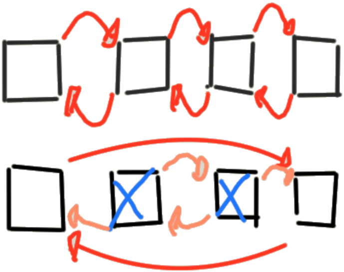
- For each query, list the soldier to the left and to the right

```
A1: 1,5        1 x x x 5 6 7
A2: 5,*        1 - - - 5 x x
A3: *,5        x - - - 5 - -
```

How do we solve this problem?

Example 2: Army Buddies (UVA 12356)

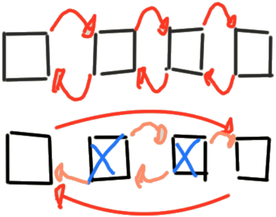
Idea 1: Linked Lists



- Represent the line as a linked list.
- Find the 1st soldier (This is $O(n)$)
- Find the 2nd soldier (This is $O(n)$)
- Update the list and print the neighbors. (This is $O(1)$)

Example 2: Army Buddies (UVA 12356)

A solution using linked lists



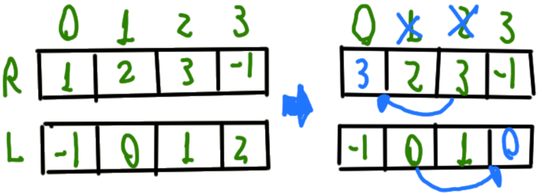
Problem! The input is too big, and O(n) takes too much time.

- $1 \leq S \leq B \leq 10^5$; $(O(n^2)) = 10^{10}$
- Also multiple cases; $(O(n^2 k)) = 10^{10} k$

Think of a different solution! (Do not look at the next slide :-))

Example 2: Army Buddies (UVA 12356)

A solution using arrays



- Problem: Do not update ALL soldiers, just the edge.
- Idea: Neighbor Array
  - Let **R** be: **Int** Array of Right neighbors
  - Let **L** be: **Int** Array of Left neighbors
- Question: how do we update R and L after query (r, l)?

Data Structures and Programming Contests

- Choosing the right data structure makes the program easy and fast
- Always avoid pointers in programming contests
  - Too many bugs
  - Not enough gains
  - Better to use arrays!
- Remember the STL for common data structure!

The simple array!

Arrays are the simplest data structure, but also the most often used.

Merits

- Easy to implement! No worries about pointers;
- Can simulate pointers using index operations;
- Many library Functions;

Concerns

- Reordering many items can be expensive;

Implementing arrays/vectors (C++)

```
#include <vector>

int arr[5] = {7,7,7};         // arr = {7,7,7,0,0}
vector<int> v(5, 5);          // v = {5,5,5,5,5}

int x = arr[2] + v[2];        // x = 12

arr[5] = 5;                   // Runtime error
cout << v[7];                 // 0 !! Be careful.

v.push_back(6);               // v = {5,5,5,5,5,6}
```

Trying to access indexes outside of an array is a common source of Runtime Errors (RTE)

How do you reset an array?

Implementation matters

```
#include <vector>
#include <string.h>
vector<int> v(10000,7)

memset(v, 0, 10000*__SIZEOF_INT__);          // Method 1
fill(v.begin(), v.end(), 0);                 // Method 2
for (int i = 0; i < 10000; i++) v[i] = 0;    // Method 3
v.assign(v.size(), 0);                       // Method 4
```

```
Method       |  executable size    |  Time Taken (in sec)    |
             |  -O0      |  -O3    |  -O0        |  -O3      |
-------------|-----------|---------|-------------|-----------|
1. memset    | 17 kB     | 8.6 kB  | 0.125       | 0.124     |
2. fill      | 19 kB     | 8.6 kB  | 13.4        | 0.124     |
3. manual    | 19 kB     | 8.6 kB  | 14.5        | 0.124     |
4. assign    | 24 kB     | 9.0 kB  | 1.9         | 0.591     |
```

Operations in Arrays

Example – Vito's Family (UVA 10041)

Input: A list of integers (street addresses):

10, 20, 10, 10, 40, 80, 30, 90, 20, 55, 20

Output: The address (integer) with minimal distance to all others.

- **10**: $0+10+0+0+30+70+20+80+10+45+10=275$
- **40**: $30+20+30+30+0+40+10+50+20+15+20=265$
- **20**: $10+0+10+10+20+60+10+70+0+35+0=225$

Result: 20!

How do we solve this problem?

Operations in Arrays

- Solution: Find the Median address.
- 1- sort the address array, 2- select the middle value.

```
#include<iostream>
#include<algorithm>
using namespace std;

int main() {
    int n; int add[100];
    cin >> n;
    for (int i=0; i<n; i++) { cin >> add[i]; }

    sort(add, add+n);

    cout << add[n/2] << endl;
}
```

Using Sorting

Sorting can be used for many, many things:

- Finding the Highest n values, Finding duplicate values;
- Binary Search ($O(\log n)$)
- Pre-processing data for other algorithms.

Let's do Sorting!

algorithm, sorting and binary search

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

int main () {
  int n, t, search; vector<int> v;
  cin >> n >> search;
  for (int i=0; i<n; i++) { cin >> t; v.push_back(t); }

  sort (v.begin(), v.end());
  vector<int>::iterator low,up;
  low = lower_bound (v.begin(), v.end(), search);
  up  = upper_bound (v.begin(), v.end(), search);
  cout << (low-v.begin()) << " and " << (up-v.begin());
}
```

Sorting with specific sorting function

Imagine you need to sort by number of points (bigger is best), penalty (smaller is best), and name (alphabetical order)

```
#include <algorithm>
#include <vector>
#include <string>
struct team{ string name; int point; int penal;
             team(string _n, int _po, int _pe) :
               name(_n), point(_p), penal(_g){} };

bool cmp(team a, team b) {
  if (a.point != b.point) return a.point > b.point;
  if (a.penal != b.penal) return a.penal < b.penal;
  return strcmp(a.name,b.name); }

vector<team> v;
sort(v.begin(), v.end(), cmp); // sort using cmp
reverse(v.begin(), v.end()); // and reverse
```

A Permutation Problem

- Input: A set of item costs: $c_1, c_2, c_3, \ldots, c_n$, and your money m.
- Output: The list of indexes of items you can buy.
- how to do it?: Test all possible item permutations
- But how?

Looping Permutations

```
#include <iostream>
#include <vector>
using namespace std;

int main () {
  int n, t, m; vector<int> p;
  cin >> n >> m;
  for (int i=0; i<n; i++)
    { cin >> t; p.push_back(t); }

  for (int i = 0; i < (1 << n); i++) {
      t = 0;
      for (int j = 0; j < n; j++)
          t += (i & (1 << j) ? p[j] : 0);
      if (t == m)
          cout << "found!" << endl;
}}
```

Bitmasks

```
for (int i = 0; i < (1 << n); i++) {
t += (i & (1 << j) ? p[j] : 0);
```

- Bitmasks are sets of booleans using integers.
- They are very useful for representing sets
  - Set Looping;
  - Set Union (OR);
  - Set Intersection (AND);
- They are Very fast too

Binary Operatons on Bitmasks (2)

- Multiply/Divide an integer by two :: shift bits left, right

```
S                      = 34 =  100010
S = S << 1 = S*2 = 68 = 1000100
S = S >> 2 = S/4 = 17 =   10001
S = S >> 1 = S/2 =  8 =    1000
```

- Check if the i-th bit is turned on:

```
S           = 34       =  100010
j = 3, 1 << j          =  001000
i = 1, 1 << 1          =  000010
                          ------
Tj= S & ( 1 << j)      =  000000  = 0 # 3 is not set
Ti= S & ( 1 << i)      =  000010 != 0 # 1 is set
```

Binary Operations on Bitmasks (2)

- To set/turn on the jth item, use bitwise OR operation S |= (1 « j)

```
S              = 34         =   100010
j = 3, 1 << j               =   001000
                                ------ OR (S |= 1 << j)
S              = 42         =   101010
```

- To set/turn off the jth item, use bitwise AND operation S &= (1 « j)

```
S              = 50         =   110010
j = (1<<5)|(1<<3)           =   101000 # unset items 5,3
~j                          =   010111
                                ------
S &= ~(j)                   =   010010 # 18
```

- There is the bitset class, but it cannot be used as an array index.

Long Live the STL!

- The standard library implements many data structures that are useful for programming contests.
- Let’s review a few of them here.
- The website `https://visualgo.net/` has good reviews of many data structures;

Deque, Queue, Stack

Sometimes you want special access to the start or end of a vector.

- stack: *pop* and *push* from the front;
- queue: *pop* from the back, *push* from the front;
- deque: *pop_front, push_front, pop_back, push_back*;

Behind C++

Actually, *Queue* and *Stack* are high level constructs, List or Deque are used to implement them.

Queue and Stacks

Queues and Stacks are useful to simplify common cases of vectors

Stack Example: Testing if a set of parenthesis is balanced.

```
#include <stack>
stack<char> s;
char c;

while(cin >> c) {
  if (c == '(') s.push(c);
  else {
    if (s.size() == 0) { s.push('*'); break; }
    s.pop();
  }
}
cout << (s.size() == 0 ? "balanced" : "unbalanced");
```

Problem Example: CD – 11849

Input:

- Jack CD collection: Up to 10^6 CDs, with ID up to 10^9
- Jill CD collection: Up to 10^6 CDs, with ID up to 10^9

Output:

- How Many CDs are in both Collections?

Problem Example: CD – 11849

Naive Solution:

1. Store all IDs in collection 1 in a Vector (n)
2. Sort the Vector (nlogn)
3. For each ID in collection 2, test if it exists in Vector with Binary Search (nlogn)

Total Cost: $n + n\log n + n\log n$

Let's use a MAP for $O(\log N)$ search using a balanced search tree

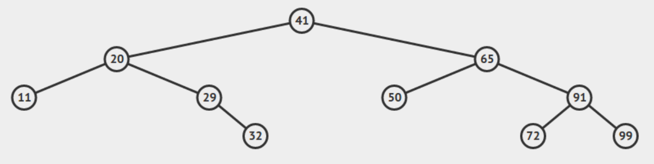
Solving CD with a MAP (Approximate Solution)

```
#include <iostream>
#include <set>
using namespace std;

int main() {
    int N, M, num;
    cin >> N >> M;

    set<int> first, second;
    while (N--) { cin >> num; first.insert(num); }
    while (M--) { cin >> num; second.insert(num); }
    int count = 0;
    for (set<int>::iterator iter = first.begin();
         iter != first.end(); ++iter)
      if (second.find(*iter) != second.end())
        ++count;
      cout << count << '\n';
}
```

Balanced Search Trees



- *Search Trees* Keep items in an ordered relationship.
- For example: Left children always have smaller values, Right children always have larger values;
- Insertion/Search/Deletion in a tree costs $O(h)$, where h is the height of the tree;
- For a tree with n elements, the minimum height is $\log n$
- For a balanced tree, the maximum height is also $\log n$
- How to keep the tree balanced?

Balanced Search Trees

How to keep the tree balanced?

There are many Tree implementations/algorithms for keeping an BST balanced, and minimizing the tree height efficiently:

- AVL Tree (Adelson-Velskii-Landis);
- Red-Black Tree;
- B-Tree;
- Splay Tree;

However, in a programming context (or even day to day life), implementing these trees from scratch is Dangerous.

Luckly, most standard libraries include some implementation of BST.

ABLs in C++: Map and Set

- In C++, the *Map* and *Set* classes are implemented using BSTs
- *Map* Accept Key-value pairs;
- *Set* Accepts only Keys;

Using Map in C++

```
#include <map>
map<string, int> ages;    ages.clear();

ages["john"] = 40;
ages["billy"] = 39;
ages["andy"] = 29;
ages["steven"] = 42;
ages["felix"] = 33;

// What is the age of andy?
map<string, int>::iterator it = ages.find("andy");
cout << it->second << endl;

// Which names are between "f" and "m" ??
for (map<string, int>::iterator it =
     age.lower_bound("f");                 // finds felix
     it != age.upper_bound("m"); it++)   // finds johm
        cout << " " << ((string)it->first).c_str();
```

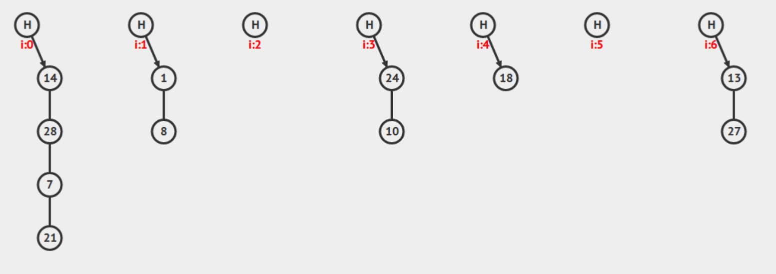
Using Set in C++

```
#include <set>
set<int> CDs;
CDs.clear();

// Adding some values
CDs.insert(1000); CDs.insert(999); CDs.insert(1337);
CDs.insert(1313); CDs.insert(100020);

// Testing if a particular value exists (O(logn))
set<int>::iterator f = used_values.find(79);
if (f == used_values.end())
  cout << "not found!\n";
else
  cout << *f;     // Index!
```

Hash Tables



- Insertion and Search: O(1) – Slow iteration;
- C++ library: *std::unordered_map*;
- Hash parameter – Defines Collision results.
- Learn more about hash tables here: `https://visualgo.net/ja/hashtable`

Hand-making Data Structures

- Sometimes, it is necessary to extend the standard data structures (arrays, maps, etc)
- Other times, it is necessary to implement data structures not included in the standard libraries (graphs, UFDS, etc)
- Let's see a few examples.

Union-Find Disjoint Set (UFDS)

Motivating Problem

Network Connections – UVA793

In a network with *n* computers, some are connected to others.

Input: A series of “commands”

- c i j – Means computer *i* is connected to computer *j*
- q i j – Question: is computer *i* connected to computer *j*?

Output: The number of “q” with answer yes, and the number of “q” with answer no.

Union-Find Disjoint Set (UFDS)

Motivating Problem – Naive answer

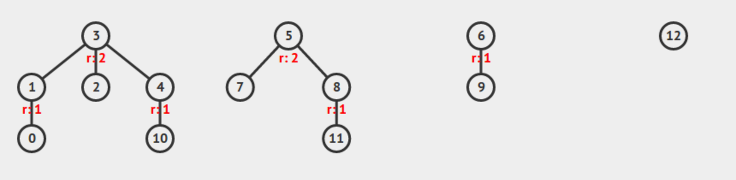
- One idea: Use a Neighborhood Matrix ($n \times n$) initalized with zeros.
- For every “c i j”, $N_{i,j}$, $N_{j,i}$ becomes 1.
- We can follow the graph to answer “q i j”.

How good is this solution?

- Cost to insert a new connection: O(1)
- Cost to check if “q i j”: O(n) (worst case)

We can do better!

Union-Find Disjoint Set



- The UFDS keeps sets of items, each is represented by a parent;
- When you join two sets You join their parents;
- When you test the parent of an item You flatten the tree;
- Test_item and Join_item are both O(1);
- More Information `https://visualgo.net/ja/ufds`;

UFDS Implementation using Arrays

```
int p[MAX], r[MAX];
int find(int x) {
    return x == p[x] ? x : p[x]=find(p[x]);
}
int join(int x, int y) {
    x = find(x), y = find(y);
    if(x != y) {
        if(r[x] < r[y])
            p[x] = y, r[y] += r[x];
        else
            p[y] = x, r[x] += r[y];
        return 1;
    }
    return 0;
}
void init() {
    for(int i = 0; i < MAX; i++)
        p[i] = i, r[i] = 1; }
```

Union Find Disjoint Set

Problem II – War

From a set of 10k people, some are friends, other are enemies.

- If A,B are friends, and B,C are friends, then A,C are friends
- If A,B are friends, and B,C are enemies, then A,C are enemies
- If A,B are enemies, and B,C are enemies, then A,C are friends

Input: A series of commands from the set below:

- SetFriends(i,j)
- SetEnemies(i,j)
- TestFriends(i,j)
- TestEnemies(i,j)

Output:

- If a “SetFriends” or “SetEnemies” is impossible, output “-1”
- For a “TestFriends”, “TestEnemies”, output 0 - false, 1 - true

Union Find Disjoint Set

Problem II – War

This problem is similar to “Networking”, but now you need to keep track of **TWO** relations.

Some ideas:

- Keep UFDS for friends, and UFDS for enemies?
- Keep an “enemy” flag for each person?
- Add “negative people” to friend-set on UFDS?

Which idea is easier to implement?

Range Maximum Query – RMQ

Suppose you have an array of values:

```
Value: 18 17 13 19 15 11 20
Index:  0  1  2  3  4  5  6
```

The Range Maximum Query problem asks you to find the index with the maximum value between two indexes:

- RMQ(0,0) = 0
- RMQ(0,6) = 6
- RMQ(1,4) = 3

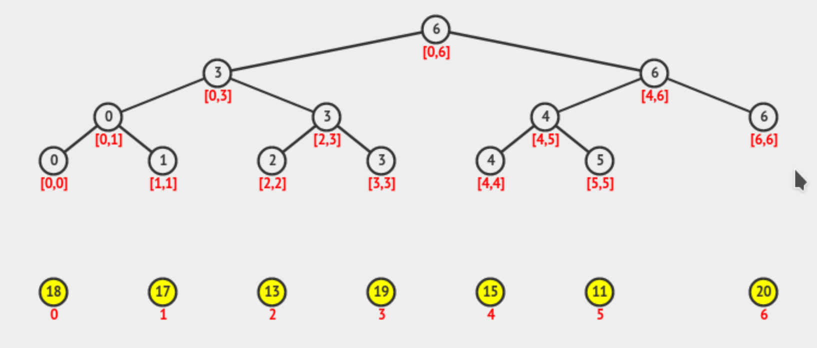
Naive Method: loop from i to j, find maximum value. ($O(nk)$)

But what is the number of **Values** or **Queries** is too big?

Segment Tree

- Basic idea: index the array data in a binary tree
- Creation of the tree: $O(n)$
- Query of a segment: $O(\log n)$
- Update of the tree: $O(\log n)$ Important Part
- Many Implementations
  (this implementation: vector based heap)

Segment Tree



Let’s see the segment tree animation at VISUALGO.

Coding the Segment Tree

Creating the Tree

```
typedef vector<int> vi; // We always use this!

class SegmentTree { // OOP implementation,
private: vi st, A;  // vi: typedef vector<int> vi;
  int n;
  int left (int p) { return  p<<1; }  // heap-like index;
  int right(int p) { return (p<<1) + 1; }

  void build(int p, int L, int R) {   // O(n log n)
    if (L == R)
      st[p] = L;                        // store the index
    else {                              // recursive build
      build(left(p) , L          , (L+R)/2);
      build(right(p), (L+R)/2 + 1, R      );
      int p1 = st[left(p)], p2 = st[right(p)];
      st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  } }
```

Code from `https://github.com/stevenhalim/cpbook-code`

Coding the Segment Tree

Query the Tree

rmq(1, 0, n-1, i, j) – Query from i to j.

```
int rmq(int p, int L, int R, int i, int j) // O(log n)
{
  if (i >  R || j <  L)
    return -1;     // outside query range
  if (L >= i && R <= j)
    return st[p]; // inside query range

  // compute the min position in the left and right part
  int p1 = rmq(left(p) , L         , (L+R)/2, i, j);
  int p2 = rmq(right(p), (L+R)/2+1, R       , i, j);

  if (p1 == -1) return p2;    // segment outside query
  if (p2 == -1) return p1;    // segment outside query
  return (A[p1] <= A[p2]) ? p1 : p2;
}
```

Code from `https://github.com/stevenhalim/cpbook-code`

Coding the Segment Tree

Update the Tree

update(1, 0, n-1, i, v) – update index i to value v

```
int update(int p, int L, int R, int idx, int new_value) {
  int i = idx, j = idx;    //for point update i = j = idx
  // if the curr interval does not intersect the update,
  if (i > R || j < L) return st[p];  //return node value!
  // if the current interval is in the update range,
  if (L == i && R == j) {
    A[i] = new_value;     // update the underlying array
    return st[p] = L;     // this index
  }
  // compute the min pos in L/R part of the interval
  int p1, p2;
  p1=update(left(p) , L        , (L+R)/2, idx, new_value);
  p2=update(right(p), (L+R)/2+1, R      , idx, new_value);
  // return the position where the overall minimum is
  return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
}
```

Code from `https://github.com/stevenhalim/cpbook-code`

End of the Class!

- Questions about the problems?
- Other questions?