
DON'T PANIC

Formation beOI

Data structures

Benoît Legat

beOI

8 février 2015

Introduction

What is pedagogy

Noob Just tell them to listen.

Pro Don't answer question they didn't ask. If they don't ask question, give them problems.

Expert (WIP) Give them problems and don't answer their question.

Being competitive

- Know your classics !
- Master the STL.
- Master `iostream` AND `stdio`.
- Avoid common coding guidance.
- Don't allocate with `malloc`, `new`, do static allocation of the worst case (you need to pass it anyway).
- Don't optimize the easy cases
- Don't do useless optimisation
- Beware time waster problems ! They might kill you while you are trying to kill them.
- Pick the solution that is fastest to code and works.
- If want to commit suicide when thinking about your solution. That means there exists a better one.

Fast IO in Java

```java
public class Main {
public static void main(String[] args) throws IOException {
BufferedReader in = new BufferedReader(new InputStreamReader(System.
    in));
PrintWriter out = new PrintWriter(new BufferedWriter(new
    OutputStreamWriter(System.out)));
StringTokenizer st = new StringTokenizer(in.readLine());
int tests = Integer.parseInt(st.nextToken());
for (int test = 1; test <= tests; test++) {
st = new StringTokenizer(in.readLine());
int n = Integer.parseInt(st.nextToken());
            ...
}
in.close();
out.close(); // don't forget me
}
}
```

GDB

```
$ gdb a.out
NO WARRANTY this program won't work !
>> run
>> run < input
# Once it runs
>> break 42
>> break gridland # Break the function gridland
>> break 42 if n == 2
# Once it is stopped
>> step # Goes inside functions
>> next # Doesn't
>> continue
>> bt
>> bt full
>> frame 3
>> info locals
>> quit
```

Useful data structures

Array

- Keys : $\{0, \ldots, n-1\}$
- Access in $\mathcal{O}(1)$
- Modify in $\mathcal{O}(1)$

With dynamic array (`ArrayList` or `std::vector`), n is not fixed.

- Add in amortized $\mathcal{O}(1)$
- Remove in $\mathcal{O}(n)$ if not at the end

Linked list

- Access and modify in $\mathcal{O}(n)$
- Modify and modify in $\mathcal{O}(1)$ at extremities
- Add and remove in $\mathcal{O}(1)$

Stack and Queue

Specific Linked List (a bit faster and limited capability)

Stack LIFO, Last in First out.

```
Stack<Integer> s = new Stack<Integer>();
q.push(1);
int top = q.peek(); // just watch
top = q.pop();
```

Queue FIFO, First in First out.

```
Queue<Integer> q = new LinkedList<Integer>();
q.add(1);
int first = q.peek(); // just watch
fitst = q.poll();
```

Trap problem – 10226

- If you want to do it in C, it will take you 1 h, you loose a precious time.
- With the STL, you kill it in 5 min !

Input

```
Don't Panic
Mostly Harmless
42
Don't Panic
The Hitchhiker's Guide
```

Output

```
42 20.0000
Don't Panic 40.0000
Mostly Harmless 20.0000
The Hitchhiker's Guide 20.0000
```

- `System.out.printf("%s %.4f\n", s, p)`
- Exactly 4 decimals `cout << setiosflags(ios::fixed)<< setprecision(4)`
- Strings with spaces `std::getline`

Let's do some shopping !

Unique key

	sorted	unsorted
existence	`std::set`	`std::unordered_set`
	`TreeSet`	`HashSet`
association	`std::map`	`std::unsorted_map`
	`TreeMap`	`HashMap`
complexity	$\mathcal{O}(\log(n))$	$\mathcal{O}(1)$ on average

Different elements can have the same key

	sorted	unsorted
existence	`std::multiset`	`std::unordered_multiset`
association	`std::multimap`	`std::unsorted_multimap`
complexity	$\mathcal{O}(\log(n))$	$\mathcal{O}(1)$ on average

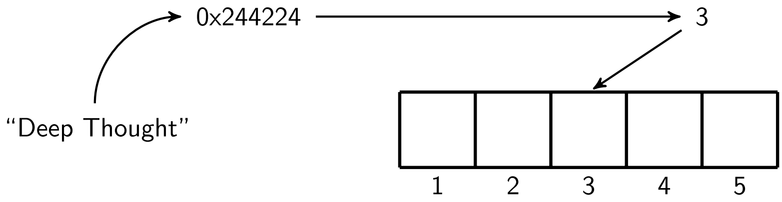
Hash Map I

Principle

There is a number N of buckets. The key is mapped to a bucket by hashing it to a number between 0 and $N - 1$.

1. Transform the key to a number k of type `size_t` ;
2. Get that number between 0 and $N - 1$ (e.g. $k \pmod{N}$) ;
3. Access the bucket with that index.

For now, $\mathcal{O}(1)$!



Hash Map II

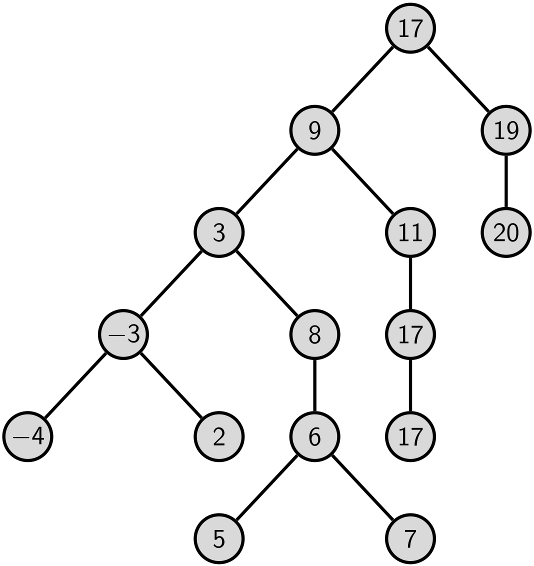
Collision

But there can be collision so it is $\mathcal{O}(1)$ on average ! In case of collision, 2 solutions

- Store in each bucket all the collisions ;
- Probe a empty spot (linear probing, quadratic probing, ...).

It is only $\mathcal{O}(1)$ on average if the load factor is good (e.g. $\ll 1$). When load factor get close to 1, increase N.

Binary Search Tree I



Binary Search Tree II

To get $\mathcal{O}(\log(n))$

- Balanced
- Comparison in $\mathcal{O}(1)$ (strings have $\mathcal{O}(m\log(n))$ where m is their length)
- Balancing operation in $\log(n)$

Balanced trees are tricky to code ! Use `std::set`, `TreeSet` and `std::map`, `TreeMap` !

When to use a BST or a hash map

- When you want an array to be indexed by another thing than an int
- When an array is too small and you don't need all
- (*BST only*) When you want a structure to be
  - an array to get $\mathcal{O}(\log(n))$ binary search access time
  - a linked list to get $\mathcal{O}(1)$ insertion time

Interesting example

Longest Increasing Subsequence : LIS, a classic !

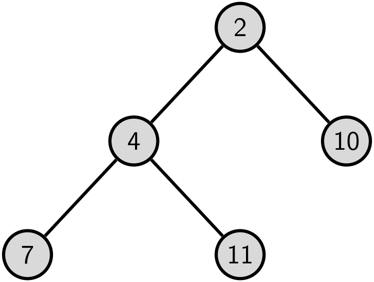
- Find an $\mathcal{O}(n^2)$ algorithm.
- Improve it to $\mathcal{O}(n \log(n))$.
- Solve problem 481 in uva with it (even if $\mathcal{O}(n^2)$ is enough).

Let's do a problem, shall we ?

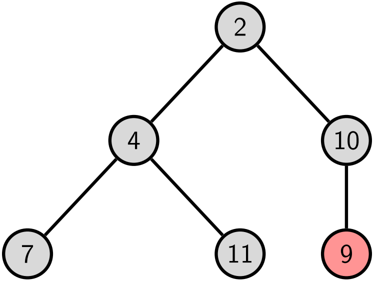
Do problem 10954 from uva !

Heap

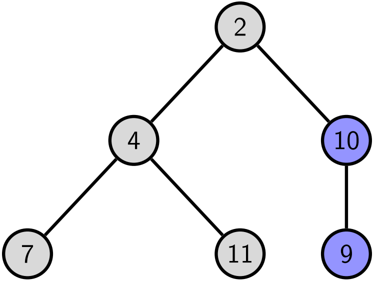
If we are only interested by the minimum, the BST is simplified and much faster and easier to code. Still $\mathcal{O}(\log(n))$ though.
The tree can be always complete !



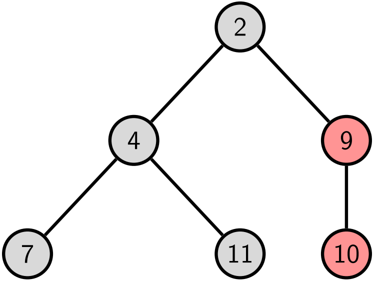
Push I



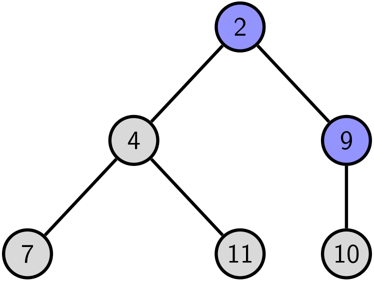
Push II



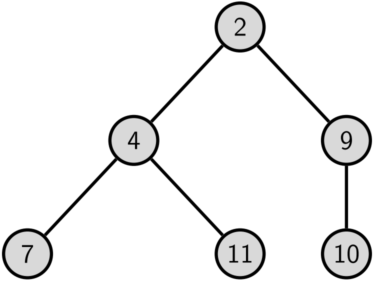
Push III



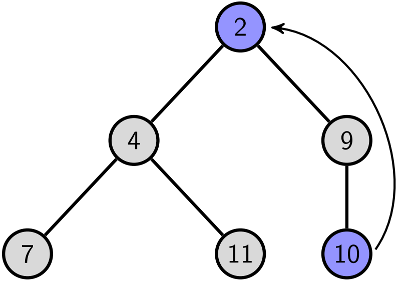
Push IV



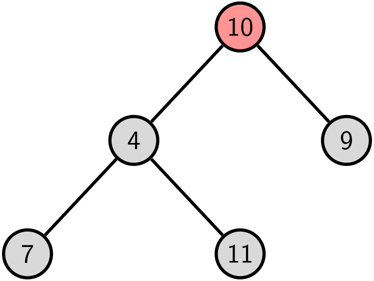
Push V



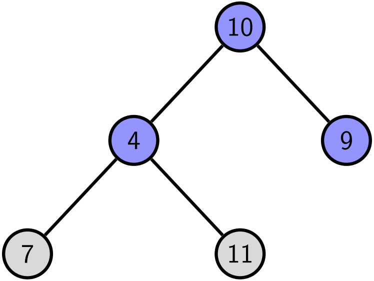
Pop I



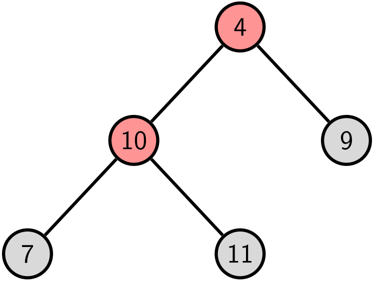
Pop II



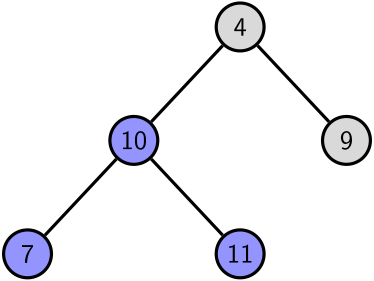
Pop III



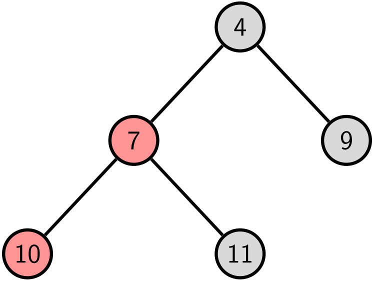
Pop IV



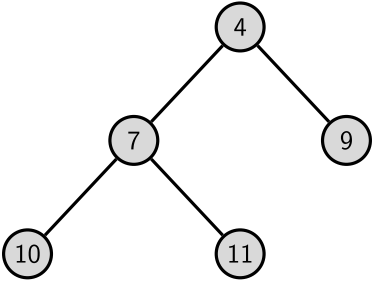
Pop V



Pop VI



Pop VII



Implémentation I

```
int hsize;
int hid[MAX]; // give id from index
int hval[MAX]; // give val from index
int hlookup[MAX]; // give index from id

void heap_swap (int a, int b) { // a and b are the indexes
int tmp = hval[a];
hval[a] = hval[b];
hval[b] = tmp;

hlookup[hid[a]] = b;
hlookup[hid[b]] = a;

tmp = hid[a];
hid[a] = hid[b];
hid[b] = tmp;
}

void heap_up (int a) { // a is the index
  int up = (a-1)/2;
if (0 < a && hval[a] < hval[up]) {
heap_swap(a, up);
heap_up(up);
}
}

```

Implémentation II

```
void heap_down (int a) { // a is the index
int left = 2*a+1, right = 2*a+2;
if (left < hsize && (hsize <= right || hval[left] < hval[right]) &&
     hval[left] < hval[a]) {
heap_swap(a, left);
heap_down(left);
}
else if (right < hsize && hval[right] < hval[a]) {
heap_swap(a, right);
heap_down(right);
}
}
```

A wild problem appear

n objects $0, \ldots, n-1$ are initially alone in a set. We have two types of query

- We merge 2 sets designated by 2 respecitve members
- We ask if 2 objects are in the same set

	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$
$(1,5)$	$\{1,5\}$	$\{2\}$	$\{3\}$	$\{4\}$	
$(2,4)$	$\{1,5\}$	$\{2,4\}$	$\{3\}$		
$(2,3)$	$\{1,5\}$	$\{2,3,4\}$			
$(3,4)$	$\{1,5\}$	$\{2,3,4\}$			
$(4,5)$	$\{1,2,3,4,5\}$				

Union find solution I

```
class UnionFind {
  int rank[MAX_N];
  int leader[MAX_N];
  UnionFind(int n) {
    memset(rank, 0, n * sizeof(int));
    for(int i = 0; i < n; i++) leader[i] = i;
  }
  int find(int a) {
    if(a != leader[a])
      leader[a] = find(leader[a]);
    return leader[a];
  }
  void union(int a, int b) {
    int leaderA = find(a);
    int leaderB = find(b);
    if(leaderA == leaderB) return;
    if(rank[leaderA] > rank[leaderB]) {
      union(leaderB, leaderA); return;
    }
    leader[leaderA] = leaderB;
    if (rank[leaderA] == rank[leaderB])
      rank[leaderB]++;
  }
};
```

Union find solution II

Complexity

time Amortized $\mathcal{O}(\alpha(n))$.

space $\mathcal{O}(n)$.

Example : gridland I

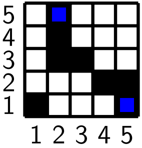
Grid $n \times m$ avec $1 \leq n, m \leq 1 \times 10^3$. Two squares are connected if they are activated and their is a path of activated square from one to the other. Initially, squares are only connected to themselves.
There is $1 \leq q \leq 1 \times 10^6$ queries

add `a x y` activate the square at (x, y)

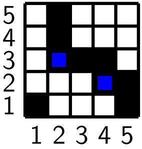
connected ? `c xa ya xb yb` see if (xa, ya) and (xb, yb) are connected.

Example : gridland II

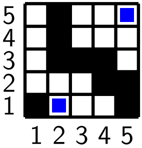
```
5 5 15
a 1 1
a 2 3
a 2 4
a 2 5
a 3 3
a 4 2
a 5 2
a 5 1
c 2 5 5 1
>> 0
```



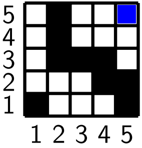

```
a 4 3
c 2 3 4 2
>> 1
```



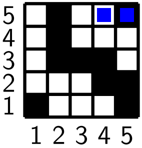

```
c 2 1 5 5
>> 0
```




```
c 4 4 4 4
>> 1
```




```
a 5 5
c 4 5 5 5
>> 0
```



Other examples

- uhunt.felix-halim.net 2.4.2.
- Connected componentents (overkill), Kruskal (impossible to get $\mathcal{O}(n\log(n))$ without it) (see next day !)

Segment Tree I

- Dynamic Range Minimum Query
- Dynamic Range Sum Query (Fenwick Tree is simpler)
- Dynamic Range Anyfunction Query

```
class SegmentTree {
  int[] st, A;
  int n;
  int left (int p) { return p << 1; }
  int right(int p) { return (p << 1) + 1; }

  void build(int p, int L, int R) {
    if (L == R)
      st[p] = L; // or R
    else {
      int mid = (L + R) / 2;
      build(left(p) , L       , mid);
      build(right(p), mid + 1, R);
      int p1 = st[left(p)], p2 = st[right(p)];
      st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  } }

  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 (i <= L && R <= j) return st[p]; // inside  query  range
```

Segment Tree II

```
    int mid = (L + R) / 2;
    int p1 = rmq(left(p) , L       , mid, i, j);
    int p2 = rmq(right(p), mid + 1, R   , i, j);

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

  int update(int p, int L, int R, int i, int j, int v) {
    if (i > R || j < L)                     // outside update range
      return st[p];
    //if (i <= L && R <= j) // could be lazy here !! Depends on
        application
    if (L == R) {
      A[i] = v;
      return st[p] = L; // or R
    }
    int mid = (L + R) / 2;
    int p1 = update(left(p) , L       , mid, i, j, v);
    int p2 = update(right(p), mid + 1, R   , i, j, v);
    return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  }

  public:

  SegmentTree(int[] _A) {
    A = _A; n = A.length;
```

Segment Tree III

```
    st = new int[4 * n];
    for (int i = 0; i < 4 * n; i++) st[i] = 0;
    build(1, 0, n - 1);
  }
  int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); }
  int update_point(int i, int v) {
    return update(1, 0, n - 1, i, i, v); }
  int update_interval(int i, int j, int v) {
    return update(1, 0, n-1, i, j, v); }
};
```

- IOI-2013 day 2 : “game”
- http://codeforces.com/problemset/problem/474/E

Fenwick Tree I

Dynamic Range Sum Query

What are the numbers smaller that 1011001000 ?

- 1011000???
- 1010??????
- 100???????
- 0?????????

$$\mathrm{rsq}(1011001000) = \mathrm{ft}(1011001000) + \mathrm{ft}(1011000000) + \mathrm{ft}(1010000000) + \mathrm{ft}(1000000000)$$

```
    adjust(01011001000,1):
        ft[01011010000]++
        ft[01011100000]++
        ft[01100000000]++
        ft[10000000000]++
```

```
x              00000000000000000100101101000000
~x             11111111111111111011010010111111
-x or (~x)+1   11111111111111111011010011000000
x & (-x)       00000000000000000000000001000000
```

Fenwick Tree II

```
class FenwickTree {
  int *ft;
  int n;
  int LSOne(int S) { return (S & (-S)); }
  public:
  FenwickTree(int n) { // ignore index 0
    this->n = n;
    ft = new int[n+1];
    for (int i = 0; i <= n; i++) ft[n] = 0;
  }
  int rsq(int b) {          // returns RSQ(1, b)      PRE 1 <= b <= n
    int sum = 0; for (; b > 0; b -= LSOne(b)) sum += ft[b];
    return sum;
  }
  int rsq(int a, int b) { // returns RSQ(a, b)      PRE 1 <= a,b <= n
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1));
  }
  void adjust(int k, int v) { // n = ft.size() - 1 PRE 1 <= k <= n
    for (; k <= n; k += LSOne(k)) ft[k] += v;
  }
};
```

Fenwick Tree III

Exemples :

- NWERC 2011 Problem C
- http://codeforces.com/contest/504/problem/B
- http://codeforces.com/problemset/problem/459/D

Sparse Table

Static Range Minimum Query
$m[i][j]$ stores the smallest of $[i; i + 2^j[$.
$\min([a; b[) = \min(m[a][k], m[b - 2^k][k])$ for k such that

$$2^{k-1} < b - a \leq 2^k$$

Tricky example

Get a tree flat with DFS then apply RSQ and RMQ.

- Least Common Ancestor : LCA
- http://codeforces.com/contest/383/problem/C

Conclusion

- Codeforces
- Codechef
- Usaco
- UVa online judge
- Competitive Programming 3 : https://sites.google.com/site/stevenhalim/
- uhunt.felix-halim.net
- IOI syllabus : http://people.ksp.sk/~misof/ioi-syllabus/