BeOI training - carnival 2015

Data structures II

Guillaume Derval Benoît Legat

Belgian Olympiad in Informatics

19 février 2015

A short reminder

You should already know...

- Arrays
- Linked Lists
- Stack & Queues

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.

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

Queue FIFO, First in First out.

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

Sets and Maps

Sets

"A collection that contains no duplicate elements" (Java 7 doc)
Three main actions :

- Add element
- Delete element
- Check presence of element

Maps

"An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value." (Java 7 doc)
Four main actions :

- Add (key,value) pair
- Delete value from its key
- Check presence of value by its key
- Get value from its key

Overview of the implementations

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

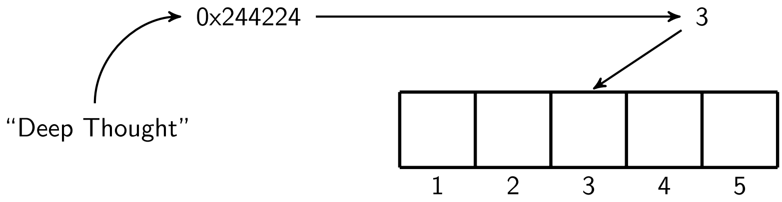
Hash maps I

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 maps II

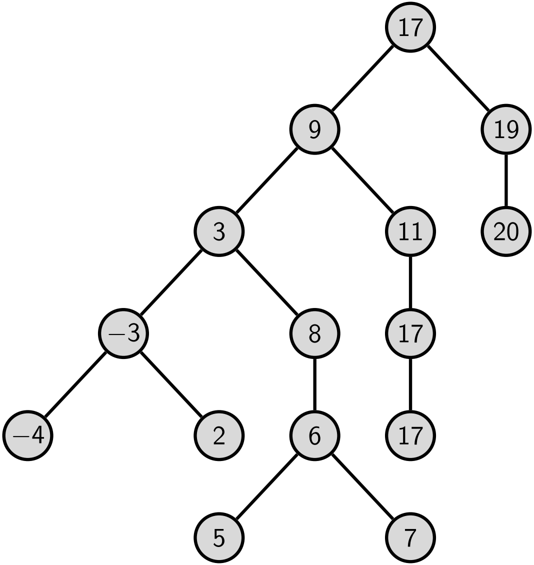
Collision

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` !

Array, TreeMap, or HashMap ?

When to use a BST or a hash map

- When you want an array to be indexed by another thing than an int
- When there is lot of empty "key" space in your array

Choosing between HashMap and TreeMap

- HashMap is "magically" $\mathcal{O}(1)$, most of the time faster than TreeMap
- But TreeMap can provide you all the values, ordered by keys

Heaps

A short introduction

Read problem 10954 from UVA !

A naive method

With a linked list or an array

1. Sort everything ($\mathcal{O}(n \log(n))$)
2. Take the two smallest numbers ($\mathcal{O}(1)$)
3. Make the sum ($\mathcal{O}(1)$)
4. Add the result to the collection (keeping it sorted) ($\mathcal{O}(n)$)
5. Go back to step 2 (do this n times)

$$\mathcal{O}(n^2)$$

We can do better

We want a structure such that...

1. Sort the collection ($\mathcal{O}(n\log(n))$)
2. Take the two smallest numbers ($\mathcal{O}(\log(n))$)
3. Make the sum ($\mathcal{O}(1)$)
4. Add the result to the collection (keeping it sorted) ($\mathcal{O}(\log(n))$)
5. Go back to step 2 (do this n times)

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

Heap

(here we explain Binary Heap, but there are more types of heaps)
A binary tree that satisfies these properties :

Shape

The tree is a complete binary tree (only the last level can be not fully filled)

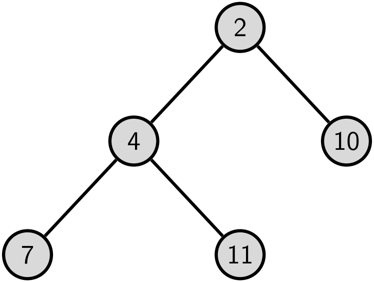
Heap property

All nodes are lesser than (or any other comparison you may choose) each of their children

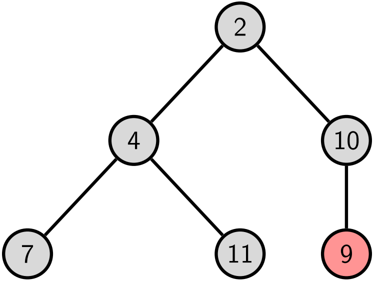
Heap vs. BST

- BST is for **searching**, Heap is for **sorting**
- BST is always fully sorted
- Only the first element of Heap is sorted
- Optimal BST is very difficult to implement, while Heap is very simple and effective
- Always choose the simplest data structure for what you want to do !

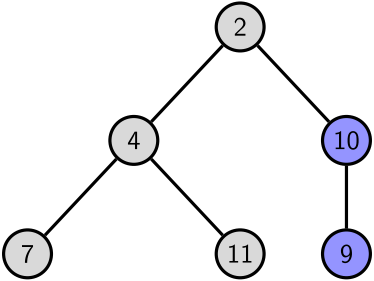
An example of Binary Heap



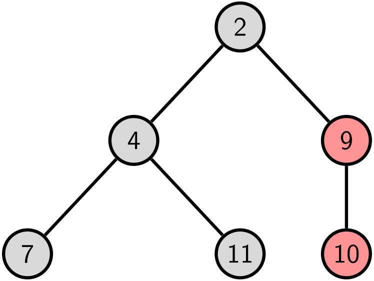
Push I



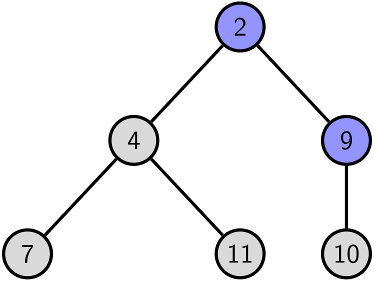
Push II



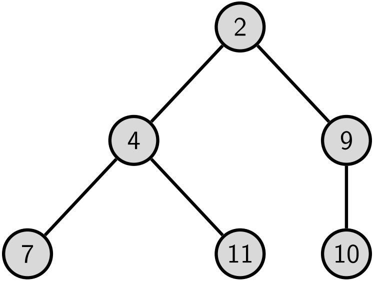
Push III



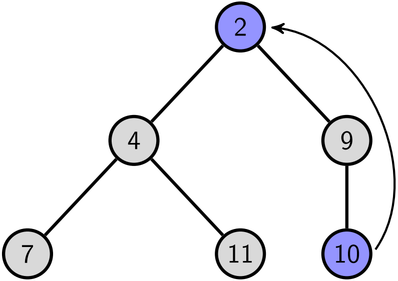
Push IV



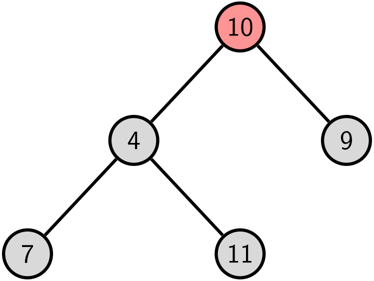
Push V



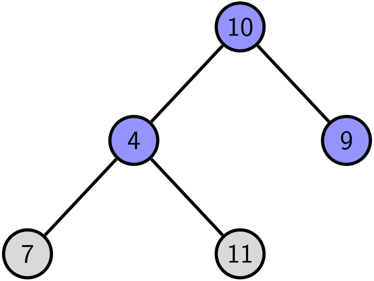
Pop I



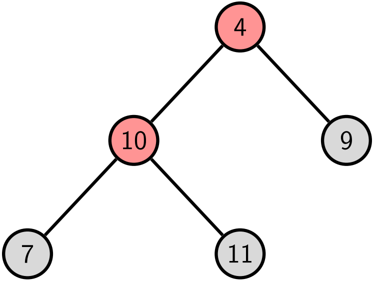
Pop II



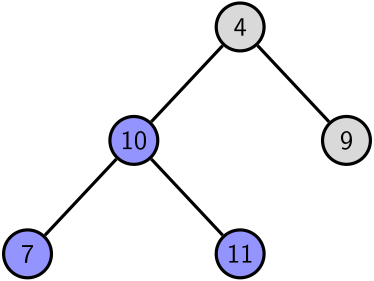
Pop III



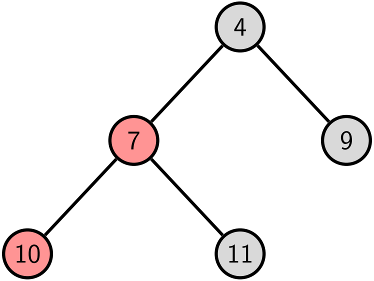
Pop IV



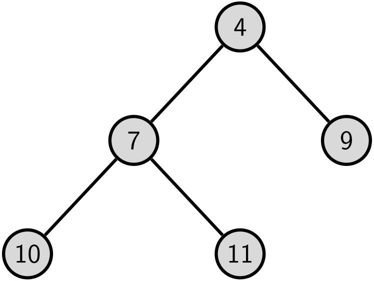
Pop V



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);
}
}
```

In C++/Java

- Do not implement it yourself ! Most of the time not needed.
- Java : `PriorityQueue`
- C++ : `std::priority_queue`

We are nearly done with heaps...

... Finish the problem 10954 from UVA ! (and/or do a short break before we view the union-find)

Union find

A wild problem appear

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

Union We merge 2 sets designated by 2 respective members

Find We ask if 2 objects are in the same set

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

A *naive* solution

- Array + Linked Lists
- Array containing linked list entries
- Class = first element of the list
- *Union* : merge two linked lists ($\mathcal{O}(n)$)
- *Find* : move to the first node of the list ($\mathcal{O}(n)$)

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

A *less naive* solution

- Array + Trees
- Array containing tree nodes
- Class = root of the tree
- *Union* : merge two trees by connecting one root to the other($\mathcal{O}(n)$)
- *Find* : move to the root of the tree ($\mathcal{O}(n)$)

(a better) $\mathcal{O}(n)$

A *non-that-naive* solution

- Array + Trees
- Array containing tree nodes
- Class = root of the tree
- *Union* : merge two trees by connecting root ***of the smaller tree*** to the other, ***which is taller, minimizing the resulting height*** ($\mathcal{O}(\log(n))$)
- *Find* : move to the root of the tree ($\mathcal{O}(\log(n))$)

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

A perfect solution

- Array + Trees
- Array containing tree nodes
- Class = root of the tree
- *Union* : merge two trees by connecting root of the smaller tree to the other, which is taller, minimizing the resulting height ($\mathcal{O}(\alpha(n))$)
- *Find* : move to the root of the tree, ***and move all nodes in the path to the root of the tree*** ($\mathcal{O}(\alpha(n))$)

$$\mathcal{O}(\alpha(n))\,!$$

$\alpha(n)$

$$\alpha(n) = f^{-1}(n)$$

where

$$f(n) = A(n, n)$$

where

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1, 1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m, n-1)) & \text{si } m > 0 \text{ et } n > 0. \end{cases}$$

Arckermann's function grows rapidly

n	$A(n,n)$
0	1
1	3
2	7
3	61
4	$2^{2^{2^{65536}}}$
5	[insert a very big number here]

$$\alpha(n) \leq 4 \text{ for } n \leq 2^{2^{2^{65536}}} \;!!!$$

(for all n in fact...)

Union find solution

```
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]++;
  }
};
```

Example : gridland I

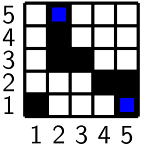
Grid $n \times m$ avec $1 \leq n, m \leq 1 \times 10^3$. Two squares are connected if they are activated and there is a path of activated square from one to the other. Initially, squares are only connected to themselves.
There is $1 \leq q \leq 1 \times 10^6$ queries

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

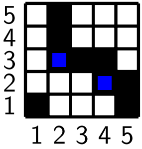
connected ? `c xa ya xb yb` see if (xa, ya) and (xb, yb) are connected.

Example : gridland II

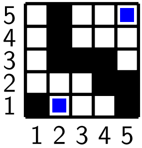
```
5 5 15
a 1 1
a 2 3
a 2 4
a 2 5
a 3 3
a 4 2
a 5 2
a 5 1
c 2 5 5 1
>> 0
```



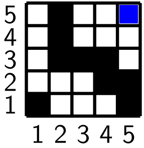

```
a 4 3
c 2 3 4 2
>> 1
```



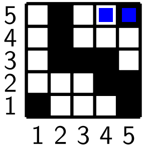

```
c 2 1 5 5
>> 0
```




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




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



Other examples

- 11503 from UVA (do it !)
- uhunt.felix-halim.net 2.4.2.
- Kruskal (impossible to get $\mathcal{O}(n \log(n))$ without it)

Segment Trees

Range Minimum Query

You have an array A of size n, with integer values inside.
Given two integer a and b ($a < b$), can you give me the minimum value of A between $A[a]$ and $A[b]$?

$$min\ A[i] \text{ for } a \leq i \leq b$$

Range Minimum Query

You have an array A of size n, with integer values inside.
Given two integer a, b, can you give me the minimum value of A between $A[a]$ and $A[b]$?

$$min\ A[i] \text{ for } a \leq i \leq b$$

100000 times ?

Similar problems

- Dynamic Range Minimum Query
- Dynamic Range Sum Query (Fenwick Tree is simpler)
- Dynamic Range [insert any function here] Query

I love naive solutions

Using an array...

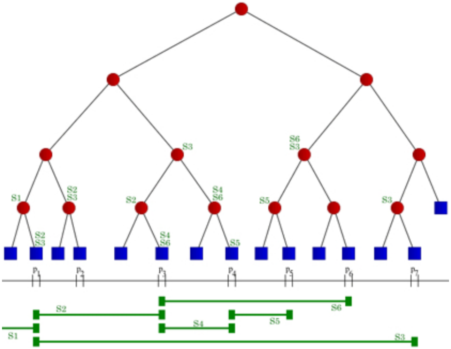
Spatial complexity $\mathcal{O}(n)$

Update time complexity $\mathcal{O}(1)$

Query time complexity $\mathcal{O}(n)$

Total Query time complexity $\mathcal{O}(mn) \Rightarrow$ ***TLE***

A better solution : the segment tree



Spatial complexity $\mathcal{O}(n\log(n))$

Update time complexity $\mathcal{O}(\log(n))$

Query time complexity $\mathcal{O}(\log(n))$

Total Query time complexity $\mathcal{O}(m\log(n)) \Rightarrow \boldsymbol{AC}$

Segment Tree implementation I

```
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
    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;
```

Segment Tree implementation II

```
    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;
    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) {
```

Segment Tree implementation III

```
    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

Fenwick Tree I

Dynamic Range Sum Query

What are the numbers smaller that 1011001000 ?

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

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

```
    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

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