Contest programming

Linear data structures

Maksym Planeta

20.04.2018

Table of Contents

Update on organisation

- 11 weeks
- 9 exercises:
  - Introduction
  - 7 lectures + exercises
  - A practice session
- Contest on June, 16th
- After Lange Nacht der Wissenschaft

Updated outline

1. Introduction
2. Linear data structures. Long arithmetic.
3. Greedy and Divide and conquer algorithms. Dynamic programming.
4. Algorithms on string. Sorting and searching.
5. Computational geometry. Floating point arithmetic.
6. Mathematical and combinatorial problems.
7. Algorithms on graphs
8. Algorithms on graphs
9. Practice session
10. Contest

Updated outline

1. Introduction
2. Linear data structures. Long arithmetic.
3. Greedy and Divide and conquer algorithms. Dynamic programming.
4. Algorithms on string. Sorting and searching.
5. Computational geometry. Floating point arithmetic.
6. Mathematical and combinatorial problems.
7. Algorithms on graphs
8. Algorithms on graphs
9. Practice session
10. Contest

Linear data structures

- std :: vector<int>
- std :: vector<bool>
- int array [4]
- std :: array<int, 4>
- std :: bitset <444>

Searching

- std :: lower_bound
- std :: upper_bound
- std :: binary_search
- std :: find
- std :: max_element
- std :: min_element

Sorting

- std :: sort
- std :: partial_sort
- std :: stable_sort

Permutations

- std :: next_permutation
- std :: prev_permutation

10038 - Jolly Jumpers

Problem Statement

A sequence of $n > 0$ integers is called a *jolly jumper* if the absolute values of the difference between successive elements take on all the values 1 through $n - 1$. For instance,

```
1 4 2 3
```

is a jolly jumper, because the absolutes differences are 3, 2, and 1 respectively. The definition implies that any sequence of a single integer is a jolly jumper. You are to write a program to determine whether or not each of a number of sequences is a jolly jumper.

10038 - Jolly Jumpers (contd.)

Input

Each line of input contains an integer $n \leq 3000$ followed by n integers representing the sequence.

Output

For each line of input, generate a line of output saying "Jolly" or "Not jolly".

Sample Input

```
4 1 4 2 3
5 1 4 2 -1 6
```

Sample Output

```
Jolly
Not Jolly
```

10038 - Jolly Jumpers (Solution)

```
#include <bits/stdc++.h>
using namespace std;

#define N 3000
int jolly[N];

int main() {
  int n, a, b;

  while (scanf("%d %d", &n, &a) > 0) {
    // Body follows
  }
}
```

10038 - Jolly Jumpers (Solution)

```cpp
// The body
fill(jolly, jolly + N, 0);
for (int i = 1; i < n; i++, a = b) {
  scanf("%d", &b);
  jolly[abs(a - b)]++;
}

auto f = find_if(jolly+1, jolly + n,
                 [](int count) {return count != 1;})
if (n == 1 || f == (jolly + n)) {
  cout << "Jolly\n";
} else {
  cout << "Not jolly\n";
}
```

Linear data structures

- N-dimensional arrays
  - Vector of vectors of vectors ...
  - 1D-vector with proper indexing
    - Row major
    - Column major
- Heaps
- Trees
- Queues

Disjoint Sets

- Set of sets
- Sets may change
- Find which set an item belongs to
- Merge sets efficiently

Disjoint Sets: Operations

- Make-Set(x)
  Creates a set with a single element x
- Union(x, y)
  Unite sets with elements x and y
- Find-Set(x)
  Return a representative element of the set containing x

Disjoint Sets: Connected components

```
Connected-Components(G)
for each vertex v ∈ G.V
        Make-Set(v)
for each edge (u, v) ∈ G.E
        if Find-Set(u) ≠ Find-Set(v)
                Union(u, v)
```

```
Same-Components(G)
if Find-Set(u) == Find-Set(v)
        return True
else return False
```

Disjoint Sets: Implementation

Make-Set(x)

$x.p = x$
$x.rank = 0$

Union(x, y)

Link(Find-Set(x), Find-Set(y))

Link(x, y)

if $x.rank > y.rank$
    $y.p = x$
else $x.p = y$
    if $x.rank == y.rank$
        $y.rank = y.rank + 1$

Find-Set(x)

if $x \neq x.p$
    $x.p =$ Find-Set($x.p$)
return $x.p$

Range Minimum Queries

- Given an array $a_0 \dots a_n$
- Find a minimum value in range $[i, j]$
- Trivial algorithm complexity $O(n)$
- With m queries the complexity is $O(mn)$
- Possible complexity: $O(n + m \log n)$

Segment Tree

`https://visualgo.net/en/segmenttree`

- Build a tree with $O(2n)$ nodes

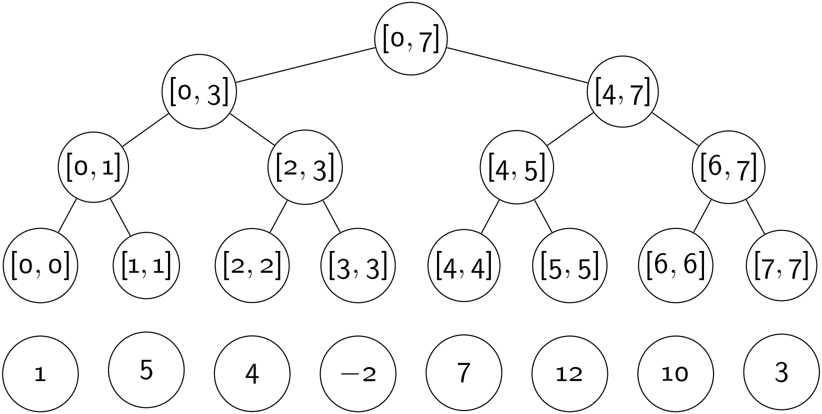

Figure: Segment Tree

Segment Tree: Structure

- Mark a parent for each node

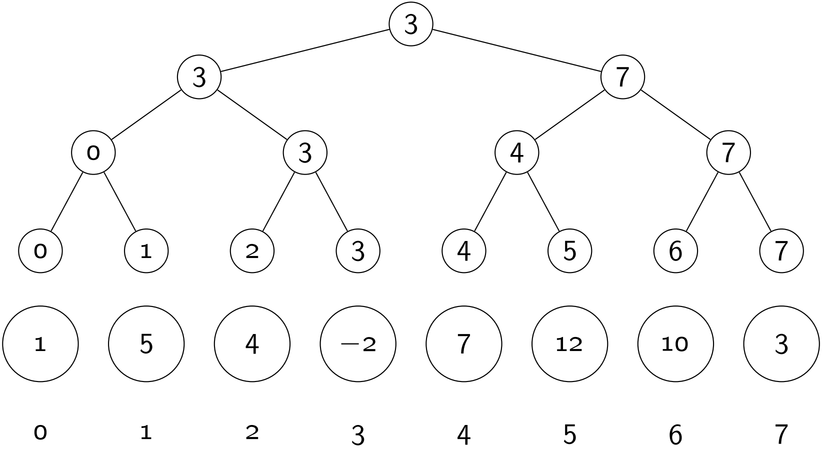

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

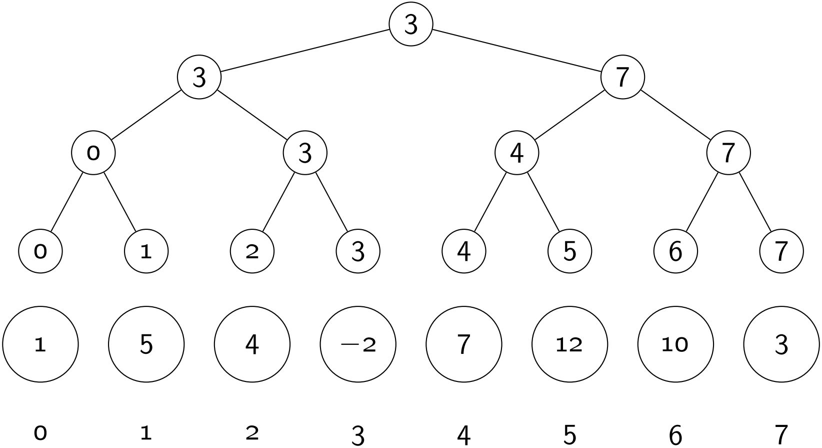

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

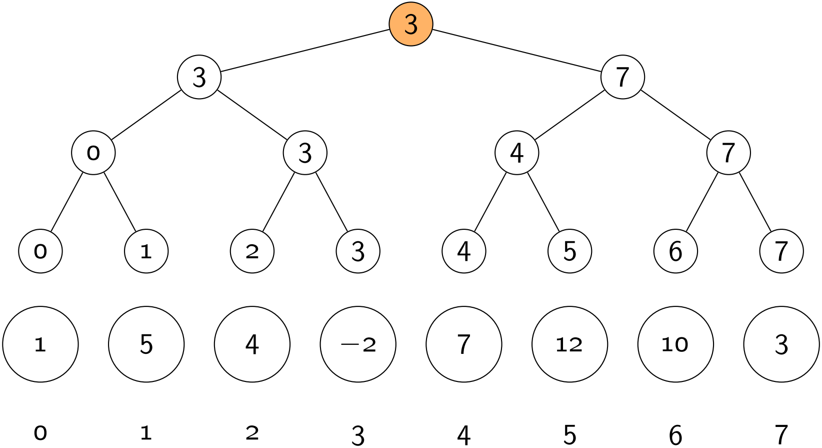

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

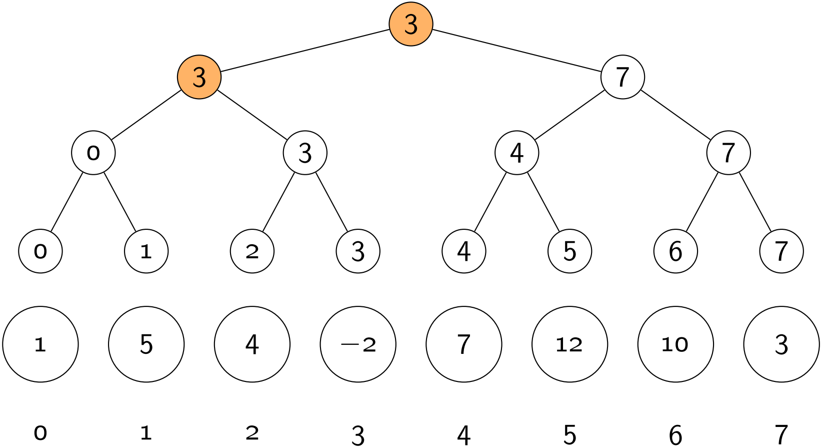

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

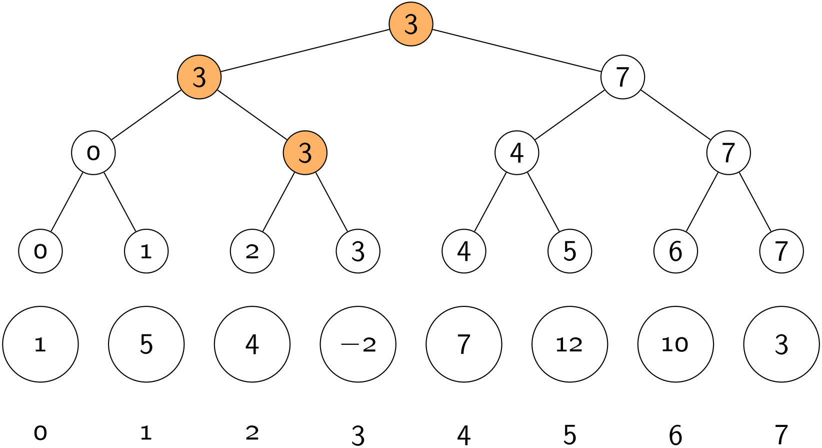

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

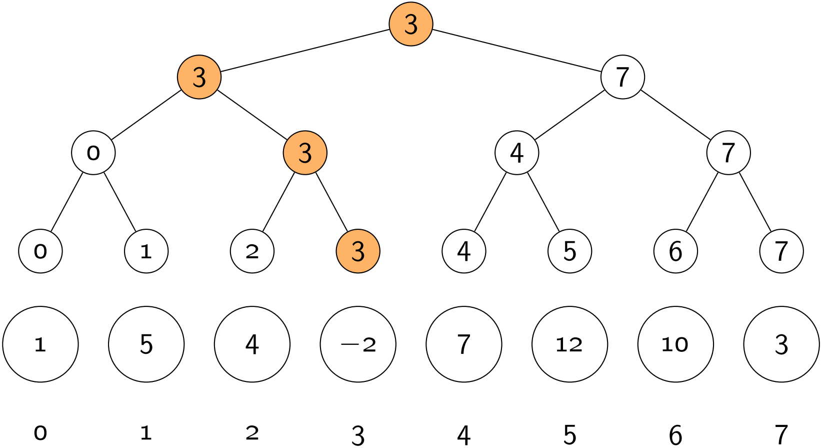

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

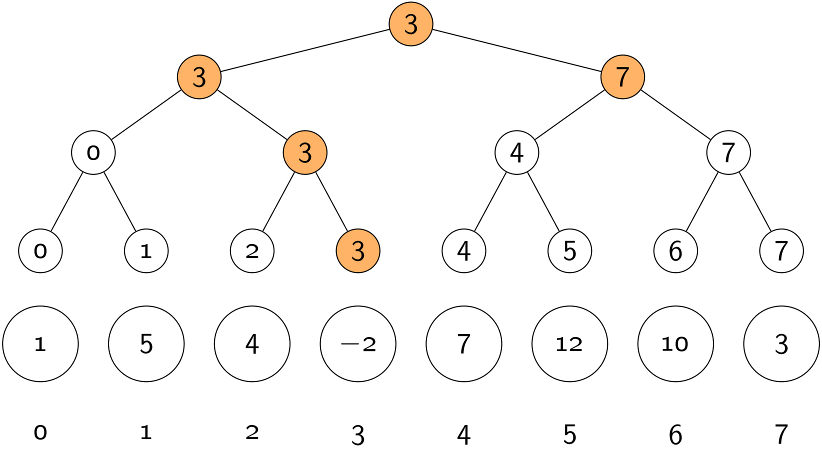

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

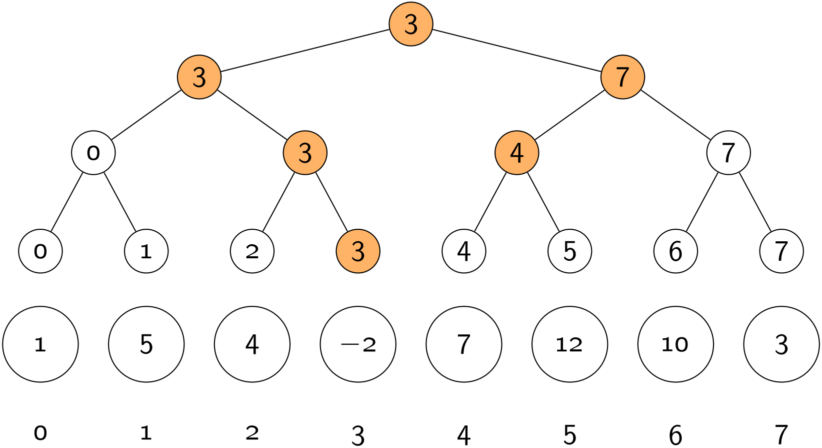

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

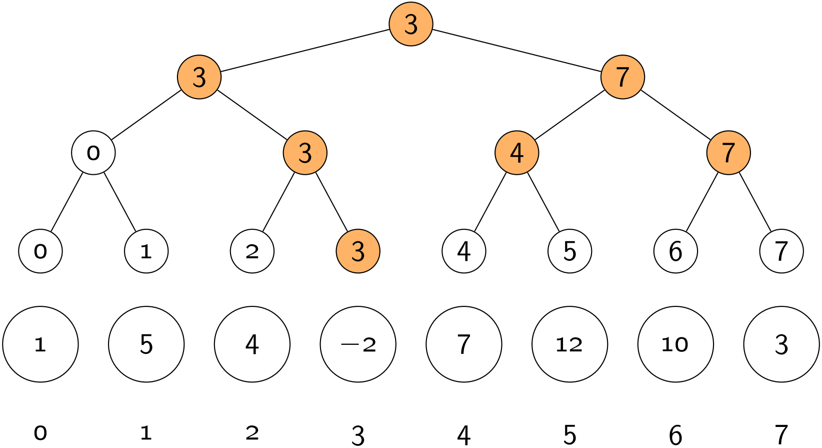

Figure: Segment Tree

Segment Tree: Query

- Execute a query $RMQ(3, 6)$

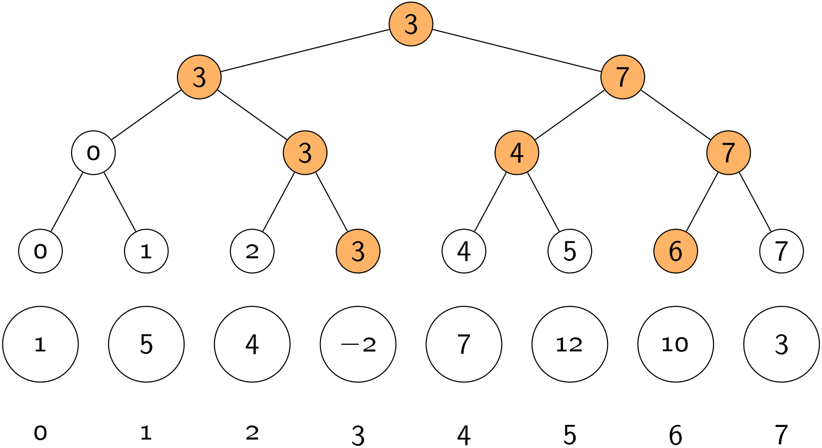

Figure: Segment Tree

Segment Tree: Building

Build-Segments(p, L, R)

if $L == R$
  $tree[p] = L$
else Build-Segments(Left(p), L, $\left\lceil \frac{L+R}{2} \right\rceil$)
  Build-Segments(Right(p), $\left\lceil \frac{L+R}{2} \right\rceil + 1$, R)
  $p_1 = tree[\text{Left}(p)]$
  $p_2 = tree[\text{Right}(p)]$
  if $value[p_1] \leq value[p_2]$
    $tree[p] = p_1$
  else $tree[p] = p_2$

Segment Tree: Querying

Range-Min-Query(p, L, R, i, j)

if $i > R$ or $j < L$
    return nil
if $L \geq i$ and $R \leq j$
    return $tree[p]$
    $p_1 =$ Range-Min-Query(Left(p), L, $\lceil (L+R)/2 \rceil$, i, j)
    $p_2 =$ Range-Min-Query(Right(p), $\lceil (L+R)/2 \rceil + 1$, R, i, j)
    if $p_1 ==$ nil
        return p_2
    if $p_2 ==$ nil
        return p_1
    if $value[p_1] \leq value[p_2]$
        return p_1
    else return p_2

Long arithmetic

java.math.BigInteger

- add
- subtract
- pow
- modPow
- multiply
- divide

Practice

Solve following set of problems in a group:

1. 00394 – Mapmaker
2. 00123 – Searching Quickly
3. 00146 – ID Codes
4. 10608 – Friends
5. 10523 – Very Easy!!!
6. 11034 – Ferry Loading IV
7. 11235 – Frequent Values
8. 11402 – Ahoy, Pirates
9. 11448 – Who Said Crisis?
10. 11991 – Easy Problem from Rujia Liu?

Home reading

Cormen.

1. Recommended
   - Section 4 (intro)
   - Section 4.1
2. Optional
   - Section 21 (intro)
   - Section 21.1 - 21.3

Literature

Thomas H Cormen.
Introduction to algorithms.
MIT press, 2009.

Steven Halim and Felix Halim.
Competitive Programming 3.
Lulu Independent Publish, 2013.