Contest programming
Linear data structures

Maksym Planeta

20.04.2018
Table of Contents

Organisation

Linear Data Structures

Long arithmetic

Practice
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.
5. Computational geometry. Floating point arithmetic.
7. Algorithms on graphs
8. Algorithms on graphs
9. Practice session
10. Contest
Updated outline

1. Introduction
2. Linear data structures. Long arithmetic.
5. Computational geometry. Floating point arithmetic.
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<4>`
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`
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.
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
#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
    }
}
// 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($G$)
1. for each vertex $v \in G. V$
   2. Make-Set($v$)
3. for each edge $(u, v) \in G. E$
4. if Find-Set($u$) $\neq$ Find-Set($v$)
5. Union($u, v$)

Same-Components($G$)
1. if Find-Set($u$) $==$ Find-Set($v$)
2. return True
3. else return False
Disjoint Sets: Implementation

Make-Set(\(x\))
\[
1 \quad x.p = x \\
2 \quad x.rank = 0
\]

Union(\(x, y\))
\[
1 \quad \text{Link(Find-Set}(x), \text{Find-Set}(y))
\]

Link(\(x, y\))
\[
1 \quad \text{if } x.rank > y.rank \\
2 \quad \text{y.p = x} \\
3 \quad \text{else } x.p = y \\
4 \quad \text{if } x.rank == y.rank \\
5 \quad \text{y.rank = y.rank + 1}
\]

Find-Set(\(x\))
\[
1 \quad \text{if } x \neq x.p \\
2 \quad x.p = \text{Find-Set}(x.p) \\
3 \quad \text{return } x.p
\]
Range Minimum Queries

- Given an array $a_0 \ldots a_n$
- 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

![Segment Tree Diagram]

**Figure**: Segment Tree
Segment Tree: Query

- Execute a query $RMQ(3, 6)$
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 $RMQ(3, 6)$
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 $RMQ(3, 6)$
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\((p, L, R)\)

1. **if** \(L == R\)
2. \(\text{tree}[p] = L\)
3. **else** Build-Segments(Left\((p)\), \(L, \left\lceil \frac{L+R}{2} \right\rceil \))
4. Build-Segments(Right\((p)\), \(\left\lceil \frac{L+R}{2} \right\rceil + 1, R\))
5. \(p_1 = \text{tree}[\text{Left}(p)]\)
6. \(p_2 = \text{tree}[\text{Right}(p)]\)
7. **if** value\([p_1]\) \(\leq\) value\([p_2]\)
8. \(\text{tree}[p] = p_1\)
9. **else** \(\text{tree}[p] = p_2\)
Segment Tree: Querying

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

1. if $i > R$ or $j < L$
   return nil
2. if $L \geq i$ and $R \leq j$
   return $tree[p]$
3. $p_1 = \text{Range-Min-Query}(\text{Left}(p), L, \lceil(L + R)/2\rceil, i, j)$
4. $p_2 = \text{Range-Min-Query}(\text{Right}(p), \lceil(L + R)/2\rceil + 1, R, i, j)$
5. if $p_1 ==$ nil
   return $p_2$
6. if $p_2 ==$ nil
   return $p_1$
7. if $\text{value}[p_1] \leq \text{value}[p_2]$
   return $p_1$
8. 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. 

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