INTRODUCTION
to competitive programming

Thursday 7th February, 2019

Santiago Hincapie-Potes

Universidad EAFIT

TODAY WE'RE GOING TO COVER

1. Basic data types
2. Big integers
3. Complexity theory basis
4. Data structures you already know

BASIC DATA TYPES

YOU SHOULD ALL BE FAMILIAR WITH

- → `bool`: a boolean (`true`/`false`)
- → `char`: an 8-bit signed integer (often used to represent characters with ASCII)
- → `short`: a 16-bit signed integer
- → `int`: a 32-bit signed integer
- → `long long`: a 64-bit signed integer
- → `float`: a 32-bit floating-point number
- → `double`: a 64-bit floating-point number
- → `long double`: a 128-bit floating-point number
- → `string`: a string of characters

RANGE

Type	Bytes	Min value	Max value
unsigned char	1	0	255
unsigned short	2	0	65535
unsigned int	4	0	4294967295
unsigned long long	8	0	18446744073709551615
	n	0	$2^{8n} - 1$

SIMPLE ADDITION

https://open.kattis.com/problems/simpleaddition

BIG INTEGERS

BIG INTEGER

→ What if we need to represent and do computations with very large integers, i.e. something that doesn't fit in a `long long`

BIG INTEGER

- → What if we need to represent and do computations with very large integers, i.e. something that doesn't fit in a `long long`
- → Simple idea: Store the integer as a string

BIG INTEGER

- → What if we need to represent and do computations with very large integers, i.e. something that doesn't fit in a `long long`
- → Simple idea: Store the integer as a string
- → But how do we perform arithmetic on a pair of strings?
- → We can use the same algorithms as we learned in elementary school

BIG INTEGER

- → What if we need to represent and do computations with very large integers, i.e. something that doesn't fit in a `long long`
- → Simple idea: Store the integer as a string
- → But how do we perform arithmetic on a pair of strings?
- → We can use the same algorithms as we learned in elementary school
- → C/C++ → implement from scratch
- → Java → `java.math.BigInteger`
- → python → default integers

COMPLEXITY THEORY BASIS

COMPLEXITY TABLE

n	Worst AC Algorithm
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 10^6$	$O(n \log n)$ o $O(n)$
n largest	$O(1)$ o $O(\log n)$

DATA STRUCTURES YOU ALREADY KNOW

VECTOR

- → `vector<type>` in C++, with `push_back` and `pop_back`
- → `ArrayList<Type>` in Java, with `.add` and `.remove(list.size()-1)`
- → `list` in Python, with `.append` and `.pop`
- → Indexing as `list[i]` or `list.git(i)`
- → Can be use as a **stack**
- → $\mathcal{O}(1)$ (Amortized)

QUEUE

- → `queue<type>` in C++, with `push`, `front` and `pop`
- → `ArrayDeque<Type>` in Java, with `.add`, `getFist` and `.remove`
- → `collections.deque` in Python, with `.append`, `deque[0]` and `.popleft`
- → Can be use as a **queue**
- → $\mathcal{O}(1)$ (Amortized)

DEQUE

- → `queue<type>` in C++, with `push_front`, `push_back`, `pop_front` and `pop_back`
- → `ArrayDeque<Type>` in Java, with `.addFirst`, `.addLast`, `.removeFirst` and `.removeLast`
- → `collections.deque` in Python, with `.appendleft`, `.append`, `popleft` and `.pop`
- → Indexing as `list[i]` (**Not in Java!**)
- → Can be use as a **queue** or **stack**
- → $\mathcal{O}(1)$ (Amortized)

HASHSET

- → `unordered_set<type>` in C++
- → `HashSet<Type>` in Java
- → `set` in python
- → insert, delete and consult membership in $\mathcal{O}(1)$

HASHMAP

- → `unordered_map<type1, type2>` in C++
- → `HashMap<Type1, Type2>` in Java
- → `dict` in python
- → insert, delete and consult membership in $\mathcal{O}(1)$
- → Just like HashSet but with key-value storage

TREESET

- → `set<type>` in C++
- → `TreeSet<Type>` in Java
- → ∼`collections.OrderedDict` in python
- → insert, delete, consult membership, lower_bound and upper_bound in $\mathcal{O}(\log n)$

TREEMAP

- → `map<type1, type2>` in C++
- → `TreeMap<Type1, Type2>` in Java
- → `collections.OrderedDict` in python
- → insert, delete, consult membership, lower_bound and upper_bound in $\mathcal{O}(\log n)$
- → Just like TreeSet but with key-value storage

CONTEST

https://a2oj.com/register?ID=38718

NEXT WEEK

Problem solving paradigms