Universiteit Leiden

Competitive Programming

Frank Takes

LIACS, Leiden University

https://liacs.leidenuniv.nl/~takesfw/CP

Lecture 2 — Data structures and libraries

Universiteit Leiden

Recap

After last week . . .

Universiteit Leiden

- You are familiar with the basic concepts of competitive programming,
- have the practical individual skills in C or C++ to participate in a programming contest,
- know roughly how input size constraints dictate maximum time complexity,
- can use DOMjudge and have an account at `http://domjudge.liacs.nl`,
- know the specifics of program input/output via `stdin`/`stdout`
- understand how to output the various data types,
- and have seen a dynamic input problem.

RANK	TEAM	SCORE	BOOLFIND	FLTCMP	HELLO
1	-[--->+<]>-----.[-->+++<]>+.-----.------------.+++++++.-.	3 316	106 2 tries	92 2 tries	78 1 try
2	0x000000	3 321	107 1 try	115 2 tries	79 1 try
3	elgar	3 326	107 2 tries	97 2 tries	82 1 try
4	BIsDeTeamNaam	3 343	113 4 tries	92 1 try	78 1 try
5	joepert	3 359	139 1 try	114 2 tries	86 1 try
6	Being Judged	3 389	100 4 tries	104 2 tries	105 1 try
7	Tema	3 439	149 4 tries	109 3 tries	81 1 try
8	Putman	3 444	128 8 tries	93 1 try	83 1 try
9	;)	3 453	155 4 tries	113 3 tries	85 1 try
10	156	3 495	201 2 tries	166 2 tries	88 1 try
11	Bergman	3 512	144 6 tries	161 2 tries	87 1 try
12	Janssen Leiden University	3 519	155 4 tries	166 2 tries	98 2 tries
13	BruteForce	3 540	179 3 tries	162 4 tries	99 1 try
14	mathe	3 561	135 2 tries	163 3 tries	103 6 tries
15	TeamLuuk	3 6379	6021 7 tries	109 2 tries	89 2 tries
16	1551396	3 12507	158 6 tries	6112 1 try	6117 2 tries
17	CPteam	2 265	4 tries	146 2 tries	99 1 try
18	mocp	0 0			1 try
	T	0 0	3 tries		

Practicalities

Universiteit Leiden

- Judgehost hung: manual action required after large file was uploaded (please be nice)
- Time is now fixed on DOMjudge server
- C++ 11 (compiled using `g++ -g -O2 -std=gnu++11`)
- Java 8
- Character encoding: `UTF-8`
- Clarification requests "work"

Your programming language(s)

Universiteit Leiden

1. C, C++ and Java
2. C and Java
3. C++ and Java
4. C and C++
5. C
6. C++
7. Java

Data structures

Universiteit Leiden

Linear data structures

Standard Template Library

Universiteit Leiden

- C++ Standard Library
- Data structures
  e.g., `vector`, `list`, `queue`, `stack`, `set`, `map`
- Functions in `#include<algorithm>` (but also `bitset` and `string`),
  e.g., `sort`, `min`, `max`, `count`, `binary_search`
- Generally fast enough
- (Re-)familiarize yourself:
  `http://www.cplusplus.com/reference/stl`

Linear data structures

- Static: arrays `int A[100];`
  Or `array<int, 3> A;`
- Dynamic: C++ STL vector `vector<int> A(100);`
  Alternative for `vector<bool> B` `bitset<100> A;`
- Linked lists (e.g., using `list<int> A`) are rarely used

Linear data structures

Universiteit Leiden

- Static: arrays `int A[100];`
  Or `array<int, 3> A;`
- Dynamic: C++ STL vector `vector<int> A(100);`
  Alternative for `vector<bool> B` `bitset<100> A;`
- Linked lists (e.g., using `list<int> A`) are rarely used
- Common operations:
  - Sort
  - Search

Problem type: sorting

- Given some linear data structure, sort it
- Complexity: $O(n \log n)$
- $O(n)$ for special cases (e.g., small number of distinct elements)
- In practice: `algorithm::sort`
  - Works for basic data types (`int`, `double` etc.)

```
vector<int> myvalues;
...
sort(myvalues.begin(), myvalues.end());
```

  - Custom data types if the comparison operator < is defined

Sorting with custom data types

Universiteit Leiden

```
// a coin is worth its value, but worth slightly more when unused
struct Coin {
    int value;
    bool used;
}; // Coin

bool operator < (const Coin & a, const Coin & b) {
    if(a.value == b.value)
        return(a.used > b.used);
    return a.value < b.value;
} // < operator for Coin

...
vector<Coin> coins;
... // fill it with data
sort(coins.begin(), coins.end());
...
```

Problem type: search

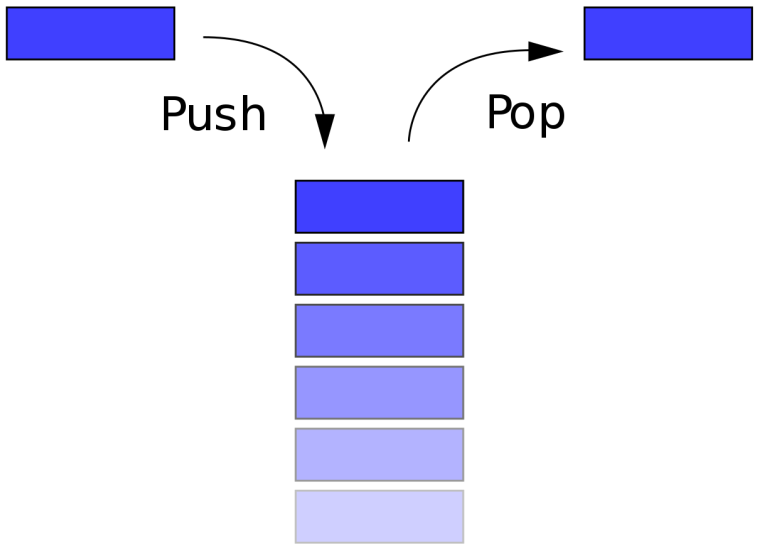
Universiteit Leiden

- Given some linear data structure, search for an element
- Search for an element once or less than $log(n)$ times: linear scan
- Searching more often:
  1. sort the list in $O(n \log n)$
  2. use binary search to find elements in $O(\log n)$
     e.g., from `algorithm::binary_search`
     or the easier `algorithm::lower_bound`

Stack

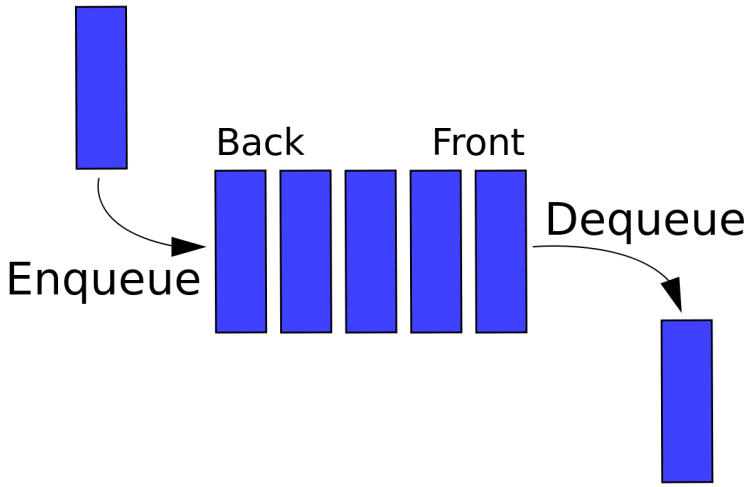
- Stack
  Operations: `push()`, `top()`, `pop()`, etc.
  (STL implementation is very similar to `vector`)

```
stack<int> S;
```



Queue

- Queue `queue<int> Q;`
  Operations: `front()`, `push()`, `pop()`, etc.



- Double-ended queue `deque<int> Q;`
  Operations: `front()`, `back()`, `push_back()`, `pop()`, etc.

Universiteit Leiden

Nonlinear data structures

Priority queue

Universiteit Leiden

```
// a coin is worth its value, but worth slightly more when unused
struct Coin {
    int value;
    bool used;
}; // Coin

struct CustomCompare {
    bool operator() (const Coin & a, const Coin & b) {
        if(a.value == b.value)
            return(a.used > b.used);
        return a.value < b.value;
    } // < operator for myCoin
}; / /CustomCompare

...
priority_queue<int,std::vector<int>,CustomCompare> Q;
// default is std::less<int>, sometimes std::greater<int> suffices
...
```

Mapping and hashing

Universiteit Leiden

- Given some key, check if it exists `set<keytype> S;`
- Given some key, find a bit of data `map<keytype, datatype> M;`

Sets

Universiteit Leiden

```cpp
#include <iostream>
#include <set>
using namespace std;

int main() {
    int t;
    set<int> S;
    int N = 100;

    for(int i=0; i<N; i++)
        S.insert(rand() % 100);

    cout << S.size() << endl;
    // ouputs 68 (of course, this is dependent on random seed),
    // value of 100 expected as N gets larger

    return 0;
} // main
```

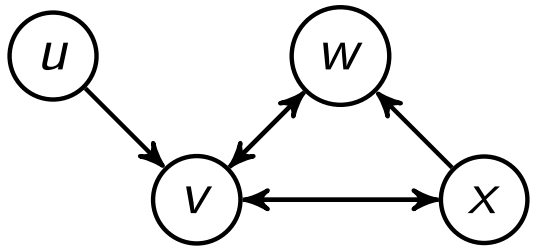
Map

Universiteit Leiden

```cpp
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main() {
    int t;
    map<string, int> M;
    M["een"] = 1;
    M["dertien"] = 13;

    cout << M["een"] + M["dertien"] << endl;
    // the above outputs: 14

    return 0;
} // main
```

Graphs

- Directed graph $G = (V, E)$ with $E \subseteq V \times V$
- Variants: undirected (simple) graph, bipartite graph, weighted graph, directed acyclic graph (dag), tree
- Example below has $V = \{u, v, w, x\}$ and $E = \{(u, v), (v, w), (v, x), (w, v), (x, v), (x, w)\}$ so, $|V| = 4$ and $|E| = 6$



Graph datastructures

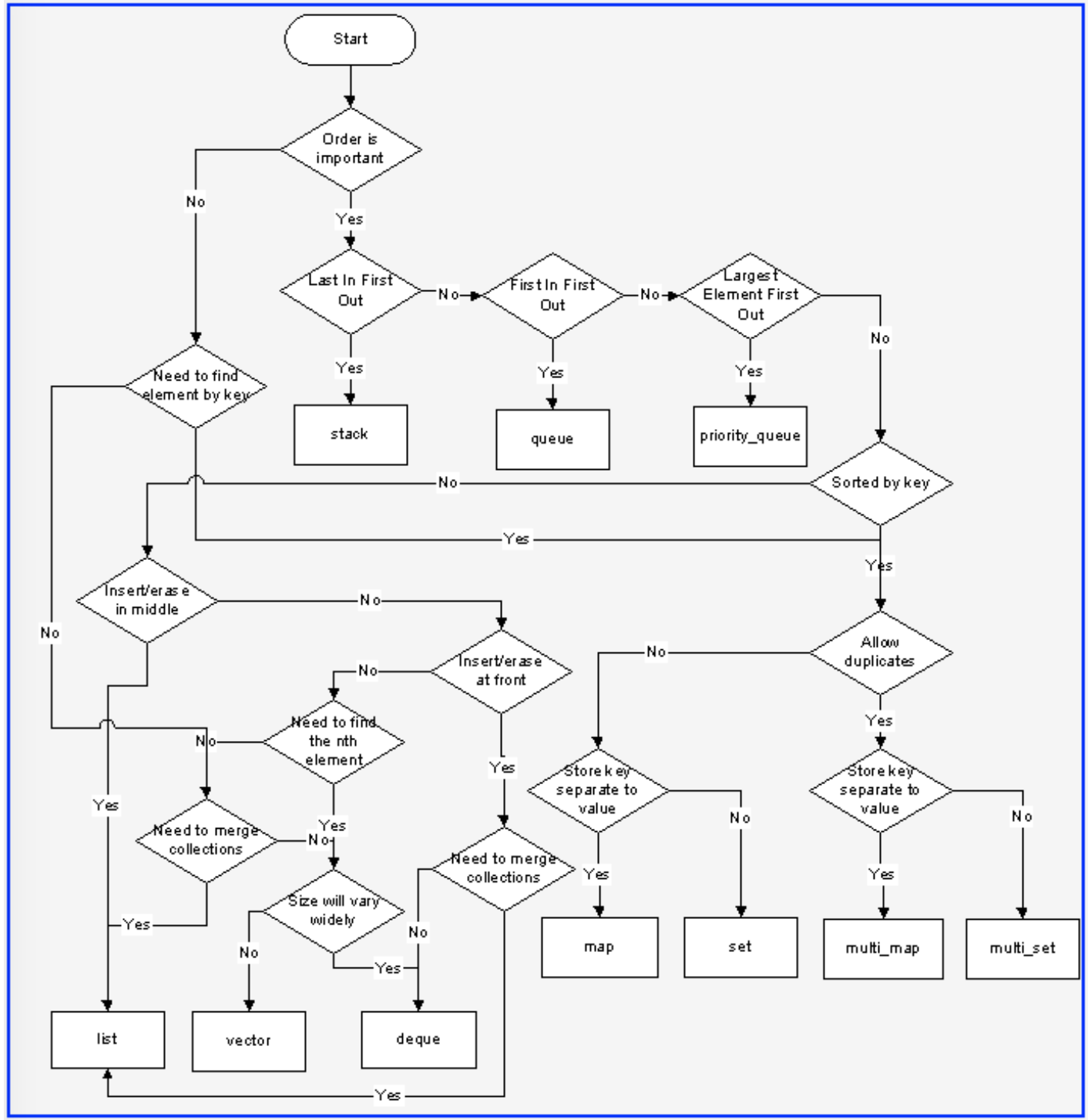
- Adjacency matrix: store whether all `N` $\times$ `N` node pairs are linked
  `bool A[N][N];`
  `int` for weights, additional instances for edge labels
- Adjacency list: store the neighbors of all `N` nodes
  `vector< vector<int> > A(N);`
  store something more complex than `int` for edge labels or weights
- Edge list: store all `M` existing edges linearly
  `int source[M]; int target[M];`
  additional arrays for edge labels or weights

Here, `N` is the number of nodes and `M` the number of edges

Graph datastructures

Universiteit Leiden

	Adjacency list	**Adjacency matrix**	**Edge list**
Storage	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|E\|)$
Add vertex	$O(1)$	$O(\|V\|^2)$	$O(1)$
Add edge	$O(1)$	$O(1)$	$O(1)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|E\|)$
Remove edge	$O(\|E\|)$	$O(1)$	$O(\|E\|)$
Query $(u, v) \in E$?	$O(\|V\|)$	$O(1)$	$O(\|E\|)$


Start
Order is important
No
Yes
Last In First Out
No
First In First Out
No
Largest Element First Out
No
Yes
Yes
Yes
stack
queue
priority_queue
Need to find element by key
No
Sorted by key
Yes
Yes
Insert/erase in middle
No
No
Insert/erase at front
No
Allow duplicates
Yes
Need to find the nth element
No
Yes
Yes
Store key separate to value
No
Store key separate to value
No
Yes
Yes
Yes
Need to merge collections
No
Need to merge collections
Yes
Yes
Size will vary widely
No
Yes
No
Yes
map
set
multi_map
multi_set
list
vector
deque
Yes

		Sequence containers					Associative containers				Unordered associative containers				Container adaptors		
Headers		<array>	<vector>	<deque>	<forward_list>	<list>	<set>		<map>		<unordered_set>		<unordered_map>		<stack>	<queue>	
		array	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
	(constructor)	(implicit)	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
	(destructor)	(implicit)	~vector	~deque	~forward_list	~list	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map	~unordered_multimap	~stack	~queue	~priority_queue
	operator=	(implicit)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=
	assign		assign	assign	assign	assign											
Iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin			
	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin			
	end	end	end	end	end	end	end	end	end	end	end	end	end	end			
	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend			
	rbegin	rbegin	rbegin	rbegin		rbegin	rbegin	rbegin	rbegin	rbegin							
	crbegin	crbegin	crbegin	crbegin		crbegin	crbegin	crbegin	crbegin	crbegin							
	rend	rend	rend	rend		rend	rend	rend	rend	rend							
	crend	crend	crend	crend		crend	crend	crend	crend	crend							
Element access	at	at	at	at					at				at				
	operator[]	operator[]	operator[]	operator[]					operator[]				operator[]				
	front	front	front	front	front	front										front	top
	back	back	back	back		back									top	back	
Capacity	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty
	size	size	size	size		size	size	size	size	size	size	size	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size			
	resize		resize	resize	resize	resize											
	capacity		capacity														
	reserve		reserve								reserve	reserve	reserve	reserve			
	shrink_to_fit		shrink_to_fit	shrink_to_fit													
Modifiers	clear		clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear			
	insert		insert	insert	insert_after	insert	insert	insert	insert	insert	insert	insert	insert	insert			
	emplace		emplace	emplace	emplace_after	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace			
	emplace_hint						emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint			
	erase		erase	erase	erase_after	erase	erase	erase	erase	erase	erase	erase	erase	erase			
	push_front			push_front	push_front	push_front											
	emplace_front			emplace_front	emplace_front	emplace_front											
	pop_front			pop_front	pop_front	pop_front										pop	
	push_back		push_back	push_back		push_back									push	push	push
	emplace_back		emplace_back	emplace_back		emplace_back									emplace	emplace	emplace
	pop_back		pop_back	pop_back		pop_back									pop		pop
	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap
List operations	merge					merge											
	splice					splice											
	remove					remove											
	remove_if					remove_if											
	reverse					reverse											
	unique					unique											
	sort					sort											
Lookup	count						count	count	count	count	count	count	count	count			
	find						find	find	find	find	find	find	find	find			
	lower_bound						lower_bound	lower_bound	lower_bound	lower_bound							
	upper_bound						upper_bound	upper_bound	upper_bound	upper_bound							
	equal_range						equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range			
Observers	key_comp						key_comp	key_comp	key_comp	key_comp							
	value_comp						value_comp	value_comp	value_comp	value_comp							
	hash_function										hash_function	hash_function	hash_function	hash_function			
	key_eq										key_eq	key_eq	key_eq	key_eq			
Allocator	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator			
		array	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
		Sequence containers					Associative containers				Unordered associative containers				Container adaptors		

C++ Containers Library cross-reference table from http://en.cppreference.com/w/cpp/container PDF version with red & orange lines by Robin Whittle December 2012.

- functions present since C++11

- functions present in C++03

Problem types

1. Simulation
2. Brute-force
3. Searching
4. Sorting
5. Graphs
6. Greedy
7. Divide and conquer
8. Dynamic programming
9. String processing
10. Geometry
11. Mathematics

Universiteit Leiden

Course organization (continued)

Throughout the course

- Practice

Throughout the course

- Practice
- Practice
- Practice
- Practice
- Practice
- Practice

Concrete course format

Universiteit Leiden

- Contest ("soft") — individual, 20%
  starts February 27, 2020
  with short report deadline March 9, 2020
- Presentation and report — individual, 35%
  throughout semester; list of topics divided in week 3
  deadline for topic report: May 4, 2020
- Three (live) programming contests — teams, $3 \times 15 = 45\%$
  In March and April 2020 (dates TBD)

Planning live contests

Universiteit Leiden

- Three 3-4 hour contests
- Problemset of 6–9 problems
- One computer
- Teams of 2–3 students
- Only use programming language manuals
- Team manual is allowed
- So, when? ...

Kattis

- Kattis
- Online practice platform
- `https://open.kattis.com`
- We will use it for weekly exercises
- Create an account
- Will not be used for course contests or grading

Lab session today

Universiteit Leiden

- From 10.15 to 11:00 in Snellius room 302/304
- Try out Java (if you want)
- (Re)-familiarize yourself with different algorithms and data structures via `https://github.com/gibsjose/cpp-cheat-sheet/blob/master/Data%20Structures%20and%20Algorithms.md`
- (Re)-familiarize yourself with the C++ standard library via `http://www.cplusplus.com/reference/stl`
- Get started with the practice problems of week 2: `https://open.kattis.com/contests/yhoxxs`
- Please register to "officially" take the course: `https://is.gd/cp2020`

Credits

Universiteit Leiden

This course, in particular these slides, are largely based on:

- Antti Laaksonen, *Guide to Competitive Programming*, Springer, 2017.
- Steven Halim and Felix Halim, *Competitive Programming 3*, Lulu.com, 2013.
- T-414-AFLV: A Competitive Programming Course, `https://github.com/SuprDewd/T-414-AFLV`

Where applicable, full credit for text, images, examples, etc. goes to the authors of these books.