Contest programming

Introduction

Maksym Planeta

13.04.2018

Table of Contents

What is programming contest?

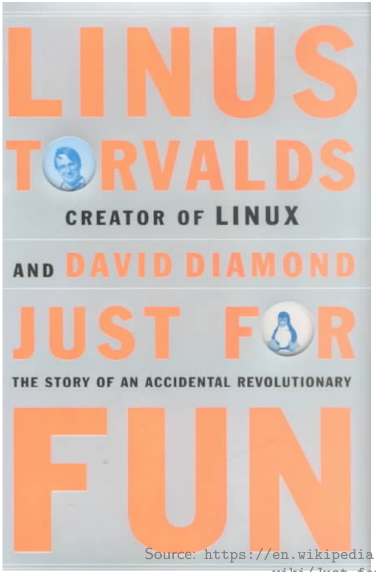
- Type of competition
- Participants solve algorithmic problems
- Limited time frame
- Specific number of problems

What type of contests exist

- ACM ICPC and subordinates – the classical one
- Project Euler – online, mathematical
- Kaggle – online, data science
- Company organized (Facebook Hacker Cup, Google Codejam, Google Hash Code)
- Pi calculation, largest prime number calculation
- Student cluster competition as Supercomputing conference

Why participate?

- Improve programming skills
- Exercise work in a team
- Learn algorithms and data structures
- For glory



Source: https://en.wikipedia.org/wiki/Just_for_Fun

This course

"ACM ICPC" type:

- Algorithmic and mathematical problems
- Individual or collective
- Built around universities

Organization:

- 12 weeks
- 10 exercises:
    - Introduction and entrance test
    - 8 lectures + exercises
    - Full practice session

Structrue of the course

A normal session comprises:

- Short lecture
  - Explain one of the contest topic
  - Look into a task
- Practice session
  - Today individual
  - Later in teams

Particular topics

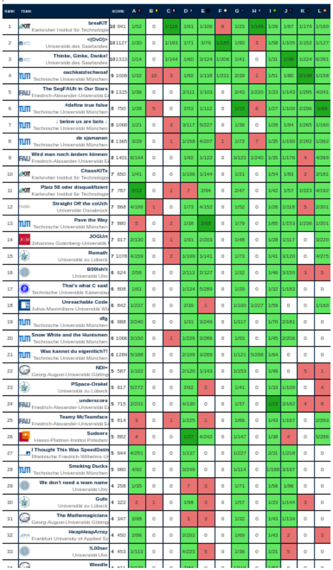
Not exclusive list of topics:

1. Computational complexity
2. Greedy algorithms. Dynamic programming
3. Sorting and Searching
4. Combinatorial problems
5. Algorithms on graphs
6. Shortest path/Network flow
7. Algorithms on strings
8. Computational geometry
9. Long arithmetic. Floating-point arithmetic.

Basic ACM ICPC rules

- One computer
- One team (2 to 3 People)
- 5 hours
- Up to 10 Problems
- Limited usage of supplementary material

Evaluation



RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K	L
1	breaKIT Karlsruher Institut für Technologie	10 941	1/52	0	1/118	1/61	1/106	6	1/22	1/149	1/26	1/67	1/179	1/160
2	<{DvD}> Universität des Saarlandes	10 1127	1/30	0	1/193	1/71	3/76	1/185	1/90	3	1/58	1/105	1/152	1/127
3	Thinke, Ginke, Danke! Universität des Saarlandes	10 1313	1/14	0	1/144	1/60	2/124	1/206	1/41	0	1/21	1/48	1/224	6/281
4	oachkatzlschwoaf Technische Universität München	9 1009	1/32	10	3	1/92	1/128	1/231	2/39	1	1/51	1/80	2/148	1/158
5	The SegFAUlt in Our Stars Friedrich-Alexander-Universität E	9 1315	1/36	0	0	2/111	1/103	0	2/42	1/220	1/23	1/142	1/295	4/241
6	#define true false Technische Universität München	8 750	1/38	5	0	2/52	1/112	0	1/15	6	1/27	1/100	2/296	1/69
7	⁝ below us are bots ⁝ Technische Universität München	8 1068	1/21	0	3	3/117	5/227	0	1/36	0	1/28	1/94	1/265	1/160
8	de sjamanen Technische Universität München	8 1365	3/29	0	1	1/158	4/207	1	1/72	7	1/25	1/190	2/292	1/262
9	Wird man noch ändern können Friedrich-Alexander-Universität E	8 1401	6/144	0	0	1/92	1/122	0	3/123	1/240	1/25	1/176	4	4/269
10	ChaosKITs Karlsruher Institut für Technologie	7 650	1/41	0	0	1/106	1/144	0	1/21	0	1/54	1/83	2	2/181
11	Platz 56 oder disqualifiziert Karlsruher Institut für Technologie	7 787	2/12	0	1	7	2/94	0	2/47	0	1/42	1/57	1/223	4/192
12	Straight Off the coUch Universität Osnabrück	7 868	4/106	1	0	1/72	4/152	0	1/52	0	1/26	1/118	5	2/201
13	Pave the Way Technische Universität München	7 880	5	0	2	1/36	2/66	0	1/79	0	1/65	1/152	1/226	1/201
14	JOGUrt Johannes Gutenberg-Universität	7 917	2/130	0	1	1/91	2/203	0	1/48	0	1/28	1/117	0	3/220
15	Romath Universität zu Lübeck	7 1078	4/159	0	2	1/109	1/141	0	1/73	0	1/41	3/120	0	4/275
16	B00lsh!t Universität Ulm	6 624	2/56	0	0	2/113	2/127	0	1/32	0	1/46	3/150	3	5
17	That's what C said Technische Universität Kaiserslau	6 808	1/61	0	0	1/124	5/289	0	1/29	0	1/22	1/182	0	0
18	Unreachable Code Julius-Maximilians-Universität Wü	6 842	1/227	0	0	2/39	1	0	1/100	1/227	1/59	0	0	1/160
19	dfg Technische Universität München	6 968	3/240	0	0	1/31	3/249	0	1/117	0	1/70	2/181	0	0
20	Snow White and the Huntsmen Technische Universität München	6 1066	3/150	0	1	1/226	3/286	0	1/53	0	1/45	2/206	0	0
21	Was kannst du eigentlich?! Technische Universität München	6 1289	5/188	0	0	2/109	3/289	0	1/121	5/298	1/64	0	0	0
22	NDI= Georg-August-Universität Götting	5 587	1/102	0	0	2/120	1/143	0	1/153	0	1/49	0	5	1
23	PSpace-Orakel Universität zu Lübeck	5 617	5/272	0	0	2/62	2	0	1/41	0	1/23	1/109	0	4
24	_underscore Friedrich-Alexander-Universität E	5 715	2/221	0	0	4/130	0	0	1/37	0	1/15	3/182	4	8
25	Teamy McTeamface Friedrich-Alexander-Universität E	5 814	3	0	1	1/225	1	0	1/66	0	1/43	1/167	0	2/292
26	Sudoers Hasso-Plattner-Institut Potsdam	5 882	4	0	0	1/27	4/242	0	1/147	0	1/38	4	0	5/288
27	I Thought This Was SpeedDatin Rheinische Friedrich-Wilhelms-Ur	5 944	4/251	0	0	1/127	0	0	1/227	0	2/31	1/218	0	0
28	Smoking Ducks Technische Universität München	5 980	4/92	0	0	3/249	0	0	1/114	0	1/188	3/197	0	0
29	We don't need a team name Universität Ulm	4 258	1/35	0	0	7	3	0	1/71	0	1/56	1/96	0	0
30	Gufo Universität zu Lübeck	4 322	2	1	0	1/98	3	0	1/57	0	1/23	1/144	3	0
31	The Mathemagicians Georg-August-Universität Götting	4 347	3/98	0	0	3	3	0	1/32	0	1/43	1/134	0	0
32	HeapHeapArray Frankfurt University of Applied Sc	4 450	2/96	0	0	2/202	0	0	1/69	0	1/43	2	0	3
33	!L00ser Universität Ulm	4 453	1/113	0	0	4/223	5	0	1/36	0	1/21	5	0	0
34	Weedle	4 511	[illegible]	0	0	[illegible]	0	0	[illegible]	0	[illegible]	0	0	0

Position in the rating determined by:

- Number of solved Problems
- Total submission time
- Penalties for incorrect submissions
- Additional tiebreaks

Source: icpc.tum.de/contests/history/2016/gcpc

What does a problem look like?

1. Problem description
2. Resource limits
3. Input description
4. Output description
5. Sample input
6. Sample output

Example: $3n + 1$

Problem description[2]

Consider the following algorithm:

```
1.  input $n$
2.  print $n$
3.  if $n = 1$ then STOP
4.      if $n$ is odd then $n \longleftarrow 3n + 1$
5.      else $n \longleftarrow n/2$
6.  GOTO 2
```

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed before and including the 1 is printed. For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between and including both i and j.

`https://uva.onlinejudge.org/external/1/100.pdf`

Example: $3n + 1$

Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 10,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.

You can assume that no operation overflows a 32-bit integer.

Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

`https://uva.onlinejudge.org/external/1/100.pdf`

Example: $3n + 1$

Input example

```
1 10
100 200
201 210
900 1000
```

Output example

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

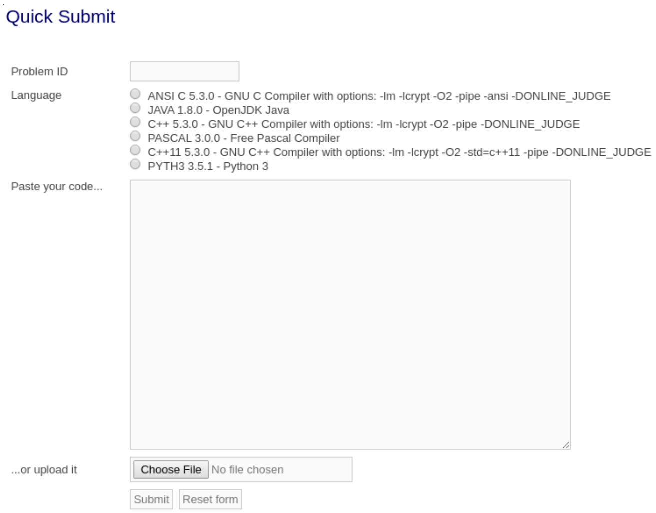
https://uva.onlinejudge.org/external/1/100.pdf

Register

`https://uva.onlinejudge.org/`

Send me your ids.

Online judge



Quick Submit

Problem ID	
Language	ANSI C 5.3.0 - GNU C Compiler with options: -lm -lcrypt -O2 -pipe -ansi -DONLINE_JUDGE JAVA 1.8.0 - OpenJDK Java C++ 5.3.0 - GNU C++ Compiler with options: -lm -lcrypt -O2 -pipe -DONLINE_JUDGE PASCAL 3.0.0 - Free Pascal Compiler C++11 5.3.0 - GNU C++ Compiler with options: -lm -lcrypt -O2 -std=c++11 -pipe -DONLINE_JUDGE PYTH3 3.5.1 - Python 3
Paste your code...	
...or upload it	Choose File No file chosen

`https://uva.onlinejudge.org/`

Bootstrap

Makefile

```
CXXFLAGS=-Wall -Wextra -O0 -ggdb3
p100: p100.cpp
```

p100.cpp

```
#include <stdio.h>

int main()
{
  int i, j;
  while (scanf("%d %d", &i, &j) > 0) {
    /* your code */
  }
}
```

How to approach a problem[3]

1. Analyse the problem statement
2. Sketch out a solution
3. Build a formal solution model
4. Implement the algorithm
5. Test and debug
6. Submit

Analyse the problem statement

1. Problem statement is often a story
2. Don't miss important detail or possible trick
   - Pay **HUGE** attention to input constraints
   - Watch out the resource limitations
   - A general *NP*-problem may become a *P*-problem.
   - Don't rush.
3. Look into examples
4. Draw a picture (geometry, graph theory)
5. If stuck, try to read other problems

Compose a solution

Often a problem can be decomposed into subproblems.

1. Identify subproblems
2. Do you know analogous problems?
3. Can you reduce a problem to a known one?
4. Did you utilize all input data?
5. Try to alter input data
6. Check corner cases (0, 1, 2, *maximum*, $maximum - 1$)

Build a formal solution model

When there is a high level understanding of the problem.

1. Come up with an algorithm for a machine with virtually unlimited resources.
2. Compose solutions to subproblems together
3. Identify potential efficiency problems
    - Computation and memory complexity

Implement the algorithm

1. You need to have a template with input/output prepared
2. Top-down implementation
   - First you write high level functions
   - Then low-level ones
   - Good understanding of the problem
3. Bottom-up implementation
   - Start with auxiliary functions, solutions to subproblems
   - Hope that understanding comes on the way
   - Solution seems to be less clear
4. Combine top-down and bottom-up

Test and debug

1. Compile
2. Check with given tests
3. Add your own simple tests
4. May be useful to check intermediate stages of the program
5. If feasible, consider creating a “big” test
6. Don’t hesitate to use `-Wall -Wextra`, ASAN, valgrind
   - My favorite bug: forget to return a value from a function

Submit

1. Read problem statement again
2. Check compliance to required output format
   - Remove debugging output
   - You may rely on macro `ONLINE_JUDGE`

Computational complexity[1]

- Problems impose limits:
  1. Run time
  2. Memory size
- How do you know that your algorithm fits the requirements?
- Analyze the algorithm

Algorithm analysis

1. Compute number of “primitive” operations to run the algorithm:
   add, subtract, multiply, divide, remainder, floor, ceiling, load, store, copy etc.
2. Compute number of bytes your algorithm needs
3. Both number of instructions and amount of memory depends on input
4. Find the worst case

Approximation

- Exact numbers are hard
- Find worst case complexity
- Average case complexity may be useful for subproblems
- Ensure that worst case fits into the limitations (with some margin)

Asymptotic notation

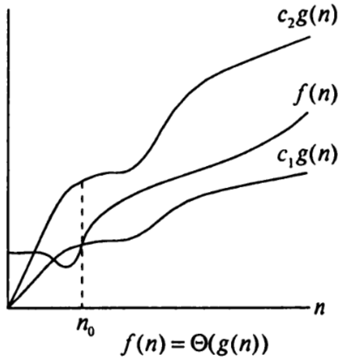


Figure: Graphical example of Θ-notation[1, p. 45]

$$\Theta(g(n)) = \{f(n) : \quad \forall n \geq n_0, \\ \exists c_1 > 0, c_2 > 0, n_0 > 0 \\ \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Asymptotic notation

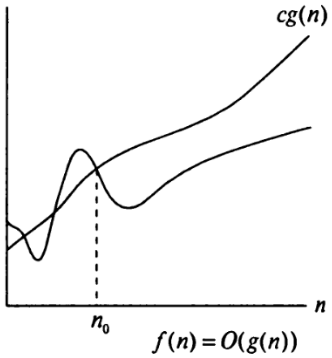


Figure: Graphical example of O-notation[1, p. 45]

$$O(g(n)) = \{f(n) : \forall n \geq n_0, \exists c > 0, n_0 > 0 \mid 0 \leq f(n) \leq cg(n)\}$$

Asymptotic notation

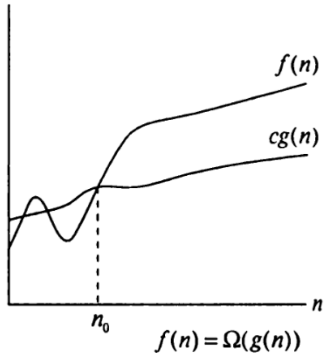


Figure: Graphical example of Ω-notation[1, p. 45]

$$O(g(n)) = \{f(n) : \forall n \geq n_0, \exists c > 0, n_0 > 0 \mid 0 \leq cg(n) \leq f(n)\}$$

Properties of O-notation

- $O(n)$ defines an upper bound. Exactly what we need
- $O(g(n)) < O(f(n)) \implies O(g(n)) + O(f(n)) = O(f(n))$
  Example: $O(n^3) + O(n^2) + O(1) = O(n^3)$
- $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \land f(n) = \Omega(g(n))$
- $f(n) = O(g(n)), \quad g(n) = O(h(n)) \implies f(n) = O(h(n))$

stl containers

C++ provides a set of containers, like

1. `vector`
2. `deque`
3. `list`
4. `priority_queue`
5. `array`
6. `map`
7. `unordered_map`
8. ...

Sources on complexity:
EECS 311: STL Containers
`http://www.cplusplus.com/reference`
`http://en.cppreference.com`

Vector

```
#include <vector>
```

Operations:

1. Random access: $O(1)$
2. Insert or removal at the end: amortized $O(1)$
3. Insert or removal: linear in distance to the end $O(n)$

See also: `array`, `vector<bool>`, `bitset`

Deque

```
#include <deque>
```

Similar to vector. Reallocations are cheaper. Access is more expensive, memory requirements are higher.

Operations:

1. Random access: $O(1)$
2. Insert or removal at the end: amortized $O(1)$
3. Insert or removal: linear $O(n)$

List

```
#include <list>
```

Double-linked list.

Operations:

1. Access to head or tail: $O(1)$
2. Access to element by index: $O(n)$
3. Insert or removal at the head or tail: $O(1)$

Stack, queue

```
#include <list>
#include <queue>
```

Adaptors usually to a list. Provide either FIFO or LIFO semantics.

Operations:

1. Access to head or tail: $O(1)$
2. Insert or removal at the head or tail: $O(1)$

Priority queue

```
#include <priority_queue>
```

Implementation of a heap. A data structure, where the largest item is always on the top. Backed by deque.

Operations:

1. Access largest element: $O(1)$
2. Insert or removal: amortized $O(\log n)$

Set, map, multiset, multimap

```
#include <set>
#include <map>
```

Red-black trees to implement either set of dictionary. Keeps elements ordered.

Operations:

1. Find element: $O(\log n)$
2. Insert or removal: $O(\log n)$

Unordered_set, unordered_map

```
#include <unordered_set>
#include <unordered_map>
```

Hash tables to implement either set of dictionary. Keeps elements unordered.

Operations (may trigger rehash):

1. Find element: average $O(1)$ worst-case $O(n)$
2. Insert or removal: average $O(1)$ worst-case $O(n)$

Dark side of contest

- You learn to write code fast.
  Fast also often means short.
- You concentrate on the input data.
  Preallocating everything for biggest input size is normal
- You write code you probably never going to read against
  You don't care about non-obscurity
- You do not check function error codes
  It is safe to assume that functions work correctly

Don't forget these patterns may be *very bad* for normal life programming.

Home work

Read chapters 2.2 (Analyzing algorithms) and 3 (Growth of functions) from Cormen.
Solve the rest of the problems from entrance test.

Entrance test

Solve following problems (order by complexity):

1. 11498 - Division of Nlogonia
2. 10114 - Loansome Car Buyer
3. 706 - LC-Display
4. 10038 - Jolly Jumpers
5. 13161 - Candle Box
6. 1717 - Ship Traffic
7. 1737 - Branch Assignment

Send me your code and account id at `https://uva.onlinejudge.org/` to mplaneta@os.inf.tu-dresden.de

Literature

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

Steven S Skiena and Miguel A Revilla.
Programming challenges: The programming contest training manual.
Springer Science & Business Media, 2006.

S. Zhykovskyi.
The analysis, research and solution of problems during the students olympiad in informatics.
Visnyk of ZDU, 2010.