Towards an IOI Syllabus (draft)

Sergejs Kozlovičs[1]
Mārtiņš Opmanis[2]

Abstract

This paper contains comments to be considered when developing a Syllabus for the International Olympiad in Informatics (IOI). Special attention is paid to differences in programming languages and topics which should be included in Syllabus.

1. Introduction

The International Olympiad in Informatics (IOI) is an annual competition targeted at secondary school students talented in computing science. Like it or not, but the IOI regulations do not describe clearly the scope of the competition. Paper [Verhoeff et. al.] contained arguments for having such a syllabus and provided a proposed syllabus. The conclusion of that paper and special call at IOI'2006 encouraged for debates, so this paper is a reply to [Verhoeff et al.] which we consider a primary reference here.

The structure of this paper is organized as follows. In the next section we point to some desired aspects of a Syllabus. Then, in Section 3, we list some differences in programming languages that may be crucial for the contents of a Syllabus. After that, we list some topics to be added to a proposed Syllabus in [Verhoeff et al.]. Finally, Section 5 concludes the paper.

2. Desired properties of a syllabus

Syllabus should be changeable

On the one hand, a syllabus should give information to teachers about what students should be taught. Students also should have a confidence of what they study is worth and that the topics they have studied can help solve competition tasks at the IOI. On the other hand, a syllabus should not be a dogmatic standard. Sometimes it may be desirable to present an innovative task that is not based on syllabus. Also, there are teachers that may prefer not to use syllabus at all.

This discrepancy may be solved, for example, in the following way. Having an official IOI syllabus, it can be proclaimed that at least x (and, maybe, at most y) tasks will be based on this syllabus. The remaining tasks may have deviations from the syllabus or even rely on topics not mentioned there. A good idea would be to update the syllabus regularly, for example, every year (or every k years) depending on the previous years' tasks and/or new trends.

It must be pointed out that "updating" means not only "adding", but also "removing" some topics from syllabus not used for a long time in competitions. For topic removing (as well as for adding) there must be stated clear procedure how it will be done (established list of competitions absence in which leads to placing particular topic under consideration of removal, discussion with final voting at IOI, etc.). Some paralels with Olympic games may be provided where particular kind of sports at first is nominated as candidate and only afterwards become a full member of games. At the same time number of olympic sports is constant and therefore adding of one leads to removing of other. Keeping size of syllabus (amount of topics) in reasonable limits also must be one of permanent goals.

It should be taken into account that some topics of a syllabus depends on programming language used

Some standard algorithms and data structures mentioned in syllabus may already be implemented in libraries that come with programming languages. So, contestants do not need to know how to implement particular algorithm and/or data structure. They just have to know how to use existing

1 sergejs.kozlovics@gmail.com
2 martins.opmanis@mii.lu.lv

implementation. However, there are differences between programming languages, so, for example, contestants that use C++ may have an advantage over contestants that use Pascal. This problem is discussed later in Section 3.

Syllabus may contain sketches about topics for later studying at high school

The idea is to divide each syllabus category into two parts. The first one would be used to prepare undergraduate school students for IOI, and the second one would list the topics for later studying. That would help students to see what else each topic consists of, so they would know a bit more about what they can learn after graduating from the school. Students can also consider the second part of each category as possible topics for self-studying. University lecturers that teach one of the syllabus categories may want to look at its first part in this syllabus to see what knowledge students might have taught at school.

Each topic should have its own identifier

Currently, [Verhoeff et al.] marks only categories (e.g., PF1 is a part of "Programming Fundamentals" and AL10 is a part of "Algorithms and Complexity"). However, if topics also had such markers or identifiers, it could be possible to add the same identifiers to texbooks about algorithms. This would help students and teachers when searching for an appropriate book explaining the desired topic. Some examples of topic identifiers follow:

- SORTING-$O(n^2)$ (for BubbleSort, InsertionSort, etc.)
- SORTING-$O(n \log n)$ (for QuickSort, HeapSort, etc.)
- CONVEX-HULL (for Graham's scan and Jarvis's march)
- DFS (for Depth-first search strategy)
- BFS (for Breadth-first search strategy)
- SST (for Minimum/Shortest spanning tree).

3. Differences in programming languages that can affect the contents of a syllabus

Here we are touching the type of tasks given at the IOI. On the one hand, if the tasks have to be of an algorithmic nature (see [Kenderov and Maneva]) then differences between programming languages should not matter. This also implies that it is sufficient for a participant to know only one programming language, so he/she can concentrate only on algorithm design.

On the other hand, one practical aspect, namely, the skill of choosing a programming language that best meets the needs, is avoided. This practical aspect, however, forces participants to study several programming languages that may be not preferable due to the IOI focus on problems of algorithmic nature. Also, different teachers may prefer to teach different programming languages.

We cite [Verhoeff et al.]:

> "... the overwhelming details of modern computing platforms (programming languages, programming tools, operating systems, system and processor architecture) should not become and obstacle to success."

One more item to be considered is allowing new programming languages at the IOI. For example, Java may be introduced. There are also some other interesting types of programming languages, e.g., declarative (PROLOG, etc.) or functional (ML).

We continue citing [Verhoeff et al.]:

> "Currently, imperative programming languages are used in the IOI, but the authors wish to point out that functional languages would also be suitable for this purpose."

Currently, allowed languages at IOI are Pascal and C++. We do not exclude the possibility that Java can be added to them. In any case, Java and C++ are the only languages[3] allowed at ACM ICPC (International Collegiate Programming Contest), and many IOI contestants participate there

3 Pascal is no more allowed since 2006.

after joining the high school.

So, in this section we consider three programming languages mentioned above, i.e., Pascal, C++ and Java with the following implementations:

- FreePascal for Pascal (Turbo Pascal is almost not used at olympiads today; FreePascal is commonly used at the IOI and regional competitions),
- GNU C++ for C++ (it was used both at the IOI and the ACM ICPC final in 2006), and
- Sun Java Development Kit for Java (Java 1.5 was used at the ACM ICPC final in 2006).

Taking into a consideration these programming languages and implementations, we list some of the differences and nuances below. In some places we also mention other implementations.

High precision arithmetic

Currently, both Pascal and C++ implementations doesn't include high precision arithmetic library, but Java does (see class `BigInteger` and `BigDecimal` for arbitrary-precision integer/decimal numbers from `java.math` package). So, contestants that program Java needn't implement high precision arithmetic for the tasks that require it. That gives an advantage to Java programmers. Currently, this is not a problem for the IOI because only C++ and Pascal solutions are accepted there. However, it might become a problem if Java will be introduced.

Sometimes, constraints for input data may be satisfied by using 64-bit integers. Popular C++ compilers including GNU C++, Borland C++ and Microsoft Visual C++, allow to use `__int64` type (there is also `unsigned __int64` type). Some additional steps must be performed in order to scan or print values of such type. For example, when using C/C++ `printf` function, we should specify `%I64d` format parameter in format-string for a variable of `__int64` type (`%I64u` for `unsigned __int64`). Popular Pascal compiler, FreePascal, also has similar types for 64-bit signed and unsigned integers: `Int64` and `QWord` respectively. No additional statements are needed to read or write variables of such types; standard `Read/Readln` and `Write/Writeln` procedures can be used as usual. We can also mention that legacy Borland Pascal compiler neither does have 64-bit integer type, nor even 32-bit unsigned integer (the latest is called LongWord in FreePascal). Java has a native `long` type for 64-bit integers, so there should not be problems with it.

Random generator

A random generator can be used in some reactive tasks or just when participants want to generate their own input files to test their solutions. The following table shows some differences between programming languages that refer to random-number generation.

	Pascal	C/C++	Java
Used library	Unit `System` (is available in every program)	`stdlib.h` for generating random number and `time.h` for initializing random generator	class `java.util.Random`
How to initialize the random generator	Call `Randomize` procedure.	`time_t t;` `srand((unsigned)time(&t));`	`Random r = new Random();`

How to set the random seed to a specified value (e.g., 5)	`RandSeed := 5;`	`srand(5);`	`Random r = new Random(5);`
			or (if r has already been initialized)
			`r.setSeed(5);`

How to generate a random integer between 0 and n-1.	`x := Random(n);`	`x = rand()%n;`	`x = r.nextInt(n);`
How to generate a floating-point number between 0	`y := Random;`	`y = rand()/(float)RAND_MAX;`	`y = r.nextFloat();`

	Pascal	C/C++	Java
and 1.			

As we can see, C/C++ programmers are required some language-specific tricks (e.g., taking a remainder modulo *n* when calculating a random integer, or calling function time to initialize a timer).

It should be also mentioned that Pascal function `Randomize` relies on system timer that changes every 1/18 of second, but C/C++ function `time` changes its value every 1 second. So, in some cases, some modifications may be needed to the testing program (i.e., testing program should wait for a second between every two tests) in order participant's solution could correctly re-initialize random-number generator for the next test.

Sorting and binary search

Both C++ and Java have routines that implement sorting and binary search. In C++ `qsort` and `bsearch` functions from `stdlib.h` can be used for arrays, and `sort` and `binary_search` functions from `algorithm` header can be used for STL containers. In Java class `Arrays` and its methods `sort` and `binarySearch` can be used. Data structures like `set` in C++ STL or `TreeSet` in Java can also be used for sorting elements and searching for them. Pascal doesn't have such standard functions. We can mention only `TFPList` data structure that comes with FreePascal and has a `sort` method. However, an interesting story about legacy Borland Pascal is provided below. It also shows another aspect to be considered.

> Not so long ago participants could use Turbo Pascal 7.0 for DOS (or more advanced Borland Pascal 7.0 with Objects). Full installation came with examples containing sources that demonstrated the use of Borland Graphics Interface (BGI), many examples demonstrating OOP paradigm, and some other sources. Most of them seem not be useful in programming competitions like IOI, however, there was a (!) QuickSort implementation (installed in EXAMPLES subdirectory, file qsort.pas; file dirdemo.pas also contained its own implementation). If competition environment used by contestants contained the full installation of Borland Pascal, contestants could use that QuickSort implementation simply by copying/pasting it into their programs. That implementation is a standard one. One could write almost the same or actually the same implementation personally without knowing about EXAMPLES subdirectory. So, having only a contestant's program, it was impossible to determine whether QuickSort was written from scratch, or copied from examples. Some competition environments had full Borland Pascal installation, some had only IDE and RTL. In the last case, when needed to sort something, contestants were forced to write their own sorting routine, often learnt by heart.

Data structures

C++ STL and Java come with the following widely used data structures:

- `vector` (C++) and `ArrayList`/`Vector` (Java) implementing an array that can change its size dynamically;
- `set` (C++) and `TreeSet` (Java) implementing a data structure like binary tree (there is also `HashSet` in Java);
- `map` (C++) and `TreeMap`/`HashMap` (Java) are very efficient data structures allowing to map elements of one type to elements of another type.

None of the mentioned structures come with FreePascal.

The following data structures have some analogs in FreePascal:

- `list` (C++), `LinkedList` (Java) and `TFPList` (Pascal);

- `bitset` (C++), `BitSet/BigInteger` (Java) and `set of SomeType` (Pascal); note, that sets is Pascal can contain elements of small types only, e.g., `set of Byte`, `set of Boolean`, `set of Char` and the set can contain no more than 256 elements;
- `string` (C++), `String/StringBuffer` (Java) and `String` (Pascal); Pascal string takes 256 bytes of memory by default and can store up to 255 character long strings. However, with an `$H` compiler directive we can force FreePascal to use longer strings. There is also `PChar` type in Pascal that is analogue for C null-terminated strings but in order to use features of such strings, a programmer must be familiar with dynamic memory allocation.

Input and output

Since Java 5 there is a new class `Scanner` that can be used as a wrapper to an input stream. The C-like function `printf` for output streams has also appeared in Java 5. C++ has also object-oriented input/output routines located in `iostream` and `fstream` headers. Pascal doesn't have format-strings. Only limited formatting is possible when using `Write/Writeln` procedure.

We can also mention that sometimes standard routines for reading/writing data from/to text files are very slow. Sometimes it is possible to use routines for block files (e.g., `BlockRead` in Pascal). For example, consider a file representing a $n \times n$ adjacency matrix for some graph consisting only of 0 and 1. Instead of reading matrix elements one-by-one, we can read the whole row, or even the whole matrix at once. This can rapidly decrease the time a program spends on reading the input data. Of course, we should correctly calculate the size of the buffer where to read the matrix taking into a consideration spaces and line delimiters that are specific to text files.

How to eliminate differences between programming languages

Several solutions to decrease the differences between programming languages are:

- Prohibit to use certain libraries/include-files (e.g., using STL in C++). Not a good idea because it doesn't stimulate students to learn new programming paradigms like generic programming.
- Provide lacking functionality, e.g. provide STL to PASCAL or see the examples that come with programming language distribution, whether some of them can be used instead of libraries. In this case the provided library (or suggested examples) must be good enough and widely used in order participants may learn how to use it (them).
  Pascal do not provide support for generic programming using templates like in C++. So, it very likely, that from contestant's point of view, it is better to take the existing code and to use it with appropriate modifications (in the example below, participants has to replace Integer with the appropriate type).

```
Type
  BaseType = Integer;
Procedure QuickSort(a: array [1..N] of BaseType; L, R: Integer);
```

- Provide a common library (`ioi.h`/`ioi.cpp`, `ioi.pas`, `ioi.java`). This is a composition of previous two items.
- Do not give tasks that can give an advantage depending on programming language. This restriction may lead to refuse of many good tasks.
- Provide a code or a hint to participants that use programming languages lacking some features.

The last solution comes from USACO (USA Computing Olympiad) as is explained in the following story.

> Once upon a time in USACO, solutions to a given problem had to output floating point number in the appropriate format. Contestants that programmed in C could use operator `printf`, specifying the appropriate format-string. However, Pascal operator `Write/Writeln` could not provide the same functionality. A solution was simple: a problem description came with Pascal code that participants programming in Pascal could use in their programs. That code converted a floating point number to a string, and then, processed

that string to obtain the desired format.

4. Recommended categories and topics to be included in Syllabus

This section contains several topics we propose to be included in the Syllabus. When referenced, category identifiers (such as PF3 or AL3) and section numbers are the same as in [Verhoeff et al.]. Examples of referenced tasks are taken from IOI, BOI (Baltic Olympiad in Informatics) and ACM ICPC.

Data structures

- A mention of **Heap data structure** in PF3 (priority queue) or/and AL3 (heap sort). This data structure is important itself.
- **Processing arrays** in PF3 (arrays), AL2 (induction/scanning, see below) and/or AL3. That includes filling (correct usage of Pascal `FillChar` and C `memset` functions), cyclical shifting (also using Euclide's algorithm), cumulative sums (for 2D arrays see IOI2001 task "Mobiles"), a subarray with maximal sum of elements (for 1D arrays see Sect. 8.4 [Bentley] about scanning algorithm), rotations of 2D arrays. We can also mention vectors, i.e., extensible arrays (it may not be required to know how to implement it, but it is worth to know how to use, for example, STL implementation).
- **Interval tree** data structure in PF3 or AL3.
- Marked as "Not needed" **Hash tables** in AL3 seemed to be useful at IOI2001 for the task "Double crypt".

Algorithmic strategies

The following topics should be added to AL2.

- **Induction/scanning** (that is connected to DS3, "Mathematical induction"). That usually is a calculation from left to right preserving some invariant (e.g., min/max selection). It can be used also to find an array interval with maximal sum at each step considering two invariants: maximal sum so far, and maximal sum ending at the current position (see [Bentley]).
- **Scaling**, i.e., approximation of the result using binary search-like strategy (See Ex. 26-5 [Cormen et. al.]). This strategy can be used in many tasks where we know the bounds the result will fit, say L and R. First, we take a value C=(L+R)/2 and search the solution for this value. If the solution has been found, set the interval to L..C and continue. Otherwise, set the interval to C..R and continue.
- **Heuristics**.

Strings, grammars and automata

We recommend to create separate category growing up from AL2 "Pattern matching and string/text algorithms" and AL7. The topics included would be:

- **Anagrams** (i.e., sorting the letters in each word)
- Knuth-Morris-Pratt **string search algorithm** (also the notion of **prefix-function**)
- **Finite automata** (is the word in the given language? how to construct an automaton accepting words contatining "abracadabra")
- **Pattern matching**
- **Context-free grammars** and **arithmetic expression** definition (also: prefix/infix/postfix notation and conversion between these notations)
- Properties of strings that arise from **cyclically shifting the given string** (see a backup task "Binary Codes" from IOI2001)
- **Suffix trees**
- **Archiving** (also, RLE-compression and Huffman codes). There was a task about RLE compression at BOI2006.

Arithmetics and number theory

The following topics may be included in AL3:

- Along with Euclid's algorithm it could be useful to know **Extended Euclid's algorithm** (e.g., for solving linear Diophantine equations that can arise in many tasks).
- **Chinese remainder theorem**
- **Factorization**
- **Floating-point arithmetics** (also, representation and comparison using an epsilon). Geometry tasks may require floating-point arithmetic (i.e., when we try to determine intersections).
- **Arbitrary-precision arithmetics** (classical algorithms). At least addition/subtraction and multiplication should be known.
- **Radix conversion**, **Roman numbers**. (A task about Roman numbers was at ACM ICPC quaterfinal in Minsk, 2005.)

Combinatorics and counting

Some of the following may be included in DS4.

- **Gray codes**
- Natural number **decomposition into terms**
- **Catalan's numbers** (the number of correct placements of brackets)

Graph algorithms

The following topics (some of them are marked as "Excluded" in AL3) were very common at programming olympiads:

- **Strongly-connected components** (for oriented graphs). There were several tasks at ACM ICPC required the knowledge of at least one algorithm for finding stronly-connected components.
- **Biconnected components, bridges and articulation points** (for non-oriented graphs). See a demo task "Break" from IOI2001.
- **Flows** (Ford-Fulkerson or Push-relabel algorithms). See task "Electronic plate" at BOI2000.
- **Maximal matching in bipartite graph.** We can adapt Ford-Fulkerson algorithm, but more efficient Hopcroft-Karp's algorithm was needed at BOI2001 for the task "Knights".
- **Minimum-cost flow** (also, transportation problem) + how to determine **negative cycle** to reduce the cost (was needed at ACM semifinal in St.Petersbourg, 2002).

Geometry and Computational geometry

The following topics can be added to AL10.

- **Circle** (marked ar "Not needed" in 6.1.1). A good task, for example, is: given *n* circles, we need to find a minimal perimeter of a hull such that all the circles ar inside of the hull.
- **Angles** (also, sorting by polar angle) and **trigonometric functions** (marked as "Excluded" in 6.1.1). Some simple properties of trigonometric functions are required in tasks concerning physics (e.g., refraction of light).
- **Squared papers** and **grids** (also: Pick's theorem, see [Pick's theorem]). There were many tasks concerning this topic. See, for example, task "XOR" at IOI2002.
- **Orthogonal rectangles** (common area, common perimeter).
- The **intersection of two convex polygons**.
- **Sweeping-line method** (see [Cormen et. al.]).

Miscellaneous

- **Cryptographic algorithms** are marked as "Excluded" in AL9. However, there were task concerning cryptography at BOI2001 (task "Crack the Code") and IOI2001 (task "Double

Crypt").

- Excluded "**Normal forms**" in DS2 may be useful when constructing a Boolean expression that corresponds to the given truth table.

5. Conclusion

The aim of this paper is to point to some problems concerning IOI Syllabus as well as provide some extra topics to be included in that Syllabus. We hope it will help students and educators in preparation for the IOI. Contest designers can also use the approaching Syllabus. However, we consider that the Syllabus should not become a dogmatic standard.
We think also about creation of a textbook that may reflect the topics mentioned in the Syllabus.

References

[Cormen et al.] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 2nd edition, McGraw-Hill, 2001.

[Kenderov and Maneva] P. S. Kenderov and M. N. Maneva (eds.), *Proceedings of the International Olympiad in Informatics*, Pravetz, Bulgaria, May 16-19, 1989. Sofia: Union of the Mathematicians in Bulgaria, 1989.

[Pick's theorem] *Pick's Theorem*

`http://mathworld.wolfram.com/PicksTheorem.html` (accessed November 2006).

[Verhoeff et al.] T. Verhoeff, G. Horvath, K. Diks and G. Cormack, *A proposal for an IOI Syllabus*, Teaching Mathematics and Computer Science, 4/1 (2006), pp. 193-216. University of Debrecen.