Programming Seminar

Zdeněk Sawa

Department of Computer Science, FEI
Technical University of Ostrava
17. listopadu 2172/15, Ostrava-Poruba 708 00
Czech republic

September 17, 2025

Lecturer

Name: Zdeněk Sawa

E-mail: zdenek.sawa@vsb.cz

Room: EA413

WWW: http://www.cs.vsb.cz/sawa/spr-en

Seminar:

- Wednesday 9:00–10:30, room EB130

Requirements

- During the semester, problems will be published on the following page. You will obtain points for (accepted) solutions of these problems:

  http://www.cs.vsb.cz/sawa/spr-en/problems.html

- For each problem, there will be specified the number of points you can obtain for its successful solution, and a date until which you can send your solutions.
- Send your solutions by e-mail to address zdenek.sawa@vsb.cz.
- You must present your solutions to the lecturer at the end of semester.
- To obtain your credit, you need to obtain at least 51 points.
- It will be also possible to obtain points for problems solved in CTU Open 2025 programming contest
  (20 points for each problem solved in this contest).

Content of the course

The goal of the course is to better understand how to design and implement algorithms.

In particular:

- general methods used to create algorithms (recursive algorithms, dynamic programming, greedy algorithms)
- some graph algorithms
- algorithms working with (big) numbers
- solutions of some combinatorial problems
- solutions of some problems from the area of computational geometry
- ...

Algorithms and Problems

Algorithm are used as solutions of **problems**.

In a description of a problems there should be specified:

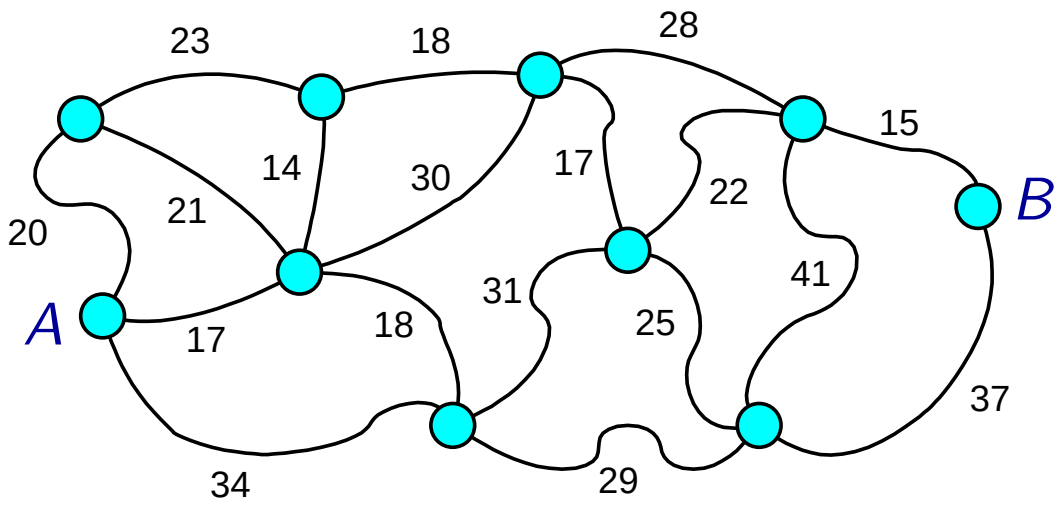
- What is the **input**.
- What is the **output**.
- How the output depends on the input.

Remark: A particular input of a problem is called an **instance of the problem**.

Example of a problem

Input: A list of cities and roads connecting these cities.
It is specified for each road, from which city to which it goes, and its length (in km).
Two cities from the list of the cities – Let us denote them a cite A and a city B.

Output: A shortest path from city A to city B.



Why this course

- Organization **ACM (Association for Computing Machinery)** organizes a world-wide programming constest called **ACM International Collegiate Programming Contest (ICPC)** every year since 1977.
- Student teams from universities from all over the world participate in this contest.
- Before world finals there are regional contests (for example Central European Regional Contest), which is preceded by national contests.
- For many years, there is a Czech and Slovak contest called CTU Open (Prague – Brno – Ostrava – Pilsen – Bratislava – Žilina – Banská Bystrica – Košice).
- One of the motivations for this course: To prepare our students for types of problems that typically appear in this contest.

Server Online Judge

- In programming contests, solutions are evaluated automatically – evaluation system sends some testing data as an input, and evaluates the output of the program.
- Similar systems as are used in such contests are also available on Internet. One of the biggest and the best known of them is on address

  https://onlinejudge.org/

- Several thousands of problems are available on this server.
- In Programming Seminar, problems from this server will be specified on the web page

  http://www.cs.vsb.cz/sawa/spr-en/problems.html

  Successful solutions will be those programs that will be accepted by this server.

Evaluation of solutions of problems on Online Judge

- It is necessary to register on the server, to send solutions there.
- Problems have assigned numbers.
- There is a web form for submitting solutions, to which you go by clicking on "Submit" button.
- A source code is send to the server, not a compiled binary.
- A whole program must be in one file.
- The following programming languages can be used:
  - C
  - C++
  - C++11
  - Java
  - Pascal
  - Python

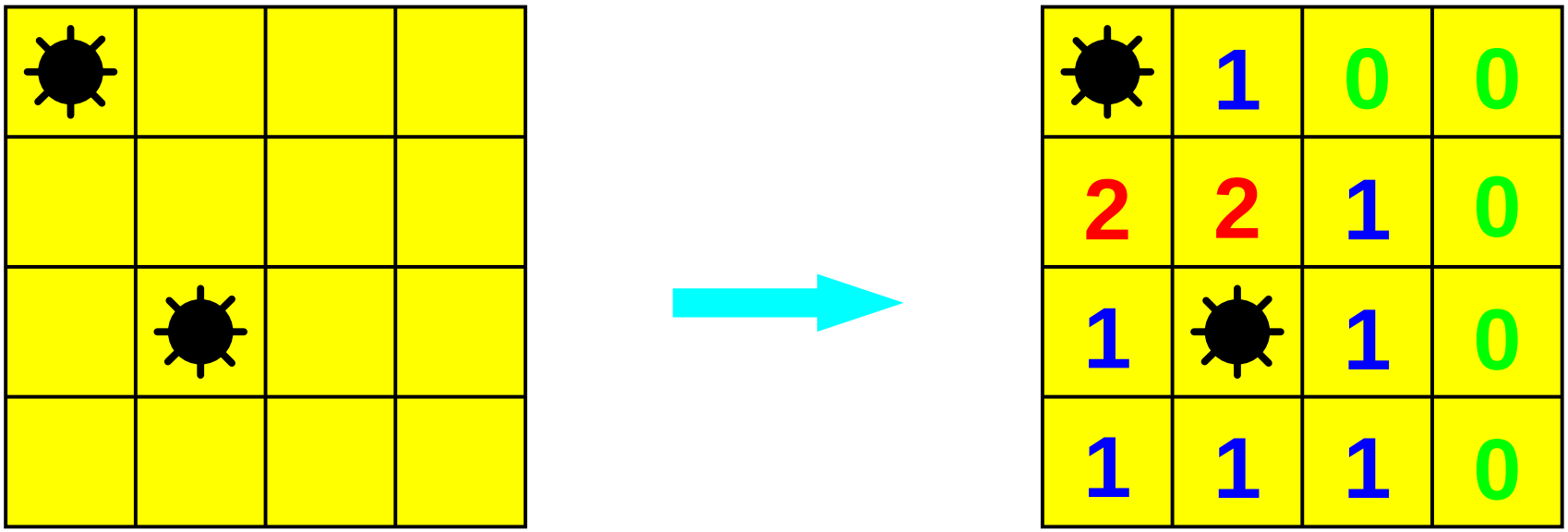
Evaluation of solutions of problems on Online Judge

- All problems are of the form where:
  - a program reads data from the standard input
  - the program writes data to the standard output
- Statements of problems contain detailed description of the format of input and output data.
- In all cases, both input and output are pure ASCII text.
- The server tests only if your program gives correct output for given input data.
- The statements of problems always contain an example of input data and corresponding output data.

  Remark: If your program works on sample input, this does not necessarily mean that your solution is correct and will be accepted by the server.

Example of a problem: Minesweeper (10189)

The problem: to find out for each empty cell the number of mines in nighbouring cells



An example of input and output

Sample Input

```
4 4
*...
....
.*..
....
3 5
**...
.....
.*...
0 0
```

Sample Output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

Evaluation of problems on Online Judge

How solutions are evaluated on Online Judge:

- You will send your solution to the server using a web form.
- The program is compiled.
- The program is run and test data are sent to its standard input.
- If the program does not finish its computation in a specified time limit, it is killed.
- If the program successfully finishes, its output is compared with an expected output (remark: sometimes a special test program is used on the server for this purpose if there can be more than one correct output).
- You will find the result of the evaluation (i.e., if your solution was accepted or not) on the webpage accessed by "My submissions" item in the menu on the left.

Evaluation of problems on Online Judge

The possible answers of the server (Online Judge):

- **Accepted** – the problem was successfully solved
- **Compile Error** – it was not possible to compile the program
- **Restricted Function** – the program uses some function whose use is restricted
- **Runtime Error** – the program ended with some error during its execution (e.g., Segmentation Fault)
- **Time Limit Exceeded** – the program was running too long, and so it was killed
- **Wrong Answer** – the program produced an incorrect output
- **Presentation Error** – the output seems to be almost correct but its format is not exactly as expected

Evaluation of problems on Online Judge

There are some restrictions on programs sent to the server concerning what kinds of operations can be executed by the program.

It is not allowed to:

- work with files (except reading from the standard input and writing to the standard output)
- communicate using network
- run other processes
- communicate with other processes
- call any other system calls of an operating system

On the other hand, the following is allowed:

- to read from the standard input and to write to the standard output
- to allocate memory (the maximal amount of memory available to the program is limited (approx. 10–20 MB))
- to use function from standard libraries (mathematical functions, manipulation with strings, data structures, . . .)

Evaluation of problems on Online Judge

Some remarks:

- Statement of each problem precisely specifies the format of inputs. Programs **do not need** to solve situations when data on the input are not of this format.
- On the other hand, programs should not assume anything about the data that is not explicitly stated in the statement of the problem.
- Statements of problems precisely specify a format of the output. The output must be **exactly** of this format.
- The system does not provide any information about particular test data that were used. It is not possible to obtain the data causing for example response **Wrong Answer**.
- The number of tries (i.e., the number how many times a program is sent to the server) is not limited.

Read statements of the problems carefully!

Evaluation of problems on Online Judge

Some other remarks:

- Input data are of typically of the form that allows to test a program for many differrent instances of a problem. The statement of the problem specifies how individual instances are separated in the input (e.g., by an empty line).
- Test data are usually quite big (it can be many MB of data).
- It is necessary to be careful with boundary cases that can be allowed by the statement of the problem.

Evaluation of problems on Online Judge

Some remarks concerning programs in Java:

- A program is sent to the server as a single file.
- The program can contain several classes but none of them should be `public`.
- There should not be a `package` name at the beginning of the file, i.e., all classes defined in the program belong to the default unnamed package.
- The program must contain a class called `Main`.
- The class `Main` must contain a static method

  ```
  public static void main(String[] args)
  ```

  that is called by the system to start the program.

Sending solutions of problems

- Send your solutions by e-mail to address `zdenek.sawa@vsb.cz`.
- Send source codes of your solutions as attachment to e-mails.
- The name of the file with a source code should be of the form ⟨*problem_number*⟩.⟨*extension*⟩ where:
  - ⟨*problem_number*⟩ – the number of the problem,
  - ⟨*extension*⟩ – filename extension determines the programming language (`.c`, `.cpp`, `.c11`, `.java`, `.pas`, `.py`)

  For example: `10189.c`, `10189.cpp`, `10189.c11`, `10189.java`, `10189.pas`, `10189.py`
- The e-mail should also contain: your name and surname, login, and numbers and names of the problems whose solutions you are sending.
- Do not pack files into archives and do not compress them.
- Do **not** send any additional unnecessary files (e.g., compiled executable binaries).

Points

- The number of points specified for each problem is the maximal number of points you can get for solving this problem.
- The number of obtained points is finally determined at the presentations of your solutions at the end of semester.
- Cancelling of points in the cases when solutions were copied:
  - the first case: **–20 points**
  - the second case: **–50 points**
  - the third case: not obtaining the credit for the course

Remark: A solution is considered copied also in the cases when it was obtained by modifications of an existing solution (renaming identifiers, changes in formatting, changing order of functions or methods in the source code, etc.).

Programming contest
CTU Open Contest 2025

November 7 and 8, 2025

Info for contestants from TU Ostrava:
http://acm.vsb.cz

Official pages of the contest, registration form:
http://contest.felk.cvut.cz/25prg

CTU Opem programming contest

- Team with three members solve problems (typically 8–12 problems).
- Limited time for solving (5 hours).
- The contest is distributed and runs concurrently at several universities (Prague, Brno, Ostrava, Pilsen, Bratislava, Žilina, Bánská Bystrica, Košice).
- The best teams from each university have a chance to advance to Central European Reginal Contest that will be held in Wroclaw in Poland, from 5th to 7th December 2025 .

Remarks about implementation

Standard input and output

In all common operating systems (MS Windows, all different variant of Unix), it is possible to **redirect** standard input and output when programs are run from command line.

- Standard input can be read from file instead of keyboard:

```
./program < input.txt
```

```
program.exe < input.txt
```

- Standard output can be written to file instead of screen:

```
./program > output.txt
```

- Both can be used at the same time:

```
./program < input.txt > output.txt
```

Reading from standard input in C

- Reading individual chars (bytes) one by one:

```
int getchar(void);
```

- It is possible to return back the last read char:

```
int ungetc(int c, FILE *stream);
```

Usage:

```
ungetc(c, stdin );
```

Reading from standard input in C

Reading lines:

```
char* gets(char *s);
```

It is strongly discouraged to use this function!

Better variant:

```
char* fgets(char *s, int size, FILE *stream);
```

Usage:

```
#define BUF_SIZE 1024
char buffer[BUF_SIZE];
char *s = fgets(buffer, BUF_SIZE, stdin);
```

Reading from standard input in C

Formatted input:

```
int  scanf(const char *format,   ...);
```

Example of usage:

```
int m, n;
while (scanf("%d %d", &m, &n) == 2) {
    ...
```

Function scanf() returns the number of parameters that were successfully read. It returns EOF in the case of error.

Reading from standard input in C

Reading blocks:

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

Usage:

```
#define BUF_SIZE 1024
char buffer[BUF_SIZE];
size_t ret = fread(buffer, 1, BUF_SIZE, stdin);
```

Writing to standard output in C

One char:

```
int putchar(int c);
```

One line (it automatically adds '\n' at the end):

```
int puts(const char *s);
```

Formatted output:

```
int printf(const char *format, ...);
```

Writing blocks:

```
size_t fwrite(const void *p, size_t sz, size_t nmemb, FILE *stream);
```

Function qsort

Declaration

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int(*compare)(const void *, const void *));
```

Return value of function compare:

- < 0 – the first argument is smaller than the second
- $= 0$ – the first and the second arguments are equal
- > 0 – the first argument is greater than the second

Function qsort

Usage

```
int a[LEN];
   ...
int compare(const void *xv, const void *yv) {
    const int *x = (int *)xv;
    const int *y = (int *)yv;
    if (*x < *y) return -1;
    else if (*x > *y) return 1;
    return 0;
}
   ...
qsort(a, n, sizeof(int), compare);
```

Function qsort

Usage

```
int a[LEN];
   . . .
int compare(const void *x, const void *y) {
     return *(const int *)x - *(const int *)y;
}
   . . .
qsort(a, n, sizeof(int), compare);
```

Remark: To compare strings, the function strcmp can be used.

ASCII chars

- In C, C++, and Jave, there is no distinction between a character and its ASCII (resp. Unicode) code. We can do the same operations with characters as can be done with integers.

```
for (int i = 'A'; i <= 'Z'; i++) {
    a[i] = . . .
}
```

```
for (int i = 65; i <= 90; i++) {
    a[i] = . . .
}
```

ASCII table

0	NUL	16	DLE	32		48	0	64	@	80	P	96	‘	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	’	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Examples of taking advantage of properties of ASCII table

- Transformation of a digit to a corresponding value:

```
char c; // c contains chars '0', '1', '2', ..., '9'
int x = c - '0';
```

- Transformation of a value x in interval 0, 1, . . . , 9 to char:

```
char c = x + '0';
```

- Transformation of letters 'A', 'B', ..., 'Z' to numbers 0, 1, ... , 25:

```
int x = c - 'A';
```

- Testing whether variable c contains a lower-case letter, and if it is the case, transformation of this letter to upper-case:

```
if (c >= 'a' && c <= 'z') {
    c = c - 'a' + 'A';
}
```

Examples of taking advantage of properties of ASCII table

- Transformation of a hexadecomal digit to the corresponding value:

```
int hex2num(char c) {
    if (c >= '0' && c <= '9') return c - '0';
    if (c >= 'A' && c <= 'F') return c - 'A' + 10;
    if (c >= 'a' && c <= 'f') return c - 'a' + 10;
    return -1;
}
  . . .
int x = hex2num(c);
```

Examples of taking advantage of properties of ASCII table

Version 2: to precompute a table

```
int hex_table[256];
int init() {
    for (int i = 0; i < 256; i++) {
        if (i >= '0' && i <= '9') hex_table[i] = i - '0';
        else if (i >= 'A' && i <= 'F') hex_table[i] = i - 'A' + 10;
        else if (i >= 'a' && i <= 'f') hex_table[i] = i - 'a' + 10;
        else hex_table[i] = -1;
    }
}
  ...
int x = hex_table[c];
```

Remark: In this case, the effect of using table is negligible. It makes more sence for example for Unicode for transformations between lower-case and upper-case, etc.

Function strlen in C

Is something wrong with the following contruction?

```
char *s;
    ...
for (int i = 0; i < strlen(s); i++) {
      // do something with s[i]
     ...
}
```

Function strlen in C

Is something wrong with the following contruction?

```
char *s;
    ...
for (int i = 0; i < strlen(s); i++) {
        // do something with s[i]
      ...
}
```

Function strlen requires time growing linearly with the length of the string!
If n is the length of the string then the loop shown above has the time complexity $O(n^2)$, not $O(n)$.

Function strlen in C

Simple solution:

```
char *s;
    ...
int l = strlen(s);
for (int i = 0; i < l; i++) {
        // do something with s[i]
    ...
}
```

Using String class in Java

Is something wrong with the following loop?

```
String[] a;
    . . .
String s = "";
for (int i = 0; i < a.length; i++) {
     s += a[i];
}
return s;
```

Using String class in Java

Is something wrong with the following loop?

```
String[] a;
    . . .
String s = "";
for (int i = 0; i < a.length; i++) {
     s += a[i];
}
return s;
```

If the length of array a is n, it unnecessarily creates n instances of String class.
If m is the sum of lengths of all string in the array a, copying content between instances of String class requires time $O(m \cdot n)$.

Using String class in Java

Better solution:

```
String[] a;
    . . .
StringBuilder s = new StringBuilder();
for (int i = 0; i < a.length; i++) {
     s.append(a[i]);
}
return s.toString();
```

The total time is this case is $O(m)$ where m is the sum of lengths of all strings in array a.