INFO0027: Project 1 (20% – Groups of 2)

Bytestream mapper

Submission before 2025-04-04

1 Bytestream mapper

You will design a first-class data structure and associated algorithms to support the operations of a bytestream modification system. The system in question can be seen as a black-box that receives a stream of bytes as input, and outputs a modified stream of bytes. The possible modifications to the input stream are removing bytes and inserting bytes.

Your job is to design an efficient ADT to answer questions about the mapping of byte positions from the input stream to the output stream, and vice versa.

For instance, in the example of Figure 1, a query for byte position 1 in the input stream would return 1 (as the position of that byte in the outputstream is 1), a query for 5 would return 3, and a query for 9 would return 6 . Note that some queries cannot be answered: a query for byte position 4 in the input stream would return -1 (as this byte has been removed from the output stream).

Likewise, a query for byte position 2 in the output stream, would return 2, a query for position 7 would return 10. Again, a query for byte position 4 in the output stream, would return -1 (as that byte was added to the output stream).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q

remove(3, 2)

a	b	c	f	g	h	i	j	k	l	m	n	o	p	q		

remove(4, 3)

a	b	c	f	j	k	l	m	n	o	p	q					

add(4, 2)

a	b	c	f	R	S	j	k	l	m	n	o	p	q			

add(9, 3)

a	b	c	f	r	s	j	k	l	T	U	V	m	n	o	p	q

Figure 1: Example of 4 operations on a bytstream.

2 MAGIC ADT

More precisely, you will implement a `MAGIC` ADT supporting the following operations:

1. `MAGIC MAGICinit()`, that initializes the data structure and where MAGIC is a pointer type identifying the created instance of the MAGIC ADT;
2. `void MAGICremove(MAGIC m, int pos, int length)`, which indicates the removal of length bytes, starting from stream position pos;
3. `void MAGICadd(MAGIC m, int pos, int length)`, which indicates the addition of length bytes at stream position pos;

4. `int MAGICmap(MAGIC m, enum MAGICDirection direction, int pos);`, which queries and returns the mapping of byte position pos in the stream direction indicated.

   ```
   enum MAGICDirection { STREAM_IN_OUT=0, STREAM_OUT_IN=1 };
   ```

5. `void MAGICdestroy(MAGIC m)`, which disposes of this instance of the ADT.

The context in which this ADT must operate is such that it must be efficient in terms of space and time. Indeed, it can be used in a system dealing with rapidly changing number of byte streams, with each stream size ranging from a few bytes to a few gigabytes, in an environment where bytestreams arrive over multiple 40Ge ethernet interfaces.

There may not be a way to know the size of an input stream in advance (the system sees each input stream "on-the-fly") and your ADT must then answer queries dynamically

The MAGIC ADT will be used as a `C` library, from our point of view the data structure is opaque (`typedef struct magic *MAGIC;`).

3 Evaluation

This project counts towards 20% of your final mark. In this project, you will be evaluated on:

- algorithms and data structures: implementation and motivation;
- code quality and maintainability: as in your code should be made for human to read not only for the computer to execute. Imagine a team handover, what should they know?
- Functional Suitability,
- Performance Efficiency,
- Test Plan.

4 Deliverables

Projects must be submitted before 2024-04-04, 23:59, on the submission platform. After this time, a penalty will be applied to late submissions. This penalty is calculated as a deduction of 2^{N-1} marks (where N is the number of started days after the deadline).

Your submission archive should contain your **implementation** of the MAGIC ADT, as well as your **report** in PDF format, structured as showed in Figure 2:

```
teamName.tar.gz
├── src
│   ├── magic.c
│   └── magic.h
├── *.pdf
└── * for examples tests
```

Figure 2: Submission archive structure

The report should be in English, up to 3 pages long, explaining your algorithms, its complexity, and include your test plan and performance study. Your implementation will be compiled with `gcc`, with the following flags: `-Wall -pedantic -std=c11 -O3`, if you need something different contact us early.

Failure to compile will result in an awarded mark of 0. Likewise, poor spacial or time performance when run over large input will result in an awarded mark of 0. Warnings systematically result in lost marks.

Ask your questions early, and have fun...