Practical Work: Algorithm puzzles

Master CSMI
Compilation & Performance
Bérenger Bramas

December 9, 2020

1 Summary

In this work, you will solve algorithm problems, and you will do it under time constraint.

2 Warning

- You have two (2) hours to complete the test, and you must push all your code when you will heard that the test is over.
- Only `https://en.cppreference.com/` is allowed, all other websites are forbidden (including stackoverflow or even Google).
- There is no *report.md* file, simply the code to commit.
- You can ask me questions!
- Coding style/quality will be evaluated too.

3 Practical work organization (always the same)

In the practical work, you will obtain the code from my repository and push it to your repository. Therefore, you will have to clone one branch per session and push it to your own repository. You will also have to put a Git tag at the end of the session such that I can easily look at what you have coded at the end of the session (and potentially compare it with the latest version you will have).

In the rest of the document, we consider you have a repository named *cnp-tp-2019* on *git.unistra.fr* that is private but that I can access in read.

3.1 Get the practical work

Consider you are in your project directory do the following:

```
# Clone my repo
git clone https://git.unistra.fr/bbramas/csmi-tp-2019.git --branch=TP8 csmi-tp8
# If you use SSH replace [USER] and use:
# git clone git@git.unistra.fr:[USER]/csmi-tp-2019.git --branch=TP8 csmi-tp8
# Go in the newly created directory
cd csmi-tp8
```

3.2 Add your repository as remote

You will push on your own repository:

```
# Rename my remote
git remote rename origin old-origin
```

```
# Add your own remote
git remote add origin https://git.unistra.fr/[YOU LOGIN HERE]/cnp-tp-2019.git
# If you use an SSH key:
# git remote add origin git@git.unistra.fr:[YOU LOGIN HERE]/cnp-tp-2019.git
# Push the current branch and active the tracking
git push -u origin TP8
```

3.3 During the session and while you work on the project

After each question or important modification push the current changes:

```
# No matter where you are in the project directory
git commit -a -m "I did something"
git push
```

3.4 When you are done

You have fully finished your work (at most D+14 H-2):

```
git commit -a -m "I did something"
git push
```

3.5 Compilation

To compile, we use CMake:

```
cd Code
mkdir build
cd build
cmake ..
make # Will make all
make something # Will build only something
VERBOSE=1 make # Will show the commands used to compile (including the flags)
```

4 Problem 1 - Merge A and B

- File: merge_AB.cpp
- Problem: We consider two sorted arrays A and B and we want to obtain a single sorted array that contains the elements of A and B. Said differently, we want to merge A and B. However, we want to merge B into A, considering that there is enough space that was allocated (ie. A is on a memory block of size $|A| + |B|$). Implement this algorithm in the function *merge_AB*.
- Complexities: Space complexity should be O(1), algorithm complexity should be linear regarding the size of A and B.
- Example:

```
# input
A = [0 1 1 2 3 4 4]
B = [0 0 4 4 5]
# ouput
A = [0 0 0 1 1 2 3 4 4 4 4 5]
```

5 Problem 2 - Are all chars unique

- File: unique_char.cpp

- Problem: We consider a given string s, and we want to check if all chars in s are unique. Implement this algorithm in the function *has_unique_chars*.
- Complexities: Space complexity should be constant (that is not related to the length of s), algorithm complexity should be linear regarding the length of s.
- Example:

```
has_unique_chars("abc") => true
has_unique_chars("aba") => false
```

6 Problem 3 - Queue when only stack is available

- File: queue_with_stacks.cpp
- Problem: We want to implement a queue (FIFO) using two stacks (a stack is FILO). Implement this approach in the class *Queue*. You will notice that the *pop* function not only remove the top value, but also returns it.
- Complexities: Space complexity should be linear regarding the number of inserted elements, algorithm complexity should be linear regarding the number of inserted elements.
- Example:

```
Queue q;
CheckEqual(0, q.size());
q.push(1);
CheckEqual(1, q.size());
q.push(2);
CheckEqual(2, q.size());
CheckEqual(1, q.pop());
CheckEqual(2, q.pop());
```

7 Problem 4 - Stairs

- File: stairs.cpp
- Problem: Imagine that a kid is climbing the steps of a stair to go up. He can climb them by one, two or three steps at a time. We are interested in the number of possibilities there are to climb N steps. Implement a way to know it in the function *stairs*.
- Complexities: Space complexity should be linear regarding N, algorithm complexity could be N^3, but ideally it should be linear or the large tests may failed.