Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of “plagiarism”:

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may…
• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

You may not …
• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one’s account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don’t think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else’s work.

CS100 Homework 7 (Spring, 2021)

Submission deadline:

2021-06-09 23:59

Problem 1. Warm-up: Rational Operations and CMake

In this warm-up problem, you will implement a class for rational numbers in separated header and source files, and compile a multiple-file project using CMake. CMake is the only new knowledge required for this problem beyond the scope of last homework (homework 6).

All **rational numbers** can be written as quotients p/q, where p is an integer and q a positive integer. The result of addition, subtraction, multiplication, and division of two rational numbers will still be a rational number.[1] You need to implement these operations for the `Rational` class, as well as a few other functions, as shown in this template:

```
class Rational
{
public:
    // Constructors
    Rational();
    Rational(int value);
    Rational(int p, unsigned int q);

    // Assignment operator
    Rational& operator=(const Rational& that);

    // Arithmetic operators
    Rational  operator+(Rational that) const;
    Rational& operator+=(Rational that);
    Rational  operator-(Rational that) const;
    Rational& operator-=(Rational that);
    Rational  operator*(Rational that) const;
    Rational& operator*=(Rational that);
    Rational  operator/(Rational that) const;
    Rational& operator/=(Rational that);

    // Comparison operators: equal and less than
    bool operator==(Rational that) const;
    bool operator<(Rational that) const;

    // Output
    friend std::ostream& operator<<(std::ostream& os, const Rational& number);
private:
    int m_numerator;
    unsigned int m_denominator;
};
```

- There are 3 constructors for this class. They respectively create a rational number of 0, `value`, and p/q.
  - If q is 0 in the third constructor, you should print a message `"ERROR: initializing with a denominator of 0!"` with a new line, and set this rational number to 0 instead.
  - The rational number you created should be simplified. For example, the line `Rational r(-6, 10);` should instead initialize the number to -3/5.

[1] In sense of mathematics, a set with such properties (the set of all rational numbers) is called a field.

- The assignment operator is self-explanatory.

- The arithmetic operators are also self-explanatory. The only notes are:
  - When a division by 0 happens, you should print a message `"ERROR: dividing by 0!"` with a new line, and you can set the result to any value. In fact, such case would not appear in either Autolab tests or usage for problem 2. If in problem 2 you see your error message, your calculations are probably wrong.

  - Any result of these operators should be simplified. For example, the expression `Rational(1, 3) + Rational(1, 6)` should evaluate to 1/2.

  - Pay attention to the difference of `operator+` and `operator+=`. Hint: the `operator+=` modifies the number itself and returns a reference.

- At least the two comparison operators == and < are needed for next problem.

- The `operator<<` of `std::ostream` should print the number into `os` in the form of p/q, p for numerator and q for denominator.

  - If the number is an integer, you should print as an integer instead, i.e. omit the "/q" part.

Instruction on multiple files and CMake:

- Your "rational.hpp" file should only contain the definition of Rational class and its functions.

- Your "rational.cpp" file should contain implementation of functions in your Rational class.

- You should write a "CMakeLists.txt", in which you should:
  - Set the minimum version limit to `"VERSION 2.8"`.
  - Create a project named "rational".
  - Generate a build system for your computer such that it compiles "rational.cpp" along with a file "main.cpp" (you can create this file when debugging, but don't submit it—we will provide a "main.cpp" when grading.), and creates an executable named "rational".

  - **It's okay if CMake behaves differently on your computer (for example, creates a Visual Studio solution). On the Autolab server (Linux), running `cmake` will generate a "Makefile". Then running `make` will build the executable "rational".**

- Your "rational.hpp" should have a proper **include guard**. That is, if your code is compiled with a "main.cpp" of the following content, it must compile:

```
#include "rational.hpp"
#include "rational.hpp"
int main(){ return 0; }
```

Submission:

Please submit to Autolab a tarball (*.tar) file that contains three files: "rational.hpp", "rational.cpp", and "CMakeLists.txt". The name of this tarball file could be arbitrary.

If you have trouble making a tarball, you can download any archiver software.
(Recommended: Bandizip)

Problem 2. Templated Gauss-Jordan Elimination

In this problem, you will write a class template of `Matrix` to create matrices for both `double` and `Rational`, and perform a Gauss-Jordan Elimination (also known as Gaussian Elimination).

Gauss-Jordan Elimination is a method to solve systems of linear equations by transforming it into row echelon form with row operations. This method can also be used to compute the rank of a matrix, the determinant of a square matrix, and the inverse of an invertible matrix.[2] We will provide a detailed pseudocode for the algorithm so that you could implement even if you are unfamiliar with linear algebra.

For now, let's look at the structure of the templated class Matrix:

```
template<typename T>
class Matrix
{
public:
    Matrix();
    Matrix(unsigned int rows, unsigned int cols, const T& value = T());

    T& operator()(unsigned int r, unsigned int c);

    void Print();
    void GaussJordan();
private:
    // Your implementation here.
};
```

- Your `Matrix` should store its data in a vector of vectors.
- The second constructor creates a matrix with given `rows` and `cols`, with every entry initialized to the given `value`. Note that a default argument of `T()`, the empty constructor, is assigned to `value`, so that you may omit it when declaring a `Matrix` with default entries.
- The `operator()` is used for assigning to a particular entry in the matrix.
- The `Print()` method prints the matrix to `std::cout`, in the simple format below:
    - For each entry, print it to `std::cout`, followed by a space `" "`.
    - After each line, print a newline. It's allowed that each line ends with a trailing space, and a new line is after the last row.
    - No setw() or setprecision() is needed.
- Finally, the method `GaussJordan()`. This method does the Gauss-Jordan Elimination in-place (altering contents of the matrix), so that the matrix is in row echelon form (not reduced row echelon form). As the resulted matrix is not unique, you should follow the given pseudocode to implement the algorithm:

[2] Gaussian elimination - Wikipedia

```
h := 1 /* Initialization of the pivot row */
k := 1 /* Initialization of the pivot column */

while h ≤ m and k ≤ n
   /* Find the k-th pivot */
   /* record the max and min of k-th column after h-th row */
   i_max := argmax (i = h ... m, A[i, k])
   i_min := argmin (i = h ... m, A[i, k])
   if A[i_max, k] = 0
      /* max = 0, swap max with min */
      i_max := i_min
   if A[i_max, k] = 0
      /* max = min = 0, no pivot in this column, pass to next */
      k := k+1
   else
       swap rows(h, i_max)
       /* Do for all rows below pivot: */
       for i = h + 1 ... m:
          f := A[i, k] / A[h, k]
          /* Fill with zeros the lower part of pivot column: */
          A[i, k] := 0
          /* Do for all remaining elements in current row: */
          for j = k + 1 ... n:
             A[i, j] := A[i, j] - A[h, j] * f
       /* Increase pivot row and column */
       h := h + 1
       k := k + 1
```

- Note that the highlighted parts have been modified: the original algorithm chooses the row with greatest absolute value to swap as pivot, but the Rational class from problem 1 cannot use the same method as double to calculate absolute values, so we need to instead check both max and min, prioritizing max.

Testing:

We have provided two simple test functions on your matrix in the given file “tests.cpp”. The expected output of your program is included in comments.

Instruction on multiple files and CMake:

- As declaration and implementation of a templated class are tricky to separate into header files and source files (because member functions of different types are generated and compiled given the template types), you should write all your code in a single file named “matrix.hpp”.

- You should write a “CMakeLists.txt”, in which you should:
  - Set the minimum version limit to `"VERSION 2.8"`.

- Create a project named “matrix”.
- Generate a build system for your computer such that it compiles “rational.cpp” along with a file “main.cpp” (you can create this file when debugging, but don’t submit it—we will provide a “main.cpp” when grading.), and creates an executable named “matrix”. You do not need to compile “matrix.hpp” along, as it’s a header file and it will be included in “main.cpp”.
- **It’s okay if CMake behaves differently on your computer (for example, creates a Visual Studio solution). On the Autolab server (Linux), running `cmake` will generate a “Makefile”. Then running `make` will build the executable “matrix”.**

- Your “matrix.hpp” should have a proper **include guard**. That is, if your code is compiled with a “main.cpp” of the following content, it must compile:

```
#include "matrix.hpp"
#include "matrix.hpp"
int main(){ return 0; }
```

- Your “matrix.hpp” should not `#include "rational.hpp"`.

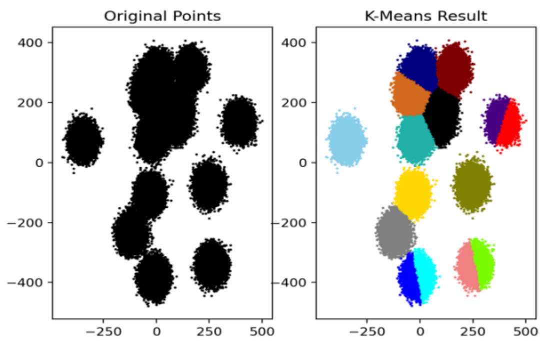
Submission:

Please submit to Autolab a tarball (*.tar) file that contains four files: “matrix.hpp”, “rational.hpp”, “rational.cpp”, and “CMakeLists.txt”. The name of this tarball file could be arbitrary.

Problem 3. Parallel K-Means Clustering

The *K-Means* Clustering Algorithm is one of the most straight-forward clustering algorithms in data science. It aims at solving *Vector Quantization* problems.

We will be dealing with K-Means on a 2-Dimensional data set. As shown in the left-hand side of the following figure, given N points (coordinates) and a number K, you need to classify these points into K clusters. For example, the right-hand side of following figure shows the K-Means clustering result of an example data set, with number of data sample points N=300,000 and desired number of clusters K=15 (each color indicates a cluster).



This is the procedure of K-Means algorithm:

1. Each center point c represents the center (i.e. mean) of a cluster.
2. For every sample point p and every center point c_i, calculate the distance between p and c_i: d(p, c_i).
3. Assign every sample point p to the cluster x, where d(p, c_x) is the smallest among all d(p, c_i)s.
4. Update the center point c_i of each cluster i to be the current mean of all points p_i within the cluster.
5. Goto 2. and repeat, unless no updates happened in this iteration (i.e. converges).

This is the pseudo-code for K-Means algorithm, while it may be hard to understand:

```
1. Initialize cluster centroids $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$ randomly.
2. Repeat until convergence: {
       For every $i$, set
           $c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$
       For each $j$, set
           $\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$
   }
```

In this homework, we provide you a framework for this algorithm. In this framework, each data point is an instance of `struct Point`. Some functions of `struct Point` is provided for you. You only need to implement these 3 functions:

- `double Point::Distance(Point& other)`
    - Calculate the Euclidian distance between this point and “other”

- `std::ostream& operator<<(std::ostream& os, Point& pt)`
    - Output the content of `Point& pt` to the `os`, the format is: first, output `pt.x`, then output a space (“ ”), then output `pt.y`. Notice you don’t need to start a new line (i.e., no “\n” will be printed). For example, you should output the `Point(3, 4.5)` by “`3 4.5`”.
- `std::istream& operator>>(std::istream& is, Point& pt)`
    - Scan “x” and “y” of `Point& pt` from the `std::istream& is`, the input is two doubles separated by a single space (“ ”).

For the K-Means class, you need to implement the constructor and the function `Kmeans::Run` .

- `Kmeans::Kmeans(const std::vector<Point>&, const std::vector<Point>& initialCenters)`
    - Constructor of K-Means class, `points` is a vector contains coordinates of data sample points, `initialCenters` contains coordinates of initial centers.
- `std::vector<index_t> Kmeans::Run(int maxIterations)`
    - Run the K-Means algorithm. Returns the index (color) of all data points. For example, if you have 1,000 data points, and you want to classify them into 5 clusters. Then, the return value of this function is a vector with size=1000, indicates which cluster of each point belongs to. The value of returned vector is in {0, 1, 2, 3, 4}. (e.g., `result[9]=3` means `points[9]` was divided to cluster 3).
    - The clustering loop runs at most `maxIterations` times. If it does not converge in `maxIterations` times, just break from the loop and return the current result.
    - **<u>IMPORTANT: YOU MUST USE MULTI-THREADING TO SPEED IT UP</u>.**
      As discussed in class, we can use `std::thread` to speed up programs by multi-threading. Notice that the distance calculation of points and centers is individual, so it’s easy to calculate them in parallel. We suggest you to use **<u>4 threads</u>** on Autolab, using too many threads will make it even slower.

There are 6 files provided for you. You need to modify code in “`kmeans.cpp`” and “`kmeans.hpp`”. DO NOT modify any code in “`main.cpp`”, or your code may not accept by OJ. You can use “`CMakeLists.txt`” for building this project, “`generate.py`” to generate samples for testing and debugging, and use “`visualize.py`” to visualize your clustering result.

Hope you have already installed Python3 on your own computer. For this homework, you may need to install dependencies by the following command:

```
python -m pip install numpy sklearn matplotlib
```

After that, you can test your program by these 3 commands (suppose your executable file is “`./kmeans`”):

```
python generate.py in.txt
./kmeans out.txt < in.txt
python visualize.py out.txt
```