CSCI3100 Software Engineering

Assignment 5

Due – 11:59:59pm, 18th April 2021 (Sunday)

Please submit the homework online through Blackboard.

Late submission penalty within 24 hours: 50%; after 24 hours: 100%.

Remember to go through Veriguide for Academic Honesty Declaration.

Missing Veriguide report: 50% mark deduction.

Answer the following problems based on lecture Topic 5 notes. Each question is assigned 25 points.

1. UML Class and Sequence Diagrams (25 points)

In the CSE Department in CUHK, the course CSCI3100 is known for its interesting and challenging homework design. Recently, there are a lot of questions in Piazza showing that many students struggled with homework 3 (HW3). What if we develop an AI system called **SuperBoard** to help us automatically solve the problems? The system is introduced as follows. You are required to draw UML diagrams to design different entities involved in such a system.

(1) (12 points) Each CUHK student has a unique **CUHK Account** which contains an *account id* and a *student name*. Based on this parent account, a CSCI3100 **Student** will have an heir **CSCI3100 Account** to record the *coupon balance* received from the CSCI3100 class, which will be used to obtain problem solutions. A student should use his/her CSCI3100 account to sign in the **SuperBoard** platform. SuperBoard includes a *problem list* and a **Solver**. Students can upload a **Problem** to SuperBoard. Each problem includes a *problem ID*, *problem description*, *number of coupons required*, and *the solution.* The core of SuperBoard is the Solver powered by some kind of strong AI[1]. The Solver will assign the required number of coupons to each uploaded problem automatically. A student can submit coupons to the solver on SuperBoard platform and the solver can solve the problem and return the solution. Because of the power of strong AI, we can assume the solver is able to assign the required number of coupons and solve any problems. After getting the solution, the student will delete the problem from SuperBoard platform.

Please draw a UML class diagram to describe all the objects of this system and specify the attributes and operations associated with them.

---

[1] https://en.wikipedia.org/wiki/Artificial_general_intelligence

(2) (13 points) The following steps show the process of signing in **SuperBoard** platform and buying the solution of HW3.

a. First, a **Student** sends a sign-in request in **SuperBoard** by entering the account ID and password.

b. The account ID and password will be sent to **CSCI3100 account server**, the server will check the correctness. If it is confirmed, the **Student** successfully signs in the **SuperBoard**; otherwise, an error page will be shown.

c. Next, the **Student** uploads the description of a HW3 problem to **SuperBoard**, which is resent to **Solver server** for decide the number of required coupons for solving that problem. After that, the priced problem is sent back to **SuperBoard**.

d. The **Student** selects the problem to solve in **SuperBoard**, which will send the required number of coupons to the **CSCI3100 account server** to check if the **Student** has enough coupon balance. If the balance is not enough, the request is rejected and a notification message will be sent to the **Student**.

e. If the balance is enough, **Solver service** will send the solution to **SuperBoard**, which is returned to the **Student**.

Please draw a UML sequence diagram to describe it. There are four objects involved: **Student**, **SuperBoard**, **Solver server**, and **CSCI3100 account server**.

2. UML Sequence Diagram (25 points)

AI has been developed for a long time; unfortunately, the general Solver mentioned in Problem 1 that can understand and solve any CSCI3100 homework problem does not exist at this time. In real-world life, however, there are problem-specific solvers which attempt to find the solution for the users, such as Socratic[2], Course Hero[3], and so on. Generally, these applications maintain a large solution database with many problem-solution pairs. Given a query problem, the applications will run a complicated matching algorithm to find the most similar problems and return the corresponding solutions. If a problem is regarded as mismatched by the user, it will be assigned to a human employee to solve. Once solved, the solution database could be updated accordingly. For simplicity, we assume that solutions in the solution database and the human employee's solutions are correct.

More details are described as follows.

a. A student uploads a problem to the **Solver**. The **Solver** requests the **ProblemAnalyzer** to understand the problem.

b. The **ProblemAnalyzer** extracts the features of the problem and uses the features to search similar problems in the **Problem-solution Database**.

c. If the returned problems' similarity is larger than a threshold θ, then the top 5 problems will be sent to the **Solver**. Otherwise, the problem will be sent to the **HumanEmployee**.

d. The **Solver** requests the **Student** to choose which problem is best matched. If no one is matched, the problem will be sent to the **HumanEmployee**; otherwise, the solution to the problem will be presented to the **Student**.

e. After the problem is assigned to a **HumanEmployee**, the problem will be solved manually, and the problem together with the solution will be sent to the **Solver**.

f. The **Solver** will request confirmation from the **Student**. If the solution is regarded as reasonable, the problem-solution pair will be sent by the **Solver** to the **Problem-solution Database** for an update.

Please draw a UML *sequence diagram* to describe the working process of such a problem-solving application. Particularly, there are five objects involved: **Student**, **Solver** (frontend), **ProblemAnalyzer**, **Problem-solution Database** and **HumanEmployee**.

[2] https://socratic.org/
[3] https://www.coursehero.com/

3. Program Implementation Techniques (25 points)

In Problem 2, the key to find the most similar problem is to compute the similarity between two problem descriptions. In Problem 3, we study classical metrics to measure the similarity between two textual contents with the following three sentences (S1, S2, and S3) as examples:

S1: A B A A B

S2: A C B A B

S3: A C D C E

It is obvious to us that S1 and S2 are more similar compared with S3. But a computer needs to compute the textual similarities to figure this out. We introduce three metrics to measure the similarity between two sentences: Jaccard Distance, Hamming Distance and Edit Distance.

Jaccard Distance[4]

Jaccard Distance is based on word sets of two sentences. A word set is a set containing non-duplicated words, e.g., the word set of S1 is {A, B}. Given two word sets, the Jaccard Distance can be computed as the ratio between *the number of words in both sets* (i.e., their intersection) and *the number of words in either set* (i.e., their union). For example,

$$J(S1, S2) = 0.667,\ J(S1, S3) = 0.2,\ J(S2, S3) = 0.4$$

Hamming Distance[5]

We first build a vocabulary based on all unique words in all sentences.

Vocab (S1, S2, S3) = ['A', 'B', 'C', 'D', 'E']

Then, we convert each sentence to one-hot encoding. For each word in the vocabulary, we mark 1 if the word shows in the sentence, and 0 otherwise.

Onehot(S1) = 1 1 0 0 0 Onehot(S2) = 1 1 1 0 0 Onehot(S3) = 1 0 1 1 1

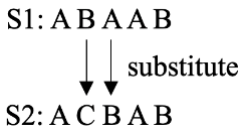
Given the one-hot encoding, Hamming Distance is the number of the cases that two values are different in the same position. So, we can obtain the following Hamming Distances:

HammingDist(S1, S2) = 1 HammingDist(S1, S3) = 4 HammingDist(S2, S3) = 3

[4] https://en.wikipedia.org/wiki/Jaccard_index
[5] https://en.wikipedia.org/wiki/Hamming_distance

Edit Distance[6]

Edit Distance is also named Levenshtein Distance. It is the minimum number of single-character edits (*insertions*, *deletions* or *substitutions*) required to change one sentence into the other. For example, we can change S1 to S2 by two substitution operations as follows:



Hence, the Edit Distance between S1 and S2 is 2.

The distances for all pairs of sentences in our examples are as follows.

EditDist(S1, S2) = 2 EditDist(S1, S3) = 4 EditDist(S2, S3) = 3

In the above edit distance examples, the distances are computed at the character level. In real-world practice, we can also compute the edit distances of two sentences at the word level. Namely, we do not split each word into characters. In Problem 2, we introduced a solution database which stores problem-solution pairs. We can compute the word-level similarities and retrieve the solution to the required problem.

---

(1) (5 points) In the following, S1 and S2 are parts of the problem descriptions in the database. *Query* is the sentence you want to search. Please calculate the three kinds of distances between the S1, S2 and Query and fill the results in Table 3-1.

S1 (from HW4): after the due date of each project phase tutors need to pull code from github

S2 (from HW3): the health code has three colors red yellow and green

Query (from HW3): if the color of health code is yellow

Table 3-1 Different Textual Similarities

Pair	Jaccard	Hamming	Edit
Query-S1			
Query-S2			

[6] https://en.wikipedia.org/wiki/Levenshtein_distance

(2) (20 points) Implement a function for each of the three distances with your preferred programming language (e.g., C++, Java, Python, etc.). For each function, it takes two raw strings as the input and outputs a value indicating the distance. Show all your code (**three functions only**) and fill your results in Table 3-2 based on your program output.

In particular, in the real-world application, there could be millions of problem descriptions in the database, so the efficiency of your implementation is important. Please also comment on your implementation of **Edit Distance** and elaborate your effort to achieve better efficiency. Hint: you can resort to *dynamic programming*[7].

S1 (from HW4): after the due date of each project phase tutors need to pull code from github the tutors have to input the team id and phase type initial code or completed code or commented code then the project management module will 1 pull the code of the team 2 generate a status report and 3 send an email notification to all team members of the team

S2 (from HW3): if the color is red then this student must go to hospital sent to hospital and conduct a fast coronavirus test covid test if the test result is negative then this student can leave and go through a 14 day quarantine go through quarantine

Query (from HW4) if the color of health code is yellow then this student needs to measure his her body temperature measure body temperature if the body temperature is normal then this student can leave and go through a 14 day quarantine after which he or she can enter

Table 3-2 Different Textual Similarities

Pair	Jaccard	Hamming	Edit
Query-S1			
Query-S2			

[7] https://en.wikipedia.org/wiki/Dynamic_programming

4. Stepwise Refinement Revisit (25 points)

Bucket sort[8] is a sorting algorithm that could achieve $O(n)$ time complexity in the best case, where n is the number of elements sorted. Given an array composed of n integers, the algorithm first distributes the integers into some buckets according to the values, for example, put the integers within the range of [e1, e2] in the first bucket. We make sure that all elements in the latter bucket (i.e., with larger index) are larger than those in the previous bucket (i.e., with smaller index). After that, elements in each bucket are sorted by some sorting algorithms such as quick sort, merge sort, and so on. Finally, the sorting results of each bucket are concatenated according to the indices of buckets. Namely, the elements in the latter bucket follow the previous bucket. In particular, when there is only one number in each bucket, bucket sort only needs to go through the buckets without element-wise comparison and achieves $O(n)$ time complexity.

(1) (10 points) Let n be the length of the integer array **a** to be sorted. Please first show the overall module structure for bucket sort with GDN then develop proper steps of stepwise refinement (at least 3 refinements).

(2) (15 points) Show your implementation of last refinement with your preferred programming language (e.g., C++, Java, Python, etc.).

Hint: you can choose any proper *method for distributing the elements to buckets* and *sorting algorithm used for sorting within the bucket*. For simplicity, we assume the largest value of the input array is 10,000.

[8] https://en.wikipedia.org/wiki/Bucket_sort