CSCI3100 Software Engineering

Assignment 4

Due – 11:59:59pm, 28th March, 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 Topics 5 notes, from which you can consider for solutions. Each question is assigned 25 points as its full marks.

1. Software Design Notation

In this question, you are going to design a simplified *project management* module for the CSCI3100 Project, and describe how it interacts with other relevant modules in an overall system. The *project management module* provides the following functionalities for students and tutors.

- **Team Registration (for students)**

You can assume a CSCI3100 Project team consists of exactly 5 students. After the project assignment is released and students have already formed teams, one of the team members should register their team in the *project management module*. During the registration, he/she should provide 1) all team members' names and student IDs, and 2) the project name of their team. After that, the *project management module* invokes a function to check whether any team member has been included in other teams. Note that this checking function should not be visible from outside the *project management module*. The function to check conflict will accept a student ID and return a Boolean value indicating whether the student has been included in other teams. If there is no conflict, the *project management module* will generate an integer team ID and send an email notification about the team information (team ID, all team members' names and their student IDs, and project name) to all team members; otherwise, the *project management module* will send an email notification about the reason of failure to all team members.

- **Submission of GitHub URL of project (for students)**

For the submission of GitHub URL, one team member needs to input the team ID and the GitHub URL. The *project management module* will then send an email notification to all team members.

- **Submission of project documents (for students)**

Project documents include initial design document and final report. For the submission of project

documents, one team member needs to input team ID, document type (initial design document or final report), and upload a file. The *project management module* will then send an email notification to all team members.

- **Pull code from GitHub (for tutors)**

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.

- **Release marks (for tutors)**

Tutors will release marks for different phases. The input is team ID, phase type, marks, and comments. Then the *project management module* will send an email notification to all team members of the team.

The *project management module* uses an abstract data type (ADT) named *GitHubRepository*. The ADT encapsulates the data structure of a GitHub repository and provides two interfaces. The first interface accepts an URL string, pulls code from the URL. The second interface generates a text report of a given repository.

The *project management module* uses the modules *EmailNotification* and *GitHubRepository*. Below are their respective designs in TDN.

```
module EmailNotification
exports
   procedure sendEmail(subject: in STRING, body: in STRING,
receivers: in ARRAY[1..5] of STRING)
   This procedure sends an email to all `receivers`.
end EmailNotification
```

(1) Using TDN, describe the ADT of *GitHubRepository*. Concentrate your design in the *module interface*. The implementation section could be omitted with comments only. Please explain your design with proper comments in your TDN.

(2) Using TDN, describe the module of the *project management module*. Concentrate your design in the *module interface*. The *implementation* section could be simplified by using comments. Please explain your design with proper comments in your TDN.

(3) Assuming *students* and *tutors* are two other modules (i.e., **module** Students and **module** Tutors), whose details are omitted. Please use GDN to depict the whole system, which includes all modules described above.

2. Software Design Principle and Approach

Suppose you are going to design an instant messaging application called *Telegraph*. *Telegraph* is a modularized application composed of *UI* module, *Chat* module, and *Contact* module. The *UI* module uses the *Chat module* and the *Contact* module to render the user interface. The *Chat* module deals with sending and receiving messages. The *Chat* module uses the *Network* module for network communication. When users send/receive messages via the network, the *Chat* module uses the *Crypto* module to encrypt/decrypt the message. The *Contact* module manages the contact list. It also uses the *Crypto* module for encryption/decryption. The *Crypto* module uses the *Math* module for calculation.

Now we propose a metric to measure the stability of modules. Formally, we define the "stability" of a module i as $S_i = \frac{d_i^{in}}{d_i^{in}+d_i^{out}}$, where d_i^{in} (i.e., *incoming connection* or *dependency*) is the number of modules that directly or indirectly USES module i and d_i^{out} (i.e., *outgoing connection* or *dependency*) is the number of modules that module i directly or indirectly USES. For example, Figure 2-1 shows module X, that is used by three modules, so $d_x^{in} = 3$ and $d_x^{out} = 0$. Thus, module *x* should be unlikely to change because it has three *incoming dependencies*. Conversely, in Figure 2-2, module Y uses three modules, i.e., $d_y^{out} = 3$, so it is more likely to change due its three *outgoing dependencies*. If a USES b and b USES c, then $d_c^{in} = 2$.

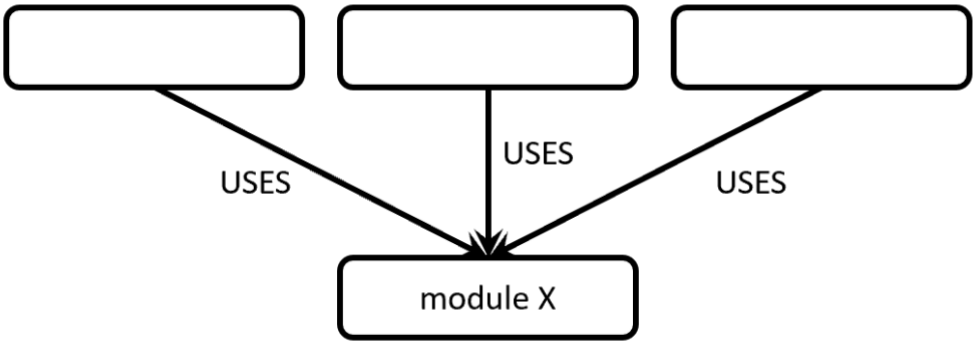

Figure 2-1: Three modules USES module X

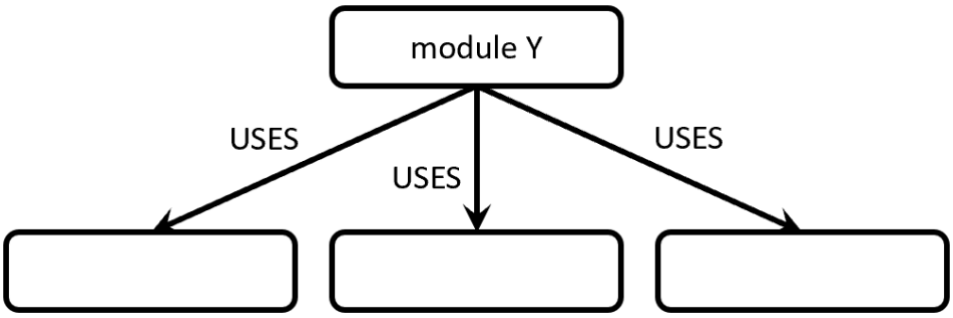

Figure 2-2: Module Y USES three modules

(1) Please use GDN to depict the whole system. Include all modules described above and show subcomponents when applicable.

(2) Draw the IS_COMPONENT_OF relation and the USES relation among the modules whose names are identified.

(3) Please calculate the "stability" of all modules described above.

(4) Can you infer what the meaning of "stability" measure of modules is in software engineering concept, and what is the purpose?

3. Program Complexity and Design Technique

The USES relation naturally forms a graph. As mentioned in the lecture, the USES relation graph should be a Directed Acyclic Graph (DAG).

(1) A graph could be represented with two different data structures, i.e., *adjacency list* and *adjacency matrix*. Represent the USES relation graph in Figure 3-1 with adjacency list and adjacency matrix.

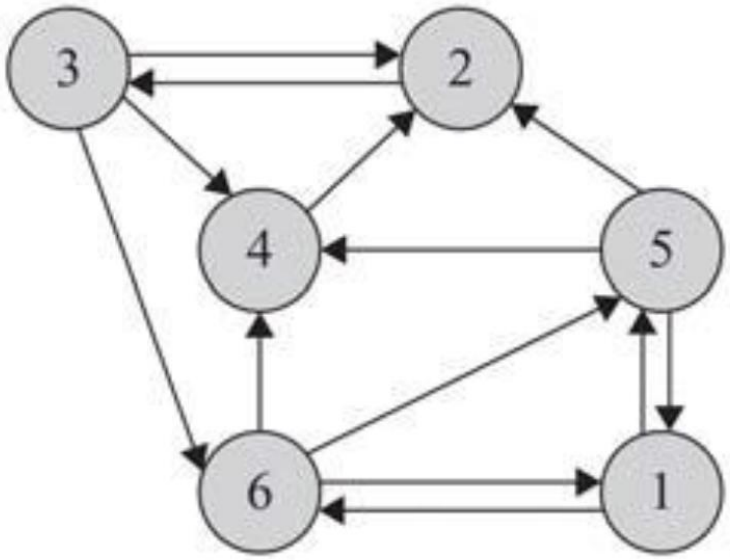

Figure 3-1: USES relation graph of the modules in a system

(2) Suppose the input of a USES relation graph is represented with adjacency list, develop a function in *pseudocode* to check whether the USES relation graph is an acyclic graph, using Stepwise Refinement technique taught in lectures. You should provide with at least 2 refinements. Assuming there are V vertices and E edges, please state the time complexity of your algorithm.

4. Software Design: Recursive and Non-Recursive Modules

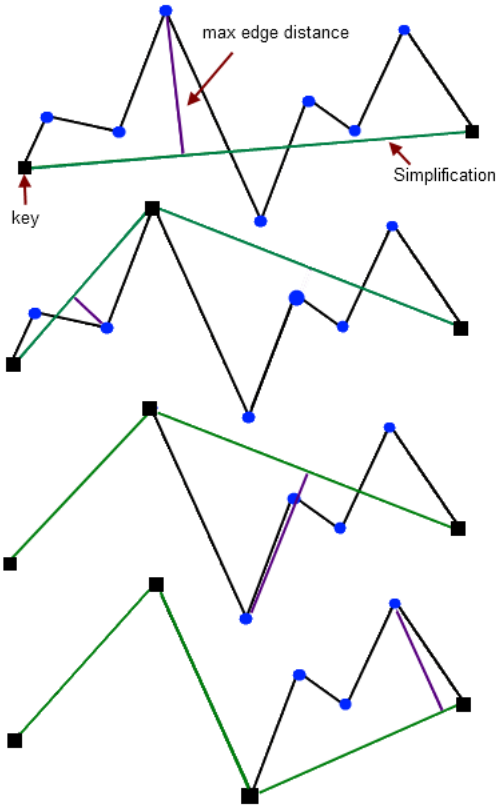

Figure 4-1: A sample process for the Douglas-Peucker algorithm

The **Douglas-Peucker** algorithm is for the selection of representative points to simplify a curve composed of line segments. It uses a point-to-edge distance tolerance. The algorithm starts with a crude simplification that is the single edge joining the first and last vertices of the original polyline. It then computes the perpendicular distance of all intermediate vertices to that edge. The vertex that is furthest away from that edge, and that has a computed distance that is larger than a specified tolerance, will be marked as a key and added to the simplification. This process will recurse for each edge in the current simplification until all vertices of the original polyline are within tolerance of the simplification results. This process is illustrated in Figure 4-1.

(1) Given three points $(x_p, y_p), (x_a, y_a), (x_b, y_b)$, show a detailed process to compute the perpendicular distance from p to line ab.

(2) Suppose we have an abstract data type (ADT) of **`Point`** that contains the x-axis and y-axis of a point. We also have a function **`pDistance`** that computes the perpendicular distance from p to line ab. The input of **`pDistance`** is (p, a, b), and the return value is a float value representing the distance. Fill in the function **`farthestPoint`** regarding how to locate the point with the maximum perpendicular distance to the starting point **`points[0]`** and the ending point **`points[-1]`**.

```
def pDistance(p, a, b): # p, a, b are of type Point
    # return the distance from p to line ab
```

```
def farthestPoint(points): # points is a list of type Point
    maxIdx = 1
    maxDist = 0
    for idx in range(1, len(points) - 1):
        dist = __ ①__(__②__, __③__, points[-1])
        if __④__ > __⑤__:
            maxIdx = idx
            maxDist = dist
    # returns the index and distance of the farthest point
    return maxIdx, maxDist
```

(3) Suppose the abstract data type of **`Point`** also contains a boolean value **`selected`** that indicates whether this point is selected. The following program is a recursive solution. Fill out the numbered blanks in the program and highlight your answers with the corresponding numbers.

```
# Assume all points are selected at the beginning
def DouglasPeuker_R(points, threshold):
    if len(points) <= __①__:
        return
    maxIdx, maxDist = farthestPoint(points)
    if __②__:
        DouglasPeuker_R(__③__, threshold)
        DouglasPeuker_R(__④__, threshold)
    else:
        for p in points[1:-1]:
            p.selected = __⑤__
        return
```

(4) The following program is a non-recursive solution. Fill out the numbered blanks in the program and highlight your answers with the corresponding numbers.

```
# Assume all points are not selected at the beginning
def DouglasPeuker_NR(points, threshold):
    stack = Stack() # initialize a stack
    # push a tuple indicating the index of the starting and ending point
    stack.push((0, __①__))
    while not __②__:
```

```
e = stack.pop()
if __③__:
    points[e[0]].selected = True
    points[e[1]].selected = True
else:
    maxIdx, maxDist = farthestPoint(points[__④__:__⑤__])
    maxIdx += e[0]
    if maxDist >= threshold:
        stack.push((__⑥__, maxIdx))
        stack.push((__⑦__, e[1]))
    else:
        points[e[0]].selected = True
        points[e[1]].selected = True
```