CMPE 224-343
Fall 2025
Programming Homework 2

This assignment is due by 23:55 on Sunday 9, November 2025.

You are welcome to ask your HW related questions. You should use only one of these options:

1. Moodle Homework **Question Forum**: HW Question-and-Answer (Q&A) Forum on Moodle is always available. Use the "Forum" link at the course Moodle page.
2. Homework **RECITATION HOURS**: There will be two Q&A RECITATION HOURs on the following days:

   - CMPE224-343 HW2 OfficeHour1: October 31 Friday, 14:30-16:00, Zoom ID: https://tedu.zoom.us/j/98807474943
   - CMPE224-343 HW2 OfficeHour2: November 3 Monday, 14:30-16:00, Zoom ID: https://tedu.zoom.us/j/93605289386

Additionally, the tutors will be able to help you understand the homework requirements. Please see the LMS page for their office hours.

Note: Please make sure that you have read the HW document well before participating. However, no HW related questions will be accepted except from the above options.

PROGRAMMING TASK

In this part, you must implement your own graph data structure by taking inspiration from your textbook and use it to help to solve problem. You are not allowed to use any external library or .jar file. Any solutions without using graph data structure are not evaluated!

Question 1(25 points):

You've been hired as the Conductor of a smart factory's assembly line orchestra. Each task is an instrument (a vertex), and a directed edge u → v means *"instrument u must finish its part before instrument v can begin."* Your mission is to produce a valid performance order that respects all "play-before" rules. If that's impossible because some instruments form a loop of dependencies (a directed cycle), you must point out the loop so the floor managers can fix the sheet music.

Goal:

- If the dependency graph is a **DAG**, compute a **valid topological order** of tasks.
- If the graph is **not** a DAG, **output one directed cycle** as a witness (list vertices in order around the cycle, starting and ending at the same vertex).

Hint:

- A classic approach is **DFS**: use **reverse postorder** for topological ordering, and **on-stack detection** to detect and reconstruct a cycle.

Input Format:

- The first line contains two space-separated integers: **N M**
    - **N**: number of tasks (vertices), labeled **0..N−1**
    - **M**: number of precedence constraints (directed edges)
- The next **M** lines each contain two space-separated integers **u v**, meaning **u → v** (*task u must complete before task v can start*).
- There are no self-loops or parallel edges guaranteed, but your program should be robust to handle or ignore them safely.

Output Format:

- If the tasks are schedulable (graph is a DAG), print:

```
Schedulable
Order: t0 t1 ... tN-1
```

where order is a valid topological order (any valid one is acceptable).

- If the tasks are not schedulable (a directed cycle exists), print:

```
Not schedulable
Cycle: v0 v1 v2 ... v0
```

where cycle lists one directed cycle as a witness. (Any rotation of the same cycle, or the same cycle reversed with directionally correct edges, is acceptable.)

Sample Input1:

```
6 6
0 2
0 3
1 3
1 4
3 5
4 5
```

Sample Output1:

```
Schedulable
Order: 0 1 2 3 4 5
```

Sample Input2:

```
4 4
0 1
1 2
2 0
2 3
```

Sample Output2:

```
Not schedulable
Cycle: 0 1 2 0
```

Question 2(25 points):

You are the network architect of a global intercloud. Each data center is a vertex, and a one-way communication link is a directed edge u → v meaning "messages can flow from u to v." Two data centers belong to the same communication zone if each can reach the other (i.e., they are in the same Strongly Connected Component, SCC). If you contract each SCC into a single node, you obtain the condensation DAG of the network.

Your task is to:

1. **Compute SCCs** of the digraph and output each component's members.
2. Conceptually **build the condensation DAG** (no need to print it).

3. If there is **more than one SCC**, compute the **minimum number of new directed links** needed to make the whole network **strongly connected**, and output **one valid set** of such links.

4. Use the **two-pass Kosajaru-Sharir algorithm** for SCC detection (DFS on the reversed graph to obtain reverse postorder, then DFS on the original graph in that order to label components.)

Known fact: In the condensation DAG, let #sources be the number of SCC nodes with no incoming edges, and #sinks be the number with no outgoing edges. If there is only one SCC, the answer is 0.

Input Format:

- First line: two space-separated integers **N M**
  - **N:** number of data centers (vertices), labeled **0..N−1**
  - **M**: number of directed links (edges)
- Next **M** lines: two space-separated integers **u v**, denoting a directed link **u → v**.
- Assume no self-loops or parallel edges in the intended inputs; your code should still behave robustly.

Output Format:

1. Print each **SCC** in increasing order of its **smallest vertex ID**:

```
Component i: v1 v2 ... vk
```

List the vertices of that component in ascending order.

2. If there is only one SCC:

```
Already strongly connected
New edges needed: 0
```

3. Otherwise, print:

```
New edges needed: K
add: a1 b1
add: a2 b2
...
add: aK bK
```

Each line add: x y describes a new directed link x → y to achieve strong connectivity. Any valid constructive set is acceptable (e.g., connect sinks to sources in a ring-like pattern).

Sample Input1:

```
8 8
0 1
1 2
2 0
2 3
3 4
4 5
5 3
6 7
```

Sample Output1:

```
Component 1: 0 1 2
Component 2: 3 4 5
Component 3: 6
Component 4: 7
New edges needed: 2
add: 5 6
add: 7 0
```

Sample Input2:

```
4 5
0 1
1 2
2 3
3 0
0 2
```

Sample Output2:

```
Component 1: 0 1 2 3
Already strongly connected
New edges needed: 0
```

- **You need to upload your code into VPL on LMS for each question.** If you do not upload your code into VPL on LMS, your homework will **not be graded.**
- The Java sources should be WELL DOCUMENTED as comments, as part of your grade will be based on the level of your comments.
- You need to upload **maximum-3 pages** PDF report document that explains your own answers for programming task in a clearly readable PA report format (refer to **PA REPORT FORMAT** section).

PA REPORT FORMAT

A programming assignment report is a self-description of a programming assignment and your solution. The report must not be hand-written. You may use a word processor or the on-line editor of your choice and prepare as a PDF document. The report must be grammatically correct and use complete English sentences. Each report should include the following sections, in the order given:

Information (%2.5): This section includes your ID, name, section, assignment number information properly.

Problem Statement and Code Design (%15): Include a brief summary of the problem and/or your sub-tasks to be completed in this assignment. You should show your modular design rationale by creating a structure chart that indicates your top-down, stepwise refinement of the problem solution. You may create the structure chart using available graphical tools like MS PowerPoint, SmartDraw etc.

Implementation and Functionality (%20): Since you have modular source code, you should describe each sub-module (program) in this section. Each sub-module should include names and types of any input/output parameters as well as the pseudocode algorithm that used for completing its task. By this way, you give meaning to each chart boxes from the previous section.

Testing (%7.5): You should provide a tester class that is able to identify key test points of your program. This class should be able to generate additional (apart from the given sample input/output) test data for the purpose of being clear on what aspects of the solution are being tested with each set. This section should also include a description of any program *bugs* that is, tests which has incorrect results. You should write these to describe your tests, summarize your results, and argue that they cover all types of program behavior.

Final Assessments (%5): In this final section, you should briefly answer the following questions:

1. What were the trouble points in completing this assignment?
2. Which parts were the most challenging for you?
3. What did you like about the assignment? What did you learn from it?
4. Did you use any Artificial Intelligence (AI) tools (e.g., ChatGPT, Copilot, Gemini, etc.) while completing this assignment?
    a. If yes, specify which tools you used and how you used them (e.g., to clarify a concept, check logic, debug, or generate comments).
    b. How did you ensure that your final submission reflects your own understanding and original work?
5. How did AI assistance (if used) affect your learning process?

a. Did it make problem-solving easier, or did it change how you approached the task?
b. What are the ethical boundaries you think should apply when using AI in academic work?

GRADING:

- Codes (%50: %25 for Q1 and %25 for Q2)
  - Available test cases evaluation on VPL: %15
  - Hidden test cases evaluation: %15
  - Approach to the problem: %20
- Report (%50: %25 for Q1 and %25 for Q2)
  - Information: %2.5
  - Problem Statement and Code design: %15
  - Implementation, Functionality: %20
  - Testing: %7.5
  - Final Assessments: %5

IMPORTANT

IMPORTANT NOTES: Do not start your homework before reading these notes!!!

1. This assignment is due by 23:59 on Sunday, November 9th.

2. You should upload your homework to LMS before the deadline. No hardcopy submission is needed. You should upload your codes into VPL and your report into submission place on LMS.

3. The standard rules about late homework submissions apply (20 points will be deducted for each late day). Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.

4. You ARE NOT ALLOWED to modify the given method names. However, if necessary, you may define additional data members and member functions.

5. Your classes' name MUST BE as shown in the homework description.

6. The submissions that do not obey these rules will not be graded.

7. To increase the efficiency of the grading process as well as the readability of your code, you have to follow the following instructions about the format and general layout of your program.

8. Do not forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of your Java files. Example:

```
//-------------------------------------------------------
// Title: Scheduler tester class
// Author: Name/Surname
// ID: 2100000000
// Section: 1
// Assignment: 1
// Description: This class tests the …
//-------------------------------------------------------
```

9. Since your codes will be checked without your observation, you should report everything about your implementation. Add detailed comments to your classes, functions, declarations etc. Make sure that you explain each function in the beginning of your function structure. Example:

```
void setVariable(char varName, int varValue)
//-----------------------------------------------------------
// Summary: Assigns a value to the variable whose
// name is given.
// Precondition: varName is a char and varValue is an
// integer
// Postcondition: The value of the variable is set.
//-----------------------------------------------------------
{
      // Body of the function
}
```

10. Indentation, indentation, indentation...

11. This homework will be graded by your TA, Deniz Merve Uzun. Thus, you may ask her your homework related questions through HW forum on Moodle course page. You are also welcome to ask your course tutors Alperen Karadağ, Aysel Arpacı or Mert Temür for help.