

Data Structure and Algorithm, Spring 2025

Mini-Homework D

TA E-mail: dsa.ta@csie.ntu.edu.tw

Due: 13:00:00, Tuesday, April 15, 2025

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources (including chatGPT) can be consulted, but not copied from.
- Since everyone needs to write the final solutions alone, there is **absolutely no need to lend your homework solutions and/or source codes to your classmates at any time**. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- In each mini-homework, we will have one programming problem that is closely connected to what we have taught in the class. The purpose of the problem is for you to *practice* what you have learned, before conquering the more challenging problems in other homework sets.
- Like other programming problems, you should write your code in C programming language, and then submit the code via the *DSA Judge*:

<https://dsa-2025.csie.org/>.

Each day, you can submit up to 5 times for each problem. The judge system will compile your code with

```
gcc [id].c -static -O2 -std=c11
```

- For debugging in the programming part, we strongly suggest you produce your own test-cases with programs and/or use tools like `gdb` or compile with flag `-fsanitize=address` to help you solve issues like illegal memory usage. For `gdb`, you are strongly encouraged to use compile flag `-g` to preserve symbols after compiling.

Note: For students who use IDE (VScode, DevC++...) you can modify the compile command in the settings/preference section. Below are some examples:

- `gcc [id].c -O2 -std=c11 -fsanitize=address # Works on Unix-like OS`
- `gcc [id].c -O2 -std=c11 -g # use it with gdb`

- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$Late\ Score = \max \left(\left(\frac{5 \times 86400 - Delay\ Time(sec.)}{5 \times 86400} \right) \times Original\ Score, 0 \right)$$

Please note that the “gold medal” of the class cannot be used on mini-homework problems.

- If you have questions about this mini-homework, please go to the Discord channel and discuss the issues with your classmates. This is *strongly preferred* because it will provide everyone with a more interactive learning experience. If you really need email clarifications, you are encouraged to include the following tag in the subject of your email:
 - The subject should contain one tag, "[hwd]", specifying the problem where you have questions. For example, "[hwd] Will there be a repeated number?". Adding these tags potentially allows the TAs to track the status of each email and to provide faster responses to you.
 - If you want to provide your code segments to us as part of your question, please upload it to [Gist](#) or similar platforms and provide the link. Please remember to protect your uploaded materials with proper permission settings. Screenshots or code segments directly included in the email are discouraged and may not be reviewed.

Cycle Detection (15 pts)

Problem Description

Given a “linked list” where each node contains $\{data, next\}$ and $next$ points to another node, run the Floyd Cycle Detection algorithm below to detect whether there is a reachable cycle from the $head$. You need to print out all the data within the nodes that the hare visited before (including) meeting the tortoise.

Note that there are many different variant versions of the Floyd Cycle Detection algorithm. You are asked to implement the version below *exactly* to produce the correct answer for this problem.

FLOYD-CYCLE-DETECTION($head$)

```
1  hare = head
2  tortoise = head
3  while hare  $\neq$  NIL and hare.next  $\neq$  NIL
4      hare = hare.next.next
5      tortoise = tortoise.next
6      if hare = tortoise
7          return TRUE
8  return FALSE
```

Input

The first line includes one integer N , representing the number of nodes. The n -th line of the next N lines contains two integers, which means $data$ and $next$ for node n . We assume that $head$ initially points to node 1, and a $next$ of 0 means connecting to NIL.

Output

- If there is a reachable cycle from the $head$, output a line of

[head.data] [head.next.next.data] [head.next.next.next.next.data] ... [finish.data]

where **finish** is the location where the hare and tortoise meet.

- If there is no reachable cycle from the $head$, output a line of

[head.data] [head.next.next.data] [head.next.next.next.next.data] ... [goodbye.data]

where **goodbye** is the last node that the hare jumps into before leaving the linked list from the $head$.

Constraint

- $1 \leq N \leq 10^5$
- $-2^{30} \leq data \leq 2^{30}$
- $0 \leq next \leq N$

Sample Testcases

Sample Input 1

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

Sample Output 1

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

Sample Input 2

```
3
11 2
26 2
64 1
```

Sample Output 2

```
11 26
```

Sample Input 3

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

Sample Output 3

```
1 3
```

Hint

- By design, you can pass this homework by simulating the Floyd Cycle Detection algorithm properly. There is no need for other arithmetic calculations or cuts.