Assignment 1
Looney Troons - Train (Troon) Network Simulation using Parallel Programming

CS3210 – 2022/23 Semester 1

Learning Outcomes

This assignment is designed to enhance your understanding of parallel programming with shared-memory (OpenMP). You will apply synchronization constructs you learned in class to simulate a simplified train network.

1 Problem Scenario

1.1 Simulation Rules

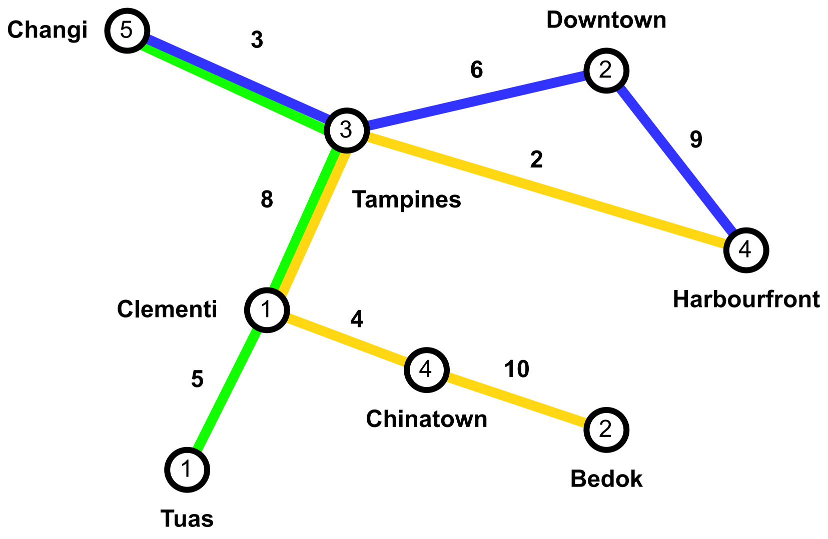

Figure 1: Example Troon Network

The above shown graph in Figure 1 is a hypothetical troon (train) network in Singapore (MRT network). The nodes in the graph are *stations* and edges are *links* connecting stations. The weight of each edge represents the length of the link (in km) they are associated with and the weight of each node represents the popularity of the troon station. Assume each link consists of two parallel uni-directional links dedicated for travelling in opposite directions.

There are three troon services operating in this network: Green Line, Yellow Line and Blue Line throughout the day in both directions. The stations serviced by each troon service are as follows:

- **Green:** Tuas → Clementi → Tampines → Changi (and reverse direction)
- **Yellow:** Bedok → Chinatown → Clementi → Tampines → Harbourfront (and reverse direction)
- **Blue:** Changi → Tampines → Downtown → Harbourfront (and reverse direction)

As illustrated in the graph, some links (edges) are shared by multiple troon services. For example, the link between Clementi and Tampines stations is shared by both Green Line and Yellow Line. However, there can only be a maximum of one troon on a given link in one direction between two stations at any given time.

When a troon reaches a *terminal station* - a station at either end of a line - it reverses direction. In this way every troon, once spawned, will "bounce" between the terminal stations of its line until the end of the simulation.

1.1.1 Troon IDs and Spawning

When the simulation starts, troons will be spawned into the network according to the following rules:

- Troons are given integer IDs corresponding to the order in which they were spawned, starting from 0 and incrementing by 1.
- The first tick is 0.
- On each tick beginning from the first tick, at most one troon is spawned per terminal station in each line. That is, since there are always exactly 3 lines, there will be at most 6 troons (3 lines times 2 terminal stations) spawned every tick until all troons have been spawned.
- Within a tick, we first spawn troons on the green line, then on the yellow line, and finally on the blue line.
- Within a line, priority for spawning is given to the first station in the line (rather than the last station in the line).
- All troons spawn as if they just arrived at a station. That is, they need to open their doors and load passengers before they can leave, and follow all normal queuing rules at a station. See the next section for details.

In the given example troon network, if we were to spawn 3 troons per line (total 9 troons), then in tick 0 we spawn 6 troons in exactly this order:

- Troon 0: Tuas (green line; going forward)
- Troon 1: Changi (green line; going backward)
- Troon 2: Bedok (yellow line; going forward)
- Troon 3: Harbourfront (yellow line; going backward)
- Troon 4: Changi (blue line; going forward)
- Troon 5: Harbourfront (blue line; going backward)

In tick 1, we spawn 3 troons in exactly this order:

- Troon 6: Tuas (green line; going forward)
- Troon 7: Bedok (yellow line; going forward)
- Troon 8: Changi (blue line; going forward)

1.1.2 How troons transit from station to station

When a troon arrives at a station (possibly by spawning), it needs to load/unload passengers before it can leave for the next station. A troon may only load/unload passengers at a station platform. A station has exactly *one platform per outgoing link*, each with a corresponding holding area. In the given example network, Changi has just one platform (for passengers heading to Tampines) but Tampines has four platforms (for passengers heading to Changi, Clementi, Downtown or Harbourfront). See another example in Figure 2. Each station platform only accommodates up to one troon; all other troons waiting to use the same link must wait in the platform's holding area.

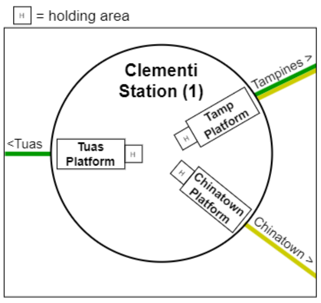

Figure 2: Platforms

Upon arriving at a station, a troon goes to the platform leading to the next link it needs to transit to if that platform is free. If the platform is not free, the troon goes to queue up in the corresponding holding area. If more than 1 troon arrives at the same station at the same tick and wish to use the same link, they queue up in ascending order of troon ID (i.e. lower IDs go before higher IDs for the same arrival time). Every holding area can hold an unlimited number of waiting troons. Only troons at a platform can open or close their doors; troons in a holding area can only wait until the troon currently at the platform leaves (Figure 4), at which point the troon at the front of the holding area moves to the platform in 0 ticks (instantly). Once a troon arrives at a platform, it spends 1 tick opening its doors. Then, it loads passengers and closes its doors, taking a number of ticks equal to the popularity of the station it is at (Figure 3). Once its doors are closed, it waits to transit on the link. If the link is free, the troon waits 1 tick before transiting. Otherwise, it waits until the link is free then transits onto the link in the following tick (Figure 4).

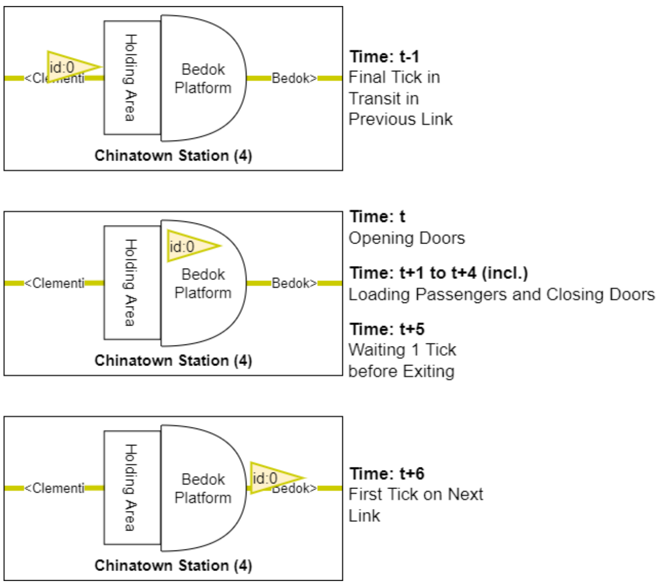

Figure 3: Train in station

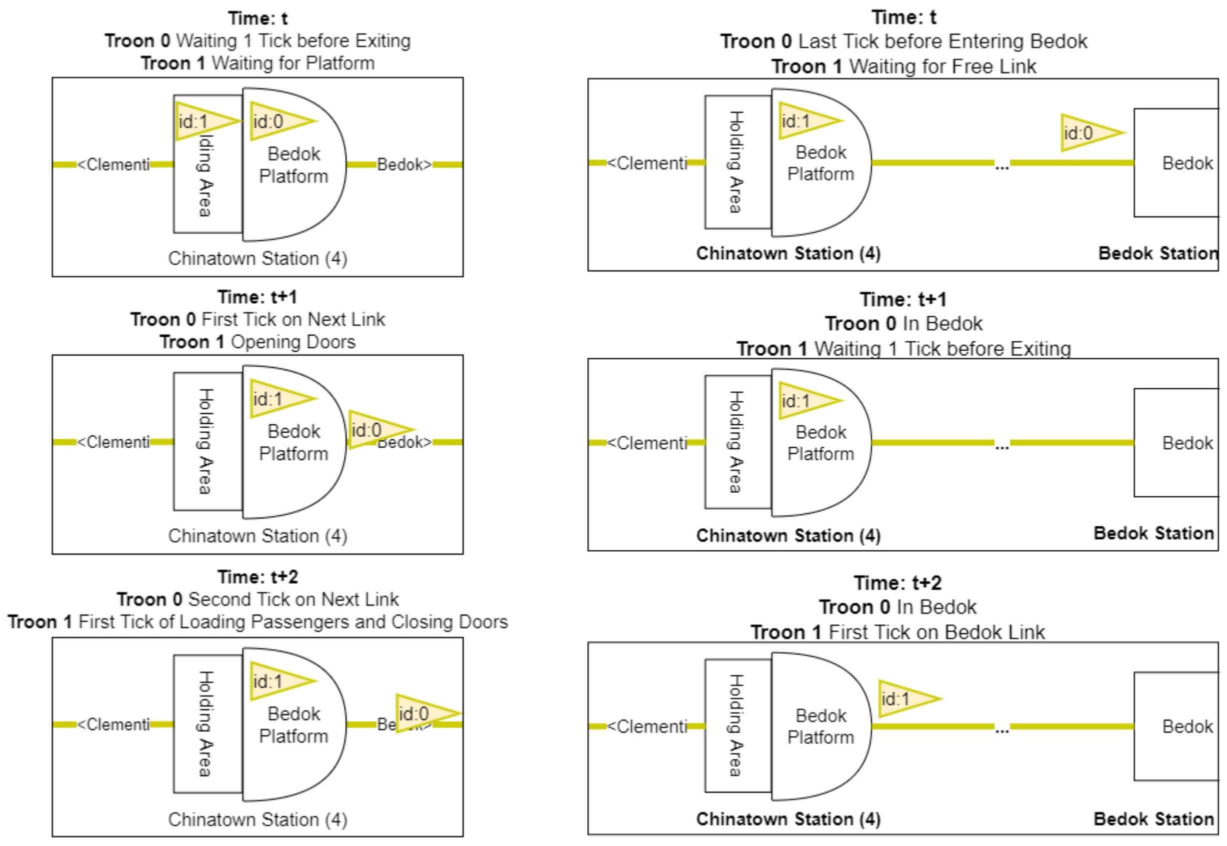
Time: t
Troon 0 Waiting 1 Tick before Exiting
Troon 1 Waiting for Platform
<Clementi
Holding Area
id:1
id:0
Bedok Platform
Bedok>
Chinatown Station (4)
Time: t+1
Troon 0 First Tick on Next Link
Troon 1 Opening Doors
Time: t+2
Troon 0 Second Tick on Next Link
Troon 1 First Tick of Loading Passengers and Closing Doors
Time: t
Troon 0 Last Tick before Entering Bedok
Troon 1 Waiting for Free Link
Bedok
Bedok Station
Time: t+1
Troon 0 In Bedok
Troon 1 Waiting 1 Tick before Exiting
Time: t+2
Troon 0 In Bedok
Troon 1 First Tick on Bedok Link

Figure 4: Waiting to enter the station (left) and waiting to leave the station(right)

A troon transiting on a link remains on that link for a number of ticks equal to the distance of the link. That is, every troon transits at a constant speed of 1km/tick with instant acceleration/deceleration. After transiting, a troon arrives at its destination station and instantly goes to either a station platform or to a holding area. As mentioned before, a troon will need 1 tick to open its doors upon arrival at a platform.

A summary of a troon's lifecycle is as follows:

1. Arriving at a station (upon spawning or transiting)
    - If needed, waiting to enter platform until the platform's corresponding holding area becomes available.
    - Opening doors at platform (for 1 tick)
2. Loading passengers and closing doors
    - Only when at a station platform
    - For as many ticks as the station's popularity
3. Waiting to leave station
    - Only when at a station platform
    - If there is a troon on the link, then this troon waits until the tick after that troon arrives at the next station
    - If there is no troon on the link, then this troon waits 1 tick before transiting on the link

4. In transit on a link
    - Only when on a link
    - For as many ticks as the link's distance

1.1.3 Terminal Stations

When a troon arrives at a terminal station (a station at the end of a line), it goes to the platform (or holding area) corresponding to the link that will take it in the opposite direction of the same line. In the given example, when a troon on the blue line which originally started its journey from Harbourfront reaches Changi, it changes its direction by queuing to use the link from Changi to Tampines. After Tampines, it will head to Downtown then Harbourfront then back to Downtown and so on. These troons are automated, never break down and never change lines, so once they are placed on the troon network they will continue "bouncing" between their line's terminal stations until the simulation ends.

1.1.4 End of the simulation

The simulation ends after a positive number of ticks (taken as input) have been simulated. The first tick is 0, so if the simulation is for 15 ticks, then the simulation should run from tick 0 to tick 14 inclusive then end.

1.1.5 Hints

Two possible ways (non-exhaustive) to model the troon network are as follows:

1. One thread per troon
2. One thread per link in one direction

You are free to choose either approach, or come up with your own way to model the simulation.

1.2 Inputs and Outputs

1.2.1 Input Format

An input file strictly follows the structure below:

1. S - number of troon stations in the network.
2. L - a single line containing the space-separated list of stations. No station name should contain whitespace.
3. P - a single line containing the space-separated popularity of each station. The order of the stations follows L. Each popularity value must be a non-negative integer.
4. M - S consecutive lines of a 2-dimensional, symmetrical adjacency matrix representing the graph. The ordering of the columns and rows of M follows the ordering of L. Each link length must be a non-negative integer.
5. G - the list of stations serviced by the green line. The order of the stations from left-to-right should be taken as the forward direction of the line.
6. Y - the list of stations serviced by the yellow line. The order of the stations from left-to-right should be taken as the forward direction of the line.
7. B - the list of stations serviced by the blue line. The order of the stations from left-to-right should be taken as the forward direction of the line.

8. N - number of ticks to run for the simulation. Must be a non-negative integer.
9. g, y, b - number of troons to spawn per line (green, yellow, blue). Must be a non-negative integer.
10. NUM_LINES - number of ticks (from the end) to output the simulation state. E.g. if $NUM\_LINES = 5$, then the program should output the simulation states for the last 5 ticks. $1 \leq NUM\_LINES \leq N$

1.2.2 Example Input

The following input uses the graph depicted in Figure 1.

Example Input

```
8
changi tampines clementi downtown chinatown harborfront bedok tuas
5 3 1 2 4 4 2 1
0 3 0 0 0 0 0 0
3 0 8 6 0 2 0 0
0 8 0 0 4 0 0 5
0 6 0 0 0 9 0 0
0 0 4 0 0 0 10 0
0 2 0 9 0 0 0 0
0 0 0 0 10 0 0 0
0 0 5 0 0 0 0 0
tuas clementi tampines changi
bedok chinatown clementi tampines harborfront
changi tampines downtown harborfront
20
2 2 2
5
```

1.2.3 Output Format

For each of the last NUM_LINES ticks of the simulation, output a line following this given structure:

1. The tick number (0-based) of the simulation, followed by a colon and space character.
2. The position of every troon, separated by a space, according to the following rules
   (a) Each troon is prefixed by the line it is on, e.g. `b` for blue line, `g` for green, and `y` for yellow. This is followed by the troon's ID, and a hyphen character.
   (b) If a troon is in transit, it should output the source station name, followed by "`->`", and the destination station's name.
   (c) Otherwise, simply output the station name the troon is at.
   (d) The positions should be sorted lexicographically (refer to the sample output).

1.2.4 Example Output

Note that some lines may have wrapped in the following output. If the line does not begin with a tick number followed by a colon, then it belongs to the previous line.

Example Output

```
15:  b4-changi->tampines b5-downtown g0-clementi->tampines g1-tampines
y2-chinatown y3-tampines->clementi
16:  b4-changi->tampines b5-downtown g0-clementi->tampines g1-tampines
y2-chinatown y3-tampines->clementi
17:  b4-tampines b5-downtown g0-clementi->tampines g1-tampines
y2-chinatown y3-tampines->clementi
18:  b4-tampines b5-downtown g0-clementi->tampines g1-tampines
y2-chinatown y3-tampines->clementi
19:  b4-tampines b5-downtown->tampines g0-tampines g1-tampines
y2-chinatown y3-tampines->clementi
```

2 Grading

You are advised to work in groups of two students for this assignment (but you are allowed to work independently as well). You may discuss the assignment with others but in the case of plagiarism, both parties will be severely penalized. Cite your references or at least mention them in your report (what you referenced, where it came from, how much you referenced, etc.).
The grades are divided as follows:

- 8 marks – OpenMP implementation and a `Makefile` that builds your implementation when calling `make submission`, and emits it as an executable named `troons`
- 4 marks – Assignment report; mark breakdown detailed in section 2.2.2

2.1 Program Requirements

Your OpenMP implementation should:

- Be written in C or C++
- Give exactly the same result (output) as the provided sequential implementation (only an executable is provided)
- Have no memory leaks
- Run faster than our sequential implementation on any lab machine. Specifically, to obtain full marks for the performance component, your OpenMP implementation should achieve a speedup greater than 1x when running with at least 8 threads (on a machine with at least 8 threads), for an input with roughly 17,000 stations and 200,000 trains per line running for 100,000 ticks, and requiring output for only the last 5 ticks (i.e. $NUM\_LINES = 5$).

2.1.1 Correctness

The starter code includes an executable (no source code) of a correct sequential implementation for you to test the correctness of your implementation(s). When grading, we will test for correctness by diffing the outputs of our implementation and your implementation.

We have tested our sequential implementation but as always it is possibly incorrect. If you notice discrepancies between the sequential implementation and the simulation rules, please do let us know. Our contact details are available in section 3.1.

2.2 Report Requirements

2.2.1 Format

Your report should follow these specifications:

- Four pages maximum for main content (excluding appendix).
- All text in your report should be no smaller than 11-point Arial (any typeface and size is ok so long as it's readable English and not trying to bypass the page limit).
- All page margins (top, bottom, left, right) should be at least 1 inch (2.54cm).
- Have visually distinct headers for each content item in section 2.2.2.
- It should be self-contained. If you write part of your report somewhere else and reference that in your submitted "report", we reserve the right to ignore any content outside the submitted document. An exception is referencing a document containing measurement data that you created as part of the assignment - we encourage you to do this.
- If headers, spacing or diagrams cause your report to *slightly* exceed the page limit, that's ok - we prefer well-organized, easily readable reports.

2.2.2 Content

Your report should contain:

- (2 marks) A brief description of:

  1. your program's parallelization strategy
  2. how link contention is resolved in your implementation
  3. the key data structures and synchronization constructs used to implement this strategy

  Diagrams are not required but may help you explain something clearly without taking much space.
- (1 mark) Answers to the following questions :

  1. Which aspect of the input size affects speedup the most for your implementation?
  2. Why do you think this is the case?

  Take measurements to support your answers. You may vary the input size (number of stations, number of links, number of trains, number of ticks) and use different number of cores and different lab machines to run your implementation.
- (1 mark) A description of ONE performance optimization you attempted (if any) with analysis and supporting performance measurements. Refer to an optimization related to your parallelization efforts. Include at least one summary graph in your report and link to your supporting measurements (for example, a .csv file in the repository or a Google sheet).

Additionally, your report should have an appendix (does not count towards page limit) containing:

- Details on how to reproduce your results, e.g. inputs, execution time measurement, etc.
- A list of lab machine nodes you used for testing and performance measurements.
- Relevant performance measurements, if you don't want to link to an external document like Google Sheets.

Tips:

- There could be many variables that contribute to performance, and studying every combination could be highly impractical and time-consuming. You will be graded more on the quality of your investigations, not so much on the quantity of things tried or even whether your hypothesis turned out to be correct.
- Performance analysis may take longer than expected and/or run into unexpected obstacles (like your program failing halfway). Start early and test selectively.
- The lab machines are shared with the entire class. Please be considerate and do not hog the machines or leave bad programs running indefinitely. Again, start early or you may be contending with everyone else. Use Slurm to get accurate results in your performance measurements.

2.3 Bonus - Speedup Contest

You may obtain up to 2 bonus marks for achieving the best speedup among all submissions in the class on a lab machine of either of these types:

- **Xeon Silver 4114 machine**
- **Intel Core i7-7700 machine**

We chose only these two types because they are the most common configurations among the lab machines.

2.3.1 Bonus Mark Distribution

Each student in the two teams that achieve the best speedup on either machine type will receive 2 bonus marks. If a team's implementation tops on both machine types, that team's implementation will be taken as the winner for the Intel Core i7-7700 type, and the next best team for the Xeon Silver 4114 type will be taken as that type's winner.

2.3.2 Participating in the Contest

To participate in the contest, you must do two things. Firstly, you must have a maximum 2-page section at the end of your report (before the appendix) entitled "Bonus" explaining the approach and most important optimizations you implemented for your bonus submission. This section does not count towards the main report page count.

Secondly, you must create a makefile recipe named `bonus` such that when `make bonus` is run on any lab machine, an executable named `troons_bonus` is generated. That executable must:

1. follow the same correctness specification as your submission build
2. run on both lab machine types involved in the contest
3. be written in C or C++
4. not communicate with any other machine (i.e. no cluster or cloud computing)

Other than that, you can do whatever you think best: your implementation could use pthreads, processes, or even be sequential. You may use external libraries or frameworks (and request installation of these) but it is **your responsibility** to ensure that your submission can build and run on *any* relevant lab machine by the assignment deadline.

If any requirement above is not met, your submission may be disqualified from the contest.

3 Admin

3.1 FAQ

Frequently asked questions (FAQ) received from students for this assignment will be answered here. The most recent questions will be added at the beginning of the file, preceded by the date label. **Check this file before asking your questions**.
If there are any questions regarding the assignment, please use the Discussion Section on Canvas or email Benedict (benedictkhoo.mw@u.nus.edu) or Jing Yen (ljy@u.nus.edu).

3.2 Github Classroom and Starter Code

We will use Github Classroom to provide starter code and for submission. Use this link to get started: https://classroom.github.com/a/puJ8esQ9

The first member of the team to access this link will be prompted to accept the assignment that gives your team access to the new repository. Create a new team by typing `A1-A0123456Z` or `A1-A0123456Z_A0173456T`, using the student numbers of the students forming a team. Note that the naming convention must be followed strictly, e.g. capitalisation, dash, and spacing. Use the same name for your report file (e.g. `A1-A0123456Z_A0173456T.pdf`) for easy identification.

The other member in the team will be able to see an existing team with your team id under "Join an existing team" section.

The repository is automatically populated with the starter code provided for this assignment. See the README in your team's GitHub repository for details about the starter code.

3.3 Accessing the Lab Machines

Please refer to Lab 01 and Tutorial 01 sheets for the list of lab machines, and instructions on how to connect to them. You may also wish to use Slurm to schedule jobs when testing correctness and taking performance measurements. A Slurm guide can be found at https://bit.ly/cs3210-slurm-guide.

Before using the SoC compute services (i.e. computer cluster nodes, login to Sunfire, etc), you should enable SoC Computer Cluster from your MySoC Account page.
To access the lab machines, ssh into Sunfire using the command below. Enter your SoC account and password when prompted.

```
> ssh your_soc_account_id@sunfire.comp.nus.edu.sg
```

Next, from Sunfire, ssh into the lab machines using the command below. Enter your account and password published in LumiNUS Gradebook when prompted.

```
> ssh your_account_id@hostname ('hostname' might be 'soctf-pdc-004.comp.nus.edu.sg')
```

Once entered, you should be looking at the shell of the remote lab machine, and you will be placed at the home directory of your account. Note that you should use your SoC account and password to access Sunfire, but use your NUSNET id and password published under LumiNUS Gradebook to login to the lab machines.
To login without inputting a password, configure SSH key-based authentication. Use ssh-keygen and ssh-copy-id to copy the public key to the remote node.
List of machines:

```
- soctf-pdc-001 - soctf-pdc-008:  Xeon Silver 4114 (10 cores, 20 threads)
- soctf-pdc-009 - soctf-pdc-016:  Intel Core i7-7700 (4 cores, 8 threads)
- soctf-pdc-018 - soctf-pdc-019:  Dual-socket Xeon Silver 4114 (2*10 cores, 40 threads)
- soctf-pdc-020 - soctf-pdc-021:  Intel Core i7-9700 (8 cores, 8 threads)
- soctf-pdc-022 - soctf-pdc-023:  Intel Xeon W-2245 (8 cores, 16 threads)
```

To view the usage of the lab machines, you can use this Telegram bot: `@cs3210_machine_bot`. Simply type `/start` to get a real time status update for all machines.
You may request for installation of new tools and compilers on the lab machines. During development you might use your personal computer or the computers from the SoC Compute Cluster reserved for CS3210. Their hostnames are: `xcne0`, `xcne1`, and `xcne2`.

3.4 Deadline and Submission

Assignment submission is due on **Wed, 21 Sep, 2pm**. The implementation and report must be submitted through your Github Classroom repository. The **commit hash** that you intend to use for submission must be submitted in the text entry for Assignment 1 in Canvas.

- Push your **code and report** to your team's Github Classroom repository.
- Your report must be a PDF named `<team_name>.pdf`. For example, `A1-A0123456Z_A0173456T.pdf`. `<team_name>` should **exactly match** your team's name; if you are working in a pair, please DO NOT flip the order of your admin numbers.
- Submit the **commit hash** you want us to grade (need not be the latest one in your repo) using the text entry box for Assignment 1 on Canvas. **Failing to submit this commit hash would be considered incomplete submission** (penalties apply), and we will grade the latest commit found in your repository dated before the deadline of the assignment.
- The time of latest assignment attempt (including late submissions) **on Canvas** will be taken as assignment submission time. Multiple attempts are allowed if re-submission is needed. Note that for submissions made as a group, only the most recent submission (from any of the students) will be graded, and both students receive that grade.

  A penalty of 5% per day (out of your grade) will be applied for late submissions.

4 Sample Inputs and Outputs with Explanation

This section provides a few sample inputs and outputs with a tick-by-tick explanation of the simulation. The sample inputs correspond to inputs provided in the starter code.

Sample Input 1

```
3
changi tampines clementi
1 2 1
0 2 0
2 0 1
0 1 0
changi tampines clementi
changi tampines clementi
changi tampines clementi
15
1 0 0
15
```

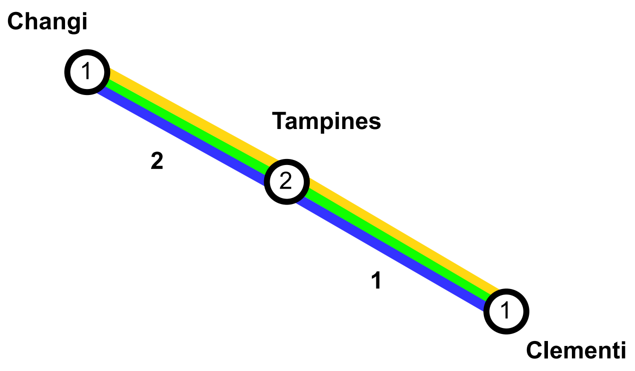

Figure 5: Sample input 1 troon network

Sample Output 1

```
0:  g0-changi
1:  g0-changi
2:  g0-changi
3:  g0-changi->tampines
4:  g0-changi->tampines
5:  g0-tampines
6:  g0-tampines
7:  g0-tampines
8:  g0-tampines
9:  g0-tampines->clementi
10:  g0-clementi
11:  g0-clementi
12:  g0-clementi
13:  g0-clementi->tampines
14:  g0-tampines
```

Table 1: Explanation of Sample Output 1

Tick	Explanation
0	Troon 0 is spawned at changi (start of green line) and spends this tick opening its doors
1	Troon 0 has opened doors and is loading passengers for 1 tick (popularity of changi is 1)
2	Troon 0 has closed doors and is waiting to leave. No troons on the link to tampines so it waits for 1 tick
3	Troon 0 is in transit on the link from changi to tampines (1st tick; total travel time is 2 ticks)
4	Troon 0 continues on the link (2nd tick)
5	Troon 0 arrives at the platform in tampines leading to clementi (no other troon there) and spends this tick opening its doors
6	Troon 0 has opened doors and is loading passengers for 2 ticks (popularity of tampines is 2)
7	Troon 0 continues loading (2nd tick)
8	Troon 0 has closed doors and is waiting to leave. No troons on the link to clementi so it waits for 1 tick
9	Troon 0 is in transit on the link from tampines to clementi (1st tick; total travel time is 1 tick)
10	Troon 0 arrives at the platform in clementi leading to tampines (no other troon there; also, clementi is the final station in the green line so the troon wants to reverse and go back to tampines) and spends this tick opening its doors
11	Troon 0 has opened doors and is loading passengers for 1 tick (popularity of clementi is 1)
12	Troon 0 has closed doors and is waiting to leave. No troons on the link to tampines so it waits for 1 tick
13	Troon 0 is in transit on the link from clementi to tampines (1st tick; total travel time is 1 tick)
14	Troon 0 arrives at the platform in tampines leading to changi (no other troon there) and spends this tick opening its doors

Sample Input 2

```
5
changi tampines clementi downtown harbourfront
1 2 1 1 1
0 4 0 0 0
4 0 1 1 2
0 1 0 0 0
0 1 0 0 0
0 2 0 0 0
changi tampines clementi
clementi tampines harbourfront
changi tampines downtown
9
3 1 1
9
```

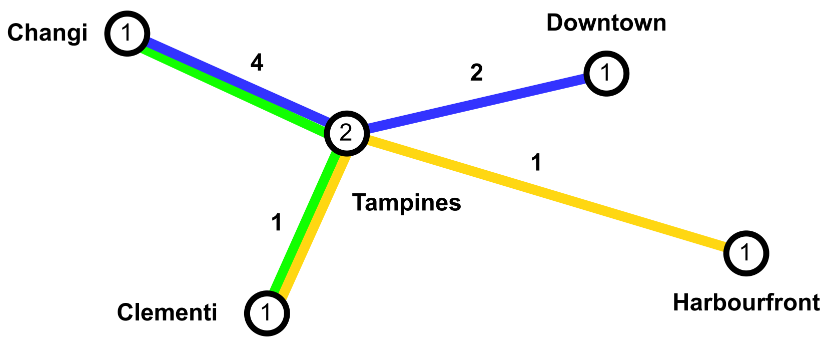

Figure 6: Sample input 2 troon network

Sample Output 2

```
0:  b3-changi g0-changi g1-clementi y2-clementi
1:  b3-changi g0-changi g1-clementi g4-changi y2-clementi
2:  b3-changi g0-changi g1-clementi g4-changi y2-clementi
3:  b3-changi g0-changi->tampines g1-clementi->tampines g4-changi
y2-clementi
4:  b3-changi g0-changi->tampines g1-tampines g4-changi y2-clementi
5:  b3-changi g0-changi->tampines g1-tampines g4-changi y2-clementi
6:  b3-changi g0-changi->tampines g1-tampines g4-changi
y2-clementi->tampines
7:  b3-changi g0-tampines g1-tampines g4-changi y2-tampines
8:  b3-changi->tampines g0-tampines g1-tampines->changi g4-changi
y2-tampines
```

Table 2: Explanation of Sample Output 2

Tick	Explanation
0	4 troons spawn. Troon 0 is green going forward; troon 1 is green going backward; troon 2 is yellow going forward; troon 3 is blue going forward. The third green troon does not spawn yet because at most 1 troon per terminal per line spawns in a tick. Troons 0 and 3 both want to use the link from changi to tampines and they both arrived (spawned) in the same tick. But troon 0 has a lower ID than troon 3 so it goes to the platform and spends this tick opening its doors and troon 3 goes to the holding area for that platform. For the same reason, troon 1 goes to the platform from clementi to tampines and spends this tick opening its doors while troon 2 goes to the corresponding holding area.
1	The last green troon (troon 4) spawns at changi (going forward) and queues behind troon 3 in the holding area for the link headed to tampines. Troons 0 and 1 load passengers. Troons 2 and 3 simply wait in holding areas.
2	Troons 0 and 1 close their doors and spend the tick preparing to leave (respective links are free). Troons 2, 3 and 4 continue waiting.
3	Troons 0 begins transiting to tampines (4 ticks left). Troon 1 begins transiting to tampines (1 tick left). Troons 2 and 3 instantly move to the now-free station platforms and spend the tick opening their doors. Troon 4 waits in the holding area behind troon 3.
4	Troon 0 is still in transit (3 ticks left). Troon 1 has finished transiting and arrives at the station platform from tampines to changi. It spends this tick opening its doors. Troons 2 and 3 begin loading passengers (both 1 tick left). Troon 4 continues waiting behind troon 3.
5	Troon 0 is still in transit (2 ticks left). Troon 1 begins loading passengers (2 ticks left). Troons 2 and 3 have closed their doors and are waiting to leave. Troon 4 continues waiting.
6	Troon 0 is still in transit (1 tick left). Troon 1 is still loading passengers (1 tick left). Troon 2's link is free so it begins transiting from clementi to tampines (1 tick left). Troon 3 is ready to leave but is waiting for troon 0 to finish transiting. Troon 4 continues waiting behind troon 3.
7	Troon 0 arrives at tampines, heads to the platform to clementi, and spends the tick opening doors. Troon 1 is done loading passengers and waits to leave. Troon 2 arrives at tampines, heads to the platform to harbourfront, and spends the tick opening doors. Troon 3 has to wait until the tick *after* its link becomes free (troon 0 just exited the link). Troon 4 continues waiting behind troon 3.
8	Troon 0 begins loading passengers (2 ticks left). Troon 1 begins transiting to changi (4 ticks left). Troon 2 begins loading passengers (2 ticks left). Troon 3 begins transiting to tampines (4 ticks left). Troon 4 instantly moves to the now-free station platform and spends the tick opening its doors.