

Assignment 3

Simulation of MRT Network in MPI

CS3210 – 2024/25 Semester 1

Learning Outcomes

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

Contents

1 Problem Scenario	2
1.1 Simulation Rules	2
2 Inputs and Outputs	8
2.1 Starter Code	8
2.2 Input Format	8
2.3 Output Format	10
3 Grading	11
3.1 Program Requirements	11
3.2 Report	12
4 Admin	14
4.1 Connecting to the Lab Machines	14
4.2 FAQ	14
4.3 Deadline and Submission	14
5 Bonus – Train Breakdowns	15
5.1 Requirement 1: Variable Number of Lines	15
5.2 Requirement 2: Faults	16
5.3 Bonus Submission	17
A Sample Inputs and Outputs with Explanation	18
B Generation of Loading/Unloading Times	23

1 Problem Scenario

Singapore's Mass Rapid Transit, commonly known as the MRT, is one of the most comprehensive rapid transit networks in Southeast Asia. This rail service forms the backbone of the country's public transportation system with a daily ridership of more than 3 million people. This requires the operators of the MRT system to have robust management of the trains in the network, as any faults or breakdowns in the train system could result in major inconveniences across the country.

In this assignment, you will be **simulating a simplified rapid transit network like that of the MRT network using MPI**. The goal for this assignment is to correctly and efficiently simulate the network with the communication and synchronization constructs of MPI.

1.1 Simulation Rules

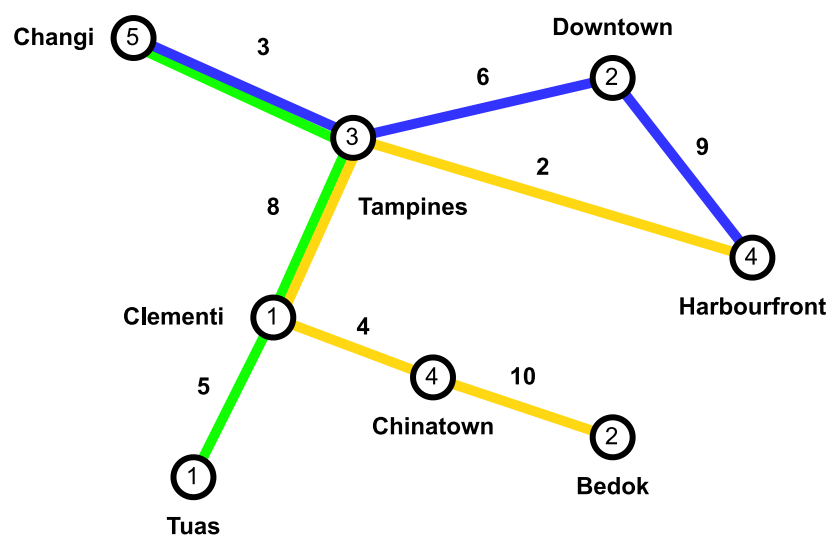


Figure 1: Example train Network

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

1.1.1 Lines and Links

Three lines are operating in this network: Green Line, Yellow Line, and Blue Line throughout the day in both directions. The stations serviced by each line 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 lines. 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 train on a given link in one direction between two stations at any given time.**

In the base version of this assignment (i.e. excluding the bonus), there will always be three lines; the *first* line will always be the Green Line; the *second* line will always be the Yellow Line; and the *third* line will always be the Blue Line.

1.1.2 Train IDs and Spawning

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

- Trains 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 train is spawned per terminal station in each line. That is, since there are always exactly 3 lines, there will be at most 6 trains (3 lines times 2 terminal stations) spawned every tick until all trains have been spawned.
- In a tick, trains are first spawned 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).
- All trains spawn as if they just arrived at a station. That is, they need to unload 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 train network, if we were to spawn 3 trains per line (total 9 trains), then in tick 0 we spawn 6 trains in exactly this order:

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

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

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

1.1.3 Station Layout

When a train arrives at a station (possibly by spawning), it needs to load/unload passengers before it can leave for the next station. A train 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 train; all other trains waiting to use the same link must wait in the platform's holding area.

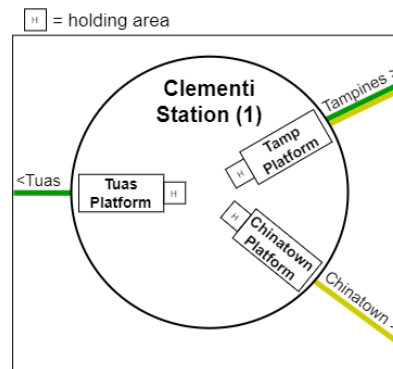


Figure 2: Platforms

1.1.4 Train Behaviour on Arrival

Upon arriving at a station:

- if the platform is free and no other trains are queuing, it goes to the platform corresponding to the station it wants to go next, and starts loading/unloading passengers.
- otherwise, it queues up at the holding area.

Note that each queue is exclusive to each platform, i.e. in Figure 2 above, there are three queues: one for trains going to Tuas, one for trains going to Tampines, and one for trains going to Chinatown.

Trains at the holding area will move to the platform at the same tick when the train occupying the platform moves down to the next link (Figure 3).

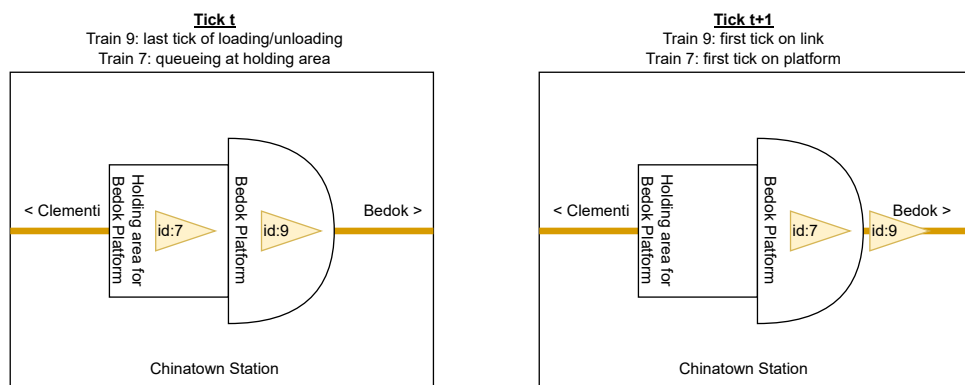


Figure 3: Movement from Holding Area to Platform

The above specification also implies that if two trains arrive at the same tick and both the platform and the holding area are previously empty, one of the trains would go straight to the platform and the other train would go to the holding area. The priority on which train gets the platform is described in the section below.

1.1.5 Queueing at Holding Area

A train's position in the queue is determined by the arrival time, followed by the train ID. A train that arrives earlier would be positioned in front of trains that arrive later. If multiple trains are arriving at the same time, they queue up in ascending order of train ID (i.e. lower IDs go before higher IDs for the same arrival time).

1.1.6 Loading / Unloading

Each train would spend some number of ticks on the station platform to unload and load passengers. To simulate different crowd sizes in each station, a function would be run to determine the time taken in the station platform. **This function should be called exactly once when the train arrives at the platform.** Details on how the function operates are included in Section 2.1 and Appendix B. It would then take that number of ticks to unload/load passengers.

After unloading and loading passengers, if the link is free, the train transits to the link. Otherwise, it waits until the link is free before transiting. The details on the movement of the train transiting to the link are shown in Figures 4 and 5.

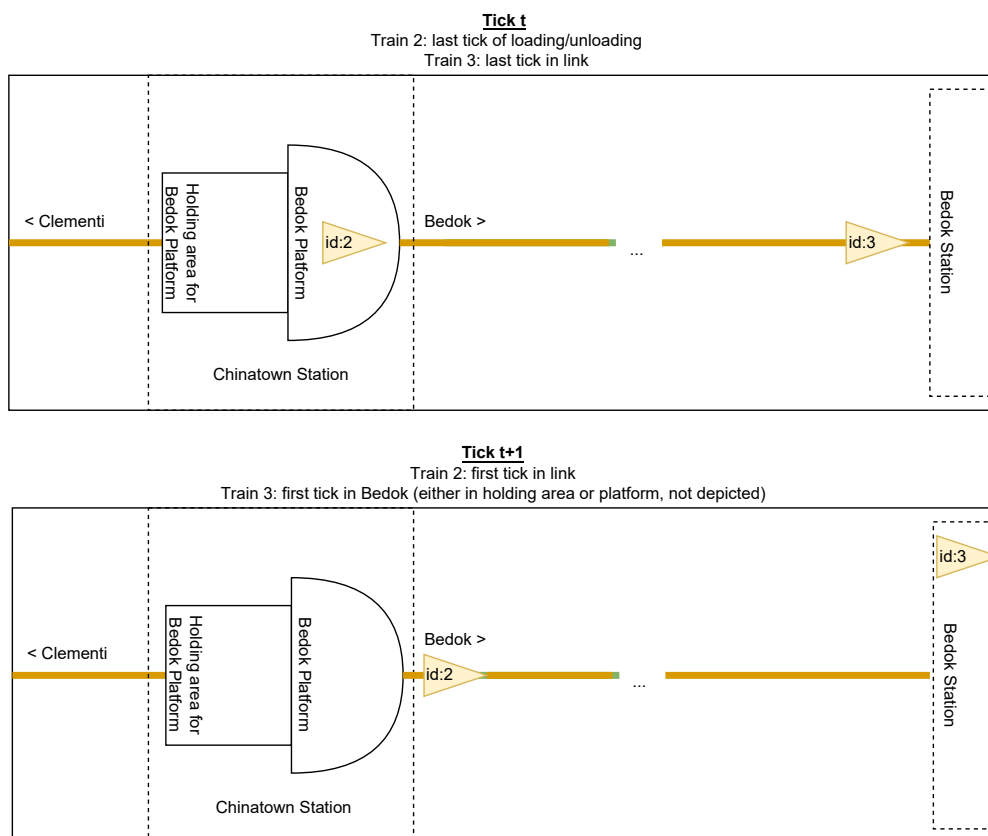


Figure 4: Movement from Platform to Link (no-waiting scenario)

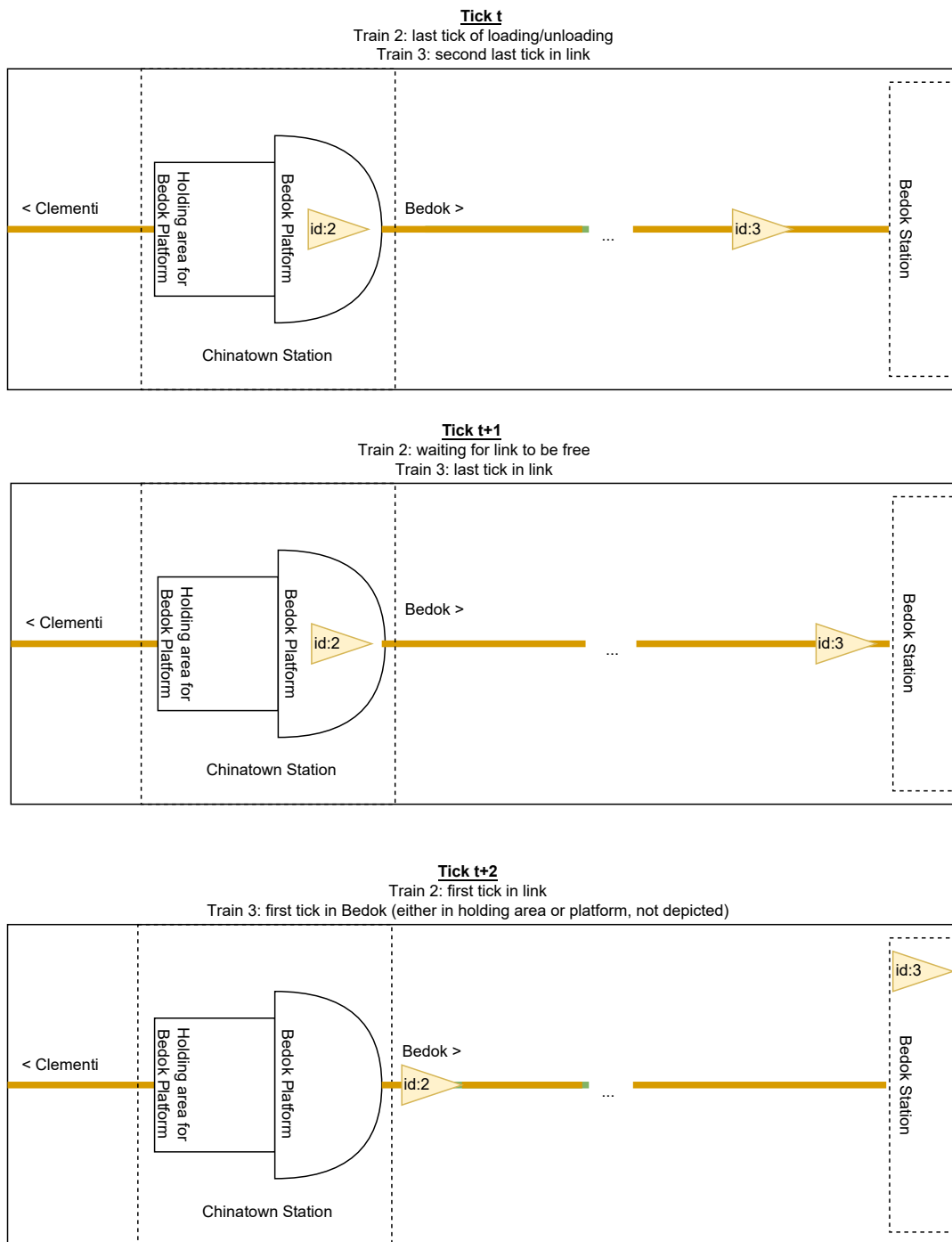


Figure 5: Movement from Platform to Link (waiting scenario)

1.1.7 Travelling on a Link

A train travelling on a link remains there for a number of ticks equal to the distance of the link. Afterwards, the train arrives at its destination station and instantly goes to either a station platform or to a holding area.

1.1.8 Terminal Stations

When a train 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 train 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 trains are automated, never break down and never change lines, so once they are placed on the train network they will continue “bouncing” between their line’s terminal stations until the simulation ends.

1.1.9 Summary of Train Lifecycle

A summary of a train’s lifecycle is as follows:

1. Arriving at a station (upon spawning or transiting from link)
 - The train will go to the holding area or platform associated with the next station
 - If a train arrived at the terminal station, it will go to the opposite direction of the same line.
2. Unloading and loading passengers
 - Only at a station platform (not holding area)
 - For as many ticks as needed, as specified by the function in [Appendix B](#).
 - Might need to for link to be free before transiting to the link
3. Travelling between stations
 - Only when on a link
 - For as many ticks as the link’s distance

1.1.10 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.

2 Inputs and Outputs

2.1 Starter Code

For this assignment, you will be given starter code, similar to Assignments 1 and 2. The starter code will include the following:

- Skeleton code (`main.cc`) to help with processing input/output and timing your code. **Do not change!**
- A header file with the implementation for the pseudo-random number generator (PRNG) needed to generate the waiting time of each train in the station (as specified in Appendix B). **Do not change!**
- An executable (`bench-seq`) of the sequential implementation (no code provided), as a reference implementation.
- `gen_test.py`, which is a script to help you generate test cases. Run `python3 gen_test.py -h` for details on what arguments you have to pass to the script.
- A file to implement your solution (`simulate.cc`) – you can add more files if you add them to your Makefile, as in the previous assignments.

Note that *each platform within every station* needs to be initialized with a copy of the PRNG, by using our provided class `PlatformLoadTimeGen` within the file `platform_load_time_gen.hpp`. **Do not modify this file.** Initialize the PRNG with the popularity value of the station. As an example, if your platform is modelled as a struct, below is a code fragment outlining the initialization of the platform:

```
struct Platform {
    ...
    PlatformLoadTimeGen pltg;

    Platform(const size_t popularity, ...) :
        ..., pltg(popularity) {}
}
```

To generate each new waiting time at a platform for a given train, call the `next` function with the train's ID. Example below:

```
Platform my_platform;
...
platform_waiting_time = my_platform.pltg.next(next_train_id);
...
```

2.2 Input Format

An input file strictly follows the structure below:

1. S: number of train stations in the network. S is an integer ≥ 2 .
2. V: number of lines in the network. For base (non-bonus) submission, this will always be 3.
3. L: a single line containing the space-separated list of stations. All station names only contain the following characters: 'a'-'z', 'A'-'Z'.
4. 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.

5. 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 entry in the matrix must be a non-negative integer. An entry of 0 indicates that the corresponding link does not exist.
6. L : V consecutive lines denoting the list of stations; each line denotes a train line (i.e. inputs for the base version have three lines; each for the green, yellow, and blue lines respectively). The order of the stations from left-to-right should be taken as the forward direction of the line.
7. N : number of ticks to run for the simulation. This must be a non-negative integer.
8. T : a single line containing V numbers. Each denotes the number of trains to spawn per line. This must be a non-negative integer.
9. NUM_PRINT - number of ticks (from the end) to output the simulation state. E.g. if $NUM_PRINT = 5$, then the program should output the simulation states for the last 5 ticks. $1 \leq NUM_PRINT \leq N$

Additionally, the following constraints/specifications apply to each train line above:

1. The number of stations in a train line is an integer ≥ 2 .
2. Every link in a train line must exist. That is, every link in a train line must correspond to a *positive* integer value in the adjacency matrix M .
3. Every train line in a given direction must be a *simple path*. That is, a train line must contain no repeated stations and no repeated links. Note that two different train lines may still share stations and/or links, as in Figure 1.
4. There will be no loops within each line.

2.2.1 Example Input

The following input uses the graph depicted in Figure 1.



Example Input

```
8
3
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
```

2.3 Output Format

For each of the last NUM_PRINT 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 train, separated by a space, according to the following rules
 - (a) Each train 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 train's ID, and a hyphen character.
 - (b) If a train is travelling on a link, it should output the source station name, followed by "->", and the destination station's name.
 - (c) Otherwise if a train is at a platform, output the station name the train is at, followed by a percent sign "%".
 - (d) Otherwise, a train must be in a holding area. In this case, output the station name the train is at, followed by a hash sign "#".
 - (e) The positions should be sorted lexicographically (refer to the sample output).

2.3.1 Example Output

Below is an example to outline the format of the output, but the following example might not correspond to the example input given above. Note that some lines may have wrapped in the following example. 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
```

3 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 have a different teammate compared to Assignments 1 and 2. You may discuss the assignment with others but in the case of plagiarism, both parties will be severely penalised. Cite your references or at least mention them in your report (what you referenced, where it came from, how much you referenced, etc.). *This also includes the use of AI tools such as ChatGPT, Copilot, etc.* Please refer to our policy in the introductory lecture on AI tool usage.

The grades are divided as follows:

- 5 marks – MPI implementation correctness; detailed in section [3.1.1](#).
- 2 marks – MPI implementation speedup; detailed in section [3.1.2](#)
- 2 marks – Assignment report; mark breakdown detailed in section [3.2.2](#)

3.1 Program Requirements


Your MPI implementation should:

- Be written in C++.
- Have no memory leaks.
- Include a Makefile that builds your implementation when calling `make`, and emits it as an executable named `trains`. We will use this executable to generate the output of your implementation.
- The Makefile must work at the top-level of your repository, and the executable emitted must be at the top-level of the repository.
- Only use MPI for parallel execution. Do not use any other parallelisation constructs, paradigms, and capabilities like OpenMP, threading, and GPGPU.
- Be able to run on any combination of up to two nodes in the CS3210 PDC cluster, with up to n ranks, where n is the total number of physical cores of the nodes allocated. You may assume that for the case of two nodes, the ranks would be distributed in a round-robin fashion (i.e. rank 1 in node 1, rank 2 in node 2, rank 3 in node 1, rank 4 in node 2, etc.), where every rank is bound to a physical core.

However, your MPI implementation **does not** have to handle network issues; you can assume the delivery of messages between nodes will not be affected by network issues.

3.1.1 Correctness (5 marks)

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 by diffing the standard outputs of our implementation and your implementation, as follows:



```
$ diff <file a> <file b> -ZB
```

As usual, 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 [4.2](#).

3.1.2 Speedup (2 marks)

Your program should:

- Achieve a speedup greater than 1x as compared to our reference sequential implementation, following the restrictions below for inputs for performance runs. The runtime will be taken from the line printed to `std::cerr`, starting with “sequential_time:” and “mpi_time:” respectively for the sequential executable, and your executable. You may want to redirect the standard output of your executable to `/dev/null` once you are sure your program is correct – this will allow you to clearly observe the timing printout.
- Achieve a speedup of at least 1.5x when using at least 8 processes as compared to running a few (< 3) processes. That is, it should scale reasonably with increasing processes. **Do remember to test this on a configuration with at least 8 physical cores** (i.e. not with a single i7-7700 node, as there is only a total of 4 physical cores).
- Note that for any performance measurements, the sequential executable and your MPI program will be run on a single node. You do not have to meet the speedup requirements if your MPI program is run across multiples nodes.

The provided test cases in the skeleton code are only used for testing correctness. With reference to our test case generator (`gen_test.py`), the specifications of the input that you should use to test your code for performance are as below:

- S (total no. of stations) ≥ 30 .
- $\text{max_link_weight} \leq \text{max_popularity} \leq 10$.
- `max_num_trains`: Between N (no. of ticks to simulate) and $2 * N$.
- $\text{max_line_len} \geq 0.5 * S$.
- N (no. of ticks to simulate) ≥ 500 .
- `num_ticks_to_print` = 1.
- Does not exceed 1.5 minutes of execution on our sequential reference program on an i7-7700 machine.

You can generate the minimum testcase with the following command: `python3 gen_test.py 30 10 10 500 15 500 > testcases/performance/min_test.in`. We reserve the right to penalise the speedup grade should the program submitted exhibit correctness issues.



You are also expected to benchmark your program **across two nodes** & provide performance measurements in your report. However, **it is normal that a single-node config will achieve a greater speedup than a multi-node config**. This is okay for the assignment, because the workload is not intensive enough to outweigh the cost of cross-node MPI communication.

3.2 Report

3.2.1 Format

Your report should follow these specifications:

- Four pages maximum for the 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 3.2.2.
- 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-organised, easily readable reports.

3.2.2 Content

Your report should contain:

- (1 mark) A description of:
 1. your program's parallelisation strategy and design
 2. how deadlocks/race conditions are resolved in your implementation
 3. the key MPI constructs used to implement this strategy and why they are used

Diagrams are not required but may help you explain something clearly without taking up much space.

- (1 mark) An analysis of which parameters affect speedup the most. Include possible explanations that cause it. 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 cores, ranks, and nodes to run your implementation.

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 impacting the 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.

4 Admin

4.1 Connecting to the Lab Machines

The link to the documentation on how to use the PDC lab machines can be found in your Lab 4 sheet. For convenience, the link to the CS3210 guide can be accessed at <https://nus-cs3210.github.io/student-guide/accessing/>.

4.2 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 Theodore (theo@nus.edu.sg).

4.3 Deadline and Submission

Assignment submission is due on **Fri, 15 Nov, 2pm**.

- **Canvas Quiz Assignment 3:** Take the quiz and provide the name of your repository and the commit hash corresponding to the `a3-submission` tag; if you are working in a team, only one team member needs to submit the quiz. If both of you submit, we will take the latest submission.
- **GitHub Classroom:** The implementation and report must be submitted through your Github Classroom repository. The GitHub Classroom for Assignment 3 can be accessed through <https://classroom.github.com/a/-M35kRc8>.

The GitHub Classroom submission must adhere to the following guidelines:

- Name your team `a3-e0123456` (if you work by yourself) or `a3-e0123456-e0173456` (if you work with another student) – substitute your NUSNET number accordingly.
- Push your **code and report** to your team's GitHub Classroom repository.
- Your report must be a PDF named `<teamname>.pdf`. For example, `a3-e0123456-e0173456.pdf`. `<teamname>` should **exactly match** your team's name; if you are working in a pair, please DO NOT flip the order of your admin numbers.
- Tag the commit that you want us to grade with `a3-submission`.

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



Final check: Ensure that you have submitted to both **Canvas** and **GitHub Classroom**. Canvas should contain your commit hash and repository name (**in the right format!**), and GitHub Classroom should contain your code and report.

5 Bonus – Train Breakdowns

You may obtain up to 1 bonus mark for efficiently implementing additional capabilities to the base version of the assignment. **Both of these additional capabilities need to be implemented to get the bonus mark.** The implementation needs to still be correct, and satisfy the scaling requirement of the base submission. **Note that the given main.cc file will not work for the bonus** as it does not do all the I/O necessary, so you should not use it as-is. Do not modify main.cc as we will replace it with our own version for the base submission.

5.1 Requirement 1: Variable Number of Lines

In the base version of this assignment, there are always 3 lines in the network. Your bonus submission needs to be able to accommodate differing amount of train lines.

As such, the input file will have the following modification:

- V can now take any value between 1 and 10, inclusive.

The train lines will be named as following:

- | | |
|----------------------|-------------------------|
| • line 1: green (g) | • line 6: purple (p) |
| • line 2: yellow (y) | • line 7: turquoise (t) |
| • line 3: blue (b) | • line 8: pink (k) |
| • line 4: red (r) | • line 9: lime (l) |
| • line 5: brown (w) | • line 10: grey (e) |

For avoidance of doubt, all the previous spawning rules apply and extend to the number of lines supplied. For example, if we were to spawn 2 trains per line in a network with 4 lines, we will spawn in the following order:

- train 0: line 1 (g), going forwards
- train 1: line 1 (g), going backwards
- train 2: line 2 (y), going forwards
- train 3: line 2 (y), going backwards
- train 4: line 3 (b), going forwards
- train 5: line 3 (b), going backwards
- train 6: line 4 (r), going forwards
- train 7: line 4 (r), going backwards

5.2 Requirement 2: Faults

For the second requirement, we are simulating the case where faults occur in the network. There are two kinds of faults which can occur: link failure and train failure.

Link Failure

Upon a link failure, two things occur:

- Any train in the link needs to finish its travel to the next station. To maintain safety, the train would move at half speed after the tick when the link fails, i.e. it will take twice the remaining time to finish travelling to the next station. Example is given in Figure 6.
- When there are no train left in the link, maintenance on the link will begin for x ticks, where x is twice the length of the link. No train may move from the station platform to the link when maintenance is in progress.

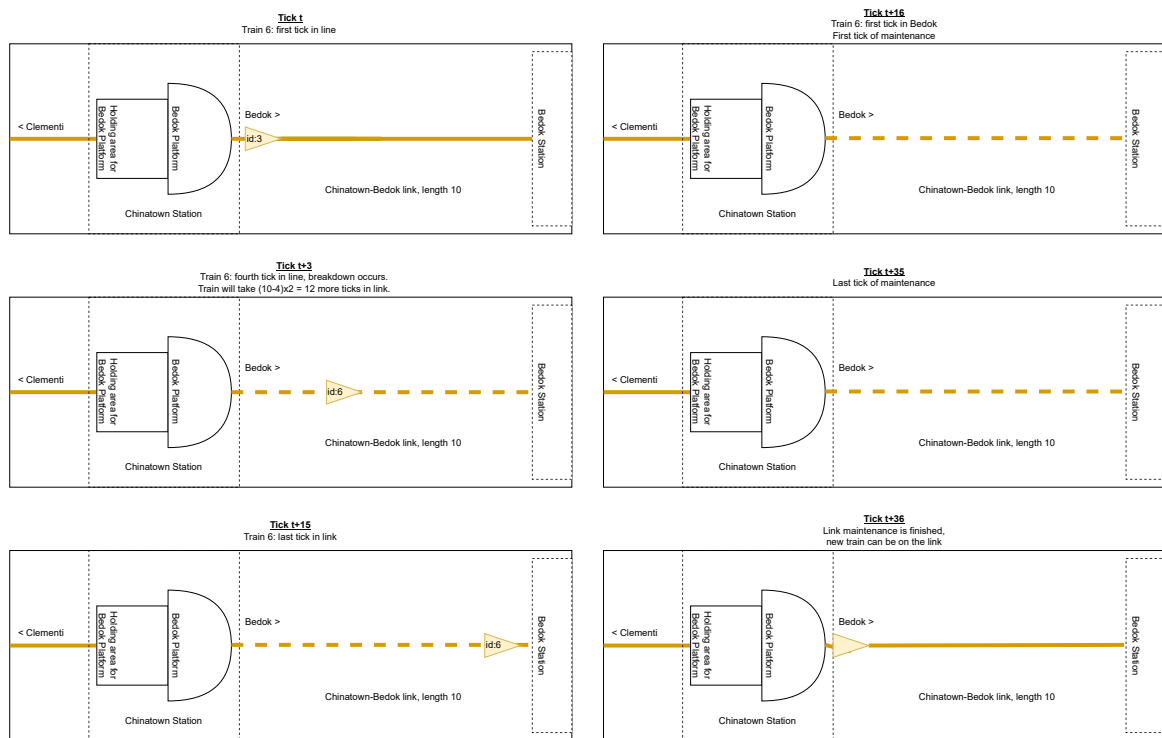


Figure 6: Link Failure (three diagrams on the left, followed by three diagrams on the right)

Train Failure

Upon a train failure:

- If the train is at a link, it will move at half speed towards the next station. Upon arrival of the train to the next station, the link previously occupied by the train will be down for maintenance for x ticks, where x is twice the length of the link. No train may move from the station platform to the link when maintenance is in progress. In the output, denote this with the usual notation of writing trains on transit.
- If/when the train is at a station (either at holding area, platform, or just arrived after moving at half speed due to the failure), the train gets decommissioned and taken out of circulation. This train will no longer move and will stay at the station indefinitely. In the output, denote this with "\$" (e.g. a decommissioned train with id 0 at Clementi on the Green line will be denoted by g0-clementi\$).

Note the following:

- It is possible that multiple failures coincide.
- If a *train failure* coincides with a *link failure*, the train will move at quarter speed. Trains will not move slower than quarter speed, i.e. if a train is moving at quarter speed and another fault is found at the link or on the train, it does not move slower.
- Multiple failures of the same type coinciding will not cause the train to go slower than it already is.
- A decommissioned train will remain in a decommissioned state if it suffers from another train failure.
- The maintenance time due to multiple train failures happening to a single train on the link will not stack. For example, if a train failure happens twice when a train transits between Chinatown and Bedok (length 10), the maintenance time due to these train failures remain at 20 ticks, starting when the train arrives at Bedok Station.
- If a link is under maintenance due to a train or link failure, and another link failure is found during this maintenance period, add the new maintenance time to the current remaining maintenance time.

To accommodate for the bonus, there will be further changes to the input file. After the `NUM_PRINT` line, the input will contain:

- *NLF*: the number of link failures in the simulation. This is a non-negative number.
- *NTF*: the number of train failures in the simulation. This is a non-negative number.
- *LF*: *NLF* lines of link failures. They will be denoted by the format `lf <station1> <station2> <tick>`
- *TF*: *NTF* lines of train failures. They will be denoted by the format `tf <train id> <tick>`



Warning:

You may need to extend the skeleton code to accommodate for these changes in the input file. To avoid a possible conflict with the base submission, you may want to create separate source files for bonus I/O.

5.3 Bonus Submission

To qualify for the bonus marks, these extra submissions are required:

- One extra page in your report, with the header "Bonus". This section should include an outline on how you implemented the bonus requirements, as well as performance comparison on sufficient number of test cases to show that the speedup criteria still holds.
- A recipe in your Makefile called `make bonus` which creates an executable called `trains-bonus`. We will use this executable to check the correctness of your bonus implementation.

A 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
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
```

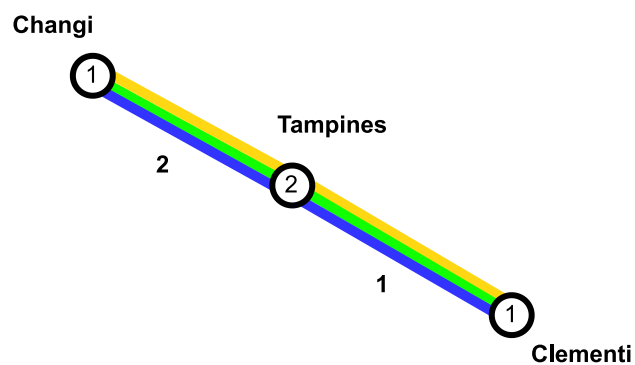


Figure 7: Sample input 1 train network



Sample Output 1

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

Table 1: Explanation of Sample Output 1

Tick	Explanation
0	Train 0 spawns at changi (start of green line), immediately arrives at the platform leading to tampines (no other trains there) and starts to load / unload passengers (1st tick; total waiting time is 1 tick)
1	Train 0 is in transit on the link from changi to tampines (1st tick; total travel time is 2 ticks)
2	Train 0 continues on the link (2nd tick)
3	Train 0 arrives at the platform in tampines leading to clementi (no other trains there) and starts to load / unload passengers (1st tick; total waiting time is 1 tick)
4	Train 0 is in transit on the link from tampines to clementi (1st tick; total travel time is 1 tick)
5	Train 0 arrives at the platform in clementi leading to tampines (no other trains there; also, clementi is the final station in the green line so the train wants to go back to tampines) and starts to load / unload passengers (1st tick; total waiting time is 1 tick)
6	Train 0 is in transit on the link from clementi to tampines (1st tick; total travel time is 1 tick)
7	Train 0 arrives at the platform in tampines leading to changi (no other trains there) and starts to load / unload passengers (1st tick; total waiting time is 1 tick)
8	Train 0 is in transit on the link from tampines to changi (1st tick; total travel time is 2 ticks)
9	Train 0 continues on the link (2nd tick)
10	Train 0 arrives at the platform in changi leading to tampines (no other trains there; also, changi is the first station in the green line so the train wants to go back to tampines) and starts to load / unload passengers (1st tick; total waiting time is 1 tick)
11	Train 0 is in transit on the link from changi to tampines (1st tick; total travel time is 2 ticks)
12	Train 0 continues on the link (2nd tick)
13	Train 0 arrives at the platform in tampines leading to clementi (no other trains there) and starts to load / unload passengers (1st tick; total waiting time is 2 ticks)
14	Train 0 continues to load / unload passengers (2nd tick)



Sample Input 2

```
5
3
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
```

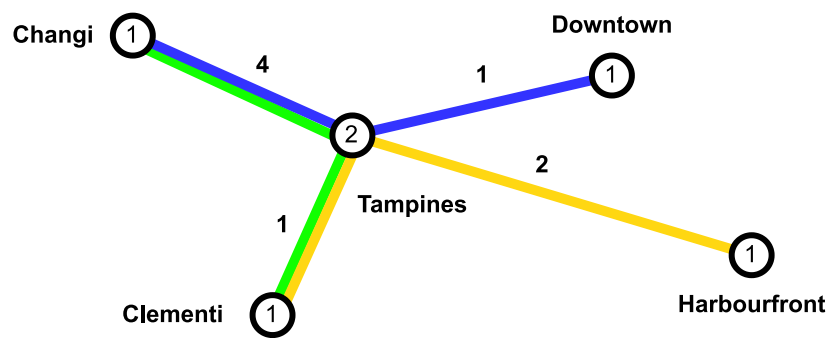


Figure 8: Sample input 2 train network



Sample Output 2

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

Table 2: Explanation of Sample Output 2

Tick	Explanation
0	4 trains are spawned. Train 0 is green going forward; Train 1 is green going backward; Train 2 is yellow going forward; Train 3 is blue going forward. The third green train has not spawn yet because at most 1 train per terminal station in each line spawns in a tick. Trains 0 and 3 both want to use the link from changi to tampines and they both arrived (spawned) in the same tick. But train 0 has a lower ID than train 3 so it goes to the platform and starts to load / unload passengers and train 3 goes to the holding area for that platform. For the same reason, train 1 goes to the platform from clementi to tampines and starts to load / unload passengers while train 2 goes to the corresponding holding area.
1	Train 0 begins transiting from changi to tampines (4 ticks left). Train 1 begins transiting from clementi to tampines (1 tick left). Trains 2 and 3 instantly move to the now-free station platforms and start to load / unload passengers (1 tick left). The last green train (train 4) spawns at changi (going forward) and queues behind train 3 in the holding area for the link headed to tampines.
2	Train 0 is still in transit (3 ticks left). Train 1 arrives at the station platform from tampines to changi and starts to load / unload passengers (1 tick left). Train 2 begins transiting from clementi to tampines (1 tick left). Train continues waiting at the platform because the link is occupied. Train 4 continues waiting behind train 3.
3	Train 0 is still in transit (2 ticks left). Train 1 begins transiting from tampines to changi (4 ticks left). Train 2 arrives at the station platform from tampines to harbourfront and starts to load / unload passengers (1 tick left). Train 3 continues waiting at the platform because the link is occupied. Train 4 continues waiting behind train 3.
4	Train 0 is still in transit (1 tick left). Train 1 is still in transit (3 ticks left). Train 2 begins transiting from tampines to harbourfront (2 ticks left). Train 3 continues waiting at the platform because the link is occupied. Train 4 continues waiting behind train 3.
5	Train 0 arrives at the station platform from tampines to clementi and starts to load / unload passengers (1 tick left). Train 1 is still in transit (2 ticks left). Train 2 is still in transit (1 tick left). Train 3 begins transiting from changi to tampines (4 ticks left). Train 4 instantly moves to the platform from changi to tampines and starts to load / unload passengers (1 tick left).
6	Train 0 begins transiting from tampines to clementi (1 tick left). Train 1 is still in transit (1 tick left). Train 2 arrives at the station platform from harbourfront to tampines and starts to load / unload passengers (1 tick left). Train 3 is still in transit (3 ticks left). Train 4 continues waiting at the platform because the link is occupied.
7	Train 0 arrives at the station platform from clementi to tampines and starts to load / unload passengers (1 tick left). Train 1 arrives at changi and queues behind train 4 in the holding area of the station platform from changi to tampines. Train 2 begins transiting from harbourfront to tampines (2 ticks left). Train 3 is still in transit (2 ticks left). Train 4 continues waiting at the platform because the link is occupied.
8	Train 0 begins transiting from clementi to tampines (1 tick left). Train 1 continues waiting behind train 4. Train 2 is still in transit (1 tick left). Train 3 is still in transit (1 tick left). Train 4 continues waiting at the platform because the link is occupied.

B Generation of Loading/Unloading Times

The loading times of each train depends on the popularity of the station, which determines the average length of time that the train takes. To generate the actual loading time of the train on the platform, we are using a Poisson distribution (from the C++ standard library) to ensure that the average waiting time would approach the popularity of the station.

To ensure that the results are replicable, we are using a pseudo-random number generator to create the seed for `std::poisson_distribution`. You should call the waiting time generation function only when necessary (e.g., no pre-generation of waiting times, only call when the train is at the platform). The specific details are in `platform_load_time_gen.hpp` in the class `PlatformLoadTimeGen`. Details on how to initialize and call this class are in the Starter Code section.