COMP3702 Artificial Intelligence (Semester 2, 2024)
Assignment 2: BeeBot MDP

Key information:

- **Due: 1pm, Friday 20 September 2024**
- This assignment assesses your skills in developing discrete search techniques for challenging problems.
- Assignment 2 contributes 20% to your final grade.
- This assignment consists of two parts: (1) programming and (2) a report.
- This is an individual assignment.
- Both code and report are to be submitted via Gradescope (`https://www.gradescope.com/`). You can find a link to the COMP3702 Gradescope site on Blackboard.
- Your code (Part 1) will be graded using the Gradescope code autograder, using the testcases in the support code provided at `https://github.com/comp3702/2024-Assignment-2-Support-Code`.
- Your report (Part 2) should fit the template provided, be in .pdf format and named according to the format `a2-COMP3702-[SID].pdf`. Reports will be graded by the teaching team.

The BeeBot AI Environment

You have been tasked with developing a planning algorithm for automatically controlling BeeBot, a Bee which operates in a hexagonal environment, and has the capability to push, pull and rotate honey 'Widgets' in order to reposition them to target honeycomb locations. To aid you in this task, we have provided support code for the BeeBot environment which you will interface with to develop your solution. To optimally solve a level, your AI agent must efficiently find a sequence of actions so that every Target cell is occupied by part of a Widget, while incurring the minimum possible action cost.

For A2, the BeeBot environment has been extended to model non-deterministic outcomes of actions. Cost and action validity are now replaced by a reward function where action costs are represented by negative received rewards, with additional penalties (i.e., negative rewards) being incurred when a collision occurs (between the Bee or a held Widget and an obstacle, or between Widgets). Updates to the game environment are indicated in pink font.

Levels in BeeBot are composed of a Hexagonal grid of cells, where each cell contains a character representing the cell type. An example game level is shown in Figure 1.

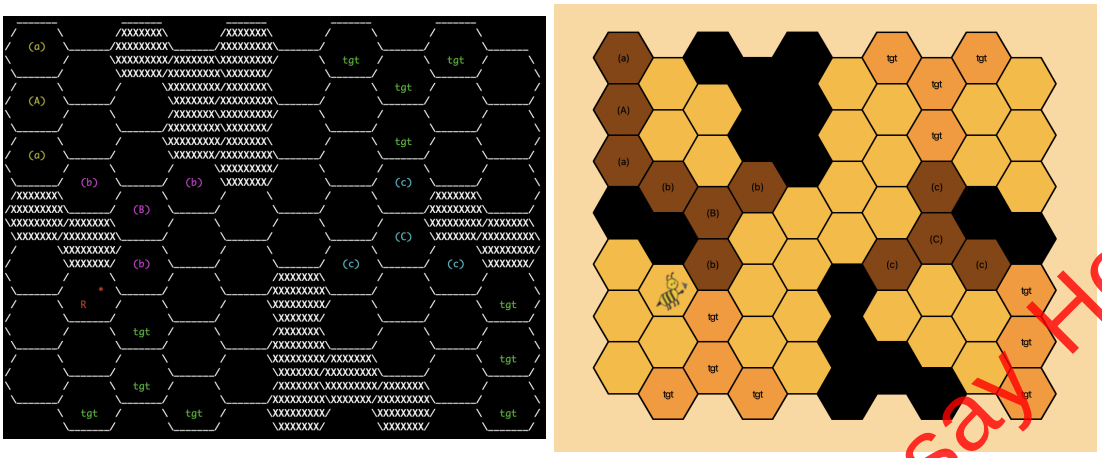

Figure 1: Example game level of BeeBot, showing character-based and GUI visualiser representations

Environment representation

Hexagonal Grid

The environment is represented by a hexagonal grid. Each cell of the hex grid is indexed by (row, column) coordinates. The hex grid is indexed top to bottom, left to right. That is, the top left corner has coordinates $(0, 0)$ and the bottom right corner has coordinates $(n_{rows} - 1, n_{cols} - 1)$. Even numbered columns (starting from zero) are in the top half of the row, and odd numbered columns are in the bottom half of the row. An example is shown in Figure 2.

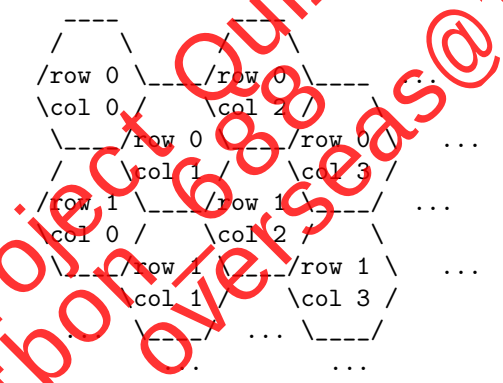

Figure 2: Example hexagonal grid showing the order that rows and columns are indexed

Two cells in the hex grid are considered adjacent if they share an edge. For each non-border cell, there are 6 adjacent cells.

The BeeBot Agent and its Actions

The BeeBot Bee occupies a single cell in the hex grid. In the visualisation, the Bee is represented by the cell marked with the character 'R' (or the Bee in the graphical visualiser). The side of the cell marked with '*' (or the arrow in the graphical visualiser) represents the front of the Bee. The state of the Bee is defined by its (row, column) coordinates and its orientation (i.e. the direction its front side is pointing towards).

At each time step, the agent is prompted to select an action. The Bee has 4 available nominal actions:

- Forward → move to the adjacent cell in the direction of the front of the Bee (keeping the same orientation)
- Reverse → move to the adjacent cell in the opposite direction to the front of the Bee (keeping the same orientation)
- Spin Left → rotate left (relative to the Bee's front, i.e. counterclockwise) by 60 degrees (staying in the same cell)
- Spin Right → rotate right (i.e. clockwise) by 60 degrees (staying in the same cell)

Each time the Bee selects an action, there is a fixed probability (given as a parameter of each testcase) for the Bee to 'drift' by 60 degrees in a clockwise or counterclockwise direction (separate probabilities for each drift direction) before the selected nominal action is performed. The probability of drift occurring depends on which nominal action is selected, with some actions more likely to result in drift. Drifting CW and CCW are mutually exclusive events.

Additionally, there is a fixed probability (also given as a parameter of each testcase) for the Bee to 'double move', i.e. perform the nominal selected action twice. The probability of a double move occurring depends on which action is selected. Double movement may occur simultaneously with drift (CW or CCW).

The reward received after each action is the minimum/most negative out of the rewards received for the nominal action and any additional (drift/double move) actions.

The Bee is equipped with a gripper on its front side which allows it to manipulate honey Widgets. When the Bee is positioned with its front side adjacent to a honey Widget, performing the 'Forward' action will result in the Widget being pushed, while performing the 'Reverse' action will result in the honey Widget being pulled.

Action Costs
Each action has an associated cost, representing the amount of energy used by performing that action.

If the Bee moves without pushing or pulling a honey widget, the cost of the action is given by a base action cost, `ACTION_BASE_COST[a]` where 'a' is the action that was performed.

If the Bee pushes or pulls a honey widget, an additional cost of `ACTION_PUSH_COST[a]` is added on top, so the total cost is `ACTION_BASE_COST[a] + ACTION_PUSH_COST[a]`.

The costs are detailed in the `constants.py` file of the support code:

```
ACTION_BASE_COST = {FORWARD: 1.0, REVERSE: 1.0, SPIN_LEFT: 0.1, SPIN_RIGHT: 0.1}
ACTION_PUSH_COST = {FORWARD: 0.8, REVERSE: 0.5, SPIN_LEFT: 0.0, SPIN_RIGHT: 0.0}
```

Obstacles
Some cells in the hex grid are obstacles. In the visualisation, these cells are filled with the character 'X' (represented as black cells in the graphical visualiser). Any action which causes the Bee or any part of a honey Widget to enter an obstacle results in collision, causing the agent to receive a negative obstacle collision penalty as reward. This reward replaces the movement cost which the agent would have otherwise incurred. The outside boundary of the hex grid behaves in the same way as an obstacle.

Additionally, the environment now contains an additional obstacle type, called 'thorns'. Thorns behave in the same way as obstacles, but when collision occurs, a different (larger) penalty is received as the reward. As a result, avoiding collisions with thorns has greater importance than avoiding collisions with obstacles. Thorns are represented by '!' in the text-based visualisation and red cells in the graphical visualisation.

Widgets
Honey Widgets are objects which occupy multiple cells of the hexagonal grid, and can be rotated and translated by the BeeBot Bee. The state of each honey widget is defined by its centre position (row, column) coordinates and its orientation. Honey Widgets have rotational symmetries - orientations which are rotationally symmetric are considered to be the same.

In the visualisation, each honey Widget in the environment is assigned a unique letter 'a', 'b', 'c', etc. Cells which are occupied by a honey Widget are marked with the letter assigned to that Widget (surrounded by round brackets). The centre position of the honey Widget is marked by the uppercase version of the letter, while all other cells occupied by the widget are marked with the lowercase.

Three honey widget types are possible, called Widget3, Widget4 and Widget5, where the trailing number denotes the number of cells occupied by the honey Widget. The shapes of these three honey Widget types and each of their possible orientations are shown in Figures 3 to 5 below.

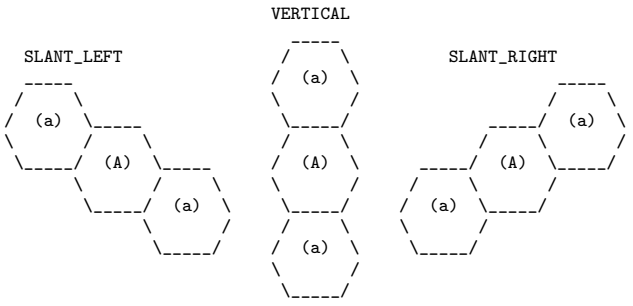

Figure 3: Widget3

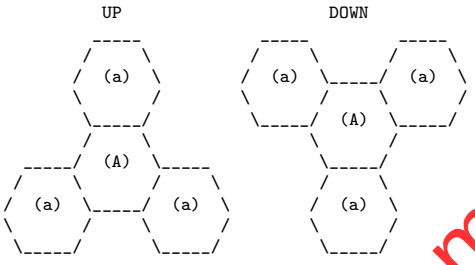

Figure 4: Widget4

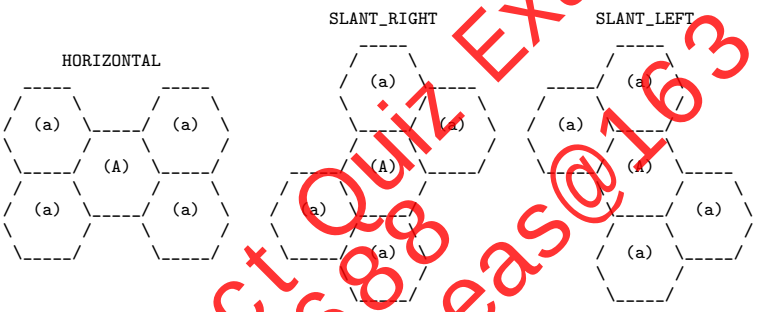

Figure 5: Widget5

Two types of widget movement are possible – translation (change in centre position) and rotation (change in orientation).

Translation occurs when the Bee is positioned with its front side adjacent to one of the honey widgets' cells such that the Bee's orientation is in line with the honey widget's centre position. Translation results in the centre position of the widget moving in the same direction as the Bee. The orientation of the honey Widget does not change when translation occurs. Translation can occur when either 'Forward' or 'Reverse' actions are performed. For an action which results in translation to be valid, the new position of all cells of the moved widget must not intersect with the environment boundary, obstacles, the cells of any other honey Widgets or the Bee's new position.

Rotation occurs when the Bee's new position intersects one of the cells of a honey Widget but the Bee's orientation does not point towards the centre of that widget. Rotation results in the honey widget spinning around its centre point, causing the widget to change orientation. The position of the centre point does not change when rotation occurs. Rotation can only occur for the 'Forward' action - performing 'Reverse' in a situation where 'Forward' would result in a widget rotation is considered invalid.

The following diagrams show which moves result in translation or rotation for each honey Widget type, with the arrows indicating directions from which the Bee can push or pull a widget in order to cause a translation or rotation of the widget. Pushing in a direction which is not marked with an arrow is considered invalid.

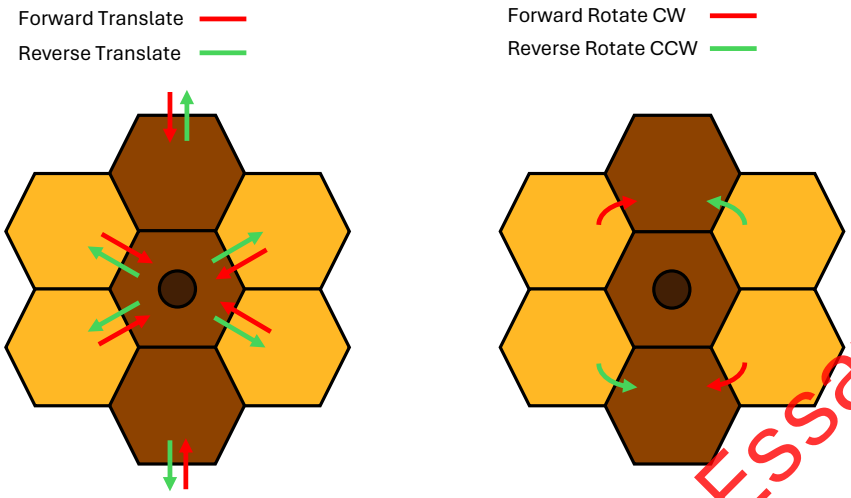

Figure 6: Widget3 translations and rotations

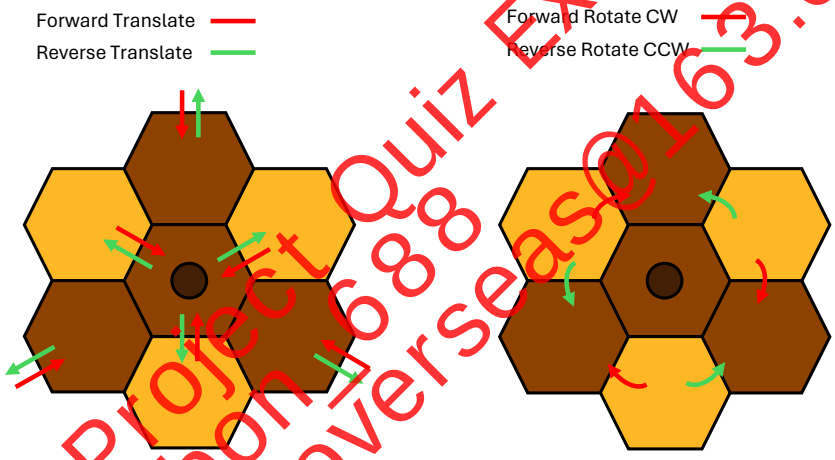

Figure 7: Widget4 translations and rotations

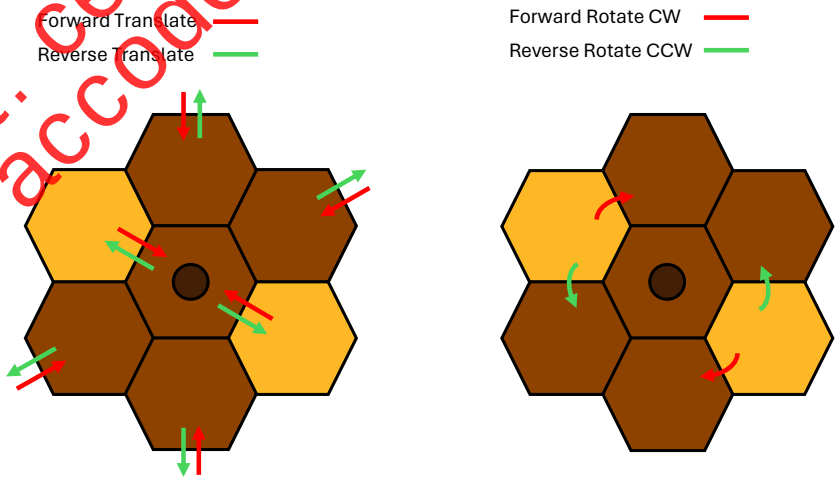

Figure 8: Widget5 translations and rotations

Targets
The hex grid contains a number of 'target' honeycomb cells which must be filled with honey. In the visualisation, these cells are marked with 'tgt' (cells coloured in orange in the graphical visualiser). For a BeeBot environment to be considered solved, each target cell must be occupied by part of a honey Widget. The number of targets in an environment is always less than or equal to the total number of cells occupied by all honey Widgets.

Interactive mode

A good way to gain an understanding of the game is to play it. You can play the game to get a feel for how it works by launching an interactive game session from the terminal with the following command for the graphical visualiser:

```
$ python play_game.py <input_file>.txt
```

or the following command for the command-line ASCII-character based visualiser:

```
$ python play.py <input_file>.txt
```

where `<input_file>.txt` is a valid testcase file (from the support code, with path relative to the current directory), e.g. `testcases/ex1.txt`.

Depending on your python installation, you should run the code using `python`, `python3` or `py`.

In interactive mode, type the symbol for your chosen action (and press enter in the command line version) to perform the action: press 'W' to move the Bee forward, 'S' to move the Bee in reverse, 'A' to turn the Bee left (counterclockwise) and 'D' to turn the Bee right (clockwise). Use 'Q' to quit the simulation, and 'R' to reset the environment to the initial configuration.

BeeBot as an MDP

In this assignment, you will write the components of a program to play BeeBot, with the objective of finding a high-quality solution to the problem using various sequential decision-making algorithms based on the Markov decision process (MDP) framework. This assignment will test your skills in defining a MDP for a practical problem and developing effective exact algorithms for MDPs.

What is provided to you

We provide supporting code in Python, in the form of:

1. A class representing BeeBot game map and a number of helper functions
2. A parser method to take an input file (testcase) and convert it into a BeeBot map
3. A tester
4. Testcases to test and evaluate your solution
5. A script to allow you to play BeeBot interactively
6. A solution file template

The support code can be found at: `https://github.com/comp3702/2024-Assignment-2-Support-Code`. Autograding of code will be done through Gradescope, so that you can evaluate your submission and continue to improve it based on this feedback — you are strongly encouraged to make use of this feedback. You can also test locally using the supplied `tester.py` file.

Your assignment task

Your task is to develop algorithms for computing policies, and to write a report testing your understanding of and analysing the performance of your algorithms. You will be graded on both your submitted **code (Part 1, 60%)** and the **report (Part 2, 40%)**. These percentages will be scaled to the 20% course weighting for this assessment item.

The provided support code formulates BeeBot as an MDP, and your task is to submit code implementing the following MDP algorithms:

- Value Iteration (VI)
- Policy Iteration (PI)

Individual testcases specify which strategy (value iteration/policy iteration) will be applied, but you may modify the strategy specified in your local copy of the test cases for the purpose of comparing the performance of your two algorithms. Note that any local changes you make to the test cases will not modify the test cases on Gradescope (against which your programming component will be graded). The difficulty of higher level testcases will be designed to require a more advanced solution (e.g. linear algebra policy iteration).

Once you have implemented and tested the algorithms above, you are to complete the questions listed in the section "Part 2 - The Report" and submit the report to Gradescope.

More detail of what is required for the programming and report parts are given below.

Part 1 — The programming task

Your program will be graded using the Gradescope autograder, using the testcases in the support code provided at `https://github.com/comp3702/2024-Assignment-2-Support-Code`.

Interaction with the testcases and autograder

We now provide you with some details explaining how your code will interact with the testcases and the autograder (with special thanks to Nick Collins and Aryaman Sharma).

Information on the BeeBot implementation is provided in the Assignment 2 Support Code *README.md* and code comments.

Implement your solution using the supplied solution.py template file. You are required to fill in the following method stubs:

- `__init__(game_env)`
- `vi_initialise()`
- `vi_is_converged()`
- `vi_iteration()`
- `vi_plan_offline()`
- `vi_get_state_value()`
- `vi_select_action()`
- `pi_initialise()`
- `pi_is_converged()`
- `pi_iteration()`
- `pi_plan_offline()`
- `pi_select_action()`

You can add additional helper methods and classes (either in `solution.py` or in files you create) if you wish. To ensure your code is handled correctly by the autograder, you should avoid using any try-except blocks in your implementation of the above methods (as this can interfere with our time-out handling). Refer to the documentation in `solution.py` for more details.

Grading rubric for the programming component (total marks: 60/100)

For marking, we will use 6 test cases to evaluate your solution. Each test case uses the algorithm specified as the solver type. Each test case is scored out of 10.0 marks, in the following four categories:

- Agent successfully reaches the goal
- Total reward
- Time elapsed
- Iterations performed

The 10 marks for each test case are evenly divided amongst the four categories (i.e. 2.5 marks are allocated for each category in each test case).

- Each test case has targets for total reward, time elapsed, and iterations performed.
- Maximum score is achieved when your program matches or beats the target in each category
- Partial marks are available for up to 1.3x total reward, 2x time elapsed and 1.3x number of iterations
- Total mark for the test case is a weighted sum of the scores for each category
- Total code mark is the sum of the marks for each test case

In addition, there is a hidden test which evaluates the correctness of your computed state-values in Value Iteration. If your values are found to deviate significantly from the correct values, marks may be deducted from your Gradescope code score. For the programming component of the assignments, the teaching staff will conduct interviews with a subset of students about their submissions for the purpose of establishing genuine authorship, as per the Electronic Course Profile.

Part 2 — The report

The report tests your understanding of the methods you have used in your code, and contributes 40/100 of your assignment mark. **Please make use of the report templates provided on Blackboard**, because Gradescope makes use of a predefined assignment template. Submit your report via Gradescope, in .pdf format, and named according to the format `a2-COMP3702-[SID].pdf`, where SID is your student ID. Reports will be graded by the teaching team.

Your report task is to answer the questions below, in the report template provided.

Question 1. MDP problem definition **(15 marks)**

a) Define the State space, Action space, Transition function, and Reward function components of the BeeBot MDP planning agent as well as where these are represented in your code. Ensure you complete both parts of the question: (10 marks)

   i) Define the four components in relation to BeeBot

   ii) Identify where they are represented in your code

b) Describe the purpose of a discount factor in MDPs. (2.5 marks)

c) State and briefly justify what the following dimensions of complexity are for the BeeBot MDP agent: (2.5 marks)

- Planning Horizon
- Sensing Uncertainty
- Effect Uncertainty
- Computational Limits
- Learning

Refer to the P&M textbook `https://artint.info/3e/html/ArtInt3e.Ch1.S5.html` for the possible values and definitions of each dimension, and format your answer in a neatly formatted table with the following column headings: [Dimension, Value, Justification].

Question 2. Comparison of algorithms and optimisations **(15 marks)**

This question requires comparison of your implementation of Value iteration (VI) and Policy iteration (PI). If you did not implement PI, you may receive partial marks for this question by providing insightful relevant comments on your implementation of VI. For example, if you tried standard VI and asynchronous VI, you may compare these two approaches for partial marks.

a) Describe your implementations of Value Iteration and Policy Iteration in one sentence each. Include details such as whether you used asynchronous updates, and how you handled policy evaluation in PI. (2 marks)

b) Pick three representative testcases to compare the performance of VI and PI, reporting the numerical values for the following performance measures:

- Time to converge to the solution. (3 marks)
- Number of iterations to converge to the solution. (3 marks)

In order to do this, you'll need to modify the `# solver type` in your local copy of the test cases (the text files in the testcases directory).
You should report the numerical results in a neatly formatted table (not screenshots from your computer or Gradescope).

c) Discuss the difference between the numbers you found for VI and PI, including any differences observed between testcases. Explain and provide reasons for why the differences either make sense, or do not make sense. (7 marks)

Question 3. Investigating optimal policy variation **(10 marks)**

One consideration in the solution of a Markov Decision Process (i.e. the optimal policy) is the trade off between a risky higher reward vs a lower risk lower reward, which depends on the probabilities of non-deterministic dynamics of the environment and the rewards associated with certain states and actions.

Consider testcase `ex6.txt`, which includes a risky (but lower cost) path through the top half of the grid, and a less risky (but higher cost) path through the bottom half of the grid. Explore how the policy of the agent changes with `thorn penalty` and `transition probabilities`.

If you did not implement PI, you may change the solver type to VI in order to answer this question.

a) Describe how you expect the optimal path to change as the `thorn penalty` and `transition probabilities` change. Use facts about the algorithms and Bellman optimality equation to justify why you expect these changes to have such effects. (5 marks)

b) Picking three suitable values for `thorn penalty` and three suitable values for the `transition probabilities`, explore how the optimal policy changes over the 9 combinations of these factors. Present the results in a table, clearly denoting the optimal behaviour (i.e. top/risky, bottom/safe, something else) for each combination. Do the experimental results align with what you thought should happen? If not, why?

(5 marks)

References and Generative AI

At the end of your report, you must cite any references (e.g. using IEEE or APA referencing style). If you utilised Generative AI to assist in producing your code or report, briefly describe what tool you used and how you used it, citing the version and date. e.g. "This work was corrected using Copilot (Microsoft, `https://copilot.microsoft.com/`) on 30 July 2024." See the following link: `https://guides.library.uq.edu.au/referencing/chatgpt-and-generative-ai-tools`, and read the Academic Misconduct section for further details.

Academic Misconduct

The University defines Academic Misconduct as involving "a range of unethical behaviours that are designed to give a student an unfair and unearned advantage over their peers." UQ takes Academic Misconduct very seriously and any suspected cases will be investigated through the University's standard policy (`https://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct`). If you are found guilty, you may be expelled from the University with no award.

It is the responsibility of the student to ensure that you understand what constitutes Academic Misconduct and to ensure that you do not break the rules. If you are unclear about what is required, please ask.

In the coding part of COMP3702 assignments, you are allowed to draw on publicly-accessible resources and provided tutorial solutions, but you must make reference or attribution to its source, in comments next to the referenced code, and include a list of references you have drawn on in your `solution.py` docstring.

If you have utilised **Generative AI** tools such as ChatGPT, you must clearly cite any use of generative AI in each instance. To reference your use of AI see:

- https://guides.library.uq.edu.au/referencing/chatgpt-and-generative-ai-tools

Failure to reference use of generative AI tools constitutes student misconduct under the Student Code of Conduct.

It is the responsibility of the student to take reasonable precautions to guard against unauthorised access by others to his/her work, however stored in whatever format, both before and after assessment. You must not show your code to, or share your code with, any other student under any circumstances. You must not post your code to public discussion forums (including Ed Discussion) or save your code in publicly accessible repositories (check your security settings). You must not look at or copy code from any other student.

All submitted files (code and report) will be subject to electronic plagiarism detection and misconduct proceedings will be instituted against students where plagiarism or collusion is suspected. The electronic plagiarism detection can detect similarities in code structure even if comments, variable names, formatting etc. are modified. If you collude to develop your code or answer your report questions, you will be caught.

For more information, please consult the following University web pages:

- Information regarding Academic Integrity and Misconduct:
  - https://my.uq.edu.au/information-and-services/manage-my-program/student-integrity-and-conduct/academic-integrity-and-student-conduct
  - http://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct
- Information on Student Services:
  - https://www.uq.edu.au/student-services/

Late submission

Students should not leave assignment preparation until the last minute and must plan their workloads to meet advertised or notified deadlines. It is your responsibility to manage your time effectively.

It may take the autograder up to an hour to grade your submission. It is your responsibility to ensure you are uploading your code early enough and often enough that you are able to resolve any issues that may be revealed by the autograder *before the deadline*. Submitting non-functional code just before the deadline, and not allowing enough time to update your code in response to autograder feedback is not considered a valid reason to submit late without penalty.

Assessment submissions received after the due time (or any approved extended deadline) will be subject to a late penalty of 10% per 24 hours of the maximum possible mark for the assessment item.

In the event of exceptional circumstances, you may submit a request for an extension. You can find guidelines on acceptable reasons for an extension here https://my.uq.edu.au/information-and-services/manage-my-program/exams-and-assessment/applying-extension, and submit the UQ Application for Extension of Assessment form.