Assignment 1

Introduction

- Download the code for this assignment and then unzip the archive.
- This assignment uses Python 3. Do not use python 2.
    - We have tested the assignment with Python 3.11.4.
- You can work on the assignment using your favourite python editor. We recommend VSCode.
- Post any questions or issues with this assignment to our discussion forum.

Problem 1: DFS-GSA (Weight: 20%)

In this part of the assignment you are going to implement
- a parser to read a search problem in the file `parse.py`, and
- Depth First Search (DFS) - Graph Search Algorithm (GSA) in the file `p1.py`

Both these python files have already been created. Do not change anything that has already been implemented. Our auto grader (`grader.py`) may expect on the existing code that you are not supposed to change.

IMPORTANT: Do not make changes to the file `grader.py`.

To test if your code passes the first test case, you may run

```
(base) scdirk@Dirks-Air a1 % python p1.py 1
Grading Problem 1 :
----------> Test case 1 PASSED <----------
```

The parameter `1` here specifies the test case number. You may omit it to run all test cases for this problem.
As you can see we have passed test case 1 here. This is because we have hardcoded the solution for you in this function.

```
def dfs_search(problem):
   #Your p1 code here
   solution = 'Ar D C\nAr C G'
   return solution
```

Note that this is exactly what the test case expects. To find out what the test case expects, you can open the file `test_cases/p1/1.sol`.

```
Ar D C
Ar C G
```

As you can see, the content of the file `test_cases/p1/1.sol` is identical to the return value of the function `dfs_search(problem)`.

Note that `test_cases/p1/1.sol` and other solution files consist of two lines of text that represent the following.

- Exploration order (i.e., the order in which states are added to the explored set)
- solution path (i.e., the first solution found)

You should return these two lines of text in the `dfs_search(problem)`.

The exploration order is `Ar`, `D`, `C` and the solution path is `Ar`, `C`, `G`. The search problem definition can be found in the file `test_cases/p1/1.prob`, which we will introduce shortly.

Let's try another test case by changing the argument to the python program to `2`.

```
(base) scdirk@Dirks-Air a1 % python p1.py 2
Grading Problem 1 :
----------> Test case 2 FAILED <----------
Your solution
Ar D C
Ar C G
Correct solution
A D
A D G
```

We failed this test case. The correct solution can be found in `test_cases/p1/2.sol.`

```
A D
A D G
```

We will have to look at the `*.prob` files in the `test_case/p1/` folder to load the problem definition and then determine a corresponding solution. The `*.prob` files define weighted directed graphs with a single start state, a list of goal states, heuristics and arcs. For example, consider the problem definition of the first test case defined in the file `test_cases/p1/1.prob`.

```
start_state: Ar
goal_states: G
Ar 0
B 0
C 0
D 0
G 0
Ar B 1.0
Ar C 2.0
Ar D 4.0
C G 8.0
```

The search problem is specified as follows.

line 1: start state
line 2: list of goal states separated by a space (order not relevant)
line 3 … (n+2): (n = number of states) heuristic for each state (order not relevant)

```
<state> <heuristic>
```

line (n+3) … end: state transitions of the form (order is relevant)

```
<start state> <end state> <cost>
```

Note that we don't need the transition cost nor the heuristic for solving problem 1. However, you should implement the parsing for everything now so that you don't have to make modifications later. HINT: We made use of the following string functions `strip()`, `split()` and `replace()` in our implementation.

You should decide on an appropriate data structure for the problem and return it from the following function in `parse.py`.

```
def read_graph_search_problem(file_path):
     #Your p1 code here
     problem = ''
     return problem
```

Once your implementation of both `read_graph_search_problem(file_path)` and `dfs_search(problem)` is complete, check that you pass the first test case.

```
(base) scdirk@Dirks-Air a1 % python p1.py 1
Grading Problem 1 :
----------> Test case 1 PASSED <----------
```

You may check if you pass all test cases as follows.

```
(base) scdirk@Dirks-Air a1 % python grader.py 1
Grading Problem 1 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
```

You may import anything from the Python Standard Library. IMPORTANT: Do not import packages that are not part of this library, such as NumPy.
Make sure that you pass all provided test cases before moving on to the next question. When marking your code, we will …

- use novel testcases to ensure that you are actually solving the problem and not merely get around it by tricking the auto grader, and
- read your code and comments that you have written.

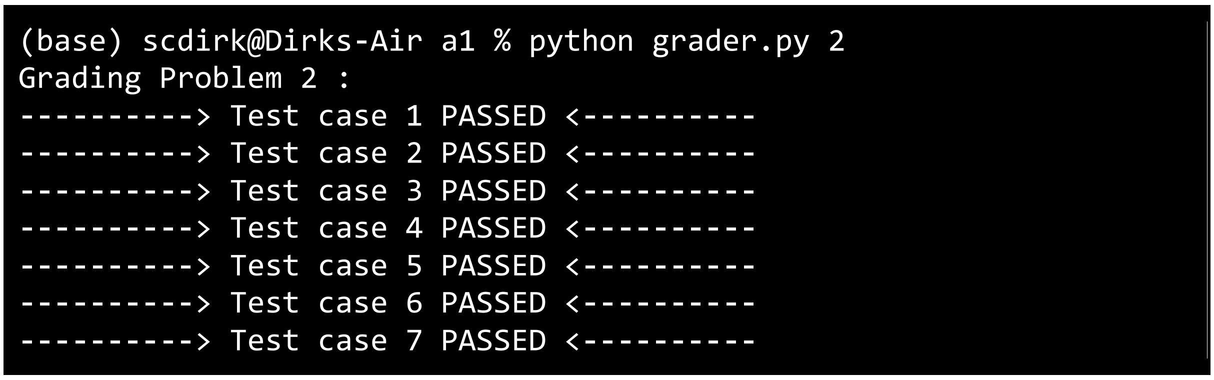
To solve this question, you may start with the code provided to you in the lecture and make appropriate modifications to it.

Problem 2: BFS-GSA (Weight: 5%)

In this part of the assignment you are going to implement Breadth First Search (BFS) - Graph Search Algorithm (GSA) in the file `p2.py`.

HINT: The solution to this problem should be almost identical to the solution to Problem 1.

Once done, check if you pass all test cases as follows.

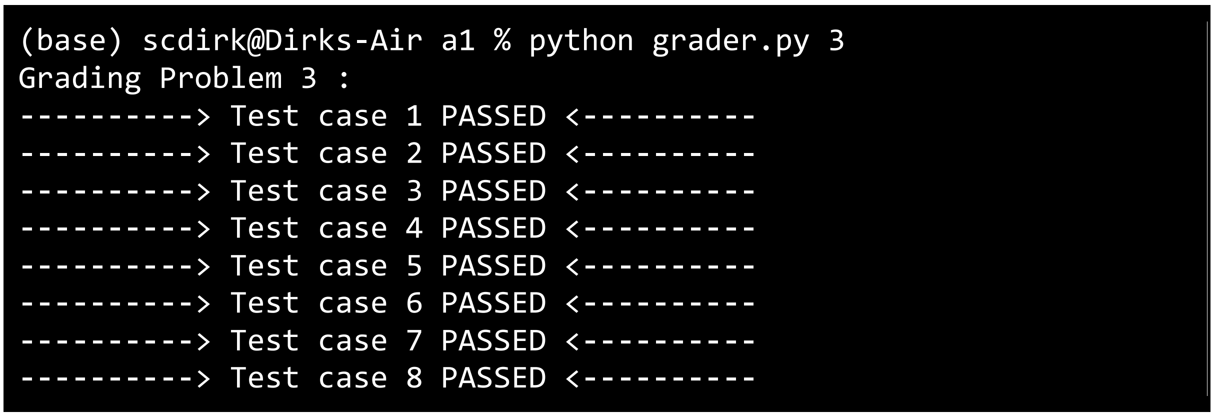


```
(base) scdirk@Dirks-Air a1 % python grader.py 2
Grading Problem 2 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
```

Problem 3: UCS-GSA (Weight: 10%)

In this part of the assignment you are going to implement Uniform Cost Search (UCS) - Graph Search Algorithm (GSA) in the file `p3.py`.

Once you have done this, check if you pass test cases as follows.



```
(base) scdirk@Dirks-Air a1 % python grader.py 3
Grading Problem 3 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
----------> Test case 8 PASSED <----------
```

Problem 4: Greedy (Weight: 10%)

In this part of the assignment you are going to implement Greedy Search - Graph Search Algorithm (GSA) in the file `p4.py`.

Once you have done this, check if you pass test cases as follows.

```
(base) scdirk@Dirks-Air a1 % python grader.py 4
Grading Problem 4 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
```

Problem 5: A* (Weight: 10%)

In this part of the assignment you are going to implement A* - Graph Search Algorithm (GSA) in the file `p5.py`.

Once you have done this, check if you pass test cases as follows.

```
(base) scdirk@Dirks-Air a1 % python grader.py 5
Grading Problem 5 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
```

Problem 6: 8 Queens Local Search - Number of Attacks (Weight: 15%)

The last three problems of this assignment use a different problem. Consider the content of the file `test_cases/p6/1.prob.`

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . q . . . .
q . . . q . . .
. q . . . q . q
. . q . . . q .
. . . . . . . .
```

It defines a state of the 8 Queens problem. In this problem you will load the problem in the following function of the file `parse.py`.

```
def read_8queens_search_problem(file_path):
     #Your p6 code here
     problem = ''
     return problem
```

Next, you will determine the attacks for each square in the following function of the file `p6.py`.

```
def number_of_attacks(problem):
     #Your p6 code here
```

The number of attacks for a particular square is defined as the number of direct and indirect attacks of other queens to that square, assuming the queen in the same column would move to that square.

For the first test case the correct number of attacks can be found in the file `test_cases/p6/1.sol` and is as follows.

```
18 12 14 13 13 12 14 14
14 16 13 15 12 14 12 16
14 12 18 13 15 12 14 14
15 14 14 17 13 16 13 16
17 14 17 15 17 14 16 16
17 17 16 18 15 17 15 17
18 14 17 15 15 14 17 16
14 14 13 17 12 14 12 18
```

Once you have implemented both functions correctly you should be able to pass all test cases.

```
(base) scdirk@Dirks-Air a1 % python p6.py
Grading Problem 6 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
```

Problem 7: 8 Queens Local Search - Get a Better Board (Weight: 15%)

In this problem you will return a better board in the function `better_board` of the file `p7.py`. The better board moves one queen to the best position (i.e., lowest number of attacks). Note that queens can only move to a position in the same column. If there are multiple best positions, you should select the first best position found if iterating row by row starting in the upper left.

Consider the first test case available in the file `test_cases/p7/1.prob`

```
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . q . . . .
q . . . q . . .
. q . . . q . q
. . q . . . q .
. . . . . . . .
```

and its corresponding solution available in the file `test_cases/p7/1.sol`.

```
. q . . . . . .
. . . . . . . .
. . . . . . . .
. . . q . . . .
q . . . q . . .
. . . . . q . q
. . q . . . q .
. . . . . . . .
```

Note that you may import existing code such as helper functions from your `p6.py` solution.

Once you are done, check for correctness as follows.

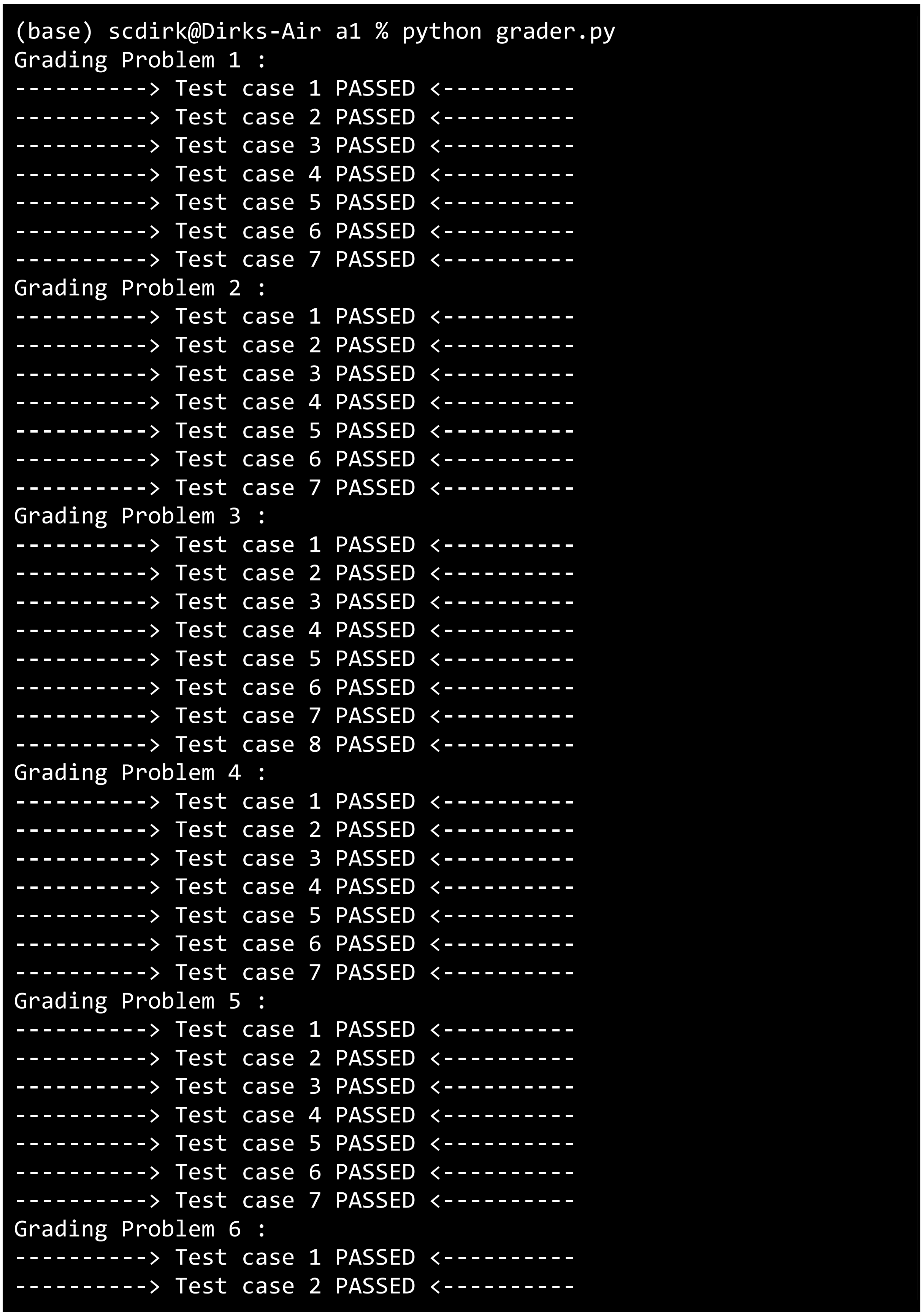
```
(base) scdirk@Dirks-Air a1 % python p7.py
Grading Problem 7 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
```

Problem 8: 8 Queens Local Search – All solutions (Weight: 15%)

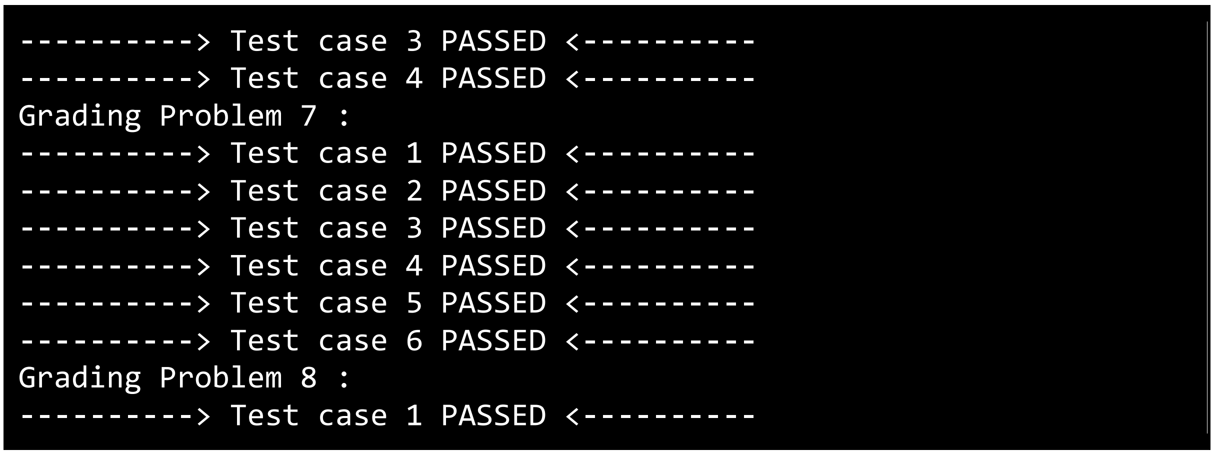
Import the `better_board_array` and `calc_attacks` functions from Problems 6 – 7 (and any other helper functions you need) and implement local search in the function `all_solutions` of Problem 8. You should continue to run local search until all 92 unique solutions have been found. Return all solutions in sorted order using `sorted(solutions)`, where `solutions` is a set of all solutions in string format.
You can check the output requirements in the file `p8/1.sol`. Note that there is no input file for this problem.
Fun fact: This problem focuses on 8 queens with 92 solutions. Note that n-Queens with n=20 has almost 40 billion solutions.

Congratulations you have completed this assignment.

You may also use the following command to make sure you pass all test cases for this assignment.



```
(base) scdirk@Dirks-Air a1 % python grader.py
Grading Problem 1 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
Grading Problem 2 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
Grading Problem 3 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
----------> Test case 8 PASSED <----------
Grading Problem 4 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
Grading Problem 5 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
----------> Test case 7 PASSED <----------
Grading Problem 6 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
```



```
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
Grading Problem 7 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
Grading Problem 8 :
----------> Test case 1 PASSED <----------
```

Short Written Report

Write a short (at most two A4 pages) written report in PDF format explaining which problem were completed and where you have struggled. Also write down the runtime (if $>$ 1s) for each problem. Write down the approximate number of hours you have spent per questions for this assignment.

Submission

To submit your assignment to Moodle, `*.zip` the following files ONLY:

- `p1.py`
- `p2.py`
- `p3.py`
- `p4.py`
- `p5.py`
- `p6.py`
- `p7.py`
- `p8.py`
- `parse.py`
- `report.pdf`

Do not zip any other files. Use the `*.zip` file format. Name the file `UID.zip`, where UID is your 10 digit university number. Make sure that you have submitted the correct files and named all files correctly. We will deduct up to 5% for files with incorrectly file names. We will not allow late submissions. Submission of incorrect files may result in 0 marks.

Check that you have submitted the correct files before the deadline.