

National University of Singapore  
School of Computing  
CS3243 Introduction to Artificial Intelligence

**Project 2.1: Constraint Satisfaction Problems**

Issued: 18 September 2023

Due: 8 October 2023

## 1 Overview

In this project, you will **implement a solver for a general constraint satisfaction problem (CSP)**. You are to use the **backtracking algorithm**, along with any other optimisations you deem necessary (eg: forward checking, AC-3, etc).

---

**This project is worth 3% of your module grade.**

---

### 1.1 General Project Requirements

The general project requirements are as follows:

- **Individual** project, but you are *allowed to consult P2ST (ChatGPT) App*.
- Python Version:  $\geq 3.10$
- Deadline: **8 October 2023**
- Submission folder: **Canvas > CS3243 > Assignments > Project 2.1**

More details can be found in Section 4.

### 1.2 Academic Integrity and Late Submissions

Note that any material that does not originate from you (e.g., is taken from another source, with the exception of the P2ST (ChatGPT) App) should not be used directly. You should do up the solutions on your own. Failure to do so constitutes plagiarism. Sharing of materials between individuals is also strictly not allowed. Students found plagiarising or sharing their code will be dealt with seriously.

For late submissions, there will be a 20% penalty for submissions received within 24 hours after the deadline, 50% penalty for submissions received between 24-48 hours after the deadline, and 100% penalty for submissions received after 48 hours after the deadline. For example, if you submit the project 30 hours after the deadline and obtain a score of 92%, a 50% penalty applies and you will only be awarded 46%.

## 2 Project 2.1: CSP Solver

### 2.1 Task: Backtracking algorithm

#### 2.1.1 Functionality

You will be given a set of variables, and their associated domains. You will also be given some constraints between the variables. The objective is to find a set of variable assignments such that all constraints are satisfied.

#### 2.1.2 Input

You will be given a `dict`. The dict is guaranteed to have two keys, “domains” and “constraints”.

The value corresponding to “domains” will be another `dict` containing `str : list[int]` elements. The keys of this dictionary corresponds to names of the variables, and the values corresponds to the domain of each variable.

The value corresponding to “constraints” will be yet another `dict` containing `(str, str) : function` elements, where each element corresponds to a binary constraint. Thus, the keys for each element in this “constraints” dictionary corresponds to a pair of variables defining the scope of a constraint, while its value corresponds to the relationship binding the two variables specified in the key, which is specified in the form of a lambda function, `lambda x, y : <logical expression>`. This function will take in two integers, and return `True` if the constraint is satisfied, and `False` otherwise.

An example input is given below in 2.1.5.

#### 2.1.3 Requirements

You will define a python function, named `solve_CSP(input)`. This function takes in the input dictionary described above, and must return a `dict` with `[str : int]` elements, where the key `str` corresponds to the given variable symbols, and the value `int` corresponds to an assignment. This output must correspond to a complete and consistent solution to the given CSP. When no such solution exists, then the output should be `None`. You may define any other convenience functions or classes you require.

#### 2.1.4 Assumptions

You may assume the following:

- All variables are strings, and all domain values are `ints`.

- All constraints will be binary constraints.
- There will be at most one constraint between each pair of variables. That means that if (“A”, “B”) appears as a key in the “constraints” dictionary, (“B”, “A”) will NOT be a key.
- No exceptions will be thrown by any constraint function as long as all function inputs satisfy their associated variable domains. More specifically, you do not have to check for division by zero error when checking constraints.
- If a key (“A”, “B”) appears in the constraints dictionary, the variable “A” will be the first argument of the constraint function, and the variable “B” will be the second argument.
- If the CSP has no solution, the function should return **None**.

### 2.1.5 Example

An example input is the following:

```
input = {
    'domains' : {
        'A' : [1,2,3,4,5],
        'B' : [2,3,4,5,6],
        'C' : [3,4,5,6,7],
        'D' : [5,7,9,11,13]
    },
    'constraints' : {
        ('A', 'B') : lambda a, b : a + b == 8 and a >= b,
        ('B', 'C') : lambda b, c : b <= c/2,
        ('C', 'D') : lambda c, d : (c + d) % 2 == 0
    }
}
```

One possible solution to the above CSP is the following:

```
output = {'A' : 5, 'B' : 3, 'C' : 7, 'D' : 5}
```

## 3 Grading

### 3.1 Grading Rubrics (Total: 3 marks)

Your code will be graded based on the following criteria.

- Correct implementation of backtracking algorithm by passing test cases correctly. (1m)
- Efficient implementation of backtracking algorithm by passing all test cases within the time limit. (2m)

### 3.2 Grading Details

#### 3.2.1 Correctness

We will run your code on a set of public and private test cases. There are 4 possible outputs when grading an implementation on a given test case:

- **Accepted (AC):** You have returned a solution passing all the requirements within the given time limit. You have **passed** the test case.
- **Wrong Answer (WA):** You have returned a solution failing at least one requirement within the given time limit. You have **not passed** the test case.
- **Time Limit Exceeded (TLE):** Your code runs beyond the given time limit. You have **not passed** the test case.
- **Runtime Error (RTE):** Your code has thrown an exception or caused an exception to be thrown. You have **not passed** the test case.

Different test cases may have different weights. You will get full credit for the implementation only if you obtain AC for all test cases using that implementation. For the specific points distribution, refer to Section 5.2.

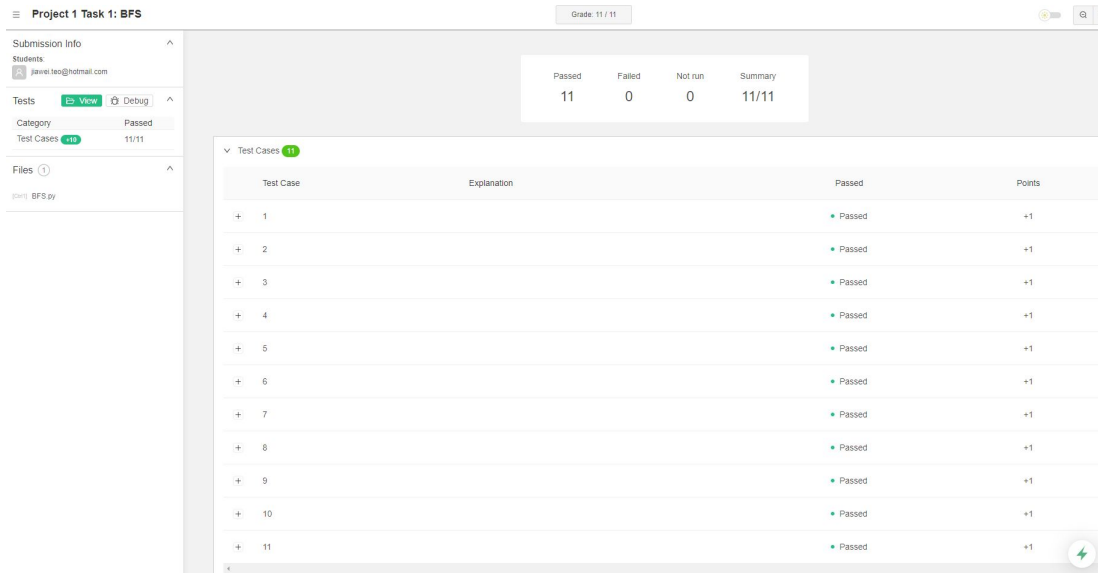
### 3.3 CodePost (Platform for Code Testing)

We will be using **CodePost** as our standardised platform for you to run and test your code on public and private test cases. You should have used CodePost to test your solutions in Project 1. Contact the course staff if you face any issues when accessing or uploading to CodePost during this project.

#### 3.3.1 Uploading to CodePost

1. For each task, click on “Upload assignment”, and upload your Python file with the following name: `backtracking.py` (DO NOT rename the files).

2. After the submission has been processed, refresh the page and select “**View feedback**”.
3. Ideally, if your implementation is correct and is within the runtime threshold, the output will look like this:



Project 1 Task 1: BFS

Grade: 11 / 11

Submission Info

Students

javier.tee@hotmail.com

Tests

View Debug

Category

Test Cases

Passed 11/11

Files

BFS.py

Test Case	Explanation	Passed	Points
+	1	Passed	+1
+	2	Passed	+1
+	3	Passed	+1
+	4	Passed	+1
+	5	Passed	+1
+	6	Passed	+1
+	7	Passed	+1
+	8	Passed	+1
+	9	Passed	+1
+	10	Passed	+1
+	11	Passed	+1

Figure 1: Codepost output

### 3.3.2 CodePost for Testing

CodePost hosts both the public test cases (which have been released via Canvas together with the skeleton implementation files) and the private test cases. The output in CodePost has been purposefully sanitised to prevent attempts at finding out the private test cases. As such, you are **expected to check your implementations thoroughly** via **custom-made test cases run locally**, as you will not find CodePost useful for debugging purposes.

Note that CodePost is for you to run and test your code on the test cases. Your solution will **not be graded** using CodePost; instead, your Canvas submission will be used and run on the same environment as CodePost. Therefore, passing all test cases in CodePost **does not guarantee** full credit. Some possible reasons for not obtaining full credit despite passing all test cases in CodePost include: being lucky in CodePost due to randomness in the algorithm, being caught plagiarising, etc. **We will check for any plagiarism and students found plagiarising will be dealt with seriously.**

Note that CodePost is run by an external organisation – we are not responsible for any downtime.

---

## 4 Submission

### 4.1 Submission Details via Canvas

- For this project, you will need to submit 1 Python file: `backtracking.py`.
- Place the `backtracking.py` file in a folder, name the folder as `studentNo` and zip it as `studentNo.zip`. An example will be: `A0123456Z.zip`.
- When unzipped, your submission file must contain the 1 file in a folder: **A0123456Z.zip → unzip → A0123456Z folder → backtracking.py**. There should not be any subfolders.
- Do not modify the file names.
- Please follow the instructions closely. **If your files cannot be opened or if the grader cannot execute your code, this will be considered failing the test cases as your code cannot be tested.**
- Submission folder: **Canvas > CS3243 > Assignments > Project 2.1**

You may submit your files as many times as you like, but only the latest one will be graded, **even if it means incurring a late penalty**.

### 4.2 P2ST (ChatGPT) App Usage

The P2ST (ChatGPT) app has been developed for CS3243 (this course). It is a tool that can be used to generate code from natural language descriptions. The objective is to help you better understand the contents of the course while skipping some of the more tedious parts of coding.

You are **permitted** to use the P2ST (ChatGPT) app to generate code to help you with this project. No other app or AI-generated code may be used.

The plagiarism-checking phase will be conducted using a tool that can identify code that has been copied from other sources. If your code is flagged as duplicated, you may appeal to the teaching team only if the code was generated from or with the help of the P2ST (ChatGPT) app. If the teaching team finds that the code was not generated using the P2ST (ChatGPT) app and is instead generated via other means like using other AI assistance tools, plagiarising other people's work, and more, the appeal will not be successful.

Note that the correctness and efficiency of the code generated by the app are not guaranteed. You are responsible for any submissions made.

## 5 Appendix

### 5.1 Allowed Libraries

The following libraries are allowed:

- Data structures: queue, collections, heapq, array, copy, enum, string
- Math: numbers, math, decimal, fractions, random, numpy
- Functional: itertools, functools, operators
- Types: types, typing

### 5.2 Points distribution

1. Correctness: each test case is worth 0.25 points. Total is 1 point.
2. Efficiency: each test case is worth 0.5 points. Total is 2 points.