Homework Three

Constraint Satisfaction

Introduction

In this homework assignment, you will write a basic scheduling agent for an air freight terminal. This is a simplification[1] of a real-world problem: creating a schedule for the loading and unloading of cargo.

[1] An extreme simplification.

Collaboration

You may, optionally, collaborate with one (1) other person on this assignment. You must explicitly state who this person is in the initial comments of your program. If you choose to do this, both of you must submit separately, and both of you will be responsible for completely understanding how your program works.

Submitting

Files to Submit: You will write one 'main' file: `terminalScheduler.py`. Once you have completed the assignment, submit this file to the submit server. **Please submit a single (uncompressed) `.tar` containing this file at the root of the tar, along with associated files you create.** (Your homework could possibly be completed with just this file.) Include a comment at the start of `terminalScheduler.py` indicating what help, if any, you got on this assignment. If you didn't get any help, indicate this. Failure to include this comment will result in losing 10 points on the assignment grade.

Evaluation: Evaluation of your code's correctness will be performed by against provided test cases and one "hidden" test case based on the problem spec. Passing provided test cases (without hard coding, if the problem is solved as a CSP) will result in at least 75/100.

Academic Dishonesty: This assignment is individual effort, with the option to work with one partner. Please review the collaboration policy for the course and adhere to it.

Early Submission

You'll get a **20-point extra-credit bonus** on this assignment if you submit it by 4 Mar. Grace days can't apply to this early deadline.

Otherwise, it's due 15 Mar, and grace days *can* apply to that deadline.

Autograder

Because any grading script would also be a partial solution, no local autograder is provided. At-home test cases are provided: four of the tests should "pass" (can be solved) and the fifth should "fail" (cannot be solved).

The Problem

An air cargo terminal receives aircraft loaded with freight, unloads the aircraft, sorts the cargo, and transfers the cargo to trucks for distribution.

Arrivals

Aircraft arrive on a schedule. Each arriving aircraft has:

- A unique name
- An arrival time
- A cargo manifest:
  - The manifest indicates how many cargo pallets are carried on the aircraft

Scheduling:

- Aircraft must be scheduled to unload at hangars at specific times.
- Once an aircraft arrives at a hangar, it remains until unloaded
  - No aircraft can unload before arriving at a hangar
- Aircraft can wait for a hangar after landing at the terminal (be at "no hangar" between landing at the terminal and arriving at a hangar)

Hangars

Aircraft are unloaded at hangars. Each terminal hangar can handle one aircraft at a time.

After pallets are unloaded, they are stored at the hangar until they are subsequently loaded onto a cargo truck

Forklifts

Aircraft are unloaded by forklifts. Each forklift can unload one pallet in 20 minutes.

- Any number of forklifts can work to unload a plane at any time
- One pallet can only be unloaded by one forklift
- Multiple pallets can be unloaded from a plane simultaneously
  - Each must be unloaded by a separate forklift
- The aircraft must remain at the hangar until unloading is complete

Scheduling:

- Forklifts must be scheduled to work at hangars at specific times.
- Forklifts must be scheduled to *unload* (from an aircraft) or *load* (to a truck) (see below)

Trucks

Each cargo truck can receive one pallet. Cargo trucks are loaded by forklifts, similar to how aircraft are unloaded by forklifts. Each forklift can load one pallet in 5 minutes:

- A cargo truck can be loaded by one forklift in 5 minutes
- A cargo truck can be loaded by two forklifts in 5 minutes
  - The pallet is already being loaded, so the second forklift has nothing to do
- Each hangar can accommodate one truck at any given time
- Assume the truck leaves as soon as it is loaded

Scheduling:

- Cargo trucks must be scheduled to be loaded at hangars at specific times.
- The truck can arrive at the hangar before loading
  - The truck can't load until it has arrived at a hangar
- Trucks can wait at the terminal (at no hangar) before arriving at a hangar to load

Scheduling

- Each aircraft must be scheduled to arrive a specific hangar at a specific time
  - The time must be at the same time or after the plane arrives at the terminal
  - The aircraft must be scheduled to depart from the hangar after it is finished unloading
- Forklifts must be assigned to a specific hangar
  - Each forklift must be assigned to *load* or *unload* at a specific hangar at a specific time
  - A forklift can only unload if there is a plane with cargo waiting to be unloaded
  - A forklift can only load if there is a truck waiting to be loaded and a pallet to load

Time is always expressed as minutes on a 24h clock, i.e. `900` is five minutes after `855` and `1300` is five minutes after `1255`. Events should be scheduled to the closest five minutes (e.g. `1000`, `1005`, `1010`) and not more granularly. "Movement" can be considered instantaneous, e.g., if a plane departs a hangar at some time, another plane can arrive at the exact same time; if a forklift finishes unloading/loading at some time, it can begin another unload/load (even at another hangar) at that exact same time.

The scheduling problem is solved if an assignment of aircraft, forklifts, and trucks results in all cargo being unloaded from aircraft and loaded onto trucks by the end of the day.

Data Format - Problem

The `meta.json` file indicates what hangars are available and what forklifts are available. There is also a start time and a stop time for the day's work: all jobs must be scheduled after the start time and before the stop time.

Listing 1 `meta.json`

```
{
    "Start Time": 730,
    "Stop Time": 1100,
    "Hangars": ["North Hangar", "South Hangar"],
    "Forklifts": ["F01", "F02"]
  }
```

- Hangars will be named via an arbitrary unique string.

The `aircraft.json` file indicates the arrival schedule of aircraft:

Listing 2 `aircraft.json`

```
{
    "D213": {
      "Time": 805, "Cargo": 2
      },
    "AT-LX-2Y2": {
      "Time": 815, "Cargo": 1
      },
    "NW-19-1A": {
    "Time": 945, "Cargo": 1
    }
  }
```

- Aircraft will be named via an arbitrary unique string.

The `trucks.json` file indicates the arrival schedule of trucks:

Listing 3 `trucks.json`

```
{
    "JLX-9828": 745,
    "PK2-341": 735,
    "ADR-005": 1020,
    "TX-241-Y": 755
  }
```

- Trucks will be named via an arbitrary unique string.

> **Data Format - Solution**
>
> Your solution should be JSON in the following format:

Listing 4 `schedule.json`

```
{
    "aircraft": {
        "D213": {
            "Hangar": "North Hangar",
            "Arrival": 810,
            "Departure": 830
        },
        "AT-LX-2Y2": {
            "Hangar": "North Hangar",
            "Arrival": 830,
            "Departure": 850
        },
        "NW-19-1A": {
            "Hangar": "North Hangar",
            "Arrival": 950,
            "Departure": 1010
        }
    },
    "trucks": {
        "JLX-9828": {
            "Hangar": "North Hangar",
            "Arrival": 830,
            "Departure": 835
        },
        "PK2-341": {
            "Hangar": "North Hangar",
            "Arrival": 835,
            "Departure": 840
        },
        "ADR-005": {
            "Hangar": "North Hangar",
            "Arrival": 1020,
            "Departure": 1025
        },
        "TX-241-Y": {
            "Hangar": "North Hangar",
            "Arrival": 850,
            "Departure": 855
        }
    },
    "forklifts": {
        "F01": [
            {
                "Hangar": "North Hangar",
                "Time": 810,
                "Job": "Unload"
            },
            {
                "Hangar": "North Hangar",
```

If the problem cannot be solved, your solution should be:

Listing 5 `solution.json`

```
{
    "aircraft": null,
    "trucks": null,
    "forklifts": null
}
```

You do **not* need to "partially" solve a problem that cannot be fully solved.

How To Solve It

- First, frame the problem as an assignment of values to variables:
  - You will need to determine what the variables are
  - You will need to determine what the domains of each variable are
  - You will need to express the constraints
- Implement an algorithm that assigns values to variables, consistent with constraints
  - Naive recursive backtracking search is the easiest method
  - You are welcome to use inferences such as AC3
  - You could also try a min-conflicts approach
  - Approaches less efficient than basic backtracking search may time out my grader, and will not receive credit
    - If you can pass all the "pass" test cases inside of 30 minutes using a recent laptop, you'll be fine
- **Do not** use any libraries that solve constraint satisfaction or SAT problems (or anything similar)

Implementation

- Your scheduler should run from `terminalScheduler.py`
  - You can write other modules if you like
- `terminalScheduler.py` should require four command-line arguments in this order:

- `-meta_path` for the `meta.json` file
- `-aircraft_path` for the `aircraft.json` file
- `-trucks_path` for the `trucks.json` file
- `-schedule_path` for the name of the output `schedule.json`

Calling:

```
terminalScheduler.py META_PATH AIRCRAFT_PATH TRUCKS_PATH SCHEDULE_PATH
```

should run your scheduler on the specified input files and write to the specified output file.