

## Requirements

- *Due date:* November 22, 2023 at 11:55PM. This is an individual assignment. This assignment is weighted 15% of the course grade.
- Each student submits three files: `assignment3.pdf`, `matching.lp`, `frogs.lp`.

## Social Choice Theory

Consider the school choice problem in which students have strict preferences over schools and schools have strict priorities over students. Students are allowed to express certain schools as unacceptable (a student would rather be unmatched than be matched to an unacceptable school). We are interested in finding a matching that satisfies justified envy-freeness such that the matching has as many pairs as possible.

Reminder from Week 4.

*Justified envy-freeness:* there exists no agent  $i$  who prefers another school  $s$  over her match and  $s$  admits  $j$  a lower priority agent than  $i$ .

**Unmatched pairs.** A matching also includes unmatched students; therefore, if an agent  $i$  is unmatched in the matching, this implies that no school has admitted agent  $i$ . A *justified envy-free* matching should also ensure that no unmatched agent is left *envious* (i.e. there additionally exist no unmatched agent  $i$  who prefers another school  $s$  over not getting matched, and  $s$  admits  $j$  who is a lower priority agent than  $i$ ).

- [1.1] Is the Student Proposing Deferred Acceptance algorithm (described in Week 4) a suitable approach to solving this problem? If yes, prove it in one or two sentences (or by referring to the appropriate theorem). If not, then explain why not in one or two sentences.
- [1.2] The instance will be provided to your program via three predicates.

The predicates `schoolRank/3` and `studentRank/3` indicate the preference of school over students and students over school respectively. For instance `schoolRank(randwick, jane, 1)` means that Randwick School would love to have Jane as student, as she is ranked first in their list. If a student finds a school unacceptable, then there is no `studentRank` statement for that student for that school (e.g., the situation of Jane in the example below who is not ranking Maroubra School because it's unacceptable to her). The predicate `quota/2` indicates the number of students that can be accepted by some school.

The solution should be output using a predicate `match/2` taking a school  $s$  and a student  $p$  arguments to indicate that  $p$  is matched with  $s$ , for instance `match(randwick, jane)`.

Consider the following instance

```
schoolRank(randwick, jane, 1).
schoolRank(randwick, kurt, 2).
```

```

schoolRank(randwick, lisa, 3).
schoolRank(randwick, maud, 4).
schoolRank(maroubra, jane, 1).
schoolRank(maroubra, maud, 2).
schoolRank(maroubra, lisa, 3).
schoolRank(maroubra, kurt, 4).
studentRank(kurt, randwick, 1). studentRank(kurt, maroubra, 2).
studentRank(lisa, randwick, 1). studentRank(lisa, maroubra, 2).
studentRank(maud, randwick, 1).
quota(randwick, 2).
quota(maroubra, 2).

```

Find a justified-envy-free matching of largest cardinality in the above instance, and return it using the predicate `match/2`.

- [1.3] Design an ASP program named `matching.lp` that identifies a largest justified-envy-free matching on any input instance.

## Answer Set Programming

The problem is based on the Crazy Frog Puzzle and is inspired by the Logic and Constraint programming contest LP/CP2021 <https://github.com/alviano/lpcp-contest-2021/tree/main/problem-1>.

You have the control of a little frog, capable of very long jumps, and today is your lucky day!

The little frog just woke up in an  $S \times S$  land, with few obstacles and a lot of insects. Time to eat them all! The frog can jump as far as you want within the land, but only in the four cardinal directions (don't ask why) and you cannot land on any obstacle or already visited places (you will see why). Don't leave any insect alive!

**Input format** An input file contains a predicate instance `size/1` indicating the size of the land. The coordinates of the place where you woke up in a predicate `start/2`. And a predicate `obstacle/2` where each instance indicates the presence of an obstacle at the coordinates.

The size  $S$  is guaranteed to be between 4 and 60.

**Output format** Your program should compute a model containing a predicate `jump/3` indicating for each time step (starting at 1) where the frog lands.

**Example** Consider the following instance.

```

size(4).
obstacle(2,1).

```

```
obstacle(4,2).
obstacle(1,3).
obstacle(2,4).
start(3,3).
```

One possible solution would contain the predicate instances

```
jump(1,3,4)
jump(2,4,4)
jump(3,1,4)
jump(4,1,2)
jump(5,1,1)
jump(6,3,1)
jump(7,4,1)
jump(8,4,3)
jump(9,2,3)
jump(10,2,2)
jump(11,3,2)
```

The starting state of the instance as well as the successive steps represented by the solution are depicted in Figure 1.

- [2.1] Given a size  $S$ , a list of  $k$  obstacles, and assuming the frog starts in location  $(i, j)$ , is it possible to compute the number of jumps needed so that all non-obstacle location are visited exactly once? If yes, explain how and why. If not, then provide a concrete counter example with two solutions having a distinct number of jumps.
- [2.2] Consider the starting instance in Figure 2. Provide the ASP code describing this instance using the input format given above.
- [2.3] Consider the instance in Figure 2. Identify a solution, if any, and provide the list of jumps required in the output format described above. If there are no solutions, then state it and explain how you can prove/detect the absence of any solution.

You can identify the solution by any reasonable means you want (any reasonable means = non-cheating, non-collaborative, non-looking it up online, etc.), including human reasoning and manual computation, computer reasoning using ASP, running an appropriate algorithm in the programming language of your choice. If you find a solution, list it and then provide a very short sentence in English describing your approach. If you find that no solution exists, then explain your approach in details.

- [2.4] Give an ASP program named `frogs.lp` identifying solutions to any input instance and computing answer sets such that the projection of a model on the `jump/3` predicate constitute a solution list of jumps.

Figure 1: Crazy Frog Puzzle instance 0 and sequence of positions corresponding to a solution. The initial state is given in the top left picture, and then reading left to right then up to down, we have a sequence of states until all insects have been eaten without visiting the same place twice.



Figure 2: Crazy Frog Puzzle instance 1.

