Discrete Optimization Specialization: Assignment 4
The Royal Hunt

1 Problem Statement

Emperor Xian desires a hunt to be organized by Liu Bei to celebrate the emperor's birthday. To make the hunt a success, Liu Bei must carefully match the court members to their horses so that everyone enjoys the day. There may be more court members than horses, in which case some court members will not ride, but each horse must be used. There may be more horses than court members in which case each court member must ride. But there are certain unwritten rules the hunt must follow:

- The emperor must enjoy the day more than anyone else.
- All court members must ride unless there is no horse left available,
- If a court member holds a higher rank than another, then either (a) the beauty of their horse can be no less than that assigned to the other, (b) the lower rank member does not ride, or (c) both court members do not ride.
- If a horse is faster than another then either (a) the rider of the faster horse has no less riding ability than that of the slower horse, (b) the faster horse has no rider, or (c) both horses have no rider.

The aim is maximize the total enjoyment of the hunt over all riders. In truth the constraints are too hard to satisfy usually. So the last constraint can be violated for a penalty of 100 for each violation to the objective.

2 Data Format Specification

The input form for the Royal Hunt is a file named `data/royalhunt_`p`.dzn`, where p is the problem number, n being the number of court members (the first court member is the emperor), *rank* is an array mapping court members to rank (the higher the better), *ability* is an array mapping court members to riding ability (the higher the better), m is the number of horses available, *beauty* is an array mapping horses to their beauty (the higher the better), *speed* is an array mapping horses to their speed, and *enjoy* is a two dimensional array mapping court members and horses to the enjoyment of the rider on that horse. If an entry in the array is negative, then the horse cannot be assigned to the court member.

The data declarations are hence:

```
int: n; % number of court members
set of int: COURT = 1..n;
int: emperor = 1;
array[COURT] of int: rank;
array[COURT] of int: ability;
```

```
int: m; % number of horses
set of int: HORSE;
array[HORSE] of int: beauty;
array[HORSE] of int: speed;

array[COURT,HORSE] of int: enjoy;
```

An example data file is

```
n = 6;
rank =    [8,5,5,4,2,2];
ability = [0,1,0,1,0,1];
m = 5;
beauty =  [1,5,3,8,8];
speed =   [6,3,5,4,4];

enjoy = [| 3,4,5,7,3
         | 1,6,3,4,8
         | 2,3,4,3,3
         | 9,6,2,3,4
         | 4,-1,3,3,3
         | 4,4,4,4,4 |];
```

which considers 6 court members and five horses. Your model's output should give the horse assigned to each rider (or 0 if they are assigned no horse), and the objective value. For example it might output (a not necessarily optimal solution)

```
horse = [4,2,5,3,1,0];
obj = -178;
```

There are two violations of the horse constraint: horse 1 is faster than horse 2 but its rider 5 has less ability than horse 2's rider 2; and similarly horse 1 is faster than horse 3 but its rider has less ability than horse 3's rider 4. The total enjoyment is $7 + 6 + 3 + 2 + 4 = 22$.

3 Instructions

Edit `royalhunt.mzn` to solve the optimization problem described above. Your `royalhunt.mzn` implementation can be tested on the data files provided. In the MiniZinc IDE, use the *Run* icon to test your model locally. At the command line use,

```
mzn-gecode ./royalhunt.mzn ./data/<inputFileName>
```

to test locally. In both cases, your model is compiled with MiniZinc and then solved with the GECODE solver.

Resources You will find several problem instances in the `data` directory provided with the handout.

Handin This assignment contains 7 solution submissions and 1 model submissions. For solution submissions, we will retrieve the best/last solution the solver has found using your model and check its correctness and quality. For model submissions, we will retrieve your model file (.mzn) and run it on some hidden data to perform further tests.

From the MiniZinc IDE, the *Submit to Coursera* icon can be used to submit assignment for grading. From the command line, `submit.py` is used for submission. In both cases, follow the instructions to apply your MiniZinc model(s) on the various assignment parts. You can submit multiple times and your grade will be the best of all submissions.[1] It may take several minutes before your assignment is graded; please be patient. You can track the status of your submission on the *programming assignments* section of the course website.

4 Technical Requirements

For completing the assignment you will need MiniZinc 2.1.x and the GECODE 5.0.x solver. Both of these are included in the bundled version of the MiniZinc IDE 2.1.2 (`http://www.minizinc.org`). To submit the assignment from the command line, you will need to have Python 3.5.x installed.

[1] Solution submissions can be graded an unlimited number of times. However, there is a limit on grading of **model submissions**.