Modeling Discrete Optimization Assignment: **Fox Geese Corn**

1 Problem Statement

Every one knows the puzzle of a farmer with a fox, a goose, and a bag of corn to take to the market. She has to cross the river with a boat that can carry one object, if she ever leaves the fox with the goose, the goose is eaten, or the goose with the corn the corn is eaten, and she has to get everything across the river. This is a generalization of that problem.

The farmer has f foxes, g geese, and c bags of corn on the west side of the river. She has a boat that can carry k objects (any mix of types is allowable) and the time to make t trips. Note that the first trip is west to east, then the second trip is east to west, then the third trip is west to east, etc. When ever the farmer leaves some goods alone on either side of the river then by the time she returns the following happens,

- if there is only one kind of good, nothing.
- if there are only foxes and corn, out of boredom one fox eats a bag of corn, its stomach explodes and it dies.
- if there are foxes and geese,
  - if there are more foxes than geese one fox dies in argument over geese, no geese die, and no geese eat any corn
  - if there are no more foxes than geese, each fox eats a goose, and no geese eat any corn.
- if there are no foxes but there is geese and corn,
  - if there is no more geese than corn each goose eats a bag of corn
  - otherwise all the geese fight, one dies and one bag of corn is eaten.

Once she has completed her last trip then the farmer can take the goods from the east side to the market where she receives pf for each fox, pg for each goose and pc for each corn. The aim is to maximize profit.

2 Data Format Specification

The input is given as a file `data/fgc_*.dzn`, where p is the problem number, in MiniZinc data format:

f = f;
g = g;
c = c;
k = k;
t = t;

pf = pf;
pg = pg;
pc = pc;

The solution should be given in MiniZinc data format, with three arrays of size t,

`fox = [` number of foxes taken in boat on each trip $1..t$ `];`
`geese = [` number of geese taken in boat on each trip $1..t$ `];`
`corn = [` number of bags of corns taken in each trip $1..t$ `];`
`trips =` number of trips actually made;
`obj =` profit made;

Note that the entries in the `fox`, `geese` and `corn` arrays should all be 0 for entries after `trips`.

The classic problem, assuming the max trips is 5, the price of the fox is 10, the goose 5 and the corn 2 is given by the data file

```
f = 1;
g = 1;
c = 1;
k = 1;
t = 5;
pf = 10;
pg = 5;
pc = 2;
```

The standard solution of taking the goose across first doesn't work in this version (since the fox eats the corn and dies). It is represented as

```
fox   = [0,0,0,0,0];
geese = [1,0,0,0,0];
corn  = [0,0,0,0,0];
trips = 1;
obj = 5;
```

The optimal solution is

```
fox   = [1, 0, 0, 0, 0];
geese = [0, 0, 1, 0, 0];
corn  = [0, 0, 0, 0, 0];
trips = 3;
obj = 15;
```

Tip: In these planning style problems sometimes it is difficult for the solver to find a feasible solution. In that case, you can help the solver to find feasible plans by slowly growing the time horizon (i.e. trips in this case). This is achieved by building a search strategy that increments the `trips` variables, e.g.

```
solve :: int_search([trips], input_order, indomain_min, complete)
      maximize obj;
```

3 Instructions

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

```
mzn-gecode ./foxgeesecorn.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 From the MiniZinc IDE, the *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 2.0.x and the GECODE 4.4.x solver. Both of these are included in the bundled version of the MiniZinc IDE 0.9.9 (`http://www.minizinc.org`). To submit the assignment from the command line, you will need to have Python 2.7.x installed.

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