Discrete Optimization Specialization: Assignment 6
Weighing an Elephant: Part 2

1 Problem Statement

Cao Chong has managed to get the elephant onto the boat, and marked the side of the boat where it was displaced in the water by the weight of the elephant. He now needs to move stones onto the boat in order to sink it to the same displacement and count the stones.

He has g guards to move the stones. They can carry some amount of stones easily without tiring, but they can carry more which will cause them to tire, and be unable to carry stones for a while. There is a limit on how many guards can move on the pier between the bank and the boat, and in any time step all of them have to move in the same direction: either from bank to boat or boat to bank. There is a limit of the number of people that can stay on the boat at any one time. Cao Chong needs to plan how to move at least E stones from the bank to the boat in the least number of steps.

2 Data Format Specification

The input form for Weighing an Elephant: part 2 is a file named `data/elephant2_*.dzn`, where p is the problem number, T is the maximum number of steps available, E is the minimum number of stones to be put on the boat, G is the number of guards available, *easy* is an array giving for each guard how many stones they can carry without tiring, *hard* is an array giving for each guard the maximum number of stones they can carry, *tired* is an array giving for each guard g the number of steps before they can carry again after carrying more then $easy[g]$ stones, p is the maximum number of guards on the pier between boat and bank at one time, and b is the maximum number of guards on the boat at one time.

The representation of the plan shows for each guard at each time step, what the guard does: either

−1: goes from boat to bank;

0: wait where they are (resting); or

n > 0: moves n stones from bank to boat.

The end time of the plan is the step where at least E stones are on the boat. After the end time each guard must only wait.

The data declarations and decisions are hence:

```
int: T; % maximum time allowed;
set of int: TIME = 1..T;

int: E; % weight of elephant in STONES;
set of int: STONE = 0..E;
```

```
int: G; % number of guards
set of int: GUARD = 1..G;
array[GUARD] of STONE: easy;
array[GUARD] of STONE: hard;
array[GUARD] of TIME: tired;

GUARD: p; % maximum people on pier;
GUARD: b; % maximum people on boat;

set of int: ACT = -1..E; % -1 = goto bank, 0 = wait, > 0 carry stones
int: wait = 0;
int: to_bank = -1;
array[GUARD,TIME] of var ACT: act;        % action at time t
var TIME: end;                            % end time;
```

An example data file is

```
T = 10;
E = 50;
G = 4;

easy = [5,5,5,5];
hard = [10,10,10,10];
tired = [3,3,3,3];
p = 3;
b = 4;
```

which considers 4 guards each of identical abilities, who can carry 5 stones without tiring, and up to 10 stones, but they then need 3 steps rest. The maximum number of guards on the pier is 3 and on the boat is 4.

Your output should be the number of steps used (`end`), and the actions of the guards at each time point (`act`). For example a solution to the above data might be

```
act = array2d(GUARD,TIME,[
10,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 5, -1, 10,  0,  0,  0,  0,  0,  0,  0,
 5, -1, 10,  0,  0,  0,  0,  0,  0,  0,
 0,  0, 10,  0,  0,  0,  0,  0,  0,  0]);
end = 3;
```

which shows the first guard carrying 10 stones, while guards 2 and 3 carry 5 in the first step. Guards 2 and 3 return in step 2, and in step 4 guards 2, 3 and 4 carry 10 stones. After 3 steps there are 50 stones on the boat. Note that after 3 steps the guards just wait.

For the data file

```
T = 10;
E = 40;
```

```
G = 2;

easy = [5,10];
hard = [10,15];
tired = [3,4];
p = 1;
b = 2;
```

an example solution is

```
act = array2d(GUARD,TIME,[
 0,  0,  5, -1, 10,  0,  0,  0,  0,  0,
15, -1,  0,  0,  0, 15,  0,  0,  0,  0]);
end = 6;
```

Note how guard 2 can still move back to the bank at step 2 even though they are still tired from a heavy carry at step 1.

Note that the output formatting for `act` here is not required. It is just to illustrate the solutions better.

3 Instructions

Edit `elephant2.mzn` to solve the optimization problem described above. Your `elephant2.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 ./elephant2.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 4 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.

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

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.