CS4626 Constraint Programming and Optimisation
2nd Continuous Assessment 2019/20

After graduation, doctors must spend time working in hospitals as *residents* under the supervision of senior medics in order become fully qualified to practice. Each hospital or department must organise a rota (work schedule) for a group of residents over a set of shifts. During some of these shifts, focused tutorials take place for different topics, and each resident must attend specific sessions in order to obtain qualifications. But the resident are also essential for the functioning of the hospital, and so each shift must have a minimum number of residents on duty. There are various regulations in place on the number of shifts worked in any one week, on minimum breaks between blocks of work, and on the need for regular full days off.

The hospital service is looking for a software solution to help schedule the work rotas for the residents.

In any given instance, a period of time (e.g. a week) to be scheduled is divided into shifts. Each resident must be assigned a set of shifts over the period. The number of residents assigned to each shift must be at least the specified minimum number. For each individual resident, the assigned shifts must contain enough tutorials of the required type to allow the resident to satisfy the qualification requirement. For each resident, each scheduled shift must appear in a block of no more than *M* successive shifts. There must be at least *b* unassigned shifts between any two blocks of scheduled shifts. During the entire period, there must be a sequence of at least *p* unassigned shifts. Finally, each resident must work at least *s* shifts during the period. The overall aim is to minimise the total number of worked resident shifts over the period (the breaks *b* and *p* ensure no-one works beyond the legal maximum, and any imbalance between residents will be evened out in later periods).

The hospital service wants to build the software in two stages.

Stage (1) should decide which residents work on which shifts, ensuring all the qualification requirements are met, ensuring that at least the minimum number of residents are on duty each shift, ensuring that each resident works the minimum number of shifts, and minimising the total number of scheduled resident shifts (i.e. if resident_1 works for 3 shifts, and resident_2 works for 5 shifts, then the total scheduled resident shifts is 8).

Stage (2) should extend the solution for (1) to ensure that every resident gets a break of at least *p* shifts in a row during the total period, each working shift is in a block of at most *M*, and that successive blocks are separated by at least *b* unassigned shifts.

For example, if we have 2 residents, and 2 qualifications {A,B}, such that resident_0 requires B and resident_1 requires A, where A is taught on shift 2 and B is taught on shift 4, and we have 7 shifts, with staffing requirements [1,2,1,1,1,1,1], where the maximum block length *M* is 2, the between block rest *b* is 1, and the extended break *p* is 2, and each resident must have at least 3 shifts, then one solution is:

	0	1	2(A)	3	4(B)	5	6	Total
resident_0 (B)	1	1	0	1	1	0	0	4
resident_1(A)	0	1	1	0	0	1	1	4
Total	1	2	1	1	1	1	1	8

but the following is not a solution:

	0	1	2(A)	3	4(B)	5	6	Total
resident_0 (B)	1	0	0	1	0	0	1	3
resident_1 (A)	0	1	1	0	1	1	0	4
Total	1	1	1	1	1	1	1	7

-- shift 1 does not have 2 residents, r0 does not get qualification B, and r1 does not have 2 free days in a row.

The assignment

1. Create a Choco class for modelling and solving problems as described above. Your class must be called "Residents<Your ID>.java", where <Your ID> is to be replaced by your student ID number, it must have an executable main method; and it must read data from input text files using the reader class described below. Your class should run without requiring any other classes, apart from the reader class below , the Choco package and the standard Java library classes. Do not include the Choco jar files. Remember that the idea is to describe the problem to Choco and then ask Choco to solve it. You can help Choco by telling it what search strategy to use.
2. Five sample text files `residents{0,1,2,3,4}.txt` are provided on the Assessment page. The file `residents0.txt` describes the problem instance on the previous page. The samples get steadily more difficult. Run your Choco model on each of them. Write a brief report as a comment at the end of your .java file describing your models, explaining any different ways of representing the problem that you considered and why you chose the one you did, and showing the results you get when you run your Java/Choco code on the data files.

A class `ResidentsReader.java` is provided, which will read the problem instance from a data file, and supports the following public getter methods:

```
public int getNumResidents(); //total number of residents - n
public int getNumShifts();    //total number of shifts - m
public int getNumQuals();     //total number of qualifications offered - q
public int[] getMinResidents(); //array of size m - min staff for each shift
public int[][] getQualsOffered(); //a qxm matrix of 0/1, where
                        //qualsOffered[i][j] means qual i offered on shift j
public int[][] getQualsNeeded();  //a nxq matrix of 0/1, where
                        //qualsNeeded[k][i] means resident k needs qual i
public int getMaxBlock();         //the maximum length of a block of shifts
public int getRestPeriod();    //minimum free shifts required between blocks
public int getBreakPeriod(); //must be a sequence of free shifts this long
public int getMinShifts();   //minimum shifts to be scheduled per resident
```

Marking scheme

Marks will be awarded for submitting a working Java/Choco program, for generating correct solutions for the two levels of the problem, for the quality of model, for good use of Choco facilities including search strategies and global constraints if needed, and for the explanation in the report. Marks will not be awarded for the quality of the Java code that supports the model, so don't waste time implementing a GUI to show the results (although if your code is inadequately commented and is difficult to understand, you may lose marks). If you do not manage to get a complete solution, you will get credit for how far you got, as long as you submit a working version and you explain in your report what parts you were able to implement. If you attempt to add constraints but cannot get them working, comment out those constraints – don't submit code that does not run.

Submission Deadline

All submissions must be a single .java file, emailed to *k.brown@cs.ucc.ie* by 5pm, Thursday 27th February.

Plagiarism

This submission is part of the formal assessment for the module, and so must be your own work. You should not submit anyone else's work (or part of their work) as if it were your own, and you should not be collaborating with each other on how to solve the problem. However, you are free to re-use, without acknowledgement, any piece of Java/Choco code that has been posted this year on the CS4626 web pages, or included in the Choco manual. If you use code obtained from anywhere else, you must give a citation for it in your report.