The objective of this assignment is to experiment with variations of a constraint model, as well as to let a program conduct the experiments and generate the tedious parts of the report: in CP, we usually deal with NP-hard problems, hence quantitative work is necessary. Also keep in mind that you need to document that you have conducted the requested experiments and drawn conclusions therefrom, rather than from thin air.

**Question 1: Revisiting \( n \)-Queens**

We have seen the probably most popular model of the \( n \)-queens problem (see [http://mathworld.wolfram.com/QueensProblem.html](http://mathworld.wolfram.com/QueensProblem.html)), based on three DISTINCT constraints, but without any implied constraints or symmetry-breaking constraints. This model is implemented, with smallest-domain variable selection and bottom-up value selection, in [http://www.it.uu.se/edu/course/homepage/consprog/ht12/homeworks/queens.cpp](http://www.it.uu.se/edu/course/homepage/consprog/ht12/homeworks/queens.cpp); this file is derived from the current Gecode distribution by dropping the Qt instructions and adding an experiment scaffold. Perform the following sequence of tasks:

A. Give the runtimes and numbers of failures of `queens` for \( n \in \{8, 13, 18, 23, 28, 33, 38, 43, 48\} \).

B. Are the runtimes and numbers of failures always increasing with \( n \) for `queens`? Why?

C. Consider the new choice of decision variables where each position on the chessboard is represented by a Boolean decision variable \( q_{ij} \) such that \( q_{ij} = 1 \) if and only if there is a queen on row \( i \) and column \( j \). Design the corresponding model (without any unasked features) and implement it in Gecode as a new program, called `queens01.cpp`.

D. List all uninteresting predefined variable selection heuristics\(^1\) of Gecode (upon self-studying the ones not seen in the course) for `queens01`, and explain why they are uninteresting.

E. Repeat the previous task for the value selection heuristics.

F. Motivate at least eight combinations\(^2\) of an interesting predefined variable selection heuristic with an interesting predefined value selection heuristic for `queens01`.

G. For each chosen combination, give the runtimes and numbers of failures of `queens01` for \( n \in \{8, 13, 18, 23, 28, 33, 38, 43, 48\} \).

\(^1\)At least eight uninteresting predefined heuristics, for teams of one person.
\(^2\)At least three combinations, for teams of one person.
H. Identify the seemingly best combination and explain why it beats the other combinations.

I. Compare the runtime and number of failures of the seemingly best combination for queens01 with those for queens, and explain any differences.

Common Instructions: All statistics are to be collected until the first solution, if any. If randomisation is involved in a heuristic, then report the average statistics over 10 independent runs. All runtimes are to be given in seconds, rounded up to a centi-second precision. Always use a time-out of one CPU minute. All statistics can be given in tabular or graphic form.

Question 2: Magic Hexagon

We have seen the magic square problem. We now consider a related problem, where the layout is hexagonal (see http://mathworld.wolfram.com/MagicHexagon.html):

A B C
D E F G
H I J K L
M N O P
Q R S

There are 19 unknowns. The goal is to find an assignment of integers within \( \{1, 2, \ldots, 19\} \) such that any two unknowns take distinct values and the sums of all rows and diagonals are the same:

\[
A + B + C = D + E + F + G = \cdots = A + D + H = B + E + I + M = \cdots = H + M + Q.
\]

Perform the following sequence of tasks:

A. Write a Gecode program called hexagon1.cpp that posts only the problem constraints above. Apply smallest-domain variable selection (called INT_VAR_SIZE_MIN in Gecode).

B. How many solutions are there for hexagon1?

C. Give the runtimes and numbers of failures of hexagon1.cpp under all combinations of all predefined value selection heuristics and the three consistency levels available in Gecode for the DISTINCT constraint.

D. Identify the seemingly best combination for hexagon1 of value selection heuristic and DISTINCT consistency level, and explain why it beats the other ones.

E. Derive logically implied constraints in a way similar to the one for the magic square problem. Add these constraints to hexagon1.cpp, giving a new program, called hexagon2.cpp.

F. Repeat Tasks B to D for hexagon2.

G. Are the seemingly best value selection heuristic and DISTINCT consistency level for hexagon2 the same as for hexagon1? Is hexagon2 more efficient? Is the first solution the same? Is the number of solutions the same? Explain your answers.

H. Identify the symmetries of the problem and state suitable symmetry-breaking constraints. Add these constraints to hexagon2.cpp, giving a new program, called hexagon3.cpp.

I. Repeat Tasks B to D for hexagon3.

J. Repeat Task G for hexagon3 versus hexagon2.

---

3 At least four predefined value selection heuristics, for teams of one person.
**Common Instructions:** All statistics are to be collected until the first solution. Since there is only one problem instance and the runtimes are very small, report the average statistics over 10 independent runs. All runtimes are to be given in seconds, rounded up to a milli-second precision. Always use a time-out of 5 CPU seconds. All statistics can be given in tabular or graphic form.

**Hints**

Consider using the MiniModel facility of Gecode. It is strongly recommended to let a program run all the experiments automatically and write the results in tabular form directly into text files that you import into your report (with \input{...} in LaTeX, see the source code of the demo report for an example) or plotting software. See [http://www.it.uu.se/edu/course/homepage/consprog/ht12/homeworks/queens.cpp](http://www.it.uu.se/edu/course/homepage/consprog/ht12/homeworks/queens.cpp) for inspiration on how to do that.

**Submission Instructions**

All program documentation and question answers (other than the programs) must be in a single report in PDF format; all other formats are rejected. Take Section 1.1 of the demo report as a strict guideline for structuring the documentation of a constraint program, as well as an indication of the expected quality of content. State the question and task number of each answer in the report. Write clear answers, code, and documentation. Justify all claims and answers, except where explicitly not required. State any assumptions you make. Thoroughly proof-read, spell-check, and grammar-check your report.

Comply strictly with any program filenames imposed by the questions, since we reserve the right to process your programs automatically.

Only one of the teammates submits the files via the Student Portal (whose clock may differ from yours) by the hard deadline given above.

**Grading**

If all questions have been seriously attempted and all requested programs exist in files with the imposed names, have the documentation prescribed in the demo report, compile without error under gcc, and produce correct outputs to some of our grading tests in reasonable time under version 3.7.3 of Gecode on one of the computers of the IT department, then you get an initial score of at least 1 point:

- If your programs pass most of our grading tests, then you get an initial score of 4 or 5 points, depending also on the quality of the report; you are not invited to the grading session and your initial score is your final score.

- If your programs fail many of our grading tests, then you get an initial score of 1 or 2 points, depending also on the quality of the report; you are invited to the grading session, where you can increase your initial score by 1 point into your final score.

Otherwise you get a final score of 0 points and fail this part of the course.