

# CS637: Backtracking Programming Project

November 26, 2004

In this project, you will get acquainted with the concept of *backtracking* as a search technique for finding the solution among a very large set of possibilities. The idea is to avoid examining the entire set of possible solutions, but eliminate whole classes of them if we can make sure they cannot yield the right answer.

## 1 The project

**Part I:** Find an implementation of the *8-queens* (or even better *n-queens*) problem in at least 3 different languages (only one of those can be in C, C++, Java, or Ada). Print them out. Figure out how they work. Hand them in. One of the three examples should be in a language that directly supports backtracking, such as Icon, Prolog, or SETL.

For those who are not familiar with the terminology, the *8-queens* problem is the problem of finding how to place eight queen pieces on the  $8 \times 8$  chess board so that no two of them threaten each other. In other words, mark 8 squares on an  $8 \times 8$  square grid so that no two marked squares lie in the same row, column, or diagonal. There are several answers. The program is supposed to find them all. The “*n-queens*” problem is the generalization to  $n$  queens on an  $n \times n$  board, for any integer  $n$ . The idea is that enumerating all of the  $n^n$  possibilities ( $n$  choices of placing each of the queens anywhere on its row) blindly and then checking each for validity is too slow (I just checked that in fact my pretty old machine runs through  $8^8$  possibilities (not doing anything intelligent for each of them) in under one second, so I guess  $n = 8$  is no longer a good test case). A simple modification reduces the number from  $n^n$  to  $n!$  ( $n$  factorial), but this is still unacceptably slow, for larger  $n$ . The search is much sped up by placing the queens one by one, row by row, making sure that each queen is placed at a position that is not threatened by any previously placed queens. One goes through all possibilities for the current row, recursively invoking the same strategy for the next row. If we reach row  $n + 1$ , we report success and backtrack. If we cannot find a legal position for the current row, we backtrack to previous row and try the next legal position there. This is the description of the algorithm in human language. Coding it in a language that does not support backtracking directly is not difficult, but a bit tricky.

**Part II:** Solve one of the two problems below in the language of your choice. Both problems require backtracking search (feel free to let me know if you see how to avoid an exhaustive

search). You can implement it yourself by hand, or use a language that has facilities for backtracking.

## 2 Some definitions

In this section we give some definitions. We consider only square  $n \times n$  matrices filled with integers  $0, \dots, n^2 - 1$ , each *exactly* once; for example a  $3 \times 3$  matrix filled with the nine numbers  $0, 1, 2, 3, 4, 5, 6, 7, 8$  as follows

$$\begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}.$$

A  $2 \times 2$  *submatrix* of a matrix is simply a choice of two consecutive rows and two consecutive columns, it gives four matrix entries, we are interested in the *sum* of the values in the submatrix. (For technical reasons, we also need to consider the last and first rows of the matrix consecutive, and the last and first columns consecutive as well.)

## 3 Problem 1

*Initial question:* Write a program to enumerate all integer  $4 \times 4$  matrices filled with numbers  $0, \dots, 15$  as above so that *all* the sums of all possible  $2 \times 2$  submatrices are equal to each other (and are equal to 30, it turns out). One such matrix is

$$\begin{pmatrix} 0 & 15 & 1 & 14 \\ 13 & 2 & 12 & 3 \\ 4 & 11 & 5 & 10 \\ 9 & 6 & 8 & 7 \end{pmatrix}.$$

To eliminate some of the symmetries, you may assume that the top left element of the matrix is always 0.

*More challenging question:* Repeat for larger even-sized matrices, such as  $6 \times 6$ .

*Research-level question:* Devise a description of all such matrices (for  $n = 4$  or for general  $n$ ). Is there a pattern that allows you to generate them all without an explicit search?

## 4 Problem 2

*Initial question:* Given a  $5 \times 5$  matrix as above, for each  $2 \times 2$  submatrix compute its sum. The difference between the largest and the smallest sum is called the *discrepancy* of the matrix. For example, the discrepancy of

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 & 24 \end{pmatrix}$$

is  $(18 + 19 + 23 + 24) - (0 + 1 + 5 + 6) = 76$ , while the discrepancy of

$$\begin{pmatrix} 0 & 24 & 1 & 23 & 2 \\ 22 & 3 & 21 & 4 & 20 \\ 5 & 19 & 6 & 18 & 7 \\ 17 & 8 & 16 & 9 & 15 \\ 10 & 14 & 11 & 13 & 12 \end{pmatrix}$$

would have been 10 ( $54 - 44$ ), except that the four corner elements are considered a single  $2 \times 2$  contiguous submatrix, giving a sum of  $0 + 2 + 10 + 12 = 24$  and thus discrepancy  $54 - 24 = 30$  (if I made no mistakes in my calculations). It is known that (a) there is no such matrix of discrepancy zero (this is what makes odd-size matrices different from even-size matrices) and (b) there exists such a matrix of discrepancy 10.

Write a program to find what is the minimum possible discrepancy for  $5 \times 5$  matrices and to enumerate all matrices with this minimum discrepancy.

*More challenging question:* Repeat for larger odd-sized matrices, such as  $7 \times 7$ .

*Research-level question:* What is the minimum number, for general  $n$ ? How to construct matrices with discrepancy  $2n$  is known.