February 21, 2001

n-Queens

The table shows the size of the state space for different ways of representing an $n \times n$ chess board with n queens placed on it.

A. n queens in different squares, with no other restrictions, gives $\binom{n^2}{n}$ possibilities.

B. Restricting to exactly one queen per row, but no restrictions on columns or diagonals, gives n^n possibilities.

C. Restricting to exactly one queen per row and exactly one per column, but no restrictions on diagonals, gives permutations, so there are $n!$ possibilities.

D. A "solution" is n queens positioned so that no two are in the same row, column, or diagonal.

E–F. The algorithm shown on the back of the page (based on the backtracking algorithm in Chapter 5.1–5.2), constructs space B described above one row at a time, pruning when the rows already constructed are not promising. The number of (E) promising and (F) non-promising nodes are as indicated. An upper bound on $E + F$ is to do a depth-first search without pruning, which explores $n^0 + n^1 + \cdots + n^n = (n^{n+1} - 1)/(n - 1)$ nodes (n^k nodes at depths $k = 0, \ldots, n$).

n	A. $\binom{n^2}{n}$	B. n^n	C. $n!$	D. # solutions	E. # promising	F. # not promising
1	1	1	1	1	2	0
2	6	4	2	0	3	4
3	84	27	6	0	6	13
4	1820	256	24	2	17	44
5	53130	3125	120	10	54	167
6	1947792	46656	720	4	153	742
7	85900584	823543	5040	40	552	3033
8	4426165368	16777216	40320	92	2057	13664
9	260887834350	387420489	362880	352	8394	63985
10	17310309456440	10000000000	3628800	724	35539	312612
11	1276749965026536	285311670611	39916800	2680	166926	1639781
12	103619293824707388	8916100448256	479001600	14200	856189	9247680

Homework #7, Due February 28

Chapter 5# 3; **30 WITH $W = 13$ (BOOK HAS TYPO);** Chapter 6# 18; and the problem below: H-4

Problem H-4. Write an algorithm that takes a positive integer n as input, and prints out all the permutations of $1, \ldots, n$ in the notation shown below. (This can be done recursively in a manner similar to the n-Queens algorithm, although there are other ways.) For $n = 3$, it should output the following 6 permutations. They may be listed in any order:

$$[1,2,3] \quad [1,3,2] \quad [2,1,3] \quad [2,3,1] \quad [3,1,2] \quad [3,2,1]$$

```cpp
// Math 188, Winter 2001, Prof. Tesler
// The Backtracking Algorithm for the n-Queens Problem, in C++
// Based on pseudocode in Neapolitan & Kaimipour, p. 186

#include <iostream.h>      // for cin, cout
#include <iomanip.h>       // for setw
#include <stdlib.h>        // for abs
typedef int index;

class nQueens {
public:
    nQueens(int n) {
        this->n = n;
        col = new int[n];  // array with coordinates of queens
    }

    ~nQueens() {
        delete col;
    }

    void start();
    void finish();

    void queens(index i);
    bool promising(index i);

    void OutputSolution();

private:
    int n;          // Dimension of board
    index *col;     // col[0..n-1]: col[i]=j means queen at row i, column j

    // Statistics: count number of solutions and (non)promising nodes examined.
    int numSolutions, numNonPromising, numPromising;
};

void nQueens::start() {
    numSolutions = 0;           // initialize statistics
    numNonPromising = 0;
    numPromising = 0;

    queens(0);                  // start search for solutions
}

void nQueens::finish() {
    // Display statistics
    cout << "# solutions = " << numSolutions;
    cout << "    # promising nodes = " << numPromising;
    cout << "    # non-promising nodes = " << numNonPromising << endl;
}

// Main routine to traverse nodes of state space tree
void nQueens::queens(index i) {
    // Continue only if columns 0,...,i-1 are promising.
    if (promising(i-1)) {
        numPromising++;
        if (i==n) {               // Have a complete solution.
            numSolutions++;
            OutputSolution();
        } else {
            for (index j=0; j<n; j++) { // place queen in
                col[i] = j;       // row i, column j
                queens(i+1);      // and continue to next row
            }
        }
    } else numNonPromising++;
}

// Check if a node is promising
bool nQueens::promising(index i) {
    // Check if queen in row k threatens queen in row i
    for (index k=0; k<i; k++)
        if (col[i] == col[k]  ||  abs(col[i]-col[k]) == i-k)
            return false;   // does threaten, so not promising

    return true;            // no threats, so promising
}

// Display each solution as it's found, and statistics
void nQueens::OutputSolution() {
    cout << setw(3) << numSolutions
         << " " << setw(3) << numPromising
         << " " << setw(3) << numNonPromising << "  ";
    for (index i=0; i<n; i++)
        cout << "(" << i+1 << "," << col[i]+1 << ") ";
    cout << endl;
}

int main(int argc, char *argv[]) {
    int n;

    cout << "n-Queens" << endl;
    do {
        cout << "Enter n, or 0 to quit: ";
        cin >> n;
        if (n>0) {
            cout << "  #  #P #~P  coordinates" << endl;
            nQueens *nq = new nQueens(n);
            nq->start();
            nq->finish();
            delete nq;
        }
    } while (n>0);
    return 0;
}
```