The Queens Problem Revisited

Kirt Undercoffer

644 G Street, NE, Washington, DC 20002

One of the classic computer science problems is the Queens Problem, which is the problem of placing eight queens on a chessboard so that no queen can take any other queen. The solution to this problem inevitably involves a discussion of backtracking. (A particularly good discussion of this subject can be found in *Algorithms and Data Structures* by N. Wirth [1].) As it turns out, backtracking is the sole technique ever mentioned in conjunction with the Queens Problem, leading one to conclude that no other technique exists to solve the problem, a conclusion which is erroneous indeed. As it turns out, an alternative approach can be formulated using a random walk.

This approach is conceptually simple and consists of randomly selecting a valid position until no available squares remain under consideration. The actual algorithm follows:

1. Randomly choose any square on the chessboard and put a queen on it.
2. Eliminate all squares that are under the queen's control from further consideration.
3. If any empty squares are left then go to step 1 else count the number of queens on the chessboard.
4. If the number of queens is less than eight then clear the chessboard and begin again; otherwise, output the solution.

It will be noted that step # 2 serves to ensure that no queen could ever be in conflict with another queen and forces the rapid elimination of empty and undominated squares (an undominated square being one not controlled by any queen). It is because the algorithm successively reduces the set of possible squares to choose from, that the approach works and a solution tends to "pop out" about once every fifteen times the algorithm is entered.

The accompanying Pascal program is a solution to the Queens problem utilizing the aforementioned algorithm and is not to be considered an optimized program, although it works quite well. Run time could be reduced right away by keeping a running tally of the number of queens on the board and only checking for remaining empty squares when the number of queens is greater than five (I've written the program with this modification). It can be shown that five queens is the minimum number necessary to dominate a chessboard. When the solution matrices are output, queens are represented by '9' 's and dominated squares are represented by '1' 's. Although not intuitively obvious, the random walk approach is an elegant alternative to backtracking for the Queens problem.

Reference

1. Niklaus Wirth, *Algorithms and Data Structures,* Prentice-Hall, Englewood Cliffs, NJ, 1986 (2nd edition).

Journal of Pascal, Ada, & Modula-2, Vol. 6, No. 2, pp. 45–49 (1987)

CCC 0735-1232/87/020045-05$04.00

```
PROGRAM Queens(output);
  VAR
     Row,
     Column,
     index,
     LdRow,
     LdColumn,
     RdRow,
     RdColumn,
     Zero,
     Queen,
     Track        : integer;
     Board        : array[1..8,1..8] of integer;
     Print,Filled : Boolean;


   PROCEDURE Initialize;     { Initialize chessboard }

   { Initialize values used as flags : Track keeps track of the number of
     Queens on the chessboard , Filled is used to determine if the chessboard
     is full, and Print is used to tell the program to print a solution.
     Set each square on the chessboard to 0.                              }

     BEGIN
       Filled      := false;

       Print       := false;
       Track       := 0;
         for Row := 1 to 8 do
           for Column := 1 to 8 do
             Board[Row,Column] := 0
     END;                              { End Initialize }


   PROCEDURE PickPosition;             { Pick a Position }

   { Randomly pick any position. If the value of that square is 0 then
     stick a Queen there, otherwise keep picking positions              }

       BEGIN
           repeat
             Row     := Random(8) + 1;
             Column  := Random(8) + 1
           until (Board[Row,Column] = 0);
           Board[Row,Column] := 9;
           Track := Track + 1
       END;                                { End PickPosition }


   PROCEDURE EvaluatePosition;

   { This procedure finds the squares that are dominated by a queen, sets
     their value to 1 (thus removing them from further consideration).
```

```
  It runs in 4 directions : horizontally, vertically, and diagonally
  (from left to right and from right to left).                    }

BEGIN
                                  { X Out Row }
        for index := 1 to 8 do
          if index < > Column then Board[Row,index] := 1;
                                  { End X Out Row   }

                                  { X Out Column }
        for index := 1 to 8 do
        if index < > Row then Board[index,Column] := 1;
        { End X Out Column                        }

    begin                         { X Out Left Diagonal }
        LdRow     := Row;
        LdColumn  = Column;
          while (LdRow >1) and (LdColumn >1) do
            begin

              LdRow     := LdRow - 1;
              LdColumn  := LdColumn - 1
          end;
          repeat
            begin
              if (LdRow < > Row) and (LdColumn < > Column) then
              Board[LdRow,LdColumn] := 1;
              LdRow     := LdRow +1;
              LdColumn  := LdColumn +1
            end;
          until (LdRow = 9) or (LdColumn = 9)
    end;                          { End X Out Left Diagonal }

    begin                         { X Out Right Diagonal }
        RdRow     := Row;
        RdColumn  := Column;
        while (RdRow >1) and (RdColumn <8) do
          begin
            RdRow     := RdRow -1;
            RdColumn  := RdColumn +1
          end;
        repeat
          if (RdRow< > Row) and (RdColumn < > Column) then
          Board[RdRow,RdColumn] := 1;

          RdRow     := RdRow +1;
          RdColumn  := RdColumn -1;
        until (RdRow = 9) or (RdColumn = 0)
    end                           {End X Out Right Diagonal }
END;                              { End Procedure EvaluatePosition }

PROCEDURE Check;

{ Check the Chessboard to see if any empty (i.e. undominated)
  squares are left. Also count the number of Queens on the Board }
```

```
    BEGIN
       Zero := 0;
       Queen := 0;
         for Row := 1 to 8 do
           begin
             for Column := 1 to 8 do
               begin
                 if Board[Row,Column] = 0 then Zero := Zero + 1;
                 if Board[Row,Column] = 9 then Queen := Queen + 1
               end
           end;
       if Zero = 0 then Filled := true;
       if Queen = 8 then Print := true


    END;

  PROCEDURE Output;

  { Print a solution if there aren't any more empty squares and if the
    number of Queens equals eight.                                    }

      BEGIN
         writeln;
          for Row := 1 to 8 do
            begin
              writeln;
                for Column : = 1 to 8 do
                  write(Board[Row,Column]:3)
             end


        END;                          {END Output}

  BEGIN                             { Main Program }
    REPEAT                          { Repeat until a solution is found }
         Initialize;
        REPEAT                      { Repeat until all the squares are dominated }
          PickPosition;
          EvaluatePosition;
           If Track > 4 then Check
        UNTIL Filled = true
      UNTIL Print = true;
         Output                       { Print a solution }
  END.                                  { End Main }
```

Running

```
1  9  1  1  1  1  1  1
1  1  1  1  1  1  1  9
1  1  1  1  1  9  1  1
9  1  1  1  1  1  1  1
1  1  9  1  1  1  1  1
1  1  1  1  9  1  1  1
```

```
  1  1  1  1  1  1  9  1
  1  1  1  9  1  1  1  1
>
```

Running

```
  1  1  1  1  9  1  1  1
  1  9  1  1  1  1  1  1
  1  1  1  9  1  1  1  1
  1  1  1  1  1  9  1  1
  1  1  1  1  1  1  1  9
  1  1  9  1  1  1  1  1
  9  1  1  1  1  1  1  1
  1  1  1  1  1  1  9  1
>
```

Running

```
  1  1  1  1  1  1  1  9
  1  1  9  1  1  1  1  1
  9  1  1  1  1  1  1  1
  1  1  1  1  1  9  1  1
  1  9  1  1  1  1  1  1
  1  1  1  1  9  1  1  1
  1  1  1  1  1  1  9  1
  1  1  1  9  1  1  1  1
>
```

Running

```
  1  9  1  1  1  1  1  1
  1  1  1  1  1  9  1  1
  9  1  1  1  1  1  1  1
  1  1  1  1  1  1  9  1
  1  1  1  9  1  1  1  1
  1  1  1  1  1  1  1  9
  1  1  9  1  1  1  1  1
  1  1  1  1  9  1  1  1
>
```