Lab 4
Perfect Square - prove the computation time estimate

- Write a program to find and print the first perfect square ($i^2$) whose last two digits are both odd.

- Prove that, in theory, any problem that you write to solve perfect square will run infinite time.

- Answer:
  1. If the last digit of perfect square to be an odd number, the $i$ should be odd. So there are 5 odd numbers can be the last digit of $i$, which are 1,3,5,7,9.

  2. Then we can break down the $i = 10a + b$, then $i^2 = (10a + b) \times (10a + b) = 100a^2 + 20ab + b^2$.

  3. Since $b$ should be one of 1,3,5,7,9, so the second digit is even for $b^2$, so the second last digit of $20ab + b^2$ is also even.

  4. Therefore, there is no solution for this problem, and the program will run infinite time.
Eight Queens

- The eight queens puzzle is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.

- Now, how can we compute the number of possible positions to place eight Queens to the chessboard?
Brute force

• There are 8 possible positions to place the Queen in the first column.

• For each position in the first column, there are 8 possible positions to place the Queen in the second column.

• For each position in the seventh column, there are 8 possible positions to place the Queen in the last column.

• Thus, there are $8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8$ possible ways in total $\sim 16.8$ million. We need to set up all these board to identify which ones are valid.

• This approach is called brute force.
Brute force Outline

bool ok( ... what goes here ...){
    ... and here ...
};
void print( ... what goes here ... ){
    ... and here ...
};
int main( ) {
    int board[8][8]={0};
    int count = 0;
    for(int i0 =0; i0 <8; i0 ++)
        for(int i1 =0; i1 <8; i1 ++)
            for(int i2 =0; i2 <8; i2 ++)
                for(int i3 =0; i3 <8; i3 ++)
                    for(int i4 =0; i4 <8; i4 ++)
                        for(int i5 =0; i5 <8; i5 ++)
                            for(int i6 =0; i6 <8; i6 ++)
                                for(int i7 =0; i7 <8; i7 ++){
            // use the indices of the loops to set a configuration in array board
            ...
            // if this configuration is conflict-free, print the count and the board
            if(ok(board)) print(board, ++count);
            // reset the board for the next configuration
            ...
    }
    return 0; }

Eight Queens 2D backtrack

- The disadvantage of brute force approach is time complexity.
- Is there any other approach which can spend less time.
- Two dimension backtrack for eight queens.
Eight Queens 2D backtrack

- If a position is not suitable then try the next one.

- If none of the position are suitable then back track to pervious column and try the next position in that column.

- When a solution is found call backtrack to look for others. (at this point column would be 8)

- When backtrack takes you back to the very beginning (column = 0, row = 7) backtracking again would push column to -1, therefore you found all solutions.
Pseudo code

- Step 1: Place eight Queens to chessboard.

```java
int b[8][8] = {0}
```
Pseudo code

• Step 2: Make sure the positioning of the Queens can not attack each other. If this is the case, this board passes ‘the test’.

• Starting from right most piece:
  - Find where the Queens is on the current column
  - Make sure there isn’t another Queen on this row
  - Make sure there isn’t another Queen in up diagonal direction
  - Make sure there isn’t another Queen in down diagonal direction
Pseudo code

• Step 3: If a board setup passes the test
• Print out the board setup
• Keep track how many boards are found
Eight Queens 2D backtrack outline

```c
int main() {
    /*board setup section*/
    next_col:
    /*column section*/
    next_row:
    /*row section*/
    backtrack:
    /*backtrack section*/
    print:
    /*print section*/
}
```
int b[8][8] = {0}, //8x8 board, 0s are blanks, 1s are queens
r, //row index
c = 0; //column index, initially 0
b[0][0] = 1; //putting 1st queen piece on upper-left corner
//advance column position
if (c == 8) //solutions for columns 0 through 7 found,
goto print; //so print
r = -1; //otherwise start at the “top” of the column
Row Section

```c
++r; //advance row position
if (r == 8) //tried every row of current column, none work,
goto backtrack; //so we backtrack
```
for (int i = 0; i < c; ++i)
if (b[r][i] == 1) //if another queen on same row,
goto next_row; //try the next row
Row Section, continued

for (int i = 1; (r-i) >= 0 && (c-i) >= 0; ++i)
    if (b[r-i][c-i] == 1) //if conflict found in up-diagonal,
        goto next_row; //try the next row

Note:

b[r-i][c-i] //Row and Column indexes decrease,
    // cannot dip below 0!

(r-i) >= 0 && (c-i) >= 0
Row Section, continued

for (int i = 1; (r+i) < 8 && (c-i) >= 0; ++i)
    if (b[r+i][c-i] == 1) //if conflict found in down-diagonal,
        goto next_row; //try the next row

Note:

b[r+i][c-i] //Row index increases, cannot go above 7.
    //Column index decreases, cannot dip below 0.

(r+i) < 8 && (c-i) >= 0
Row Section, continued

\[
b[r][c] = 1; \quad \text{//no conflicts found, so place queen}
goto \text{next}_\text{col}; \quad \text{//move on to the next column}
\]
Backtrack Section

--c; //go back one column
if (c == -1) //if past first column, all solutions found,
return 0; //so terminate program

| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|----|---|---|---|---|---|---|---|---|---|
Backtrack Section, continued

r = 0; // start looking from first row
while (b[r][c] != 1) // while we haven’t found the queen piece,
++r; // move to next row of current column
b[r][c] = 0; // remove queen piece
goto next_row; // move on to next row
print(b); //print the board
goto backtrack; //go back to find more solutions