Assignment 5

CS5310

Jonah Kubath

Nov 22, 2018

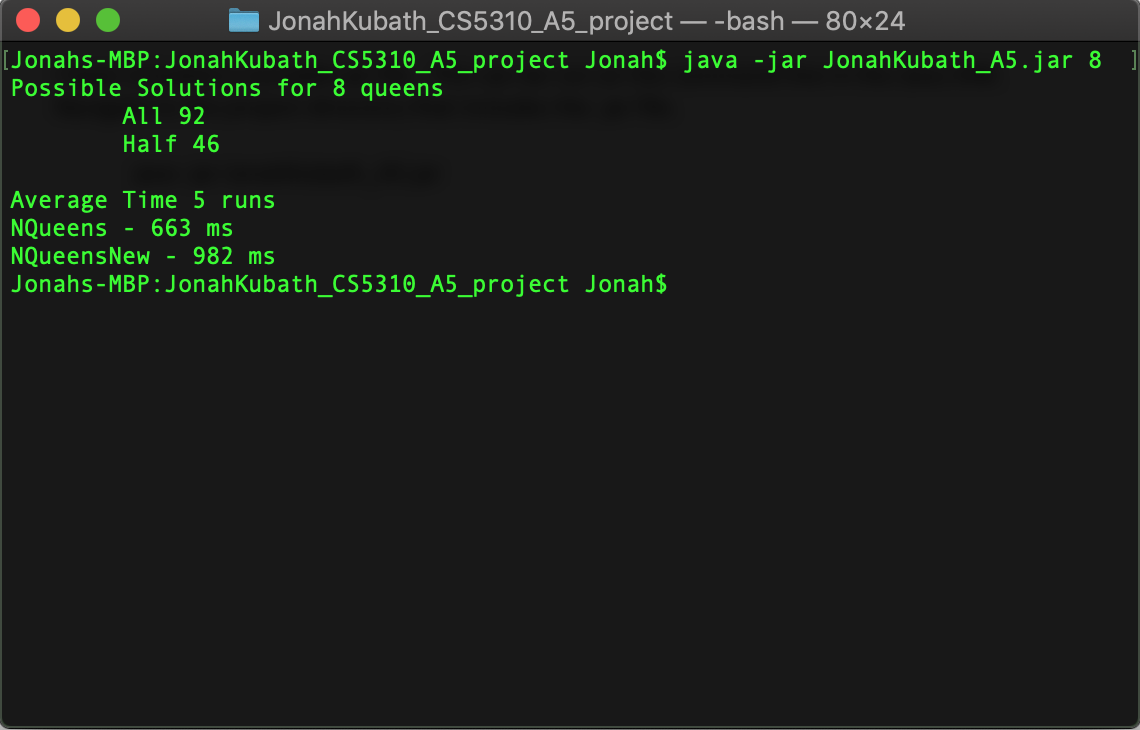
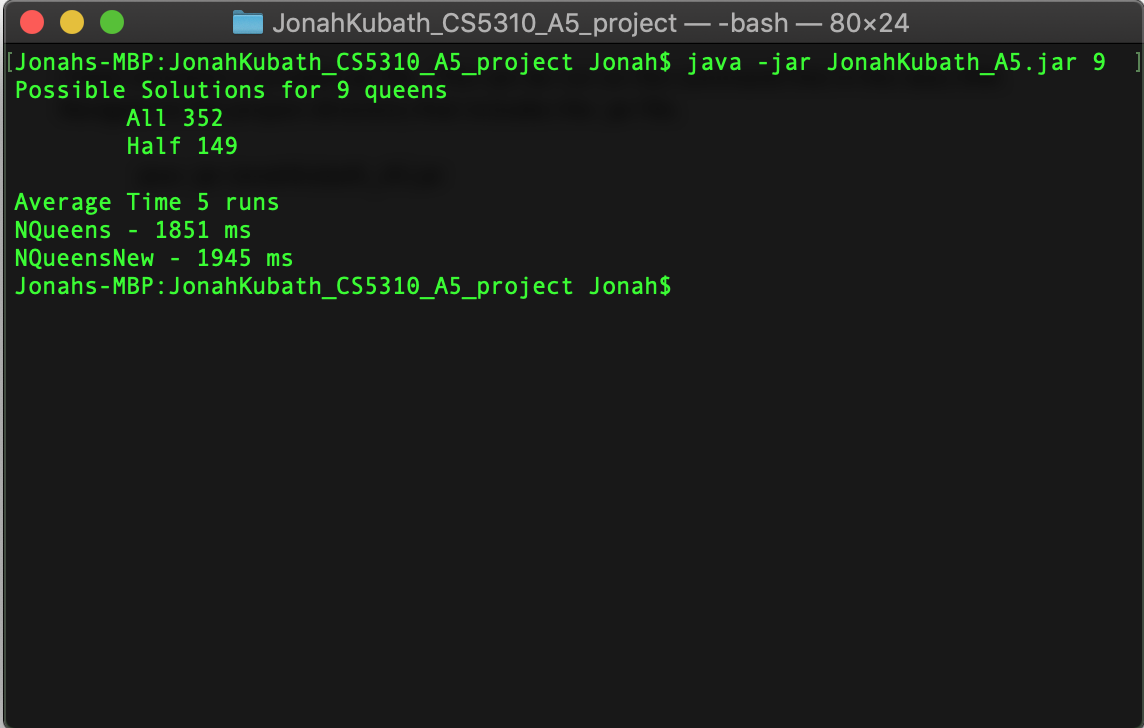
Assignment 5 is implementing the HSR algorithms for the N-Queens problems. The problem is described by placing “n” number of queens on chess board. The constraints are that no queen can be in another row, column, or diagonal of another queen on the board. The basic algorithm is as follows. Check if the current queen can be placed in the current column. If it can, place it and move to the next queen. If it cannot be placed, check the next column. Once the n queens are placed, the current iteration has finished, and the columns values can be printed.

Execution:

I used Eclipse to load my project folder and run the program for the test files.

I have included a runnable jar file. This can be run on the command line in the Java JVM. Navigate to the project directory that includes the .jar file.

java -jar JonahKubath\_A5.jar <# of Queens>



**package** jonahKubath\_A5;

**public** **class** Main {

**static** **int**[] *x*;

**static** **int**[] *y*;

**static** **int** *all* = 0;

**static** **int** *half* = 0;

**static** **int** *called1* = 0;

**static** **int** *called2* = 0;

**static** **boolean** *printCount* = **false**;

**static** **boolean** *printArray* = **false**;

**public** **static** **void** main(String[] args) {

**int** n = 8;

**int** numberOfRuns = 5;

**if**(args.length > 0) {

n = Integer.*parseInt*(args[0]);

}

*x* = **new** **int**[n];

*y* = **new** **int**[n];

**long** time[][] = **new** **long**[2][3];

time[0][2] = time[1][2] = 0;

**for**(**int** j = 0; j < numberOfRuns; j++) {

**for**(**int** i = 0; i < n; i++) {

*x*[i] = -20;

*y*[i] = -20;

}

**if**(*printCount*) {

*called1* = 0;

*called2* = 0;

}

*half* = *all* = 0;

time[0][0] = System.*nanoTime*();

*NQueens*(0, n);

time[0][1] = System.*nanoTime*();

time[0][2] += time[0][1] - time[0][0];

time[1][0] = System.*nanoTime*();

*NQueensNew*(0, n);

time[1][1] = System.*nanoTime*();

time[1][2] += time[1][1] - time[1][0];

**if**(j == 0) {

System.***out***.println("Possible Solutions for " + n + " queens");

System.***out***.println("\tAll " + *all*);

System.***out***.println("\tHalf " + *half*);

}

}

System.***out***.println();

System.***out***.println("Average Time " + numberOfRuns + " runs");

System.***out***.println("NQueens - " + time[0][2] / numberOfRuns / 1000 + " ms");

System.***out***.println("NQueensNew - " + time[1][2] / numberOfRuns / 1000 + " ms");

**if**(*printCount*) {

System.***out***.println("Called1 " + *called1*);

System.***out***.println("Called2 " + *called2*);

}

}

/\*\*

\* Determines where n queens can be placed on an n x n chess board with

\* queens not allowed in the same column, row, or diagonal of another queen

\* **@param** k Which queen from 0 to n-1 we are currently placing

\* **@param** n Number of queens

\*/

**public** **static** **void** NQueens(**int** k, **int** n) {

**if**(*printCount*)

*called1*++;

**for**(**int** i = 0; i < n; i++) {

//Can node k go into column i

**if**(*Place*(k, i)) {

*x*[k] = i;

**if**(k == n-1) {

**if**(*printArray*)

*printLine*(*x*);

*all*++; // Global counter

}

**else** {

//Place the next queen

*NQueens*(k + 1, n);

}

}

}

}

/\*\*

\* Can the queen k be placed in column i

\* **@param** k The queen we are attempting to place

\* **@param** i The column we are attempting to place her in

\* **@return** True if allowed, false if it would break the conditions

\*/

**public** **static** **boolean** Place(**int** k, **int** i) {

/\* Verify queen is not in the same column vertically, horizontally, or

\* diagonally to another queen \*/

**for**(**int** j = 0; j < k; j++) {

**if**(*x*[j] == i || Math.*abs*(*x*[j] - i) == Math.*abs*(j - k)) {

**return** **false**;

}

}

**return** **true**;

}

/\*\*

\* Determines the next available column to place the queen

\* or -1 on none found

\* **@param** k Which queen we are placing

\* **@param** i Which column to start looking at

\* **@param** n Number of columns

\* **@return** The index of the next available column

\*/

**public** **static** **int** PlaceNew(**int** k, **int** i, **int** n) {

**boolean** good = **false**;

// Iterate through all possible columns

**while**(i < n) {

good = **true**;

**for**(**int** j = 0; j < k; j++) {

**if**(*y*[j] == i || Math.*abs*(*y*[j] - i) == Math.*abs*(j - k)) {

//Cannot place value here

good = **false**;

**break**;

}

}

**if**(good) {

**return** i;

}

**else** {

i++;

}

}

//Cannot place a value anywhere

**return** -1;

}

/\*\*

\* Determines where n queens can be placed on an n x n chess board with

\* queens not allowed in the same column, row, or diagonal of another queen

\* This version is optimized by not iterating solutions that are mirror

\* images of previous versions. This is done by only iterating over half

\* of the first queens possible sets 0 to Ceil( n / 2)

\* **@param** k Which queen from 0 to n-1 we are currently placing

\* **@param** n Number of queens

\*/

**public** **static** **void** NQueensNew(**int** k, **int** n) {

**if**(*printCount*)

*called2*++;

**int** max = n;

**if**(k == 0) {

max = (**int**) Math.*ceil*(n / 2);

}

//System.out.println("Place: " + k);

**for**(**int** i = 0; i < max; i++) {

//System.out.println("\t" + i);

//Can node k go into column i

**int** column = *PlaceNew*(k, i, n);

**if**(column != -1) {

i = column;

*y*[k] = column;

**if**(k == n-1) {

**if**(*printArray*)

*printLine*(*y*);

*half*++; // Global counter

}

**else** {

// Place the next queen

*NQueensNew*(k + 1, n);

}

}

}

}

/\*\*

\* Print the queen array as n x n with the number where the queen is placed

\* and '-' where no queen is placed.

\* **@param** array n length array of queen data

\*/

**public** **static** **void** printArray(**int**[] array) {

**for**(**int** i = 0 ; i < array.length; i++) {

**for**(**int** j = 0; j < array.length; j++) {

**if**(j == *x*[i]) {

System.***out***.printf(" %d ", array[i]);

}

**else**

System.***out***.print(" - ");

}

System.***out***.println();

}

}

/\*\*

\* Print the array in a line separated with spaces

\* **@param** array n length array of queen data

\*/

**public** **static** **void** printLine(**int**[] array) {

**for**(**int** i = 0 ; i < array.length; i++) {

System.***out***.printf(" %d ", array[i]);

}

System.***out***.println();

}

}