COSCI 21a, Programming Assignment P1

Purpose: Experience with some basic data structures, including circular queues, linked lists, hash tables, and specifically with pointers and the associated dynamic memory allocation.

Problem: Write a C program to find a solution of a minimal number of steps to the *Nite and Day puzzle*, where one step is defined as moving a piece vertically or horizontally to an adjacent unoccupied square, and the two $ pieces cannot move:

![Nite and Day puzzle diagram]

Guidelines:
At a high level, your program should perform breadth-first as described by this pseudo-code:

```
INSERT start position into an initially empty hash table
ENQUEUE start position

while (goal position not found) do begin
  DEQUEUE a position $P$ 
  for each position $X$ that is reachable from $P$ by one move, but is not in the hash table,
  INSERT $X$ into the hash table and ENQUEUE $X$.
end
```

Output a solution by reversing the sequence going from the goal back to the start position.

Although there are a number of alternatives, for this assignment you must use these two data structures:

*Hash Table:* Stores all positions seen thus far.
- Each entry is either a NULL pointer or a pointer to the first vertex of a singly linked list of the positions that have thus far hashed to that bucket.
- Each position stored in the hash table is a structure that includes the hash table bucket linked list pointer, a pointer to the position from which this position came (NULL for the start position), the piece that has moved (0 for the start position), the direction in which the piece moved (0 for the start position), and an array representing the position.

*Queue:* Stores the positions waiting to be explored.
- Each entry is a pointer to a position in the hash table.
- A circular queue array implementation is used as presented in class.

There are a number of ways to output the solution once the goal position has been reached (see the questions); for full credit, you must reverse the back pointers as the path back to the start is traversed, and then traverse them in the forward direction to print the solution.

NOTE: Although collaboration on a programming project can in general be a good thing in the work place or in other learning situations, for this assignment, all code you submit should be yours, and yours alone; *you may not collaborate with others to write it.* General high level discussions with others (and the TA’s) for the purpose of learning is fine, but you must independently author every line of your code. Be sure to read the handout on academic honesty. The one exception is that you may, if you like, use or modify portions of code from handouts on the COSCI 21a web page or the sample solution to Assignment P0, so long as you clearly cite this with comments in your code.
Requirements for writing your program:
In order to facilitate grading, you are provided these two files; they may not be modified (the grader will be using exactly these files, without modifications, when compiling your code):

\textit{Puzzle.h}

The definition of the puzzle, including the size you must use for your queue and hash table.

\textit{Output.h}

The format of your output must be the positions visited in each step of the solution followed by statistics printed by using the functions in \textit{Output.h} (see the sample output for the \textit{AB Puzzle}). A software tool will be employed to check your output. All output must be produced by calling the functions in \textit{Output.h}; that is, the code you submit should have no output statements (e.g., no \texttt{printf} statements). Also, your code may not read or write to any files.

\textit{Note:} A board is defined in these functions as a simple one dimensional array. However, you can use any representation for a puzzle position you like. For example, if you are representing boards with 2-dimensional arrays, you can copy the board information to a temporary one dimensional array and pass that to the function in \textit{Output.h} that prints a position.

What To Pass In:

1. \textit{Assignment P1a:} Submit your code as the single file named \textit{LastnameFirstname.c} (e.g., if your name is \textit{John Smith}, the file is named \textit{SmithJohn.c}).

2. \textit{Assignment P1b:} Submit question answers as the single file named \textit{LastnameFirstname.pdf}.

   *** \textit{Do not submit anything else}; include any additional comments, notes to the graders, etc., at the end of this file.

Grading:

1. Your program will be compiled and run in a Unix terminal window on a machine in the COSCI Vertica lounge by doing:
   \begin{verbatim}
   gcc -std=c99 -Wall -I . LastnameFirstname.c; ./a.out > output
   \end{verbatim}
   This compilation must complete \textit{without any errors or warnings}; be sure to check this in the COSCI Vertica lounge before submitting your work.
   Your code may not accept any input and may not write to any files.
   Your code must have these and only these \#include statements:
   \begin{verbatim}
   #include <stdio.h>
   #include <stdlib.h>
   #include <Puzzle.h>
   #include <Output.h>
   \end{verbatim}

2. A program will be run to check that your output is a correctly formatted minimal solution with reasonable statistics.

3. Your code will be examined to evaluate the general style and commenting, and to verify that it adheres to the assignment directions (algorithm, data structures, etc.).

4. Credit may be deducted if you did not successfully complete and submit Assignment P0 on time.

5. Answers to questions will be evaluated and any comments at the end taken into account.
Questions:

Note: Although the questions will be graded on a nominal scale of 10 points per question, there is no specific percentage of the total assignment score for they count. For programs that do not work, good answers to questions may increase partial credit, and for programs that work, poor or missing answers to questions may cause loss of credit. Following your answers to these questions you may include any additional comments, explanations, etc. that you would like the grader to read.

1. Compare time and space used if instead of a separate circular queue, the queue was incorporated into the position structures a singly linked list (i.e., position structures now include a NEXT field).

2. Compare the worst-case time and average time of representing a hash table bucket as a singly linked list versus representing a bucket with a binary search tree; consider both the situation where the hash table size is larger than the number of positions, and when it is set to 3.

3. Depending on the puzzle and size of the hash table, the number of positions may exceed the number of buckets. Describe how you would implement closed hashing to address this issue, and compare the time and space with the open hashing.

4. The assignment asked you to print a solution by reversing back pointers. Another approach is to push onto a stack pointers to the positions visited as back pointers are traversed. A third alternative is the following recursive function to print the path. In terms of time and space, ease of programming, and issues such as not knowing the length of the solution in advance, compare these three options.

   procedure PrintSolution(position)
   if position is not NULL then PrintSolution(position -&gt; back)
   Output position.
   end

5. When a position X is dequeued, the program could not generate the position Y that results moving back in the same direction from which X came, or it could generate all positions reachable from X and discover that Y was already in the hash table. Discuss the pros and cons of these alternatives.

6. How you would generalize your program to solve puzzles like the Traffic Jam Puzzle:

7. In this assignment, one move is moving one piece one unit in one direction, and solutions expressed using this type of unit movement tend to be a bit tedious. Two other types of movement are straight movement (one piece can be moved any distance in a single direction) and rectilinear movement (one piece can be moved any distance along a rectilinear path). Note that one cannot in general combine successive moves of a minimal solution for the unit metric to obtain a minimal solution in one of the other two metrics. Discuss how you could generalize your program to find minimal solutions for the straight and rectilinear metrics.

8. What was the most difficult part of this assignment? Would it have helped if you were allowed to write your program using multiple files. If instead of being required to write this code on your own, if you could divide the work of writing and debugging with three other students, how would you organize the project in such a way so as to most efficiently complete the project while having a relatively even work load between the four of you?
Example Output For The "AB Puzzle"

Step 0:  
A B $  
$ . $  

Step 1, move B south:  
A . $  
$ B .  
$ . $  

Step 2, move A east:  
. A $  
$ B .  
$ . $  

Step 3, move B east:  
. A $  
$ . B  
$ . $  

Step 4, move A south:  
.. $  
$ A B  
$ . $  

Step 5, move A south:  
.. $  
$ . B  
$ A $  

Step 6, move B west:  
.. $  
$ B .  
$ A $  

Step 7, move B north:  
. B $  
$ . .  
$ A $  

Step 8, move B west:  
B . $  
$ . .  
$ A $  

Step 9, move A north:  
B . $  
$ A .  
$ . $  

Step 10, move A north:  
B A $  
$ . .  
$ . $  

Queue statistics:  
Queue array size: 25  
Max positions ever in queue: 4  
Index of Qfront: 20  
Index of Qrear: 20  
|Qfront-Qrear| = 0  

Hash table statistics:  
Hash array size: 23  
No. positions in hash table: 20  
Max bucket size: 3  

Hash buckets at the end:  
buckets of size 0 = 10  
buckets of size 1 = 7  
buckets of size 2 = 5  
buckets of size 3 = 1