CSCE 210 Fall 2017 Dr. Amr Goneid
Assignment #6

Date: Sat Nov 25, Due: Tue Dec 5, 2017

This assignment is an individual or a group work of maximum **Three members**. Group members submitted for assignment 4 on Monday, November 6th 2017, will be considered and will remain the same.

The problem: 8 Puzzle

The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is a special case of the 15-puzzel. Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to rearrange the tiles to match a final configuration using the empty space using as few moves as possible. We can slide four adjacent (left, right, above and below) tiles into the empty space. The following shows a sequence of legal moves from an *initial board* (left) to the *goal board* (right).

	1	3
4	2	5
7	8	6

Initial

1		3
4	2	5
7	8	6

1 Left

1	2	3
4		5
7	8	6

2 Up

1	2	3
4	5	
7	8	6

5 Left

1	2	3
4	5	6
7	8	

6 Up (Goal)

Best-first search. A solution to the problem can be achieved using a general artificial intelligence methodology known as the A* search algorithm. We define a *search node* of the game to be a board, the number of moves made to reach the board, and the predecessor search node. First, insert the initial search node (the initial board, 0 moves, and a null predecessor search node) into a priority queue. Then, delete from the priority queue the search node with the highest priority, and insert onto the priority queue all neighboring search nodes (those that can be reached in one move from the dequeued search node). Repeat this procedure until the search node dequeued corresponds to a goal board. The success of this approach depends on the choice of *priority function* for a search node. We choose the ***minimum cost*** node (x) to be of highest priority. Such cost is:

c(x) = g(x) + h(x), where,
g(x) = cost of reaching current node from the root = the number of ***moves*** made so far to get to (x)
h(x) = cost of reaching an answer node from (x).

For ***h(x)*** we can use the sum of the ***Manhattan distances*** (sum of the vertical and horizontal distance) from the tiles to their goal positions. In the example shown below, the cost ***h(x)*** = 10.

1	2	3
4	5	6
7	8	

Node (x)

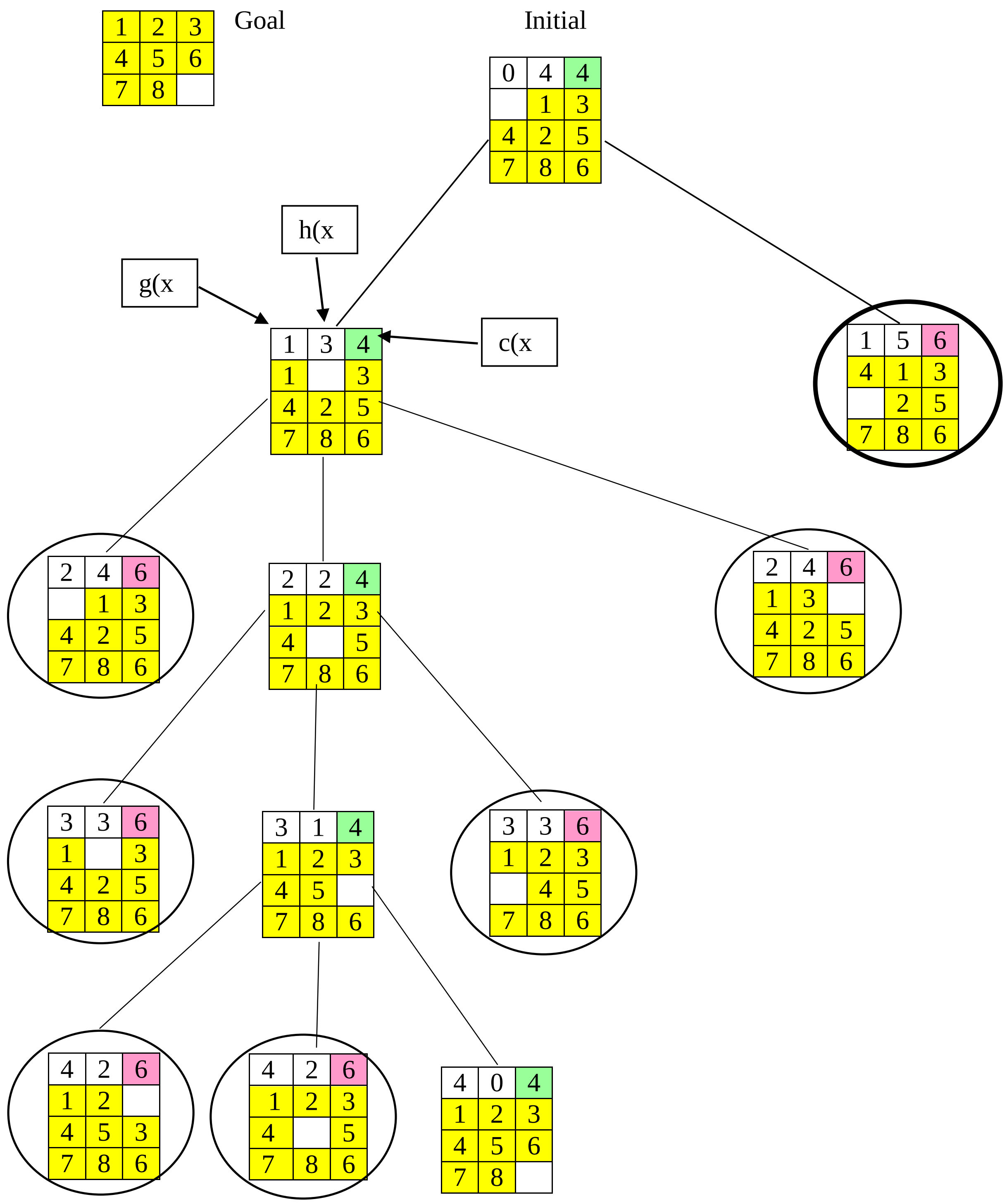
8	1	3
4		2
7	6	5

Goal

1	2	3	4	5	6	7	8	sum
1	2	0	0	2	2	0	3	10

Manhattan distances

The following is an example of the best-first method. Notice that the first row of a node (x) gives its cost ***g(x)***, ***h(x)*** and the sum ***c(x).***



Final Solution

Notice that the first row in a node (x) contains the cost for that node. Also, at a given level, only the node with the least cost (highest priority) is expanded to the next level.

We make a key observation: To solve the puzzle from a given search node on the priority queue, the total number of moves we need to make (including those already made) is at least its priority, using the Manhattan priority function. This is true because each block must move its Manhattan distance from its goal position. Note that we do not count the blank square when computing the priorities. Consequently, when the goal board is dequeued, we have discovered not only a sequence of moves from the initial board to the goal board, but one that makes the fewest number of moves.

A critical optimization. Best-first search has one annoying feature: search nodes corresponding to the same board are enqueued on the priority queue many times. To reduce unnecessary exploration of useless search nodes, when considering the neighbors of a search node, don't enqueue a neighbor if its board is the same as the board of the predecessor search node.

```
  8  1  3       8  1  3       8  1          8  1  3       8  1  3
  4     2       4  2          4  2  3       4     2       4  2  5
  7  6  5       7  6  5       7  6  5       7  6  5       7  6

predecessor   search node    neighbor       neighbor      neighbor
                                           (disallow)
```

<u>Objective</u>
Implement a Priority Queue data type to support the A* algorithm: where the smallest priority is extracted from the queue, then it is processed by adding its children to the priority queue.

<u>Input and output formats.</u> The input and output format for a board is the board dimension *n* followed by the *n*-by-*n* initial board, using 0 to represent the blank square. As an example,

```
// Initial board (Input): Level 0
3
```

0	1	3
4	2	5
7	8	6

```
// Solution steps (Output)
Same as above for the least cost nodes at subsequent levels
```

<u>Required Implementation:</u>

1. Design and implement a template class ***PQ***
2. ***Corner cases.*** You may assume that the constructor receives an *n*-by-*n* array containing the n^2 integers between 0 and $n^2 - 1$, where 0 represents the blank square.
3. Your program will read an input .txt file that defines the board size and the initial board elements
4. ***Performance:*** Your implementation should support all methods in time proportional to n^2 (or better) in the worst case.