CS 499 (Fall 2006)- Assignment 1
Due: Tuesday, 08/29/2006

(1) The 8-puzzle is the following: there are 8 tiles, and one empty square, arranged in a 3×3 grid. The tiles are numbered $1 \ldots 8$, and we denote the empty square by $\square$. Thus, a configuration could look as follows: $\begin{array}{|ccc|}\hline 7 & 2 & 4 \\ \square & 1 & 5 \\ 9 & 3 & 6 \\ \hline\end{array}$ The rules of the puzzle are as follows: you can slide the tile from the top, left, bottom, or right onto the empty position. Then, of course, the position where the sliding tile was before becomes empty. The goal is to transform one configuration to another.

As you can see, there are a total of $9! = 362880$ configurations. Let's say that the puzzle starts in the configuration $\begin{array}{|ccc|}\hline 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \square \\ \hline\end{array}$.

It turns out that only half of all configurations can be reached from the starting configuration by sliding tiles (you'd have to break the puzzle and reassemble it to get the other half).

You are supposed to write a program which generates all configurations reachable from the starting configuration, and for each of them writes down the minimum number of moves required to get there. Write all of them into a file, in a format such as

```
[7,2,4], [X,1,5], [9,3,6]:  16 moves
```

for the example above (I don't know if the number 16 is correct).

Hint: you will probably see that BFS is just fine (no A* search needed). One problem that you will face is how to make sure that no configuration is visited multiple times. That will require some sort of intelligent indexing of configurations. Two thoughts come to mind: (1) Find a natural way to number them, so you can store them in an array, or (2) generate a list of all visited states in which you look up whether a given state is visited. Probably, you should keep this sorted so you can do binary search.

The second option is probably less efficient and more complicated. But you can choose which one you like better. Please do not rely on Hash Table functionality in C++ or Java.

(2) One of the graph problems we discussed in class is the following: you are given an undirected graph G with distances $d_e \geq 0$ on the edges. You are also given a number k. You now get to select a set S of k nodes, and your goal is to minimize the distance of nodes to S (you can assume that the graph satisfies the triangle inequality: there is an edge between each pair of vertices, and its distance is no larger than any multi-hop path between the same vertices). Specifically, for each node u, let $d(u, S)$ be the distance from u to the closest node in S (if u itself is in S, this distance is of course zero). Our goal is to select S to minimize the average distance $\frac{1}{n} \sum_u d(u, S)$.

Suppose that you had an efficient way to solve this problem (which you probably won't — it is NP-complete). What interesting real-world problems could you address with it? Give at least three examples from significantly different networks. Explain what the theoretical problem would let you do. Also be critical of your own suggestions: state what simplifying assumptions you are making, where perhaps the real objective might be slightly different (or where there would be different options that might all be right). Give some discussions of how your criticisms could be addressed, how they could lead to different problems, etc.

[In case this seems vague and open-ended, it is intended to be. You are supposed to be a bit creative, and follow your own ideas. The model should be the kind of discussions we have been having in class.]