CS 341: Algorithms @ Waterloo **Last Updated:** Feb 8, 2016
Homework 3 **Due Date: Mar 3, 2016, 11:59pm**

The full mark is 50. This homework is counted 5% of the course grade.

For written problems, please write pseudocode for the algorithms, prove their correctness and analyze their time complexity.

---

1. **Programming Problem: Snake and Ladder** (18 marks)

   Probably you have played the game "snake and ladder" before. There is an $n \times n$ grid (most commonly $n = 10$), where the n^2 squares are numbered from 1 to n^2. All players start from square one. Each takes turns to roll a dice, and move one to six steps depending on the outcome of the dice (i.e. if the current square is i, we may move to square $i+1, \ldots, i+6$). After a move, if there is a ladder with its bottom in the square that we are in, we move up to the top of the ladder; on the other hand, if there is a snake with its head in the square that we are in, we have to move down to the tail of the snake. The goal is to race to the square n^2 as quickly as possible.

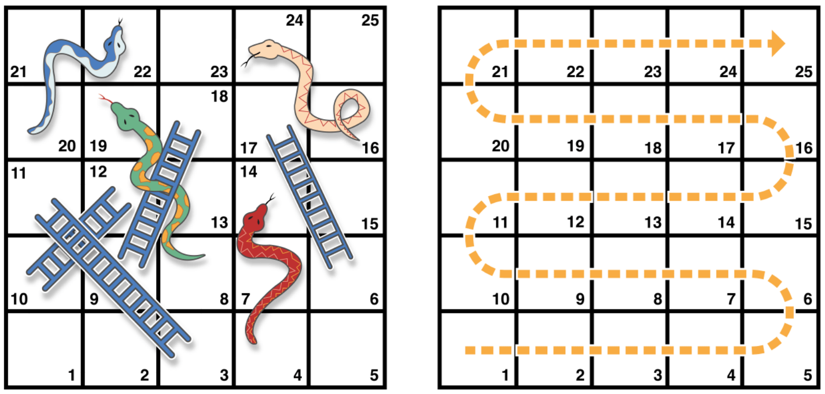

   Figure 1: Picture from developer.apple.com.

   Your friend is an amateur in designing new maps for this game. You would like to write a computer program to check whether the new maps are good. There are a couple things that we would like to check. First, we would like to check whether it is possible at all to reach square 100 (it may be impossible if there are too many snakes). Then, we would like to compute how many turns it takes in the worst case to reach square 100 (which could be infinity).

   Input: The first line will have three integers, n, l and s, where $2 \leq n \leq 1000$ is the side length of the grid, and $0 \leq l \leq 500000$ is the number of ladders in the map, and $0 \leq s \leq 500000$ is the number of snakes in the map. The next l lines are about the locations of the ladders. Each line will have two numbers $2 \leq b < u \leq n^2$, where b is the bottom square of the ladder and u is the top square of the ladder (note that there is no ladder bottom at square 1). The next s lines are about the locations of the snakes. Each line will have two numbers $n^2 - 1 \geq h > t \geq 1$, where h is the head square of the snake and t is the tail square of the snake (note that there is no snake head at square n^2). You can assume that the ladders and the snakes do not share the same square.

Output: There is only one output for each test case. If it is impossible to reach square n^2, output "impossible". Otherwise, if it is possible to reach square n^2, output the largest possible number of steps to reach square n^2 if this number is finite, or output "infinity".

Sample Input 1:

```
10 0 6
99 1
98 2
97 3
96 4
95 5
94 6
```

Sample Output 1:

```
impossible
```

Sample Input 2:

```
10 0 1
99 1
```

Sample Output 2:

```
infinity
```

Sample Input 3:

```
10 0 0
```

Sample Output 3:

```
99
```

2. **Written Problem: Huffman coding** (8 marks)

   Prove the following two properties of the Huffman encoding scheme.

   (a) If some character occurs with frequency more than 2/5, then there is guaranteed to be a codeword of length 1.

   (b) If all characters occur with frequency less than 1/3, then there is guaranteed to be no codeword of length 1.

3. **Written Problem: Thickest Paths** (12 marks)

   Imagine that an online video provider would like to find paths of highest bandwidth to send its videos to the receivers. This can be modeled as a graph problem. We are given a directed graph $G = (V, E)$ where every directed edge e has a positive integer thickness b_e (representing the bandwidth of a link). Given a source vertex s and a receiver vertex t, let $\mathcal{P}_{s,t}$ be the set of all directed paths from s to t in G. We would like to find a path in $\mathcal{P}_{s,t}$ that maximizes the minimum thickness on the path, i.e.,

$$\max_{P \in \mathcal{P}_{s,t}} \min_{e \in P} b_e.$$

   Design an efficient algorithm to find a thickest path from s to t for all $t \in V - s$. You will get full marks if the time complexity of the algorithm is $O((|V| + |E|) \log |V|)$ and the proofs are correct.

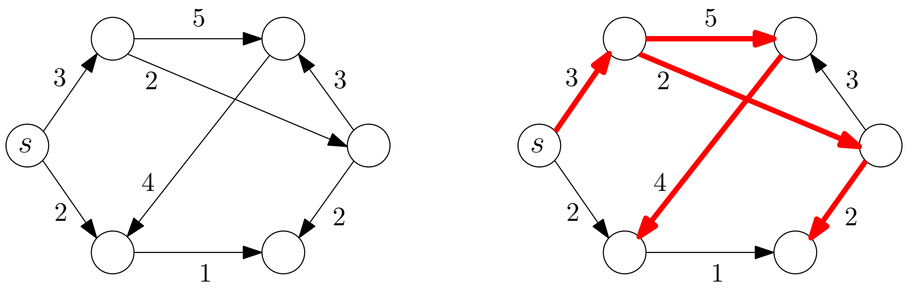

Figure 2: The highlighted edges on the right form the thickest paths from s to all other vertices.

4. **Written Problem: Maximum Interval Coloring** (12 marks)

   In class, we have studied the interval scheduling problem and the interval coloring problem. Here we consider a common generalization of these two problems. We are given n intervals $[s_i, f_i]$ where each s_i and f_i are positive integers with $s_i < f_i$, and a positive integer k. Our goal is to use k colors to color as many intervals as possible, so that each interval receives at most one color (could have no color on an interval) and no two overlapping intervals can receive the same color (two intervals $[s_i, f_i]$ and $[s_j, f_j]$ are overlapping if $[s_i, f_i] \cap [s_j, f_j] \neq \emptyset$). Notice that the interval scheduling problem is the special case when $k = 1$, and the interval coloring problem is the special case to find the minimum k so that all the intervals can be colored. You can think of this problem as using k rooms (k colors) to schedule as many activities (intervals) without time conflicts as possible.

   Design an efficient algorithm to find such a coloring. You will get full marks if the time complexity of the algorithm is $O(n \log n)$ and the proofs are correct. To implement the algorithm efficiently, you can use a balanced search tree data structure (such as AVL tree) and assume all the operations (add, delete, search) can be done in $O(\log |T|)$ time where $|T|$ is the size of the search tree.

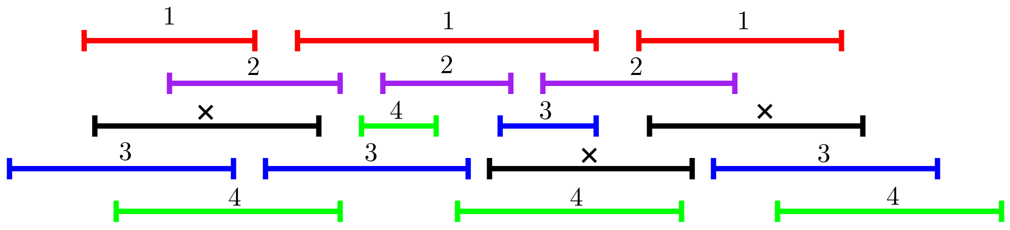

Figure 3: Coloring these intervals with four colors. The intervals with a cross are uncolored.