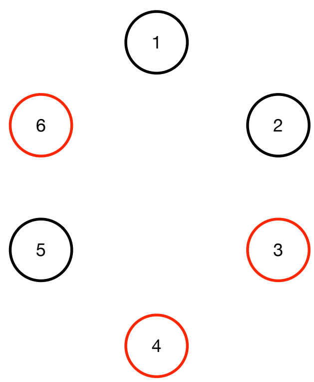
CS 561, HW 9

Prof. Jared Saia, University of New Mexico

1. A cat hops on posts arranged in a circle. There are $2n$ posts, with n red and n black. The cat can start at any post, and always hops to the next post in the clockwise direction, until it visits all posts. It "wins" if, at every point during its trip, the number of red posts visited so far is always at least the number of black posts visited so far. In the figure below, the cat wins by starting at post 3, but loses if it starts at any other post.



Prove that the cat can always win if it starts on the right post. Prove this by induction on n for $n \geq 1$. Let your IH be that the cat can always win in the case where there are between 2 and $2(n-1)$ posts.

2. Design and analyze a data structure that maintains a bag of numbers and supports the following operations:

   - INSERT(x) inserts the number x into the bag.
   - DELETE-LARGER-HALF() deletes the largest $\lceil n/2 \rceil$ items from the bag where n is the bag size

Show how you can implement this so that the amortized cost of both operations is $O(1)$ and so that you can output all n numbers in the bag in $O(n)$ time.

3. Problem 16-2 (Making Binary Search Dynamic)

4. Professor Curly conjectures that if we do union by rank, *without path compression*, the amortized cost of all operations is $o(\log n)$. Prove him wrong by showing that if we do union by rank without path compression, there can be m MAKESET, UNION and FINDSET operations, n of which are MAKESET operations, where the total cost of all operations is $\Theta(m \log n)$.

5. Describe and analyze a data structure to support the following operations on an array $A[1 \ldots n]$ as quickly as possible. Initially, $A[i] = 0$ for all i.

   - **SetToOne(i)** Given an index i such that $A[i] = 0$, set $A[i]$ to 1.
   - **GetValue(i)** Given an index i, return $A[i]$
   - **GetClosestRightZero(i)** Given an index i, return the smallest index $j \geq i$ such that $A[j] = 0$, or report that no such index exists.

   The first two operations should run in worst-case constant time, and the amortized cost of the third operation should be as small as possible.