Illinois Institute of Technology
Department of Computer Science

Solutions to Third Examination

CS 330 Discrete Structures
Spring Semester, 2015

45 students took the exam; the statistics were:

Minimum	12
Maximum	98
Median	54
Average	57.78
Std Dev	20.53

1. In the implementation of BFS in the course notes, when we enqueue vertex v, we have only retrieved v_1 from queue, but not yet removed it. In the following solution, we assume that the dequeuing is done before the enqueuing; either order of operations is okay, however, and full credit was given for assuming either order.

   Initially, when the queue contains only s, the statement holds trivially. For the inductive step, we must prove that the lemma holds after both dequeuing and enqueuing a vertex. If the head v_1 of the queue is dequeued, v_2 becomes the new head. (If the queue becomes empty, then the lemma holds vacuously). By the inductive hypothesis, $d[v_1] \le d[v_2]$. But then we have $d[v_r] \le d[v_1] + 1 \le d[v_2] + 1$, and the remaining inequalities are unaffected. Thus, the lemma follows with v_2 as the head.

   When we enqueue a vertex v, it becomes v_{r+1}. At that time, we have already removed vertex u, whose adjacency list is currently being scanned, from the queue Q, and by the inductive hypothesis, the new head v_1 has $d[v_1] \ge d[u]$. Thus $d[v_{r+1}] = d[v] = d[u] + 1 \le d[v_1] + 1$. From the inductive hypothesis, we also have $d[v_r] \le d[u] + 1$, and so $d[v_r] \le d[u] + 1 = d[v] = d[v_{r+1}]$, and the remaining inequalities are unaffected. Thus, the lemma follows when v is enqueued.

2. Here is DFS as presented in the lecture of April 10, appropriately modified to set a color-bit in each vertex to either 0 or 1, if the graph is 2-colorable. The color-bit allows us to eliminate the white/gray/black coloring of the original DFS; we no longer need the time stamps:

   function DFS(G)

   for all $u \in V[G]$ **do**
       $bit[u] \leftarrow$ NIL
       $\pi[u] \leftarrow$ NIL
   end for
   for all $u \in V[G]$ **do**
       if $bit[u] =$ NIL **then**
           $bit[u] \leftarrow 0$
           DFS-visit(u)
       end if
   end for
   G is 2-colorable, as given by the color bits

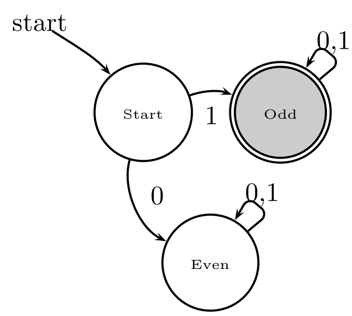
   function DFS-visit(u)

for all $v \in Adj[u]$ **do**
  if $bit[v] = \text{NIL}$ **then**
    $bit[v] \leftarrow 1 - bit[u]$
    $\pi[v] \leftarrow u$
    DFS-visit(v)
  else
    if $bit[v] = bit[u]$ **then**
      ABORT: G is not 2-colorable; the π values give an odd-length cycle starting at v
    end if
  end if
end for

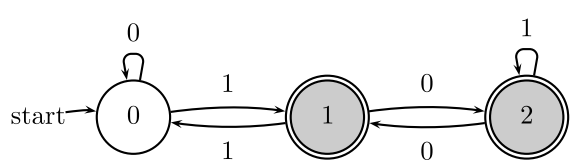
This is a (minor) embellishment of DFS as presented and analyzed in class on April 6, so the same analysis works to show the time required is $O(|V| + |E|)$. Of course, if there is an odd-length cycle, the graph cannot be 2-colored. If the algorithm succeeds, there is no odd-length cycle and the graph has been 2-colored.

3. The converse is false. To get a counterexample, take K_5, which we know to be non-planar, and add a sixth vertex connected by a single edge to one of the other vertices. We now have 6 vertices and 11 edges, so that $11 = |E| \leq 3|V| - 6 = 12$, but the graph is not planar because it contains K_5.

4. (a) Regular, recognized by the FSM, as in the lectures on both April 15 and 20:

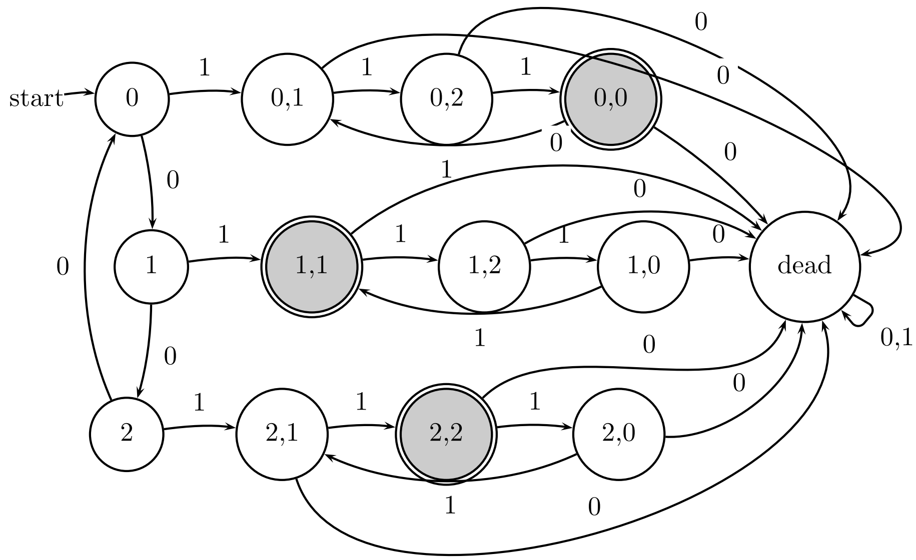


(b) Regular: accepted by an FSM similar to those discussed in class on April 15 and 20:



(c) Not regular: by contradiction. If L were regular, then because the intersection of regular languages is regular, $L \cap 0^*1^*$ would be regular; but this intersection is just $\{0^n1^n | n \geq 0\}$ which we know to be non-regular (class on April 22 or Rosen, example 6 on page 885).

(d) Regular: we can recognize it with a finite state machine consisting of 13 states:



(e) Not regular: by an argument similar to that given in class on April 22 (also in Rosen, example 6 on page 885): Suppose it were regular, accepted by an FSM with k states. Consider the action of the FSM on a string $0^{2k+1}1^k$. The FSM goes through $k+1$ states as it reads the k 1s, so it must repeat a state; the portion of the input between these two instances of the same state can be repeated as many times as we want, fooling the FSM into accepting a string that it should reject.

5. (a) Any string of zeros and ones in which the number of zeros is 1 mod 3.

   (b) 1*01*(1*01*01*01*)* because 1*01* is any string with a single zero, while (1*01*01*01*)* is any string with a multiple of 3 zeros.