Homework 4 Solutions
Fundamental Algorithms, Spring 2008, Professor Yap

Due: Wed Apr 2, in class.
HOMEWORK with SOLUTION, prepared by Instructor and T.A.s

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.
- Please write succinctly, to the point. Every sentence must be an complete English sentence.
- You may post general questions to the homework discussion forum in class website. Also, bring your questions to recitation on Monday.

---

1. (10 Points)
   Question 2.3 in Lect.IV (algorithm to determine if two closed paths p and q are equivalent).

   Give an algorithm which, given two closed paths $p = (v_0{-}v_1{-}\cdots{-}v_k)$ and $q = (u_0{-}u_1{-}\cdots{-}u_\ell)$, determine whether they represent the same cycle (i.e., are equivalent). The complexity of your algorithm should be $O(k^2)$ in general, but $O(k)$ for when q is a simple cycle. NOTE: Assume that vertices are integers, and the closed path $p = (v_0{-}\cdots{-}v_k)$ is represented by an array of integers $p[0..k]$, where $p[i] = v_i$ and $p[0] = p[k]$.

SOLUTION: We introduce the following procedures: Procedure $Find$ returns the position of vertex v in path p if found, -2 otherwise.

```
int  Find(int v, path p, int z (starting position) )
     i := z;
     while (i ≤ length(p))
          if (v = p[i]) return i;
          i := i + 1;
     return −1;
```

Procedure $Match$ returns True/False depending on whether path p matches path q starting at $q[j]$ modulo length L.

```
Bool Match(path p, path q, int j, int L)
     for  i = 0 to L
          if p[i] ≠ q[j + i mod L] return False;
     return True;
```

Procedure $Equal$ will determine if two given cycles are equivalent:

```
Bool Equal(path p, path q)
     if length(p) ≠ length(q) return False;
     L := length(p);
     j := 0;
     while j ≥ 0
          j := Find(p[0], q, j);
          if j < 0 return False;
          if Match(p, q, j, L) return True;
          j := j + 1;
```

The above solution will determine equivalence even of non-simple cycles. If L is the length of the cycles, the running time on simple cycles is $\mathcal{O}(L)$, since $Find$ and $Match$ will be called only once. The running time on non-simple cycles can be as high as $O(L^2)$ in the worst case.
To see the $O(L^2)$ behavior is possible, consider the closed paths $p = a^n ba$ and $q = a^{n+2}$.
We suggest you convince yourself that the behaviour is $O(L)$ when the paths are simple.
REMARK: this problem can be solved in linear time using more sophisticated algorithms from string pattern matching (e.g., the KMP algorithm).

2. (0 Points)
   Question 3.1 in Lect.IV (reversing a graph). Do not hand in, just for your thought.

3. (3 Points)
   Question 3.5 in Lect.IV. Show that $K_{3,3}$ is nonplanar. This should be a fun problem – we will lead you through.

   HINT: assuming it is planar, then $K_{3,3}$ can be embedded in the Euclidean plane. This partitions the plane into regions (called faces). Let there be f faces. NOTE that the number f includes the infinite face (that stretches to infinity). Now, the faces are each bounded by 4 edges or 6 edges (why?). But at most one face can be bounded by 6 edges (why?). By Euler's formula, we know that $v - e + f = 2$ (note that you know $v = 6$ and $e = 9$). Let S be the set of pairs of the form (f', e') where f' is a face and e' is an edge that bounds f'. Let us compute the count $|S|$ in two ways: $s_1 := \sum_{f'} |\{(f', e') : (f', e') \in S\}|$ and $s_2 = \sum_{e'} |\{(f', e') : (f', e') \in S\}|$. Clearly, $s_1 = s_2 = |S|$. But you should get different expressions for s_1 and s_2 in terms of v, e, f, and contradict Euler's formula.

SOLUTION: Assuming $K_{3,3}$ is planar, let it be embedded in the plane with f faces (this includes the infinite face). Note that $v = 6, e = 9$ and Euler's formula says $v - e + f = 2$. Let I be the number of edge-face incidences. Every cycle of $K_{3,3}$ is at least 4. If $d(f')$ denotes the number of edge-face incidences that a face f' is involved in, we conclude that $d(f') \geq 4$.
Using the hint, we consider the two expressions s_1 and s_2 for I. Summing over all faces f', we have

$$I = s_1 = \sum_{f'} d(f') \geq \sum_{f'} 4 \geq 4f.$$

But each edge e' is involved in exactly two such incidences. Summing over all edges e',

$$I = s_2 = \sum_{e'} 2 = 2e = 18.$$

It follows that $4f \leq= s_1 = s_2 = 18$ or $f \leq 4$. Then Euler's formula gives $2 = v - e + f \leq 6 - 9 + 4 = 1$. This is a contradiction!

4. (12 Points)
Question 4.8 in Lect.IV (detecting cycles in bigraphs).

Give an algorithm that determines whether or not a bigraph $G = (V, E)$ contains a cycle. Your algorithm should run in time $O(|V|)$, independent of $|E|$. You must use the shell macros, and also justify the claim that your algorithm is $O(|V|)$.

SOLUTION: Sketch: First, assume the bigraph is connected. We can use the BFS shell (Lect IV, §4), and stop as soon as a level or cross edge is discovered. (NOTE: We could also use DFS shell, and stop the moment a back edge is discovered.) Here are the shell macros to implement:
INIT(G, s): initialize the depth of each vertex, $d[u] = \infty$ if $u \neq s_0$ and $d[s_0] = 0$. We interprete u to be unseen iff $d[u] = \infty$.
VISIT(v, u): set $d[v] = 1 + d[u]$.
PREVISIT(v, u): if v is seen, we have detected a cycle. Return(CYCLE FOUND);
If after the BFS shell has terminated the main loop without finding a cycle, we Return(ACYCLIC).
In case the bigraph may not be connected, we just use the BFS or DFS driver to run as many searches as there are connected components.
Complexity analysis: Until a non-tree edge is found, each new vertex and edge is added to a growing forest of size at most n. The minute we find a non-tree edge, we terminate. This makes it clear that the running time is $O(|V|)$.

5. (12 Points)
Question 5.7 in Lect.IV (classifying edges using DFS tree of bigraphs).

Suppose T is the DFS Tree for a connected bigraph G. Recall our standard treatment of edges of a bigraph in DFS. Let $u{-}v$ be an edge of G. Prove that
(a) Either u is an ancestor of v or vice-versa in the tree T.
(b) If $u{-}v$ is a non-tree edge, it is a back edge.
(c) Give a complete classification of the edges as produced by the DFS algorithm.

SOLUTION:

(a) This is a consequence of the Unseen Path Lemma: suppose that u is visited before v. Then at the time we first see u, there is an unseen path from u to v (since $u{-}v$ is an edge). The Lemma tells us that v would become a descendent of u.

(b) Hence if the edge $u{-}v$ is not a tree-edge, then v is a descendent of u. That means that while we are exploring v, we will discover the edge from v to u, i.e., $v{-}u$ will become a back edge.

(c) All non-edges are back edges or unseen. In particlar, there are no forward or cross edges.

6. (12 Points)
The following is result is discussed in the notes:

LEMMA 1. *Let u be a vertex in the DFS tree T for a bigraph G. Then u is a cut-vertex iff one of the following conditions hold:*
(i) u is the root of T and has more than one child.
(ii) u is not the root, but it has a child u' such that for every descendent v of u', if $v{-}w$ is an edge, then w is also a descendent of u. Note that a node is always a descendent of itself.

Use this lemma to design an algorithm to detect cut-vertices in a connected bigraph. HINT: Let $ft(u)$ be the smallest value of `firstTime`$[w]$, where w is a vertex that can be reached by a back edge $v{-}w$, for some proper descendent v of u in the DFT tree; if there is no such back edge, then we define $ft(u)$ to be `firstTime`$[u]$. You need to address two questions: (a) How can $ft(u)$ help you determine whether a vertex v is a cut-vertex? (b) How can you compute $ft(u)$?