Honors Theory of Computation, Spring'00 (Yap).
HOMEWORK 4:
Out: Feb 29, 2000
Due: Mar 8, 2000

1 QUESTION 1

Page 68, Problem 3.4.8. Prove the recursion theorem as formulated by Papadimitriou.

SOLUTION: from Alexander Totok. We will check that $g(\overline{M_{f(g)}})$ is the required fixed point of f. By definition,

$$g(\overline{M_{f(g)}})(x) = \begin{cases} (M_{f(g)}(\overline{M_{f(g)}})(x) & \text{if } M_{f(g)}(\overline{M_{f(g)}}) \text{is a proper description of a TM} \\ \uparrow & \text{otherwise} \end{cases}$$

But $M_{f(g)}(\overline{M_{f(g)}})$ is always a proper description of a TM because $M_{f(g)}$ computes the composition of f and g. Hence

$$g(\overline{M_{f(g)}})(x) = (M_{f(g)}(\overline{M_{f(g)}})(x) = f(g(\overline{M_{f(g)}}))(x).$$

This is exactly what we wanted.

2 QUESTION 2

Page 211, Problem 9.5.19. NP-completeness of of 2 node-disjoint path of bounded length.

Solution from Daniel Kristjanssen. Let us call this the 2-DISJOINT-PATH problem. Many students prefer to reduce PARTITION to this problem instead. We follow the hint of reducing 3SAT to 2-DISJOINT-PATH.

Given a 3CNF formula F, we construct a graph G as follows:

- Create two special nodes, s and t.
- For each variable x, we create two vertices, $TRUE_x$ and $FALSE_x$. (Actually, we will see that this is not quite right – you need to create two chains corresponding to $TRUE_x$ and $FALSE_x$, where the length of the chain is equal to the number of occurrences of the respective literals in clauses. But for now, let us skip this complication).
- We now introduce edges that form a system of paths from s to t which passes through the nodes $TRUE_x, FALSE_x$. Each path in our system will pass through exactly one of $TRUE_x$ or $FALSE_x$. Next, arbitrarily order the variables. If variable x is ordered before variable y, then each of the paths in our system will pass though the pair $(TRUE_x, FALSE_x)$ before the $(TRUE_y, FALSE_y)$.

  Suppose x is ordered just before y. Then we create the following gadget: $G(TRUE_x, FALSE_x, TRUE_y, FALSE_y)$ which introduces a new vertex, say $u_{x,y}$, and the four edges $(TRUE_x, u), (FALSE_x, u), (u, TRUE_y)$ and $(u, FALSE_y)$.
- Note that a path from s to t through these gadgets must choose one node from each pair of the form $(TRUE_x, FALSE_y)$. This amounts to a truth assignment. The length of this path is basically twice the number of variables. By inserting artificial nodes in this sequence, we can make sure that the path is exactly B. Call this path the "truth assignment" path.
- Now, for each clause C in F, introduce a new vertex $CLAUSE_C$. If $C = \{u, v, w\}$ then we insert the edges from $CLAUSE_C$ to each of the literals u, v and w.
- Next, we string all these clauses together in some arbitrary sequence. If C is followed by C', and $C = \{u, v, w\}$, then we introduce the three edges $(u, CLAUSE_{C'})$, $(v, CLAUSE_{C'})$ and $(w, CLAUSE_{C'})$. Of course, if C is the final class, we introduce the three edges (u, t), (v, t) and (w, t).

- Clearly, another path from s to t through this chain of clauses must pick one literal from each clause. Let us call this path the "Clause Path". Moreover, the picked literal must disjoint from the truth assignment path. The problem, of course, is that if two clauses are satisfied by the same literal, say x, then the clause path would go through $TRUE_x$ twice, which is illegal. But it is easy to modify the construction of the variable chain above so that each occurence of a literal in a clause has its own version the the $(TRUE_x, FALSE_x)$ pair.
- Clearly, any clause path has length roughly twice the number of clauses. But by inserting artificial nodes, we can make the clause path exactly B.

You now note three easy facts:

- (1) The transformation from F to G is computable in log space.
- (2) If F is satisfiable, then there exists 2 vertex-disjoint paths from s to t in G of length at most B.
- (3) Conversely, if there exists such a pair of paths, then F is satisfiable.