CSC 2420 Spring 2015, Assignment 2
Due: March 5 at start of class

NOTE: If you are taking the course for credit, then you may either work by yourself or with at most one other student taking the course for credit (but *not* the student with whom you collaborated in P1 if you collaborated in P1). You must specify with whom you are collaborating and the extent of collaboration. It is certainly preferable for you to solve the questions without consulting a published source. However, if you are using a published source then you must specify the source and you should try to improve upon the presentation of the result.

1. The following questions relate to computing least cost paths in a directed edge weighted graph.

   - Show that Dijkstra can give a wrong answer when applied to a graph with negative edges but no negative weight directed cycles.
   - Show how to adapt the Bellman-Ford DP algorithm for the single source least cost path problem so as to detect the presense of a negative cycle.
   - Can you do the same for the DP that was presented for the all pairs least cost paths problem?

2. Consider the secretary problem where now instead of the inputs coming from an adversary (and then presented in random order), the n inputs now come i.i.d. from a *known* finite distribution D. That is, we know the probability of drawing any given u in the *support* of D. Let $|D|$ denote $|support(D)|$.

   - Show how to use dynamic programming to compute the following table $P[k, u, v]$: if v is the k^{th} input and u is the maximum value amongst the first $k-1$ inputs, then $P[k, u, v]$ is the probability that a larger element will occur in the remaining $(n-k)$ inputs.
   - What is the complexity (as a function of n and $|D|$) for computing the entire table?

- How would you possibly use this table (and any other information you can compute within time polynomial in n and $|D|$) to define an algorithm that would determine whether or not to select the k^{th} input when it is revealed?

3. This question concerns local search for the exact Max-2-Sat problem where the input is a CNF formula with exactly 2 literals per clause. The goal is to maximize the number (or the total weight) of clauses that can be satisfied by some truth assignment. Khanna et al consider the locality gap achieved by oblivious and non-oblivious local search. (See L4 lecture notes and, in particular, the proof of the $\frac{2}{3}$ locality gap for the 1-flip neighbourhood oblivious search.) As suggested by Khanna et al, one can also obtain a locality gap of $\frac{3}{4}$ by an oblivious local search algorithm that defines the neighbourhood of a solution (i.e. truth assignment) to include flipping any single variable and also flipping all variables. Modify the $\frac{2}{3}$ locality gap result to obtain the improved locality gap for this larger neighbourhood.

4. Consider the makespan problem for the restricted machines model. That is, each job J_i is represented by a pair (p_i, A_i) where p_i is the processing time (or load) for the job and $A_i \subset \{1, \ldots, m\}$ is the subset of machines on which job J_i can be scheduled.

   - (a) Assuming all jobs have processing time $p_i = 3$, show how to modify the reduction of bi-partite matching to the max flow problem so as to compute an optimal makespan solution for these input instances.
     Aside: Note that the natural greedy online algorithm only gives a $\log m$ approximation for these instances where m is the number of machines.
   - (b) Now assume that all jobs have processing time $p_i \in \{1, 3\}$. Does your approach for case (a) above extend to this case? If so, how and if not, why?

5. Consider the following call routing makespan problem. There is an n node bi-directional ring network $G = (V, E)$ upon which calls $\{C_1, C_2, \ldots, C_t\}$ must be routed. That is, $V = \{0, 1, \ldots, n-1\}$ and $E = \{(i, i+1 \ mod \ n)\} \cup \{(i, i-1 \ mod \ n)\}$ and calls C_j are pairs (s_j, f_j) originating

at node s_j and terminating at node f_j. Each call C_j can be routed in a clockwise or counter-clockwise direction and and incurs a routing cost p_j on each directed edge it uses. The load L_e on any *directed edge* e is the sum of the routing costs for all calls using that edge. The goal is to minimize $\max_e L_e$.

- Formulate this problem as an IP. Indicate the intended meaning of each variable in the IP.
- Using an LP relaxation of this problem, show how to derive a constant approximation algorithm. What is the constant you obtain?

6. Consider the maximum matching problem. That is, given a graph $G = (V, E)$, find a subset of edges $E' \subseteq E$ such that for all nodes $u \in V$, the degree of u in $G' = (V, E')$ is at most 1. Let $IN(u) = \{e : e = (u, v) \in E \text{ for some } v \in V\}$. We can express the maximum matching problem as the following natural IP:
maximize $\sum_{e \in E} x_e$
subject to : $\sum_{e:e \in IN(u)} x_e \leq 1$ for all $u \in V$
$x_e \in \{0, 1\}$

   - Consider the LP relaxation $\mathcal{P}$ (in standard form) of this IP; that is, : maximize $\sum_{e \in E} x_e$
     subject to : $\sum_{e:e \in IN(u)} x_e \leq 1$ for all $u \in V$
     $x_e \leq 1$
     $x_e \geq 0$

     State the dual $\mathcal{D}$ of the primal $\mathcal{P}$ using dual variables y_u for $u \in V$. Can you explain this dual as the relaxation of a known optimization problem?
   - Suppose now that we restrict attention to bipartite graphs. Explain (from anything you already know without any IP/LP theory) why the value of the LP OPT equals the value of the IP OPT.

7. Consider the unweighted vertex cover problem.

(a) Suppose you have a polynomial time algorithm $\mathcal{A}$ to compute the *size* of an optimal vertex cover for some class $\mathcal{G}$ of graphs closed under removal of edges and nodes (e.g. bipartite graphs). Show how to use $\mathcal{A}$ to compute an optimal *solution* (i.e. a subset of the vertices) for the vertex cover problem restricted to graphs in $\mathcal{G}$.

(b) Use ideas from any of the above questions to argue why computing an optimal vertex cover in bipartite graphs can be done in polynomial time.

8. Perhaps one more question to follow.