CSOR W4246–Fall, 2020
Eleni Drinea

Homework 3 (110 points)

Out: Monday, November 30, 2020
Due: 11:59pm, Monday, December 7, 2020

Homework Instructions.

1. For all algorithms that you are asked to "give" or "design", you should

   - Describe your algorithm clearly in English.
   - Give pseudocode.
   - Argue correctness, even if you don't give an entirely formal proof.
   - Give the best upper bound that you can for the running time.

   You are also encouraged to analyze the space required by your algorithm but we will not remove marks if you don't, unless the problem explicitly asks you to analyze space complexity.

2. If you give a reduction, you should do so as we did in class, that is

   (a) Give the inputs to the two problems.

   (b) Describe in English the reduction transformation and argue that it requires polynomial time. (You do not need to give pseudocode.)

   (c) Prove carefully equivalence of the original and the reduced instances.

3. You should submit your assignment as a **pdf** file on Gradescope. Other file formats will not be graded, and will automatically receive a score of 0.

4. I recommend you type your solutions using LaTeX. For every assignment, you will earn 5 extra credit points if you type your solutions using LaTeX or other software that prints equations and algorithms neatly. If you do not type your solutions, make sure that your hand-writing is very clear and that your scan is high quality.

5. You should write up the solutions **entirely on your own**. Collaboration is limited to discussion of ideas only and you should adhere to the department's academic honesty policy (see the course syllabus). Similarity between your solutions and solutions of your classmates or solutions posted online will result in receiving a 0 in this assignment and possibly further disciplinary actions. You should list your collaborators on your write-up.

Homework Problems

1. (30 points) A *flow network with demands* is a directed capacitated graph with potentially multiple sources and sinks, which may have incoming and outgoing edges respectively. In particular, each node $v \in V$ has an integer *demand* $d(v)$; if $d(v) > 0$, v is a *sink*, while if $d(v) < 0$, it is a *source*. Let S be the set of source nodes and T the set of sink nodes.

   A *circulation with demands* is a function $f : E \to R^+$ that satisfies

   (a) *capacity constraints:* for each $e \in E$, $0 \leq f(e) \leq c(e)$.

   (b) *demand constraints:* For each $v \in V$, $f^{\text{in}}(v) - f^{\text{out}}(v) = d(v)$.

   We are now concerned with a decision problem rather than a maximization one: *is there* a circulation f with demands that meets both capacity and demand conditions?

   i. Derive a necessary condition for a feasible circulation with demands to exist.

   ii. Reduce the problem of finding a feasible circulation with demands to Max Flow.

2. (20 points) Similarly to a flow network with demands, we can define a *flow network with supplies* where each node $v \in V$ now has an integer *supply* s_v so that if $s_v > 0$, v is a *source* and if $s_v < 0$, it is a *sink*, and the supply constraint for every $v \in V$ is $f^{\text{out}}(v) - f^{\text{in}}(v) = s_v$.

   In a *min-cost flow* problem, the input is a flow network with supplies where each edge $(i, j) \in E$ also has a cost a_{ij} (the cost is per unit of flow). Given a flow network with supplies and costs, the goal is to find a feasible flow $f : E \to R^+$ —that is, a flow satisfying edge capacity constraints and node supplies— that minimizes the total cost of the flow.

   Show that max flow can be formulated as a min-cost flow problem.

3. (40 points) There are n villages and m directed roads forming a graph $G = (V, E)$. There is a wolf at village 1 and a rabbit at village n. There is no direct road from village 1 to village n. The wolf wants to catch the rabbit and can move around but the rabbit is stationed at village n.

   As a friend of the rabbit, you are helping it organize its defense.

   (a) Design and analyze an algorithm to determine whether or not the wolf can catch the rabbit. If the answer is yes, also determine the minimum number of roads the wolf has to traverse to catch the rabbit.

   (b) Suppose your answer to part (a) is yes. Your plan is to set up some traps inside villages 2 to $n-1$ to prevent the wolf from catching the rabbit. For every village i with $2 \leq i \leq n-1$, you need to set up t_i traps to make sure the wolf cannot get through this village (t_i is known and you may assume it is small constant).

   Traps are expensive so you want to install as few of them as possible. Design and analyze an efficient algorithm that returns (a) the minimum number of traps to install to guarantee that the wolf cannot catch the rabbit, and (b) the villages where the traps should be installed.

   Hint: Think how you can solve this problem by reducing it to a min cut problem in an appropriately defined flow network $G' = (V', E')$.

4. (20 points) Consider an instance of the Satisfiability problem where all literals in every clause are unnegated variables. Such an instance is called a *monotone* instance. For example,

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_4)$$

is a monotone instance. Monotone instances are trivially satisfiable: set *every* variable to 1 (true). However, the instance above would still be satisfied if, instead of setting $x_1 = x_2 = x_3 = x_4 = 1$, we only set $x_1 = x_2 = 1$. Our goal is to minimize the number of variables set to 1 so that the monotone instance evaluates to 1.

State the decision version of this problem and prove it is NP-complete.

Problem 3 was contributed by Lei Duan. Enjoy!

EXTRA CREDIT exercise: *You do NOT have to solve this problem. If you do, you will receive extra credit for hw2t.*

1. (30 points) Given a bipartite graph $G = (X \cup Y, E)$, a matching M and a number k, we want to decide if there is a matching M' such that

   (a) M' has k more edges than M, *and*

   (b) every node $y \in Y$ that is an endpoint of some edge in M is also an endpoint for some edge in M'.

   i. Give a polynomial-time algorithm that takes as input a triplet (G, M, k) and outputs either a matching M' or **no**.

   ii. Give an example instance, that is, a triplet (G, M, k), such that there is some M' satisfying (a) and (b), but no edge from M belongs to M'.

   iii. Let

   - K_1 be the size of the largest matching M' such that every node $y \in Y$ that is an endpoint in M is also an endpoint in M';
   - K_2 be the size of the largest matching M'' in G.

   Prove that $K_1 = K_2$.

RECOMMENDED exercises: do NOT return, they will not be graded.)

1. *(Using reductions to prove $\mathcal{NP}$-completeness)*

   (a) A *clique* in an undirected graph $G = (V, E)$ is a subset S of vertices such that *all* possible edges between the vertices in S appear in E. Computing the maximum clique in a network (or the number of cliques of at least a certain size) is useful in analyzing social networks, where cliques corresponds to groups of people who all know each other.

   State the decision version of the above maximization problem and show that it is $\mathcal{NP}$-complete. *Hint: reduction from* `Independent Set`.

   (b) We say that G is a *subgraph* of H if, by deleting certain vertices and edges of H we obtain a graph that is, up to renaming of the vertices, identical to G.

   The following problem has applications, e.g., in pattern discovery in databases and in analyzing the structure of social networks.

   `Subgraph Isomorphism:` Given two undirected graphs G and H, determine whether G is a subgraph of H and if so, return the corresponding mapping of vertices in G to vertices in H.

   Show that `Subgraph Isomorphism` is $\mathcal{NP}$-complete.

   (c) Similarly, consider the following problem.

   `Dense Subgraph:` Given a graph G and two integers a and b, find a set of a vertices of G such that there are at least b edges between them.

   Show that `Dense Subgraph` is $\mathcal{NP}$-complete.

2. Suppose you are given n cities and a set of non-negative distances d_{ij} between pairs of cities.

   (a) Give an $O(n^2 2^n)$ dynamic programming algorithm to solve this instance of **TSP**; that is, compute the cost of the optimal tour and output the actual optimal tour.

   (b) What are the space requirements of your algorithm?

   Hint: Let $V = \{1, \ldots, n\}$ be the set of cities. Consider progressively larger subsets of cities; for every subset S of cities including city 1 and at least one other city, compute the shortest path that starts at city 1, visits all cities in S and ends up in city j, for every $j \in S$.