Sheetal Parikh
Foundations of Algorithms
EN.605.621.81.SU19
Homework #2

1. *[10 points] CLRS 34.3-2: Show that the ≤P relation is a transitive relation on languages. That is, show that if L1 ≤P L2 and L2 ≤P L3, then L1 ≤P L3.*

We want to prove that polynomial reducibility is transitive. As given by the problem, we know that $L_1 \leq_p L_2$ or that L_1 is a polynomial reducible to L_2. This implies that L_2 is at least as hard as L_1. Furthermore, we know that $L_2 \leq_p L_3$ or that L_2 is a polynomial reducible to L_3 implying that L_3 is at least as hard as L_2. As proven by Lemma 34.3 in our algorithms textbook, we also know that if $L_1 \leq_p L_2$ and L_2 is in P then L_1 is also in P. This implies that if there is a polynomial time algorithm for L_2, then there is a polynomial time algorithm for L_1 and vice versa.

With the information that we have, let's say that we have a polynomial program A that reduces L_1 to L_2 and we have a polynomial program B that reduces L_2 to L_3. If we pass an instance of L_1 represented by x through program A, we would get A(x). Then if we pass A(x) through program B, we would get B(A(x)). Overall through this entire procedure which we can represent as C , we would be turning an instance of L_1 into an instance of L_3. Similar to A and B, process C is also polynomial time. So if $x \in L_1 \Leftrightarrow A(x) \in L_2 \Leftrightarrow B(A(x)) \in L_3$ we can say that $x \in L_1 \Leftrightarrow C(x) \in L_3$. Therefore $x \in L_3$ proving that if $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ then $L_1 \leq_p L_3$.

2. *[15 points] Recall the definition of a complete graph Kn is a graph with n vertices such that every vertex is connected to every other vertex. Recall also that a clique is a complete subset of some graph. The graph coloring problem consists of assigning a color to each of the vertices of a graph such that adjacent vertices have different colors and the total number of colors used is minimized. We define the chromatic number of a graph G to be this minimum number of colors required to color graph G. Prove that the chromatic number of a graph G is no less than the size of the maximal clique of G.*

If we have a graph with a clique size of *k*, then we know that at least *k* colors are required to color just the clique portion because the clique is a subset of graph *G*. Since we are given in the problem that the clique graph is maximal, we also now that we need at least *k* colors to color the entire graph *G*. Since the chromatic number is the minimum number of colors required to color graph *G*, the chromatic number must be at least the clique size of *k,* in order to color the clique. However, the chromatic number could be greater than *k,* in order to color any vertices not part of the clique. Therefore, because graph *G* contains a clique, the chromatic number cannot be less than the size of the maximal clique.

3. *[20 points] Collaborative Problem: Suppose you're helping to organize a summer sports camp, and the following problem comes up. The camp is supposed to have at least one counselor who's skilled at each of the n sports covered by the camp (baseball, volleyball, and so on). They have received job applications from m potential counselors. For each of the n sports, there is some subset of the m applicants qualified in that sport. The question is "For a given number k < m, is is possible to hire at most k of the counselors and have at least one counselor qualified in each of the n-sports?" We'll call this the Efficient Recruiting Problem. Prove that Efficient Recruiting is NP-complete.*

To prove that the Efficient Recruiting problem is NP-Complete, we have to prove that the problem is NP or easily verifiable in polynomial time and that the problem is NP-Hard or as hard as any other problem in NP.

Proving Efficient Recruiting is NP:
If given the list of *k* counselors highered you can easily check in polynomial time that *k* is less than *m*, the number of potential counselors as well as check that all *n* sports have been covered.

Proving NP-Hard:
To prove that the Efficient Recruiting problem is NP-Hard we can reduce a known NP-complete problem to the Efficient Recruiting Problem. We can visualize the problem by representing each counselor, *k*, by a vertex. Each sport, *n*, can be represented by an edge between each counselor or vertex. We want to make sure that each sport is represented or each edge is covered between each subset of vertices. Overall in the problem, we want to find the minimum number of counselors or vertices that covers all sports or edges. This visualization closely resembles the vertex cover problem which is a known NP-complete problem. As noted in the Module 4 Famous NP-Complete Problem Notes, the object of the vertex cover problem , "is to find the fewest vertices in a graph that, when taken together, contains at least one of the endpoints of all of the edges in the graph"

If we have the intended situation in which each counselor is qualified in each of the *n*-sports, we would have *k* counselors that are qualified and cover all the sports. Thus, every vertex, or counselor would have at least one incoming edge, or sport, creating a vertex cover the same size as the number of counselors, *k.* Overall, the only situation in which we would have a vertex cover of *k* size would be when we have at most *k* counselors. Therefore the $Vertex\ Cover\ Problem \leq_p Efficient\ Recruiting\ Problem$ and because the vertex cover problem is NP - complete, we can say that the efficient recruiting problem is NP - complete.

4. *[20 points] Collaborative Problem: We start by defining the Independent Set Problem (IS). Given a graph G = (V, E), we say a set of nodes S ⊆ V is independent if no two nodes in S are joined by an edge. The Independent Set Problem, which we denote IS, is the following. Given G, find an independent set that is as large as possible. Stated as a decision problem, IS answers the question: "Does there exist a set S ⊆ V such that |S| ≥ k?" Then set k as large as possible. For this problem, you may take as given that IS is NP-complete. A store trying to analyze the behavior of its customers will often maintain a table A where the rows of the table correspond to the customers and the columns (or fields) correspond to products the store sells. The entry A[i, j] specifies the quantity of product j that has been purchased by customer i. For example, Table ?? shows one such table. One thing that a store might want to do with this data is the following. Let's say that a subset S of the customers is diverse if no two of the customers in S have ever bought the same product. A diverse set of customers can be useful, for example, as a target pool for market research. We can now define the Diverse Subset Problem (DS) as follows: Given an m × n array A as defined above and a number k ≤ m, is there a subset of at least k customers that is diverse? Prove that DS is NP-complete.*

To prove that the Diverse Subset problem is NP-Complete, we have to prove that the problem is NP or easily verifiable in polynomial time and that the problem is NP-Hard or as hard as any other problem in NP. We are already given that the Diverse Subset problem closely resembles the Independent Set Problem and so we need to prove NP-Hard we need to show that $Independent\ Set\ Problem \leq_P Diverse\ Subset\ Problem$.

Proving Diverse Subset is NP:
If given a set *S* of *k* customers, it can be easily verified in polynomial time that there are no common products between customers.

Proving NP-Hard:
To prove the Diverse Subset problem is NP-Hard we can reduce the Independent Set problem to the Efficient Recruiting Problem. We can visualize the problem by representing an array of customers and products as a graph. Each customer, *n* from the array, would be a node. If a product, *m* from the array, is common between customers then an edge is drawn. If there are no connecting edges, then there is no common product between the customers. In the Independent Set Problem, if given a graph *G = (V, E)*, we say a set of nodes $S \subseteq V$ is independent if no two nodes in *S* are joined by an edge. Similarly, in the diverse subset problem, if we are given an array *A,* there exists an independent set *S* of *k* if no two customers have purchased the same product as the nodes are not joined by an edge. We can say the set, *S* is also a diverse subset because this also indicates that no two customers have ever brought the same product. Therefore the $Independent\ Set\ Problem \leq_P Diverse\ Subset\ Problem$ and because the independent set problem is NP - complete, we can say that the diverse subset problem is NP - complete.

5. *[15 points] CLRS 5.3-3: Suppose that instead of swapping element A[i] with a random element from the subarray A[i...n], we swapped it with a random element from anywhere in the array.*

Algorithm 1 Permute with All
function PERMUTEWITHALL(A)
n ← A.length
for i ← 1 to n do
swap A[i] with A[RANDOM(1,n)]
end for
end function

Does this code produce a uniform random permutation? Why or why not?

The code does not produce a uniform random permutation when you choose a random integer from 1 to n. By selecting a random element from anywhere in the array, you will have n^n possible swaps instead of $n!$. This produces sequences that are more likely to occur than others and so is not uniform. For example, let's say we have three cards A, B, & C. By using the algorithms above, we will produce 27 possible outcomes (3^3) each with an equal probability. If we review all the 27 outcomes, we see that there are actually 6 sequences, each of which would occur at a different number of times. The six sequences and the number of times that they occur would be the following:

ABC = 4
ACB = 5
BCA = 5
BAC = 5
CAB = 4
CBA = 4

We can see that certain sequences, such as ACB, BCA, and BAC are more likely to occur than the other sequences, ABC, CAB, and CBA. Furthermore, we realize that we have 27 outcomes but 6 different sequences. Therefore, if each sequence had equal probabilities of occurring, each permutation would have to occur an x number of times to produce a probability of ⅙, due to there being 6 sequences:

$$\frac{1}{27}\ x\ =\ \frac{1}{6}$$

No integer exists to satisfy the equation since 6 is not divisible by 27. It is not possible that the code above produces a uniform random permutation.

6. *[20 points] Collaborative Problem: A number of peer-to-peer systems on the internet are based on overlay networks. Rather than using the physical internet as the network on which to perform computation, these systems run protocols by which nodes choose collections of virtual "neighbors" so as to define a higher-level graph whose structure may bear little or no relation to the underlying physical network. Such an overlay network is then used for sharing data and services, and it can be extremely flexible compared with a physical network, which is hard to modify in real time to adapt to changing conditions. Peer-to-peer networks tend to grow through the arrival of new participants who join by linking*

into the existing structure. This growth process has an intrinsic effect on the characteristics of the overall network. Recently, people have investigated simple abstract models for network growth that might provide insight into the way such processes behave in real networks at a qualitative level. Here is a simple example of such a model. The system begins with a single node v1. Nodes then join one at a time; as each node joins, it executed a protocol whereby it forms a directed link to a single other node chosen uniformly at random form those already in the system. More concretely, if the system already contains nodes v1, ..., vk−1 and node vk wishes to join, it randomly selects one of v1, ..., vk−1 and links to that node. Suppose we run this process until we have a system consisting of nodes v1, ..., vn; the random process described above will produce a directed network in which each node other than v1 has exactly one outgoing edge. On the other hand, a node may have multiple incoming links, or none at all. The incoming links to a node vj reflect all the other nodes whose access into the system is via vj ; so if vj has many incoming links, this can place a large load on it. Then to keep the system load-balanced, we would like all the nodes to have a roughly comparable number of incoming links. That is unlikely to happen, however, since nodes that join earlier in the process are likely to have more incoming links than nodes that join later. Let us try to quantify this imbalance as follows.

(a) [10 points] Given the random process described above, what is the expected number of incoming links to node vj in the resulting network? Give an exact formula in terms of n and j, and also try to express this quantity asymptotically (via an expression without large summations) using Θ(·) notation.

The probability of V_k can be represented by $\frac{1}{k-1}$. V_k represents the nodes that come after V_j since we have been given that V_k randomly selects from $k-1$ nodes. Therefore $\frac{1}{k-1}$ also represents the probability of incoming links to node V_j. The probability of nodes that come before V_j would be 0. The formula in terms of n and j representing the expected number of links to V_j can be simplified as the following:

$$\sum_{k=j+1}^{n} \frac{1}{k-1} = \sum_{k=1}^{n-1} \frac{1}{k} - \sum_{k=1}^{j-1} \frac{1}{k}$$

(#Harmonic Numbers : $\sum_{k=1}^{n-1} \frac{1}{k} = H(n-1)$)

$$= H(n-1) - H(j-1)$$

$$= \theta(\ln n) - \theta(\ln j)$$

$$= \theta(\ln \tfrac{n}{j})$$

(b) [10 points] Part (a) makes precise a sense in which the nodes that arrive early carry an "unfair" share of connections in the network. Another way to quantify the imbalance is to observe that, in a run of this random process, we expect many nodes to end up with no incoming links. Give a formula for the expected number of nodes with no incoming links in a network grown randomly according to this model.

As mentioned in part a, $\frac{1}{k-1}$ represents the probability that of incoming links to V_j and 0 represents the probability of incoming links prior to V_j. Therefore, the probability of no incoming links can be represented by $1 - \frac{1}{k-1}$. To find the expected number of nodes we can

first find the total probability of no incoming links to V_j. Then that equation can be used to find the sum of each outcome. If we choose a random variable X_j for nodes with no incoming links , we know the either X_j is 1 if V_j does not have an incoming note or X_j is 0 otherwise.

Total Probability(no incoming links to V_j) = $E\left[X_j\right]$

$$= \prod_{k=j+1}^{n} \left(1 - \frac{1}{k-1}\right)$$

$$= \left(1 - \frac{1}{j}\right)\left(1 - \frac{1}{j+1}\right)\left(1 - \frac{1}{j+2}\right)\left(1 - \frac{1}{j+3}\right)\ldots..$$

$$= \left(\frac{j-1}{j}\right)\left(\frac{j}{j+1}\right)\left(\frac{j+1}{j+2}\right)\left(\frac{j+2}{j+3}\right)\ldots.$$ Converges to $\frac{n-2}{n-1} = \frac{j-1}{n-1}$

(we can see at each value of k until n, the numerator is always 2 less then the value of k and the denominator is always 1 less then the value of k.)

$$= \frac{j-1}{n-1}$$

Finding the Expected Number of Nodes with no incoming links:

$$\sum_{j=1}^{n} E\left[X_j\right] = \sum_{j=1}^{n} \frac{j-1}{n-1}$$

$$= \frac{1}{n-1} \sum_{j=1}^{n} (j-1) \quad \text{\# arithmetic series}$$

$$= \frac{1}{n-1} \frac{n(n-1)}{2}$$

$$= \frac{n}{2}$$