CS 3510 - Spring 2009
Homework 4
Due: April 15

1. Problem 6.3 from [DPV] In this problem we define P[i] as the maximum possible profit attainable, by placing a restaurant at m_i miles from the start of this highway.
   Following the constraints mentioned in the question we define $P[i]$ recursively as follows,
   $P[i] = \max_{j<i}[\ P[j] + \beta(m_i, m_j) \times p_i, p_i]$
   The function β is defined as: $\beta(m_i, m_j) =$

$$\begin{cases} 1, & \text{if } m_i - mj \geq k \\ 0, & \text{if } m_i - mj < k. \end{cases}$$

   Now let us see what is happenning in the above recursion, we try to maximize the profit at every possible restaurant location by considering which of the previously known best solutions fetch us the maximum profit when we place a new restaurant at position. For this, we iterate over the entire list of possible restaurant locations and we use the optimal values of profit at these locations calculated in past by the same recursion. We maximize on addition of this previous best profit and the profir obtained by having a new restaurant at ith position. The only thing remaining to check now, is whether it is feasible to have this combination, that function is performed by the β function which determines if the two locations are atleast k miles apart as per the constraint mentioned in this problem.

   The base case is the first location on the highway which will give the maximum profit of P_1

   The pseudo code can be written as follows:
   procedure expected-profit(N,P)
   Input: N locations; P[1,..,N] where P[i] denotes profit at location i Output: Maximum expected profit Pmax
   Declare an array of maximum Expected profit Profit[1..N]:

Profit[i] denotes maximum expected profit at location i
for i = 1 to N:
Profit[i] = 0
for i = 2 to N
for j = 1 to i-1
temp = Profit[j] + $\beta(mi, mj) \times P[i]$
if $temp > Profit[i]$:
temp = Profit[i]
if $Profit[i] < P[i]$:
Profit[i] = P[i]
Based on the above mentioned strategy the optimal solution for the given set of markers can easily be verified. The algorithm is $O(n^2)$.

2. Problem 8.4 from [DPV]

(a) Clique-3 is in NP Clique-3 Problem is just the clique Problem with additional constraint that the degree of every vertex is restricted to 3. Now to prove that this problem is in NP, we have to prove that given a proposed solution to this problem, we can verify in polynomial time (on non deterministic turing machine) whether it is the solution.
Here a point to note is that Clique-3 is not restricting the size of the clique to 3, but its restricting the degree of all vertices to atmost 3. Verifying a solution for this problem is trivial, given a set of nodes and edges which form the clique we can easily check if those nodes are maximally connected and whether the given edges actually belong to the original graph in $O(|V| \times |E|)$ time.

(b) What is wrong with the following proof of NP-completeness for CLIQUE-3?
We know that if problem Y is NP-Complete and X is a problem in NP and Y ¡=p X, then X is NP Complete.

Now we know the CLIQUE Problem which is NP-Complete, to show that CLIQUE-3 is in NP-complete, we need to reduce the problem of CLIQUE to CLIQUE-3. The direction of reduction in the presented argument is incorrect.

(c) Solution:
The statement that C is a vectex cover of G iff the complementary set V-C is a clique in G is wrong. For example, consider a graph of four nodes V=1, 2, 3, 4, and E= 1,2, 1,3, 1,4, C= 1 is a vertex cover, V-C=2,3,4 is not a clique. Thus the above proof is incorrect.

3. Problem 8.9 from [DPV]
Solution:

It is easy to verify that there exists a polynomial time algorithm for checking a proposed solution for hitting set, that means given a set S' we can easily verify by comparing each of its elements with elements of each of C where C is a set of subsets of set S, this can be done in $O(n^2)$, thus this problem is NP. To see why HITSET is NP-complete, we observe that we can reduce VERTEX-COVER to HITSET: Given a graph G = (V;E), we set S = V and C = E, in this case when we say C = E we model C as set of subsets of V where each subset contains the endpoints of and edge, then immediately we establish the following: G has a vertex cover of size k if and only if C has a hitting set of size k. This fact is very easy to verify, As VERTEX-COVER is NP-complete, the reduction (obviously) takes polynomial time, and HITSET is in NP, we have proved that HITSET is NP-complete.

4. Problem 8.10 from [DPV]
(a) SUBGRAPH ISOMORPHISM:
Subgraph isomorphism is a special case of CLIQUE. Now, given a graph G and an integer K, build a complete graph H such that H has K vertices. Clearly G has a subgraph G' isomorphic to H iff G has a clique with at least K vertices (each clique of size K' ¿ K has a clique of size K as a subgraph).

(b) LONGEST PATH:
The longest path problem is a generalizaiton of the Hamiltonia path problem. To see that we just need to note that Hamiltonian path is a special case of the LOGEST PATh problem by setting g to be —V—.

(c) MAX SAT:
MAX-SAT is a generalization of the SAT problem. To see this, we just need to note that SAT is a special case of MAX SAT by setting g = the total number of clauses in the formula.

(d) DENSE SUBGRAPH:
DENSE SUBGRAPH is clearly a generalization of CLIQUE problem where instead of complete connectivity between nodes and a set of K nodes we have the two parameters a and b mentioned in the problem.

(e) SPARSE SUBGRAPH:
Again for similar reasons its a generalization of CLIQUE.

5. Problem 8.12 from [DPV] Solution: (a) It is obviously a search problem, since there is exponentally big space of possible solutions from which we try to find the correct solution. (b)

(a) We must show that given a solution S to the k-Spanning Tree problem, that we can check in polynomial time whether it is in fact a k-Spanning Tree. This means we have to verify that every node in the original G is used in S, that S has no cycles (i.e. it is a tree), and that every node in the tree has degree at most k. All of these can be checked efficiently, and therefore k-Spanning Tree is a search problem.

(b) We have already shown that k-Spanning Tree is in NP in part (1). We now must show that we can solve some already known NP-complete problem given an efficient solution to k-Spanning Tree. We will show that we can solve the Rudrata Path problem given an efficient algorithm,call it KST for k-Spanning Tree. Let G be an unweighted undirected graph for which we would like to decide whether or not a Rudrata Path exists in. Imagine running KST with k=2 on G where we add weights equal to 1 on every edge. Notice that at tree that has each vertex with degree at most 2 is a path. If KST says YES, then that means there exists a path that spans all the vertices in G (i.e. a Rudrata Path). If it says NO TREE EXISTS, then there is no path without loops that touches all the vertices (i.e. no Rudrata path). We have therefore shown that Rudrata Path can be solved using k-Spanning Tree. In other words, we have reduced Rudrata Path to k-Spanning Tree, which along with the fact that k-Spanning Tree is in NP, shows that it is NP-complete.

6. Problem 8.18 from [DPV] Use of RSA involves "public key cryptography": A user of the system publishes the product pq, but doesn't publish p, q, or (p-1)(q-1). That way anyone can send a message to that user by using the RSA encryption, but only the user can decrypt it. Breaking this scheme can also be thought of as a different NP problem: given a composite number pq, find a factorization into smaller numbers.
One can test a factorization quickly (just multiply the factors back together again), so the problem is in NP, solving the problem seems difficult, there is a general argument on the fact that this problem may not be in P and NP both, but given P = NP, factorization can be done in polynomial time, and thus eavesdropper can factorize the product $\phi(n)$ and get the secrete numbers p and q. Thus RSA cryptosystem can be broken in polynommial time given that P = NP.