CS 581 Homework 10 Solution

Due on 03/26/2018 5:00 pm EST

Problem 1.
Consider an RSA key set with $p = 11$, $q = 29$, $n = 319$, and $E = 3$.

a) What value of D can be used in the secret key? Justify your choice.

Answer: Since we have E we need to find a value for D where $DE \equiv 1 \ (mod \ (p-1)(q-1)) \rightarrow DE \equiv 1 \ (mod \ 280)$ and D is co-prime to 280. For this system we can choose $D = 187$. This gives us $DE = 3 \times 187 = 561$ and $561 \equiv 1 \ (mod \ 280)$. Since $gcd(187, 280) = 1$ we know that 187 and 280 are coprime so $D = 187$ is a valid key.

b) What is the encryption of the message $M = 100$?

Answer: The encrypted message $C = M^E \ (mod \ n)$. So,

$$C = M^E \ (mod \ n)$$
$$C = 100^3 \ (mod \ 319)$$
$$C = 1000000 \ (mod \ 319)$$
$$C = 254$$

c) What message space is provided by this system?

Answer: The message space for this system would be the set {0,...,318}.

d) Identify all unconcealable messages, and state which depend on this particular choice of E. You may write a small program to help you if you wish.

Answer: There are two kinds of unconcealable messages. Any $C = M$ or $C = p$ or $C = q$ won't work.

To find these messages the following pseudocode could be used.

```
for M in {0, 1, ..., 318}
  C = M^E (mod n)
  if C == M:
    display M
  else if C == p:
    display M
  else if C == q:
    display M
  end if
end for
```

Executing the program we find that these numbers are $\{0, 1, 87, 88, 144, 175, 231, 232, 318\}$ ($C = M$) and {44,116} ($C = q$ *or* q).

Next we can have a program to find all possible Es. Many possible Es are valid because at least one corresponding E exits, such as 3, 9, 11, 13, 17, 19, 23, 27, 29, 31, 33, etc. Checking those we find a pattern that $\{0, 1, 87, 88, 144, 175, 231, 232, 318\}$ are not dependent on the value of E, but {44,116} are.

Problem 2.
Based on the information provided in each of the following scenarios, answer "yes (Y)", "no (N)" or "Unknown (U)" based on what we can conclude about whether A is a member of P, a member of NP, and/or NP-hard. You do not need to justify your answers. Put your answers in the table below. Note that co-P and co-NP are the respective complements P and NP. And no assumption is made whether $P = NP$.

a) B is NP-hard, B $\in NP$ and A $\propto$ B.

b) B is NP-hard, B $\in NP$ and B $\propto$ A.

c) B $\in NP$ and A $\propto$ B.

d) B $\in NP$ and B $\propto$ A.

e) B $\in$ co-NP and A $\propto$ B.

f) B $\in$ co-P and B $\propto$ A.

Question	Is A $\in P$	Is A $\in NP$	Is A NP-hard
a)	U	Y	U
b)	U	U	Y
c)	U	Y	U
d)	U	U	U
e)	U	U	U
f)	N	U	U

Here is a brief explanation for some of them:

a) B is NP-hard, B $\in NP$ and A $\propto$ B. A is at most as hard as B so it should be in NP. It could be in P or not in P. It could in NP-complete (implies in NP-hard too), or in non-NP-complete NP. It's possible to poly-time reduce one NP-complete problem to another NP-complete problem.

b) B is NP-hard, B $\in NP$ and B $\propto$ A. A could be some non-NP-complete NP-hard problem, in which case A is not in NP. A could also be in P if P = NP = NP-complete.

c) B $\in NP$ and A $\propto$ B. A is definitely in NP since you can poly-time it to B. Whether A can be in P depends on whether P = NP. A could be in NP-hard, and could also be just another non-NP-complete NP problem.

d) B $\in NP$ and B $\propto$ A. This one doesn't really give you much information. A could be in non-NP-complete NP, or non-NP-complete NP-hard. Whether A can be in P depends on whether P = NP.

e) B $\in$ co-NP and A $\propto$ B. A could be in P, it's possible to polytime reduce a P problem to a non-NP-complete NP-hard problem. A also could be a non-P NP problem. A could be in NP or could also be in non-NP-complete NP-hard.

f) B $\in$ co-P and B $\propto$ A. A is definitely not in P because if A is in P, B should also be in P. Other than this, we can make any conclusion about if A is in NP or NP-hard.

Problem 3.
A dominating set for a graph $G = (V, E)$ is a subset D of V such that every vertex not in D is adjacent to at least one member of D. The domination number $\gamma(G)$ is the number of vertices in a smallest dominating set for G.
Given a graph G and a positive integer p, the decision problem of dominating set is defined as: *is there a dominating set of size p for G?* Suppose we have an algorithm A that can answer "yes" or "no" for a given decision problem of dominating set. Use A as subroutine to solve:

a) The optimization version of dominating set (determine the smallest value of p).

Answer: Set $p = 1$ and check if $A(G, p)$ returns "yes." If it doesn't, increment p by one and try again. Do this until you find the smallest value of p where $A(G, p)$ returns "yes."

b) The search version of dominating set (find some dominating set of size at most p).

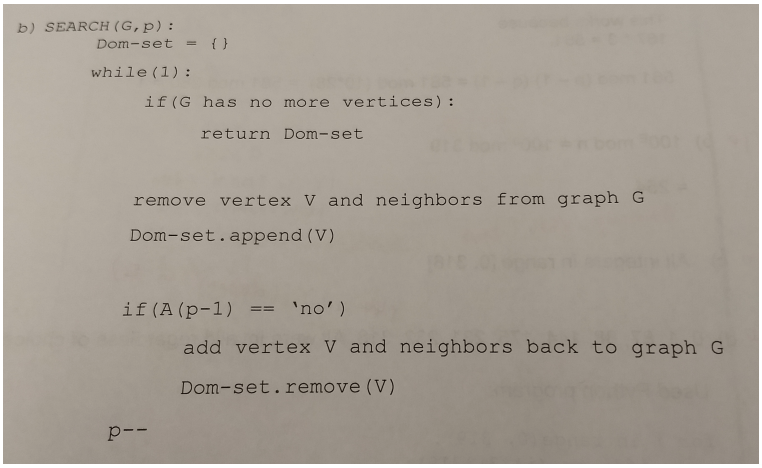
Answer: Note that the following algorithm only finds one possible dominating set. We refer to vertices in the found dominating set as black node, and others as white node. The idea is that, for a black node, if we add a new vertice and connect with it, the smallest p won't change for the new graph. But if we connect a new vertice to a white node, the smallest p will need to increment, since the new node is not adjacent to any black node.

Let the algorithm in problem a) be algorithm B. The following pseudocode uses subroutines A and B to find a dominating set of at most size p. B could be implemented by A, so for the sake of simplicity, we use both A and B here instead of just A.

```
FIND-DOMINATING-SET(G, p):
    dom_set = new empty set
    min_p = B(G)
```

```
if min_p > p:
  // no dominating set of size p or less exists
  pass
else:
  for v in the original G.V:
    introduce a new vertice vn and connect vn and v
    G' = updated G with the new node vn
    new_min_p = B(G')
    if new_min_p equals min_p:
      dom_set.add(v)
      G = G'
    end if
  end for
end if
return dom_set
```

Here's another working solution that was a number of people's answer that is based on removing nodes as opposed to adding nodes. I copied and pasted it here.



```
b) SEARCH(G,p):
      Dom-set = {}
      while(1):
          if(G has no more vertices):
              return Dom-set

          remove vertex V and neighbors from graph G
          Dom-set.append(V)

          if(A(p-1) == 'no')
              add vertex V and neighbors back to graph G
              Dom-set.remove(V)
       p--
```