Lai Wei
laiw12@bu.edu
U37976440

CS332 Homework 4

1.
Question: Given a graph with n vertices, how many 4 cliques can it have (at most) ?

If we want to have the maximum number of 4- clique in the graph, the graph should be assumed to be a n-clique with the vertex size of n. Thus, the question is reduced to:

Given a n-clique graph with n vertex, how many 4 cliques are there? Which is same as asking given an n vertex graph, how many different subgraphs of size 4 can be chosen ?
As a result, the answer is clearly(using the formula of combination):

nC4 = n ! / 4!(n - 4)! = n!/ 24 * (n-4)!
= 1/24 * (1*2*3*4*.....*n/ 1*2*3*4…..*n-4)
= 1/24 * (n-3)*(n-2)*(n-1)*n

Note that the value of (n-3)*(n-2)*(n-1)*n is approximately n^4. If we are to develop an algorithm to solve this problem, the running time is in O(n^4). I will use this running time to solve next question.

Question: show that L = { G | G contains a 4- clique } is in P.

To prove a problem is in P, we just have show that there is an algorithm to solve this problem that runs in polynomial time. As a result, I will present an algorithm 4-clique(G) that returns True if on input graph G that G contains a 4-clique. The function returns False vice versa.

Helper function: check(M) where M is an input of a 4-vertex graph and returns True if graph G is a 4-clique and returns False otherwise.

```
check(M){
 for each vertex in M{
        If this vertex is not connected to all other vertex in M{
                              return False
}
}
return True
}
```

Then, put this helper function in the main function 4-clique(G):

```
4-clique(G){
For each subgraphs of 4-vertex{
              If check(subgraph) == True{
                              return True
              }
}
return false
}
```

Running time for this algorithm: (assume the number of vertex of G is n)

1. There are n^4 subgraphs to check on input graph G with the size of vertex n.
   This is proved in the previous question that the running time for this part is O(n^4).
2. For each subgraphs, it takes 16(4*4) steps to check if it is a 4-clique. (Based on different algorithms, this number may be different. However, it is a constant.

It’s not hard to see that the running time for this algorithm is O(n^4+16) = O(n^4). Hence, this problem may be solved in polynomial time.

2.

i. J = { G | G has n vertices and contains an n/2-clique}.
I will modify the algorithm above to adapt this question: Instead of checking of each subgraphs of 4 vertices, the algorithm should check each subgraphs of n/2 vertices. To calculate the running time of this algorithm, we need to calculate:

The value nCn/2 is approximately exponential, as a result, this problem is not in P

ii.
Show that K = {G | G has n vertices and contains n-clique} is in P.

I will introduce an algorithm n-clique(G) that returns true if graph G of n vertices contains n-clique. Note here, the input graph G is in the format of the adjacency matrix. The algorithm is as follows:

```
n-clique(G){
for i in range(n){
      for  j in range(n){
          If G(i,j) == 0 && ( i!= j){         (Note: i = j means to check if there is an edge to the vertex
               return False                    itself, which will not be considered to solve this problem.)
}
}
}
 return True
}
```

This algorithm is simple and it just goes through every element in the adjacency matrix. If there is a value of zero, then there are two vertices that are not connected.(return False). When all element in the matrix is checked,then True is returned.

Running time:
The running time of this algorithm O(n^2) because it simply goes through the n*n adjacency matrix. As a result this problem is in P since it can be solved in polynomial time.

3. Show that P is closed under intersection

Let A1 and A2 where A1 ∈ P and A2 ∈ P. We want to show that A1 ∩ A2 is also in P.

According to the definition 7.12 from the textbook, a class of languages is in P if it is decidable in polynomial time on a deterministic single-tape Turing machine. As a result :

If A1 is in P, then there is a deterministic single-tape TM M1 with complexity of O(n^c1) (c1 is a constant) that decides A1
If A2 is in P, then there is a deterministic single-tape TM M2 with complexity of O(n^c2) (c2 is a constant) that decides A2

The goal for us is to construct a TM D that decides A1 ∩ A2 that runs in polynomial time.

D =" On input w:

1. Run M1 on input w
2. If M1 rejects, reject.
3. If M1 accepts, run M2 on w
4. If M2 rejects, reject.
5. If M2 accepts, accept."

Since M1 and M2 runs in polynomial time, the running time for D is just:
O(n^c1) + O(n^c2) which is the same as O(n^a) where a is a constant.
As a result, D also runs in polynomial time and P is closed under intersection.

4.
Prove that TIME(N^4) is closed under difference.

According to definition 7.7 from the textbook, TIME(N^4) is the collection of all languages that are decidable by an O(t(n)) time Turing machine.

Let A1 and A2 be the collection of languages that are decidable by O(N^4) time Turing machine M1 and M2 respectively.
Hence, the goal for us is to construct an O(N^4) time Turing machine that decides A1-A2

D =" On input w :

1. Run M2 on W, if M2 accepts, reject
2. If M2 rejects, Run M1 on W,
3. if M1 rejects, reject. Otherwise, accept."

Obviously, D is a decider that decides A1-A2. We know that M1 and M2 runs in O(N^4). The approximate running time for D is just O(N^4) + O(N^4) which is the same as O(N^4). We have showed that there exists an O(N^4) time turing machine that decides A1-A2. Hence, TIME(N^4) is closed under difference.

5. Assume that a function f is in polynomial time and can be computed in time 0(n 7) and that g is in polynomial time and can be computed in time 0(n 2). Prove that f composed with g, that is f(g(x)), can be computed in time 0(n 14).

To solve this problem, we need to remember the definition of big "O" notation.
Basically, f(x) = O(g(x)) means that there exists a real number c and x0 that f(x) <= c *g(x) for all x >=x0.

Hence, In this question:

f(n) <= c1 *(n^7) --------------------------1
g(n) <= c2* (n^2) --------------------------2

Let M = g(n) <= c2 *(n^2) for every n > n0

Hence, f composed with g is :

f(g(x)) =f(M) <= c1*(M^7) using equation (1)
<= c1*((c2*(n^2))^7) using equation (2)
< = c1* (c2^7) * n^14

Since c1 and c2 are constant, as a result, f(g(x)) can be computed in O(n^14)