LAI WEI
laiw12@bu.edu
U37976440

CS332 Homework 5

1.
Question: Show that vertex cover problem is in NP.

Problem definition:
VC = {(G,k) | G is a graph and k is an integer and G has a vertex cover of size at most k}

What I need to do:
According to the definition of the book, NP is the class of languages that have polynomial time verifiers. Hence the goal for this proof is to find a polynomial time verifier.
Note that a polynomial time verifier for a language A is an algorithm V, where A = {w| V accepts (w,c) for some string c} that runs in polynomial time in the length of string w.
Basically, the proof will have three steps:

1. Find input W (Some graph G and some number k)
2. Find C (any subset of vertices in graph G of size k)
3. Construct an algorithm V that given W and C that return whether that set of vertices of size k is a vertex cover for G (V has to run in polynomial time).

Proof:
Let W be the input of some graph G =(V,E) and some number k.
Let C be any possible subset of vertices in graph G of size less or equal k.
Construct the algorithm V: (Pseudocode) on input “<<G,k>, C> {

```
C = the chosen subset of vertices with size less or equal k.
List_Edges = all the edges in graph G
For every edge(u,v) in List_edges:{
        If [u does not belong to C ] and [v does not belong to C]{
                Return NO (the subset is not a vertex cover)
}
}
Return YES ( the subset is a vertex cover)
}
```

Clearly, this algorithm runs in polynomial time and for the worst case the number of edges that needs to check is just n(n-1)/2 in an undirected graph. As a result, I a found a polynomial time verifier and vertex cover problem is in NP.

2.

Question: Show that NP is closed under intersection.

Assume L1 and L2 ∈ NP. According to Theorem 7.20 in the textbook, A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine. As a result:

L1 is decided by some nondeterministic polynomial time Turing machine.

L2 is decided by some nondeterministic polynomial time Turing machine.

Assume M1 and M2 such that:

M1 is a nondeterministic polynomial time Turing machine that decides L1 in nondeterministic time complexity O(n^k).

M2 is a nondeterministic polynomial time Turing machine that decides L2 in nondeterministic time complexity O(n^c).

I will construct a nondeterministic polynomial time Turing machine M3 that decides L1 ∩ L2.

M3 = “On input w:

1. Run M1 on w, If M1 rejects, reject.
2. Run M2 on w, If M2 rejects, reject.
3. If M2 accepts, accept.

Hence, It is obvious that M3 is also a nondeterministic polynomial time Turing machine that decides L1 ∩ L2. The reason is that we know that the time complexity for M1 and M2 are nondeterministic time complexity O(n^k) and O(n^c). The time complexity for M3 is just nondeterministic time complexity O(n^max(k,c)).

3.

The reason is that simply reverse accept and reject for the output of a NTime(n^k) algorithm will not give us the complement of L. Just reverse the accept and reject state is not enough.

Recall the difference between deterministic turing machine and nondeterministic turing machine:

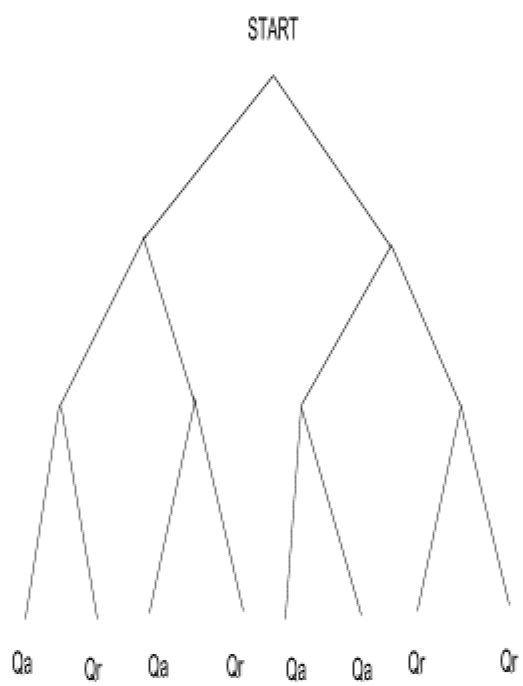
Deterministic Turing Machine: the set of rules prescribes at most on action to be performed for any given situation.

Nondeterministic Turing Machine: may have a set of rules that prescribes more than one action for a given situation.

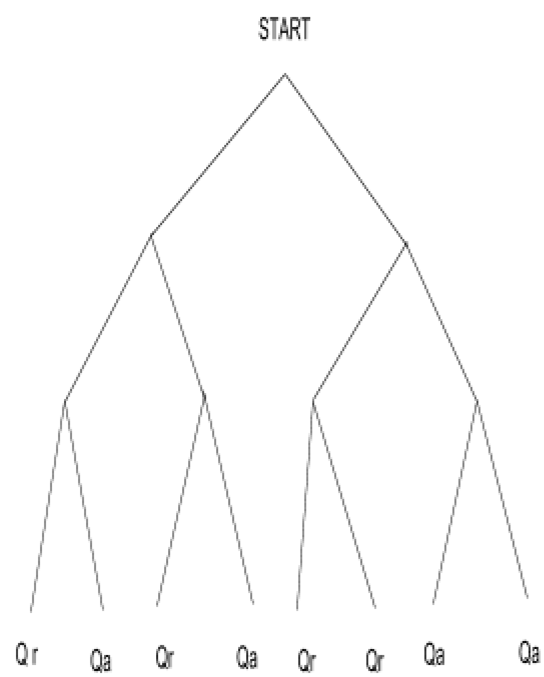
The nondeterministic Turing machine accepts its input by applying the transition function and eventually reach the accept state. However, at a given state, you can not determine exactly what their next state will be.It accepts the input if and only if there exists an accepting path in the computational tree. (If there is at least one sequence of choices that lead to an accepting state). Hence, for Nondeterministic Turing Machine, d simply reverse the accept and reject for the output will not give us the complement of L.

I will also demonstrate an example of the computational tree to show that reverse the accept and reject for the output will not give us the complement of L: Let Qa and Qr be the accept and reject state in the computational tree for the nondeterministic turing machine.

Original Version which accepts L:



Version that reverse accept and reject state:



If we still input language L into the reversed version(graph on the right), L may still be accepted because there still exist a sequence of choices that lead to an accepting state. As a result, filp the accept and reject will not give us the complement of L. (The NTM works in parallelism).

4.

According to the definition of polynomial time mapping reduction:

Language A is mapping polynomial time reducible to language B if there is a polynomial time computable function:

$$f : \Sigma^* \longrightarrow \Sigma^*, \text{ where for every } w,\ w \in A \Longleftrightarrow f(w) \in B.$$

Back to homework problem and apply this definition:

$A \leq_{(p,m)} B \quad \rightarrow \quad w \in A \Longleftrightarrow f(w) \in B$
$B \leq_{(p,m)} C \quad \rightarrow \quad w \in B \Longleftrightarrow g(w) \in C$

Thus, combine the equations above:
$w \in A \Longleftrightarrow f(w) \in B \quad \Longleftrightarrow \quad g(f(w)) \in C$
which is :
$w \in A \Longleftrightarrow g(f(w)) \in C$

Function *f* and *g* are polynomial time computable function which implies that g(f(w)) is also polynomial computable. Hence, A is polynomial time mapping reducible to C and <=(p,m) is transitive.

5.
Yes, NP is closed under polynomial time mapping reduction.

Here is a list of definition that I will use in my proof:

1. Definition 5.17: A function $f : \Sigma* \longrightarrow \Sigma*$ is a computable function if some Turing machine M, on every input w, halts with just f(w) on its tape.
2. Definition 5.20: Language A is mapping reducible to language B if there is a computable function: $f : \Sigma* \longrightarrow \Sigma*$, where for every w, $w \in A \Leftarrow\Rightarrow f(w) \in B$.
3. Definition 7.12: P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine.
4. Theorem 7.20: A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Proof:
If A <=(p,m) B, then there is some polynomial time deterministic Turing machine **M1** that for every w, $w \in A \Leftarrow\Rightarrow f(w) \in B$. (From 1, 2 and 3).

If B is in NP, then B is decided by some nondeterministic polynomial time Turing machine **M2**. (From 4).

I will construct a nondeterministic turing machine M3 that decides A using M1 and M2. Then I will prove that it runs in polynomial time.

M3 = “ on input A: (brief description)

1. Run M1 on w and record the results on the tape. (Reduce A to B in polynomial time).
2. Run M2 on the results recorded in step 1. (B is in NP and we can run M2 on B).
3. If M2 accepts, accept. Otherwise, reject.

Clearly, the nondeterministic turing machine M3 decides A and it runs in polynomial time. The reason is that M1 runs in polynomial time and M2 also runs in polynomial time. As a result, A is in NP according to definition 4 → NP is closed under polynomial time mapping reduction.

6.
Question: Show that the problem max-cut, as defined in problem 7.25 on page 296 is in NP.

Problem definition: MAX-CUT = {<G, k> | G has a cut of size k or more}.

What I need to do:
The goal for this problem is to still find a polynomial time verifier V that for language A such that A = {w| V accepts (w,c) for some string c} that runs in polynomial time in the length of string w.

Proof:
Let W be some graph G and some number k.
Let C be any subset of edges in graph G of size k.

Construct the algorithm V: (Pseudocode) “ on input <<G,k>, C>{

```
List_edges =  subset of edges in graph G of size k.

For each edge(u,v) in List_edges {
        If there is path from u to v {
                Return NO ( the set edges is not a cut)
        }
 Return YES ( every edge is checked and the graph is separated into two pieces)
}
```

Note that to check if there is a path from u to v, we can use graph traverse algorithm such as **Breadth-first search (BFS).** The time complexity for this algorithm is O(|V| + |E|) since every vertex and edges will be visited in the worst case. (The graph is not disconnected into two pieces). In a nutshell, the time complexity for this algorithm is in polynomial time since the maximum number of edges that an undirected graph can have is n(n-1)/2. A polynomial time verifier is constructed and MAX-CUT problem is in NP.