LAI WEI
laiw12@bu.edu
U37976440

CS 332 Homework 6

1.
Question:
Call graphs G and H isomorphic if the nodes of G may be reordered so that it is identical to H. Let ISO = {<G, H> | G and H are isomorphic graphs}. Show that ISO ∈ NP.

To show ISO is in NP, I will construct a polynomial time verifier V that decides two graphs G and H if they are isomorphic graphs.

Algorithm V : “on input <G = (V1,E1),H = (V2,E2), C >, where G and H are input graphs with adjacency matrix format and C is a permutation of {1, 2, 3, …, n} where n is |v1|.

1. Check if { |V1| = |V2| }. If not, reject.
2. Let the permutation of V1 in graph G be the same as C (take elements in C as indices). Check if the elements(edges) in adjacency matrix G (the permutations of V1) and H are the same. That is: if G(i,j) ≠ H(i,j), reject.
3. If step1 and step2 are passed, accept.

Time Complexity: The time complexity is is simple. Step1 checks the length of the vertices,which takes approximately linear time O(n). Step2 checks every element is the adjacency matrix which takes approximately O(n^2) time. The result time complexity is just roughly O(n^2) which implies V is a polynomial time verifier. Hence, ISO is in NP is proved.

2.
Question: SuperSat = { F | F is a propositional formula and F has at least 3 satisfying assignments} . Prove that SuperSat is NP complete.

To prove a problem is NP complete, we need to show:

1. SuperSat is in NP, and
2. SAT ≤p SuperSat

Since F is a propositional formula and F has at least 3 satisfying assignments

STEP 1: Prove SuperSat is in NP.

I will construct a polynomial time verifier V. The algorithm V takes the input of the propositional formula F and a string C that with the boolean assignments of the variables.
The algorithm simply plugs in all the boolean assignments and outputs the boolean results.
If V returns 1 → Satisfiable for the current assignments. If V returns 0 → not Satisfiable for the current assignments. You just have to repeat this algorithm for any three inputs. It is also obvious that this algorithm runs in O(n^2). Hence, SuperSat is in NP.

STEP 2: Prove SAT ≤p SuperSat

Let m1 and m1 be two additional variables apart from the input F. I will construct a new formula N that has at least 3 satisfying assignments if F is satisfiable.

N = F ∧ **(m1 V (¬ m1)) ∧ (m2 V (¬ m2))**
It is obvious bold part of this formula is always evaluated to TRUE, and there are always at least three satisfying assignments if F is evaluated to TRUE.
Here is the all possible assignments for m1 and m2:

M1	M2
T	T
T	F
F	T
F	F

The bold part will always be evaluated to TRUE based on these assignments.
Hence, I find a formula that reduce the SAT to SuperSat.
The reduction formula is correct because:

1. The satisfiability only depends on F which is a simple SAT problem.
2. The are at least 3 satisfying assignments iff F has a satisfying assignment.

3.
We know that SAT ∈ NP. If P = NP is assumed, then we can conclude: SAT ∈ P = NP.
Then there exist a polynomial time computable function to solve the problem below:
SAT = {<φ>| φ is a satisfiable Boolean formula}.
This offers a way that tells me whether a formula is satisfiable or not.

I will briefly construct an algorithm V that runs in polynomial time to produce a truth assignment which satisfies a input formula F.

```
V on input formula F with variables(x1,x2,x3,x4 …,xk) {
     If formula F is not satisfiable {
        return  “not satisfiable, cannot produce valid assignments”
    }

    for i in [1,k] {
        replace xi in formula F with 1
        If F is satisfiable {
            keep the change and keep looping
        }
        If F is not satisfiable {
            replace xi in formula F with 0 and keep looping
        }
    }

    Return F
}
```

Logic: The logic behind this algorithm is that if we know that a given input is satisfiable and each variable has to be either 1 or 0, we can set xi to 1 and test if the formula is satisfiable, if it is not satisfiable, then the value must be 0. Also this algorithm ensures F is satisfiable in the looping process.
Time Complexity: It is obvious that this algorithm runs in polynomial time. The loop will take n steps where n is the length of the input. The result time complexity is approximately:
O(n) * (The time needed to decide whether a formula is satisfiable).
Since we can compute the latter part in polynomial time, the overall time complexity for this algorithm is in O(n^c) where c is a constant which is a polynomial time complexity.

4.
Question: Show that subset sum is in linear space.

The space complexity of M is the function f : N → N , where f(n) is the maximum number of tape cells that M scans on any input of length n.
For this problem we need to show that there exists a Turing Machine M to solve this problem and the maximum number of tape cells that M scans on any input length is in O(f(n)) where f(n) is a linear function.

Following a similar proof from the textbook which shows SAT is in linear space:
Let F be the input set and t be the target number.
M = “on input <set F, t>:

1. For each subset of F:
2. Get the sum of each element in the subset.
3. If the the sum is equal to t, accept. If not, reject.”

Space Complexity: M runs in linear space because each iteration of the loop can reuse the same portion of the tape. The reason is that only the current subset is stored and the number of elements is at most n where n is the subset length. Hence, the machine runs in space O(n).

5.

i. Briefly state why NP is contained in PSPACE.

First NP is contained in NPSPACE. This is by the basic fact that TIME is contained in SPACE in lecture.

The Savitch's theorem also implies that PSPACE = NPSPACE. This is proved by following steps: (GIVEN in the question).

1. PSPACE ⊆ NPSPACE (this is by basic facts and definitions)
2. NPSPACE ⊆ PSPACE (by Savitch's theorem)

We know NP is contained in NPSPACE , and PSPACE = NPSPACE,
Hence, NP is contained in PSPACE.

ii. Prove that if some PSPACE-complete set C is in NP then NP = PSPACE.
To prove NP = PSPACE, We need to show:

1. NP ⊆ PSPACE (proved)
2. PSPACE ⊆ NP

Step 1 is proved in question (i). Then we need to show PSPACE ⊆ NP.
Given that some PSPACE-complete set C is in NP.

If a PSPACE-complete set C is in NP, It must satisfy the following properties:

1. C is in PSPACE. (by definition of PSPACE-complete)
2. Every A in PSPACE is polynomial time reducible to C. (by definition of PSPACE-complete)
3. C can be decided by a nondeterministic turing machine in polynomial time. (by definition of NP)

To show PSPACE is a subset of NP, I need to show: problems in PSPACE must be decided by a nondeterministic turing machine in polynomial time.Hence, I will construct a nondeterministic turing machine M that decides problems in PSPACE: (Assume C can be decided to by an **NTM M1**).

M = "on input w (in PSPACE):

1. Compute(f(w)) (from property 2, this reduces w to C where f is the polynomial time reduction function).
2. Run M1 on C. (from property 3)
3. If M1 accept, accept. Otherwise reject."

Time Complexity: Step 1 takes polynomial time and Step 2 takes nondeterministic polynomial time. Hence, PSPACE can be decided by an nondeterministic turing machine in polynomial time. PSPACE ⊆ NP is proved.