gradiance

Gradiance Online Accelerated Learning

Zayd

- Home Page
- Assignments Due
- Progress Report
- Handouts
- Tutorials
- Homeworks
- Lab Projects
- Log Out

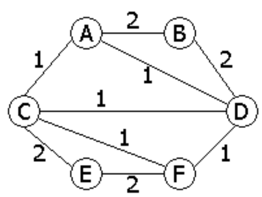
Help

Submission number:	84180
Submission certificate:	EE366514
Submission time:	2014-05-09 02:04:00 PST (GMT - 8:00)

Number of questions:	8
Positive points per question:	3.0
Negative points per question:	1.0
Your score:	20

Questions about reductions and the meaning of certain NP-complete problems based on Section 10.4 of HMU.

1. Find all the minimum-weight Hamilton circuits in the graph below:



Then, identify in the list below the edge that is NOT on any minimum-weight Hamilton circuit.

a) (A,C)
b) (C,E)
c) (B,D)
d) (C,D)

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

Nodes B and E only have two incident edges, so each of those four edges must be in any Hamilton circuit, even though they are the edges of highest weight. Thus, there is only one Hamilton circuit --- the one that goes around the outside of the diagram. If we start at A, for example, this circuit is A,B,D,F,E,C,A.

2. A *k-clique* in a graph G is a set of *k* nodes of G such that there is an edge between every pair of nodes in the set. The problem *k*-CLIQUE is: Given a graph G and a positive integer *k*, does G have a *k*-clique?

We can prove *k*-CLIQUE to be NP-complete by reducing the 3SAT problem to it. Consider the 3-CNF expression:

$E = (x_1' + x_2 + x_3)(x_1' + x_2' + x_3')(x_3 + x_4 + x_5')(x_1 + x_2 + x_4)$

[Note: *a*' denotes the negation NOT(*a*) of variable *a*.] Let G be the graph constructed from this expression as in Theorem 10.18 (p. 460). Then, let H be the *complement* of G, that is, the graph with the same nodes as G and an edge between two nodes if and only if G DOES NOT have an edge between those two nodes. Let us denote the vertices of H by using the corresponding (clause, literal) pair. The node (i,j) corresponds to the j^{th} literal of the i^{th} clause. Which of the following nodes form a maximum-sized clique in H?

a) {(1,1),(2,2),(3,3),(4,2)}

b) (1,2),(2,1),(4,3),(3,2),(4,2)}
c) {(1,2),(2,1),(3,1),(3,2),(4,3)}
d) {(1,1), (2,1), (3,1), (4,3) }

Answer submitted: **d)**

You have answered the question correctly.

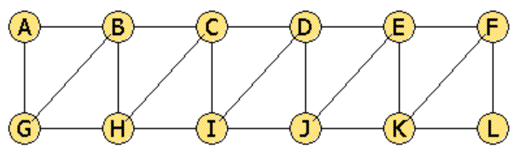
Question Explanation:

The solution is based on establishing a correspondence between satisfiable assignments of the 3-CNF expression E and cliques in the complement H of G. Let *m* be the number of clauses in the E. Then we claim that E is satisfiable if and only if H has an *m*-clique.

To prove the above claim, remember that in Section 10.4.2. (p. 459), it is proved that E is satisfiable if and only if G has an independent set of size *m*. If I is an independent set in G, no two nodes in I are connected by an edge in G. Hence there is an edge between every pair of such nodes in H, forming a clique. Hence G has an independent set of size *m* if and only if H has an *m*-clique. This in turn implies that E is satisfiable if and only if H has an *m*-clique.

---

3. How large can an independent set be in the graph below?



Identify one of the maximal independent sets below.
a) {A,E,H,J,L}
b) {A,C,I,L}
c) {D,F,G,I}
d) {A,D,F,H}

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

An independent set can have at most four nodes. Since Independent-Set is an NP-complete problem, it should not surprise us that even for as simple an instance as this, it is rather hard to reason about how many independent nodes there are. But here is a rough argument for why there can be no more than four.

First, we may as well pick A, because if a maximal independent set included B or G instead, we could replace it by A. If it included neither B nor G, we could add A and get a larger independent set Likewise, we may as well include L. Now, G, B, K, and F are eliminated; they cannot be chosen for a maximal independent set containing A and L.

We can surely pick two of the remaining nodes --- C, D, E, H, I, and J. For example, we can add H and E to the maximal independent set, giving us 4 nodes: {A,E,H,L}. However, picking any of these six nodes eliminates at least two others. Thus, we could never pick three of these six. There is no possibility of a maximal independent set with 5 nodes.

---

4. Consider the descriptions of the following problems:

1. Problem P_1: Given a set S of positive integers $\{s_1, s_2,..., s_n\}$ and an integer C as input, is there a subset T of S such that the integers in T add up to C?

2. Problem P_2: Given a set S of positive integers $\{s_1, s_2,..., s_n\}$ as input, is there a subset T of S such that the integers in T add up to c_0?. Here c_0 is a positive integer constant.

3. Problem P_3: Given an undirected graph G and an integer K as input, is there a clique in G of size K?

4. Problem P_4: Given an undirected graph G as input does G contain a clique of size m? Here m is a positive integer constant.

Now consider some additional propositions about the above problems (These may be TRUE or FALSE):

1. Proposition F_1: There is an algorithm A_1 that solves problem P_1 in O(nC) time.
2. Proposition F_2: There is an algorithm A_2 that solves problem P_2 in O(n) time.
3. Proposition F_3: P_3 is NP-complete.
4. Proposition F_4: There is an algorithm A_3 that solves problem P_4 in $O(m^2n^m)$ time.

Choose a correct statement from the choices below:

a) If F_1 and F_2 are both TRUE, then P_2 is in P but we cannot conclude so about P_1.
b) F_1 must be FALSE. Such an algorithm A_1 cannot exist.
c) If F_3 and F_4 are both TRUE, then P=NP.
d) If F_1 and F_2 are both TRUE, then P_1 is in P but we cannot conclude so about P_2.

Answer submitted: **a)**

You have answered the question correctly.

Question Explanation:

All the propositions are in fact TRUE. Whether something is part of the input or not may make a difference about whether a problem is NP-complete or not or solvable in polynomial time or not. The algorithm A_1 will not count as a polynomial time algorithm for problem P_1, where C is part of the input. But algorithm A_2 is a polynomial-time algorithm for P_2 and A_3 is a polynomial time algorithm for P_4.

How does algorithm A_1 work? Let S = $\{x_1, x_2,..., x_n\}$. Let P(i,s) be TRUE if there is a subset of $\{x_1, x_2,..., x_i\}$ that sums to *s* and FALSE otherwise. Then the recurrence

$P(i,s) = P(i-1,s) \text{ OR } P(i-1,s-x_i)$

can be used to design a simple dynamic programming algorithm that fills up an *n*-by-C sized table with P (i,s) values. Algorithm A_2 works almost in the same way as A_1, except that c_0 is known in advance and hence the size of the table and the time to fill it are both O(n).

Problem P_3 can be shown to be NP-complete by a reduction from 3SAT.

Algorithm A_3 for problem P_4 works by finding all possible subsets of size *m* of nodes of graph G. For each such subset V' of size *m*, check all possible pairs (u,v) of nodes in V' for existence of an edge (u,v).

---

5. The proof that the Independent-Set problem is NP-complete depends on a construction given in Theorem 10.18 (p. 460), which reduces 3SAT to Independent Sets. Apply this construction to the 3SAT instance:

(u+v+w)(-v+-w+x)(-u+-x+y)(x+-y+z)(u+-w+-z)

Note that - denotes negation, e.g., -v stands for the literal NOT v. Also, remember that the construction involves the creation of nodes denoted [*i,j*]. The node [*i,j*] corresponds to the *j*th literal of the *i*th clause. For example, [1,2] corresponds to the occurrence of *v*.

After performing the construction, identify from the list below the one pair of nodes that does NOT have an edge between them.

a) [3,2] and [4,1]
b) [2,1] and [2,2]
c) [2,2] and [4,3]
d) [1,1] and [1,3]

Answer submitted: **c)**

You have answered the question correctly.

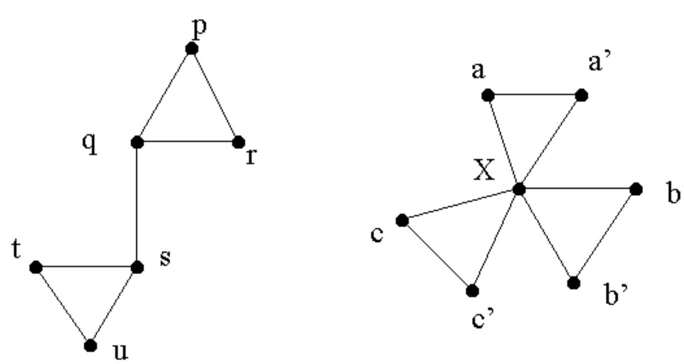
Question Explanation:

There are always edges between nodes [*i*,1], [*i*,2], and [*i*,3] for any *i*. In addition, there are edges between nodes corresponding to a literal and its complement. These edges are:

- For *u*: [1,1]-[3,1] and [3,1]-[5,1].
- For *v*: [1,2]-[2,1].
- For *w*: [1,3]-[2,2] and [1,3]-[5,2].
- For *x*: [2,3]-[3,2] and [3,2]-[4,1].
- For *y*: [3,3]-[4,2].
- For *z*: [4,3]-[5,3].

All other edges are absent.

6. The graph k-coloring problem is defined as follows: Given a graph G and an integer *k*, is G k-colorable?, i.e. can we assign one of *k* colors to each node of G such that no edge has both of its ends colored by the same color. The graph 3-coloring problem is NP-complete and this fact can be proved by a reduction from 3SAT.



As a part of the reduction proof, we assume that there are three colors GREEN, RED and BLUE. For variable *a*, let *a*' denote NOT(a). We associate with each variable *a*, a "gadget" consisting of 3 nodes labeled *a*, *a*' and X. This gadget is shown at the right in the diagram above. X is common for all variables and is labeled blue. If *a* is TRUE (respectively FALSE) in a particular assignment, node *a* is colored green (respectively red) and node *a*' is colored red (respectively green).

With each clause we associate a copy of the gadget at the left in the diagram above, defined by the graph H = (V,E) where V = {*p*,*q*,*r*,*s*,*t*,*u*} and E = {(*p*,*q*), (*q*,*r*), (*r*,*p*), (*s*,*t*), (*t*,*u*), (*u*,*s*), (*q*,*s*)}. Nodes *t*, *u* and *r* from this copy of the gadget are connected to the nodes constituting the literals for the clause, in left to right order.

Consider a clause in the 3-CNF expression: *a* + *b*' + *c*. We must color the above gadget using the colors red, blue and green in such a way that no two adjacent nodes get the same color. In each choice below, you are given an assignment for the variables *a*, *b*, and *c* and a possible assignment of colors to some nodes in the gadget above. Indicate the choice of colors that is valid for the given truth assignment:

a) (a=TRUE, b=FALSE, c=TRUE, t=blue, r=red, p=blue)

b) (a=FALSE, b=TRUE, c=FALSE, t=blue,u=green,p=green)

c) (a=FALSE, b=TRUE, c=FALSE, t=red, r=green, p=blue)

d) (a=TRUE, b=FALSE, c=TRUE, t=red, u=blue, p=red)

Answer submitted: **a)**

You have answered the question correctly.

Question Explanation:

The solution is based on establishing a correspondence between satisfiable assignment for a 3SAT expression and a valid coloring for the corresponding graph.

Claim: The node *p* can be colored green in a valid coloring of the graph if and only if the clause is TRUE under the particular assignment.

To prove the only-if assume that the node *p* is colored green. Let the clause be FALSE. Then *t*, *u*, and *r* are adjacent to red nodes and must be green or blue. If *p* is green, then *r* cannot be green and must be blue. If *r* is blue, *q* being adjacent to *p* and *r*, is red. Since *t* and *u* cannot be red, and must be of different colors, *s* being adjacent to *t* and *u* must be red. Then *s* and *q* are adjacent and both are red, a contradiction. Thus the clause must be TRUE.

To prove the if part, assume that the clause is TRUE. Then consider three cases:

Case 1: One of *a* and *b*' is TRUE.
Then one of *t* and *u* can be colored red and the other green. Then *s*,*r*, *p* and *q* can be colored blue,blue,green and red respectively, without any pair of adjacent vertices being colored the same.

Case 2: Both *a* and *b*' are FALSE.
Then one of *t* and *u* can be colored green and the other blue. Then *s* can be colored red. Since *c* has to be TRUE, it is green. Then *r* can be colored red, *q* blue and *p* green without any adjacent pair of vertices getting the same color.

Case 3: Both *a* and *b*' are TRUE.
Then one of *t* and *u* can be colored red and the other blue. Then *s* can be colored green. If *c* is TRUE, it is green. Then *r* can be colored red, *p* green and *q* blue, without any adjacent pair of vertices getting the same color. Else if *c* is FALSE, it is red. Then *r* can be colored blue, *p* green and *q* red, without any adjacent pair of vertices getting the same color.

---

7. The Independent-Set problem is shown to be NP-complete in the text (Theorem 10.18, Section 10.4.2, p. 460) by a reduction from 3SAT. In the text, the Node-Cover problem is proved to be NP-complete (Theorem 10.20, Section 10.4.3, p. 463) by a reduction from the Independent-Set problem.

We can prove the Node-Cover problem to be NP-complete by a reduction from 3SAT directly. For variable *a*, let *a*' denote NOT(*a*). Given a 3-CNF expression,

$E = (x_1 + x_2 + x_3')(x_1 + x_2' + x_4)(x_3' + x_4' + x_5)(x_1 + x_3 + x_5')$

construct a graph G using the transformation given by Theorem 10.18. Let us denote a node in G by the corresponding (clause, literal) pair (See Figure 10.8 on p. 461).

Which of the following is a node cover for G and of the smallest possible size for a node cover for G?

a) {(1,1), (1,2), (2,2), (3,1), (3,2), (4,1)}
b) {(1,1), (1,3), (2,1), (2,3), (3,1), (3,2), (4,1), (4,2)}
c) { (1,2), (1,3), (2,2), (2,3), (3,2), (3,3), (4,1), (4,2),(4,3)}
d) {(1,2), (1,3), (2,1), (2,3), (3,2), (3,3), (4,1), (4,2)}

Answer submitted: **c)**

Your answer is incorrect.

While this set is a node cover, it is not the smallest possible node cover. The node-cover problem is defined in Section 10.4.3 (p. 463).

Question Explanation:

The solution is based on establishing correspondence between satisfiable assignments for the expression E and node covers for the graph G. Each column of the graph corresponds to a clause (See Figure 10.8 on p. 461).

Let *m* be the number of clauses in E. We claim that E is satisfiable if and only if G has a node cover of size *2m*.

To prove the if part, note that since there are edges between each pair of the three nodes in a column, at least two of the three nodes must be selected in a node cover. Moreover if all the three nodes in a column are included in the node cover, then some column must have fewer than two nodes in the cover to keep the total at *2m*. Hence exactly two nodes from each column must be in the node cover. We choose an assignment T as follows: for each column, we choose the node not chosen in the node cover and assign the corresponding variable to be TRUE. This way all clauses have a literal that is true. Moreover no two such literals chosen will correspond to a variable and its negation. This is because the nodes chosen are not a part of the node cover and cannot have an edges between them whereas a pair of nodes corresponding to a variable and its complement must have an edge between them.

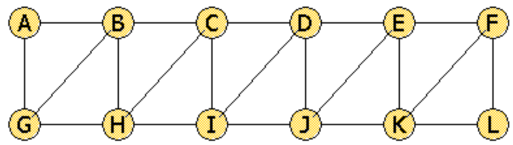
To prove the only-if part, let E be satisfied by a truth assignment T. Since T makes each clause of E true, we identify one literal in each clause which is true under assignment T. Clearly there are no edges between the nodes corresponding to these literals. Hence all these nodes can be excluded from the node cover. We chose the remaining *2m* nodes and form the node cover.

Example:
For the expression E =$(x_1 + x_2 + x_3')(x_1 + x_2' + x_4)(x_3' + x_4' + x_5)(x_1 + x_3 + x_5')$, the set {(1,1), (1,3), (2,2), (2,3), (3,2), (3,3), (4,1), (4,2)} forms a node cover under the satisfiable assignment (x_1 = TRUE, x_2 = TRUE, x_3 = FALSE, x_5 = FALSE).

The correct choice is: **d)**

---

8. What is the size of a minimal node cover for the graph below?



Identify one of the minimal node covers below.

a) {A,C,E,G,H,I,J,K,L}
b) {A,C,E,H,I,L}
c) {B,C,D,F,G,I,K}
d) {B,C,D,F,G,I,J,K}

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

A node cover must have at least eight nodes. Since Node-Cover is an NP-complete problem, it should not surprise us that even for as simple an instance as this, it is rather hard to reason about how many nodes we need to cover all edges. But here is a rough argument for why there must be at least eight.

First, we may as well eliminate A. The triangle among A, B, and G tells us we must pick at least two of these three, and A covers no edge that the other two do not. Likewise, we may as well assume L is not in the minimal node cover. So we have {B,F,G,K} as a subset of our node cover.

This set covers all edges except those that have both ends in {C,D,E,H,I,J}. The triangle among C, H, and I tells us we must pick at least two of those, and the triangle among D, E, and J says we need two of those. That is sufficient; for example, picking C, D, I, and J covers all the remaining edges, giving us a node cover of {B,C,D,F,G,I,J,K}.
