CSE 6140 / CX 4140 Assignment 3
due Oct 16, 2020 at 11:59pm on Canvas

Ting Liao GTID: 903278773
(collaborator: Chin Wang GTID: 903506612 / Shen-Yi Cheng GTID: 903514405)

10/17/2020

1 Dominating set [12 pts]

You're configuring a large network of workstations, which we'll model as an undirected graph G; the nodes of G represent individual workstations and the edges represent direct communication links. The workstations all need access to a common core database, which contains data necessary for basic operating system functions.

You could replicate this database on each workstation; this would make look-ups very fast from any workstation, but you'd have to manage a huge number of copies. Alternately, you could keep a single copy of the database on one workstation and have the remaining workstations issue requests for data over the network G; but this could result in large delays for a workstation that's many hops away from the site of the database.

So you decide to look for the following compromise: You want to maintain a small number of copies, but place them so that any workstation either has a copy of the database or is connected by a direct link to a workstation that has a copy of the database. In graph terminology, such a set of locations is called a *dominating set*.

Thus we phrase the *Dominating Set Problem* as follows. Given the network G, and a number k, is there a way to place k copies of the database at k different nodes so that every node either has a copy of the database or is connected by a direct link to a node that has a copy of the database?

Show that Dominating Set is NP-complete. Follow all steps we have outlined in class for a complete proof. *Hint*: consider the Vertex Cover problem.

Answer:

To proof that Dominating Set is NP-complete, we can use the following step to achieve it.

1. Verify that Dominating Set is in NP problems:
   Dominating Set is a sequence of vertices which generate the Dominating Set in the graph, G(V, E). We can check that all the vertices of the dominating set were in the graph and all the vertices which were not in the set were adjacent to some of the vertices in this set in polynomial time, which is O(V+E). To be more specific, the pseudo code will loop through each vertex and use an if-statement to verify whether the vertex is in the sequence. Then, if the vertex is not in the sequence and the vertex is not adjacent to any vertex in the Dominating Set, it will break the for-loop and make the result to return false. Otherwise, the function will return true.
2. Verify that DS is NP-hard:
   Considering the reduction from the Vertex Cover (VC) problem, which we know VC problem is NP-hard. From the question that we have a network G, which we can consider as a graph G(V, E). Also, k for different nodes and copies can be considered as the size of the Vertex Cover. Then, we can use the above info to transform to a DS problem with a graph G'(V', E').

3. Poly-time Reduction:
   We can construct the graph G' in polynomial time, by adding new edges to the new vertex, which costs O(V+E+new edges+new vertex) = O(V+E) time.

4. From the reduction proof, it shows that DS (Y) is a "Yes" instance if and only iff VC (X) is a "Yes" instance.

   (a) Assume that graph G has a Vertex Cover (VC) of size k. Every edge in G will have one vertex belongs to the VC. So, considering that we have u, v in the graph, if u is in VC, the adjacent vertex v will be covered by some other elements in VC. Also, for all new added vertices, as mentioned above, they will be covered by VC. Therefore, if graph G has a Vertex Cover, graph G' would have the dominating set of the same size k.

   => X(i) = Yes − > Y(f(i)) = Yes

   (b) If we assumed that G' has a Dominating Set of size k, we have to proof that whether G would have a Vertex Cover. Considering that adding new vertices will cause a problem that we have to verify that the vertex in the DS is the original vertex or the new vertex. However, due to that every new vertex is connected to the vertices of the edge (u, v), we would be able to replace the the vertex by u or v. Continuing these process, we can eventually cover all the new vertices while spanning all the edges in graph G'. In other words, new vertices will be dominated by the modified DS and cover all edges in graph G. Therefore, if G' has a Dominating Set of size k, G would have a Vertex Cover with the same size k.

   => Y(f(i)) = Yes − > X(i) = Yes

5. Conclusion:

   From the above proof, the Vertex Cover problem can be reduced to the Dominating Set problem and the Vertex Cover problem is in NP, NP-hard. Therefore, Dominating Set is NP-complete.

2 Frenemies [12 pts]

Assume you are planning a dinner party and going to invite a set of friends. However, among them, there are some pairs of persons who are enemies. You need to create a seating plan and you are wondering if it is possible to arrange this set of n friends of yours around a round table such that none of the two enemies will seat next to each other. Given the set of the n friends and the set of the pairs of enemies, prove that this problem is NP-Complete. Remember to follow the steps from lecture to prove NP-completeness.

You can use the fact that `Hamiltonian Cycle (HC)` is NP-complete.

Answer:

To proof that Frenemies is NP-complete, we can use the following step to achieve it. Let's say there are n friends and k pairs of enemies.

1. Verify that Frenemies is in NP problems:
   The candidate solution of the Frenemies is a sequence of vertices (friends) which generate the Frenemies in the graph, G(n, E) (n = friends, E = edges). We can check that all the vertices and edges of the Frenemies were in the graph and all the vertices didn't have the "enemies edge" with the adjacent

vertices. The above verification can be done in polynomial time, which is O(n+E). To be more specific, the pseudo code will loop through each pair of vertices to verify whether the vertices have an "enemies edge" by checking the set of enemies pairs. Then, if the vertices in any edge are enemies, it will break the for-loop and make the result to return false. Otherwise, the function will return true.

2. Verify that Frenemies is NP-hard:
   Considering the reduction from the Hamiltonian Cycle (HC) problem, which we know HC problem is NP-hard. From the question that we have a round table graph G, which we can consider as a graph G(n, E). Also, some edges can be considered as the pairs of enemies. Then, we can use the above info to transform to a Frenemies problem with a graph G'(n', E').

3. Poly-time Reduction:
   We can construct the graph G' in polynomial time, by adding new check statement to verify whether the vertices are enemies, which costs O(n+E) + O(k) = O(n+E+k) time.

4. From the reduction proof, it shows that Frenemies (Y) is a "Yes" instance if HC (X) is a "Yes" instance.

   (a) Assume that graph G has the HC. Also, every adjacent vertices will not be enemies. So, saying that we transform the nodes of HC into a list u1, u2, u3, ... in the graph, u1 will not be the enemy of u2, but can be the enemy of u3. For all new added vertices, as proved above, they will be added to the graph and its adjacent vertices will not be its enemy. Therefore, if grape G has a HC, graph G' could construct the Frenemies.

   => X(i) = Yes -> Y(f(i)) = Yes

   (b) If we assumed that G' constructs the Frenemies, we have to proof that whether G would have the HC. Considering that the Frenemies creates a graph that each vertex connects with two adjacent vertices (think about the round table that everyone sits around and makes a circle). In other words, each vertex will only have two edges to connect to two other vertices. Therefore, if G' constructs the Frenemies, G would have the HC.

   => Y(f(i)) = Yes -> X(i) = Yes

5. Conclusion:

   From the above proof, the HC problem can be reduced to the Frenemies problem and the HC problem is in NP, NP-hard. Therefore, Frenemies is NP-complete.

3 Let's go hiking [26 pts]

Alex and Baine go hiking together. They bring a bag of items and want to divide them up. For the following scenarios, decide whether the problem can be solved in polynomial time. If yes, give a polynomial-time algorithm; otherwise prove the problem is NP-complete.

- (8 pts) The bag contains n items of two weights: 1lb and 2lb. Alex and Baine want to divide the items evenly so that they carry the same amount of weight.

  Answer:

  This problem can be solved in polynomial time by the following pseudo code.

```
pseudo code:
Assume we have the item list, which is not sorted, e.g. items = [1, 2, 1, 2, 2, 2, 1, ...] with n
items inside

Alex = []
Baine = [] => create two arrays to store the items carried by Alex and Baine

items.sort()  => O(n logn)
count1 = 0
for i = 0 to length(items):  => O(n)
        if items[i] != 1:
                break
        count1 += 1

count2 = 0
for i = 0 to length(items):  => O(n)
        if items[i] == 2:
                break
        count2 += 1

if count1 % 2 != 0:
        return False => means that we can't divide the weight evenly => O(1)
else:
        for i = 0 to length(items):  => O(n)
                if count2 % 2 == 0:
                        if i % 2 == 0: Alex.append(items[i])
                        else: Baine.append(items[i])
                else:
                        Alex will take two more items weight 1.
                        Baine will take one more item weight 2.

sumA  // sum for Alex => O(n)
sumB  // sum for Baine => O(n)

if sumA == sumB: return True
else: return False
```

(a)

Figure 1: Pseudo code for Q3-1

The time complexity is O(n logn) + O(n) + O(n) + O(1) + O(n) + O(n) + O(n) = O(n logn). Also, if we don't need to sort the items, the time complexity could be O(n). Therefore, we can solve the problem in polynomial time.

- (9 pts) The bag contains n items of different weights. Again they want to divide the items evenly.

Answer:

In this case, due to that we don't know the weight of all items, we have to check whether all items can be grouped into a subset and its weight is equal to half of total weight. Here, to make the group, we have to check whether the item can be select or not and this process will cause the time complexity to be $O(2^n)$. If we met the worst case, which means that we need to loop through every item, it will cost $O(n * 2^n)$. Therefore, this problem can't be solved in polynomial time.

Assume we have a set S of n different weights, and we want to divide them into two subsets that they have the same weight.
=> S can be divided to subset A and subset A' = S - A, and the sum of A equals to the sum of A'.

1. Verify the problem is in NP problems:
   We can verify (A, A') in O(n) time by simply looping through each set, summing the elements inside each set, and comparing the two sums. => verify in poly time => is in NP

2. Verify the problem is NP-hard:
   Considering the reduction from the Subset Sum problem. Set Subset Sum as X and our problem is Y, prove instance X can be transformed into instance of Y in polynomial time. The Subset Sum problem has a set S of different numbers and a number t, we want to find a subset T which belongs to S and the sum of T equals to t. First, we let the sum of S be s. Then, set S' = S $\cup$ $\{s - 2t\}$into our problem.

   Proof:

   First: Show X(i) = Yes $->$ Y(f(i)) = Yes

   T is a subset in S and O = S - T, so the sum of T (t) is equal to s - sum of O (o). Also, let's assume that T' = T $\cup$ $\{s - 2t\}$ and the sum of T' is t'.

   => o = s - t
   => o - t = s - 2t
   => t' = t + s - 2t = s - t

   So, the sum of T' is equal to O. What's more, assume that T satisfies the relation above. If T' is not a solution in Y, which means T' needs to add or drop items and its sum may change. That is to say, T needs to add or drop items, too. This will impact the sum of T and eventually fail the original assumption T = O.

   In conclusion, T' must be a solution in Y. Hence, the original set can be divided into two sets with same weights and we can prove that X(i) = Yes $->$ Y(f(i)) = Yes.

   Second: Show Y(f(i)) = Yes $->$ X(i) = Yes

   Let's consider that now we have two sets (A, A') of S' = S $\cup$ $\{s - 2t\}$ with equal sum. The sum of each set will be (s + s - 2t) / 2 = s - t. Also, let's assume that A' = A $\cup$ (s - 2t) and A' containing $\{s - 2t\}$.

=> s - t = A + s - 2t
=> A = t

From above S' = S $\cup$ $\{s - 2t\}$, we know that S' - S = $\{s - 2t\}$, so A is a subset with sum t of S. If A is not a solution in S, A should add or drop items, which will fail the assumption that A = A'. Therefore, we prove the subset problem can be reduced to our problem and Y(f(i)) = Yes $->$ X(i) = Yes.

3. Conclusion:
   From the above proof, the Subset Sum problem can be reduced to our problem and the Subset Sum problem is in NP, NP-hard. Therefore, our problem is NP-complete.

- (9 pts) The bag contains n items of different weights. They want to divide the items such that the weight difference of items they carry is less than 10lbs.

  Answer:

  Similar as the previous question that we have group some items into two subset and check whether the difference of the weight between the two subsets is less than 10. For example, the total weight is W and two subsets should be W/2 + P (max = 5) and W/2 - P. In the worst case, all items will need to be checked to select or not, so it will cost $O(2^n * n)$ time. Therefore, this problem can't be solved in polynomial time.

  Assume we have a set S of n different weights, and we want to divide them into two subsets that they have the same weight.
  => S can be divided to subset A and subset A' = S - A, and the | sum of A - sum of A' | is less than 10.

  1. Verify the problem is in NP problems:
     We can verify (A, A') in O(n) time by simply looping through each set, summing the elements inside each set, and comparing the two sums. => verify in poly time => is in NP

  2. Verify the problem is NP-hard:
     Considering the reduction from the Subset Sum problem. Set Subset Sum as X and our problem is Y, prove instance X can be transformed into instance of Y in polynomial time. The Subset Sum problem has a set S of different numbers and a number t, we want to find a subset T which belongs to S and the sum of T equals to t. First, we let the sum of S be s. Then, set S' = S $\cup$ $\{s - 2t\}$into our problem.

     Proof:

     First: Show X(i) = Yes $->$ Y(f(i)) = Yes

     T is a subset in S and O = S - T, so the sum of T (t) is equal to s - sum of O (o). Also, let's assume that T' = T $\cup$ $\{s - 2t\}$ Besides, $|T - O| <= 10$.

     Assume that T satisfies the relation above. If T' is not a solution in Y, which means T' needs to add or drop items and its sum may change. That is to say, T needs to add or drop items, too. This will impact the sum of T and eventually cause the original assumption $|T - O|$ to be greater

than 10.

So, T' must be a solution in Y. Hence, the original set can be divided into two sets and $|T - O|$ is less than 10 and we can prove that X(i) = Yes $->$ Y(f(i)) = Yes.

Second: Show Y(f(i)) = Yes $->$ X(i) = Yes

Let's consider that now we have two sets (A, A') of S' = S $\cup$ $\{s - 2t\}$ and $|A - A'| <= 10$. The sum of each set will be (s + s - 2t) / 2 = s - t. Also, let's assume that A' = A $\cup$ $s - 2t$ and A' containing $\{s - 2t\}$.

=> s - t = A + s - 2t
=> A = t

From above S' = S $\cup$ $s - 2t$, we know that S' - S = $s - 2t$, so A is a subset with sum t of S. If A is not a solution in S, A should add or drop items, which will fail the assumption that $|A - A'|$ = 10. Therefore, we prove the subset problem can be reduced to our problem and Y(f(i)) = Yes $->$ X(i) = Yes.

3. Conclusion:
   From the above proof, the Subset Sum problem can be reduced to our problem and the Subset Sum problem is in NP, NP-hard. Therefore, our problem is NP-complete.

Hint: Recall Subset Sum problem: given a set X of integers and a target number t, find a subset $Y \subset X$ such that the members of Y add up to exactly t.