Timur Garifullin
Due: 11:59pm Thursday, Apr 25

Homework 10

Solution: Consulting: Artur, Konstantinos, Aaron
Time spent: 24 hours
Sources: GeekForGeeks

Directions for Homework 10: You may assume from class that the following problems (both the search and decision versions) are NP-complete when doing your reductions. We will (eventually) cover the reasons that all these problems are NP-complete in class.

- SAT (f): Given a boolean formula $f(x1 \cdots xn)$ in CNF form (and of ors) with n variables and m clauses, determine if there is an assignment of the n variables that makes $f(x1, x2 \cdots xn) = T$.
- $3-$SAT (f): A special case of SAT where every clause has at most 3 terms.
- Independent Set (G,k): Given graph G and $k \in \mathbb{Z}^+$, determine if G has an independent set (vertices that are not adjacent to each other) of size $\geq k$.
- Vertex Cover (G,k): Given graph G and $k \in \mathbb{Z}^+$, determine if G has a vertex cover (every edge has at least one endpoint in the cover) of size $\leq k$.
- Clique (G,k): Given graph G and $k \in \mathbb{Z}^+$, determine if G has a clique (vertices that are all adjacent to each other) of size $\geq k$.
- Subset Sum (A,T): Given a list of integers A and a target T, determine if some sublist of A adds up to T.
- Knapsack (V,W,C,T): Given a list of n items with weights $W[i]$ and values $V[i]$ a maximum capacity C and a target value T, determine if some subset of of the items have a total weight $\leq C$ and a total value $\geq T$.
- Hamiltonian Path (G): Given graph G, determine if G has a simple path that visits every vertex exactly once.
- Hamiltonian Cycle (G): Given graph G, determine if G has a simple cycle that visits every vertex exactly once.
- Coloring (G,k): Given graph G, determine if G has a proper vertex coloring using at most k colors.

1. (20 points) Short answer:

   Answer each of the below questions. Your answer will be some subset of the numbers $1, 2, 3, 4$. Justification isn't needed if your answer is 100% correct, but will be needed if you want partial credit for a partially correct answer. You will get a question on the final that tests similar logic.

   You may assume that problem SAT is already known to be $NP-hard$, and problem $GraphConnectivity$ is known to be in P. (not necessarily $NP-complete$)

   The answer to each of the questions below is one or more of the following options.

   - 1: $SAT \rightarrow_p U$
   - 2: $U \rightarrow_p SAT$
   - 3: $GraphConnectivity \rightarrow_p U$

- 4: $U \rightarrow_p GraphConnectivity$

(a) (4 points) If U is known to be in NP, which, if any, of the above are automatically true?

> **Solution:** $U \rightarrow_p SAT$
> $GraphConnectivity \rightarrow_p U$

(b) (4 points) If U is known to be $NP-complete$, which, if any, of the above are automatically true?

> **Solution:** $U \rightarrow_p SAT$
> $GraphConnectivity \rightarrow_p U$

(c) (4 points) If U is known to be in P, which, if any, of the above are automatically true?

> **Solution:** $U \rightarrow_p GraphConnectivity$
> $U \rightarrow_p SAT$
> $GraphConnectivity \rightarrow_p U$

(d) (4 points) If you are trying to prove that U is $NP - hard$, which, if any, of the above would prove that?

> **Solution:** $SAT \rightarrow_p U$

(e) (4 points) If you are trying to prove that U is in P, which, if any, of the above would prove that?

> **Solution:** $U \rightarrow_p GraphConnectivity$

2. (10 points) Show that a set C is a vertex cover of a graph if and only if the set $V - C$ is an independent set. Hint: Do a proof by contradiction.

> **Solution:** Claim:
> A set C is a vertex cover of a graph if and only if the set V-C is an independent set.
>
> Proof:
> 1. If C is a vertex cover then V-C is an independent set:
> Assume for contradiction that C is a vertex cover of a graph G, but V-C is not an independent set. This implies that there exists an edge (u, v) in G such that both u and v are in V-C. However, since C is a vertex cover, at least one of u or v must be in C to cover this edge, which contradicts the assumption that both u and v are in V-C. Thus, if C is a vertex cover then V-C is an independent set.
> 2. If V-C is an independent set then C is a vertex cover:
> Assume that V-C is an independent set of a graph G, but C is not vertex cover. This implies that there exists an edge (u, v) in G such that neither u or v is in C. However, since V-C is an independent set, u and v cannot both be in V-C because otherwise they would

form an edge in G that is not covered by C. Therefore, if V-C is an independent set, then C must be a vertex cover.

Combining both directions of the proof, we conclude that a set C is a vertex cover of a graph if and only if the set V-C is an independent set.

3. (20 points) Show that search SAT $\rightarrow_p$ Decision SAT . Describe an algorithm that can **find** a satisfying assignment for a given SAT instance (search), if one exists, using an algorithm that can only determine **whether or not** a SAT instance *has* a satisfying assignment (decision).

   Remember, you may do polynomial work, and call the Decision version of the algorithm polynomially many times to solve the Search version.

   Hint: Start by deciding if x_1 could be T.

Solution:

```
Algorithm Pseudocode:

import SAT _decision

def SAT_search(f, vara):
    asaignment = 1
    current_formula = f

    for x in vara:
        # Test if x = True leads to a satisfiable formula
        test_formula = current_formula with x set to True
        if SAT_decision (teat _formula):
            assignment [x] = True
            # Keep the change
            current_formula = test_ formula
        else:
            assignment [x] = False
            # Change the current formula
            current_formula = current_formula with x set to False
    return asaignment
```

4. (30 points) Generalizations: Show that each of the following problems are NP-complete. Each reduction should be "simple" - each problem below is a generalization of a known NP-hard problem, or solves a known NP-hard problem with a relatively simple transformation (like from Independent Set to Clique).

   The below example has more detail than you *need* to submit in your own responses on homeworks and tests. However, you still need to be clear and correct.

   (a) (EXAMPLE) Weighted Independent Set: Given a graph $G = (V, E)$ with integer weights on the vertices V, and number k, determine if G has an independent set of total weight

$\geq k$.
Prove that this problem is NP-complete.

Solution:

- First we show this is in NP by showing that a solution could be verified in poly time. Say we are given the inputs G, k as well as an independent set with total weight $\geq k$. Our verifier will check that
  - The given vertices have total weight $\geq k$ by summing the weights of the vertices. We can do this in at most $O(n)$ time.
  - That this is a valid independent set, by checking each edge and making sure that both endpoints are never in the set. This will take $O(E)$ time.
  - Accept the solution as valid if both of the above are True

  This verification will check that the given vertices is an independent set and that the total weight is at least k in polynomial time, and is therefore a poly time verification algorithm for this problem. Therefore, the problem is in NP.
- We reduce Independent Set to Weighted Independent Set.

  Our reduction: Given an instance (G, k) of Independent Set, create a weighted version of G where every vertex has weight 1. Call this graph G'. Now, return Weighted Independent Set(G', k).

  PROOF: By construction, since the weight of each vertex is 1, any set of at least k vertices in G has total weight at least k in G'. Similarly, any set of vertices of total weight at least k in G' must correspond to at least k vertices in G. and vice versa.

  Thus an independent set of k vertices exists in G iff an independent set of total weight k exists in G'. ■

(b) (EXAMPLE) $\mathsf{SAT}_{\leq k}$: Given a SAT instance with n variables and m clauses, and an integer $k \leq n$, decide if there is a satisfying assignment in which *at most* k of the n variables are assigned TRUE.
Prove that this problem is NP-complete.

Solution:

- First we show this is in NP. Say we are given the input formula f, the integer k, and a satisfying assignment that sets at most k variables to true. Our verifier will check that
  - At most k of the variables are set to true. We can check this in $O(n)$ time.
  - That this is a satisfying assignment by evaluating the formula. Each clause takes $O(n)$ time to check as there are at most n variables per clause. There are m total clauses, so this takes a total of in $O(mn)$ time.
  - Accept if both of the above are True

This will accept iff there is a valid assignment

- We show that this problem is a generalization of SAT .

  Our reduction: On an input F to SAT , we return $\mathsf{SAT}_{\leq \mathsf{n}}(F)$.

  PROOF: Since the restriction "at most n of the variables are set to True" applies to any assignment, the problem $\mathsf{SAT}_{\leq n}$ is exactly the problem of $\mathsf{SAT}(F)$, and SAT is a special case of $\mathsf{SAT}_{\leq n}$. ■

(c) SetUnion(E,S,b): Given a set of n elements E, and a family of m subsets $S_i \subseteq E$, and "budget" b, determine if there are some b of these subsets $\{S_i\}$ such that their union is E.

Solution: We need to show that $SetUnion(E, S, b)$ is in NP. Assume we are given a set of n elements E, a family of m subsets $S_i \subseteq E$, "budget" b, and b of these subsets Si such that their union is E.

Verifier will check whether all elements in E are present in any of the b S_i, subsets. We can check this in $O(n * b)$ and accept if True.

Next step is that we reduce $VertexCover(G, k)$ to $SetUnion(E, S, b)$. Keep in mind that in Vertex Cover, we need to determine if G = (V, E) has a vertex cover of size less that k. To convert this into a $SetUnion$ problem:

- Elements E correspond to the edges in G.
- Subsets S, for each vertex v_i , in V, we create a subset S_i, which includes all edges e that v_i touches in G.

Thus, if there exists b subsets in $SetUnion$ so that their union covers all elements E, the corresponding b vertices in G touch all edges, forming a vertex cover of size k.

The verifier shows that $SetUnion$ is in NP, and the reduction indicates that $SetUnion$ is at least as difficult as solving Vertex Cover, which is NP-hard, so the $SetUnion$ is NP-hard. Thus, the $SetUnion$ problem is NP-complete.

(d) $\mathsf{Long\ s-t\ Path(G, s, t, k)}$: Given a graph G, two vertices s and t in V, and an integer k determine if G has a simple path (no repeat vertices) that uses k or more edges from s to t.

Solution: Show that $Long\ s - t\ Path$(G, s, t, k) is in NP. Assume we are given a graph G, two vertices s and t in V, an integer k and a path from s to t.

The verifier will check:

- Whether the number of edges in the path is at least k starting from s and traversing the path until we reach t by increasing a counter.
- Check that these vertices are unique, by keeping track of visited vertices. If a

vertex has been visited before so is in the visited vertices list already, return False. Else, return True.

- The verifier takes O(|V|+|E|) time.

Reduce $HamiltonianPath(G)$ to *Long* $s - t$ *Path*(G,s, t, k). Keep in mind that $HamiltonianPath(G)$: Given graph G, determine if G has a simple path that visits every vertex exactly once.
To convert this into a *Long* $s - t$ *Path*(G,s, t, k) problem:

- Graph G remains the same
- For s,t we select any two vertices in G
- For k, |V|-1 is the number of edges in a path that visits every vertex exactly one time.

If there is a simple path from s to t in G using at least |V|-1 edges, then this path must visit each vertex exactly once. This is because a simple path that includes all vertices without repenting any must have exactly |V|- 1 edges. Thus, such a path meets the conditions of the *Long* $s - t$ *Path* problem and satisfies the requirements for $HamiltonianPath$.

Verifier shows that *Long* $s - t$ *Path* is in NP, and the reduction indicates that *Long* $s - t$ *Path* is at least as hard as solving Hamiltonian Path, which is NP-hard, so *Long* $s - t$ *Path* is NP-hard. Thus, the *Long* $s - t$ *Path* problem is NP-complete.

(e) Integer Linear Programming (A,b): "Integer Linear Programming". Given a matrix of integers A (n by m) and a vector of n integers b, determine if there exists a vector of m integers x such that $Ax \leq b$. (A vector is less than another if it is smaller in every entry).

Solution: A verifier could check that the solution is correct, give the notes of the assignment for each variable (the m*1 matrix, x) by:

- multiplying A by X to make sure the solution is $\leq$ b
- making sure each value in x is an integer

Reduction (3SAT -> ILP):

Start by making a condition for each variable in the clauses (a,b, ..etc.):
$a \leq 1$
$-a \leq 0$
$b \leq 1$
...
Then, create a row in the LP matrix for each of the clauses, where a 1 represents the variable, a 0 represents the variable not existing in the clause, and a -1 represents the variable that exists in the clause but is negated. Then, multiply that matrix by a m*1 matrix corresponding to the variables in the clauses.

For example:
(a or b)
(not b)
This can be represented as:
$Ax = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$
Finally, set the b matrix to (-1 + ($\neq$ negated variables in clause)). This ensures that each row will have an assignment that will make the expression true. For example:
(a or b)
(not b)
Will become: $b = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$
Then, put everything together and run the ILP solver on the new A, x, and b. For example:
(a or b or c)
(not a or not b or c)
(b or not c)
Becomes:

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & 1 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \leq \begin{bmatrix} -1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Running ILP on this would return whether or not there is a valid solution for this set of 3SAT clauses.

Therefore, since we showed that ILP is in NP, and the 3SAT reduces to ILP, ILP is NP-complete.

5. (20 points) Partition: Show that the following problem is NP-complete by reducing from Subset Sum .

Given a list of integers A, determine if the integers of A can be partitioned into two disjoint lists A_1, A_2 that have the same sum.

For example, the list $A = [7, 3, 2, 9, 6, 13]$ can be partitioned into $[7, 13]$, $[2, 3, 6, 9]$ each with sum 20. The list $A = [4, 6, 8, 12]$ cannot be partitioned into two equal pieces.

Solution: Show Partition can be verified in polynomial time Have a verifier with the notes of the two sub-lists, then check:

- Union of both lists gives the original

- Both lists add to the same number

Reduction

```
    import Partition
    def SubsetSum(A, t):
        A.append(2t-(sum(A)))
        return Partition(A)
```

This solution works because normally, by partitioning A, the goal is to get the correct sublist that adds up to the target as one sublist, leaving $sum(A) - t$ in the other sublist. To go to the other sublist equal to t as well, add $2t - sum(A)$, so that it will eventually partition to [t] and $[2t - sum(A) + sum(A) - t] = [t]$ and [t], thus giving the correct answer for valid solutions. In cases where SubsetSum has no solution, a proof by contradiction can be used. Assume there is a valid solution to Partition when there is No valid solution to SubsetSum. After adding $2t - sum(A)$ to A, the sum of all elements in the new A is $2t$, so both lists must then add up to t to give a valid partition. But, because no solution to SubsetSum exists, there is no way to create a sublist that does add up to A => contradiction.

Therefore, since we proved that Partition is in NP and that an NP-hard problem reduces to Partition, Partition is NP-complete.

6. (10 points) (BONUS) PartialMST We've studied the minimum spanning tree problem, which has a few known polynomial time algorithms.

In this variant, there are some *important* vertices in the graph, and some *optional* vertices in the graph. You want to return a spanning tree that definitely includes all the important vertices, but can choose whether or not to include the optional vertices.

This variant is NP-hard even in the unweighted partial Spanning Tree case, where we ask:

PartialST(G, I, k): Given graph G and important vertices $T \subseteq V$, determine if G has a subtree T that includes at least the vertices I and uses at most k edges total.

Show that PartialST is NP-complete.

Hint: Reduce from Vertex Cover.

Solution: