Solutions for Assignment #3

CSC363

April 17, 2008

Note: The solutions are discussed in detail here. For some of the questions your answers were not required to be as careful as these. In particular you would get the full mark on in the first and last parts of question 3 even if you did not write some of the finer parts of the solution.

1. The goal of this question is, assuming $P = NP$, given an integer n, find a factor of n, i. e., find $1 < m < n$, such that $m|n$.

   Consider the following language

   $$L = \{\langle n, l, u\rangle : n, l, u \in \mathbb{N}, 2 \leq l \leq u < n, \text{ and } n \text{ has a factor between } l \text{ and } u\}$$

   We first prove that L is in NP. Consider the following verifier for L:

   - Given input $\langle\langle n, l, u\rangle, m\rangle$ where $n, l, u, m \in \mathbb{N}$,

   (a) Check that the input has the correct format and that $2 \leq l \leq u < n$, otherwise Reject.

   (b) Check that $l \leq m \leq u$, if it is not Reject.

   (c) Divide n by m and check that the residue is zero, i. e., $m|n$, if it is not Reject.

   (d) Accept.

   By definition of L this is a correct verifier. To prove that it is polynomial time not that the comparisons can be made in quadratic time in the length of the input, so the first two steps are polynomial time. And that division can be done in quadratic time. So the verifier is polynomial time.

   If $P = NP$, then $L \in P$, that is a polytime decider M exists for L. Consider the following algorithm that finds a factor of n in polynomial time.

   - Given input $n \in \mathbb{N}$,

   (a) Set $l = 2$.

   (b) Set $u = n - 1$.

   (c) Check that $\langle n, l, u\rangle \in L$ using the machine M, if it is not, output PRIME.

(d) While $l \neq u$,

i. Set $m = \lfloor \frac{l+u}{2} \rfloor$.

ii. Use M to check that $\langle n, l, m \rangle \in L$.

iii. If it is, set $u = m$.

iv. If it is not, set $l = m + 1$.

(e) Output l.

The above algorithm is a simple binary search. We first check that n is not prime. Then we start the for loop knowing that n has a factor in the interval $[l, u]$. You can use induction to prove that this remains true through the algorithm. That is at any point in the while loop, n has a factor in the interval $[l, u]$. So when the loop ends $l = u$ should be a factor of n.

To prove that the above algorithm is polytime note that in each iteration of the while loop the length of the interval $[u, l]$ is halved. That is, $(u - l + 1)$ at the $(i + 1)$th iteration of the loop is $\lceil \frac{u-l+1}{2} \rceil$. So the loop is run at most $\lceil \log n \rceil$ times. Assume that the running time of the M is $O(x^k)$ where x is the length of its input. We run M with inputs of length roughly at most $3 \log n$, so the running time of our algorithm will be $O((\log n + 1)(3 \log n)^k) = O((\log n)^{k+1})$ which is polynomial in the length of input, $\log n$.

2. (a) We first show a polytime verifier for $LargeSum$.

- On input $\langle \langle S, t, b \rangle, T \rangle$. Where S is a subset of integers, $t, b \in \mathbb{N}$ and $T \subseteq S$,

i. Check that $T \subseteq S$.

ii. Check that $t > b$,

iii. Sum the number appearing in T, and let their sum be x,

iv. If $t \leq x \leq b$ Accept,

v. Otherwise Reject.

The verifier is correct by the definition of $LargeSum$. To show that it is polytime, note that the first step takes cubic time on a single tape TM by comparing each element of T to all elements of S and marking the elements of S whenever they are found to be equal to an element of T. All the other second steps take quadratic time. So the verifier is polynomial time.

To prove that $LargeSum$ is NP-complete we reduce $SubSetSum$ to it. The idea is to reduce an instance of $SubSetSum$, $\langle S, t \rangle$ to $\langle S, t, t \rangle$, but in the definition of $LargeSum$ we require that the two given numbers are not equal. So we use the following reduction function f.

- On input $\langle S, t \rangle$.

i. Construct the set S' each element of which is two times an element of S.

ii. Output $\langle S', 2t+1, 2t\rangle$.

We should prove $\langle S, t\rangle \in SubSetSum$ if and only if $\langle S', 2t, 2t+1\rangle \in LargeSum$.

If $\langle S, t\rangle \in SubSetSum$, it means that a subset of S, say T, exists that sums up to t. The corresponding elements in S' are just double the elements of T so they sum up to $2t$. So there exists a subset of S' that sums up to $2t$ which is in the interval $[2t, 2t+1]$. This proves that $\langle S', 2t+1, 2t\rangle \in LargeSum$.

If $\langle S', 2t+1, 2t\rangle \in LargeSum$, it means that a subset of S', say T', exists that sums up to $2t$ or $2t+1$. But, the elements of S' are all even so no subset of S' can sum up to $2t+1$, so T' should sum up to $2t$. Now, the elements in S corresponding to these elements in T' are going to sum up to half $2t$ which is t. This proves that $\langle S, t\rangle \in SubSetSum$.

To prove that the reduction is polynomial time note that we can multiply an element is S by adding a zero to its binary expansion so the reduction can be computed in quadratic time.

(b) The idea in this part is to add a number of elements to S such that any subset of S that sums up to something between t and b can be completed to a subset that sums up to t. In order to have a correct reduction we have to make sure that no subset that summed up to something outside the interval $[b, t]$ can be completed with the added elements to a subset that sums up to t. In particular assume that given the number $d = t - b$ we can construct a set $X_d \subset \mathbb{N}$ with these properties,

- The sum of the elements of X_d is d. So, the sum of elements in any subset of X_d is between 0 and d.
- For any number between 0 and d there is at least one subset of X_d that sums up to that number.
- The number of elements of X_d is polynomial in $\log d$ and it can be construct in polynomial time.

If such a construction is known consider the following polytime reduction f from $SubSetSum$ to $LargeSum$.

- On input $\langle S, t, b\rangle$,

i. Compute $X = X_{t-b}$.

ii. Output $\langle S \cup X, t\rangle$, where in union we allow for multiplicities.

Note that the first step can be computed in polynomial time by the third property of the set X_d. Also the size of X is polynomial in $\log(t-b)$ and each element of it is less than $t - b$ so the second step can be completed in time polynomial in the length of the description of S and $\log(t-b)$. So the reduction is polynomial time.

We should prove that $\langle S, t, b\rangle \in LargeSum$ if and only if $\langle S \cup X, t\rangle \in SubSetSum$. Assume that $\langle S, t, b\rangle \in LargeSum$. This means that

a subset $T \subseteq S$ sums up to a number α between b and t. Then by the second property of X there is a subset of X lets call it U that sums up to $t - \alpha$ (because $t - \alpha$ is at most $t - b$ and at least 0.) $X \cup U$ is a subset of $S \cup X$ which sums up to $\alpha + (t - \alpha) = t$, so, $\langle S \cup X, t\rangle \in SubSetSum$.

Now assume that $\langle S \cup X, t\rangle \in SubSetSum$. This means that a subset of $S \cup X$, say T', sums up t. Assume that T' is made of two parts, T which is a subset of S and U which is a subset of X. The sum up the elements of U (call it α) can be at most the sum of all elements of X, which is by the first property $t - b$. So the sum of the elements of T is $t - \alpha$ which is at least $t - (t - b) = b$. So T is a subset of S that sums up to a number between t and b. This means that $\langle S, t, b\rangle \in LargeSum$.

The only thing left now is to construct the set X_d for every d with the above three conditions. We will first construct the set X_d when d is a power of 2, say 2^l for some l. X_{2^l} would be,

$$X_{2^l} = \{1, 1, 2, 4, 8, \ldots, 2^{l-1}\}.$$

It is clear that the sum of the elements of X_{2^l} is 2^l. Furthermore, for any number $\alpha < 2^l$, α can be written as the sum of powers of two (binary representation), so α is the sum the elements of a subset of X_{2^l}, so the first two properties hold. Also they are $l + 1$ elements in X_{2^l} and it can be constructed in time $O(l^2)$, so the last property holds.

To construct X_d for a general d. Write d in binary representation as the sum of powers of 2 as $d = 2^{\beta_1} + 2^{\beta_2} + \cdots + 2^{\beta_k}$. Let,

$$X_d = X_{2^{\beta_1}} \cup X_{2^{\beta_2}} \cup \cdots \cup X_{2^{\beta_k}}.$$

By the first property of X_{2^l} the sum of the elements of d is $2^{\beta_1} + 2^{\beta_2} + \cdots + 2^{\beta_k}$ so the first property holds. Also X_d has size at most $k \log d$ which is $O((\log d)^2)$ and can be constructed in cubic time using the construction of X_{2^l}.

It remains to prove that any number $\alpha < d$ is the sum of the elements of a subset of X_d. Consider the binary expansion of d and α. Because α is less than d when you consider their decimal expansion from left to right they should agree to some digit $i - 1$ and then d should have 1 as the ith most significant bit while α has 0 as the ith most significant bits. In other words if as above

$$d = 2^{\beta_1} + 2^{\beta_2} + \cdots + 2^{\beta_{i-1}} + 2^{\beta_i} + \cdots + 2^{\beta_k},$$

where $\beta_1 > \beta_2 > \cdots > \beta_k$. We should have

$$\alpha = 2^{\beta_1} + 2^{\beta_2} + \cdots + 2^{\beta_{i-1}} + \alpha',$$

where $\alpha' < 2^{\beta_i}$. But this means that by the second property $X_{2^{\beta_i}}$ has a subset, T', that sums up to α'. So, we can take $T \subset X_d$ to be

$$T = X_{2^{\beta_1}} \cup X_{2^{\beta_2}} \cup \cdots \cup X_{2^{\beta_{i-1}}} \cup T',$$

and the sum of the elements of T would be $2^{\beta_1} + 2^{\beta_2} + \cdots + 2^{\beta_{i-1}} + \alpha' = \alpha$. This proves that X_d has the second property.

3. (a) We first show that TSP is in NP. Consider the following verifier,

- On input $\langle\langle G, c, k\rangle, T\rangle$. Where G is a graph on n vertices with m edges.

i. Check that T is a tour of length (number of edges) at most $2m$ in G.

ii. Check that T visits all the vertices of G.

iii. Add the sum of the weights of the edges of T and check that it is at most k.

iv. If all the checks succeed Accept,

v. Otherwise, Reject.

It is clear that all the steps run in polynomial time. Note that the third step can take as long of the length of T but we have already checked that the length of T is not more than $2m$.

To prove that this is a correct verifier we show that the tour of minimum cost can not take an twice edge in the same direction, so it has length at most $2m$. Assume that the minimum cost tour takes an edge in the direction $u \to v$ twice. Assume that this tour has cost x we will construct a tour of cost $x - c(u, w)$ which is less than x. This contradicts with the fact that this is the minimum cost tour. Assume that the minimum cost tour is

$$u, w, v_1, v_2, \ldots, v_l, u, w, v_{l+1}, \ldots, v_{l'}.$$

Consider the tour

$$u, w, v_1, v_2, \ldots, v_l, u, w, v_{l'}, v_{l'-1}, \ldots, v_{l+2}, v_{l+1}.$$

It is easy to verify that this tour takes the same edges but takes the edge u to w one less time. Thus this new tour has cost $x - c(u, w)$ which contradicts with the fact that we began by looking at the minimum cost tour.[1]

To show that TSP is NP-complete, we reduce from Hamiltonian Cycle using the following function,

- On input G,

i. Construct G' which is the complete graph.

[1] This reasoning is one the parts that was not required from your solutions.

ii. Construct a weight function c as follows. Give an edge of G' weight 1 if it is in G and weight $n+1$ if it is not.

iii. Output $\langle G', c, n\rangle$.

It is clear that this is function is polynomial time computable. We should prove that G has a Hamiltonian Cycle if and only if $\langle G', c, n\rangle \in TSP$.

First, assume that G has a Hamiltonian Cycle. By construction, all the corresponding edges in G' have weight 1. So, the corresponding cycle in G' has weight n. This proves that $\langle G', c, n\rangle \in TSP$.

Now, assume that $\langle G', c, n\rangle \in TSP$. This means that G' has a tour visiting all the vertices of total weight n. As all the edges outside G have weight $n+1$ this tour can only take the edges of G. So the tour is made of n edges of G and visits all the vertices of G' at least once which means that it can not visit any vertex twice, so it is a cycle. So G has a Hamiltonian Cycle.

(b) (This part became a bonus)

The optimization version of the problem is not a language so we can not use mapping reductions for this part. Instead we give a polynomial time decider for the NP-complete problem TSP using a procedure that computes the optimal value of the optimization problem.[2] Lets assume that the procedure for computing the optimal value is called M.

- On input $\langle G, c, k\rangle$,

i. call $M(\langle G, c\rangle)$ and let the result be k'.

ii. If $k' \leq k$ Accept,

iii. Otherwise Reject.

(c) Consider the following algorithm for computing the edge set of the tour given a decider M for the decision problem TSP.

- On input $\langle G, c, k\rangle$

i. Use M to check that $\langle G, c, k\rangle \in TSP$. If it is not output NONE.

ii. let $E = \emptyset$.

iii. For all edges e in the edge set of G.

A. construct $c' = c$.

B. Set $c'(e) = k + 1$.

C. If $\langle G, c', k\rangle \in TSP$, set $c = c'$,

D. Otherwise add e to E.

iv. Output E.

If the running time of M is $O(n^d)$ the running time of our algorithm will be $O((n^2+1)n^d + n^3)$. So the algorithm is polynomial time.

[2]This is called a Cook reduction after prof. Cook, or a polynomial Turing reduction

The idea behind the algorithm is to remove the edges of the graph and test if the resulting graph still has a tour of weight at most k. But the input to the TSP problem is always a complete graph, so we set the cost of the edges to $k+1$. Any edge whose cost is more than k can not be part of any minimum cost tour.

To prove the correctness of the algorithm, note that at all times in the loop, we have $\langle G, c, k\rangle \in TSP$. The end of the for loop any edge is either in E or has cost $k+1$, so there is a tour only made of the edges in E of cost at most k. We should prove that the edges of E form a tour of cost at most k in the graph. Assume that T is a tour of cost at most k that is made of some of the edges in E. In particular assume that e is an edge that is not in the tour T. When we considered e in the for loop, the cost of none of the edges in T had changed so changing the cost of e to $k+1$ we should still have had $\langle G, c', k\rangle \in TSP$. So e should not be in E, which is a contradiction. So the edges of T form a tour of cost at most k in the graph.

4. We show that 3SAT can be decided in linear space. Consider the following decider for 3SAT.

   - Given a 3CNF formula ϕ on n variables $x_1, \ldots, x_n$.

   (a) Set $x_1 = \cdots = x_n = false$.

   (b) repeat,

      i. Check if the current assignment satisfies ϕ. If it does Accept. Otherwise,

      ii. If all x_i's are set to $true$ break the loop.

      iii. Find the first i such that $x_i = false$. Set $x_1, \ldots, x_{i-1}$ to $false$ and x_i to $true$.

   (c) Reject.

   The idea of the algorithm is to check all the possible assignments to the variables. We go over all the assignments exactly like increasing a counter. The first step of the loop checks if the current assignment satisfies the formula. The second step checks if we have exhausted all the assignments. And the last step moves to the next assignment. This step is exactly like increasing a binary counter.

   The decider only needs linear space because we only need space to keep the current assignment and to check if it satisfies the formula. Each of these requires a linear amount of tape space.