CSC373H1

Weidong A.
Eric B.
Rohan D.
Felix Z.

Assignment 2

March 31, 2015

CSC373 Assignment 2 Eric Bannatyne
31 March 2015 1000468451

1. (a) Define the decision problem D_0 as follows.

   Input: A string $x \in \Sigma^*$, where $\Sigma = \{0, 1\}$.

   Output: Does $x = 0$?

   To show that $D_0 \leq_p \overline{D_0}$, we wish to construct in polynomial time a function $f : \Sigma^* \to \Sigma^*$ such that x is a yes-instance of D_0 if and only if $f(x)$ is a yes-instance of $\overline{D_0}$ (or equivalently a no-instance of D_0). The reduction function f is simply defined as follows.

   $$f(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases}$$

   This function can trivially be computed in constant time, which means that it is computable in polynomial time. Moreover, if x is a yes-instance of D_0, then $x = 0$, so $f(x) = 1$, which is a no-instance of D_0. Conversely, if x is a no-instance of D_0, then $f(x) = 0$, which is the only yes-instance of D_0. Therefore $D_0 \leq_p \overline{D_0}$.

   (b) The problem of determining whether a binary input string x is equal to the string 0 is clearly decidable in constant time, and in particular $D_0 \in \text{P}$. Since $\text{P} \subseteq \text{NP}$, it follows that $D_0 \in \text{NP}$.

   (c) Let D_1 be NP-complete and suppose $D_1 \leq_p \overline{D_1}$. Then there is some reduction function $f : \Sigma^* \to \Sigma^*$ computable in polynomial time such that x is a yes-instance of D_1 if and only if $f(x)$ is a yes-instance of $\overline{D_1}$. This implies that we also have $\overline{D_1} \leq_p D_1$, by using the exact same reduction function f.

   Since $D_1 \in \text{NP}$, it follows that $\overline{D_1} \in \text{coNP}$. Moreover, a language L is NP if and only if $\overline{L}$ is coNP. Since D_1 is NP-complete, we have $L \leq_p D_1$, and equivalently (by the same reasoning as above), $\overline{L} \leq_p \overline{D_1}$. Therefore $\overline{D_1}$ is coNP-complete.

   Let $A \in \text{NP}$. Then $A \leq_p D_1 \leq_p \overline{D_1}$. Since $\overline{D_1} \in \text{coNP}$ it follows that $A \in \text{coNP}$. Therefore $\text{NP} \subseteq \text{coNP}$.

   Similarly, let $B \in \text{coNP}$. Then $B \leq_p \overline{D_1} \leq_p D_1$, since $\overline{D_1}$ is coNP-complete. Then $B \in \text{NP}$. Hence $\text{coNP} \subseteq \text{NP}$.

   Therefore, combining these two statements, it follows that $\text{NP} = \text{coNP}$, under the assumption that such a problem D_1 exists.

CSC373 Winter 2015 Assignment # 2

Name: Weidong An
Student Number: 1000385095

Question 2
Proof: To prove that GoldDigger is NP-complete, we need to show,

1. GoldDigger $\in NP$
2. GoldDigger is NP-hard.

(1) Show that GoldDigger $\in NP$.
Consider the verifier algorithm. The certificates are all the paths $C = [j_1, ..., j_\ell]$ for $\ell \leq m$.

```
GoldDiggerVerify($h, g, H, G, C$)
    $hardness = 0$
    $gold = 0$
    for $k = 1, ..., \ell$
        if not $1 \leq j_k \leq n$
            return FALSE
        if $k \geq 2$ and not $j_{k-1} - 1 \leq j_k \leq j_{k-1} + 1$
            return FALSE
        $hardness = hardness + H[k, j_k]$
        $gold = gold + G[k, j_k]$
    if $hardness > h$ or $gold < g$
        return FALSE
    return TRUE
```

GoldDiggerVerify(h, g, H, G, C) returns TRUE for some C if and only if there is a desired drill path.
This algorithm runs in worst case polynomial time because the for loop iterates at most m times and it takes at most polynomial time to implement each iteration. Every line not in the for loop runs in polynomial time clearly.
Hence, GoldDigger $\in NP$.

(2) Show that GoldDigger is NP-hard.
Claim: SubsetSum $\leq_p$ GoldDigger.

The definition of SubsetSum decision problem:
Input: A finite set of positive integers S and a positive integer target t.
Output: Is there some subset S' of S whose sum is exactly t?

Consider the reduction function taking the input of SubsetSum to the input of GoldDigger:

```
SubsetSumToGoldDigger($S, t$)
    Let $H$ and $G$ be $[|S| \times 2]$ arrays
    $i = 1$
    for each $k \in S$
        $H[i, 1] = k$
        $G[i, 1] = k$
        $H[i, 2] = 0$
        $G[i, 2] = 0$
        $i = i + 1$
    $h = t$
    $g = t$
    return $(h, g, H, G)$
```

s_1, s_1	0,0
s_2, s_2	0,0
$\vdots$	$\vdots$
s_n, s_n	0,0

Figure 1: On input $S = \{s_1, s_2, ..., s_n\}$, output H and G as above.

Runtime: The reduction function runs in worst case $O(n)$ ($n = |S|$) which is in polynomial time.
Correctness
Let $S = \{s_1, s_2, ..., s_n\}$.
Suppose there is a subset S' of S whose sum is exactly t. Define

$$j_k = \begin{cases} 1, & s_k \in S' \\ 2, & s_k \notin S' \end{cases}$$

for $k \in \{1, 2, ..., n\}$. Then $j_1, j_2, ..., j_n$ is a path such that

- $1 \leq j_k \leq 2$ for $k = 1, 2, \ldots, n$ (each coordinate on the path is valid);
- $j_{k-1} - 1 \leq j_k \leq j_{k-1} + 1$ for $k = 2, \ldots, n$ (each block is underneath the one just above, either directly or diagonally) since there are only two columns in both H and G;
- $H[1, j_1] + H[2, j_2] + \cdots + H[n, j_n] = \sum\limits_{i \in S'} i = t \leq h = t$
- $G[1, j_1] + G[2, j_2] + \cdots + G[n, j_n] = \sum\limits_{i \in S'} i = t \geq g = t$

Hence, $j_1, j_2, ..., j_n$ is a desired drill path for input (h, g, H, G).

In the other direction, suppose there is a drill path $j_1, j_2, ..., j_\ell$ for $\ell \leq n$. Since there are only two columns in both H and G, either $j_k = 1$ or $j_k = 2$ for $k = 1, \ldots, \ell$. Construct S' as follows:

```
$S' = \emptyset$
**for** $i = 1, \ldots, \ell$
    **if** $j_i == 1$
        $S' = S' \cup \{s_i\}$
```

Note that $\sum_{i \in S'} i = G[1, j_1] + G[2, j_2] + \cdots + G[\ell, j_\ell] = H[1, j_1] + H[2, j_2] + \cdots + H[\ell, j_\ell]$.
Also, $t = g \leq G[1, j_1] + G[2, j_2] + \cdots + G[\ell, j_\ell] = H[1, j_1] + H[2, j_2] + \cdots + H[\ell, j_\ell] \leq h = t$.
Then, $\sum_{i \in S'} i = t$.
Then, S' is a subset of S whose sum is exactly t.

Hence, SubsetSum $\leq_p$ GoldDigger.
It is proved that SubsetSum is NP-hard, so GoldDigger is NP-hard.

By (1) and (2), GoldDigger $\in NP$ and GoldDigger is NP-hard.
Hence, GoldDigger is NP-complete. ■

Question 3

Solution. Consider the following algorithm solving GOLDDIGGEROPT, which uses the algorithm for the GOLDDIGGER decision problem. Note that the notation $A[i :]$ is used to denote a matrix consisting of the rows of A from the i^th row downwards (including the i^th row). An explanation of the algorithm follows.

```
GOLDDIGGEROPT($H, G, h$):
1   $g \leftarrow$ greatest value that satisfies GOLDDIGGER($H, G, h, g$).
2   $A \leftarrow n + 1$ by $m$ matrix where the first row is filled with $(0, 0)$ and the rest is filled with NIL.
3   // A is used to keep track of promising subpaths, where each entry is a pair representing the
4   // $h$ and $g$ up to that point.
5   for $i = 1$ to $n$:
6       for $k = 0$ to $m - 1$:
7           // if on a promising subpath
8           if $A_{i-1,k} \neq$ NIL:
9               // check the possible extensions of the subpath
10              for $j = k - 1$ to $k + 1, 0 \leq j \leq m - 1$:
11                  if GOLDDIGGER($H[i - 1 :], G[i - 1 :], h - A_{i-1,j}[0], g - A_{i-1,j}[1]$) = True:
12                      $A_{i,j} \leftarrow (A_{i-1,j}[0] + H_{i-1,j}, A_{i-1,j}[1] + G_{i-1,j})$
13                  if $A_{i,j}[1] = g$:
14                      return $A, i, j$
```

Given A, i, and j, we can reconstruct the solution in a way similar the algorithm in A1, so the pseudocode for the algorithm is omitted here.

First, we find the most amount of gold attainable with the given H, G, and h on line 1. Using binary search with $g_l = 0$ as the lower bound and $g_u = \Sigma_{i=1}^{n} \Sigma_{j=1}^{m} G_{i,j}$ as the upper bound to find the first guess (if GOLDDIGGER($H, G, h, guess$) is False, $guess$ becomes the new upper bound and if it is True, then $guess$ becomes the new lower bound), even if the size of the upper bound is exponential, running binary search on it would take polynomial time. Now given this optimal value for g, we look for the path which leads to it using the GOLDDIGGER decision problem algorithm.

First, we look at the top row of H and G. For each element in this row, say $H_{1,j}$ and $G_{1,j}$, we check whether it is not on the optimal path by running GOLDDIGGER($H[1 :], G[1 :], h - H_{1,j}, g - G_{1,j}$). If this returns false, choosing to dig at the block $(1, j)$ will make it impossible to obtain the optimal amount of gold. Hence, we conclude that it cannot be on the optimal path. If the decision problem algorithm returns True, then the block might be on the optimal path. By querying each block of the first row in turn, we eliminate any blocks that cannot be in the final solution and keep any blocks that are promising. Then, we repeat the same process of using the decision problem to eliminate non-promising subpaths on all of the blocks reachable from the promising blocks from the previous iteration until we find a path with the optimal g and a valid h.

Since finding the optimal g takes polynomial time, and GOLDDIGGER is called at most once for each block in H and G, if GOLDDIGGER runs in polynomial time then GOLDDIGGEROPT runs in $O(P_1 + m * n * P_2)$, where P_1 is the time it takes to find the optimal g and P_2 is the time it takes to run GOLDDIGER. Therefore, if GOLDDIGGER takes polynomial time, then GOLDDIGGEROPT does as well.

By the correctness of the GOLDDIGGER decision problem algorithm, we know that there exists a path with hardness $\leq h$ and gold $= g$, and that for any valid path with hardness h_0 and gold g_0, if this path is a subpath of the optimal path, then GOLDDIGGER($H, G, H - h_0, G - g_0$) must return true. Hence, at any point in the algorithm, one of the promising subpaths must be a subpath of the optimal path, and eventually the algorithm will find the optimal path for the given g. Since by definition g is the highest possible gold attainable with the given h, this algorithm solves the GOLDDIGGEROPT problem. ♦

Question 4

(a). The procedure PriceIsRightGreedy employs a greedy strategy as an attempt to solve the problem. Since the input is sorted in non-increasing order, the next element of the sequence is greedily chosen as the first unseen element in the sequence X that "fits" (so that the collective sum is at most B).

```
PriceIsRightGreedy($B, X = x_1, x_2, \ldots, x_n$):
1  $S = \varnothing$ // subsequence of elements in $X$
2  $sumSoFar = 0$
3  for $i = 1$ to $n$:
4      if $sumSoFar + x_i \leq B$:
5          $S = S \cup \{x_i\}$
6          $sumSoFar = sumSoFar + x_i$
7  return $S$
```

(b). ***Solution.*** We now show that PriceIsRightGreedy has approximation ratio *at most* 2. Specifically, let the sum of S (that is, $\sum_{x \in S} x$) as returned by the procedure above be denoted by $|S|$. Then, we want to show that,

$$\frac{OPT}{|S|} \leq 2,$$

where OPT is the maximal possible subsequence sum satisfying the constraint of the problem. To see this, we consider two exhaustive cases.

Case 1: $x_1 \geq \frac{B}{2}$. Thus, here we know that $\frac{B}{2} \leq x_1 \leq B$, and so it is clear that the algorithm adds x_1 to S. Thus, $|S| \geq \frac{B}{2}$. Of course, we know that $OPT \leq B$, and so,

$$\begin{aligned} \frac{OPT}{|S|} &\leq \frac{B}{|S|} \\ &\leq \frac{B}{\frac{B}{2}} \\ &= 2, \end{aligned}$$

as desired.

Case 2: $x_1 < \frac{B}{2}$. Since the input sequence is non-increasing, we know that every subsequent element is also less than $\frac{B}{2}$. We either have that $|S| \geq \frac{B}{2}$ or $|S| < \frac{B}{2}$. If $|S| \geq \frac{B}{2}$, then we are done by the argument in Case 1. So, suppose that $|S| < \frac{B}{2}$. Each element x_i is less than $\frac{B}{2}$, and thus if it is left out of S, $|S| + x_i < B$ — which shows that every x_i must be chosen by the algorithm. Thus, the optimal algorithm must also choose every element in the input — in other words, OPT is the sum of all elements in the input, and so $OPT = |S|$. Thus, $\frac{OPT}{|S|} = 1 \leq 2$, and again we have the desired result.

Together, these cases encompass all possible scenarios, and thus we are done.

♦