CSCE 629-601 Homework 10
21st November, 2016

Name: Adil Hamid Malla **UIN:** 425008306

Problem 1. Textbook page 427, Exercise 16.2-2.

Input: Given n number of items and W as the maximum weight of the items thief can put in the knapsack. We are taking about 0-1 Knapsack problem here. Also we are given that each ith item weighs w_i and profit or value of that item is p_i.
Output: Get the maximum profit $\sum p_i$ which the thief gets by using the dynamic programming solution.
Solution:
Since the problem is to be solved using the dynamic programming approach. Here we have to solve the problem in $O(nW)$. Now since we know that we have n number of items thief can choose from and also we have to satisfy the criteria that the weight does not exceed the W limits.
Idea and proof correctness: We can check from the weight equal to 1 to W, how much he can choose each items while maximizing the profit. Now for the dynamic programming to be used we have to get the optimal sub structure. So let us take i be the highest-numbered item in an optimal solution S for W pounds and items 1,2,. . . ,n. Then $S' = S - \{i\}$ must be the optimal solution for $W - w_i$ pounds and items 1,2,. . . ,i-1 and the value of the solution S is v_i plus the value of the subproblem solution S'. We can represent the above relationship using the recurrence equation: we define a value $c[i, w]$ to be the value of the solution for the items 1,2,. . . ,i and the maximum weight w. Then our equation becomes:

$$c[i,w] = \left\{ \begin{array}{c} 0 \text{ if i=0 and w=0} \\ c(i-1,w) \text{ if } w_i > w \\ \max\{v_i + c(i-1, w-w_i), c(i-1,w)\} \text{ if } w_i < w \text{ and } i > 0 \end{array} \right\}$$

The last case says that the value of a solution for i items either includes item i, in which case it is v_i plus a subproblem solution for i-1 items and the weight excluding w_i , or does not include item i, in which case it is a subproblem solution for i-1 items and the same weight. That is, if the thief picks item i, he takes v_i value, and he can choose from items 1,. . . ,i-1 up to the weight limit $w - w_i$ and get $c[i-1, w-w_i]$ additional value. On the other hand, if he decides not to take item i, he can choose from items 1,. . . , i-1 up to the weight limit w, and get $c[i-1, w-w_i]$ value. The better of these two choices should be made. The algorithm takes as inputs the maximum weight W , the number of items n, and the two sequences $v =< v_1, v_2, \ldots, v_n >$ and $w =< w_1, w_2, \ldots, w_n >$ It stores the $c[i, j]$ values in a table$c[0 \ldots n, 0 \ldots W]$ whose entries are computed in row-major order. That is, the first row of c is filled in from left to right, then the second row, and so on. At the end of the computation,$c[n, W]$ contains the maximum value the thief can take.
The algorithm for the same will be: We can use the c table to deduce the set of items to take by starting at $c(n, W)$ and tracing where the optimal values came from. If $c(i, W) = c(i-1, w)$, then item i is not part of the solution, and we continue tracing with $c(i-1, w)$. Otherwise item i is part of the solution, and we continue tracing with $c(i-1, w-w_i)$.
Time Complexity: The above algorithm takes $\theta(nW)$ time total. $\theta(nW)$ to fill up the entries of the table we are saving and $\theta(n)$ time to trace the solution.

Problem 2. Textbook page 1060, Exercise 34.1-4.

Here in the above question we have proved that the running time of the 0-1 knapsack problem is $\theta(nW)$, moreover we assumed the both the inputs to be integers and more specifically, positive integer, where n is the number of items and W is the maximum weight that the knapsack can hold. Now to prove the problem to be solvable in polynomial time, we have to make sure that all the instances of the problem are solved in polynomial time, to prove otherwise we just have to prove one example where the algorithm is not polynomial in time. Here, assuming that the algorithm is not solvable in polynomial time. We will prove this by giving an example. This is *not* a polynomial running time for any reasonable representation of the input. In a reasonable representation, all numeric values i.e the weights,the values, etc. will be given in binary (base 2, or perhaps even a higher base). To represent the value W takes log W bits;
thus the running time of O(n*W) is exponential in the size of the input (although polynomial in the magnitude).

Algorithm 1 Knap Sack Problem

```
1: procedure KnapSack Problem(n, v_i, w_i)
2:     Let c[0...n, 0...W] be a new array
3:     for w =0 to W do
4:         c[0, w] = 0
5:     end for
6:     for i =1 to n do
7:         c[i, 0] = 0
8:         for w =1 to W do
9:             if w_i < w then
10:                if v_i + c(i − 1, w − w_i) > c(i − 1, w) then
11:                    c(i, w) = v_i + c(i − 1, w − w_i)
12:                else
13:                    c(i, w) = c(i − 1, w)
14:                end if
15:            else
16:                c(i, w) = c(i − 1, w)
17:            end if
18:        end for
19:    end for
20:    Return c(n, W)
21: end procedure
```

Problem 3. Textbook page 1065, Exercise 34.2-3.

Solution:
Lets consider we have a $HAM - CYCLE = \{< G >$ G is a hamiltonian graph$\}$ and the corresponding $HAM - CYCLE \in P$.
Now we know that for each node of the HAM-CYCLE exactly two edges are incident, which participate int the cycle. Now to get the listing of all the vertices of the hamiltonian cycle in order, in polynomial time can be done using the following algorithm: Lets say we have a function called $CheckHamCycle\,(G)$, when given an undirected graph returns true if G has Hamiltonian cycle and false if it doesnot exist. Since it is given in the question that doing this operation of finding whether a graph has Hamiltonian Cycle is solvable in polynomial time. Thus our function will also run in polynomial time. Now we will use this function to trace all the vertices of the cycle.
Idea: If we run the $CheckHamCycle\,(G)$ and we get false then graph G doesn't have any Hamiltonian Cycle, so no vertices will be outputed. Now If G does have Hamiltonian Cycles then we do the following to find the cycle. For some vertex $v \in V$ and for edge $e = (u, v) \in E$: remove e from the original graph, $G' = G - e$. Run $CheckHamCycle\,(G')$. Remove all the edges e on v until we get a false. If we get a false then the previous edge is necessary in the Hamiltonian Cycle. So we will put this e back into the G, also we will put the v in a list of vertices H, which will be the vertices forming the Hamiltonian cycle. Set the endpoint u of the most recent necessary edge equal to v, v = u. Test all the edges until we return to the first explored vertex or untill all the edges in G have been checked.
The algorithm for the same will be: Now since our $CheckHamCycle\,(G)$ is solvable in polynomial time.
(a)We know that this algorithm will terminate.
It will return an empty set of vertices if G does not have a Hamiltonian Cycle.
Otherwise it will return the vertices in order of a valid Hamiltonian Cycle.
(b)We know the algorithm will return the vertices in the correct order.
After each necessary edge e is found the next edges checked are connected to e.
There must be a connecting edge among those searched because a Hamiltonion cycle is by definition a closed loop.
Once all edges are checked, we return H, which is a list of connected vertices that results in a cycle that visits every vertex once. All unnecessary edges will not contribute to the order of vertices returned because unnecessary edges to the current Ham Cycle will be ignored, even if they are part of another Ham Cycle. They will be removed and $CheckHamCycle\,(G)$ will return true because the edges for the returned Ham Cycle will be present.
At last, we know that the original solution could take a polynomial time, so our method that records the vertices and edges could also take the same complexity, which means that if HAM-CYCLE $\in P$, then the

Algorithm 2 Hamiltonian Cycle Vertices

```
1: procedure HamiltonianCycleVertices(G)
2:     Let H be a new array to store vertices
3:     Choose an starting edge e = (u, v)
4:     while G has unchecked edges do
5:         if CheckHamCycle (G) then
6:             for all edges on v do
7:                 (v, u) → e
8:                 G − e → G′
9:                 if CheckHamCycle (G) then
10:                    G = G′
11:                else
12:                    add v to H
13:                    mark e as checked
14:                    next = u
15:                end if
16:            end for
17:            v= next
18:        end if
19:    end while
20:    Return H
21: end procedure
```

problem of listing the vertices of Hamiltonian cycle in order, is also polynomial-time solvable as we are doing the $O(eP)$. where P is the polynomial time to execute the $CheckHamCycle(G)$. Thus total time taken is polynomial solvable.

Problem 4. Textbook page 1066, Exercise 34.2-10.

Solution:
Since we have to show that given $NP \neq co-NP$, then $P \neq NP$. We will prove the above using the contrapostion. Now our contraposition will be if P = NP then NP = co-NP.
Now by the lemma of P being closed under the complement which make the base for the above contrapostion obvious. Now having assumed that if, we have P = NP, then

1. for every $L \in NP$, we have $L \in P$ and now since all the instances of the P are closed under the complement, $L' \in P$ and therefore $L' \in co-NP$.
2. for every $L \in co-NP$, we have $L' \in P$ and since the languages in P are closed under complement, $L \in P$ and therefore $L \in NP$

Thus it proves that if P = NP then NP = co-NP.