

## CS 375: Theory Assignment #3

Due on April 6, 2016 at 2:20pm

*Professor Lei Yu Section B1*

I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of 0 for the involved assignment for my first offense and that I will receive a grade of F for the course for any additional offense.

**Tim Hung**

## Problem 1

(10 points) Suppose the capacity of the knapsack is 30 and the set of items

```
S = {
    (item1, 5, $50),
    (item2, 20, $140),
    (item3, 10, $60),
    (item4, 10, $80)
}
```

where each element of set S represents (item, weight, profit). Find an optimal solution for the fractional knapsack problem using the greedy algorithm introduced in class. Show both the order in which the items are selected and the optimal solution you find.

Item	Weight	Profit	Value
1	5	\$50	10
2	20	\$140	7
3	10	\$60	6
4	10	\$80	8

Item	Quantity	Total Value	Capacity
1	5	\$50	25
4	10	\$130	15
2	15	\$235	0

## Problem 2

(30 points) Find a longest common subsequence (LCS) between two strings X = APPLE and Y = PLATE using the dynamic programming algorithm discussed in class. Provide your solution steps in a table that includes the solutions for all possible subproblems and directed arrows (diagonal, left, and up arrows) needed to find an LCS in the end.

(20 points) Use the recursive method discussed in class to find an LCS based on the information stored in the table.

(5 points) Note: show both an LCS and the path that leads to the LCS.

	x	A	P	P	L	E
y	0	0	0	0	0	0
P	0	↑0	↖1	↖1	←1	←1
L	0	↑0	↑1	↑1	↖2	←2
A	0	↖1	↑1	↑1	↑2	↑2
T	0	↑1	↑1	↑1	↑2	↑2
E	0	↑1	↑1	↑1	↑2	↖3

The longest common subsequence is "PLE".

## Problem 3

(10 points) Briefly describe how to extend the depth first search (DFS) algorithm to determine whether a directed graph has a cycle (You may give a sketch of pseudo code and highlight the lines that are different from the original DFS algorithm. Comment your pseudo code to make it easy to understand).

DFS(G)

```
    for each u in V(G)
        u.color = WHITE
    for each u in V(G)
        if u.color == WHITE
            DFS-VISIT(G, u)
```

DFS-VISIT(G, u)

```
    u.color = GREY
    for each v in Adj(u)
        if v.color == WHITE
            DFS-VISIT(G, v)
        else if v.color == GREY
            print "There's a cycle woah!!!!"
    u.color = BLACK
```

## Problem 4

(15 points) In graph theory, a connected component of an undirected graph is a subgraph in which any two vertices are reachable to each other (i.e., connected by at least one path), but the subgraph is not connected to any additional vertices in the supergraph. Briefly describe how to extend the breadth first search (BFS) algorithm to determine the number of connected components in an undirected graph (You may give a sketch of pseudo code and highlight the lines that are different from the original BFS algorithm. Comment your pseudo code to make it easy to understand).

```
int components = 0
BFS(G)
    for each u in V(G)
        u.color = WHITE
    for each u in V(G)
        if u.color == WHITE
            components++
            BFS-VISIT(G, u)
BFS(G, u)
    u.color = GREY
    Queue q;
    q.enqueue(u)
    while(q != empty)
        v = q.dequeue
        for each w in Adj(v)
            if(w.color == WHITE)
                w.color = gray
                q.enqueue(w)
        v.color = black
printf("Graph has %d components!", components)
```

## Problem 5

(10 points) Enumerate the nodes in the following graph in (a) BFS order and (b) DFS order, starting from node 1.

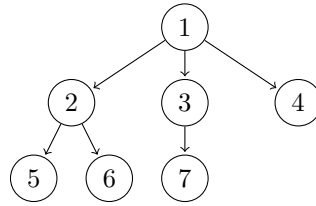


Figure 1: This sure is a graph right here yes it is.

### Solution

a) BFS: 1, 2, 3, 4, 5, 6, 7

b) DFS: 1, 2, 5, 6, 3, 7, 4

## Problem 6

(25 points) For each node  $u$  in an undirected graph  $G(V, E)$ , let  $sDegree(u)$  be the sum of the degrees of the neighbors of  $u$ , that is,  $sDegree(u) = \sum_{(u,v) \in E} Degree(v)$ . Given an adjacency-list implementation of a graph  $G(V, E)$ , provide pseudo code (comment your pseudo code to make it easy to understand) for an  $O(|V| + |E|)$  algorithm that outputs for each node  $u$  its  $sDegree(u)$  (20 points), and briefly analyze the time complexity of your algorithm to justify it is  $O(|V| + |E|)$ . (5 points)

```

degrees = int[V(G).size()]
degrees = 0

for i; i < adjList.size(); i++
    degrees[i] = adjList[i].size()

for i; i < adjList.size(); i++
    int sDeg = 0
    for each u in adjList[i]
        sDeg = degrees[u]
    printf("sDegree(node %d) = %d", i, sDeg)
  
```

## Problem 7

Bonus