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

Optimal value is \$235.

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)          //  Color all vertices WHITE
        u.color = WHITE
    for each u in V(G)          //  DFS-VISIT on each WHITE vertex
        if u.color == WHITE
            DFS-VISIT(G, u)

DFS-VISIT(G, u)
    u.color = GREY              //  Color GREY to mark
    for each v in Adj(u)
        if v.color == WHITE     //  If WHITE, then DFS-VISIT
            DFS-VISIT(G, v)
        else if v.color == GREY                 //  *** About to visit an already marked vertex
            print "There's a cycle woah!!!!"    //  i.e. there's a cycle
    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      //  Tracks number of components
BFS(G)
    for each u in V(G)          //  Color all vertices white
        u.color = WHITE
    for each u in V(G)          //  For all white vertices, BFS-VISIT
        if u.color == WHITE
            components++        //  *** If the vertex is WHITE, it must be part of a new component
            BFS-VISIT(G, u)

BFS(G, u)       //  Color every vertex
    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.

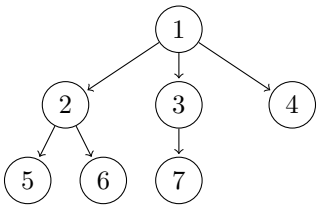

Figure 1: This sure is a graph right here yes it is.

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()]          //  int array to store degree of each vertex
degrees = 0                         //  initialize all to 0

for i; i < adjList.size(); i++      //  for each vertex in the adjList
    degrees[i] = adjList[i].size()  //  degrees[i] is how many vertices connected to current vertex i
                                    //  O(|V|+|E|) Loop through every vertex and edge

for i; i < adjList.size(); i++      //  loop through each vertex in adjList again
    int sDeg = 0
    for each u in adjList[i]        //  for each adjacent vertex, add together their degrees
        sDeg = degrees[u]
                                    //  O(|V|+|E|) Loops through every vertex and edge
    printf("sDegree(node %d) = %d", i, sDeg)    //  output the sDegree of current node
```

Total time complexity is $O(|V| + |E|)$.

Problem 7

(Bonus 30 points) Consider a modification to the activity selection problem in which each activity a_i has, in addition to a start and finish time, a value v_i. The objective is no longer to maximize the number of compatible activities scheduled, but instead to maximize the total value of the compatible activities scheduled. This is called the weighted activity selection problem. Develop a bottom-up dynamic programming solution for this problem. Your solution should have a time complexity in $O(n^2)$, where n is the total number of activities in the input.

(a 10 points) Clearly define the recursive function c[j] which represents the optimal value for the subproblem including activities ending when aj finishes. c[n] represents the optimal value for the original problem.

(b 10 points Demonstrate your solution on the following problem instance. Provide your solution steps in a table that contains optimal values for all subproblems.

	a_1	a_2	a_3	a_4	a_5
Start time	0	1	3	5	7
Finish time	2	4	5	8	11
Value	20	70	30	50	60

(c 10 points) Briefly analyze and justify the time complexity of your solution.