CS430 Introduction to Algorithms
Summer 2024
Assignment 5 Sample Solution Sketches

1. Consider the following pseudo code. I use a hash-map (or dictionary) to store the quantity of each chosen item.

GREEDY-KNAPSACK$\Big(\{s_1, s_2, ..., s_n\}, \{v_1, v_2, ..., v_n\}, B\Big)$

```
1  for i = 1...n :
2      u_i = s_i / v_i
3  Sort each u_i in decreasing order then store them in a (linked) list A
4  total_value = 0
5  Let chosen_items ⟨item, quantity⟩ be an empty hash-map
6  while B > 0 :
7      Let u_j be the head of A
8      if s_j ≤ B :
9          total_value += v_j
10         add j : 1 to chosen_items
11         B -= s_j
12     else :
13         total_value += u_j · B
14         add j : B / s_j to chosen_items
15         B = 0
16 return chosen_items
```

2. One of the possible order is: LA15, LA31, LA22, LA17, LA16, LA32, LA127,LA126,LA169 and LA141.

   To solve this problem, we can start with building a graph $G = (V, E)$ as follows: $V =$ the 10 courses, and if a course v has prerequisite u, then (u, v) is an edge in E. Then run Topological Sort on G, and any output sequence is an acceptable answer.

3. My algorithm is achieved by augmenting **DFS** and **DFS-VISIT**:

   (a) At the beginning of each iteration of the **for** loop in line 4 of **DFS-VISIT**, add "**if** $v.color =$ GREY **then** finish the procedure and **return** True."

   (b) At the end of **DFS**, add "**return** False".

   To use the above algorithm to detect cycles in the given graph G, simply call the augmented **DFS** (G) which also triggers the augmented **DFS-VISIT**. If it returns True, then there exists at least one cycle in G; and if it returns False, then G is acyclic.

   There are two ways that this algorithm terminates:

(a) If G is acyclic, then this algorithm will run a whole **DFS** procedure. In this case, G contains $O(|V|)$ edges since G is simple and acyclic. Thus, the running time is $\Theta(|V| + |V|) = \Theta(|V|)$.

(b) If G contains cycles, then this algorithm will terminate once the first cycle is detected (when the other endpoint is GREY). As an aside, although we haven't proved this in class, there is a theorem in the textbook implicitly shows that it is impossible the we see a black node in the the **for** loop while running **DFS-VISIT** on an undirected graph. In this case, the algorithm will look at at most $|V| + 1$ edges and the running time is $O(|V| + |V|) = O(|V|)$.

4. My algorithm uses **BFS**:

```
isBipartite (G)
1 for each component in G
2     Let s be an arbitrary vertex of this component
3     BFS (G, s)
4 for each u ∈ G.V
5     for each v ∈ G.Adj[u]
6         if v.d mod 2 = u.d mod 2 then return False
7 return True
```

This algorithm calls **BFS** on graph G from arbitrary vertex in each component. From line 4 to line 6, it checks whether there exist two adjacent vertices with both odd or both even d values. If there exists such pair, then the paths from their lowest common ancestor to these vertices together with the edge between these two vertices form an odd cycle.