CS325: Analysis of Algorithms, Fall 2016

Final Exam

- *I don't know policy:* you may write "I don't know" *and nothing else* to answer a question and receive 25 percent of the total points for that problem whereas a *completely* wrong answer will receive zero.
- There are 8 problems in this exam.

Problem 1	
Problem 2	
Problem 3	
Problem 4	
Problem 5	
Problem 6	
Problem 7	
Problem 8	

Problem 1. [**4 pts**] Which of the following job scheduling algorithms always construct an optimal solution? Mark an algorithm **T** if it always constructs an optimal solution, and **F**, otherwise. Provide counterexamples for those that you mark **F**.

(a) Choose the job x that ends last, discard jobs that conflict with x, and recurse.

(b) Choose the job x that starts last, discard all jobs that conflict with x, and recurse.

(c) Choose the job x with shortest duration, discard all jobs that conflict with x, and recurse.

(d) If no jobs conflict, choose them all. Otherwise, *discard* the job with longest duration and recurse.

Solution.

(a) **F**

(b) **T**, this is exactly the same algorithm we saw in class (Why?)

(c) **F**

(d) **F**

Problem 2. [**2 pts**] Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \ldots < c[n]$. (In the U.S., for example, $n = 6$ and the values are 1 (penny), 5 (nickel), 10 (dime), 25 (quarter), 50 (half dollar) and 100 (dollar) cents.) You would like to use the smallest number of coins whenever you give a customer change. In the United States, there is a simple greedy algorithm that always results in the smallest number of coins: subtract the largest coin and recursively give change for the remainder. For example, if the change is 18 cents, the greedy algorithm returns 1 dime, 1 nickel, and 3 pennies.

Show that there is a set of coin values for which the greedy algorithm *does not* always give the smallest number of coins.

Solution. Consider coin values $c[1] = 1$, $c[2] = 4$, and $c[3] = 5$. To give change for eight, the greedy algorithm uses four coins: $5, 1, 1, 1$. But, it is possible to give change for eight with two coins $4, 4$.

Problem 3. [5 pts]

(a) Consider the string $S =$"ddcabcdd", composed of characters, 'a', 'b', 'c', and 'd'. Calculate the frequencies of these four characters, draw the Huffman tree, and deduce Huffman codes for them.

(b) Now consider the string S^{325}, obtained by concatenating 325 copies of S. What are the Huffman codes for 'a', 'b', 'c', and 'd' for this longer string?

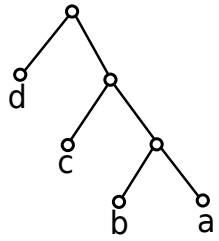
(c) Suppose we have an alphabet of $k > 3$ characters, $c_1, \ldots, c_k$. Also, suppose we have a string T of length ℓ that contains characters $c_1, \ldots, c_k$ with following frequencies:

$$f(c_1) = 1, f(c_2) = 1, f(c_3) = 2, f(c_4) = 4, \ldots, f(c_k) = 2^{k-2}.$$

What is the length of the Huffman code for each c_i, $1 \leq i \leq k$?

Solution.

(a) $f(a) = 1$, $f(b) = 1$, $f(c) = 2$, and $f(d) = 4$. (You can also compute the true frequency by dividing all these numbers by the length of S, which is 8). Here is the Huffman tree:



We deduce the codes from the tree by assigning one to right branches and zero to left branches. Thus, we have: code$(a) = 111$, code$(b) = 110$, code$(c) = 10$, and code$(d) = 0$.

(b) Since we are not changing frequencies by considering multiple copies of the same string, the same codes work here.

(c) Similar to (a), for any $1 < i \leq k$, code$(c_i) = 11\ldots10$, where the number of 1's is $k - i$, and code$(c_1) = 11\ldots11$, with $k - 1$ number of 1's

Problem 4. **[5 pts]** Let $G = (V, E)$ be a simple graph with n vertices. Suppose the weight of every edge of G is one.

(a) What is the weight of a minimum spanning tree of G?

(b) Suppose we change the weight of *two* edges of G to $1/2$. What is the weight of the minimum spanning tree of G?

(c) Suppose we change the weight of *three* edges of G to $1/2$. What is the minimum and maximum possible weights for the the minimum spanning tree of G?

(d) Suppose we change the weight of $k < V$ edges of G to $1/2$. What is the minimum and maximum possible weights for the the minimum spanning tree of G?

Solution.

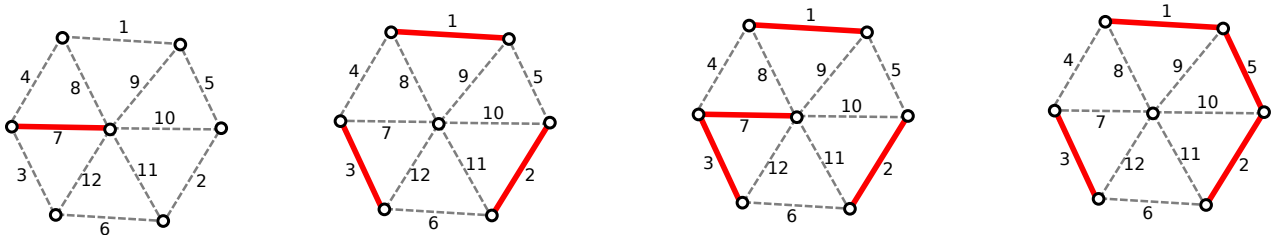
(a) Any spanning tree of G has exactly $V - 1$ edges, since all the weights are one its total weight is $V - 1$.

(b) To answer this question, consider running Kruskal on G, it will first pick the two lighter edges as they cannot form a cycle (the graph is simple), then $V - 3$ other edge each with weight one. Therefore, the total value is $V - 3 + 2(1/2) = V - 2$.

(c) Use the similar approach as (b). Kruskal first selects two light edges. There are two possibilities for the third edge: (i) it forms a cycle with the first two edges or (ii) it does not form a cycle with the first two edges not. In case (i), Kruskal selects two edges of weight $1/2$ and $V - 3$ edges of weight 1, therefore, the total weight is $V - 3 + 2(1/2) = V - 2$. In case (ii), Kruskal selects three edges of weight $1/2$ and $V - 4$ edges of weight 1, therefore, the total weight is $V - 4 + 3(1/2) = V - 5/2$.

(d) Similar to (c), the minimum weight happens when the k light edges do not form any cycle. In this case, MST contains k edges of weight $1/2$ and $n - 1 - k$ edges of weight 1. Therefore, its weight is $k/2 + n - 1 - k = n - k/2 - 1$. The maximum weight happens if the k edges span as few vertices as possible. So, let ℓ be the smallest number such that $\binom{\ell}{2} \geq k$. We can pack all k light edges between ℓ vertices. Consequently, the MST has $\ell - 1$ edges of weight $1/2$ and $n - 1 - (\ell - 1)$ edges of weight 1. Therefore, its weight is $n - \ell/2 - 1/2$.

Problem 5. **[2 pts]** Suppose you can see the edges that are chosen by an MST algorithm in the middle of its running. You want to identify which MST algorithm is running. For each of the following figures, respond Borvka, Kruskal, Jarnik (Prim), or None. (Dashed segments are graph edges and solid segments are MST edges found so far. The numbers are the edge weights.)



Solution. From left to right: Jarnik, Kruskal, Borvka, None.

Problem 6. **[4 pts]** Recall Borvka's minimum spanning tree algorithm from class, which iteratively "add all the safe edges" until it obtains a spanning tree. Here is a high-level pseudo code.

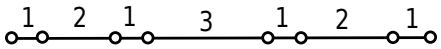
```
1: procedure BORVKA(V, E)
2:     F ← (V, ∅)
3:     while F is not connected do
4:         Add all safe edges to F
5:     return F
```

(a) Construct a graph with eight vertices for which line 4 is executed exactly three times.

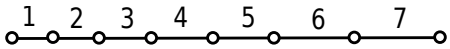
(b) Construct a graph with eight vertices for which line 4 is executed exactly once.

Solution.

(a) Borvka needs three iterations for the following graph.



(b) Borvka needs one iteration for the following graph.



Problem 7. **[6 pts]** Consider the decision version of the Knapsack problem, and the subset sum problem.

Knapsack. The input is composed of two arrays $s[1 \dots n]$, and $v[1 \dots n]$ of positive integers, which specify the sizes and values of n items, respectively, a target value V, and a knapsack size S. The output is YES if it is possible to fit a subset of items of total value V in the knapsack, and it is NO, otherwise.

Subset sum. The input is a set X of n integers and a target integer t. The output is YES if there exists a subset of X, whose elements sum to t, and NO, otherwise.

Now, answer the following questions.

(a) Describe a polynomial time reduction from subset sum to Knapsack.

(b) Show that your reduction is correct; prove that the output of the Knapsack instance is YES if and only if the output of the subset sum instance is YES.

(c) Explain why this reduction implies that Knapsack is NP-hard.

(d) Prove that Knapsack is NP-Complete.

Solution.

(a) Let $X = \{x_1, \dots, x_n\}$, t be the subset sum instance. We compose the Knapsak instance by setting $s[i] = v[i] = x_i$, $S = V = t$.

(b) We need to prove that there is a subset of X whose elements sum to t if and only of there is a subset of items in the Knapsack of total value $V = t$ and total size $S = t$. Suppose, there is a subset X' whose elements sum to t, then the set of corresponding items in the Knapsack problem has total size and value $S = V = t$. On the other hand, suppose, there is a subset of items in the Knapsack of total value V and total size S. The corresponding subset of items must sum to $S = V = t$.

(c) Subset sum is an NP-hard problem: a polynomial time algorithm for subset sum would imply $P = NP$. Our reduction shows that a polynomial time algorithm for Knapsack would imply a polynomial time algorithm for subset sum, thus, it would imply $P = NP$.

(d) Since Knapsack is NP-hard, as shown in (a). We need to show it is in NP to show it is NP-complete. But, it is straight forward to verify if the elements of a given subset X' sum to t in polynomial time.

Problem 8. **[4 pts]** For each of the following statements, respond *True*, *False*, or *Unknown*.

(a) $P = NP$. **U**

(b) If $P \neq NP$, then the satisfiability problem (SAT) cannot be solved in polynomial time. **T**

(c) If a problem is NP-complete then it is NP-hard. **T**

(d) If there is a polynomial time reduction from the satisfiability problem (SAT) to problem X, then X is NP-hard. **T**

(e) If there is a polynomial time reduction from problem X to the satisfiability problem (SAT), then X is NP-hard. **F**

(f) No problem in NP can be solved in polynomial time. **F**

(g) If there is a polynomial time algorithm to solve problem A then A is in P. **T**

(h) If there is a polynomial time algorithm to solve problem A then A is in NP. **T**