CSE 321 - Introduction to Algorithm Design
Homework 05 - Solutions
Deadline: 23:55 December 26th, 2016
PS: Upload your homework to Moodle website. Do **not** bring papers to the room of the TA.

1. Answer questions below.

a) Why do we use greedy algorithms?

Because it cares typically used in situations where the number of optimization possibilities are far too great to feasibly consider within the available timescale.

b) Do greedy algorithms always fail to find globally optimal solution? Why or why not? Provide concrete examples.

Yes, they fail because of sub-optimal solutions.

Here is one of many such instances: For the coin denominations $d_1 = 7$, $d_2 = 5$, $d_3 = 1$ and the amount $n = 10$, the greedy algorithm yields one coin of denomination 7 and three coins of denomination 1. The actual optimal solution is two coins of denomination 5.

2. Prove or disprove the following statements.

a) If e is a minimum-weight edge in a connected weighted graph, it must be among edges of at least one minimum spanning tree of the graph.

a) True. (Otherwise, Kruskal's algorithm would be invalid.)

b) If e is a minimum-weight edge in a connected weighted graph, it must be among edges of each minimum spanning tree of the graph.

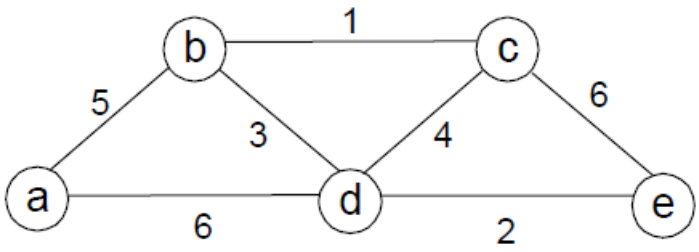
b) False. As a simple counterexample, consider a complete graph with three vertices and the same weight on its three edges

c) If edge weights of a connected weighted graph are all distinct, the graph must have exactly one minimum spanning tree.

c) True (Problem 10 in Exercises 9.1).

d) If edge weights of a connected weighted graph are not all distinct, the graph must have more than one minimum spanning tree.

d) False (see, for example, the graph below).



3. *Knapsack coding and runtime analyze.*

4. *TSP coding and runtime analyze.*

1 https://en.wikipedia.org/wiki/Knapsack_problem
2 https://en.wikipedia.org/wiki/Travelling_salesman_problem
3 https://en.wikipedia.org/wiki/Map_coloring

5. *Map coloring problem coding and runtime analyze.*

6. Consider the problem of scheduling *n* jobs of known durations $t_1, ..., t_n$ for execution by a single processor. The jobs can be executed in any order, one job at a time. You want to find a schedule that minimizes the total time spent by all the jobs in the system. (The time spent by one job in the system is the sum of the time spent by this job in waiting plus the time spent on its execution.)

Design a greedy algorithm for this problem. Does the greedy algorithm always yield an optimal solution? Prove or disprove.

a. Sort the jobs in nondecreasing order of their execution times and execute them in that order.

b. Yes, this greedy algorithm always yields an optimal solution.
Indeed, for any ordering (i.e., permutation) of the jobs $i_1, i_2, ..., i_n$, the total time in the system is given by the formula

$$t_{i_1} + (t_{i_1} + t_{i_2}) + ... + (t_{i_1} + t_{i_2} + ... + t_{i_n}) = nt_{i_1} + (n-1)t_{i_2} + ... + t_{i_n}.$$

Thus, we have a sum of numbers n, $n-1$,...,1 multiplied by "weights" t_1, t_2, ...t_n assigned to the numbers in some order. To minimize such a sum, we have to assign smaller t's to larger numbers. In other words, the jobs should be executed in nondecreasing order of their execution times.

Here is a more formal proof of this fact. We will show that if jobs are executed in some order $i_1, i_2, ..., i_n$, in which $t_{i_k} > t_{i_{k+1}}$ for some k, then the total time in the system for such an ordering can be decreased. (Hence, no such ordering can be an optimal solution.) Let us consider the other job ordering, which is obtained by swapping the jobs k and $k+1$. Obviously, the time in the systems will remain the same for all but these two jobs. Therefore, the difference between the total time in the system for the new ordering and the one before the swap will be

$$\begin{aligned} & [(\sum_{j=1}^{k-1} t_{i_j} + t_{i_{k+1}}) + (\sum_{j=1}^{k-1} t_{i_j} + t_{i_{k+1}} + t_{i_k})] - [(\sum_{j=1}^{k-1} t_{i_j} + t_{i_k}) + (\sum_{j=1}^{k-1} t_{i_j} + t_{i_k} + t_{i_{k+1}})] \\ = \quad & t_{i_{k+1}} - t_{i_k} < 0. \end{aligned}$$

7. Design a greedy algorithm for the assignment problem (see Section 3.4). Does your greedy algorithm always yield an optimal solution? Prove or disprove.

a. The all-matrix version: Repeat the following operation n times. Select the smallest element in the unmarked rows and columns of the cost matrix and then mark its row and column.

The row-by-row version: Starting with the first row and ending with the last row of the cost matrix, select the smallest element in that row which is not in a previously marked column. After such an element is selected, mark its column to prevent selecting another element from the same column.

b. Neither of the versions always yields an optimal solution. Here is a simple counterexample:

$$C = \begin{bmatrix} 1 & 2 \\ 2 & 100 \end{bmatrix}$$

8. Write a pseudocode of the greedy algorithm for the change-making problem, with an amount n and coin denominations $d_1 > d_2 > \ldots > d_m$ as its input. What is the time efficiency class of your algorithm?

Algorithm *Change*$(n, D[1..m])$
//Implements the greedy algorithm for the change-making problem
//Input: A nonnegative integer amount n and
// a decreasing array of coin denominations D
//Output: Array $C[1..m]$ of the number of coins of each denomination
// in the change or the "no solution" message
for $i \leftarrow 1$ **to** m **do**
  $C[i] \leftarrow \lfloor n/D[i] \rfloor$
  $n \leftarrow n \bmod D[i]$
if $n = 0$ **return** C
else return "no solution"

The algorithm's time efficiency is in $\Theta(m)$. (We assume that integer divisions take a constant time no matter how big dividends are.) Note also that if we stop the algorithm as soon as the remaining amount becomes 0, the time efficiency will be in $O(m)$.

9. Answer questions below.

a) How can we use Prim's algorithm to find a spanning tree of a connected graph with no weights on its edges?

a) The simplest and most logical solution is to assign all the edge weights to 1.

b) Is it a good algorithm for this problem? Discuss.

b) Applying a depth-first search (or breadth-first search) traversal to get a depth-first search tree (or a breadth-first search tree), is conceptually simpler and for sparse graphs represented by their adjacency lists faster.

10. Watch Travelling Salesman (2012), directed by Timothy Lanzone, movie and write an essay. (It should be minimum half page A-4 paper size or maximum one page A-4 paper size. Also you are expected to use 1,5 text-space and Times New Roman font family in your essay.)