CSE 321 – Introduction to Algorithm Design

Homework 05

Aikeboer Aizezi
131044086

1. a) We use greedy Algorithms, because they are easy to invent, easy to implement and most of the time quite efficient. Especially, solutions to smaller instances of the problems can be straightforward and easy to understand.

b) Greedy algorithms mostly (not always) fails to find the globally optimal solutions, because they usually do not operate exhaustively on all the data. they can make commitments to certain choices too early which prevent them from finding the best overall solution later. For example, all known greedy coloring algothms for the graph coloring problem and all other NP-complete problems, do not consistently find optimum solutions.

2. a) True. Because the tree made by adding the given edge and removing some other edge from the cycle can't have hiegher weight (becrause the new edge has minimum weight).

b) False; let's take the graph K_n where all edge weights are the same. Any spanning tree is a minimum spanning tree.

c) True, at least one spanning tree exists, e.g., the one obtained by a depth-first search traversal of the graph. And the number of spanning trees must be finite because any such tree comprises a subset of edges of the finite set of edges of the given graph.

d) False: consider the following graph, take a tree on n vertices, give all these edges weight 1, then add all non-edges, give these new edges weight 2. Clearly there's only one MST, but there's $\binom{n}{2}-n+1$ edges with the same weight.

3. Greedy Algorithm for the 0/1 napsack problem:

1. compute value/weight ratio $\frac{v_i}{w_i}$ for all items.
2. Sort the items in non-increasing order of the ratio $\frac{v_i}{w_i}$.
3. Repeat until no item is left in sorted list using following steps:
   a) If current Item fits, use it.
   b) otherwise, skip this item and proceed to next item.

best case: $O(T) = O(n)$

Average case: $O(T) = O(n\log n)$

worst case: $O(T) = O(n^2)$

4. Greedy Algorithm for Traveling Salesman Problem:

1. Sort the edges in increasing order of their weights (Ties can be broken arbitrarily). Initialize the set of ~~tour~~ tour edges to be constructed to the empty set.
2. Repeat this step n tims, where n is the number of cities in instance being solved. Add the next edge on the sorted edge list to the set of tour edges, provided this addition does not create a vertex of degree 3 or a cycle of length less than n. otherwise skip the edge.
3. Return the set of tour edges.

best case : $O(T) = O(n\log n)$, Average Case : $O(T) = O(n^2)$

5. Greedy Algorithm for Map-coloring Problem:

1. Color first vertex with first color.

2. Do following for remaing $v-1$ vertices.

a) Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

best case : $O(T) = O(n\log n)$ average case: $O(T) = O(n^2)$.

worst case : $O(T) = O(V^2 + E)$

6. a) Sort the Jobs in non decreasing 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 of the Jobs $i_1, i_2, \ldots i_n$, the total time in the system is given by the formula $t_{i_1} + (t_{i_1} + t_{i_2}) + \ldots + (t_{i_1} + t_{i_2} + \ldots + t_{i_n}) = nt_{i_1} + (n-1)t_{i_2} + \ldots + t_{in}$.

Thus, we have a sum of numbers $n, n-1, \ldots, 1$ multiplied by "weights" $t_1, t_2, \ldots t_n$ 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 non decreasing order of their execution time.

Here is a formal proof of this fact. We will show that if jobs are executed in some order $i_1, i_2, \dots i_n$. in which $t_{ik} > t_{ik+1}$ 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, the time in the system will remain the same for all but these two jobs. There-for, the difference between the total time in the sytem for the new ordering and the one before the swap will be

$$[(\sum_{j=1}^{k-1} t_{ij} + t_{ik+1}) + (\sum_{j=1}^{k-1} t_{ij} + t_{ik+1} + t_{ik})] - [(\sum_{j=1}^{k-1} t_{ij} + t_{ik}) + (\sum_{j=1}^{k-1} t_{ij} + t_{ik} + t_{ik+1})]$$

$$= t_{ik+1} - t_{ik} < 0.$$

7. 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 ~~matrix~~ smallest element in that row which is not in a previously marked column. After such an element is selected, mark it's column to prevent selecting another element from the same column.

b) Neither of the versions always yields an optimal solution. Here is a simple counter example: $C = \begin{bmatrix} 5 & 10 \\ 10 & 75 \end{bmatrix}$.

8. Algorithm Change $(n, D[1 \dots m])$

// Implements the greedy algorithm for the change-making problem.

// Input: A nonnegative integer amount ~~of~~ n and a decreasing array of coin denominations D.

// Output: Array $C[1 \dots m]$ 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 divisions 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. a) The simplest and most logical solution is to assign all the edges weights to 1.

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.