Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 6 - Greedy and Dynamic Programming

January 25, 2024

1. In this question, you are asked to execute Dijkstra's Shortest Path Algorithm starting from node 0 in Figure 1. Show the steps of the algorithm and the value of $d(v)$ computed for each node v added to the set S of explored nodes. Also show the shortest path P_u which is computed for each such new node u throughout the execution of the algorithm.

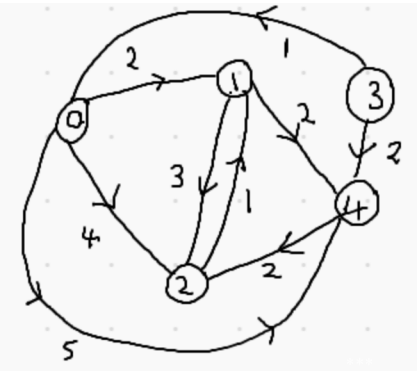

Figure 1: A directed graph with edge lengths indicated over the edges.

2. Consider the *fractional knapsack problem*, in which there is a set of n *infinitely divisible* items with values v_i, for $i = 1, \ldots, n$ and weights w_i, for $i = 1, \ldots, n$, and there is a total weight constraint W. The goal is to find fractions $(x_1, \ldots, x_n)$ of each item, with $0 \leq x_i \leq 1$ such that $\sum_{i=1}^{n} x_i \cdot v_i$ is maximised, subject to the total weight constraint $\sum_{i=1}^{n} x_i \cdot w_i \leq W$.

   A greedy algorithm for the fractional knapsack problem in each step chooses an item i based on some criterion and adds the largest fraction x_i of item i that is possible to fit in the knapsack, without violating the weight constraint, given the items already added to the knapsack in previous steps. The algorithm terminates when the total weight of items included in the knapsack exactly meets the weight constraint, or when all of the items have entirely been added (i.e., $x_i = 1$ for all i) in the knapsack.

   Consider the following three criteria for selecting the item to be added to the knapsack:

(a) Add the item with the largest value v_i among those not already considered (Criterion a).

(b) Add the item with the smallest weight w_i among those not already considered (Criterion b).

(c) Add the item with the largest ratio v_i/w_i among those not already considered (Criterion c).

The first objective is to prove that:

(a) the greedy algorithms with Criterion a or Criterion b are not optimal.

(b) the greedy algorithm with Criterion c is optimal.

Consider the greedy algorithm with Criterion c for the (0/1)-Knapsack problem that we saw in the lectures, which either adds a job entirely to the knapsack or not at all. Does it solve the 0/1-Knapsack problem optimally? Provide either a correctness proof or a counterexample to support your claim.

3. In the UK, coins have demoninations 1p, 2p, 5p, 10p, 20p, 50p, £1 and £2. A frequently-executed task in the retail sector involves taking an input value (say 88p) and calculating a collection of coins (which may include duplicates) which will sum to that value. We assume that we have an unlimited supply of coins of each value. Formally, the *coin changing problem* is the following:

   Input: An input value $v \in \mathbb{N}_0$, and a sequence of coin values $c_0, c_1, \ldots c_k \in \mathbb{N}_0$.

   Output: A *multiset* S of coins with values that sum to v, whose size is the minimum possible for v in this system. The solution will be represented as a list S of length k, with $S[i]$ being the number of coins of value c_i, for each $i \in [0, k]$.

   We may assume that a solution is always possible (e.g., by assuming that $c_0 = 1$).

   Design a dynamic programming-based algorithm which solves the coin changing problem. Run your algorithm on the following input with $k = 2$: $v = 18$, $c_0 = 1$, $c_1 = 5$, and $c_2 = 7$.

   Instead of the dynamic programming algorithm, perhaps a simple greedy algorithm could work: as the greedy criterion, always choose the coin of maximum value among those whose value is smaller than the remaining value. What is the outcome of the algorithm on the example above?

4. A *contiguous subsequence* of length k of a sequence S is a subsequence which consists of k consecutive elements of S. For instance, if S is $1, 2, 3, -11, 10, 6, -10, 11, -5$, then $3, -11, 10$ is a contiguous subsequence of S of length 3. Give an algorithm based on dynamic programming that, given a sequence S of n numbers as input, runs in linear time and outputs the contiguous subsequence of S of maximum sum. Assume that a subsequence of length 0 has sum 0. For the example above, the answer of the algorithm would be $10, 6, -10, 11$ with a sum of 17.