Greedy approaches - Exercices

Advanced Algorithms

Master DSC/MLDM/CPS2

1 Avoiding the lack of petrol

Suppose you are driving from Paris to Lisboa and that your car can travel n kilometers with full gas. There are m gas stations on the way and $(d_1, \ldots, d_m)$ is the list of locations (in kilometers) of the stations from Paris ($d_1 < \ldots < d_m$). We assume that the distance between 2 stations is at most n kilometers. Give a greedy algorithm to determine at which gas station you should stop in order to minimize the number of stops.

- Show that the greedy choice property leads to an optimal solution.
- Show the optimal substructure property.
- Give the algorithm, what is the complexity?

2 Coin changing

Let C be an amount of money to give back to one user. Consider the coins 50, 20, 10, 5 and 1. Give a greedy algorithm using the minimum number of coins to obtain C.

1. Give and prove the greedy choice property.
2. Show the property of optimal substructure.
3. Give the algorithm, what is the complexity?
4. Does this algorithm works with coins of the form $c^0, c^1, c^2, \ldots, c^k$ with $c > 1$ and $k \geq 1$?
5. Give a case for which the greedy algorithm does not found the optimal solution.
6. When the set of coins can take any values, we do not know if the greedy algorithm can find the optimal solution in the general case. A solution is to use a dynamic programming approach: $S = \{a_1, \ldots, a_m\}$ a set of coins and let $z(C, i)$ the minimal number of coins chosen among the first i coins of S to obtain C. We have

   $$z(C,i) = min \begin{cases} z(C, i-1) \text{ if coin i is not used} \\ z(C - a_i, i) + 1 \text{ if coin i is used at least once} \end{cases}$$

   What is the complexity of the algorithm.

3 Exercise: Greedy approaches and the Knapsack Problem

Given n objects: $o_1, \ldots, o_n$, such that each object has a weight w_i and a value v_i, and a knapsack of limited capacity W, the goal is to fill the knapsack so as to maximise the sum of the values of the objects in it. This formulation is known as the 0/1-knapsack problem:

1. Sort the objects according to their values in decreasing order. We consider the objects iteratively in this list and we take the first maximal set of objects that do exceed the limit W. Show that this procedure is not optimal with a counter example.

2. Sort the objects according to their weight in increasing order. Then we proceed like in the previous approach. Show that this procedure is not optimal with a counter example.

3. We now sort the objects according to the value/weight ratio ($\frac{v_i}{w_i}$) and we proceed similarly as in the previous method.

4. We consider now a fractional version of the knapsack problem. We allow the objects to be broken in smaller pieces: for any fraction x_i ($0 \leq x_i \leq 1$) of the object i, this fraction contributes $x_i w_i$ to the total weight in the knapsack and $x_i v_i$ to the value of the load. The problem can stated as: find $x_1, \ldots, x_n$ that maximise $\sum_{i=1}^{n} x_i v_i$ subject to the constraint $\sum_{i=1}^{n} x_i w_i \leq W$. In fact the last constraint can be stated as $\sum_{i=1}^{n} x_i w_i = W$ since we can always add a fraction of one of the remaining objects and increase the load. Show that the previous strategy (number 3) is optimal for this version of the problem. What is the complexity of the approach?

5. We now come back to the original 0/1-knapsack problem. Since the greedy strategy does not work, try to design a dynamic programming version of the problem.
   Write the algorithm in pseudo-code and give the complexities in time and space. What do you think of the running time complexity (argue)?