Algorithm Design - Homework 2

Academic year 2016/2017

Instructors: Prof. Luca Becchetti, Prof. Francesco Pasquale

December 23, 2016

Due date: January 8th, 2017, 11.59pm

Make sure that the solutions are typewritten or clear to read. A complete answer consists of a clear description of an algorithm (an English description is fine), followed by an *analysis of its running time and a proof that it works correctly.*

Hand in your solutions and keep a copy for yourself. Solutions will be posted or presented after due date. In the final exam, you will be asked to explain your solutions and/or to go over your mistakes.

Collaboration policy. You may discuss the homework problems with other members of the class, but you must write up the assignment alone, in isolation. Also, you must understand well your solutions and be able to discuss your choices and their motivations in detail with the instructor. Finally, you should cite any sources you use in working on a homework problem.

Late policy: Every homework must be returned by its due date. Homeworks that are late will lose 10% of the grade if they are up to 1 day (24h) late, 20% if they are 2 days late, 30% if they are 3 days late, and they will receive no credit if they are late by more than 3 days.

Validity: your homework(s) can be used only once. In more detail, once you have discussed the homework, you have to decide whether to accept the proposed vote or not. The moment you sit in a written exam after discussing your homework(s), the vote resulting from the discussion is lost forever.

Please refer to course's Web page for detailed information about above aspects.

Exercise 1. Let $G = (V, E)$ be an undirected, weighted and connected graph. We consider the `Minimum Spanning Tree` problem:

- Problem `Minimum Spanning Tree`
  INPUT: Graph $G = (V, E)$ with weights $w : E \to \mathbb{N}$ on the edges
  OUTPUT: A spanning tree $T \subseteq E$
  GOAL: minimize $w(T)$.[1]

We define the following neighbourhood relationship between spanning trees (i.e., solutions) of the problem above: given two spanning trees T_1 and T_2 of G, we say that T_1 and T_2 are *adjacent* if it is possible to obtain T_2 from T_1 (or viceversa) by only moving one edge. Formally, T_1 and T_2 are adjacent if an edge $e_1 \in T_1$ and an edge $e_2 \in T_2$ exist, such that

$$T_1 \setminus \{e_1\} = T_2 \setminus \{e_2\}\,. \tag{1}$$

[1] $w(T)$ is defined in the obvious way: if T is a spanning tree of G, $w(T) = \sum_{e \in T} w(e)$.

You are asked to address the following points:

1) Let $G = (V, E)$ be a connected graph and let $\hat{V}$ the set of all spanning trees in G. Further, denote by $\hat{E}$ the set of spanning tree pairs that are adjacent according to (1). Show that the graph $\hat{G} = (\hat{V}, \hat{E})$ is connected.

2) Prove, using Lemma 1 below, that if all edge weights are distinct, there is a unique minimum spanning tree (relatively simple proof by contradiction).

3) Prove that, for every T the following holds when all edge weights are distinct: Unless T is a minimum spanning tree, there is at least one adjacent (according to the definition above) spanning tree $\hat{T}$, such that $w(\hat{T}) < w(T)$.
Hint: consider a non-optimal spanning tree T and use the fact that the *unique* minimum spanning tree is just a set of $n - 1$ edges, like T.

4) Assuming the result of point 3) is true, define and give the pseudocode of the obvious local search heuristic corresponding to the neighbourhood relation defined by (1) and prove that, *if all edge weights are distinct*, the local search heuristic returns an optimal solution. Discuss the running time of the local search heuristic. Is it (always) polynomial or not? Prove in both cases.

Lemma 1 ([1], Chapter 4.5, (4.17) and (4.20))**.** *Assume all edge costs are distinct. Let S be any subset of the vertices that is neither empty nor equal to V. Let $e = (u, v)$ be the minimum cost edge with one end in S and the other in $V - S$. Then, every minimum spanning tree contains e. Furthermore, for any cycle C in the graph, the edge e of maximum weight in C does not belong to any minimum spanning tree.*

Exercise 2. We have k trucks $T_1, \ldots, T_k$, that need to be loaded with a set of n boxes, $B_1, \ldots, B_n$, the i-th box having weight w_i. The goal is to allocate boxes to the trucks, so that the overall weights allocated to the trucks are as close as possible.[2] More formally, if W_i denotes the total weight on T_i $(i = 1, \ldots, k)$, the problem can be formulated as follows:

- Problem `Truck loading`
  INPUT: n boxes $B_1, \ldots, B_n$ with weights $w_1, \ldots, w_n$, k trucks $T_1, \ldots, T_k$
  OUTPUT: an allocation of boxes to trucks
  GOAL: minimize $\max\{W_1, \ldots, W_k\}$.

You are requested to address the following points: i) design a (approximate) polynomial time algorithm to solve this problem; ii) is this problem NP-hard or not? Prove or propose a polynomial time, optimal algorithm.

Remarks: i) there is a simple, greedy algorithm that yields a a 2-approximation (in fact, slightly better than 2); ii) the following obvious lower bound(s) to the value of the optimal solution might be useful: $\max\{W_1, \ldots, W_k\} \geq \max\{\max_i w_i, \frac{1}{k}\sum_{i=1}^{n} w_i\}$.

Exercise 3. In many data analysis applications, one is given a set of points $X = \{x_1, \ldots, x_n\}$, where $x_i \in \mathbb{R}^d$ for some number d of dimensions. Subsets of points that are "close" represents subset of items that are "similar" in some sense. Given some integer k, one common task is devising a *partition* of X, so that points that are closer are mostly assigned to the same subset. One way to formalize this is the following:

[2] Assume that volume or anyway geometry of the boxes plays no role, maybe because their overall volume is small compared to available room in the trucks.

- Problem `Point partitioning`
  INPUT: Set $X = \{x_1, \ldots, x_n\}$ of points in some metric space with distance $d(\cdot, \cdot)$, integer k
  OUTPUT: a partition $C_1, \ldots, C_k$ of the points, i.e., all points are covered and $C_j \cap C_h = \emptyset$ for $j \neq h$
  GOAL: minimize $\max_j \max_{x,y \in C_j} d(x, y)$ (i.e., the maximum *diameter* of a cluster).

You are asked to propose a polynomial time approximation algorithm for this problem and to analyze its performance (approximation ratio and complexity).

Hints: 1) it is probably useful to first prove the following fact: Assume $\mu_1, \ldots, \mu_k$ points from X are selected as partition "representatives" according to some strategy. Define C_j as the subset of points in X whose closest representative is μ_j. Then, $\max_{x,y \in C_j} d(x, y) \leq 2 \max_{z \in C_j} d(z, \mu_j)$; 2) There is a simple greedy heuristic to select a set of k representatives that, though it might seem counterintuitive at first,[3] will result in a 2-approximation algorithm.

Exercise 4. The on line store Nozama has to open a set of warehouses to serve a set $U = \{c_1, \ldots, c_n\}$ of n cities. Denote by S the set of available sites for the warehouses. Opening a warehouse at location $s \in S$ will incur a cost w_s. Due to geographical distance, each city c_i can only be served from a warehouse located at a subset $S_i \subseteq S$ of the sites. The goal is selecting a subset W of the sites for the warehouses, so that each city can be served from at least one warehouse and the overall cost of opening the warehouses is minimized. More precisely:

- Problem `Warehouse opening`
  INPUT: Set $U = \{c_1, \ldots, c_n\}$ of cities; set S of available sites for the warehouses; For every $s \in S$: cost w_s for opening a warehouse at site s; for every city c_i: subset S_i of sites from which c_i can be served.
  OUTPUT: a set $W \subseteq S$ of sites where warehouses will be opened, such that each city can be served by at least one warehouse
  GOAL: minimize $\sum_{s \in W} w_s$

i) Give an optimal polynomial algorithm or prove that the problem is NP-hard; ii) Analyze correctness, complexity and approximation ratio (if applicable) of the algorithm.

Hint: Meditate on point i) before thinking of something very complicated.

Exercise 5. In graphics applications, it is sometimes necessary to accurately estimate the area of a possibly complex shape. In a simplified scenario, imagine you have a black and white screen consisting of a set P of n pixels that can be ON (lit) or OFF. A shape is simply the set of ON pixels and their number is (proportional to) the area of the shape.

Assume you can access the state of each pixel by calling the function `ON(p)` that, given the position `p` of a pixel, returns the pixel's state in $O(1)$ time. Then, a simple linear (in the total number of pixels) algorithm is simply invoking `ON(p)` for every pixel (so, P times), counting the number of pixels that are ON. Unfortunately, a linear algorithm may be too slow for some applications. This is one case in which randomization could be of some help.

Your goals in this exercise are the following:

1) design a randomized algorithm that, with sufficiently high probability, accurately estimates the area A (number of ON pixels) of a complex shape in $o(n)$ time. In more detail, the randomized algorithm is given the set P of pixels a subset A of which are ON. The algorithm can use `ON(p)` as a subroutine and needs to return an estimate X of A, with the following requirements:

[3] Before you give the analysis :-)

- $\mathbf{P}\left(|X - A| > \epsilon A\right) \leq \delta$. This means that, with probability at least $1 - \delta$ (e.g., $\delta = 0.05$), the estimate will not deviate by more than a factor ϵ (e.g., $\epsilon = 0.01$) from the true value of the area.
- The algorithm makes the smallest possible number of calls to `ON(p)`, compatibly with the constraints above and using the probabilistic tools you have (see below).

Note that, since you have to design a randomized algorithm, the estimate X is in general a random variable.

2) Bonus: Discuss for which ranges of the parameters ϵ, δ and the area A to estimate, the algorithm runs in $o(n)$ time.

`Hints:` i) relax and meditate on the scenario; ii) consider the following, simple primitive, that might be the core of your algorithm: sample a pixel uniformly at random[4] and invoke `ON(p)`. What information do you obtain? What is the probability that the pixel you sample is ON? iii) At some point you might need the following result, which is actually a consequence of Chebyshev's inequality[5] applied to the sum of independent Bernoulli variables: assume Z is the sum of independent Bernoulli variables. Then:

$$\mathbf{P}\left(|Z - \mathbf{E}\left[Z\right]| > \epsilon \mathbf{E}\left[Z\right]\right) \leq \frac{1}{\epsilon^2 \mathbf{E}\left[Z\right]}.$$

References

[1] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley, 2005.

[4] In practice, this means picking a couple of the screen's coordinate uniformly at random.

[5] https://en.wikipedia.org/wiki/Chebyshev's_inequality