CS345/CS345A : Design and Analysis of Algorithms
Semester II, 2013-14, CSE, IIT Kanpur

Assignment IV

Deadline : 5:00PM, 11th March

Important note:

There are three problems in this assignment. These problems deal with designing an efficient algorithm for some problems. In the solution for each of these problems, please provide only the following details.

1. For each algorithm, provide a short description in plain English. This should not be longer than 5-6 lines. This should be followed by a very neatly designed pseudocode.

2. If your pseudocode is unnecessarily long or if you are using vague names for variables or operations, marks may be deducted.

3. For each algorithm you design, you **must** analyse its time complexity and provide a proof of correctness. Strive for a short, formal, and complete proof. There will be **penalty** if your proof is too long or there is too much of digression. Instead of wasting time typing 3-page proof, think a bit hard for some time to come up with a short proof.

1 shortest routes

Attempt any one of the following problems.

1. (**An adventurous drive in Thar desert**)(*25 marks*)

   Your friend X has recently purchased a **THUNDERBIRD** motorcycle from Royal Enfield in India. He is very excited and wants to drive it from some town s to another town d in Thar desert (think of Sahara desert if you are more adventurous). He is provided with the complete road map of the desert - the roads, their lengths and junctions (where two or more roads meet). The mobike has a very natural limitation - it can drive for c kilometers with full fuel tank. Since the destination is very far, X must plan his route so that he can refill the fuel tank along the way. Note that the shortest route may not necessarily be the feasible route. You friend is bit confused and scared - what if he gets lost in Thar desert due to improper planning.

   Your friend is very proud of you since you are a student of IITK. He also knows that you have done a course on algorithms. He approaches you with full faith that you will help him find the *shortest feasible* route from s to d if it exists. Model the problem in terms of a weighted, undirected graph in which roads are edges, junctions are vertices, and some junctions have fuel stations, and design an efficient algorithm to find *shortest feasible* route from s to d and inform if no route is feasible. Assume that both s and d are also at junctions and the source s has a fuel station. Your algorithm should take time of the order of $O(mn \log n)$ where m is the number of edges and n is the number of vertices. You should not exploit the planarity of the input graph while designing your algorithm. In other words, your algorithm should work for any arbitrary undirected graph with positive edge weights and under the fuel constraint mentioned above. Also note that the shortest feasible route need not be a simple path. For example, consider the following situation : There are five towns : s, a, b, e, d where fuel stations are available at s, b, e only. The connecting roads are : (s, a) of length 200 km, (s, d) of length 300 km, (a, b) of length 40 km, (a, d) of length 150 km, (b, e) of length 200 km, (e, d) of length 200 km; and motor bike can travel 250 km with full fuel tank. For this road network, shortest feasible route from s to d is is $s \to a \to b \to a \to d$, and has length 430 km.

2. (**a special shortest path**)(*15 marks*) Given a directed graph on n vertices and m edges, where each edge has been assigned a positive weight. There are two parameters that characterize *lightness* of a path P: the first parameter is its *length* $\ell(P)$ which is defined as the number of edges of P. The second parameter is its *weight* $w(P)$ which is defined as the sum of weights of all its edges. A path P_1 is said to be *lighter* than path P_2 if either $\ell(P_1) < \ell(P_2)$ or $\ell(P_1) = \ell(P_2)$ and $w(P_1) < w(P_2)$. A path P from u to v is said to be the lightest path from u to v if no other path from u to v is lighter than P. Design an $O(m+n)$ time algorithm that for a source s and a destination d, compute the lightest path from s to d.

2 Optimizing your time during exam days

Suppose it's nearing the end of the semester and you're taking n courses each with a final project that still has to be done. Each project will be graded on the following scale : It will be assigned an integer number on a scale of 1 to $g > 1$, higher numbers being better grades. Your goal, of course, is to maximize your average grade on n projects. You have a total of $H > n$ hours in which to work on the n projects cumulatively, and you want to decide how to divide up this time. For simplicity, assume that H is a positive integer, and you will spend an integer number of hours on each project. To figure out how best to divide up your time, you have come up with a set of functions $\{f_i : i = 1, 2, ..., n\}$ (rough estimates, of course) for each of your n courses; if you spend $h \leq H$ hours on the project for course i, you will get a grade of $f_i(h)$. (You may assume that the functions f_i are nondecreasing : if $h < h'$, then $f_i(h) \leq f_i(h')$.).

So the problem is : Given these functions $\{f_i\}$, decide how many hours to spend on each project (in integer values only) so that your average grade, as computed according to the f_i, is as large as possible. In order to be efficient, the running time of your algorithm should appear as polynomial in n, g,and H; none of these quantities should appear as an exponent in your running time.

3 Dynamic Programming again

Attempt one of the following problems.

- (*25 marks*) A large collection of mobile wireless devices can naturally form a network in which the devices are the nodes, and two devices x and y are connected by an edge if they are able to directly communicate with each other (e.g., by a short-range radio link). Such a network of wireless devices is a highly dynamic object, in which edges can appear and disappear over time as the devices move around. For instance, an edge (x, y) might disappear as x and y move apart from each other and lose ability to communicate directly.

  In a network that changes over time, it is natural to look for efficient ways of maintaining a path between certain designated nodes. There are two opposing concerns in maintaining such a path: we want paths that are short, but we also do not want to have to change the path frequently as the network structure changes. That is, we would like a single path to continue working, if possible, even as the network gains and loses edges. Here is a way we model this problem.

  Suppose we have a set of nodes V, and at a particular point in time there is a set E_0 of edges among the nodes. As the nodes move, the set of edges changes from E_0 to E_1, then to E_2, then to E_3, and so on, to an edge set E_b. For $i = 0, 1, 2, ..., b$, let G_i denote the graph (V, E_i). So if we were to watch the structure of the network on the nodes V as a "time lapse", it would look precisely like the sequence of graphs $G_0, G_1, \ldots, G_b$. We will assume that each of these graphs G_i is connected.

  Now consider two particular nodes $s, t \in V$. For an $s - t$ path P in one of the graphs G_i, we define the *length* of P to be simply the number of edges in P, and we denote this by $\ell(P)$. Our goal is to produce a sequence of paths $P_0, \ldots, P_b$ so that for each i, P_i is an $s - t$ path in G_i. We want the paths to be relatively short. We also do not want there to be too many *changes* - points at which the identity of the path switches. Formally, we define $\mathit{changes}(P_0, \ldots, P_b)$ to be the number of indices i $(0 \leq i \leq b-1)$ for which $P_i \neq P_{i+1}$.

  Fix a constant $K > 0$. We define the *cost* of the sequence of paths $P_0, \ldots, P_b$ to be

$$\mathit{cost}(P_0, P_1, \ldots, P_b) = \sum_{i=0}^{b} \ell(P_i) + K \cdot \mathit{changes}(P_0, P_1, \ldots, P_b)$$

  Give a polynomial time algorithm to find a sequence of paths $P_0, \ldots, P_b$ of minimum cost.

- (*20 marks*) There are n jobs: $J_1, \ldots, J_n$. Job J_i has a deadline d_i and a required processing time t_i, and all jobs are available to be scheduled starting at time t_i, and all jobs are available to be scheduled starting at time s. For a job i to be done, it needs to be assigned a period from $s_i \geq s$ to $f_i = s_i + t_i$, and different jobs should be assigned nonoverlapping intervals. As usual, such an assignment of times will be called a schedule.

  Each job must either be done by its deadline or not at all. We will say that a subset J of jobs is schedulable if there is a schedule for the jobs in J so that each of them finishes by its deadline. Your problems is to select a schedulable subset of maximum possible size and give a schedule for this subset that allows each job to finish by its deadline.

  1. Prove that there is an optimal solution J (i.e., a schedulable set of maximum size) in which the jobs in J are scheduled in increasing order of their deadlines.
  2. Assume that all deadlines d_i and the required times t_i are integers. Give an algorithm to find an optimal solution. Your algorithm should run in time polynomial in the number of jobs n, and the maximum deadline $D = \max_i d_i$.