CS 180: Introduction to Algorithms and Complexity Midterm Exam

Feb 17, 2021
Name -
UID -
Section -

* Print your name, UID and section number in the boxes above, and print your name at the top of every page.
* Exams will be scanned and graded in Gradescope. Use Dark pen or pencil. Handwriting should be clear and legible.
. The exam is a closed book exam. You can use your notes taken during the lecture.
. There are 4 problems. Each problem is worth 25 points.
. Do not write code using C or some programming language. Use English or clear and simple pseudo-code. Explain the idea of your algorithm and why it works.
. Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.
. Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.

Problem 1

Cellphone company wants to install k towers to serve customers at locations $\mathcal{X} = x_1, ...x_n$. For simplicity we assume all locations are co-linear - they are points on the x axis. Each customer phone will be programmed to communicate with a single tower, and the company also has to decide the location of the k towers (the location of a tower is not necessarily the location of a customer). The assignment of customers to towers and the location of the towers are to be chosen such as to minimize the sum of the squares of the distances from a customer to the tower it is assigned to.

a.

Given a tower. Argue that in an optimal solution the customers assigned to it are consecutive locations in the sorted sequence of locations of customers.

b.

In an optimal solution what is the relation between between the location of customers served by a tower and the location of the tower? (High-school algebra)

c.

Assume in $\mathcal{X} = x_1, ..., x_n$ the locations are sorted. For consecutive customer locations in the sorted order $x_i, x_{i+1}, \ldots, x_j$ if we assign these customers to a single tower, and the tower serves only these locations, the cost is denoted by $C_{i,j}$. Give an efficient algorithm to find these costs for all (i, j), $i < j$. Each arithmetic operation is a unit cost. (hint: try to reuse computations)

d.

Outline an Dynamic Programming algorithm to solve the problem (Define the subproblems, and write the recursive algorithm). Analyse the complexity of the bottom up algorithm.

Solution 1

a.

By contradiction lets assume that the optimal group assigned to a single tower will not have consecutive elements in it. For example x_1, x_3. That means that for the tower location, we could either have:

1. A tower could be in between x_1, x_3 but that would mean that x_2 will be closer than at least one of the other group members.
2. A tower could be to the left of x_1 which means x_2 will be closer to the tower than x_3

3. A tower could be to the right of x_3 which means x_2 will be closer to the tower than x_1

Therefore consecutive items are going to be part of the optimal solution.

b.

Let the location of the towers be $\mathcal{G} = g_1, g_2...g_k$. Given an array with n elements $\mathcal{X} = x_1, ...x_n$ representing points on the x axis. We want to partition the n points into k parts where each group has a center $\mathcal{G} = g_1, g_2...g_k$. We partition the points such that the all distances from each point to its group center is minimal. $\sum_{x \in \mathcal{X}} min_{g \in \mathcal{G}} ||x - g||^2$. To minimize this expression, it is clear each tower will be the mean of the points in its group.

c.

Let $C_{i,j} = \sum_i^j (x_i - \mu_{i,j})^2$ be the cost of grouping $x_i, ..., x_j$ into one cluster with the optimal choice of centroid, $\mu_{i,j} = \frac{1}{j-i+1} \sum_i^j x_i$ the mean of the points. We want to construct a data structure that computes $C_{i,j}$ in constant time. This is a standard application of prefix sums. $C_{i,j} = \sum_i^j (x_i - \mu_{i,j})^2 = \sum_i^j x_i^2 + \mu_{i,j}^2 - 2x_i\mu_{i,j} = (j - i + 1)\mu_{i,j}^2 + \mu_{i,j} \sum_i^j x_i + \sum_i^j x_i^2$. At each iteration we can reuse the previous computation to compute the next $C_{i,j}$.

d.

Let $D[i][m]$ be the cost of optimally clustering $x_1, ..., x_m$ into i clusters. If $i = 1$ the cost of optimally clustering $x_1, ..., x_m$ is the cluster cost $C_{1,m}$. That is, $D[1][m] = C_{1,m}$ for all m. From sub-question c, this takes $O(n)$ time.

For all $i > 1$:
$D[i][m] = min_j^m D[i-1][j-1] + C_{j,m}$.

Notice that $D[i-1][j-1]$ is the cost of optimally clustering $x_1, ..., x_j - 1$ into $i - 1$ clusters and $C_{j,m}$ is the cost of clustering $x_j, ..., x_m$ into one cluster. This makes x_j the first point in the last and rightmost cluster. Let $T[i][m]$ be the argument that minimizes

$T[i][m] := argmin_{j=1}^m D[i-1][j-1] + C_{j,m}$.

It is possible there exists multiple j obtaining same minimal value. To make the optimal clustering unique, such ties are broken in favour of smaller j. Notice $x_{T[i][m]}$ is the first point in the rightmost cluster of the optimal clustering. Thus, given T one can find the optimal solution by standard backtracking. One can naively compute each entry of D and T using the two equations. This takes $O(n)$ time for each cell, thus D and T can be computed in $O(kn2)$ time using $O(kn)$ space.

Problem 2

Professor Gafni has developed a hardware priority queue on his laptop. The priority queue can store up to p elements. This special priority queue can perform the INSERT and EXTRACT-MIN operations in $O(1)$ time no matter how many elements are stored in the queue. We wish to use this new Gafni-queue to sort $n = p^2$ elements stored in memory.

Give an linear time algorithm to sort the n elements.

Solution 2

The ideas is to use a p-way merge of the elements using the hardware queue. Divide our n elements into p groups of p elements. Sort each of the groups using the hardware queue. This takes a total of $O(p)$ time. Now that we have p sorted lists, we feed the hardware queue the first element of each queue along with an indicator to the list it came from. This takes another $O(p)$ time to insert p elements to the queue. Extracting max from the hardware queue we add the minimum element to a result list, and add the next element from the list where the extracted minimum came from. Repeating this step until all elements have been extracted, the step takes $O(n)$ time. Total time $O(n + p) = O(n)$

Problem 3

In a country far far away, all highways connecting cities are one way, such that if the cities are nodes and highways are directed edges, then we get a simple strongly connected directed graph. Alas, this country is a greedy country: All highways have a toll (different for each highway), and so going from city i to city j we will have to pay the tolls on the road between them. In addition, each city has fee payable upon leaving the city. So, if I'm going from city i to city j through city w I will have to pay 2 tolls $i->w$ and $w->j$ in addition to two exit fees, from i and from w.

You relocated to the country because of the Pandemic.

a.

From your city, you wish to find the cheapest route to all other cities in the country. Suppose you have a computer program that solves the shortest path problem as discussed in class. How will you use it to solve our problem? The computer algorithm accepts directed weighted graphs. It does not understand the fees of cities.

b.

Design an $O(|E|logn)$ algorithm to find the shortest path to all other cities ($|E|$ is the number of roads, n is the number of cities. To get this complexity you have to make an obvious simple assumption).

c.

You live in city i and for all cities j, if you visit j you cannot crash there for too long, and eventually will have to return home to i. For all j find the minimum cost of the round trip. Design, give $O(|E|logn)$ algorithm.

Solution 3

a.

Since the program only accepts weighted graphs, we will construct a graph $G = (V, E)$ where each vertex is a city, and edges are the highways going between cities. Weights of the edges will be the toll cost on the highway + the exit fee from the originating city such that for an edge between city i and city j the cost $w(e_{i,j})$ = toll of highway (i, j) + exit fee from i. We feed the graph to the program and get our answer.
The cost for constructing this graph is $O(V + E)$

b.

We would like to solve this question using Dijkstra's algorithm. The assumption we need to make is that the costs on weights are strictly positive. That is a fair assumption since we are referencing fees and those are typically positive. We modify Dijkstra to consider the toll on a road in addition to the exit fee from a city. The change requires an additional constant cost to add the two weights. Overall we satisfy the requirement since Dijkstra has complexity $O(ElogV)$.

0.1 c.

The idea here is to use Dijkstra in the same way we used it in the previous questions only on the transposed graph (edged are reversed) G^T. Running in this way starting from our destination we will get all cheapest paths back home. For example: the way back from i to j in the original graph had an edge from i to j but has an edge from j to i in the transposed graph. The exit fee remains that of i as well as the toll from i to j. However when we run Dijkstra from the destination on the transpose we will then find the cheapest way back home.

Question 4

Given a undirected weighted graph $G = (E, V)$. With each spanning tree we associate a number which is the weight of the heaviest edge in the tree. Find the spanning tree whose number is minimum among all spanning trees.
Explain.
(Hint: this involves a simple observation. The question is how to justify this observation. There are many ways to justify. The easiest is probably to recall where "exponentiation" appeared in the HW and use it.)

Solution 4

Kruskal (or any other MST finding algorithm) Solves this.

By contradiction lets assume we can find a tree T' that has a lower heaviest weight edge e' then our MST T. That would mean that the weight of our heaviest edge $w(e) > w(e')\forall e' \in E'$ which contradicts our tree being an MST.