CSC373 Fall'19
Assignment 2

Due Date: Nov. 1, 2019, by 11:59pm

Instructions

1. Be sure to include your name and student number with your assignment. Typed assignments are preferred (e.g., PDFs created using LaTeX or Word), especially if your handwriting is possibly illegible or if you do not have access to a good quality scanner. Please submit a single PDF on MarkUS at https://markus.teach.cs.toronto.edu/csc373-2019-09.
2. You will receive 20% of the points for any (sub)problem for which you write "I do not know how to approach this problem." (You will receive 10% if you leave the question blank and do not write this or a similar statement.) Not applicable to BONUS questions.
3. You may receive partial credit for the work that is clearly on the right track. But if your answer is largely irrelevant, you will receive 0 points.
4. This assignment has 3 questions (worth 25, 20, 30 marks) and one bonus question (worth 15 marks).

Q1 [25 Points] Fixing Pictures

Figure 1: Scanned picture with a black dog hair.

You have scanned some old black-and-white photographs at home. Unfortunately a dog hair on the scanner glass has corrupted your pictures. You want to correct the picture by removing the dog hair. But first, you need to find which pixels (more accurately, chain of pixels) represent the dog hair.

Your input is a picture P of $m \times n$ pixels. You are given the intensity $P(i,j)$ of each pixel (i,j), which is a value of grey between 0 (black) and 1 (white). The hair is long, blackish, and at most one pixel wide. You may assume that the hair only ever passes through at most one pixel in any row of pixels. The figure above shows a hair on an old picture of the CN tower.

The likelihood that pixel (i,j) is part of the hair is given by

$$\ell(i,j) = \begin{cases} 0 & \text{if } P(i,j) \geq h, \\ 1 - \frac{P(i,j)}{h} & \text{if } P(i,j) < h, \end{cases}$$

where $h \in (0,1)$ is a given threshold. The idea is that any pixel lighter than h is definitely not part of the hair, and pixels darker than h have more probability of being part of the hair the darker they are.

Your goal is to find a chain of pixels C. A chain is a set of pixels $\{(r, p_r) : r \in [i,j]\}$ containing one pixel from every row between row i and row j such that for $r \in (i,j)$, pixel p_r is a neighbour of pixels p_{r-1} and p_{r+1}. Here, we are using the standard notion of "neighbourhood" under which each pixel has at most eight neighbours (up, down, left, right, northeast, northwest, southeast, southwest). The likelihood of a chain being the hair is the sum of likelihoods of its pixels being part of the hair: $\ell(C) = \sum_{(i,j) \in C} \ell(i,j)$.

(a) [17 Points] Suppose you want to compute the maximum likelihood of any chain being the hair. Use dynamic programming to design an algorithm for this.

- [3 Points] Clearly define the quantity that your DP is computing. (For example, in the shortest path DP that we did in class, we defined: "Let $OPT(t,i)$ be the length of the shortest path from s to t using at most i edges.")
- [5 Points] Write a Bellman equation for this quantity, and briefly reason why your equation is correct.
- [3 Points] Write the initial conditions.
- [3 Points] What is the running time and space complexity of your algorithm (say, in a top-down implementation)?
- [3 Points] In what order would you compute the quantities in a bottom-up implementation? Briefly reason why this order works.

(b) [5 Points] Suppose now that you actually want to find the chain with the maximum likelihood of being the hair. If there are multiple such chains, you want to return the maximum likelihood chain with the fewest number of pixels. Modify your DP so that it allows you to return the desired chain.

(c) [3 Points] (Open Ended Question) Is the likelihood function ($\ell(i,j)$ and $\ell(C)$) well-designed in your opinion? Describe a type of image on which this algorithm would perform poorly for the ultimate objective of finding the hair. Suggest a modification to the likelihood formula that you think might perform better.

Q2 [20 Points] Approximating Functions

You are given $n+1$ points $\{p_0, p_1, .., p_n\}$, where $p_i = (x_i, y_i)$ for $0 \leq i \leq n$. The points are sorted in the increasing order of (unique) x_i's: that is, $x_0 < x_1 < \ldots < x_n$.

In reality, these are sample points on the curve of some unknown function $f : \mathbb{R} \to \mathbb{R}$ with $y_i = f(x_i)$ for each $0 \leq i \leq n$. Your goal is to approximate this function using a sequence of line segments.

If you use the sequence of n line segments joining all adjacent points (i.e. joining p_i and p_{i+1} for $0 \leq i < n$), then this provides an excellent approximation since it passes through (x_i, y_i) for each $0 \leq i \leq n$. However, it uses too many line segments.

If you replace the line segments between p_i and p_j (where $j > i$) with a single straight line connecting p_i and p_j, then the error introduced is given by

$$E[i, j] = \sum_{\ell=i+1}^{j-1} |y_\ell - \hat{y}_\ell|,$$

where $(x_\ell, \hat{y}_\ell)$ is point through which the line segment joining p_i and p_j passes. Note that $E[i, i+1] = 0$ for all i.

Suppose every line segments costs C. If you select a subset of points (without changing their order) $(p_{i_0}, \ldots, p_{i_k})$ and use the k line segments connecting adjacent points p_{i_m} and $p_{i_{m+1}}$ for $0 \leq m < k$, the total error of this approximation is $\sum_{m=0}^{k} E[i_m, i_{m+1}] + k \cdot C$. Your goal is to find the optimal approximation which minimizes this error. Note that k is not given to you. You need to optimize the number of line segments to use too. See Figure for an illustration of the problem.

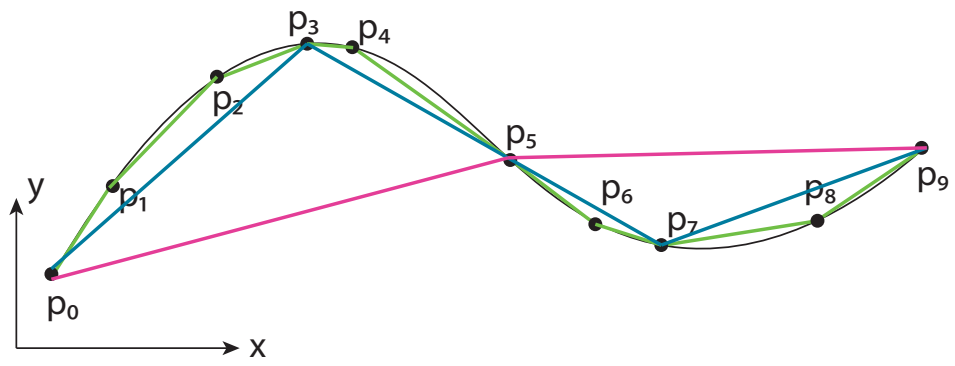

Figure 2: The function (black curve) with samples $p_0, \ldots, p_9$ is shown approximated using 9 (green), 3 (blue), and 2 (pink) line segments. The optimal approximation will try to use fewer line segments but also try to achieve a low approximation error.

(a) [3 Points] Express the total error of an approximation $\{p_{i_0}, \ldots, p_{i_k}\}$ explicitly in terms of the inputs to the problem ((x_i, y_i) for $0 \leq i \leq n$).

(b) [17 Points] Describe a dynamic programming solution that finds the optimal subset of points $\{p_{i_0}, p_{i_1}, .., p_{i_k}\}$ minimizing the total error.

- [3 Points] Clearly define the quantity or quantities that your DP is computing.
- [5 Points] Write a Bellman equation for these quantities, and briefly reason why your equation is correct.
- [3 Points] Write the initial conditions.
- [3 Points] What is the running time and space complexity of your algorithm (say, in a top-down implementation)?
- [3 Points] In what order would you compute the quantities in a bottom-up implementation? Briefly reason why this order works.

You will get the 17 marks for part (b) if you describe an $O(n^3)$ time algorithm. You will get 5 bonus marks if your solution works in $O(n^2)$ time.

Q3 [30 Points] Maximum Flow

Suppose you are given a network N, a *maximum* flow f on N, and one edge $e_0 \in N$ such that $f(e_0) = c(e_0)$. You are also given that f has *integer* flow values $f(e)$ for all edges e.

(a) [5 Points] If we decrease $c(e_0)$ by 1 (i.e., let $c'(e_0) = c(e_0) - 1$ but other capacities remain the same), then one might expect that the maximum flow on N might also decrease by one unit. But does this always happen?

Either give a specific example where the maximum flow on N does not change (and show that this is the case), or give a general argument that the maximum flow on N always changes.

(b) [10 Points] Irrespective of your answer on the previous part, there are cases when this change in capacity causes the maximum flow to decrease.

Give an efficient algorithm that takes a network N, a maximum integral flow f on N, and one edge $e_0 \in N$ such that $f(e_0) = c(e_0)$ and that determines a new maximum flow in the network N', where $N' = N$ except for $c'(e_0) = c(e_0) - 1$.

Include a brief justification that your algorithm is correct and a brief analysis of your algorithm's worst-case runtime (which should be as small as possible).

(c) [5 Points] If we increase $c(e_0)$ by 1 (i.e., let $c'(e_0) = c(e_0) + 1$ but other capacities remain the same), then one might expect that the maximum flow on N might also increase by one unit. But does this always happen?

Either give a specific example where the maximum flow on N does not change (and show that this is the case), or give a general argument that the maximum flow on N always changes.

(d) [10 Points] Irrespective of your answer on the previous part, there are cases when this change in capacity causes the maximum flow to increase.

Give an efficient algorithm that takes a network N, a maximum integral flow f on N, and one edge $e_0 \in N$ such that $f(e_0) = c(e_0)$ and that determines a new maximum flow in the network N', where $N' = N$ except for $c'(e_0) = c(e_0) + 1$.

Include a brief justification that your algorithm is correct and a brief analysis of your algorithm's worst-case runtime (which should be as small as possible).

BONUS QUESTION

Q4 [15 Points] Bloober (Remember: 20% rule does not apply to this bonus question)

A start-up in the competitive self-driving bus ride sharing space, Bloober, would like to figure out the minimum number of buses it needs in its fleet to service the n routes: each route r_i departs from bus stop $s(r_i)$ at time $d(r_i)$ and ends at stop $e(r_i)$. The travel time between any two bus stops a and b is $t(a, b)$.

Note that we only care about the start stop, the end stop, and the travel time $t(s(r_i), e(r_i))$ of going from the start stop to the end stop. We do not care about any stops in between.

A bus that is not in use (after it has reached the end stop of its current route) can be used to service another route if it can reach the start stop of that route by the scheduled departure time of that route.

(a) [8 Points] Design an algorithm that can test if Bloober can service the routes using exactly k buses. You get 4 marks for describing the algorithm, and 4 marks for proving its correctness.

You can assume that each bus can reach the start stop of the first route that it services by the departure time of that route, and once a bus reaches a stop, it is allowed to just stay there for a while. Note thus that you can trivially service n routes with n buses. The goal is clearly to service them with a smaller number of buses by reusing buses to service multiple routes where the travel times are compatible.

(b) [4 Points] What is the time complexity of your algorithm, assuming the naïve implementation of the Ford Fulkerson algorithm that runs in $O(mnC)$ time, where m, n, and C are the number of edges, the number of vertices, and the sum of capacities of edges from the source vertex, respectively.

(c) [3 Points] Use the solution to part (a) to determine the minimum number of buses needed in the fleet. What is the time complexity of the overall algorithm?