CS 458, Data Structures and Algorithms II
Homework 1

April 16, 2020

Due on 11:59am April 22, 2020. Collaboration is not allowed

For all questions you need to prove your statements or at least provide a justification for the correctness of your algorithm and the calculation of their runtime.

Problem 1 (50 points)

In the first recitation session we designed an algorithm with $O(\log k \cdot n^3)$ running time that given a graph $G = (V, E)$ computes all-pairs shortest paths where each shortest path uses **exactly** k edges, denoted by $D_{i,j}^{=k}$. Note these paths may form a circle as shown in the example below. Particularly, we modified the recursive formula with

$$\hat{d}^k(v_i, v_j) = \min_{v_l \in V \setminus v_j} \{d^{k-1}(v_i, v_l) + w(v_l, v_j)\}$$

and defined a new Extend Shortest Path procedure and applied the repeated squaring. Consider now the similar problem that for a given k we want to compute a shortest path that uses at least k edges denoted by $D_{i,j}^{\geq k}$. Note that the cost of the shortest path that uses at least k edges never larger than the cost of the shortest path that uses exactly k edges.

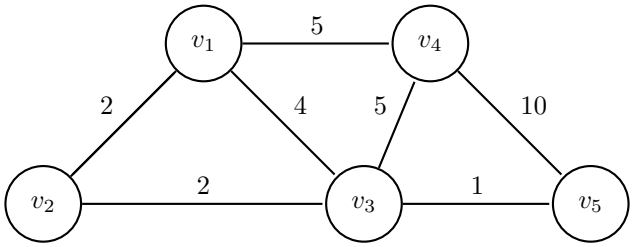

Figure 1: The shortest path with **exactly** $k = 2$ edges from v_2 to v_3 is $v_2 \to v_1 \to v_2$ for a total cost of $D_{2,3}^{=k} = w(v_2, v_1) + w(v_1, v_3) = 2 + 4 = 6$
The shortest path with **at least** $k = 2$ edges vertices from v_2 to v_3 is $v_2 \to v_3 \to v_5 \to v_3$ for a total cost of $D_{2,3}^{\geq k} = w(v_2, v_3) + w(v_3, v_5) + w(v_5, v_3) = 2 + 1 + 1 = 4$.
Note that the actual shortest path from v_2 to v_3 is $v_2 \to v_3$ has total cost of $D_{i,j}^{\geq 0} = w(v_2, v_3) = 2$

a. (**25 points**) Modify the procedure we discussed in the recitation session to compute $D_{i,j}^{\geq k}$ for all pairs i, j. The running-time of your algorithm should be $O((\log k + n - k) \cdot n^3)$.

b. (**25 points**) It turns out we can solve this problem even faster. Give an algorithm that takes as input $D_{i,j}^{=k}$ and produces $D_{i,j}^{\geq k}$ for all pairs i, j with running time $O(n^3)$. What is the total runtime?

Problem 2 (50 points)

In Lecture 3 we showed how to calculate the longest increasing subsequence. This requirement turned out too strict and we are interested to finding the **longest close-to-increasing subsequence**.

We say that a sequence of numbers Z is close-to-increasing if the size of any decreasing substring is at most 2. So given two consecutive elements in the sequence Z_i and Z_{i+1} it must either be that

- $Z_i < Z_{i+1}$
- or if $Z_i > Z_{i+1}$ it must be that $Z_{i+1} < Z_{i+2}$.

Below are some examples of of close-to-increasing sequences:

A	=	11	10	18	17	25	33	30	34	27	29
B	=	8	99	33	34	40	11	13	20	23	22

As we can see $A_2 = 10 < 11 = A_1$ but $A_3 = 12 > 10 = A_2$ so the constrain is satisfied. Same is true in other places where monotonicity fails at A_7, A_{11}, A_{14}. Similar for sequence B.

The following are examples of sequences that are not close-to-increasing:

C	=	10	11	13	12	18	30	27	22	33	40	44
D	=	17	16	1	11	13	12	15	19	30	25	33

We have $C_6 = 30 > 27 = C_7$ and $C_6 = 27 > 22 = C_9$ so the constraint is violated. Similarly $D_1 > D_2 > D_3$ so the sequence D is not close-to-increasing. For example if $X = D$ the longest close-to-increasing subsequence is $Z = 17, 1, 11, 13, 12, 15, 19, 30, 25, 33$ for a size of 11. You may assume all numbers in X are **unique**.

a. (**25 points**) Give a dynamic programming algorithm using the following optimal subproblem structure $LCIS^{+}(i, j)$: is the longest closed to increasing subsequence that ends and $X_i X_j$, i.e., its last element is X_i and its second to last element is X_j. Give a recursive formula, prove its correctness and write a bottom-up implementation. What is the runtime of your algorithm?

b. (**25 points**) Give a dynamic programming algorithm using the following optimal subproblem structure $LCIS^{+}(i)$: is the longest closed to increasing subsequence that ends and X_i and the auxiliary optimal subproblem structure $LCIS^{+,inc}(i)$ the longest closed to increasing subsequence that ends and X_i in an increasing fashion (the second to last element is smaller than X_i). Give a recursive formula for both optimal subproblem structures, prove their correctness and write a bottom-up implementation. What is the runtime of your algorithm?