CS2800: Design and Analysis of Algorithms

Mini Quiz #4

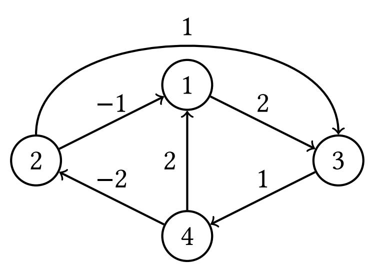
April 26, 2023

Max Marks: 10
Duration: 1 hour

Instructions

- Write your name and roll number clearly in the answer sheet before you start.
- This quiz contains 3 questions totalling 10 marks. All questions are compulsory.
- Please be as precise as possible in your answers. Write only what is necessary to substantiate your answer.
- Anything used in class can be used as is. If you are using a result that was given in the problem set directly, you can mention it and use it. Anything else must be fully justified.
- Write only one answer for a question. If you make multiple attempts, please strike off the attempts that you think are incorrect.
- Whenever you describe an algorithm, you must include a proof of its correctness and an analysis of its running time. **Algorithms/pseudocode without any explanation will receive partial/zero marks.**

1. **(3 marks)** Consider the following graph with the vertices numbered as given, and the edge weights.



Write the matrix D of length of shortest paths at each step of the execution of the Floyd-Warshall algorithm on this graph. Use the same vertex numbering as in the figure while running the algorithm.

Solution: The matrices are as follows

$$D_0 = \begin{pmatrix} 0 & \infty & 2 & \infty \\ -1 & 0 & 1 & \infty \\ \infty & \infty & 0 & 1 \\ 2 & -2 & \infty & 0 \end{pmatrix} D_1 = \begin{pmatrix} 0 & \infty & 2 & \infty \\ -1 & 0 & 1 & \infty \\ \infty & \infty & 0 & 1 \\ 2 & -2 & 4 & 0 \end{pmatrix} D_2 = \begin{pmatrix} 0 & \infty & 2 & \infty \\ -1 & 0 & 1 & \infty \\ \infty & \infty & 0 & 1 \\ -3 & -2 & -1 & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & \infty & 2 & 3 \\ -1 & 0 & 1 & 2 \\ \infty & \infty & 0 & 1 \\ -3 & -2 & -1 & 0 \end{pmatrix} D_4 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 \\ -3 & -2 & -1 & 0 \end{pmatrix}$$

2. **(3 marks)** Suppose someone has run an APSP algorithm on a graph $G(V, E, w)$ and given you the predecessor matrix P where $P[u, v]$ is the vertex before v in the shortest path from u to v. Given $G(V, E, w)$, P, and a vertex s, give an efficient algorithm to find a vertex v such that the length of the shortest path from s to v is the largest among all the vertices. You are **not** given the matrix D of the length of shortest paths, but only the graph $G(V, E, w)$ with the weights on each of the edges.

Solution: For every v we can follow the parent pointers using $P[s, v]$ to obtain the shortest path from s to v and compute the shortest path from s to v and all the vertices that lie in this shortest path. We can do this for every v, by memoizing the shortest distances of the intermediate nodes. This will give an $O(n)$-time algorithm as shown below. We will assume that the every vertex is reachable from s - the algorithm can be suitably modified otherwise.

Algorithm 1: FindDistance(v)

if $D_s(v) \neq \infty$ **then return** $D_s(v)$;
$D_s(v) \leftarrow$ FindDistance($P[s, v]$) $+ w(P[s, v], v)$
return $D_s(v)$

Algorithm 2: ShortestPaths

Input: $G(V, E, w), P, s$
Output: Array D_s where $D_s(v)$ is the shortest path distances from s to v
foreach $v \in V$ **do**
  if $v = s$ **then** $D_s(v) = 0$ **else** $D_s(v) = \infty$;
end
foreach $v \in V$ **do**
  FindDistance(v)
end
return $\arg\max_{v \in V}\{D_s(v)\}$

3. **(4 marks)** A string S is said to be a k-shuffle of two strings S_1 and S_2 if S can be written by using letters from S_1 and S_2 alternately such that at most k letters from a string are used consecutively in the same order as they appear in the string. For instance, if $S_1 = abba$ and $S_2 = abba$, then the string $S = aabbbbaa$ is a 2-shuffle of S_1 and S_2, whereas $S = aabbabba$ is not a 2-shuffle, but it is a 3-shuffle.

Suppose you are given three strings S, S_1, and S_2 and an integer k, give an efficient algorithm to check if S is a k-shuffle of S_1 and S_2. If you are using dynamic programming, you must write the recurrence, and base cases clearly before the pseudocode.

Solution: Suppose that S_1, S_2 and S are length m, n and $m + n$ length strings. Define the following function: $\mathsf{Shuffle}_k(S_1, S_2, S)$ returns 1 if S is a k-shuffle of S_1 and S_2 where a letter from S_1 is taken first in the shuffle. We are interested in finding the value

$$\max\{\mathsf{Shuffle}_k(S_1[1, m], S_2[1, n], S[1, m+n]), \mathsf{Shuffle}_k(S_2[1, n], S_1[1, m], S[1, m+n]\}.$$

We can define the function $\mathsf{Shuffle}_k(S_1[1, m], S_2[1, n], S[1, m+n])$ recursively as follows: We will first guess the number $i \leq k$ of letters from S_1 that appear in the shuffle. If this matches the first i letters of S, then we recursively check if the remaining can be a k-shuffle.

$$\mathsf{Shuffle}_k(S_1[i,m], S_2[j,n], S[i+j-1, m+n]) = \\ \max_{\ell \le k}\{\mathsf{Shuffle}_k(S_2[j,n], S_1[i+\ell, n], S[i+\ell+j-1, m+n]) \cdot \mathbf{I}_\ell\}$$

where $\mathbf{I}_\ell$ is 1 if $S_1[i, i+\ell-1] = S[i+j-1, i+j+\ell-2]$.

The base cases would be the following.

$$\mathsf{Shuffle}_k(S_1[m,m], S_2[n,n], S(m+n-1, m+n)) = 1 \text{ if } S_1[m] = S[m+n-1] \text{ and } S_2[n] = S[m+n]$$
$$\mathsf{Shuffle}_k(S_2[n,n], S_1[m,m], S(m+n-1, m+n)) = 1 \text{ if } S_2[n] = S[m+n-1] \text{ and } S_1[m] = S[m+n]$$

Also, if one of them is an empty string, say S_1, then we need to check if S_2 has at most k letters and is equal to S.

I am not writing the pseudocode, but this will give an $O(n^2k)$-time algorithm.