CS-510 Algorithms Homework 4 - All Pairs Shortest Paths **LUMS - Fall 2020**

With this set of problems, we will develop algorithms for All-Pairs Shortest Paths (APSP) in graphs.

PROBLEM: (APSP). *Given a graph $G = (V, E)$ with $w : E \to \mathbb{R}$ that gives a real weight to every edge. Find the shortest path from every vertex to every "other" vertex.*

The graph could be directed or undirected but we assume it to be a simple graph (no multiple edges and no self-loops). Let $|V| = n$ and $|E| = m$. Denote by A the weighted adjacency matrix of G, i.e.

$$A(u,v) = \begin{cases} 0 & \text{if } u = v \\ w_{ij} = w((i,j)) & \text{if } (u,v) \in E \\ \infty & \text{if } (u,v) \notin E \end{cases}$$

Definition. *For $u, v \in V$, let $d(u, v)$ be the total weight of a shortest path from u to v.*

In case v is not reachable from u, then $d(u, v) = \infty$ and $d(u, u) = 0$ for all u. This information can be visualized as a matrix (called the distance matrix) D. We will focus on computing the pairwise distance only, but the solutions that we develop can easily be extended to find the actual shortest paths too.

APSP has many applications including all scenarios that we discussed for SSSP. **The following is from blog of Sasha Goldshtein, Software Engineer at Google Research.**

> We're implementing a service that manages a set of physical backup devices. There is a set of conveyor belts with intersections and robot arms that manipulate backup tapes across the room. The service gets requests such as "transfer a fresh tape X from storage cabinet 13 to backup cabinet 89, and make sure to pass through formatting computer C or D".
> When the system starts, we calculate all shortest routes from every cabinet to every other cabinet, including special requests such as going through a particular computer. This information is stored in a large hashtable indexed by route descriptions and with routes as values.
> System startup with 1,000 nodes and 250 intersections takes more than 30 minutes, and memory consumption reaches a peak of approximately 5GB. This is not acceptable.

Such a distance matrix is also input for agglomerative or hierarchical clustering.

Suppose the graph is input in the adjacency lists representation.

Problem 1. *Briefly describe a solution for APSP problem using the Dijkstra's algorithm. Discuss when can this solution be adopted?*

Problem 2. *What is the runtime of the above algorithm?*

Problem 3. *If edge weights are allowed to be negative, how can we solve this problem using a known solution. What is it's runtime?*

For the solution we develop below we will assume that G is input as the adjacency matrix A. We assume that there are no negative weight cycles in G. If there are the algorithm will detect it and declare so. This is just for the presentation to become simple.

Just like Bellman-Ford algorithm to be able to talk about subproblems and devise a dynamic programming solution, we characterize the size of the shortest path by the number of edges it contains.

Let $S_h(u, v)$ be a shortest (w.r.t total weight) path from u to v of length at most h (w.r.t number of edges or hop-length). Let $L_h(u, v)$ be the weight of $S_h(u, v)$.

Problem 4. *Argue that the goal of APSP problem is to find $L_{n-1}(u, v)$ for all u and v, if there are no negative cycles in G.*

We therefore, focus on determining $L_{n-1}(u, v)$ for all u and v. As typical in the dynamic programming solution, we try to express the optimal solution $S_h(u, v)$ in terms of optimal solution to "smaller" subproblems.

Problem 5. *Let $S_h(u, v)$ be $u, x_1, x_2, \ldots, x_{h-1}, v$. Prove that the subpath $u, x_1, x_2, \ldots, x_{h-1}$ is $S_{h-1}(u, x_{h-1})$. That is the subpath from u to the penultimate vertex in $S_h(u, v)$ is a shortest path from u to x_{h-1} that uses at most $h - 1$ edges.*

Problem 6. *Let $S_h(u, v)$ be $u, x_1, x_2, \ldots, x_{h-1}, v$. Prove that for $i = 1, \ldots, (h-1)$, the subpaths $u, x_1, x_2, \ldots, x_i$ is $S_i(u, x_i)$.*

Next we derive a recurrence relation for $L_h(u, v)$.

Problem 7. *Prove that*

$$L_h(u,v) = \begin{cases} 0 & \text{if } h = 0 \text{ and } u = v \\ \infty & \text{if } h = 0 \text{ and } u \neq v \\ \min\left\{ L_{h-1}(u,v), \min\limits_{1 \leq k \leq n} \left\{ L_{h-1}(u,k) + w_{kv} \right\} \right\} & \text{if } h \geq 1 \end{cases} \tag{1}$$

Problem 8. *Prove that the recurrence in (1) is equivalent to*

$$L_h(u,v) = \begin{cases} 0 & \text{if } h = 0 \text{ and } u = v \\ \infty & \text{if } h = 0 \text{ and } u \neq v \\ \min\limits_{1 \leq k \leq n} \left\{ L_{h-1}(u,k) + w_{kv} \right\} & \text{if } h \geq 1 \end{cases} \tag{2}$$

As we discussed that our goal is to compute $L_{n-1}(u, v)$ for all u and v. We can implement recurrence (2) in a straightforward manner, but there will be redundant functions calls (try to picture this). Therefore, we compute the matrix $L_{n-1}(u, v)$ in a bottom-up manner but first computing the L_0 matrix, then the L_1 matrix and so on.

Problem 9. *Prove that the matrix $L_1 = A$ (the weighted adjacency matrix of G)*

Problem 10. *Show that the following algorithm computes L_h given L_{h-1} and A by just implementing (2).*

Algorithm 1 Algorithm to Compute L_h from L_{h-1} and A

```
1: L_h ← INFINITY(n × n)                          ▷ Initialize the matrix with all entries as ∞
2: for i = 1 to n do                               ▷ Index of u
3:     for j = 1 to n do                           ▷ Index of v
4:         for k = 1 to n do                       ▷ Limits of the min
5:             L_h[i][j] ← min{ L_h[i][j] , L_{h-1}[i][k] + A[k][j] }
6: return L_h
```

Problem 11. *Recall the following algorithm for multiplying two square matrices*

Algorithm Algorithm to Compute $Z = X \times Y$

```
1: Z ← ZEROS(n × n)                                ▷ Initialize the matrix with all entries as 0
2: for i = 1 to n do
3:     for j = 1 to n do
4:         for k = 1 to n do
5:             Z[i][j] ← Z[i][j] + X[i][k] · Y[k]][j]
6: return Z_h
```

Which operation would have to be redefined and how so as the L_h computation becomes matrix multiplication?

Problem 12. *Argue that the runtime of the algorithm to compute L_h is $O(n^3)$*

Problem 13. *What will be the total runtime and space requirement of the algorithm to compute the L_{n-1} matrix.*

We denote this "multiplication operation" of L_{h-1} with A by $\odot$. That is we write $L_h := L_{h-1} \odot A$. Similarly we denote by $X^{(i)}$ to be the power of a matrix X w.r.t the operator $\odot$, i.e. $X^{(i)} := \underbrace{X \odot X \odot \ldots \odot X}_{i \text{ times}}$

The above algorithm thus correspond to the following sequence of computation

$L_1 = A = A^{(1)}$
$L_2 = L_1 \odot A = A^{(2)}$
$L_3 = L_2 \odot A = (L_1 \odot A) \odot A = A^{(3)}$
$L_4 = L_3 \odot A = (L_2 \odot A) \odot A = ((L_1 \odot A) \odot A) \odot A = A^{(4)}$
and so on

We know that regular matrix multiplication is associative, i.e. for three matrices X, Y and Z, we have $(XY)Z = X(YZ)$.

Problem 14. *Show that this modified multiplication (under operator $\odot$) is also associative. In other words prove that $(X \odot Y) \odot Z = X \odot (Y \odot Z)$*

The reason we are interested in the associativity property is that it allows us to use the so-called repeated squaring to achieve the sequence of matrix multiplication. By the associativity we get $A \times A \times A \times A = (A \times A) \times (A \times A) = A^2 \times A^2 = A^4$.

Problem 15. *How many $n \times n$ matrix multiplications are required to compute $Z = \underbrace{Y \times Y \times \ldots \times Y}_{n \text{ times}}$. Assume n is a power of 2*

Problem 16. *Give an $O(n^3 \log n)$ algorithm to compute the matrix L_{n-1} using repeated squaring. Note that Algorithm 1 computes $L_h = L_{h-1} \odot A$. Assume that $n = 2^t$ for $t \in \mathbb{Z}^+$.*

0.1 Finding the actual paths

The actual shortest paths can also be found by computing another $n \times n$ matrix $Pred$. Where $Pred(u, v)$ will contain the predecessor of v on a shortest path from u to v. Note that each row of $Pred$ matrix correspond to the shortest path tree rooted at u.

It is easy to see that we can compute the shortest paths from this matrix. The shortest path from u to v is the reverse sequence of following the entries in $Pred$ matrix from $Pred(u, v)$.

The matrix $Pred$ can be computed by keeping the record of updates for $L_h(u, v)$. Every time we update $L_h(u, v)$ in Line 5 of Algorithm 1, we update $Pred(u, v)$ to the corresponding k.

0.2 Negative Cycle

We assumed that there was no negative cycle in the graph, but nowhere the algorithm uses the fact. The APSP problem is actually posed as find all pairs shortest paths or find a negative cycle where you can't find a negative cycle.

Problem 17. *Give an efficient algorithm to determine if the graph has a negative cycle.*

Problem 18. *A very beautiful and interesting exercise is to give a matrix-vector multiplication based implementation of the Bellman-Ford algorithm. This would be used for instance when the graph is input as an adjacency matrix.*

1 The Floyd-Warshall algorithm for APSP

In this section we will develop another dynamic programming solution for APSP. This algorithm uses a different way to find the substructures and subproblems. We assume that there are no negative weight cycles in G.

The previous algorithm characterized shortest paths by their hop-lengths and determined problem sizes and subproblems by length. The Floyd-Warshall formulation considers the intermediate vertices in shortest paths.

Suppose vertices are ordered arbitrarily as $v_1, v_2, \ldots, v_n$. Denote by V^k subset of first k vertices, i.e. $V^k := \{v_1, \ldots, v_k\}$. For verties u and v, let Opt-Path(u, v, k) be a shortest path from u to v such that all intermediate vertices are in V^k and let Opt-Val(u, v, k) be the weight of the path Opt-Path(u, v, k). Please try to understand the definition of Opt-Path(u, v, k) with an small example.

Problem 19. *What is the minimum and maximum possible hop-length of Opt-Path(u, v, k) (can't have repeated vertices).*

Problem 20. *Prove that Opt-Path(u, v, k) is a simple path (can't have repeated vertices).*

Next we argue about the structure of the paths Opt-Path(u, v, k) and find an optimal substructure property in it. We consider two cases (that cover all possibilities) on the basis of whether or not $v_k \in (u, v, k)$.

Problem 21. *Suppose $v_k \notin$ Opt-Path(u, v, k). Identify the set such that all intermediate vertices will belong to it.*

Problem 22. *Suppose $v_k \notin$ Opt-Path(u, v, k). Prove that all Opt-Path(u, v, k) is Opt-Path$(u, v, k-1)$*

Problem 23. *Suppose $v_k \in$ Opt-Path(u, v, k). Then since v_k is not necessarily the vertex immediately preceding v. Let P_1 and P_2 be the subpaths of Opt-Path(u, v, k) from u to v_k and from v_k to v, respectively. Prove that $P_1 =$ Opt-Path$(u, k, k-1)$ and P_2 is Opt-Path$(k, v, k-1)$. See Figure 1.*

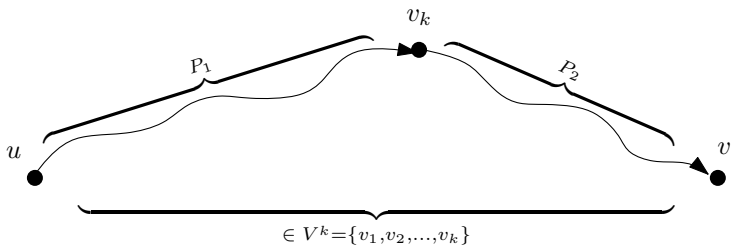

Figure 1: Subpaths

Problem 24. *Show that for $k > 0$,*

$$\textit{Opt-Val}(u, v, k) = \min \begin{cases} \textit{Opt-Val}(u, v, k-1) & \text{if } v_k \notin \textit{Opt-Path}(u, v, k) \\ \textit{Opt-Val}(u, k, k-1) + \textit{Opt-Val}(k, v, k-1) & \text{if } v_k \in \textit{Opt-Path}(u, v, k) \end{cases}$$

Problem 25. *For any u and v describe Opt-Path$(u, v, 0)$ and find Opt-Val$(u, v, 0)$.*

From the above problems, we get the following dynamic programming formulation.

$$\text{Opt-Val}(u,v,k) = \begin{cases} w_{uv} & \text{if } k = 0 \\ \min\left\{\text{Opt-Val}(u,v,k-1)\ ,\ \text{Opt-Val}(u,k,k-1) + \text{Opt-Val}(k,v,k-1)\right\} & \text{if } k \geq 1 \end{cases} \tag{3}$$

Problem 26. *Suppose we are only interesting in finding the distances $d(u,v)$ for all pairs u and v. State our goal in terms of the Opt-Val$(u,v,\cdot)$, i.e. for what value of k should we evaluate Opt-Val(u,v,k).*

Problem 27. *Give an algorithm to compute Opt-Val(u,v,n) for all u and v in a bottom-up fashion.*

You should be able to remove the need of using n matrices, only two will suffice. Also you should be able to construct the actual shortest paths by backtracking from the value of the shortest paths.

Problem 28. *What is the runtime of the above algorithm?*