Design and Analysis of Algorithms (Problem Set 2)

September 9, 2024

1 DFS and Topological Ordering

1. There is a directed acyclic graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges. Each edge has a length which is a positive real number. There is a unique vertex $s \in V$ such that every vertex in G is reachable from s. Let $N(v)$ be the number of the shortest paths from s to v. Design an $O(m + n)$ time algorithm to compute $N(v)$ for all $v \in V$. Assume that the number of such paths are only polynomial in n. (Can you see why this assumption is important ?)

2. We discussed the classification of edges by a DFS forest. Let G be a graph and let (u, v) be an edge. Try to answer the following questions.
   (a) Does there always exist some DFS forest of G such that (u, v) is present as a tree edge in it? If not, what must be the necessary condition that G must satisfy for this to happen?
   (b) Does there always exist some DFS forest of G such that (u, v) is present as a forward edge in it? If not, what must be the necessary condition that G must satisfy for this to happen?
   (c) Does there always exist some DFS forest of G such that (u, v) is present as a backward edge in it? If not, what must be the necessary condition that G must satisfy for this to happen?
   (d) Does there always exist some DFS forest of G such that (u, v) is present as a cross edge in it? If not, what must be the necessary condition that G must satisfy for this to happen?

3. Let $G = (V, E)$ be a directed graph and $u, v \in V$ be any two vertices. Prove or give a counterexample for each of the following statements.
   (a) If u and v are strongly connected, then either u is ancestor of v or v is ancestor of u in every DFS tree.
   (b) If G is a DAG and there is a path from u to v in G,then u is surely going to be an ancestor of v in every DFS tree.
   (c) Suppose u and w are strongly connected whereas u and v are not strongly connected. If (u, v) is an edge, then $F(w) > F(v)$ must hold always.
   (d) We can construct G such that one DFS traversal may produce a DFS tree with degree $n - 1$ whereas another DFS traversal may produce a DFS tree which is just a chain of n vertices.

4. Prove the following statement or give a counterexample: If $S \subseteq V$ is any strongly connected component in a given directed graph $G = (V, E)$, then there must exist a cycle that passes through all the vertices of S.

5. SCC graph is obtained by compressing each strongly connected component of a directed graph into a single vertex. If there are multiple edges between one strongly connected component s to another strongly connected component s', then in the SCC graph we keep only one edge from vertex corresponding to s to the vertex corresponding to s'. It can be observed that the SCC graph of a given graph G is always a directed acyclic graph. Moreover, using the algorithm for computing strongly connected components, we can compute SCC graph in $O(m + n)$ time only.

   You are given a weighted directed graph $G = (V, E)$ that models the road network of a country. Each node corresponds to a city and an edge (u, v) represents a direct road from city u to city v. The label on each road is a positive real number that represents the toll tax that a vehicle has to pay if the vehicle travels along that road. After the election in the country, the new government decided to remove the toll tax for all those roads that connect cities of the same state. You are provided with a list of pairs $\{(u, s(u)) \mid u \in V\}$ such that $s(u)$ is the state to which the city u belongs. Design an $O(m+n)$ time algorithm to compute the shortest path from a given source s to a destination t in this graph.

6. A strongly connected directed graph is said to be Eulerian if there exists a tour which starts from a node, traverses each edge exactly once, and returns to the same node. It is well known that a directed graph is Eulerian if and only if for each vertex the number of incoming edges is the same as the number of outgoing edges. Design an $O(m + n)$ time algorithm to determine if a graph is Eulerian. If the graph is found to be Eulerian, you must print an Euler tour of the graph.

2 Dynamic Programming

1. There are n jobs : $\langle J_1, \ldots, J_n \rangle$ and a single server. Job J_i has a start time s_i and finish time f_i. The server can execute only one job at a time. Unfortunately, there is overlap between the time spans of various jobs, and so we can not execute all the jobs on the server. Each job has also a profit associated with it which is obtained if that job is executed on the server. Let p_i denote the profit associated with job J_i. Give a counter-example to show that the greedy strategy "select the earliest finish time" won't work here. Design a polynomial time algorithm to find a subset of jobs that can be executed on the server such that the profit obtained is maximum.

2. Given a sequence $A = a_1, \ldots, a_n$, a subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is said to be monotonically increasing if $a_{i_j} < a_{i_j} + 1$ for all $1 \leq j < k$. Design an $O(n^2)$ time algorithm to compute the longest monotonically increasing subsequence of sequence A.

3. Let $G = (V, E)$ be a directed graph on n vertices and m edges where each edge has a weight which is a real number. Show that there exists an order among the vertices such that if we process the vertices according to that order in the inner For loop of the Bellman-ford algorithm, then just after one iteration, $D[v]$ will store the distance from s to v.

4. You are given a set of n types of rectangular 3-D boxes, where the ith box has height $h(i)$, width $w(i)$ and depth $d(i)$ (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

5. Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates $a_1, \ldots, a_n$ and n cities on the northern bank with x-coordinates $b_1, \ldots b_n$. Note that there is no order among a_i's. Similarly, you can't assume any order among b_i's. You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. When connecting cities, you can only connect city i on the northern bank to city i on the southern bank. Design a polynomial time algorithm to compute the maximum number of non-crossing bridges that can be built.

6. Given two text strings A of length n and B of length m, you want to transform A into B with a minimum number of operations of the following types: delete a character from A, insert a character into A, or change some character in A into a new character. The minimal number of such operations required to transform A into B is called the edit distance between A and B. Design a polynomial time algorithm to compute edit distance between A and B.

7. There is a sequence of matrices $M_1, M_2, \ldots, M_n$ storing numbers. For each $1 < i \leq n$, the number of rows of M_i is identical to the number of columns of M_{i-1}. So the product $M_1 \times M_2 \times \ldots M_n$ is well defined. We also know that matrix multiplication is associative. That is, $(M_1 \times M_2) \times M_3 = M_1 \times (M_2 \times M_3)$. However, the number of arithmetic operations required may vary in these two possible ways. For example, let M_1 be 10×100, M_2 be 100×5, and M_3 be 5×50.

   - If we multiply according to $((M_1 \times M_2) \times M_3)$, we perform $10 \cdot 100 \cdot 5 = 5000$ multiplication operations to compute the 10×5 matrix $(M_1 \times M_2)$ and then $10 \cdot 5 \cdot 50$ multiplication operations to multiply this matrix with M_3. So a total of 7500 multiplication operations are carried out.
   - If instead we multiply according to $(M_1 \times (M_2 \times M_3))$, we perform $100 \cdot 5 \cdot 50 = 25000$ scalar multiplications to compute 100×50 matrix $(M_2 \times M_3)$ and then another $10 \cdot 100 \cdot 50 = 50000$ multiplication operation to multiply M_1 by this matrix. Hence a total of 75000 multiplication operations are carried out.

   Our aim is to compute $M_1 \times M_2 \times \ldots \times M_n$ using least number of multiplication operations. Design an algorithm based on dynamic programming to solve this problem in polynomial time. Here is some additional explanation which should help you understand the problem better. The order in which we compute $M_1 \times M_2 \times \ldots \times M_n$ is defined by parenthesizing the expression of n matrices. For example, if $n = 4$, then there are the following 4 ways to compute the product:

   - $(((M_1 \times M_2) \times M_3) \times M_4)$
   - $((M_1 \times M_2) \times (M_3 \times M_4))$

- $(M_1 \times ((M_2 \times M_3) \times M_4))$
- $(M_1 \times (M_2 \times (M_3 \times M_4)))$

The number of ways to parenthesize the expression is exponential (it is catalan number). Of course, we can not afford to try out each possible way to multiply the n matrices.

Hint: Consider that parenthesized expression that corresponds to the least number of multiplication operations. If you multiply the matrices according to this parenthesization, the number of matrices will reduce by 1 each time. Finally we will be left with only 2 matrices. How do these last two matrices which are multiplied look like? What can you infer from it? Use it to come up with a recursive formulation.

3 Shortest Paths

1. Let $G = (V, E)$ be a directed graph with real edge weights but no negative cycle. Suppose $L : V \rightarrow R$ be a mapping satisfying the following equalities and inequalities.

   (a) $L(s) = 0$

   (b) For each $v \in V \setminus \{s\}$,

   - $L(v) \leq L(u) + wt(u, v)$ for each edge (u, v)
   - $L(v) = L(u) + wt(u, v)$ for at least one edge (u, v).

   Prove that $L(v) = \delta(s, v)$, i.e., $L(v)$ stores the distance from s to each vertex v in the graph.

2. Recall from the lecture on all-pairs shortest paths that we could reduce the space required by overwriting on the same distance matrix. Can we do it for Bellman-Ford algorithm as well? In other words, consider the following algorithm.

```
$L(s) \leftarrow 0$;
for $v \in V \setminus \{s\}$ do
    $L(v) \leftarrow \infty$;
end
for $i = 1$ to $n - 1$ do
    foreach $(u, v) \in E$ do
        if $L(u) + wt(u, v) < L(v)$ then
            $L(v) \leftarrow L(u) + wt(u, v)$;
        end
    end
end
```

   Does $L(v)$ store distance from s to v in the graph at the end of the above algorithm? Prove or give a counterexample.

3. Recall the Bellman-Ford algorithm discussed in the class. Augment the algorithm suitably to compute a data structure of $O(n)$ size using which we can report the shortest path from source s to any vertex v efficiently (in time of the order of the number of edges in the shortest path from s to v).

4. Consider the problem of shortest paths in directed graph with real weights but no negative cycle. An attentive student argues that there is no need of Bellman-Ford algorithm. We could just transform the given graph with some negative edges into a graph where all edge weights are positive and then apply Dijkstra's algorithm. The transformation is the following:
   Let w be the weight of the least weight negative edge in the given graph. Add $|w|$ to the weight of each edge. Let G' be the resulting graph.

   The student claims that shortest path from s to v in G is identical to the shortest path from s to v in G' for each vertex v. Do you agree with the student? Either provide a proof or provide a counterexample.

5. Which of the following statements is (are) correct?

   (a) For every directed graph, Dijkstra's algorithm will fail to compute distance from a given source if there is any negative edge present in the graph.

   (b) There exists a directed graph with some negative edge weights such that Dijkstra's algorithm will fail to compute the distance from a given source in the graph.

   (c) There exists a directed graph with some negative edge weights on which Bellman-Ford algorithm may fail to compute distance from a given source vertex.

6. Let $G = (V, E)$ be a directed graph with real edge weights but no negative cycle. Let $s \in V$ be a designated source vertex in the graph. You are given an array A storing some real numbers. We need to verify whether $A[i]$ is the distance from s to i for each $i \in V$ in the graph. Design the most efficient algorithm for this task.

7. Refer to Problem 4 in this section. Though it was a failed attempt to make the edge weights non-negative while preserving the shortest paths from the source, one still feels that there might be an alternate way to make edge weights non-negative and still preserving the shortest paths from the source. Indeed, there is such a way. In fact, it leads to an $O(mn + n^2 \log n)$ time algorithm for all-pairs shortest paths in directed graphs with real edge weights but no negative cycle. Notice that this is much better than Floyd Warshal algorithm. The following is the sketch of this brilliant algorithm. Fill in the details suitably. Without loss of generality, assume that the graph is strongly connected.

   1. Pick any arbitrary vertex s. Execute Bellman-Ford algorithm to compute distance from s.
   2. Using the distance from s, transform the edge weights of the graph so that each edge weight is non-negative. Most importantly, the shortest path in the original graph are still the shortest paths in the transformed graph.
   3. Execute Dijkstra's algorithm from each $v \in V \setminus \{s\}$.