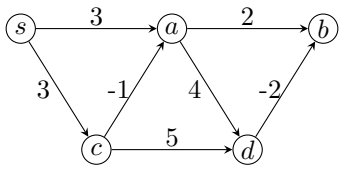
**Problem 1 (30 points).**




1. Run the dynamic programming algorithm for the single-source shortest path problem on the following graph with $s$ being the source vertex: write and fill the dynamic programming table *dist*.

   **Solution.** Suppose that $s = v_1, a = v_2, b = v_3, c = v_4, d = v_5$, which corresponds to the 5 columns of the table below.

$$dist = \begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ 0 & 3 & \infty & 3 & \infty \\ 0 & 2 & 5 & 3 & 7 \\ 0 & 2 & 4 & 3 & 6 \\ 0 & 2 & 4 & 3 & 6 \end{bmatrix}$$

2. Run the Bellman-Ford algorithm on the following graph with $s$ being the source vertex: give the value of array *dist* after each iteration.

   **Solution.** Suppose that $s = v_1, a = v_2, b = v_3, c = v_4, d = v_5$, which corresponds to the 5 elements of the *dist* array below. In every iteration, we assume that all edges are relaxed in the order of $(s,a),(s,c),(a,b),(a,d),(c,a),(c,d),(d,b)$.

   initial array: $dist = (0,\infty,\infty,\infty,\infty)$.
   after 1 iteration: $dist = (0,2,5,3,7)$.
   after 2 iteration: $dist = (0,2,4,3,6)$.
   after 3 iteration: $dist = (0,2,4,3,6)$.
   after 4 iteration: $dist = (0,2,4,3,6)$.

3. Run the Floyd-Warshall algorithm on the following graph: write the matrix corresponding to $dist(\cdot,\cdot,0)$ and $dist(\cdot,\cdot,1)$.

   **Solution.** Suppose that $s = v_1, a = v_2, b = v_3, c = v_4, d = v_5$, which corresponds to the 5 rows and 5 columns of the matrices below.

$$dist(\cdot,\cdot,0) = \begin{bmatrix} 0 & 3 & \infty & 3 & \infty \\ \infty & 0 & 2 & \infty & 4 \\ \infty & \infty & 0 & \infty & \infty \\ \infty & -1 & \infty & 0 & 5 \\ \infty & \infty & -2 & \infty & 0 \end{bmatrix}$$

   As we assume that $s = v_1$, we actually have $d(\cdot,\cdot,1) = d(\cdot,\cdot,0)$. (Notice that if we assume different indices for vertices, we may have different answer, as $k$, the 3rd parameter of $dist(\cdot,\cdot,\cdot)$, indicates the largest *index* among intermediate vertices, which therefore depends on how we assign indices for vertices.)

**Problem 2 (20 points).** In the Bellman-Ford algorithm (and the dynamic programming algorithm), we can maintain an array *prev* to store the previous vertex in the shortest path for each vertex (as we did in the Dijkstra's algorithm), so that we can reconstruct the shortest path (in addition to the distance) for each vertex. Specifically, when we relax edge $(u,v)$ we set $prev[v]$ to $u$ if we have $dist[u] + l(u,v) < dist[v]$.

1. Prove that edges $\{(v, prev[v]) \mid v \in V\}$ form a tree.

   **Solution.** NOTE: this conclusion is required; its proof given below is not required. Let $\delta(s,v)$ be the actual shortest distance from vertex $s$ to $v$ for vertex $v \in V$. Let total number of call to *RELAX* be $R$. Let $V_r^{reachable}$ be the vertex set $\{v \in V : dist[v] = \delta(s,v) < \infty\}$ after $r$-th relaxation (where $r \in \{0,1,2,\ldots,R\}$). In other words, at any point of the algorithm, $V^{reachable}$ is the set of vertices whose path from source $s$ has been found.

   We prove by induction that, after $r$ relaxations, the edges $\{(v, prev[v]) \mid v \in V_r^{reachable}\}$ form a tree.

   *Basis step:* When the algorithm has just begun, $V_0^{reachable} = \{s\}$. Since a single vertex is a tree, and $prev[s] = \infty$, the basis case is proved.

   *Induction step:* By the correctness proof of the algorithm, we know that at some point, the relaxation on edge $(u,v)$ decreases $dist[v]$ to $dist[u] + l(u,v) = \delta(s,v)$. Let this be the $r$-th relaxation. Just before this relaxation takes place, assume that we get a shortest path tree $T_{r-1}$ formed by edges $\{(v, prev[v]) \mid v \in V_{r-1}^{reachable}\}$.

   We note that, if a relaxation on edge $(u,v)$ decreases $dist[v]$ to $dist[u] + l(u,v) = \delta(s,v)$, at that moment $dist[u]$ already equals $\delta(s,u)$. Therefore $u$ is in *shortest* path tree $T_{r-1}$. As a result of the relaxation of $(u,v)$, the new vertex $v$ and the leaf edge from $u$ to $v$ are added to $T_{r-1}$, keeping it to be a tree rooted at $s$. The path to $v$ now costs $dist[u] + l(u,v) = \delta(s,u) + l(u,v) = \delta(s,v)$. Hence, the new tree $T_r$ is a shortest path tree from $s$ formed by edges $\{(v, prev[v]) \mid v \in V_r^{reachable}\}$.

   Because of the condition $dist[u] + l(u,v) < dist[v]$ in relaxation step, values of $dist$ may only decrease. Therefore after $dist[v]$ reaches $\delta(s,v)$, neither $dist[v]$ nor $prev[v]$ change. After that $V^{reachable}$ may only grow as more relaxations take place and eventually equal $V$ (provided that every vertex is reachable from source).

2. Compute $prev[v]$ for every vertex $v \in V$ in the graph used in Problem 1.

   **Solution.** Here's the final *prev* array.

| Vertex | $s$ | $a$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|---|
| $prev[v]$ | $\infty$ | $c$ | $a$ or $d$ | $s$ | $a$ |

**Problem 3 (10 points).** Given directed graph $G = (V,E)$, edge length $l(e)$ for $e \in E$, vertex $s \in V$, the dynamic programming algorithm (and the Bellman-Ford algorithm) can be used to determine whether there exists negative cycles in $G$ that can be reached from $s$. Design a new algorithm for the following problem: given directed graph $G = (V,E)$ and edge length $l(e)$ for $e \in E$ to decide whether $G$ contains negative cycles.

**Solution.** We modify $G$ to create a new graph $G'$. For each vertex $v \in V \setminus \{s\}$, if there is no edge from $s$ to $v$ in $G$, i.e., $(s,v) \notin E$, we add a new edge $(s,v)$ with edge length of $M$, where $M$ is very big number, can be set as $-\sum_{e \in E: l(e) < 0} l(e) + 1$ (the purpose of doing so is not to introduce negative cycles). After doing so, all vertices can be (directly) reached from $s$ in $G'$.

We have that $G'$ contains negative cycles if and only $G$ contains negative cycles. In fact, if $G$ contains negative cycle then $G'$ contains cycles, as $G$ is a subgraph of $G'$. To prove the other side, suppose conversely that $G'$ contains negative cycles but $G$ does not. Let $C$ be one negative cycle in $G'$. We have that $C$ must contain new edges, otherwise $G$ will contain $C$. Since the length of each new edge is large enough to dominate the sum of all negative edges, $C$ cannot be a negative cycle—a contradiction.

We can run Bellman-Ford algorithm to decide whether $G'$ contains negative cycle that can be reached from $s$. Since in $G'$ all vertices can be reached from $s$, this decides whether $G'$ contains negative cycles. Based on the above analysis, this also answers whether $G$ contains negative cycles. The running time is $O(|V|(|E| + |V|))$, as $G'$ has at most $(|E| + |V|)$ edges.

**Problem 4 (20 points).** Given a directed acyclic graph $G = (V,E)$, vertex $s \in V$, design a dynamic programming algorithm to compute the number of distinct paths from $s$ to $v$ for any $v \in V$. You should define subproblems, write recursion, give the pseudo-code, and analyze the running time.

**Solution:**
*Subproblems:* Let $C(v)$ be the number of distinct paths from $s$ to $v$.

*Recursion:* The idea is that, if $C(u) = k$, i.e., vertex $u$ can be reached from $s$ in $k$ distinct paths, and there is an edge $(u,v)$, then $v$ can be reached from $s$ by taking one of these $k$ distinct paths followed by edge $(u,v)$. Combining all in-edges of $v$ gives all possible paths. Therefore, the recursion is as follows: $C(v) = \sum_{u:(u,v)\in E} C(u)$.

In order to guarantee that when we process $v$, all vertices in $\{u \mid (u,v) \in E\}$ have been processed, we need to traverse vertices in the order of linearization. Therefore, the first step of the algorithm is to compute a linearization using DFS. The *pseudo-code*, which includes three steps, is as follows.

*Initialization:*

compute a linearization of $G$
$C(v) = 0$, for any $v \in V$
$C(s) = 1$

*Iteration:*

for each $v$ in the order of above linearization
  $C(v) = \sum_{u:(u,v)\in E} C(u)$
endfor

*Termination:* $C(v)$ gives the number of distinct paths from $s$ to $v$.

*Running time:* the linearization step takes $O(|V| + |E|)$ time. The initialization of $C(v)$ takes $O(|V|)$ time. The iteration step essentially traverses each edge at most once, and therefore it takes $O(|V| + |E|)$ time. The total running time of the algorithm is $O(|V| + |E|)$.

**Problem 5 (20 points).** Shortest path algorithms can be applied in currency trading. Let $c_1, c_2, \cdots, c_n$ be various currencies. For any two currencies $c_i$ and $c_j$, there is an exchange rate $r_{i,j}$; this means that you can purchase $r_{i,j}$ units of currency $c_j$ in exchange for one unit of $c_i$. These exchange rates satisfy the condition that $r_{i,j} \cdot r_{j,i} < 1$, so that if you start with a unit of currency $c_i$, change it into currency $c_j$ and then convert back to currency $c_i$, you end up with less than one unit of currency $c_i$ (the difference is the cost of the transaction).

1. Give an efficient algorithm for the following problem: given a set of exchange rates $r_{i,j}$, and two currencies $s$ and $t$, find the most advantageous sequence of currency exchanges for converting currency $s$ into currency $t$. Toward this goal, you should represent the currencies and rates by a graph whose edge lengths are real numbers.

   **Solution.** We can transform the currency exchange problem into the shortest path problem. We build a directed graph $G = (V,E)$, where $V = \{c_1, c_2, \cdots, c_n\}$ represents all currencies, and $E$ contains all pairs $(c_i, c_j)$, $1 \le i \ne j \le n$. We assign length for edge $(c_i, c_j) \in E$ as $-\log(r_{i,j})$. We now show that computing the shortest path from $s$ to $t$ in $G$ actually gives the optimal sequence of currency exchanges for converting $s$ into $t$. In fact, there is one-to-one correspondence between a path from $s$ to $t$ in $G$ and a sequence of currency exchange for converting $s$ into $t$. Moreover, the length of such a path $p$ equals to $\sum_{(c_i,c_j)\in p} -\log r_{i,j} = -\log \prod_{(c_i,c_j)\in p} r_{i,j}$. Hence, the shortest path in $G$ gives the path maximizes $\prod_{(c_i,c_j)\in p} r_{i,j}$, which is exactly the maximized amount of currency $t$ that can be coverted from a unit of currency $s$.

   Based on the above analysis, the algorithm will be to compute the shorest path from $s$ to $t$ in $G$

(using any of the single-source shortest path algorithm we introduced). The vertices along this optimal path gives the sequence of currencies following which we can get the maximized amount of currency $t$.

The exchange rates are updated frequently. Occasionally the exchange rates satisfy the following property: there is a sequence of currencies $c_1, c_2, \cdots, c_k$ such that $r_{1,2} \cdot r_{2,3} \cdot r_{3,4} \cdots r_{k-1,k} \cdot r_{k,1} > 1$. This means that by starting with a unit of currency $c_1$ and then successively converting it to currencies $c_2$, $c_3$, $\cdots$, $c_k$, and finally back to $c_1$, you would end up with more than one unit of currency $c_1$.

2. Give an efficient algorithm for detecting the presence of such an anomaly. Use the graph representation you found above.

   **Solution.** We need to detect whether there is $r_{1,2} \cdot r_{2,3} \cdot r_{3,4} \cdots r_{k-1,k} \cdot r_{k,1} > 1$, which is equivalent to detect $-(\log(r_{1,2}) + \log(r_{2,3}) + \log(r_{3,4}) + \cdots + \log(r_{k-1,k}) + \log(r_{k,1})) < 0$. Since we assign length for edge $e = (c_i, c_j)$ as $-\log(r_{i,j})$, this exactly implies a negative cycle in $G$. In other words, such an anomaly exists if and only if $G$ contains negative cycles. Therefore, we can use Bellman-Ford algorithm on $G$ to identify whether $G$ contains negative cycles, which also answers whether there exists anamaly.