# Workspace: Blank template

*1.* The straightforward way to do this is to run a BFS starting at every node. Then, take the farthest vertex pairs in each iteration and compare:

- Initialize set of farthest distances $S \leftarrow \emptyset$.
- For every vertex $u$, run BFS with $u$ as a root
- Let $d$ be the depth of this BFS tree, $S \leftarrow S \cup \{d\}$.
- Return the largest element of $S$.

Running time is to do BFS, and find the farthest vertex $|V|$ times. In total, $O(|V| \cdot (|V| + |E|))$ □

*2.a.* One direction is obvious: suppose that $(u, v)$ is a tree or forward edge. In this case, it is easy to verify the inequalities.

For the other direction, since we have that $d[u] < d[v]$, $u$ was discovered before $v$, so it is clearly not a back edge. Also, since we have $f(v) < f(u)$, we know that it cant be on a tree that was explored after exploring $u$. So, This rules out it being a cross edge. Leaving us with the only possibilities of being a tree edge or forward edge. □

*2.b.* Again, one direction is obvious: If $(u, v)$ is a back edge, then it is easy to verify the inequalities.

For the other direction, we use the previous part to reason that if $(u, v)$ was a tree or a forward edge, then the previous inequalities would hold. On the other hand, if it was a cross edge, then $v$ would have been finished before discovering $u$

□

*2.c.* Again, one direction is trivial. If $(u, v)$ is a cross edge, then it is easy to verify the inequalities.

However, now even the other direction is trivial because of the previous two parts. If $(u, v)$ was a tree, forward, or cross edge, then the inequalities of the previous section would hold. Therefore, $(u, v)$ can only be a cross edge. □

*3.* The algorithm is given as follows:

- Run DFS from any vertex
- If there are forward or cross edges, return FALSE, else return TRUE

Clearly, a forward or cross edge corresponds to an additional path (in addition to the DFS path). Moreover, two paths between a pair of vertices must always result in a cross edge. Running time is the same as that of DFS $O(|V| + |E|)$. □

*4.* Make a graph as follows. For every currency $c_i$, make a vertex. For the transfer rate between $c_i$ and $c_j$, put a directed edge of weight $-\log R[i,j]$. Clearly, if there is a sequence $i_1, i_2, \ldots, i_t$ such that $R[i_1, i_2] \times \cdots \times R[i_{t-i}, i_t] > 1$, then the sum of these edge weights $-\log R[i_1, i_2] - \cdots - \log R[i_{t-1}, i_t] < 0$, hence this is just a problem of detecting negative cycles which can be done by running Bellman-Ford algorithm.

Using Bellman-Ford algorithm to detect negative cycles

- Run Bellman-Ford algorithm on the graph $G$. Let $d(u,v)$ denote the shortest path between $u$ and $v$ returned by the algorithm
- If any $d(u,u) < 0$, return TRUE. Else, return false.

Running time is the same as that of running the Bellman-Ford algorithm $O(|V|^3)$. □

*5.* Straightforward approach:

- Run the Floyd-Warshall algorithm.
- If there is a directed path from a vertex $u$ to a vertex $v$ in the graph, then $d(u,v) < \infty$ is returned by the algorithm.

The above algorithm takes time $O(|V|^3)$. This can be improved by making a slight improvement (you *MIGHT* remember this one from Disc. Structures I).

Let $A$ denote the adjacency matrix of the graph, i.e., $A_{ij} = 1$ iff $(i,j) \in E$ and $A_{ij} = 0$ otherwise. The following claim is useful:

**Claim 1.** *If there is a path of length $\leq t$ from a vertex $i$ to a vertex $j$, then $(A^t)_{ij} > 0$ and $0$ otherwise..*

*Proof.* We prove this by induction on $t$. Clearly, it is true for $t = 1$ by the definition of $A$. Assume it is true for $t-1$. If there is a path of length $t$ from a vertex $i$ to a vertex $j$, then there must be a verted $k$ such that there is a $t-1$ length path form $i$ to $k$ and a 1 length path from $k$ to $j$. So, $(A^{t-1})_{ik} > 0$, and $A_{kj} > 0$. One can verify that this is enough to show that $(A^t)_{ij} > 0$. Similarly, if there is no path of length $\leq t$ from $i$ to $j$, then for every $k \in V$, either there is no $\leq t-1$ path from $i$ to $k$ or no 1 length path from $k$ to $j$. This means, by induction, either $(A^{t-1})_{ik} = 0$ or $(A)_{kj} = 0$. Again, one can verify that this means $(A^t)_{ij} = 0$. □

The above observation gives us yet another algorithm: If there is a path from $i$ to $j$, then it must be of length at most $|V|$. Compute $A^{|V|}$, and add $(i,j) \in E^*$ iff $(A^n)_{ij} > 0$.

How do we compute $A^{|V|}$ efficiently? Simple: repeated squaring. Compute $A$, $A^2$, $A^4$, and so on for $\log|V|$ steps. Each step takes $|V|^{2.7\cdots}$ (as seen in $\sim$ lec 5.). So the total running time is $|V|^{2.7\cdot\cdot} \log|V|$ which is better than the naive $|V|^3$.

□