Problem 1.

(a) Prove or disprove the following statement: Let $G$ be a flow network, with source $s$, sink $t$, and suppose that all edges have unit capacity. Let $k$ be the value of a maximum flow in $G$. Then, there exists a collection of $k$ pairwise edge-disjoint paths $P_1, \ldots, P_k$ from $s$ to $t$ in $G$. That is, for any $i \neq j \in \{1, \ldots, k\}$, there is no edge in $G$ that is traversed by both $P_i$ and $P_j$.

(b) Prove or disprove the following statement: Let $G$ be a flow network, with source $s$, sink $t$, and suppose that all edges have unit capacity. Let $k$ be the value of a maximum flow in $G$. Then, there exists a collection of $k$ paths $P_1, \ldots, P_k$ from $s$ to $t$ in $G$, such that any two distinct paths have only $s$ and $t$ as common vertices. That is, for any $i \neq j \in \{1, \ldots, k\}$, there is no vertex in $G$, other that $s$ and $t$, that is visited by both $P_i$ and $P_j$.

Problem 2. Let $G = (V, E)$ be a directed graph, and let $s, t \in V$ be distinct vertices. Give a polynomial-time algorithm that computes a maximum-cardinality collection of pairwise vertex-disjoint paths $P_1, \ldots, P_k$ from $s$ to $t$ in $G$.

Problem 3: Vijay’s shortest path algorithm. Let $G$ be a weighted directed graph, with no negative cycles (but possibly with negative edges). Consider the following algorithm for computing single-source shortest paths in $G$ from a starting vertex $s$.

```plaintext
procedure Main
    let $Q$ be a FIFO queue
    add $s$ to $Q$
    while $Q$ is nonempty
        extract the next node $v$ from $Q$
        ExploreNode($v$)

procedure ExploreNode($v$)
    for each node $u$ adjacent to $v$
        if relax($v, u$) reduces $u.d$
            add $u$ to $Q$
```

Notice that the above algorithm is somewhat similar to Dijkstra’s, but it uses a FIFO queue, instead of a min-heap. That is, at every iteration it extracts the node that was inserted in $Q$ first, instead of the node with a minimum $d$ value.

(a) What is the worst-case running of this algorithm?

(b) What is the worst-case running of this algorithm, assuming that there are no edges with negative weight?