Let (iv) the parity? Prove the following facts:

The above property shows that if we can recursively compute the distances in $H$ be the time to multiply two $n \times n$ matrices; we may assume that $M(n) \geq \Omega(n^2)$. You will now develop a recursive algorithm for our problem that runs in in $O(M(n) \log n)$ time, where $n = |V|$ as usual.

The square $H$ of $G$, denoted by $H = G^2$, is the graph on the same vertex set $V$ in which there is an edge $uv$ if $uw$ is an edge in $G$ or if there is some 2-hop path (say, $u - w - v$) in $G$. You can assume that $G$ is stored in, say, adjacency-matrix format.

1. Problem 5.20, DPV.
2. Problem 5.21, DPV.
3. Consider the all-pairs-shortest path problem (APSP) in a given undirected, unweighted\footnote{“Unweighted” means that every edge has the same length (say, 1)} graph $G = (V, E)$; suppose we are only interested in the shortest-path distances $d_G(u, v)$ for all vertex-pairs $(u, v)$. Let $M(n)$ be the time to multiply two $n \times n$ matrices; we may assume that $M(n) \geq \Omega(n^2)$. You will now develop a recursive algorithm for our problem that runs in in $O(M(n) \log n)$ time, where $n = |V|$ as usual.

The square $H$ of $G$, denoted by $H = G^2$, is the graph on the same vertex set $V$ in which there is an edge $uv$ if $uw$ is an edge in $G$ or if there is some 2-hop path (say, $u - w - v$) in $G$. You can assume that $G$ is stored in, say, adjacency-matrix format.

(i) Show how to compute $H$ from $G$ in time $O(M(n))$.

(ii) If $G$ is connected, show that repeatedly squaring it $\lceil \log n \rceil$ times results in a complete graph.

We wish to understand how to obtain information for distances in $G$ from distances in $H$. Prove the following:

(iii) Let $u, v \in V$. If $d_G(u, v)$ is even, then $d_G(u, v) = 2d_H(u, v)$; if $d_G(u, v)$ is odd, then $d_G(u, v) = 2d_H(u, v) - 1$.

The above property shows that if we can recursively compute the distances in $H$ then we would be able to obtain the distances in $G$ approximately to within an additive error of 1. In particular if we knew the parity of the distances in $G$ then we would be able to recover the distances in $G$ from those in $H$. How do we obtain the parity? Prove the following facts:

(iv) Let $u, v$ be distinct nodes in $G$. Then for every neighbor $w$ of $v$ in $G$ we have $d_G(u, v) - 1 \leq d_G(u, w) \leq d_G(u, v) + 1$. Moreover, there is at least one neighbor $w$ of $v$ such that $d_G(u, w) = d_G(u, v) - 1$.

(v) Let $u, v$ be distinct nodes in $G$. If $d_G(u, v)$ is even, then for every neighbor $w$ of $v$ in $G$ we have $d_H(u, w) \geq d_H(u, v)$; if $d_G(u, v)$ is odd, then for every neighbor $w$ of $v$ in $G$, $d_H(u, w) \leq d_H(u, v)$ and there is some neighbor $w$ of $v$ in $G$ for which $d_H(u, w) < d_H(u, v)$.

(vi) Let $u, v$ be distinct nodes in $G$. Then $d_G(u, v)$ is even if and only if

$$\sum_{w : w \text{ is a neighbor of } v \text{ in } G} d_H(u, w) \geq d_H(u, v) \cdot \deg_G(v),$$

where $\deg_G(v)$ is the degree of $v$ in $G$.

With the above in place:

(vii) Given the APSP distances for $H$ in a matrix $A$ and the adjacency matrix of $G$ in $B$, reduce the problem of finding the parity of $d_G(u, v)$ for all pairs $(u, v)$, to matrix multiplication.
(viii) Solve the APSP distances problem on $G$ in $O(M(n) \log n)$ time.

4. You are given a set of $n$ jobs that need to be scheduled on a single machine. Each job $j$ has an integer processing time $p_j > 0$ and an integer release time $r_j \geq 0$ when it becomes available. Job $j$ cannot be scheduled before time $r_j$. The objective is to schedule the jobs to minimize the total completion time of the jobs; in other words, to minimize $\sum_{j=1}^{n} C_j$ where $C_j$ is the completion time of job $j$. At any time, only one job can be processed on the machine; however, the schedule is allowed to be pre-emptive — a job that is currently being processed can be set aside to process a different job. A job $j$ is completed when it receives a total of $p_j$ units of processing. Consider the following simple algorithm: “at any time $t$, schedule the job that has the least amount of time still left to process”.

(a) Is this an optimal algorithm? If so, prove its correctness.

(b) Give a polynomial time algorithm that outputs a schedule for the jobs. The schedule should be specify for each job the time intervals (the start and end points) during which the job is processed on the machine. Note that an algorithm that runs in $O(\sum p_j)$ time is not a polynomial time algorithm.

5. Let $T = (V, E)$ be a tree and let $S \subseteq V$ be a set of special vertices called terminals. For any given subset of the edges $X \subseteq E$, let $h(X)$ be the number of connected components in the forest $T \setminus X$ that contain at least one terminal from $S$. Define a tuple $(E, I)$ where $I$ is a family of subsets of $E$ defined as:

$$I = \{ X \subseteq E \mid h(X) = |X| + 1 \}$$

(i) Prove that $M_{T, S} = (E, I)$ is a matroid. A loop in a matroid $M$ is any element $x$ of $M$’s ground set that also happens to be a circuit (i.e., $\{x\}$ is not an independent set of $M$). When is an edge $e \in E$ a loop in the matroid $M_{T, S}$?

(ii) For any matroid $M = (N, I)$ and integer $k$ define $M_k = (N, I')$ where $I' = \{ I \in I \mid |I| \leq k \}$. Show that $M_k$ is also a matroid.

(iii) Derive an efficient algorithm for the following problem. Given a tree $T = (V, E)$ with non-negative edges costs, a set of terminals $S \subseteq V$, and an integer $k \leq |S|$, find the smallest cost set of edges in $T$ whose removal results in $k$ components each of which contains at least one terminal.

6. Consider a rooted tree $T$ in which every non-leaf node has exactly 3 children; thus, letting $h$ denote the depth of $T$, $T$ has $n = 3^h$ leaves. Each leaf $\ell$ has been given some value $x(\ell) \in \{0, 1\}$. We recursively define a value $y(u)$ for each node $u$: if $u$ is a leaf, then $y(u) = x(u)$; otherwise, letting $v_1, v_2, v_3$ denote the children of $u$, $y(u)$ is defined to be the majority element among $y(v_1)$, $y(v_2)$, and $y(v_3)$. (The majority element among three binary values $z_1, z_2, z_3$ is the element that appears at least twice among them; it is easy to verify that $y(u) \in \{0, 1\}$ for all $u$.) Our problem is, given the leaf-values $x$, to determine the value $y(r)$ for the root $r$ of the tree.

(i) Show that for any deterministic algorithm $\mathcal{A}$ for our problem, there is an instance of the problem for which $\mathcal{A}$ is forced to examine all $n$ leaves.

(ii) Suggest a natural randomized algorithm for this problem, and show that on any input $x(\cdot)$, the expected number of leaves that it inspects is at most $n^{0.9}$. 

2