Problem 1
(By Chandra.)

Let $G = (V, E)$ be an undirected simple graph. Let $T \subseteq V$ be a set of terminals; we refer to $V \setminus T$ as non-terminals. For $u, v \in T$ we define the element connectivity $\kappa'(u, v)$ as the maximum number of $u - v$ paths that are disjoint in the non-terminals and edges of $G$; note that the paths can share terminals. See figure for an example.

(A) Given $G = (V, E)$, $T \subseteq V$ and $u, v \in T$, describe an efficient algorithm to compute $\kappa'(u, v)$.

(B) Prove that for $u, v, w \in T$, $\kappa'(u, v) \geq \min\{\kappa'(u, w), \kappa'(v, w)\}$. 
Figure 1: In the graph above $\kappa'(u,v) = 3$ and $\kappa'(u,w) = \kappa'(v,w) = 2$. 

Problem 2
(By Chandra.)

(A) For a $n \times n$ matrix $M$, its determinant is the quantity

$$\det(M) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} M_{i,\sigma(i)},$$

where $S_n$ is the set of all permutations of $\{1, \ldots, n\}$, and $\text{sgn}(\sigma)$ is +1 if $\sigma$ is even and −1 otherwise.\(^1\)

Let $M$ be such a matrix of integer numbers, that can be represented using $U$ bits. Prove that the number of bits required to represent $\det(M)$ is bounded by a polynomial in $U$.

(B) Let $M$ be a $n \times n$ matrix of rational numbers, that can be represented using $U$ bits. Prove that the number of bits required to represent $\det(M)$ is bounded by a polynomial in $U$. (You can assume (A) in proving this part.)

(C) The Linear-Programming problem is to solve a system of the form

$$\max \ c^t x, \quad Ax \leq b,$$

where $A$ is a $m \times n$ rational matrix and $c$ and $b$ are $n \times 1$ rational vectors.

In the decision version we are given an additional rational number $K$ and the given instance $A, c, b, K$ is a YES instance if there is a real-valued $n \times 1$ vector $x^*$ such that $c^t x^* \geq K$ and $Ax^* \leq b$. It is NO instance otherwise (either if there is no feasible solution for $Ax \leq b$ or if the maximum value is strictly less than $K$).

Prove that the decision version of Linear-Programming is in NP. Claim any facts from linear algebra that you find useful or need (in particular, you can assume (A) and (B))

\(^1\)A permutation is even if it can be converted in a even number of switches to the identity permutation.
in your solution). You cannot use the fact that there is a polynomial-time algorithm for Linear-Programming. Note also that numerical operations on integers require time that depends on the length of their binary representation (for example, adding two integers with \( m \) and \( n \) bits takes \( O(m + n) \) time).

Problem 3
(By Sariel.)

(A) [20 Points] Prove, by induction, the Cauchy-Schwarz inequality, that is, for any real numbers \( x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n \), we have that

\[
\sum_{i=1}^{n} x_i y_i \leq \sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}.
\]

(Note, that you need to provide here a carefully written formal proof.)

(B) [5 Points] Let \( G = (V, E) \) be a graph over \( n \) vertices, with the property that \( \sum_{v \in V} (d(v)) \leq \binom{n}{2} \). Prove that \( |E| = O(n^{3/2}) \).

Problem 4
(By Jeff.)

Let \( A[1..n] \) be a fixed array of integers between 1 and \( n \). For any other array \( I[1..k] \) of integers between 1 and \( n \), let \( A(I) \) denote the \( k \)-element array whose \( j \)th entry is \( A[I[j]] \), for all \( j \).

Recall that an array \( I[1..k] \) is increasing if \( i < j \) implies \( I[i] < I[j] \). If the arrays \( I \) and \( A(I) \) are both increasing, then \( I \) is the sequence of indices of an increasing subsequence of \( A \); in this case, we say that \( I \) is 2-increasing. If the arrays \( I \), \( A(I) \), and \( A(A(I)) \) are all increasing, then we say that \( I \) is 3-increasing. More generally, for any positive integer \( r \), we say that \( I \) is \( r \)-increasing if \( A(I) \) is \( (r - 1) \)-increasing. Finally, we say that \( I \) is \( \infty \)-increasing if \( I \) is \( r \)-increasing for every positive integer \( r \).

For example, if \( A = [3, 1, 4, 1, 5, 9, 2, 6, 7, 3] \), then the index array \( I = [3, 5, 6] \) is 3-increasing:

\[
I = [3, 5, 6]^\checkmark \quad A(I) = [4, 5, 9]^\checkmark \quad A(A(I)) = [1, 5, 7]^\checkmark \quad A(A(A(I))) = [3, 5, 2]^\checkmark
\]

The robustness of \( I \) is the largest integer \( r \) such that \( I \) is \( r \)-increasing, or \( \infty \) if \( I \) is \( \infty \)-increasing. A non-increasing array has robustness 0.

(A) Warmup: Given an array \( A[1..n] \), sketch an efficient algorithm to find the longest 2-increasing subsequence of \( A \).
(B) Find the smallest function $f(n)$ with the following property: For all arrays $A[1..n]$ and $I[1..k]$ with $k \leq n$, if $I$ is $f(n)$-increasing, then $I$ is $\infty$-increasing. (We are looking for a statement of the form: If an array $I$ is $2^n$-increasing then it is $\infty$-increasing. Naturally, you are looking for $f(n)$ which is as small as possible.) (Hint: Consider the special case $k = 2$.)

(C) Given two arrays $A[1..n]$ and $I[1..k]$, sketch an efficient algorithm to determine the robustness of $I$ with respect to $A$.

(D) Given an array $A[1..n]$, sketch an efficient algorithm to find the longest $\infty$-increasing subsequence of $A$. 