Algorithm Design and Analysis
Assignment 3
Wu Jiabao

1. (a) Suppose we have two maximal independent sets $A,\ B$ with $|A| < |B|$.
   There exists $x \in B \setminus A$, $\{x\} \cup A \in \mathcal{I}$. Since $A \subsetneq \{x\} \cup A$, this is a contradiction.
   Thus $|A| = |B|$.

   (b) **(hereditary property)**. Obviously S is nonempty. Since $\forall F \in S$, take $F' \subseteq F$, we have F' does not contain a cycle, thus $F' \in S$.
   (exchange property). Suppose $|A| < |B|$. If $x_0 \in B \setminus A$ is connected to a point that A doesn't reach, take $\{x_0\} \cup A$ and it must not have a cycle.
   If we cannot find such an x_0 above, we have A is connected to the same points as B. Since $|A| < |B|$, B contains at least one more edge than A. This condition only happens when A is not a single tree. Since A contains at least two trees, the project edge x_0 must connect two trees in A. (otherwise there must have a cycle in B)
   Hence $\{x_0\} \cup A \in \mathcal{I}$, M is a matroid.
   All the spanning trees without vertices are the maximal sets.

   (c) we know $\{x\} \in \mathcal{I}$. Assume a maximal set S_0 with maximal weight not containing x and $S' = \{x\}$.
   If $|S_0| = 1$, obviously $w(x)$ is the optimal. Contradiction!
   If $|S_0| > 1$, then from exchange property we know there exists $x_0 \in S_0$ that $S' = S' \cup \{x_0\} \in \mathcal{I}$. If $|S'| < |S_0|$, repeate the process till S' is maximal. Now since $|S_0| = |S'|$, let $\{y\} = S_0 \setminus S'$. Obviously $w(S') - w(S_0) = w(x) - w(y) > 0$, hence S_0 is not the optimal set. Contradiction!
   If there exist $w(y) = w(x)$, S_0 and S' are both optimal sets.
   Therefore, there must exist a maximal set with maximal weight containing x.

   (d) We proof it by induction. Let S be the subset of an optimal set.
   Assume after adding the (i-1)-th element, S is the subset of an optimal set. The first step is proved in (c).
   Now we proof $S \cup \{x_i\}$ is the subset of an optimal set. Suppose there is an optimal set that doesn't contain x_i. Then we can construct a maximal set $S' = S^* \setminus \{y\} \cup \{x_i\}$. If $w(y) < w(x)$, S' is the optimal set. Contradiction! If $w(y) = w(x)$, S' is also an optimal set. Since $S \subset S^*$, we have $S \subset S'$. Since $\{x_i\} \in S'$, we have $S \cup \{x_i\} \subset S'$.
   Thus the algorithm always return the optiaml set.

(e) Construct set $\mathcal{I} = \{F \subset U \mid \text{all vectors in F are linearly independent.}\}$.

Thus for any subset of F, the vectors it contains are linearly independent.

Take $A, B \in \mathcal{I}$ with $|A| < |B|$. Suppose $\forall b \in B$, b is linearly correlated with a corresponding vector $a \in A$. Since $b_1, b_2 \in B$ cannot be linearly correlated with a same vector in A(otherwise b_1, b_2 are linearly correlated), there must be $b_{|B|}$ with no corresponding linearly correlated vector, thus $b_{|B|} \cup A \in \mathcal{I}$.

Thus $M = (U, \mathcal{I})$ is a matroid. Then use the algorithm given in (c) to find the optimal set.

2. Select an arbitrary vertex as root, then determine the level of each vertex by DFS. The root's level is 1, its children's level is 2 and so on.

   1. $S \leftarrow \emptyset$.
   2. Each time select an uncovered leaf with maximal level:
   3. Find its k-th ancestor x. If k-th ancestor doesn't exist, choose root as x.(if $k = 2$ then just find its grandparent.)
   4. $S \leftarrow S \cup \{x\}$, add all the vertices to U that are within distance k from x.
   5. Until $U = V$.

   Proof of Correctness:
   Basic Step: Assume the first element we add to S is x_0. If $\{x_0\}$ is not a subset of any optimal set, then they must contain x_0's descendants.(otherwise the leaves won't be contained.)
   If x_0 has multiple children, choosing its descendants causes S larger because if we choose a descendants in one branch, the leaves in other branches cannot be covered. Thus x_0 is a better choice.
   If x_0 has a single child, choosing its child cannot contain x_0's k-th ancestor, thus x_0 is a better choice.
   If x_0 is root, we have all the vertices are within distance of k from x.
   Thus $\{x_0\}$ is the subset of some optimal set.
   Induction Hypothesis: Suppose after i-th iteration we have S and $S \subset S^*$, S^* is an optimal set.
   The (i+1)-th selection adds x to S. Suppose there is no optimal set that contains $S \cup \{x\}$.
   Similar to basic step, these optimal sets contain x's descendants, which causes contradiction. Thus $S \cup \{x\}$ is a subset of some optimal sets.

   Time Complexity: DFS: $O(|V| + |E|)$
   Sort vertices into decreasing order of level: $O(|V| \log |V|)$
   Add vertices into U: $O(|V|)$
   Total time complexity: $O(|V| \log |V|)$

3. (a) **Initialize:** Employ DFS on G for first. During DFS we count the amount of CCs and vertices in each CC(denoted as $w(CC)$). The set C includes all CCs that DFS finds. Let $i = 1,\ S = \emptyset$.
   Sort C in decreasing order by weight w.
   For $x \in C$ in decreasing order of w:
   If $i < k$: Take an arbitrary vertex a in x and $S \leftarrow S \cup \{a\}$, $i \leftarrow i + 1$.
   If $i = k$: **endfor**.
   endfor
   return S
   The algorithm takes $O(|V| + |E|)$ time.

   (b) Denote $f(S) = w(S) = \sum_{x \in S} w(x)$. Firstly we employ DFS to find SCCs in G and count the vertices each SCC contains as $w(SCC)$, then denote each SCC as a super node. These SCCs construct a new graph G^*, which is a DAG. For each super node in G^*, if its in-degree is 0, it is a sourse. For each sourse we employ DFS on it to find the super nodes it can reach. Let the set A contains each sourse and super nodes it can reach. Let $w(A) = \sum_{x \in A} w(x)$ and T contains all these A. U contains all these super nodes in G^*.

   1. Initialize $S \leftarrow \emptyset$
   2. Repeate the followings:
   3. find $A \in T \setminus S$ that maximizes $f(S \cup \{A\}) - f(S)$
   4. update $S \leftarrow S \cup \{A\}$.
   5. Until $f(S) = |V|$ or $|S| = k$.

   Proof of Correctness:
   Suppose the optimal set is S_{OPT}, each element of it covers $\frac{1}{k}$ fraction.
   Let $S = \{A_1 \ldots A_k\}$ be the output of the algorithm.
   $\{A_1\}$ covers $\frac{1}{k}$ fraction, $\{A_1, A_2\}$ covers $1 - (1 - \frac{1}{k})^2$ fraction.
   Thus $f(S) \geq \left(1 - (1 - \frac{1}{k})^k\right) f(S_{OPT}) \geq \left(1 - \frac{1}{e}\right) f(S_{OPT})$.
   Thus this is a $(1 - \frac{1}{e})$-approximation.

   (c) Let $U = V$ be the ground set of vertices. Consider there donot exist SCC in G so G is a DAG. We can find sourses in G denoted as s by topological order. Then the max-k-coverage problem can be view as to minimize the sourses we choose and maximize the vertices these chosen DAGs contain, which is a special case of the maximum reachability problem.
   Thus the maximum reachability problem is NP-hard.

4. It takes me 14 hours to finish the work. Difficulty is 5. Collaborators: Li Haochen, Yu Junjie.