Algorithm Design and Analysis (Fall 2021)
Assignment 5

1. (0 points) Given an $n \times n$ matrix A whose entries are either 0 or 1, you are allowed to choose a set of entries with value 1 and modify their value to 0. Design an efficient algorithm to decide if we can make A invertible (A is invertible if the determinant of A is not 0) by the above-mentioned modification. Prove the correctness of your algorithm and analyze its time complexity. Notice that your algorithm only needs to output "yes" or "no" and does not need to find the minimum number of modified entries. (Hint: Prove that this is possible if and only if there exist n entries with value 1 such that no two entries are in the same row and no two entries are in the same column.)

   We first prove the following lemma.

   Lemma 1. *We can choose a set of entries from 1 to 0 to make A invertible if and only if there exist n entries with value 1 such that no two entries are in the same row and no two entries are in the same column.*

   Proof. Suppose it is possible to choose a set of entries from 1 to 0 to make A invertible. By the definition of determinant, we have

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i},$$

   where the summation is taken over all possible permutations $\sigma \in S_n$. Since we can only change an entry from 1 to 0 (not from 0 to 1), we must have at least one permutation σ such that $a_{i,\sigma_i} = 1$ for each $i = 1, \ldots, n$. Otherwise, we have $\prod_{i=1}^{n} a_{i,\sigma_i} = 0$ for all permutations no matter how we make changes to the entries, and this will make $\det(A) = 0$. Therefore, there exist n entries with value 1 such that no two entries are in the same row and no two entries are in the same column.

   If there exist n entries with value 1 such that no two entries are in the same row and no two entries are in the same column, we can change all the remaining entries to 0, and the determinant of the matrix is either 1 or -1, which is invertible. □

   With the lemma, we only need to *decide* if A contains n entries with value 1 such that no two entries are in the same row and no two entries are in the same column. This problem reduces to the problem of deciding if a bipartite graph $G = (U, V, E)$ with $|U| = |V| = n$ contains a perfect matching. To see this, given A, we can construct G as follows: create n vertices for each of U and V, and create an edge between the i-th vertex of U and the j-th vertex of V if and only if $a_{ij} = 1$. Thus, an entry a_{ij} with value 1 corresponds to an edge in G. It is easy to see that G contains a matching of size n

(i.e., a perfect matching) if and only if A contains n entries with value 1 such that no two entries are in the same row and no two entries are in the same column.

Now the problem becomes deciding if G contains a perfect matching. This can be done by Hopcroft-Karp Algorithm.

Putting these together, our algorithm can be described as

1. Construct a bipartite graph $G = (U, V, E)$ by the above-mentioned procedure.
2. Decide if G contains a perfect matching.

The correctness of the algorithm is just based on Lemma 1.

The time complexity for building G is $O(n^2)$. The time complexity for Hopcroft-Karp algorithm here is $O(\sqrt{|U| + |V|} \cdot |E|) = O(n^{2.5})$. Thus, the overall time complexity is $O(n^{2.5})$.

2. (25 points) Consider the maximum flow problem $(G = (V, E), s, t, c)$ on graphs where the capacities for all edges are 1: $c(e) = 1$ for each $e \in E$. You can assume there is no pair of anti-parallel edges: for each pair of vertices $u, v \in V$, we cannot have both $(u, v) \in E$ and $(v, u) \in E$. You can also assume every vertex is reachable from s.

   (a) (13 points) Prove that Dinic's algorithm runs in $O(|E|^{3/2})$ time.

   (b) (12 points) Prove that Dinic's algorithm runs in $O(|V|^{2/3} \cdot |E|)$ time. (Hint: Let f be the flow after $2|V|^{2/3}$ iterations of the algorithm. Let D_i be the set of vertices at distance i from s in the residual network G^f. Prove that there exists i such that $|D_i \cup D_{i+1}| \leq |V|^{1/3}$.)

We have seen in the class that a single iteration of the Dinic's algorithm requires $O(|E|)$ time if all edges have capacity 1. It remains to show that the number of iterations is bounded by $O(\sqrt{|E|})$ (for part (a)) and $O(|V|^{2/3})$ (for part (b)).

(a) If the algorithm terminates after $\sqrt{|E|}$ iterations, we are done. Suppose this is not the case. Let G^f be the residual network after $\sqrt{|E|}$ iterations. The distance between s and t in G^f is at least $\sqrt{|E|}$, as we have seen in the class that this distance is increased by at least 1 after each iteration of Dinic's algorithm. The maximum flow on G^f must be a union of many edge-disjoint s-t paths, since the flow must be integral (as the edges have integer capacities) and each edge has capacity 1. Since the distance between s and t in G^f is at least $\sqrt{|E|}$, there can be at most $|E|/\sqrt{|E|} = \sqrt{|E|}$ edge-disjoint s-t paths. This implies the maximum flow on G^f is at most $\sqrt{|E|}$.

Since each iteration increases the value of the flow by at least 1, there can be at most $\sqrt{|E|}$ extra iterations. Thus, the total number of iterations is $2\sqrt{|E|} = O(\sqrt{|E|})$.

(b) If the algorithm terminates after $2|V|^{2/3}$ iterations, we are done. Suppose this is not the case. Let G^f be the residual network after $2|V|^{2/3}$ iterations. The distance between s and t in G^f is then at least $2|V|^{2/3}$. Let D_i be the set of vertices at distance i from s. Those layers D_i's are disjoint, and there are at least $2|V|^{2/3}$ layers. We have $\sum_{i=1}^{|V|^{2/3}} |D_{2i-1} \cup D_{2i}| \leq |V|$. Thus, there exist i such that $|D_{2i-1} \cup D_{2i}| \leq |V|^{1/3}$. The total number of edges between D_{2i-1} and D_{2i} is then at most $|V|^{2/3}$. Since removing all those edges disconnect t from s, the minimum cut of G^f is at most $|V|^{2/3}$. By the Max-Flow-Min-Cut Theorem, the maximum flow is at most $|V|^{2/3}$.

Since each iteration increases the value of the flow by at least 1, there can be at most $|V|^{2/3}$ extra iterations. Thus, the total number of iterations is $3|V|^{2/3} = O(|V|^{2/3})$.

3. (20 points) Given an undirected and unweighted graph $G = (V, E)$, a *vertex cover* is a subset S of vertices such that it contains at least one endpoint of every edge, and an *independent set* is a subset T of vertices such that $(u, v) \notin E$ for any $u, v \in T$. Notice that V is a vertex cover, and $\emptyset$, or any set containing a single vertex, is an independent set. A *minimum vertex cover* is a vertex cover containing a minimum number of vertices, and a *maximum independent set* is an independent set containing a maximum number of vertices.

   (a) (5 points) Prove that S is a vertex cover if and only if $V \setminus S$ is an independent set.

   (b) (5 points) Prove that S is a minimum vertex cover if and only if $V \setminus S$ is an maximum independent set.

   (c) (10 points) Design an efficient algorithm to find a minimum vertex cover, or a maximum independent set, on *bipartite graphs*. Prove the correctness of your algorithm and analyze its time complexity.

(a) If S is a vertex cover, then every edge (u, v) must satisfy $u \in S$ or $v \in S$ (or both). Therefore, $V \setminus S$ can contain at most one of u and v. This already implies $V \setminus S$ is an independent set.

If $V \setminus S$ is an independent set, then, for every edge (u, v), $V \setminus S$ can contain at most one of u and v. Thus, at least one of u or v is in S. This implies S is a vertex cover.

(b) As the sum $|S| + |V \setminus S| = |V|$ is fixed, minimizing $|S|$ maximizes $|V \setminus S|$, and maximizing $|V \setminus S|$ minimizes $|S|$.

(c) Given a bipartite graph $G = (A, B, E)$, construct a max-flow instance as follows. Create two vertices s and t. Create an edge (s, a) for each $a \in A$ with capacity 1. Create an edge (b, t) for each $b \in B$ with capacity 1. Create an edge (a, b) for $a \in A$ and $b \in B$ with capacity ∞ if and only if (a, b) is in the original bipartite graph. Then, by finding a maximum flow on the graph, we find a minimum s-t cut (L, R) with $s \in L$ and $t \in R$. Finally, we output $(R \cap A) \cup (L \cap B)$, which is a minimum vertex cover. This completes the description of our algorithm.

To prove the correctness, first notice that the vertex cover has size $\sigma := |R \cap A| + |L \cap B|$, which is exactly the size of the cut (L, R) (notice that no edge with weight/capacity ∞ can be cut in a min-cut). Suppose for the sake of contradiction there is another vertex cover S with size less than σ. We have $|S| = |S \cap A| + |S \cap B|$. There is no edge from $A \setminus S$ to $B \setminus S$, so (L_S, R_S) with $L_S = (A \setminus S) \cup (B \cap S) \cup \{s\}$ and $R_S = (A \cap S) \cup (B \setminus S) \cup \{t\}$ is a cut (note that edges are directed in the cut instance, so an edge between A and B can only connect from a vertex in A to a vertex in B) with value $|S \cap A| + |S \cap B| = |S|$, which is less than σ. This contradicts to (L, R) is a min-cut.

The time complexity is $O(\sqrt{|V|} \cdot |E|)$ for running Hopcroft-Karp Algorithm.

4. (20 points) Covert the following linear program to its standard form, and compute its dual program.

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 7x_2 + x_3 \\ \text{subject to} \quad & x_1 - x_3 = 7 \\ & 3x_1 + x_2 \geq 24 \\ & x_2 \geq 0 \\ & x_3 \leq 0 \end{aligned}$$

Standard form:

$$\begin{aligned} \text{maximize} \quad & -2x_1^+ + 2x_1^- - 7x_2 + x_3 \\ \text{subject to} \quad & x_1^+ - x_1^- + x_3 \leq 7 \\ & -x_1^+ + x_1^- - x_3 \leq -7 \\ & -3x_1^+ + 3x_1^- - x_2 \leq -24 \\ & x_1^+, x_1^-, x_2, x_3 \geq 0 \end{aligned}$$

Dual Program:

$$\begin{aligned} \text{minimize} \quad & 7y_1 - 7y_2 - 24y_3 \\ \text{subject to} \quad & y_1 - y_2 - 3y_3 \geq -2 \\ & -y_1 + y_2 + 3y_3 \geq 2 \\ & -y_3 \geq -7 \\ & y_1 - y_2 \geq 1 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

5. (35 points) In this question, we will prove König-Egerváry Theorem, which states that, in any bipartite graph, the size of the maximum matching equals to the size of the minimum vertex cover. Let $G = (V, E)$ be a bipartite graph.

   (a) (5 points) Explain that the following is an LP-relaxation for the maximum matching problem.

$$\begin{aligned} \text{maximize} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e:e=(u,v)} x_e \leq 1 && (\forall v \in V) \\ & x_e \geq 0 && (\forall e \in E) \end{aligned}$$

   (b) (5 points) Write down the dual of the above linear program, and justify that the dual program is an LP-relaxation to the minimum vertex cover problem.

   (c) (10 points) Show by induction that the *incident matrix* of a bipartite graph is totally unimodular. (Given an undirected graph $G = (V, E)$, the incident matrix A is a $|V| \times |E|$ zero-one matrix where $a_{ij} = 1$ if and only if the i-th vertex and the j-th edge are incident.)

   (d) (10 points) Use results in (a), (b) and (c) to prove König-Egerváry Theorem.

   (e) (5 points) Give a counterexample to show that the claim in König-Egerváry Theorem fails if the graph is not bipartite.

(a) x_e with restriction $x_e \in \{0, 1\}$ indicates whether e is selected in the matching. Thus, $\sum_{e \in E} x_e$ is the number of edges in the matching for which we want to maximize, and $\forall v \in V : \sum_{e:e=(u,v)} x_e \leq 1$ says that at most one edge incident to v can be selected.

(b) The dual program is as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} y_v \\ \text{subject to} \quad & y_u + y_v \geq 1 && (\forall (u, v) \in E) \\ & y_v \geq 0 && (\forall v \in V) \end{aligned}$$

Here, y_v with restriction $y_v \in \{0, 1\}$ indicates whether v is selected in the vertex cover. $\sum_{v \in V} y_v$ is then the size of the vertex cover, and $y_u + y_v \geq 1$ says that one or two of u, v must be selected for each edge (u, v).

(c) For the base step, each 1×1 sub-matrix, which is an entry, is either 1 or 0, which has determinant either 1 or 0.

For the inductive step, suppose the determinant of any $k \times k$ sub-matrix is from $\{-1, 0, 1\}$. Consider any $(k + 1) \times (k + 1)$ sub-matrix T. Since each edge have exactly two endpoints, each column of T can contain at most two 1s.

If there is an all zero-column in T, we know $\det(T) = 0$, and we are done.

If there is a column in T contains only one 1, then $\det(T)$ equals to 1 or -1 multiplies the determinant of a $k \times k$ sub-matrix, which is from $\{-1, 0, 1\}$ by the induction hypothesis. Thus, $\det(T) \in \{-1, 0, 1\}$.

If every column in T contains two 1s, by the nature of bipartite graph, we can partition the rows into the upper half and the lower half such that each column of T contains exactly one 1 in the upper half and one 1 in the lower half. By multiplying 1 to the upper-half rows and -1 to the lower-half rows, and adding all of them, we will get a row of zeros. This means the set of all rows of T are linearly dependent, and $\det(T) = 0$.

We conclude the inductive step.

(d) It is easy to see that the coefficients in the constraints of the primal LP form exactly the incident matrix, and the coefficients in the constraints of the dual LP form the transpose of the incident matrix. Since this matrix is totally unimodular and the values at the right-hand side of the constraints in both linear programs are integers, both linear programs have integral optimal solutions.

In addition, it is straightforward to check that any primal LP solution with $x_e \geq 2$ cannot be feasible and any dual LP solution with $y_u \geq 2$ cannot be optimal. Therefore, there exists an primal LP optimal solution $\{x_e^*\}$ with $x_e^* \in \{0, 1\}$, and there exists an dual LP optimal solution $\{y_v^*\}$ with $y_v^* \in \{0, 1\}$. We have argued in (a) and (b) that both solutions can now represent a maximum matching and a minimum vertex cover respectively. By the LP-duality theorem, the primal LP objective value for the optimal solution $\{x_e^*\}$ equals to the dual LP objective value for the optimal solution $\{y_v^*\}$. This implies König-Egerváry Theorem.

(e) The simplest counterexample would be a triangle, where the maximum matching has size 1 and the minimum vertex cover has size 2.