1

Let k, l be positive integers and let $a = (a_1, \ldots, a_k)$ and $b = (b_1, \ldots, b_l)$ be two sequences of non-negative integers. You need to decide whether there is a bipartite graph in which $a_1, \ldots, a_k$ are the degrees for one partite set, and $b_1, \ldots, b_l$ are the degrees for the other. Describe a polynomial time algorithm to solve this problem. Show correctness of your algorithm and analyze its running time.

The pair (a, b) is *bigraphic* if there is a *simple* bipartite graph in which $a_1, \ldots, a_k$ are the degrees for one partite set and $b_1, \ldots, b_l$ are the degrees for the other. Without loss of generality, we assume $a_i \leq l$ for $1 \leq i \leq l$ and $b_1 \geq b_2 \geq \ldots \geq b_l$.

Lemma 1.1 *(a, b) is bigraphic if and only if (a', b') is bigraphic, where (a', b') is obtained from (a, b) by deleting a_1 from a and subtracting 1 from $b_1, \ldots, b_{a_1}$ of b.*

Proof. Suppose there is a bipartite graph G' with degree sequences (a', b') where a' is the degree sequence of partite set X and b' is the degree sequence of partite set Y. By adding a new vertex to the partite set X adjacents to vertices in Y with degrees $b_1 - 1, \ldots, b_{a_1} - 1$, the resulting graph shows that (a, b) is bigraphic.

Suppose there is a bipartite graph G with degree sequences (a, b) where a is the degree sequence of partite set X and b is the degree sequence of partite set Y, we shall prove that there is a simple graph G' realizing (a', b'). Let S be a set of a_1 vertices in Y having degrees $b_1, \ldots, b_{a_1}$ and let $x_1 \in X$ has degree a_1. If $N_G(x_1) = S$, then we delete x_1 to obtain G'. Otherwise, since $|N_G(x_1)| = |S|$, there is a vertex $y_i \in Y$ such that $y_i \in S$ and $y_i x_1 \notin E(G)$, and there is a vertex $y_j \in Y$ such that $y_j \notin S$ and $y_j x_1 \in E(G)$. Also, since $y_i \in S$ and $y_j \notin S$, $|N_G(y_i)| \geq |N_G(y_j)|$. As y_j is adjacent to x_1 but y_i is not adjacent to x_1, there exists a vertex $x_k \in X$ such that $y_i x_k \in E(G)$ but $y_j x_k \notin E(G)$. Now, we delete $\{y_i x_k, y_j x_1\}$ and add $\{y_i x_1, y_j x_k\}$, the resulting simple graph G_1 has degree sequences (a, b) and $|N_{G_1}(x_1) \cap S| > |N_{G_0}(x_1) \cap S|$. Repeat this procedure at most a_1 times, we can obtain a simple graph G^* with degree sequences (a, b) and $N_{G^*}(x_1) = S$. Now we can delete x_1 in G^* to obtain a simple graph G' with degree sequences (a', b'). ∎

Algorithm aspects: from the proof of Lemma 1.1, it is easy to derive a polynomial time algorithm with running time $O(kl \log l)$ as follows. First, we sort b (*note that we do not need to sort a since we never use the assumption that a_1 is the largest degree*). At each iteration, we decrease the largest a_i positive number of b by 1 (if there are less than a_i positive number in b, the algorithm returns NO), then we sort b again and repeat the above procedure until a is empty.

By implementing the algorithm more carefully, we can reduce the running time of the above procedure to $O(m)$, where $m = \sum_{i=1}^{k} a_i$ is the number of edges in the resulting graph, if it exists. This is the best possible if we are required to construct a graph with degree sequences (a, b). The idea is to use a linked list of linked lists to store b.

First, we create an array A of size k and initialize $A[j] = 0$ for $1 \leq j \leq k$. Then, for each $b_i \in b$, we increment $A[j]$ if $b_i = j$ (note that we can assume $1 \leq b_i \leq k$). After building A, we create a linked list L of linked lists as follows. Each element of a linked list has a variable TIME. Now, for i starts from 1 to k, we do the following: if $A[i] > 0$, we insert a linked list L_i of $A[i]$ elements to the head of L; for all $A[i]$ newly added elements, the variables TIME are assigned to 0. (If $A[i] = 0$, we do nothing.) This completes the preprocessing step, which can be done in $O(l + k) = O(m)$ time.

In the i-th iteration (which corresponds to the step that decrease the a_i largest elements of b by 1), we start at the head of L (i.e. the highest degree list, say L_j). Repeat the following procedure a_i times before proceed to the next iteration.

1. If the element e at the head of L_j has TIME $< i$, we remove e from L_j and mark its TIME to i.
   (a) If $j = 1$, we do nothing.
   (b) Otherwise, if the next linked list in L is L_{j-1}, we insert e to the tail of L_{j-1}; else, we create L_{j-1} (so that L_{j-1} is the next linked list of L_j in L) and insert e to the tail of L_{j-1}.
2. Otherwise, if we are already at the tail of L, return NO; else, we move to the next linked list in L; if L_j is empty after removing e, remove L_j from L.

Notice that each iteration can be implemented in $O(a_i)$ time, since step (1) repeats at most a_i times, step (2) repeates at most $2a_i$ times and each step needs only $O(1)$ time because we use linked list. So the total complexity is $O(m)$. The correctness of the implementation is omitted (details are covered in tutorial).

2

In class we showed that among the n-vertex simple graphs with no $(r+1)$ clique, the Turan graph $T_{n,r}$ has the most number of edges. Show that it is the *unique* graph having no $(r+1)$ clique with that many edges.

By using Lemma 5.2.8 in West's book, we know that among simple r-partite graphs with n vertices, $T_{n,r}$ is the unique graph with the most edges. To prove the statement of this question, it suffices to prove the following lemma.

Lemma 2.1 *If G is a n-vertex simple graph with no $(r+1)$ clique and maximum number of edges, then G is r-partite.*

Proof. We prove by induction on r. The base case $r = 1$ is easy. Now suppose the lemma holds for all graphs with no r-clique. Consider a n-vertex graph G with no $(r+1)$ clique and maximum number of edges. Let v be a vertex with maximum degree Δ, $H = G[N_G(v)]$ and $m(H)$ be the number of edges in H. Then G has at most $(n-\Delta)\cdot\Delta + m(H)$ edges. Let H' be a Δ-vertex graph with no r-clique and maximum number of edges. We construct a n-vertex graph G' by adding $(n-\Delta)$ vertices to H', such that each newly added vertex is adjacent to all vertices in H' but not adjacent to the other newly added vertices. By the induction hypothesis, H' is a $(r-1)$-partite graph and thus G' is a r-partite graph with exactly $(n-\Delta)\cdot\Delta + m(H')$ edges. Note that G' has no $(r+1)$-clique. So, by our choice of G, we must have $m(H) \geq m(H')$. Notice that H is a Δ-vertex graph with no r-clique, otherwise G has a $(r+1)$-clique. By the induction hypothesis and the maximality of H', it follows that H is a $(r-1)$-partite graph. Also, by our choice of G, we must have $(n-\Delta)\cdot\Delta$ edges in G with at least one endpoint in S. Since every vertex in $V(G) - N_G(v)$ has degree at most Δ, there is no edge between vertices in $V(G) - N_G(v)$ (otherwise such an edge will be counted twice in the sum $(n-\Delta)\cdot\Delta$), which implies that $V(G) - N_G(v)$ induces an independent set in G. As a result, G is a r-partite graph; this completes the proof. ∎

3

Let S and T be trees with leaves $\{x_1, \ldots, x_k\}$ and $\{y_1, \ldots, y_k\}$, respectively. Suppose that $d_S(x_i, x_j) = d_T(y_i, y_j)$ for each pair i, j. Prove that S and T are isomorphic with x_i mapped to y_i in the isomorphism.

(Please note that we are proving a slightly stronger statement than that in the original assignment).

We prove by induction on the number of leaves k. The base case when $k = 1$ is trivial. For $k = 2$, a tree with two leaves is a path, and all paths with the same length are isomorphic. Suppose the statement holds for all trees with at most $k \geq 2$ leaves. Consider two trees S and T with leaves $\{x_1, \ldots, x_{k+1}\}$ and $\{y_1, \ldots, y_{k+1}\}$ such that $d_S(x_i, x_j) = d_T(y_i, y_j)$ for each pair $1 \leq i, j \leq k + 1$. We shall show that S and T are isomorphic. Let $P_S(x_k) = \{x_k, a_1, a_2, \ldots, a_l, x'\}$ be a maximal path starting from x_k and ending at x' with every a_i has degree 2 in S. We construct S' from S by deleting $\{x_k, a_1, \ldots, a_l\}$, and we construct T' from T and define $P_T(y_k)$, y' in the same way. Since S and T have at least three leaves, x' and y' are not leaves and so S' and T' are trees with $k - 1$ leaves. For $1 \leq i, j \leq k - 1$, $d_{S'}(x_i, x_j) = d_S(x_i, x_j) = d_T(x_i, x_j) = d_{T'}(x_i, x_j)$. By the induction hypothesis, S' and T' are isomorphic.

Suppose x' is mapped to y' in the isomorphic mapping (we shall prove it in the next page). Since x_1 is mapped to y_1 by the induction hypothesis, $d_{S'}(x_1, x') = d_{T'}(y_1, y')$. Note that $d_S(x_1, x_k) = d_S(x_1, x') + d_S(x', x_k)$ and similarly $d_T(y_1, y_k) = d_T(y_1, y') + d_T(y', y_k)$. So $d_S(x_1, x_k) = d_T(y_1, y_k)$ implies that $d_S(x', x_k) = d_T(y', y_k)$. As a result, by adding $P_S(x_k)$ to S' and $P_T(y_k)$ to T', S and T are isomorphic and each x_i is mapped to y_i.

To complete the proof, we prove that x' must be mapped to y' (i.e. $f(x') = y$). Suppose, by way of contradiction, that x' is mapped to $f(x') \neq y$ in T'. Let $\{y_a, \ldots, f(x'), \ldots, y', \ldots, y_b\}$ be a path in T' with y_a and y_b are leaves, and $d_T(f(x'), y) > 0$.

On one hand, by looking at S, we have

$$d_S(x_a, x_k) = d_S(x_a, x') + d_S(x', x_k) \tag{1}$$

and

$$d_S(x_b, x_k) = d_S(x_b, x') + d_S(x', x_k). \tag{2}$$

On the other hand, by looking at T, we have

$$d_T(y_a, y_k) = d_T(y_a, y') + d_T(y', y_k) = d_T(y_a, f(x')) + d_T(f(x'), y') + d_T(y', y_k) \tag{3}$$

and

$$d_T(y_b, y_k) = d_T(y_b, y') + d_T(y', y_k) = d_T(y_b, f(x')) - d(f(x'), y') + d_T(y', y_k). \tag{4}$$

Since f is an isomorphic mapping, we have $d_S(x_a, x') = d_T(y_a, f(x'))$ and $d_S(x', x_b) = d_T(f(x'), y_b)$. Also, we have $d_S(x_a, x_k) = d_T(y_a, y_k)$ and $d_S(x_b, x_k) = d_T(y_b, y_k)$.

Using the above facts and equating (1) and (3) gives

$$d_S(x', x_k) = d_T(f(x'), y') + d_T(y', y_k) \tag{5}$$

Again using the above facts and equating (2) and (4) gives

$$d_S(x', x_k) = -d_T(f(x'), y') + d_T(y', y_k) \tag{6}$$

Finally, subtracting (5) and (6), we have $d_T(f(x'), y') = 0$, a contradiction. So x' must be mapped to y' and this completes the proof.

4

A *minimax* spanning tree is a spanning tree that minimizes the maximal edge weight. Show that every minimum spanning tree is also a minimax spanning tree.

Suppose, by way of contradiction, that T is a minimum spanning tree but not a minimax spanning tree. Let T' be a minimax spanning tree. Choose an edge $e \in E(T)$ with the maximum edge weight. By removing e from T, there are exactly two components C_1 and C_2 in $T - e$. Since T' is a spanning tree, there is an edge e' with one endpoint in C_1 and the other endpoint in C_2. So, e' connects C_1 and C_2 and $T - e + e'$ is a spanning tree. By our choice of T and e, $w(e') < w(e)$ but this implies the total weight of $T - e + e'$ is smaller than that of T, which contradicts our assumption that T is a minimum spanning tree. Therefore, T doesn't exist and we conclude that a minimum spanning tree is also a minimax spanning tree.

Solutions to 4(a) and the second part of 4(b) are omitted.

Solutions to 5 and 6 are omitted.