1. **Max k-Cut via Randomization: 16 points**
   Let $G = (V, E)$ be graph. Max k-Cut is the problem of partitioning the node set V into k mutually disjoint sets $V_1, V_2, \ldots, V_k$ to maximize the number of edges that are cut: an edge uv is cut if u and v are in different sets of the partition. Consider the following simple randomized algorithm. Assign independently to each node $v \in V$ a color chosen uniformly at random from $\{1, 2, \ldots, k\}$. The node set V_i is simply the set of all nodes that get color i.

   - (10 points) What is the probability that an edge $e = uv$ is cut by the algorithm? That is, what is the probability that u and v get different colors?
   - (6 points) Using the previous part give a precise expression for the expected number of edges cut by the partition produced by the randomized algorithm as a function of m and k where m is the number of edges in G.

   Solution: For part (a) the answer is $1 - 1/k$. For part (b) by linearity of expectation it is $m(1 - 1/k)$.

2. **Locating Burger Joints: 21 points**
The McKing chain wants to open several restaurants along Red street in Shampoo-Banana. The possible locations are at $L_1, L_2, \ldots, L_n$ where L_i is at distance m_i meters from the start of Red street. Assume that the street is a straight line and the locations are in increasing order of distance from the starting point (thus $0 \leq m_1 < m_2 < \ldots < m_n$). McKing has collected some data indicating that opening a restaurant at location L_i will yield a profit of p_i independent of where the other restaurants are located. However, the city of Shampoo-Banana has a zoning law which requires that any two McKing locations should be D or more meters apart. Moreover, McKing has realized that its budget constraints imply that it can open at most $k < n$ restaurants. Describe an algorithm that McKing can use to figure out the locations for its restaurants so as to maximize its profit while satisfying its budget constraint and the city's zoning law.

Solution: We solve it via dynamic programming.

For $0 \leq i \leq n$ and $0 \leq j \leq k$ let $Opt(i, j)$ be the maximum profit by opening *at most* j restaurants in the locations 1 to i such that the restaurants are at least D meters apart. For the base case we have $Opt(i, 0) = 0$ for all i and $Opt(0, j) = 0$ for all j.

We now derive a recurrence for $Opt(i, j)$ as follows. For notational convenience, for each i, let $\alpha(i)$ denote the largest index $h < i$ such that $m_i - m_h \geq D$. Consider solutions that do not open a restaurant at i then the maximum profit among such solutions is given by $Opt(i-1, j)$. Now consider solutions that open a restaurant at i. Then the nearest location to the left of location i where another restaurant can be opened is at location $\alpha(i)$. Thus among solutions that open a restaurant at i the maximum profit one has value $p_i + Opt(\alpha(i), j-1)$ since we can open at most $j-1$ restaurants in locations 1 to $\alpha(i)$. Thus

$$Opt(i, j) = \max\{Opt(i-1, j), Opt(\alpha(i), j-1) + p_i\}.$$

One can easily compute all the $\alpha(i)$ values after sorting the m_i values in $O(n \log n)$ time. Once we have them, the time to compute $Opt(i, j)$ from previously computed values is $O(1)$. The number of subproblems is $O(nk)$. Hence the total time to evaluate all of them is $O(nk+n \log n)$ and space is $O(nk)$. We output the value $Opt(n, k)$.

3. **Spanning Trees: 21 points**
Let $G = (V, E)$ be an undirected graph with weights on the edges and let X be a proper subset of the vertices (that is $X \subset V$ and $X \neq V$). A spanning tree T of G is an X-tree if every vertex in X is a leaf of T. Recall that a leaf in a tree is a vertex with degree 1 in the tree. See figure below.

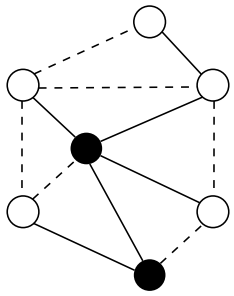

Figure 1: A graph with vertices in X shown in black. The dotted edges represent an X-tree.

- (4 points) Give an example of a connected graph $G = (V, E)$ and a proper subset $X \subset V$ such that G has no X-tree.
- (18 points) Give an algorithm that given an edge-weighted graph $G = (V, E)$ and a subset of vertices $X \subset V$, finds a minimum weight X-tree of G or correctly states that there is no X-tree in G.

Solution: For part (a) consider a three node graph with vertices u, v, w with $(u, v), (v, w)$. Let $X = \{v\}$. The graph is a tree and $v \in X$ has degree 2 and hence there is no other tree in which v has degree 1.

For part (b) we observe the following. If T is an X-tree then removing the nodes in X from T will leave a tree T' on $V \setminus X$ since each node of X is a leaf in T. Moreover, if $u \in X$ and (u, v) is the unique edge in T incident to u then $v \notin X$ for otherwise v will have to have degree 2 in T (recall that X is a proper subset of V). Therefore edges connecting two X-nodes are not useful for an X-tree.

From the above observations we derive the algorithm.

(a) From G remove all edges (u, v) where both $u, v \in X$.

(b) Let G' be the graph obtained by removing X from G. If G' is not connected, output that G has no X-tree.

(c) Compute an MST T in G'.

(d) For each $u \in X$

  i. If u has no edges incident to it in G, output that there is no X-tree in G

  ii. Else let (u, v) be the cheapest edge incident to u in G. Add (u, v) to T.

(e) Output T

The algorithm can be implemented in $O(m + n + A(m, n))$ time where $A(m, n)$ is the time to find an MST in a graph with m edges and n nodes. You can use any MST algorithm. In particular, we have an $O((m + n) \log n)$ algorithm for MST computation so we get an overall

running time of $O((m+n)\log n)$ where m is the number of edges in G and n is the number of nodes.

Note: It is possible to modify Prim's and/or Kruskal's algorithm to obtain a correct solution to the above. That is a reasonable approach but it is often better to step back and understand the problem structure first before thinking of a specific algorithm.

4. **Disjoint Paths: 21 points**
Let $G = (V, E)$ be a directed graph and let u, v, w be three distinct vertices. Suppose u has k edge-disjoint paths $P_1, P_2, \ldots, P_k$ to v in G, and v has k edge-disjoint paths $Q_1, Q_2, \ldots, Q_k$ to w in G. Note that a path P_i may not be edge disjoint from Q_j for any $1 \leq j \leq k$.

- (6 pts) Give an example with $k = 2$ where each of P_1, P_2 has an edge in common with each of Q_1, Q_2.
- (16 pts) Prove that u has k edge-disjoint paths to w in G. *Hint:* Use maxflow-mincut theorem.

Solution: This is essentially the same problem as that in one of the discussion sections on transitivity of min-cut values. Let $\alpha(s, t)$ be the min-cut between s and t in G where each edge has capacity 1. Since u has k edge-disjoint paths to v we have $\alpha(u, v) \geq k$. Similarly $\alpha(v, w) \geq k$. We had seen in the discussion problem that $\alpha(u, w) \geq \min\{\alpha(u, v), \alpha(v, w)\}$ which together with the above facts give us that $\alpha(u, w) \geq k$. Now, by the maxflow-mincut theorem, the max-flow from u to w in G is at least k. This implies that there are at least k edge-disjoint paths from u to w in G. To see why $\alpha(u, w) \geq \min\{\alpha(u, v), \alpha(v, w)\}$ we consider a min-cut (A, B) that separates u from w. If $v \in A$ then (A, B) is a cut (not necessarily a min-cut) that separates v from w and hence its capacity is at least $\alpha(v, w)$. If $v \in B$ then (A, B) is a cut that separates u from v and hence its capacity is at least $\alpha(u, v)$. Thus the capacity of (A, B) is at least $\min\{\alpha(u, v), \alpha(v, w)\}$.

5. **Freshman Orientation: 21 points**
You are tasked with a role in planning Freshman Orientation for next summer. Those higher up the ladder than you have decided that each of the n students $s_1, \ldots, s_n$ should attend two of m short workshops during the last day of orientation to discuss topics such as communicating with professors, joining extracurriculars, finding the best bars, etc. Each of the workshops $w_1, \ldots, w_m$ is being held at a separate time that day, but they are being housed in small rooms and each workshop w_i can accommodate at most c_i students. Each incoming student has given you a list $A_i \subseteq \{w_1, w_2, \ldots, w_m\}$ of workshops they are interested in attending. Describe a polynomial time algorithm that determines if it is possible to assign each of the students to two workshops they are interested in attending.

Solution: We reduce the problem to network flow. Create a flow network $G = (V, E)$ as follows. $V = \{s\} \cup S \cup W \cup \{t\}$ where $S = \{s_1, \ldots, s_n\}$ and $W = \{w_1, \ldots, w_m\}$. That is, we have one node for each student and one node for each workshop and a source node s and a sink node t. We connect s to each s_i with an arc of capacity 2 and each workshop w_j to t with an arc of capacity c_j. For each node $s_i \in S$ we add arcs (s_i, w_j) for each $w_j \in A_i$ with capacity 1.

Now we compute a max-flow in the above flow network. Let α be the value of the max-flow. If $\alpha = 2n$, output that there is a feasible assignment of students to two workshops each, otherwise output that there is no feasible assignment.

To compute the max-flow we use the plain Ford-Fulkerson algorithm. The flow-network we created has $n + m + 2$ nodes and $O(nm)$ arcs. The max-flow in the network is at most $2n$ since there is a cut $(\{s\}, V \setminus \{s\})$ of capacity $2n$. Ford-Fulkerson will terminate in at most $2n$ iterations, hence the running time of the overall algorithm is $O(mn^2)$.