1. A ***k*-orientation** of an undirected graph G is an assignment of directions to the edges of G so that every vertex of G has at most k incoming edges. Describe and analyze an algorithm that determines the smallest value of k such that G has a k-orientation, given the undirected graph G as input.

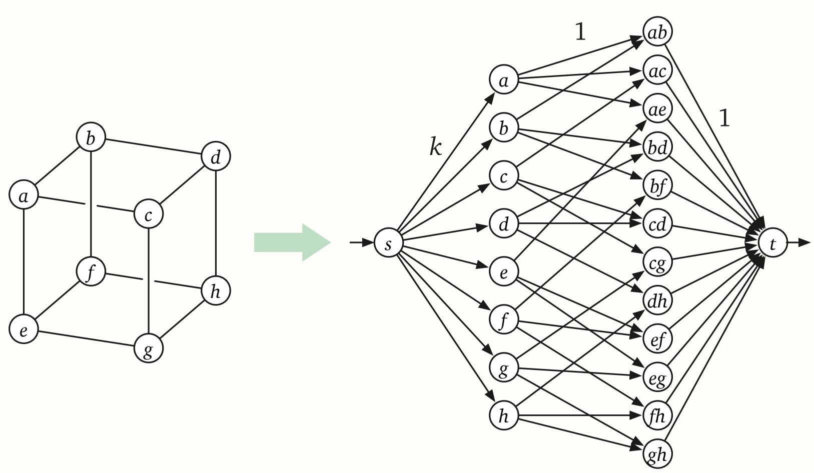
Solution: Our algorithm performs a binary search for the smallest k such that G has a k-orientation; for each value of k we consider, we intuitively look for an ***assignment*** of at most k incoming edges to each vertex. More concretely, we solve the decision problem as a generalized matching or pair-selection problem, where the two resource sets are the vertices and edges of G.

Fix an arbitrary value of k. To decide whether G has a k-orientation, we construct a flow network $H = (V', E')$ as follows:

- $V' = V \cup E \cup \{s, t\}$. Except for the source s and target t, the vertices of H correspond to the vertices *and edges* of G. Clearly $|V'| = 2 + |V| + |E| = O(E)$.
- E' contains three types of edges:
  - An edge $s \rightarrow v$, for each vertex $v \in V$.
  - An edge $v \rightarrow e$, for each edge $e \in E$ and each endpoint v of e.
  - An edge $e \rightarrow t$, for each edge $e \in E$.

  Altogether we have $|E'| = |V| + 2|E| + |E| = O(E)$.
- Each edge $s \rightarrow v$ has capacity k; all other edges have capacity 1.

The following figure shows the resulting flow network for the cube graph:



Our construction guarantees a correspondence between k-orientations of G and integer (s, t)-flows in H that saturate every edge into t; specifically, each flow path from s to t in H corresponds to a choice of direction for one edge in G.

- For any k-orientation of G, we can construct an integer flow f in H as follows. For each directed edge $u \rightarrow v$ in the orientation of G, we send one unit of flow through h along the path $s \rightarrow v \rightarrow uv \rightarrow t$; the flow f is the sum of these E paths.

Because each vertex of G has at most k incoming edges, we have $f(s\to v) \le k$ for every vertex v. Because each edge uv is either oriented into v or not, we have $f(v\to uv) \le 1$. Finally, because each edge of G has exactly one orientation, we have $f(e\to t) = 1$ for every edge E. We conclude that f is a feasible flow in H that saturates every edge into t.

- On the other hand, let f be any integer flow in H that saturates every edge into t. We can decompose f into E paths of the form $s\to v\to uv\to t$, each carrying one unit of flow. For each such path, assign edge uv the direction $u\to v$. Because $f(e\to t) = 1$ for every edge e in G, every edge e in G is assigned a unique direction. Because $f(s\to v) \le k$ for every vertex v of G, at most k edges in G are directed into v. So we have constructed a k-orientation of G.

Thus, to solve the decision problem for any fixed k, we construct the flow network H as described above, compute a maximum (s,t)-flow f^* in H, and then report success if and only if $|f^*| = E$. If we use Orlin's algorithm to compute the maximum flow, the decision algorithm runs in $O(V'E') = O(E^2)$ time.[a]

Finally, to solve the optimization problem, we perform a binary search over all possible values of k. Every graph has a V-orientation, and no graph has a (-1)-orientation, so we can limit our search to the range $0 \le k \le V-1$. It follows that our binary search requires $O(\log V)$ iterations, and thus our entire algorithm runs in $\boldsymbol{O(E^2 \log V)}$ ***time***. ■

[a] If we used Ford-Fulkerson here, our decision algorithm would run in $O(E^2 k)$ time, and so the resulting optimization algorithm would run in $O(E^2 V \log V)$ time.

Rubric: 10 points; standard reduction rubric. The proof of correctness (in gray) is not required for full credit. Max 8 points for correct $O(E^2V)$ time algorithm. +5 for a correct $O(E^2)$-time algorithm. Even this is not the fastest algorithm for this problem.

Solution (extra credit: parametric flow): Instead of performing a binary search over all possible values of k, computing a maximum flow from scratch at each iteration, we consider all k from 1 up to the maximum, updating a maximum flow at each iteration.

We essentially the same flow network $H = (V', E')$ as in the previous solution:

- $V' = V \cup E \cup \{s,t\}$. Except for the source s and target t, the vertices of H correspond to the vertices *and edges* of G. Clearly $|V'| = 2 + |V| + |E| = O(E)$.
- E' contains three types of edges:
  - An edge $s\to v$, for each vertex $v \in V$.
  - An edge $v\to e$, for each edge $e \in E$ and each endpoint v of e.
  - An edge $e\to t$, for each edge $e \in E$.

  Altogether we have $|E'| = |V| + 2|E| + |E| = O(E)$.
- Initially *every* edge in this network has capacity 1.

We now proceed in several rounds. In the kth round, we set the capacity of all edges into t to k and compute a maximum flow, starting with the maximum flow from the previous round. If the maximum flow saturates all edges into t, that flow corresponds to a k-orientation of G, so we can stop and return k. Otherwise, G does not have a k-orientation, so we proceed to the $(k+1)$th round.

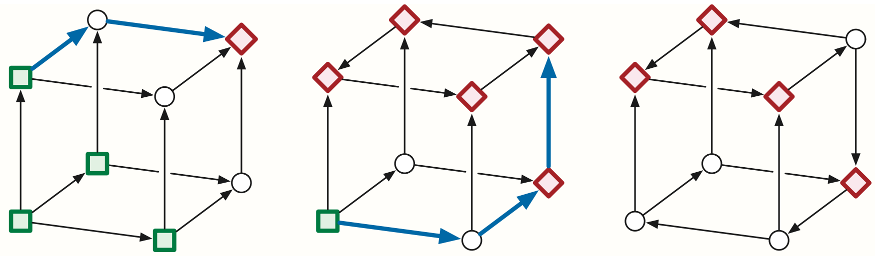
```
MinOrientation(V, E):
  Build the graph H as described above
  f ← 0        ⟨⟨flow corresponding to partial orientation⟩⟩
  for k ← 1 to V − 1
      for every vertex v ∈ v
          c(s→v) ← k
      while H_f contains a path from s to t
          P ← any path in H_f from s to t
          f = f + P        ⟨⟨push 1 unit of flow along P⟩⟩
      if f(e→t) = 1 for every edge e ∈ E
          return k
```

Finding each path P takes $O(V' + E') = O(E)$ time. Each time we push along a path P from s to t, we saturate one of the edges into t. Thus, the total number of pushes in the entire algorithm is at most E, the number of edges into t. (Equivalently: Every push increases the value of the flow by 1, and the maximum value of the flow is at most E, because the total capacity of all edges into t is E.) So the entire algorithm runs in ***$O(E^2)$ time***. ■

Solution (extra credit; greedy improvement): The following algorithm is due to Venkateswaran [2] with some later simplifications by Asahiro *et al.* [1].

```
MinOrientation(G):
  arbitrarily orient the edges of G
  repeat forever
      k ← max{indeg(v) | v ∈ V}
      Hi ← {v ∈ V | indeg(v) = k}
      Lo ← {v ∈ V | indeg(v) ≤ k − 2}
      if there is no directed path in G from Lo to Hi
          return k
      P ← any directed path in G from Lo to Hi
      reverse every edge of P
```

The integer k never increases between iterations of the main loop, so the set Lo never grows. At every iteration of the algorithm, the in-degree of one vertex in Lo increases, and the in-degree of one node in Hi decreases; otherwise, all in-degrees remain unchanged. As long as a vertex is in the set Lo, its in-degree can only increase. Thus, the number of iterations is at most the sum of the degrees (both in- and out-) of the vertices in the *initial* set Lo, which is trivially at most $2E$. Each iteration takes $O(E)$ time, so the overall algorithm runs in ***$O(E^2)$ time***.



Orienting the cube. Green square vertices are in *Lo*; red diamond vertices are in *Hi*.

We can prove this algorithm is correct as follows. Let U be the subset of vertices that are *not* reachable from the final set Lo in the final directed graph G. (In particular, if $Lo = \varnothing$, then $U = V$.) Let E_U be the set of directed edges in G whose heads (and therefore tails) are in U. Because the algorithm halted, we have $Hi \subseteq U$; every vertex in U has in-degree $k-1$ or k, and at least one vertex in T has in-degree k. Thus, $(k-1) \cdot |U| < |E_U| \leq k \cdot |U|$, which implies $k = \lceil |E_U|/|U| \rceil$. We conclude that in *every* orientation of G, some vertex in U has in-degree at least k. (In particular, some vertex in U has incoming edges from at least k other vertices in U.) ■

[1] Yuichi Asahiro, Eiji Miyano, Hirotaka Ono, and Kouhei Zenmyo. Graph orientation algorithms to minimize the maximum outdegree. *Int. J. Found. Comput. Sci* 18(2):197–215, 2007.

[2] Venkat Venkateswaran. Minimizing maximum indegree. *Discrete Appl. Math.* 143(1-3): 374–378, 2004.

2. Describe and analyze an algorithm to choose a subset of the SPU faculty to fill the Post-Factotum ~~Mascot~~ *Symbol* Committee, or correctly report that no valid committee is possible. Your input is a bipartite graph indicating which professors belong to which departments; each professor vertex is labeled with that professor's rank (assistant, associate, or full). Assume that there are n professors and $3k$ departments.

Solution: Arbitrarily index the academic ranks from 1 to 3 (for example, 1 = assistant, 2 = associate, and 3 = full), the professors from 1 to n, and the departments from 1 to $3k$. Construct a graph G with nodes $s, R_1, R_2, R_3, S_1, \ldots, S_n, D_1, \ldots, D_{3k}, t$, and the following edges:

- An edge $s \to R_i$ with capacity k for all i.
- An edge $R_i \to P_j$ with capacity 1 if and only if professor j has rank i
- An edge $P_j \to D_\ell$ with capacity 1 if and only if professor j is affiliated with department ℓ.
- An edge $D_\ell \to t$ with capacity 1 for all ℓ.

Let f be a maximum (s,t)-flow in G. If $|f| < 3k$, no legal committee assignment is possible. If $|f| = 3k$, any edge $P_j \to D_\ell$ with flow 1 indicates that professor j should be assigned to the committee as the representative of department ℓ.

The graph has $O(n + k) = O(n)$ vertices and $O(nk)$ edges—in principle, every professor could be affiliated with every department—and the maximum flow value is at most $3k$. Thus, the standard Ford-Fulkerson algorithm computes a maximum flow in $O(E \cdot |f^*|) = \boldsymbol{O(nk^2)}$ time. Once we have the flow, we can extract the committee membership in $O(k)$ additional time. ■

Rubric: 10 points: standard reduction rubric

3. (a) Describe a linear program whose solution (a, b) describes the line with minimum L_1 error.

Solution:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{i=1}^{n} R_i & \\ \text{subject to} & ax_i + b + R_i \geq y_i & \text{for all } i \\ & ax_i + b - R_i \leq y_i & \text{for all } i \end{array}$$

For each index i, the two constraints involving x_i and y_i are equivalent to the non-linear inequality

$$R_i \geq |y_i - ax_i - b|.$$

If we fix the variables a and b, the objective function $\sum_i R_i$ is minimized by setting $R_i = |y_i - ax_i - b|$ for every i. In other words, in the optimum solution, each variable R_i is the ***residue*** $|y_i - ax_i - b|$ of the ith point with respect to the regression line $y = ax + b$, and the objective function is the sum of these residues, as required. ■

Rubric: 5 points = 1 for variables + 2 for objective + 2 for constraints. The proof is not required for full credit.

(b) Describe a linear program whose solution (a, b) describes the line with minimum L_∞ error.

Solution:

$$\begin{array}{lll} \text{minimize} & R & \\ \text{subject to} & ax_i + b + R \geq y_i & \text{for all } i \\ & ax_i + b - R \leq y_i & \text{for all } i \end{array}$$

For each index i, the two constraints involving x_i and y_i are equivalent to the non-linear inequality

$$R \geq |y_i - ax_i - b|.$$

Thus, the constraints are collectively equivalent to the inequality

$$R \geq \max_i |y_i - ax_i - b|.$$

For any fixed values of a and b, the variable R is obviously minimized when $R = \max_i |y_i - ax_i - b|$. Thus, in any optimal solution, the objective function is the L_∞ error of the line $y = ax + b$, as required. ■

Rubric: 5 points = 1 for variables + 2 for objective + 2 for constraints. The proof is not required for full credit.