This homework is due Tuesday, November 17, at the start of class. The questions are drawn from the material in the lectures and Chapters 19 and 26 of the text on Fibonacci heaps and maximum flow.

The homework is worth a total of 100 points. When questions with several parts do not specify the points for each part, each part has equal weight.

Remember to write on just one side of a page, do not use scrap paper, put your answers in the correct order, and staple your pages together. If you can’t solve a problem, state this, and write only what you know to be correct. Neatness and conciseness count.

1. **(Lower-bounding heap operations)** (10 points) Suppose only comparisons are allowed on heap keys. Prove that

   Insert or (Extract and (Delete or Minimum))

   must take $\Omega(\log n)$ amortized time on a heap of $n$ elements for all heap implementations.

   (Note: This shows the amortized times for Fibonacci heaps are optimal for comparison-based heaps. In particular, it shows that Insert and Extract cannot both take $O(1)$ amortized time.)

2. **(Structure of Fibonacci heaps)** (15 points) Prove the following.

   **Lemma** For any Fibonacci heap $H$ and any $k \geq 0$, one can construct $H$ so that the Fibonacci tree $T_k$ is rooted at any specified node of $H$ of degree $k$.

   (Note: This shows that the analysis from class of the maximum degree in a Fibonacci heap is tight.)

   (Hint: First show how to construct $T_k$ for any $k \geq 0$ by a series of Fibonacci heap operations.)

3. **(Fibonacci heaps on a pointer machine)** (25 points) The implementation of the Extract operation for Fibonacci heaps uses array indexing to efficiently find nodes of equal degree when consolidating roots. A pointer machine is a restricted model of computation equivalent to a random access machine without arrays. The memory on a pointer machine may hold data structures that contain pointers, but array indexing is not allowed.

   Show how to implement a Fibonacci heap on a pointer machine so all its operations, except for Union, run in the same amortized time as on a random access machine. In particular, show how to implement Extract without using an array so it still runs in $O(\log n)$ amortized time.

   (Note: When you implement Extract without an array, you may need to add new pointer fields to the Fibonacci heap data structure. If other operations besides Extract have to maintain these additional pointers, you will need to discuss how the implementation of the other operations changes as well. Since the solution modifies the data structure, you may need to modify the potential function as well to achieve the same amortized times for all operations. A consequence of your solution to this problem should be that the implementation of a Fibonacci heap does not need to know $f(n)$, the maximum degree in a heap of size $n$.)

4. **(Flow with vertex capacities)** (10 points) Suppose we want to compute a maximum flow in a graph $G = (V, E)$ with source $s$ and sink $t$ where, in addition to edge capacities, every vertex $v \in V - \{s, t\}$ also has a vertex capacity $c(v) \geq 0$. A flow $f$ in graph $G$, in addition to satisfying the edge capacity and flow conservation constraints, also has to
satisfy the following vertex capacity constraint. For all vertices \( v \in V - \{s, t\} \), the flow out of \( v \) must be at most \( c(v) \):
\[
\sum_{w \in V \atop f(v, w) \geq 0} f(v, w) \leq c(v).
\]

Design an efficient algorithm for computing a maximum flow in a graph with vertex capacities. What is the running time of your algorithm?

(5) (Maximum cardinality bipartite matching) (25 points) In this problem, we derive an algorithm for maximum cardinality bipartite matching, from an algorithm for maximum flow. Recall that a matching in an undirected graph is a subset of the edges that are vertex disjoint. In other words, in a matching, no two edges touch the same vertex. The maximum cardinality bipartite matching problem is, given an undirected graph \( G = (V, E) \), where the vertices \( V \) are partitioned into two sets \( X \) and \( Y \) such that all the edges in \( E \) have one endpoint in \( X \) and the other endpoint in \( Y \), to find a matching \( M \subseteq E \) of maximum cardinality \( |M| \).

(a) (10 points) Suppose we want to compute a maximum flow in a graph \( H \) in which all the edge capacities are integers. Prove that \( H \) has a maximum flow \( f \) in which all the flow values \( f(v, w) \) are integers.

(Hint: Show that when the preflow-push algorithm for maximum flow that we learned in class is run on \( H \), it finds an integer-valued flow.)

(b) (15 points) Now suppose we want to compute a maximum cardinality matching in an undirected bipartite graph \( G = (V, E) \) with vertex bipartition \( X, Y \). Consider constructing the following new directed graph \( H \). The vertices of \( H \) are all the vertices of \( G \) plus a new source vertex \( s \) and a new sink vertex \( t \). The edges of \( H \) are as follows. For every undirected edge \( (v, w) \) in \( G \), where \( v \in X \) and \( w \in Y \), there is a directed edge in \( H \) going from \( v \) to \( w \). In addition, from source \( s \) there is a directed edge \( (s, v) \) in \( H \) to every vertex \( v \in X \), and to sink \( t \) there is a directed edge \( (w, t) \) in \( H \) from every vertex \( w \in Y \). Finally, all the edges of \( H \) have capacity 1.

Prove that a maximum flow in graph \( H \) yields a maximum cardinality matching for graph \( G \).

(Hint: Show how to extract a matching of \( G \) of cardinality \( k \) from any integer-valued flow of \( H \) of flow value \( k \), and similarly show how to find a flow in \( H \) of value \( k \) given any matching of \( G \) of cardinality \( k \). Then use Part (a) to argue that there is a maximum flow of \( H \) that is integer-valued.)

(6) (Path and cycle cover) (15 points) Given a directed graph \( G = (V, E) \), a path and cycle cover of \( G \) is an edge subset \( C \subseteq E \) such that the subgraph \( (V, C) \) consists of vertex-disjoint directed paths and directed cycles. In other words, in \( C \), every vertex has both in-degree and out-degree at most 1.

Given a directed graph \( G \) with edge weights \( \omega \), design an algorithm that finds a path and cycle cover \( C \) of maximum total weight \( \sum_{e \in C} \omega(e) \) in \( O(mn + n^2 \log n) \) time, where \( n \) and \( m \) are the number of vertices and edges of \( G \).

(Hint: Reduce the problem of computing a maximum weight path and cycle cover to the problem of computing a maximum weight matching in a bipartite graph.)

(Note: If graph \( G \) is acyclic, then an algorithm for path and cycle cover will find an optimal covering of \( G \) by disjoint paths.)