# Algorithms & Programming: Exam

Chien-Chung Huang (chien-chung.huang@ens.fr)
Pierre Senellart (pierre.senellart@ens.fr)

25 January 2018

The only allowed document is an A4 paper sheet, recto-verso (2 pages), with the content of your choice. This exam lasts three hours and contains three exercises and one problem, and is marked out of 20 points.

*Please write the solution to the problem (D) on a different sheet of paper than the solution to the three exercises (A, B, C), as they will be graded separately.*

## A. Exercise: Summing Subsequences (2 points)

Assume that we are given rational numbers $a_1, \cdots, a_n$. The goal is to find a subsequence $a_i, a_{i+1}, \cdots, a_j$ so that $\sum_{t=i}^{j} a_t$ is minimized. Give an $O(n)$ algorithm to achieve the task.

## B. Exercise: Maximum Flow and Infinite Capacity (2 points)

Given a directed graph $G = (V, E)$ and two vertices $s$ and $t$, assume that the capacity $c : E \to \mathbb{R}^+ \cup \{\infty\}$ allows an edge to have infinite capacity. Prove that as long as there is no $s$-$t$ path along which all capacities are infinite, then the maximum flow exists (i.e., the value of the maximum flow is finite).

## C. Exercise: Circulation (6 points)

Let $G = (V, E)$ be a directed graph $G = (V, E)$, where $u : E \to \mathbb{Z}^+$ and $l : E \to \mathbb{Z}^+$ are the *upper* and *lower* bounds of the edge capacities. (To avoid trivial cases, we can assume that $l(e) \leqslant u(e)$ for all $e \in E$.) We call $f : E \to R^+$ a *circulation* if $f$ satisfies the conditions

$$l(e) \leqslant f(e) \leqslant u(e), \quad \forall e \in E, \tag{1}$$

$$\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e), \quad \forall v \in V, \tag{2}$$

where $\delta^-(v)$ (resp., $\delta^+(v)$) denotes the edges of which $v$ is a end-vertex (resp., a start-vertex).

Not every graph allows a circulation. Your task will be to prove the following:

**Theorem.** *A graph allows a circulation as long as, for all $V' \subseteq V$:*

$$\sum_{e \in \delta^+(V')} u(e) \geqslant \sum_{e \in \delta^-(V')} l(e). \tag{3}$$

Let us give a constructive proof of this theorem by designing an algorithm. Initially, let $f = 0$. Throughout the algorithm, we call an edge $e$ *wrong* if $f(e) < l(e)$. The algorithm consists in eliminating the wrong edges continuously (if possible).

Suppose that $e = (v, w)$ is wrong, let us build an auxiliary graph $G_{v,w}(f)$ as follows. For every edge $e' = (p, q) \in E \backslash \{e\}$, create a directed arc from $p$ to $q$ if $u(e') > f(e')$, with capacity $u(e') - f(e')$; create a directed arc from $q$ to $p$ if $f(e') > l(e')$, with capacity $f(e') - l(e')$. Try to find a directed path $P$ in $G_{v,w}(f)$ from $w$ to $v$. Do the *augmentation* as in the Ford–Fulkerson algorithm along $P$.

1. (1 point) Show that after an augmentation, condition (2) holds.
2. (2.5 points) Show that if there is no path from $w$ to $v$, then some vertex set $V'$ must violate condition (3).
3. (2.5 points) Show that this algorithm stops. Analyze its running time.

## D. Problem: Fibonacci Heaps (10 points)

*Please write the solution to this problem on a different sheet of paper than the solution to the previous exercises, as they will be graded separately.*

A *priority queue* is a data structure to store items along with their *priority value* (some real number; the lower the value, the higher the priority) and that supports the following operations:

**empty()** Construct an empty priority queue.

**insert(item, priority)** Add an item to the queue with its priority value.

**pop()** Retrieve the item with lowest priority value in the queue.

**decrease(item, priority)** Update the priority value of an item, with the condition that the new priority value must be lower than the existing one.

We say a tree is *unranked* if there is no bound on the arity of each node.

A Fibonacci heap is a complex data structure used to implement priority queues. Formally, it is a collection of $t$ unranked ordered rooted trees that verify the *priority heap condition*, meaning that the priority of a node is always less than or equal to the priority of its descendants, with some additional information recorded:

- The collection is stored as a circular doubly linked list of the roots of the trees.
- In each tree, each node has a pointer to one of its children if any and to its parent (if any), and to its left and right siblings (as a circular doubly linked list, so that the right sibling of the rightmost child of a node is the leftmost child).
- Each node in a tree has an additional *marker* which is a Boolean variable initially set to false. The marker is set to true if that node *has lost a child since the last time it changed parent.*
- A special variable *min* holds a pointer to the tree whose root node has the minimum priority value.
- The number $t$ of trees and the degree $\delta(u)$ of each node $u$ are also kept in memory.

1. (0.5 point) Propose an implementation of the empty operation.
2. (1 point) Consider the following *potential function* of a Fibonacci heap: the potential of a Fibonacci heap $H$ is $\phi(H) = \gamma \times (t + 2m)$ where $t$ is the number of trees in the heap, $m$ is the number of marked nodes, and $\gamma$ is a positive constant to be defined further. In what follows we are going to use this potential function to carry out an amortized complexity analysis of the cost of the different operations.
   a) (0.5 point) What is the potential of the empty Fibonacci heap?
   b) (0.5 point) For an operation $x$ that transforms a Fibonacci heap $H$ into a new Fibonacci heap $x(H)$ with cost $c_x(H)$, consider
   $$\hat{c}_x(H) \coloneqq c_x(H) + \phi(x(H)) - \phi(H).$$
   Show that for any sequence of operations $x_1 \dots x_k$ starting from the empty Fibonacci heap $H_0$, with $H_i \coloneqq x_i(H_{i-1})$:
   $$\frac{1}{k}\sum_{i=1}^{k} c_{x_i}(H_{i-1}) \leqslant \frac{1}{k}\sum_{i=1}^{k} \hat{c}_{x_i}(H_{i-1}).$$
   We will therefore call in the following $\hat{c}_x(H)$ the *amortized cost* of operation $x$ and use it to characterize the complexity of operations on Fibonacci heaps.

3. (1 point) Propose a simple implementation of the insert operation whose asymptotic amortized complexity (and worst-case complexity) is $O(1)$.

4. (1.5 points) pop on Fibonacci heaps works as follows: we remove the node pointed to by the *min* pointer and make each of its children the root of a new tree. Then, the list of trees is modified to ensure that the degree $\delta(r)$ of every root $r$ of a tree is distinct: to ensure this, every time two root nodes have the same degree, one is made a child of the other (respecting the priority heap condition), repeatedly until no two root nodes have the same degree. Markers are updated as needed throughout the procedure.

   Show that, by choosing an appropriate value for $\gamma$, the amortized complexity of pop on a Fibonacci heap $H$ can be made to be $O\left(\max\left(\max_{u \text{ node in } H} \delta(u), \max_{u \text{ node in } \mathsf{pop}(H)} \delta(u)\right)\right)$.

5. (2 points) decrease on Fibonacci heaps works as follows: if after updating the priority of a node the priority condition is violated, this node $u$ is cut from its parent $p$ and put as a new root node of a tree in the Fibonacci heap. Subsequently, a *cascading procedure* is applied to the node $p$ as follows: if the node was already marked (meaning it had already lost another child), it is cut from its own parent $p'$ and added as a new root node of a tree in the Fibonacci heap; otherwise, it is marked. The cascading procedure is applied recursively on $p'$.

   Show that, by choosing an appropriate value for $\gamma$ (compatible with the choice already made for the pop operation), the amortized complexity of decrease on Fibonacci heaps can be made to be $O(1)$. It can be useful to introduce the number of times the cascading procedure has been called.

6. (3 points) We are now going to show that the maximum degree of any node in a Fibonacci heap of $n$ nodes is $O(\log n)$.

   a) (1 point) Show that for any node $u$ in a Fibonacci heap with children $u_1, u_2, \ldots u_k$ ordered in the chronological order they were linked to $u$, for any $2 \leqslant i \leqslant k$, $\delta(u_i) \geqslant i - 2$.

   b) (0.5 point) Let $F_k$ be the $k$-th term of the Fibonacci sequence:

   $$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_k = F_{k-1} + F_{k-2} & \text{for } k \geqslant 2. \end{cases}$$

   Show that for any node in a Fibonacci heap, $\sigma(u) \geqslant F_{\delta(u)+2}$ where $\sigma(u)$ is the size of the subtree rooted at $u$.

   c) (1 point) Show that for any $k$, $F_{k+2} \geqslant \left(\frac{1+\sqrt{5}}{2}\right)^k$.

   d) (0.5 point) Conclude that the maximum degree of any node in a Fibonacci heap is $O(\log n)$.

7. (1 point) What is the amortized complexity of Dijkstra's algorithm with a Fibonacci heap implementation for the priority queue? Justify your answer.