Homework #3
( Due: Apr 28 )

\textbf{Det-Compatible-Representatives}( \langle S_1, S_2, \ldots, S_m \rangle, n, f )

(Inputs are \( m \geq 2 \) sets \( S_1, S_2, \ldots, S_m \) of size \( n \geq 1 \) each, and a function \( f \). Function \( f(s_1, s_2, \ldots, s_m) \) with \( s_i \in S_i \) for \( 1 \leq i \leq m \), returns True provided \( s_1, s_2, \ldots, s_m \) are compatible, and False otherwise. This algorithm (i.e., \textbf{Det-Compatible-Representatives}) returns a set of compatible representatives (with one representative from each \( S_i \)) as soon as it finds one, and returns NULL provided no such set exists.)

1. \textbf{for each} \( s_1 \in S_1 \) \textbf{do}
2. \hspace{1em} \textbf{for each} \( s_2 \in S_2 \) \textbf{do}
3. \hspace{2em} \ldots \hspace{2em} \ldots \hspace{2em} \ldots \hspace{2em} \ldots
4. \hspace{1em} \textbf{for each} \( s_m \in S_m \) \textbf{do}
5. \hspace{2em} \textbf{if} \( f(s_1, s_2, \ldots, s_m) = \text{True} \) \textbf{then} \textbf{return} \( \langle s_1, s_2, \ldots, s_m \rangle \)
6. \textbf{return} NULL

\textbf{Task 1.} [ 50 Points ] \textbf{Compatible Representatives}

In this task you are given \( m \geq 2 \) sets \( S_1, S_2, \ldots, S_m \) of size \( n \geq 1 \) each, and you are required to identify one representative \( s_i \) from each set \( S_i \) (\( 1 \leq i \leq m \)) such that \( s_1, s_2, \ldots, s_m \) are compatible as a group. Compatibility is determined by calling a given function \( f \) with \( s_1, s_2, \ldots, s_m \) as input parameters. Function \( f \) returns True provided the group is compatible, and False otherwise. Suppose one can form a total of \( k \) compatible groups from the sets, where \( 0 \leq k \leq n^m \). You need to identify only one of them.

(a) [ 10 Points ] Consider the deterministic algorithm \textbf{Det-Compatible-Representatives} given in the figure above. Argue that the algorithm runs in \( \mathcal{O}((n^m - k)t) \) time, where \( t \) is the worst-case time needed by a single execution of \( f \).

(b) [ 40 Points ] Design a randomized algorithm \textbf{Rand-Compatible-Representatives} that returns a compatible group in \( \mathcal{O}((\frac{n^m}{2m})(m + t) \ln n) \) time w.h.p. in \( n \). Observe that \textbf{Rand-Compatible-Representatives} can be considerably faster than \textbf{Det-Compatible-Representatives}, e.g., if \( t = m = 4 \) and \( k = n^3 \) then \textbf{Det-Compatible-Representatives} runs in \( \mathcal{O}(n^4) \) time (worst-case) while \textbf{Rand-Compatible-Representatives} runs in \( \mathcal{O}(n \ln n) \) time (w.h.p.).

\textbf{Task 2.} [ 90 Points ] \textbf{Faster Randomized Min-Cut}

Consider the randomized min-cut algorithm we saw in the class that returns a min-cut with probability \( \geq 1 - \frac{1}{e} \). Given a connected undirected multigraph with \( n \) vertices, the strategy is to run the following algorithm \( \frac{n^2}{2} \) times and return the smallest cut identified by those runs. Each run uses an algorithm that starts with the original \( n \)-vertex graph and performs a sequence of \( n - 2 \) edge contractions. Each contraction is performed on an edge chosen uniformly at random from the current set of edges. A contraction step contracts the two endpoints of the given edge into a
single vertex and removes all edges between them, but retains all other edges (and thus leading to a multigraph). After \( n - 2 \) contraction steps only 2 vertices remain, and all edges between those two vertices are returned as a potential min-cut.

\((a)\) [10 Points] Argue that each contraction step can be implemented to run in \( \mathcal{O}(n) \) time, and thus the randomized min-cut algorithm described above takes \( \mathcal{O}(n^4) \) time to return a min-cut with probability \( \geq 1 - \frac{1}{e} \).

There is a deterministic min-cut algorithm that can return a min-cut (with certainty) in \( \mathcal{O}(n^3) \) worst-case time. So the randomized algorithm described above runs much slower than the deterministic algorithm and also does not always produce a correct solution! In order to speed up the randomized algorithm we can use the following hybrid approach. Starting with the \( n \)-vertex graph we keep performing random edge contractions until we are able to reduce the number of vertices in the graph to \( r \) for some predetermined \( r < n \). We then apply the deterministic algorithm on that \( r \)-vertex graph to find a min-cut.

\((b)\) [30 Points] Show that a single run of the hybrid algorithm executes in \( \mathcal{O}(n^2 + r^3) \) time, and produces a min-cut with probability at least \( \frac{r^2}{n^2} \).

\((c)\) [30 Points] Show that multiple independent runs of the hybrid algorithm from part \((b)\) can produce a min-cut in \( \mathcal{O}\left(\frac{n^4}{r^2} + n^2r\right) \) time with probability at least \( 1 - \frac{1}{e} \).

\((d)\) [5 Points] What value of \( r \) produces the best running time for the algorithm in part \((c)\)?

\((e)\) [15 Points] Use the algorithm from part \((c)\) with the value of \( r \) from part \((d)\) to design a Monte-Carlo algorithm that runs asymptotically faster than the best deterministic algorithm (i.e., faster than \( \Theta(n^3) \)) and can produce a min-cut w.h.p. in \( n \).

Task 3. [60 Points] Cluster of Multicores

The following problems involve load-balancing on a cluster of multicore machines.

\((a)\) [20 Points] Suppose you have bought \( n \) (\( \gg 1 \)) multicore machines for \( n \) remote users. Whenever a user has a job he/she chooses a machine uniformly at random and submits the job to that machine. A user can submit and run only one job at a time. Assuming that all \( n \) machines can run in parallel, and a \( k \)-core machine can execute \( k \) jobs in parallel (i.e., one job per core), show that w.h.p. in \( n \) each job can start running as soon as it is submitted provided each machine has at least \( \frac{2 \ln n}{\ln \ln n} \) cores.

\((b)\) [20 Points] Consider the setting in part \((a)\), but suppose now you have \( 2n \ln n \) remote users. Show that in this case w.h.p. in \( n \) each job can start running as soon as it is submitted provided each machine has at least \( 6 \ln n \) cores.

\((c)\) [20 Points] Consider the setting in part \((b)\). Show that if all users submit jobs simultaneously then w.h.p. in \( n \) no machine will remain idle.