CS 787: Advanced Algorithms	**Homework 1**
Out: 09/19/07	**Due: 10/10/07**

Guidelines

- This homework consists of a few exercises followed by some problems. The exercises are meant for *your practice only*, and do not have to be turned in. You are required to turn in the problems. We will provide solutions to all the questions.
- Collaboration is not allowed. Some of the problems are difficult, so please get started early. Late submissions do not get any credit.

Exercises

1. Given a weighted graph $G = (V, E)$ with weights c_e on edges, the maximum bottleneck path from a vertex s to a vertex t is an s-t path P that maximizes $\min_{e \in P} c_e$. The goal in the **maximum bottleneck path** problem is to find the maximum bottleneck path for *every* pair of vertices in the graph. Prove that the collection of maximum bottleneck paths is a tree and give an $O(m \log n)$ time algorithm for finding this tree.
2. Prove Hall's theorem.
3. Consider the following variant of 3-SAT called **Not-All-Equal-SAT**: Given a 3-CNF formula, decide whether there exists an assignment to the variables of the formula such that every clause contains at least one literal that is true and at least one literal that is false. Show that this problem is NP-complete.
4. A coin is said to have bias p if upon tossing it the probability that it comes up heads is p and the probability that it comes up tails is $1 - p$. In algorithm design, we are often interested in coins of bias $1/2$, but natural sources of randomness (e.g. whether the number of solar flares in a given year is even or odd) can be quite skewed.
   (a) Given a coin C of bias p, give an algorithm for generating a coin toss with bias $1/2$, using as few tosses of C (in expectation) as possible. Can you design an algorithm that **does not** require knowledge of p?
   (b) Sometimes, to the contrary, we have available a coin of bias $1/2$, but require a coin of bias p for some value $p < 1/2$. Give an algorithm for generating a coin toss of bias p using a coin C with bias $1/2$ using as few tosses of C as possible.
   (c) Given a deck of n cards, you want to shuffle the cards so that any permutation of the cards is equally likely. Show how to do this using a coin of bias $1/2$. How many coin tosses do you require?

Problems

1. Recall that in the **min-cost max-flow (MCMF)** problem, we are given a directed graph $G = (V, E)$, with source s, sink t, edge capacities c_e, and per-unit-flow costs ℓ_e. Our goal is to find a maximum s-t flow in the graph with the minimum possible cost, where the cost of a valid flow f is defined as $\ell(f) = \sum_e f_e \ell_e$.

   We will solve this problem using the **min-cost circulation (MCC)** problem. A valid circulation is a flow f that satisfies capacity constraints, as well as flow-conservation at *every* node, including s and t. In the MCC, our goal is to come up with a valid circulation f minimizing the cost $\ell(f)$. Note that we do not require a "maximum" circulation; if the graph has no negative-cost cycles, then the optimal solution to the problem is a circulation that is zero on every edge.

(a) Give a polynomial-time reduction from MCMF to MCC.

Consider the following natural variant of Ford-Fulkerson for the MCC: Start with a zero flow, $f = 0$; maintain an appropriate residual graph G_f at every step; find a negative-cost cycle in G_f; saturate this cycle with flow, add it to f, and recurse; the algorithm terminates when no negative-cost cycles are found.

(b) Specify how to set costs and capacities on the residual graph G_f.

(c) Prove that the algorithm always terminates.

(d) Prove that this algorithm solves the MCC correctly. *(Hint: Suppose that a different circulation f^* has smaller cost than f. Use f^* and f to find a witness to contradict the fact that the algorithm terminated.)*

2. In the **selection** problem we are given an unsorted list of n distinct elements, and asked to find the k-th largest element in this list. Recall the following randomized selection algorithm from class:

   SELECT(An unordered list A, and an integer k) /* Finds the k-th largest element in list A */

   (a) Pick an element of A uniformly at random, say x.

   (b) Partition A into $A_<$ and $A_>$, where $A_<$ contains all the elements in A smaller than x and $A_>$ contains all the elements in A larger than x.

   (c) If $|A_>| = k - 1$, then **return** x. Else if $|A_>| \geq k$ then run SELECT($A_>, k$). Else run SELECT($A_<, k - |A_>| - 1$).

   Prove that the expected running time of this algorithm is $O(n)$. *(Hint: Divide the process of selection into $\log n$ rounds, where in round i the size of the list under consideration is between 2^i and 2^{i+1}, and analyze the expected running time of each round separately.)*

3. In the **Unique Set Cover** problem, we are given a set of n elements and a collection of m subsets of the elements. Our goal is to pick out a number of subsets so as to maximize the number of **uniquely covered** elements, those that are contained in exactly one of the picked subsets. (Note that the cost or number of subsets picked is not important).

   (a) Consider the following naïve algorithm—for some number $p < 1$, the algorithm picks each subset independently with probability p. Assuming that every element is contained in exactly F subsets, compute the expected number of elements uniquely covered. For what value of p is this expectation maximized?

   (b) Assuming that each element is contained in at most F subsets and at least $F/2$ subsets, give a constant factor approximation to unique set cover using the algorithm from part (a).

   (c) Extend the algorithm from part (b) to obtain an $O(\log m)$ approximation in general (without assumptions on the frequency of any element). *(Hint: Try reducing this problem to the one in part (b).)*

   (d) Can you improve the approximation from part (c) to a factor of $O(\log n)$?

4. Consider the following variant of the game "Six degrees of Kevin Bacon": the game starts with a set X of n actresses and a set Y of n actors, and two players P_0 and P_1. Player P_0 names an actress $x_1 \in X$, player P_1 names an actor y_1 who has appeared in a movie with x_1, player P_0 names an actress x_2 who has appeared in a movie with y_1, and so on. Thus, P_0 and P_1 collectively generate a sequence $x_1, y_1, x_2, y_2, \cdots$ such that each actor/actress in the sequence has costarred with the actress/actor immediately preceding. A player $P_i (i = 0, 1)$ loses when it is P_i's turn to move, and he/she cannot name a member of his/her set who hasn't been named before.

   Suppose you are given a specific pair of such sets X and Y, with complete information on who has appeared in a movie with whom. Give a polynomial-time algorithm for determining which of the two players P_0 or P_1 can force a win in this instance of the game. Also give a winning strategy for that player (that is, an algorithm that takes a current sequence $x_1, y_1, x_2, y_2, \cdots$ and generates a legal next move for the player).

5. Consider the following family of hash functions from $[m]$ to $[n]$, where $n < m$, and p is a prime larger than m.

$$h_{a,b}(x) = ((ax + b) \mod p) \mod n$$

Prove that $\{h_{a,b}\}_{a \in [p] \setminus \{0\}, b \in [p]}$ is a 2-universal family of hash functions.

(Note: $[a]$ denotes the set $\{0, \cdots, a-1\}$.)

6. In this question we analyze a contention resolution system in a balls-and-bins framework. We have n resources (bins) and n jobs (balls); we allocate resources to jobs in rounds. In the first round, we throw all the n balls into n bins u.a.r. After round $i \geq 1$, we remove every ball that occupies a bin by itself in round i, and in the next round $i + 1$ we throw the remaining balls into the n bins. (One can imagine the lonely balls getting service, whereas none of the colliding ones receive service.) The process continues until no balls are left.

Prove that this process runs for at most $c \log \log n$ rounds with high probability, where c is some constant.

(Hint: Keep track of the number of balls remaining after every round.)