Qualifying Examination

Theoretical Computer Science

Saturday, February 24, 2007

Part I: Algorithms

ID Number	
Pseudonym	

Problem	Maximum Points	Points Earned	Grader
1	25		
2	25		
3	25		
4	25		
Total	100		

Instructions:

1. This is a closed book exam.
2. The exam is for 3 hours and has four problems of 25 points each. Read all the problems carefully to see the order in which you want to tackle them.
3. Write clearly and concisely. You may appeal to some standard algorithms/facts from text books unless the problem explicitly asks for a proof of that fact or the details of that algorithm.
4. If you cannot solve a problem, to get partial credit write down your main idea/approach in a clear and concise way. For example you can obtain a solution assuming a clearly stated lemma that you believe should be true but cannot prove during the exam. However, please do not write down a laundry list of half baked ideas just to get partial credit.

May the force be with you.

Problem 1:

(A) **Finding median quickly.**
[10 Points]

Given u *sorted* arrays with total size n, show how one can compute the median of the set formed by the union of these arrays, in $O(u \log^{O(1)} n)$ time. If your algorithm is randomized then this bound on the running time should hold in expectation.

(B) **k-medians.**
[15 Points]

An (unsorted) array $\mathcal{S}$ with n entries, and a sequence of k queries $Q_1, \ldots, Q_k$ is provided in advance. A query $Q_j = [l_j, r_j]$ is an interval of the array (starting from position l_j and ending at r_j). The task at hand is to compute the median element in the set $\mathcal{S}[l_j \ldots r_j]$, for $j = 1, \ldots, k$. We will refer to the problem of computing these k numbers as the *k-medians range problem.* (Note, that the query intervals are not necessarily disjoint. In fact, if the intervals are disjoint, the problem is considerably easier [and we wouldn't want this, would we?].)

Show, using (A), how to solve this problem in $O(n \log n + \text{poly}(k, \log n))$ time, where $\text{poly}(\cdot)$ is a constant degree polynomial.

Note, that the fastest algorithm currently known to this problem has running time $O(n \log k + k \log k \log^2 n)$. The closer your algorithm running time is to this bound, the better (note however, that reducing the term $O(n \log n)$ to $O(n \log k)$ is hard, so you might want to concentrate on the second term).

(C) **k-medians lower bound.** (hard)
[Bonus Points] Show that any algorithm that solves the k-medians range problem requires $\Omega(n \log k)$ time in the comparison model.

Problem 2: Given a directed graph $G = (V, A)$, two nodes $s, t \in V$ and an integer $k \leq |V|$, design a polynomial time algorithm to check if there is an *s-t walk* in G that visits at least k distinct nodes (including s, t). A walk differs from a path in that it can visit a node multiple times.

(A) **[15 Points]** Show how to solve the problem when G is a directed acyclic graph (DAG).

(B) **[10 Points]** Use the DAG algorithm to solve the problem for arbitrary directed graphs.

Problem 3: The k-median problem on an undirected graph $G = (V, E)$ with non-negative *integer* edge weights $w : E \to Z$ is the following. Choose k nodes $S \subseteq V$ to minimize $\sum_{v \in V} d(v, S)$ where $d(v, S)$ is the distance of node v to the closest node in S - distance is the shortest path distance according to the edge weights. Your goal is to obtain a polynomial time algorithm for this problem when G is a tree $T = (V, E)$.

(A) **[5 Points]** Give a polynomial time reduction to show that the problem on a tree with arbitrary degree can be reduced to a problem in which the tree has bounded degree.

(B) **[15 Points]** Give an algorithm to find an optimum solution when the tree T has bounded degree. Your algorithm needs to be polynomial in $n = |V|$ and $B = \sum_{e \in E} w(e)$ (thus it is pseudo-polynomial in the input size). You should not solve this part if you can do (C).

(C) **[20 Points]** Solve (B) with an algorithm that is polynomial in the input size (the running time should be polynomial in n and $\log B$).

You may solve (B) or (C) when T is a path to get some partial credit.

Problem 4: We consider the interval packing problem to illustrate a useful algorithmic paradigm in randomized algorithms. We are given n intervals $I_1, I_2, \ldots, I_m$ where $I_i = (\ell_i, r_i)$, $\ell_i, r_i \in \mathcal{Z}^+$ (integer coordinates). Assume for simplicity that the intervals are in the range $[0, m]$. Each interval has a non-negative weight w_i and the goal is to find a maximum weight subset of the intervals that do not overlap. This can be solved by dynamic programming optimally but we will try a different approach. We can write an integer program for this problem by introducing a $\{0,1\}$ variable x_i for each interval I_i which indicates whether we select I_i or not. Then we can write the program as:

$$\begin{aligned} \max \sum_i w_i x_i & & & \\ \sum_{i: j \in I_i} x_i &\leq 1 & & j = 0, 1, \ldots, m \\ x_i &\in \{0,1\} & & i = 1, 2, \ldots, n. \end{aligned}$$

The constraint states that at each integer point $j \in [0, m]$, only one interval can contain it. We obtain a linear program by relaxing the constraint $x_i \in \{0,1\}$ to $x_i \in [0,1]$ which can then be solved in polynomial time. In fact it is known that the LP has integral vertex solutions but assume you didn't know that. Let x be a fractional solution to the linear program and you wish to "round" it to an integer solution.

(A) **[5 Points]** Suppose you tried a simple randomized rounding idea. Pick each interval with probability x_i independently. If the solution is feasible (no chosen intervals overlap) you output the intervals, otherwise you output the empty solution. Give an example of an instance and a feasible fractional solution to the above LP (not necessarily an optimal solution) that shows that this algorithm will yield a solution whose expected weight is arbitrarily small (smaller than any given constant) when compared to the weight of the fractional solution.

(B) **[5 Points]** Given k numbers $r_1, r_2, \ldots, r_k$ from $[0, 1/2]$ such that $\sum_{i=1}^{k} r_i \leq 1/2$ prove that $\prod_{i=1}^{k}(1 - r_i) \geq c$ for some constant c that is independent of k. **Hint:** Prove that $1 - x \geq e^{-\alpha x}$ for the range in question and for a sufficiently large α.

(C) **[15 Points]** Now consider a refinement of the randomized rounding approach. We change the randomization step slightly. We pick each interval I_i independently with probability $x_i/2$ instead of x_i. Let A be the chosen intervals in the random step. We obtain a feasible solution B from A as follows. Sort intervals in A in increasing value of their left end points (that is ℓ_i) with ties broken arbitrarily. Consider the intervals in A one by one in this sorted order and when considering an interval I_i, include it in B if it does not overlap with any previously chosen interval in B. Thus B results in a feasible solution and you output B. Prove that the expected weight of the intervals in B is within a constant factor of the weight of the fractional solution. You might want to consider two events Y_i and Z_i for each interval I_i. Y_i is the event that I_i is chosen in A and Z_i is the event that I_i is chosen in B *conditioned* on the event that I_i is chosen in A. What is the probability of Z_i?

Qualifying Examination

Theoretical Computer Science

Saturday, February 24, 2007

Part II: Automata and Complexity

ID Number	
Pseudonym	

Problem	Maximum Points	Points Earned	Grader
1	25		
2	25		
3	25		
4	25		
Total	100		

Instructions:

1. This is a closed book exam.
2. The exam is for 3 hours and has four problems of 25 points each. Read all the problems carefully to see the order in which you want to tackle them.
3. Write clearly and concisely. You may appeal to some standard algorithms/facts from text books unless the problem explicitly asks for a proof of that fact or the details of that algorithm.
4. If you cannot solve a problem, to get partial credit write down your main idea/approach in a clear and concise way. For example you can obtain a solution assuming a clearly stated lemma that you believe should be true but cannot prove during the exam. However, please do not write down a laundry list of half baked ideas just to get partial credit.

May the force be with you.

Problem 5: Recall the following definitions and observations

- A *monoid* $\mathbf{M} = (M, \circ, i)$ is a set M equipped with a binary associative operation $\circ$, such that $i \in M$ is the identity (both left and right) with respect to $\circ$. A monoid $\mathbf{M}$ is said to be *finite* if M is finite.
- $(\Sigma^*, \cdot, \epsilon)$ forms a monoid, where Σ^* is the collection of all (finite) strings over (finite) alphabet Σ, $\cdot$ is string concatenation, and ϵ is the empty string.
- A *monoid morphism* from $\mathbf{M_1} = (M_1, \circ_1, i_1)$ to $\mathbf{M_2} = (M_2, \circ_2, i_2)$ is a function $f : M_1 \to M_2$ that preserves products and identity. In other words, $f(i_1) = i_2$ and for all $m_1, m_2 \in M_1$, $f(m_1 \circ_1 m_2) = f(m_1) \circ_2 f(m_2)$.
- For a function $f : A \to B$ and a set $C \subseteq B$, $f^{-1}(C) = \{a \in A \mid f(a) \in C\}$.

A language $L \subseteq \Sigma^*$ is said to be *recognized* by finite monoid $\mathbf{M} = (M, \circ, i)$, if there is monoid morphism f from $(\Sigma^*, \cdot, \epsilon)$ to $\mathbf{M}$ and a set $F \subseteq M$ such that $L = f^{-1}(F)$. Finally we will say a language L is *recognizable* if there is some finite monoid $\mathbf{M}$ that recognizes L. **Show** that L is recognizable iff L is regular (i.e., has a DFA accepting L).

Problem 6: Recall that a (total) function $f : \mathbb{N} \to \mathbb{N}$ is said to be *recursive*, if there is a Turing machine M such that on input i, M halts with $f(i)$ on its tape; M is said to compute f. Show that the collection of all recursive functions cannot be recursively enumerated, i.e., there is no r.e set $A \subseteq \mathbb{N}$ such that for every recursive function f, there is $i \in A$ such that M_i computes f and for any $i \in A$, M_i computes a (total) function.
Please Note: Recursive enumeration of the class of all recursive functions only requires that *some* Turing machine computing f be enumerated, for every f. Hence (modified) Rice's theorem does not apply here, as that requires *all* relevant Turing machines be enumerated.

Problem 7: Consider two well-known optimization problems Max-Cut and Max-2SAT. In Max-Cut we are given an undirected simple graph $G = (V, E)$ and the goal is to find a set $S \subseteq V$ so as to maximize the number of edges between S and $V \setminus S$. In Max-2SAT we are given a 2-CNF formula ϕ (each clause of ϕ has at most 2 literals) and the goal is to find an assignment to the variables to maximize the number of clauses that are satisfied. Note that there is a polynomial time algorithm to check if ϕ can be satisfied (unlike 3SAT which is NP-Complete) however the optimization problem is NP-hard.

Assuming that Max-Cut is NP-hard to approximate to within a $(1 - \epsilon)$ factor for some fixed constant $\epsilon > 0$, prove that Max-2SAT is NP-hard to approximate within a $(1-\delta)$ factor for some other fixed constant $\delta > 0$. **Hint:** Express Max-Cut as a constraint satisfaction problem to reduce it to Max-2SAT. Also use the fact that any instance of Max-Cut has a solution of size at least $|E|/2$ (briefly justify this fact if you use it in your proof but there is no need to give a formal proof).

Problem 8: We consider two restrictions to the class $\mathbf{P}^{\mathbf{NP}}$.

- Let $\mathbf{P}_{||}^{\mathbf{NP}}$ denote the class of languages that can be decided by a deterministic polynomial time Turing machine, with access to an oracle for any **NP** language (say for SAT), *with the restriction that it must submit all its (polynomially many) queries to the oracle together* (that is it cannot base its second query on the answer to the first query and so forth).
- Let $\mathbf{P}^{\mathbf{NP}[\log]}$ denote the class of languages that can be decided by a deterministic polynomial time Turing machine, with access to an oracle for any **NP** language (say for SAT), *with the restriction that it can query the oracle at most $O(\log n)$ times* (n being the input length). Note that the queries here can be adaptive, that is the second query can be based on the answer to the first query and so on.

Show that both these restrictions have the same effect. That is $\mathbf{P}_{||}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}[\log]}$.
Hint: To show $\mathbf{P}_{||}^{\mathbf{NP}} \subseteq \mathbf{P}^{\mathbf{NP}[\log]}$, it is useful to observe that if one knows the exact number of queries (among the polynomially many parallel queries asked) that will answered positively by the **NP** oracle, then a single query to SAT will determine whether the input is accepted.