15-251: Great Theoretical Ideas In Computer Science

Recitation 14 Solutions

PRIMES $\in$ NP

The set of PRIMES of all primes is in co-NP, because if n is composite and $k \mid n$, we can verify this in polynomial time. In fact, the AKS primality test means that PRIMES is in P. We'll just prove PRIMES $\in$ NP.

(a) We know n is prime iff $\phi(n) = n-1$. Additionally, if n is prime then $\mathbb{Z}_n^*$ is cyclic with order $n-1$. Given a generator a of $\mathbb{Z}_n^*$ and a (guaranteed correct) prime factorization of $n-1$, can we verify n is prime?

If a has order $n-1$, then we are done. So we must verify a has order dividing $n-1$:

$$a^{n-1} \equiv 1 \bmod n$$

and doesn't have smaller order, i.e. for all prime $p \mid n-1$:

$$a^{(n-1)/p} \not\equiv 1 \bmod n$$

a is smaller than n, and there are at most $\log(n)$ factors each smaller than n, so our certificate is polynomial in n's representation.

Similarly, each arithmetic computation is polynomial in $\log(n)$, and there are $O(\log(n))$ of them.

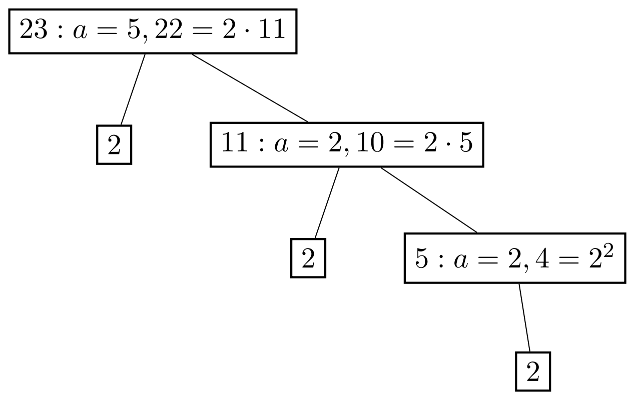
(b) What else needs to be verified to use this as a primality certificate? Do we need to add more information?

We also need to verify that our factorization $(p_1, \alpha_1), (p_2, \alpha_2), \ldots, (p_k, \alpha_k)$ of $n - 1$ is correct, two conditions:

1. $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k} = n - 1$ can be verified by multiplying and comparing, polynomial in $\log(n)$.

2. All of $p_1, p_2, \ldots, p_k$ are prime.

To verify this second condition, we are faced with the problem we started with, but for smaller primes. The natural extension is to create a tree of primality certificates:



By induction on n, the size of the tree (excluding leaves) is bounded by $4 \log(n)$, and $O(\log^2(n))$ bits are stored at each node. So our certificate is polynomial in n's representation, and since our time bounds from part (a) apply at each node, verification is poly-time as well.

Approximation Algorithms

(a) Given a graph on n vertices that we know to be 3-colorable (but not 2-colorable), we wish to give a coloring that uses as few colors as possible. Give a polynomial algorithm that is a $\sqrt{n}$-approximation for this minimization problem.
Hint: We can 2-color a 2-colorable graph in polynomial time using the greedy algorithm, and if the maximum degree in a graph is Δ, we can greedily $(\Delta + 1)$-color it in polynomial time as well.

Fun fact: Unless $P = NP$, not only is there no c-approximation for any constant c, but also there exists some $\epsilon > 0$ such that there is no $O(n^\epsilon)$-approximation.

Algorithm:
If the max degree of G is at least $\sqrt{n}$, pick a vertex v with degree $\geq \sqrt{n}$. 2-color the neighborhood of v (using two colors that have not been used before). Remove $N(v)$ and continue on the rest of the graph. Note that n is fixed as the number of vertices in the original graph, not the order of the subgraph we recursively call on.
If the max degree is less than $\sqrt{n}$, greedily color it with $\sqrt{n}$ new colors.

This gives a valid coloring because we use new colors at each iteration, and the colorings at each iteration are valid. In particular, we know we can 2-color the neighborhood of any vertex because the input graph is 3-colorable and that vertex must be colored differently than everything in its neighborhood.

Next, notice that we recursively 2-color at most $n/\sqrt{n} = \sqrt{n}$ times, as each time we remove at least $\sqrt{n}$ vertices. Finally, we color the rest of the graph with $\sqrt{n}$ colors, thus coloring the whole graph with at most $3\sqrt{n}$ colors. We know the optimal coloring uses exactly 3 colors, so we have a $\sqrt{n}$-approximation as desired.

Finally, the algorithm is polynomial because there are at most $\sqrt{n}$ iterations, each of which take polynomial time (using greedy algorithms).