Algorithms – CMSC-27200
http://alg15.cs.uchicago.edu
Homework set #7. Posted 2-19. Due Wednesday, February 25, 2015
The following problems were updated on 2-20: 7.1(a), 7.1(b), 7.9(b)

Read the homework instructions on the website. The instructions that follow here are only an incomplete summary.

Hand in your solutions to problems marked "HW." Do not hand in problems marked "DO." If you hand in homework marked **"XC" (extra credit)**, do so on **separate and separately stapled sheets, please**. PRINT YOUR NAME and SECTION NUMBER ON EVERY SHEET you submit. **Use LaTeX to typeset your solutions**. Hand in your solutions on paper, do not email.

When writing pseudocode, **explain the meaning of your variables.** Use comments to explain what is happening in each line. Also, give a short explanation of the idea behind the algorithm. **Unless otherwise stated, describe your algorithms in pseudocode. Elegance of your code matters.**

Carefully study the policy (stated on the website) on collaboration, internet use, and academic integrity. **State collaborations and sources both in your paper and in email to the instructors.**

7.1 **(Fermat test)** A number p is *composite* if p is not prime and not 0 or ± 1. We say that a number a is a *Fermat witness* for the number p if $\gcd(a, p) = 1$ and $a^{p-1} \not\equiv 1 \pmod{p}$. Observe that if $p \geq 2$ and p has a Fermat witness then p is composite (by Fermat's little Theorem).

(a) **HW (8 points)** Composite numbers that do not have a Fermat witness are called *Carmichael numbers*. In other words, a number $p \geq 2$ is a Carmichael number if p is composite yet $(\forall a)($ if $\gcd(a, p) = 1$ then $a^{p-1} \equiv 1 \pmod{p}$. Prove that the number $p = 561$ is a Carmichael number. $(561 = 3 \cdot 11 \cdot 17)$. — Hint: verify the congruence $a^{560} \equiv 1$ separately modulo 3, modulo 11, and modulo 17.

(b) **XC (4 points)** Let q, r be distinct primes. Prove: qr is not a Carmichael number. – Use the following fact. Def: Let p be a prime. An integer g is a *primitive root mod* p if $g \not\equiv 0 \pmod{p}$ and for $1 \leq j \leq p-2$ we have $g^j \not\equiv 1 \pmod{p}$. For instance, 3 is a primitive root modulo 7 but 2 is not. Theorem: For every prime p there exists a primitive root mod p.

(c) **XC (6 points)** Assume $p \geq 2$ is not a Carmichael number. Let $F(p)$ denote the set of integers between 0 and $p-1$ that are relatively prime to p. Pick $a \in F(p)$ at random. Prove: with probability at least $1/2$, the number a is a Fermat witness for p. (So if there is a Fermat witness then there are so many that it will be easy to find one just by random sampling.)

7.2 DO **(Greedy algorithm for minimum-cost spanning tree)** Let $G = (V, E, w)$ be a connected weighted undirected graph where $w : E \to \mathbb{R}$ is the weight function. We view E as a set of unordered pairs. The cost of a spanning tree is the sum of the weights of its edges.

The greedy algorithm collects a set T of edges for the minimum-cost spanning tree. Initially T is empty and at every stage, the graph (V, T) is a forest (cycle-free graph). (The latter is a loop invariant.)

```
0   T := ∅        [initialize]
1   while  the forest (V,T) is disconnected
2       split the set of connected components of (V,T)
            into two non-empty parts, say A and B
3       pick an edge e ∈ E of minimum weight among the edges
            connecting A to B
4       add e to T
5   end(while)
6   return  T
```

Note that there is a large degree of freedom in this algorithm: in each round we can choose an arbitrary partition (A, B) of the set of connected components of the current forest (V, T) (line 2). We also may have some choice in line 3 (if there are ties among the edge weights).

Prove: regardless of the choices made in lines 2 and 3, the algorithm always returns a minimum-cost spanning tree.

Note: details of implementation (and running time) very much depend on the choices made in Line 2. The choice made by Jarník's (1930) (a.k.a. Prim's, 1967) algorithm is to make all but one of the connected components of (V, T) be isolated vertices.

7.3 DO **(Jarník's algorithm)**: Implement Jarník's algorithm for minimum-cost spanning tree in pseudocode. Your code should be identical with Dijkstra's except a small difference in the RELAX routine. What is the difference? Explain why this code really implements Jarník's algorithm. State the three data-structure operations used.

7.4 DO **(uniqueness of minimum-cost spanning tree)** Prove: if all edge weights are distinct then the minimum-cost spanning tree is unique.

7.5 DO **(tree update)** Let T be a minimum-cost spanning tree of the (connected, undirected) graph $G = (V, E)$. Let us now reduce the weight of an edge $e \in E$. Show how to update T in linear time. (T is given.)

7.6 **(minimum max-cost spanning tree)**: The max-cost of a spanning tree is the maximum of the weight of its edges. We seek to minimize this quantity; we call a spanning tree that is optimal for this measure a "minimum max-cost spanning tree."

(a,b) Prove that a minimum-cost spanning tree is necessarily a minimum max-cost spanning tree, but not conversely.

(c) **XC (5 points)** Show that a minimum max-cost spanning tree can be found in linear time.

7.7 **(Fibonacci numbers)**: Recall the definition of Fibonacci numbers: $F_0 = 0, F_1 = 1$ and $F_n = F_{n-1}+F_{n-2}$. So the sequence is $0, 1, 1, 2, 3, 5, 8, 13, \ldots$. Recall that

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right). \tag{1}$$

Note that $\gamma = (1+\sqrt{5})/2 \approx 1.618$ is the Golden Ratio.

(a) DO: Prove: $F_n \sim (1/\sqrt{5})\gamma^n$.

(b) **HW (3 points)** Let d_n denote the number of binary digits of F_n. Asymptotically evaluate d_n. Your answer should be a relation of the form $d_n \sim an^b c^n$. Determine the constants a, b, c. You may use part (a) without proof.

(c) **HW (3 points)** Prove that F_n cannot be computed in polynomial time. (The input is the number n in binary.)

(d) **XC (4 points)** Prove that given the positive integers n and m in binary, the number ($F_n \bmod m$) can be computed in polynomial time. Your algorithm should be very elegant with reference to an algorithm studied in class. No pseudocode needed. (Hint: study the powers of the matrix $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$. Observe a pattern, prove by induction.)

7.8 Study the "Greedy coloring" handout (GCH).

(a) DO: GCH (a) (Greedy coloring is not so bad)

(b) **XC (4 points)** GCH (b) (Greedy coloring is terrible)

(c) **HW (3 points)** GCH (c) (Greedy coloring can be optimal)

(d) **HW (6 points)** GCH (d) (Implement greedy coloring in linear time)

(e) Research problem (discuss with L.B. if you are interested). How bad is greedy coloring of we first randomly relabel the vertices? Do there exist bipartite graphs that will, with high probability, require more than n^c colors for a constant $c > 0$?

7.9 **(Coloring planar graphs)**

(a) DO: Find a planar graph in which at least 100 vertices have degree ≥ 100.

(b) **HW (4+4 points)** Give an efficient algorithm to color every planar graph with at most 6 colors. Use the following theorem: Every planar graph has a vertex of degree ≤ 5. (b1) Describe your algorithm in high-level language like the description of the greedy coloring algorithm in the Handout. (b2) Implement your algorithm in linear time.

Note: The 4-Color Theorem says that every planar graph is 4-colorable (it can be legally colored using at most 4 colors). Moreover, it is known that such a coloring can be found in polynomial time. That is a very complicated algorithm, however.

7.10 DO: Study the "Greedy matching" handout. Solve the problems stated at the end of the handout.

7.11 DO: The *linear programming (LP) feasibility* problem takes as input a list L of linear inequalities of the form $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$. (This is the

i-th inequality.) The a_{ij} and b_j are the coefficients of L. We say that L is *feasible* if there exist real numbers $x_1, \ldots, x_n$ that satisfy all the inequalities in L.

The *integer LP (ILP) feasibility* problem asks whether or not there is a solution to this system in which the variables take integer values.

The $(0,1)$-*LP feasibility* problem asks whether or not there is a solution to this system in which the variables take values $0, 1$ only.

(a) Prove that the following LP is (a1) feasible but (a2) not ILP-feasible (has no solution in integers):

$$\begin{aligned} x_1 + 5x_2 + 2x_3 &\leq 4 \\ 2x_1 - 2x_2 + x_3 &\leq 1 \\ -2x_1 - 2x_2 - 2x_3 &\leq -3 \end{aligned}$$

(b) Karp-reduce 3-colorability of graphs to $(0,1)$-LP feasibility. Explanation: Given a graph G, you need to construct in polynomial time an LP $L(G)$ with integer coefficients such that G is 3-colorable if and only if $L(G)$ is $(0,1)$-feasible.