Algorithms – CMSC-27200
http://alg15.cs.uchicago.edu
Homework set #5 due February 11, 2015
Loop invariant in Problem 5(a) updated 2-15.

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.**

In this problem set, every digraph is given in an *adjacency-list* representation, unless expressly stated otherwise. Recall that an adjacency-list representation consists of an array of vertices, where vertex u is the start of the linked list $\mathrm{Adj}[u]$ that lists all the out-neighbors of u in some order. An algorithm on a digraph $G = (V, E)$ is said to run in ***linear time*** if the number of steps is $O(|V| + |E|)$.

5.0 Remember to hand in your solution to problem 4.4 at this time. (If you handed it in already – ahead of time, please hand it in again to reduce our administrative burden.)

5.1 DO **(Little-oh vs. logarithms)** Assume $a_n, b_n > 1$. Consider the following two statements.
(C) $a_n = o(b_n)$ and (D) $\ln a_n = o(\ln b_n)$.
Recall for problem 4.1 that (D) $\not\Longrightarrow$ (C).

(a) Prove: if $a_n \geq 1.0001$ then (D) $\Longrightarrow$ (C).

(b) Use part (a) to give a simple proof of the statement that

$$2^{(\lg n)^{1/2}} = o(n^{1/10}). \tag{1}$$

(This was problem 4.2.)

5.2 DO **(Optimal walk)** (Problem updated 2-9-2015 8pm: "zero-weight cycles" added. This fixes an error pointed out by Clayton Norris on Piazza.)

Consider a weighted rooted digraph $G = (V, E, w, s)$ where $s \in V$ is a specified source vertex (also referred to as the "root") and $w : E \to \mathbb{R}$ is the weight function. A *negative cycle* is a cycle whose edge weights add up to a negative number. A *zero-weight cycle* is a cycle whose edge weights add up to zero.

Recall that we say that a subset A of the vertices is *accessible* if there is a path from the source to at least one vertex in A. Accessible negative cycles are a source of trouble for min-cost path problems.

Recall that a *walk* is allowed to repeatedly visit the same vertex. An i-step walk from s to v is a sequence $s = u_0 \to u_1 \to \cdots \to u_i = v$ of vertices u_j where $(u_{j-1}, u_j) \in E$ for $j = 1, \ldots, i$. We call i the *combinatorial length* of this walk to avoid confusion with the *cost* of the walk, $\sum_{j=1}^{i} w(u_{j-1}, u_j)$. This latter quantity (the cost) is often, confusingly, referred to as the "length" of the walk; we avoid this terminology and stick with the term "cost of the walk" for this quantity.

Let W be an $s \to \cdots \to v$ walk of combinatorial length $\leq i$ such that W has minimum cost among all $s \to \cdots \to v$ walks of combinatorial length $\leq i$. (a) Prove: if there is no accessible negative cycle in G and no accessible zero-weight cycle then W is a *path* (has no repeated vertex). It follows in this case that W has combinatorial length $\leq |V| - 1$ (otherwise W would be forced to repeat a vertex). How is this observation relevant to the correctness of the Bellman–Ford algorithm?

(b) If we permit zero-weight cycles (but continue to exclude negative cycles), then W is not necessarily a path anymore, but we can cut some zero-weight closed walks out of W such that the resulting walk W' has the same cost as W and W' is an $s \to \cdots \to v$ path.

5.3 Recall the **Bellman–Ford algorithm** for the single-source min-cost path problem. The input is (V, E, w, s) where (V, E) is a digraph, $w : E \to \mathbb{R}$ is a weight function, and $s \in V$ is a source vertex. Negative weights and even negative cycles are permitted. E is given as a linear list; the only way we can access the list is by scanning it in the given order. The algorithm maintains four types of variables: $c(v)$ (current cost of $v \in V$), $p(v)$ (parent of $v \in V$) the cycle counter i, and a

Boolean variable z to indicate whether or not anything has changed in a given round. Here is the algorithm. It takes a parameter $N \geq |V|$, the number of rounds.

N-round Bellman–Ford ($N \geq |V|$)

```
00   for v ∈ V  do c(v) = ∞, p(v) = NIL   [initialize]
01   end(for)
02    z = TRUE, c(s) = 0, p(s) = s        [initialize]
03   for i = 1 to N do
04       if z = FALSE then break          [exit for loop if no cost got
                                           updated in previous round]
05       end(if)
06       z = FALSE
07       for e ∈ E do RELAX(e)
08       end(for)
09   end(for)
10   if z = TRUE print "accessible negative cycle exists"
11       else print "optimal costs found"
12   end(if)
```

We use the basic form of the RELAX routine (no investigation of the status of the node to be updated).

RELAX(u, v) (for $e = (u, v) \in E$)

```
13   if c(v) > c(u) + w(u, v) then do
14       c(v) = c(u) + w(u, v), p(v) = u
15       z = TRUE                         [value updated]
16   end(if)
```

(a) DO: (Updated 2-10 5pm) Modify the procedure by removing lines 04 and 05. (This makes the variable z irrelevant and simplifies the solution of part (b).)

Prove that the following statement is a loop invariant for the main loop (lines 03 to 09) of the **modified** procedure (assuming some trivial loop invariants like in the analysis of Dijkstra's algorithm; state those):

$(\forall v \in V)(c(v) \leq$ the minimum of the costs of all **walks** of combinatorial length $\leq i$ from s to $v)$

Your proof will show that the highlighted statement is a loop invariant regardless of the presence of negative cycles and regardless of the value i (including values greater than $|V|$).

(b) **HW (4 points)** Line 04 tells us that if $z =$ FALSE at the beginning of an execution of the main **for** loop then we can immediately conclude that all current costs are optimal. Justify this statement. You may use part (a) without proof. Your proof should be very short, just a few lines. Solutions exceeding 2/3 of a page will not be graded. (Don't manipulate this requirement by funny typesetting; keep ≥ 1 inch margins and font size ≥ 11.)

(c) DO: Infer from part (a) that if there is no negative cycle then iteration $i = |V| - 1$ produces optimum costs for all vertices.

(d) **HW (4 points)** Prove: an accessible negative cycle exists $\Longleftrightarrow$ iteration $i = |V|$ of the main **for** loop sets the variable z to $z =$ TRUE. (This justifies line 10.)

Your solution should be very simple, just a few lines. Solutions exceeding 2/3 of a page will not be graded. (See comment in item (b) on typesetting.)

Comment. Note that (c) and (d) combined say that a choice of $N = |V|$ is sufficient to either compute the optimum path costs or to conclude that an accessible negative cycle exists. Bellman–Ford set $N = |V|$ so the running time of the algorithm is $O(|V||E|)$.

(e) DO: Let P_i be the "parent link digraph" after the i-th iteration of the main loop, i. e., the digraph whose vertices are the vertices of finite current cost and the edges are the pairs $(v, p(v))$ for these vertices $v \neq s$ after the i-th iteration. Prove:

(e1) If there is no accessible negative cycle then for all i, the digraph P_i is a tree oriented toward the root.

(e2) If there is an accessible negative cycle then after iteration $\#|V|$, the parent digraph contains a directed cycle.

(f) DO: If a negative cycle exists, find one in time $O(|V||E|)$. (Use part (e2).)

5.4 **HW (6 points)** The midterm (posted, click here) states the "Uphill-downhill Dijkstra problem" and asks to solve it in "Dijkstra time." Recall that Dijkstra cannot be implemented in linear time. Show that if all elevations are different then you can do better: solve the "Uphill-downhill Dijkstra problem" in linear time under this assumption.

Describe your solution in unambiguous English; no pseudocode required. State exactly what subroutines you use; do not use any unnecessary subroutines.

Your solution should be very simple, just a few lines with reference to another problem in homework sets 1–5. Solutions exceeding half a page will not be graded. (See comment in item 5.3(b) on typesetting.)

5.5 DO **(Order statistics: pivot selection)** Recall that the deterministic algorithm for selection began with breaking the n items into groups of 5; selecting the median of each group of 5; and then taking the median x of these medians as our pivot. Prove: $0.3n \leq \text{rank}(x) \leq 0.7n$. (We say that $\text{rank}(x) = k$ if x is the k-th smallest item on the list.)

5.6 **HW (5 points) (Divide and conquer: order statistics)** The pivot selection as described in the preceding exercise leads to the recurrence $T(n) \leq T(n/5) + T(7n/10) + O(n)$ (ignoring rounding) where $T(n)$ denotes the total number of comparisons made to select the k-th smallest item among n items (real numbers). (This item is called the *k-th order statistic* and is said to have *rank k*.) Prove that if a monotone increasing function $T(n) \geq 0$ satisfies this recurrent inequality then $T(n) = O(n)$. Use the method of reverse inequalities; ignore rounding.

5.7 **XC (4+3 points) (Divide and conquer: order statistics)** (a) Would the procedure work with groups of size 3? (b) Would it work with groups of size 7? — Describe in each case the recurrence you get and evaluate each recurrence.