National Taiwan University
Department of Electrical Engineering
Algorithms, Spring 2024

Classes:
Prof. Shao-Hua Sun &
Prof. Hui-Ru Jiang

Homework #4

(on-line submission due: 6:00 PM, 3 June 2024; late submissions are not allowed)

TA: Yi-Chen Lin r12943096@ntu.edu.tw and Wei-Kai Liao r12921a23@ntu.edu.tw

Collaboration policy: You can discuss the problems with other students, but you must write the final answers by yourself. Please specify all of your collaborators (names and student id's) or resources (websites) for each problem. If you solve some problems by yourself, please also specify "no collaborators". Homework without collaborator specification will be penalized by 50%.

1. (15 pts) In this problem, we give pseudocode for three different algorithms. Each one takes a connected graph and a weight function as input and returns a set of edges T. For each algorithm, either prove that T is a minimum spanning tree or prove that T is not a minimum spanning tree. Also describe the most efficient implementation of each algorithm, whether or not it computes a minimum spanning tree.

(a)

Algorithm 1 MAYBE-MST-A(G, w)

```
sort the edges into nonincreasing order of edge weights $w$
$T = E$
for each edge $e$, taken in nonincreasing order by weight do
    if $T - \{e\}$ is a connected graph then
        $T = T - \{e\}$
    end if
end for
return $T$
```

(b)

Algorithm 2 MAYBE-MST-B(G, w)

```
$T = \emptyset$
for each edge $e$, taken in arbitrary order do
    if $T \cup \{e\}$ has no cycle then
        $T = T \cup \{e\}$
    end if
end for
return $T$
```

(c)

Algorithm 3 MAYBE-MST-C(G, w)

```
$T = \emptyset$
for each edge $e$, taken in arbitrary order do
    $T = T \cup \{e\}$
    if $T$ has a cycle $c$ then
        let $e'$ be a maximum-weight edge on $c$
        $T = T - \{e'\}$
    end if
end for
return $T$
```

2. (15 pts) Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 49 Indian rupees, 1 Indian rupee buys 2 Japanese yen, and 1 Japanese yen buys 0.0107 U.S. dollars. Then, by converting

currencies, a trader can start with 1 U.S. dollar and buy $49 \times 2 \times 0 : 0107 = 1 : 0486$ U.S. dollars, thus turning a profit of 4.86 percent.
Suppose that we are given n currencies $c_1, c_2, \ldots, c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i, j]$ units of currency c_j.

(a) Give an efficient algorithm to determine whether or not there exists a sequence of currencies $\langle c_{i1}, c_{i2}, ..., c_{ik} \rangle$ such that $R[i_1, i_2] \cdot R[i_2, i_3] \ldots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$. Analyze the running time of your algorithm.

(b) Give an efficient algorithm to print out such a sequence if one exists. Analyze the running time of your algorithm.

3. (15 pts) Suppose that we wish to maintain the transitive closure of a directed graph $G(V, E)$ as we insert edges into E. That is, after each edge has been inserted, we want to update the transitive closure of the edges inserted so far. Assume that the graph G has no edges initially and that we represent the transitive closure as a boolean matrix.

   (a) Show how to update the transitive closure $G^* = (V, E^*)$ of a graph $G = (V, E)$ in $O(V^2)$ time when a new edge is added to G.

   (b) Give an example of a graph G and an edge e such that $\Omega(V^2)$ time is required to update the transitive closure after the insertion of e into G, no matter what algorithm is used.

   (c) Describe an efficient algorithm for updating the transitive closure as edges are inserted into the graph. For any sequence of n insertions, your algorithm should run in total time $\sum_{i=1}^{n} t_i = O(V^3)$, where t_i is the time to update the transitive closure upon inserting the ith edge. Prove that your algorithm attains this time bound.

4. (15 pts) A path cover of a directed graph $G(V, E)$ is a set P of vertex-disjoint paths such that every vertex in V is included in exactly one path in P. Paths may start and end anywhere, and they may be of any length, including 0. A minimum path cover of G is a path cover containing the fewest possible paths.

   (a) Give an efficient algorithm to find a minimum path cover of a directed acyclic graph $G = (V, E)$. (Hint: Assuming that $V = \{1, 2, ..., n\}$, construct the graph $G' = (V', E')$, where
   $V' = \{x_0, x_1, ..., x_n\} \cup \{y_0, y_1, ..., y_n\}$
   $E' = \{(x_0, x_i) : i \in V\} \cup \{(y_i, y_0) : i \in V\} \cup \{(x_i, y_j) : (i, j) \in E\}$
   and run a maximum-flow algorithm.)

   (b) Does your algorithm work for directed graphs that contain cycles? Explain.

5. (15 pts) Let 2-CNF-SAT be the set of satisfiable boolean formulas in CNF with exactly 2 literals per clause. Show that 2-CNF-SAT $\in P$. Make your algorithm as efficient as possible. (Hint: Observe that $x \vee y$ is equivalent to $\neg x \rightarrow y$. Reduce 2-CNF-SAT to an efficiently solvable problem on a directed graph.)

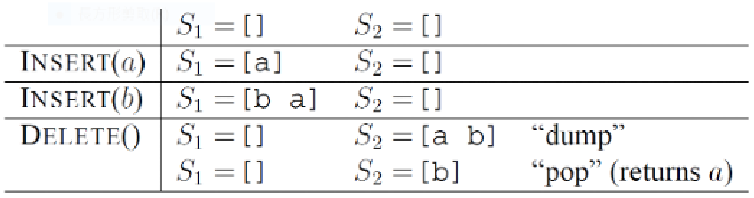
6. (20 pts) Mapmakers try to use as few colors as possible when coloring countries on a map, as long as no two countries that share a border have the same color. We can model this problem with an undirected graph $G = (V, E)$ in which each vertex represents a country and vertices whose respective countries share a border are adjacent. Then, a k-coloring is a function $c : V \rightarrow \{1, 2, ..., k\}$ such that $c(u) \neq c(v)$ for every edge (u, v). In other words, the numbers $1, 2, ..., k$ represent the k colors, and adjacent vertices must have different colors. The graph-coloring problem is to determine the minimum number of colors needed to color a given graph.

   (a) Give an efficient algorithm to determine a 2-coloring of a graph, if one exists.

   (b) Cast the graph-coloring problem as a decision problem. Show that your decision problem is solvable in polynomial time if and only if the graph-coloring problem is solvable in polynomial time.

   (c) Let the language 3-COLOR be the set of graphs that can be 3-colored. Show that if 3-COLOR is NP-complete, then your decision problem from part (b) is NP-complete.

7. (10 pts) Suppose we have code lying around that implemented a stack, and we now want to implement a queue. One way to do this is to use two stacks S_1 and S_2. To insert into our queue, we push into stack S_1. To remove from our queue, we first check if S_2 is empty, and if so we "dump" S_1 into S_2 (that is, we pop each element from S_1 and push it immediately onto S_2). Then we pop from S_2.
For instance, if we execute Insert(a), Insert(b), Delete(), the results are
Suppose each push and pop costs 1 unit of work, so that performing a dump when S_1 has n elements costs



	$S_1 = $ `[]`	$S_2 = $ `[]`	
INSERT(a)	$S_1 = $ `[a]`	$S_2 = $ `[]`	
INSERT(b)	$S_1 = $ `[b a]`	$S_2 = $ `[]`	
DELETE()	$S_1 = $ `[]`	$S_2 = $ `[a b]`	"dump"
	$S_1 = $ `[]`	$S_2 = $ `[b]`	"pop" (returns a)

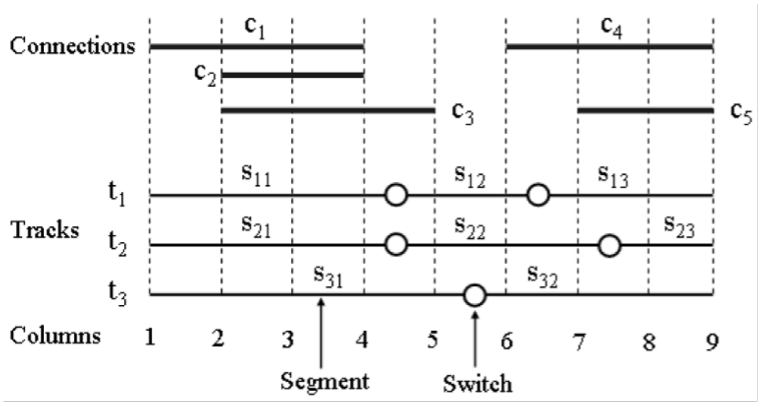
$2n$ units (since we do n pushes and n pops).
Consider the problem of making change for n dollars using the fewest number of coins. Assume that each coin's value is an integer.

(a) Suppose that (starting from an empty queue) we do 3 insertions, then 2 removals, then 3 more insertions, and then 2 more removals. What is the total cost of these 10 operations, and how many elements are in each stack at the end?

(b) If a total of n insertions and n removals are done in some order, how large might the running time of one of the operations be (give an exact, non-asymptotic answer)? Give a sequence of operations that induces this behavior, and indicate which operation has the running time you specified.

(c) Suppose we perform an arbitrary sequence of insertions and removals, starting from an empty queue. What is the amortized cost of each operation? Give as tight (i.e., non-asymptotic) of an upper bound as you can. Use the accounting method to prove your answer. That is, charge x for insertion and y for deletion. What are x and y? Prove your answer.

8. (10 pts) The figure below shows a segmented routing structure in a row-based field-programmable gate array (FPGA). There are five connections, $c_1, c_2, \ldots, c_5$, to be routed on three segmented tracks, t_1, t_2, and t_3, with eight segments $s_{11}, s_{12}, \ldots, s_{32}$ in the row-based FPGA. A track can be partitioned into a set of segments by using switches. If a switch incident on two adjacent segments is "ON", then the two segments are electrically connected; otherwise, the two segments can be used independently. You are asked to route (place) the five connections on the three segmented tracks. Suppose each connection can use at most one segment for routing, i.e., 1-segment routing. In other words, a connection c_k of the column span $[l_k, r_k]$ is said to be routed on a segment s_{ij} of track t_i if $c_k = [l_k, r_k]$ is placed within the column span of s_{ij}. For example, $c_3 = [2, 5]$ can be routed on segment s_{31} of track t_3 (which consumes only one segment) while it cannot route on track t_1 or t_2 (which would have consumed two segments, thus violating the constraint of 1-segment routing). Give an efficient algorithm to solve the 1-segment routing problem. What is the time complexity of your algorithm in terms of the number of connections and the number of segments?



Study and, in general, the pursuit of truth and beauty is a sphere of activity in which we are permitted to remain children all our lives.
– A. Einstein