*Introduction to Algorithms* November 21, 2003
Massachusetts Institute of Technology 6.046J/18.410J
Professors Shafi Goldwasser and Silvio Micali Handout 26

# Quiz 2 Solutions

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- When the quiz begins, write your name on every page of this quiz booklet.
- The quiz contains four multi-part problems. You have 120 minutes to earn 108 points.
- This quiz booklet contains **16** pages, including this one. Extra sheets of scratch paper are attached. Please detach them before turning in your quiz.
- This quiz is closed book. You may use one handwritten A4 or $8\frac{1}{2}'' \times 11''$ crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Do not put part of the answer to one problem on the back of the sheet for another problem, since the pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

| Problem | Points | Grade | Initials |
|---|---|---|---|
| 1 | 40 | | |
| 2 | 25 | | |
| 3 | 25 | | |
| 4 | 18 | | |
| Total | 108 | | |

Name: Solutions

Circle your recitation letter and the name of your recitation instructor:

David A B Steve C D Hanson E F

**Problem -1. True or False, and Justify** [40 points]

Circle **T** or **F** for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation. Parts **(a)** through **(d)** are worth **2 points** each and parts **(e)** through **(l)** are worth **4 points** each.

**(a) T F** A preorder traversal of a binary search tree will output the values in sorted order.

**Solution:** False. Consider a BST with 2 at the root and 1 as left child and 3 as right child. The preorder is 2, 1, 3. An *inorder* traversal of a BST outputs the values in sorted order.

**(b) T F** The cost of searching in an AVL tree is $\Theta(\lg n)$.

**Solution:** True. An AVL tree is a balanced binary search tree. Therefore, searching in an AVL tree costs $\Theta(\lg n)$.

**(c) T F** In a topological ordering of the vertices of a DAG, if a vertex $v$ comes after a vertex $u$ in the ordering, then there is a directed path from $u$ to $v$.

**Solution:** False. A vertex $u$ may come before $v$ in the topologically sort, but that does not mean there is actually a path from $u$ to $v$. In particular, $v$ may have no in-edges.

**(d) T F** The transitive closure of a directed graph $G = (V, E)$ can be computed in $O(V^3)$ time.

**Solution:** True. We can use FLOYD-WARSHALL to compute the all pairs shortest paths. If there is no directed path from $i$ to $j$ in $G$, then the shortest path from $i$ to $j$ is $\infty$.

**(e) T F** Any mixed sequence of $m$ increments and $n$ decrements to a $k$-bit binary counter (that is initialized to zero and is always non-negative) takes $O(m+n)$ time in the worst case.

**Solution:** False. If we allow decrement operations, then a series of $n+m$ increment and decrement operations could take $O(kn+km)$ in the worst case.

**(f) T F** Suppose we have a directed graph $G=(V,E)$ that is strongly connected. For any depth-first search of $G$, if all the forward edges of $G$ (with respect to the depth-first forest) are removed from $G$, the resulting graph is still strongly connected.

**Solution:** True. In a DFS on $G$, *forward edges* are those nontree edges $(u,v)$ connecting a vertex $u$ to a descendant $v$ in the depth-first tree. Since $v$ is a descendant of $u$ on the DFS tree, there are tree edges that create a directed path from $u$ to $v$. Therefore, after removing edge $(u,v)$, there is still a directed path from $u$ to $v$, so $G$ is still strongly connected.

**(g) T F** Consider a directed acyclic graph $G = (V, E)$ and a specified source vertex $s \in V$. Every edge in $E$ has either a positive or a negative edge weight, but $G$ has no negative cycles. Dijkstra's Algorithm can be used to find the shortest paths from $s$ to all other vertices.

**Solution:** False. Counterexample $G = \{w(s, b) = -3, w(s, c) = 2, w(b, c) = 4\}$.

**(h) T F** Suppose someone has run FLOYD-WARSHALL on a weighted graph $G = (V, E)$ and gives you a 3-dimensional array $D$ in which $D[i, j, k] = d_{ij}^{(k)}$ (for $0 \leq i, j, k \leq n$) is the length of the shortest path from vertex $i$ to vertex $j$ for which all intermediate vertices are in the set $\{1, 2, \ldots, k\}$. The array $D$ can be used to determine if $G$ contains a negative cycle in $O(V)$ time.

**Solution:** True. For each pair of vertices $i, j \in V$, FLOYD-WARSHALL computes $d_{ij}^{(0)}, d_{ij}^{(1)}, \ldots d_{ij}^{(n)}$. Thus, if there is a negative cycle, then $d_{ii}^{(n)}$ will be negative for some $i$.

**(i) T F** Given a bipartite graph $G$ and a matching $M$, we can determine if $M$ is maximum in $O(V + E)$ time.

**Solution:** True. Modified BFS is used to determine if there is an augmenting path in time $O(V + E)$. If there is no augmenting path, then the matching is maximum.

**(j) T F** Given a random skip list on $n$ elements in which each element has height $i$ with probability $(1/3)^{i-1}(2/3)$, the expected number of elements with height $\lg_3 n$ is $O(1)$.

**Solution:** True. Let $X_i$ be the indicator random variable that is a 1 if element $i$ is in the set of elements with height at least $\lg_3 n$ and 0 otherwise. An element has height at least $\lg_3 n$ with probability $(2/3)(1/3)^{\lg_3 n - 1} = 2/n$. Thus, $E[X_i] = 2/n$ and $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = 2 = O(1)$.

**(k) T F** Suppose you are given an undirected connected graph with integer edge weights in which each vertex is degree 2. An MST can be computed for this graph in $O(V)$ time.

**Solution:** True. The graph must be a cycle with $O(V)$ edges. We can remove the edge with the highest weight in $O(V)$ time.

**(l) T F** Consider an undirected, weighted graph $G$ in which every edge has a weight between 0 and 1. Suppose you replace the weight of every edge with $1 - w(u, v)$ and compute the minimum spanning tree of the resulting graph using Kruskal's algorithm. The resulting tree is a maximum cost spanning tree for the original graph.

**Solution:** True. Let $G'$ be the graph in which every edge has weight $1 - w(u, v)$. Let MaxST denote the maximum spanning tree in $G$ and let MST denote the minimum spanning tree if $G'$. Suppose MST does correspond to MaxST in $G$. Then there is a spanning tree in $G$ with higher weight, impying that if we replace each edge with weight $1 - w(u, v)$, we will find a spanning tree in $G'$ that has less weight than MST, which is a contradiction.

**Problem -2. Short Answer Problems** [25 points]

Give *brief*, but complete, answers to the following questions. Each problem part is worth **5 points**.

**Amortized Analysis** `(re-written from Fall 98)`

You are to maintain a collection of lists and support the following operations.

(i) *insert*(item, list): insert item into list (cost = 1).

(ii) *sum*(list): sum the items in list, and replace the list with a list containing one item that is the sum (cost = length of list).

**(a)** Use the Accounting Method to show that the amortized cost of an *insert* operation is $O(1)$ and the amortized cost of a *sum* operation is $O(1)$.

**Solution:** We will maintain the invariant that every item has one credit. Insert gets 2 credits, which covers one for the actual cost and one to satisfy the invariant. Sum gets one credit, because the actual cost of summing is covered by the credits in the list, but then the result of the sum will need one credit to maintain the invariant.

A common error was not putting a credit on the newly created sum.

**(b)** Use the Potential Method to show that the amortized cost of an *insert* operation is $O(1)$ and the amortized cost of a *sum* operation is $O(1)$.

**Solution:** We define $\Phi(list)$ to be the number of elements in the list. Then the amortized cost of an insert operation is $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$. The actual cost $c_i$ is 1. The change in potential is 1. So the amortized cost is 2. For a sum operation, the actual cost, $c_i$ is $k$ and the change in potential is $\Phi_i - \Phi_{i-1} = 1 - (k) = 1 - k$, so the amortized cost of a sum is 1.

**Balanced Search Trees**

**(c)** Show that the number of node splits performed when inserting a single node into a 2-3 tree can be as large as $\Omega(\lg n)$, where $n$ is the number of keys in the tree. (E.g. give a general, worst-case example.)

**Solution:** Suppose that every node in an n-node 2-3 tree is full – i.e., all internal nodes have three children. Then the height of the tree is $\log_3 n = \Omega(\log n)$. [Common error: I don't care that its height is $O(\log n)$!] When we insert into a leaf node, it's full—so we have to have to do something with the fourth element ... that is, bump the median element up to its parent. But its parent is full, too—so we have to bump up an element from there. And so on, all the way up to the root. The numbers of splits is $\Omega$(height) = $\Omega(\log n)$.

**Faster MST Algorithms**

**(d)** Given an undirected and connected graph in which all edges have the same weight, give an algorithm to compute an MST in $O(E)$ time.

**Solution:** Run DFS and keep only the tree edges.

Some common errors were: (1) not running in linear time (union-find is NOT constant!) and (2) not producing a tree (a graph where every node has degree $\geq 1$ is not necessarily connected!).

**Competitive Analysis** `(written by DLN)`

**(e)** Your TA can be bribed to give you an A+ in 6.046 if you give him at least $x$ dollars, where $x$ is a positive integer. *However, you do not know what $x$ is.* Since your TA is forgetful, you must give him $x$ dollars at once; if you give him less, he just keeps it and forgets who gave it to him. Your goal is to get an A+ and minimize the amount paid to the TA. You have the following strategy: you pay your TA 1 dollar and keep doubling the amount you pay until you get an A+. The following is pseudocode for this strategy.

BRIBE-TA
$paid \leftarrow 0$
$amount \leftarrow 1$
While TA has not given you an A+
    Pay TA $amount$ dollars
    $paid \leftarrow paid + amount$
    $amount \leftarrow 2 * amount$
Return $paid$

Analyze the competitive ratio of the Bribe-TA algorithm.

**Solution:** We will show that the Bribe-TA algorithm is 4-competitive, because $paid$ is at most $4x$ dollars and $x$ dollars is the minimum amount we can pay.
Suppose $2^k \leq x \leq 2^{k+1}$. Then the total amount paid to the TA is:

$$\sum_{i=0}^{k+1} 2^i = 2^{k+2} - 1 \leq 4x.$$

**Problem -3. Dynamic Programming** [25 points]

Santa Claus is packing his sleigh with gifts. His sleigh can hold no more than $c$ pounds. He has $n$ different gifts, and he wants to choose a subset of them to pack in his sleigh. Gift $i$ has utility $u_i$ (the amount of happiness gift $i$ induces in some child) and weight $w_i$. We define the weight and utility of a *set of gifts* as follows:

- The weight of a set of gifts is the ***sum*** of their weights.
- The utility of a set of gifts is the ***product*** of their utilities.

For example, if Santa chooses two gifts such that $w_1 = 3, u_1 = 4$ and $w_2 = 2, u_2 = 2$, then the total weight of this set of gifts is 5 pounds and the total utility of this set of gifts is 8. All numbers mentioned are positive integers and for each gift $i$, $w_i \leq c$. Your job is to devise an algorithm that lets Santa maximize the utility of the set of gifts he packs in his sleigh without exceeding its capacity $c$.

**(a)** **[5 points]** A greedy algorithm for this problem takes the gifts in order of increasing weight until the sleigh can hold no more gifts. Give a small example to demonstrate that the greedy algorithm does not generate an optimal choice of gifts.

**Solution:** Suppose that $c = 2$ and the gifts have weights $1$ and $2$ and $u_i = w_i$ for each gift. If Santa chooses gift 1, then he can not fit gift 2, so the total utility he obtains is 1 rather than 2.

**(b) [10 points]** Give a recurrence that can be used in a dynamic program to compute the maximum utility of a set of gifts that Santa can pack in his sleigh. Remember to evaluate the base cases for your recurrence.

**Solution:** Let $H(k, x)$ be the maximal achievable utility if the gifts are drawn from 1 through $k$ (where $k \leq n$) and weigh at most $x$ pounds.

For $1 \leq k \leq n$:
If $x - w_k \geq 0$, $H(k, x) = \max\{H(k-1, x - w_k) \cdot u_k, H(k-1, x), u_k\}$
Otherwise (if $x - w_k < 0$), $H(k, x) = H(k-1, x)$

**Base cases:** $H(0, x) = 0$ for all integers $x, 1 \leq x \leq c$.

Some common errors were (1) forgetting to include $u_i$ in the recurrence for the case in which $H(k-1, x - w_k)$ is 0, (2) neglecting one dimension, e.g. weight, (3) using $+$ instead of $*$ for utility, (4) writing the recurrence as $H(i, c)$ instead of (general) $H(i, x)$.

**(c) [7 points]** Write pseudo-code for a dynamic program that computes the maximum utility of a set of gifts that Santa can pack in his sleigh. What is the running time of your program?

**Solution:** We give the following pseudo-code for a dynamic program that takes as input a set of gifts, $\mathcal{G} = \{g_1, g_2, \ldots g_n\}$, a maximum weight, $c$, and outputs the set of gifts with the maximum utility.

MAXIMUM-UTILITY$(\mathcal{G}, c)$
For $1 \leq x \leq c$
  $H(0, x) = 0$
For $1 \leq k \leq n$
  For $1 \leq x \leq c$
    If $x - w_k < 0$
      $H(k, x) = H(k-1, x)$
    If $x - w_k \geq 0$
      $H(k, x) = \max\{H(k-1, x - w_k) \cdot u_k, H(k-1, x), u_k\}$
Return $H(n, c)$

The running time of this algorithm is $O(c \cdot n)$.

**(d) [3 points]** Modify your pseudo-code in part **(c)** to output the actual set of gifts with maximum utility that Santa packs in his sleigh.

**Solution:** The following pseudo-code takes as input the table $H$ computed in part **(c)** and variables $k, n$ and outputs a set of gifts that yield utility $H(k, n)$.

OUTPUT-GIFTS$(H, k, x)$
If $k = 0$
  Output $\emptyset$
Else If $H(k, x) = H(k-1, x - w_k) \cdot u_k$
  Output $g_k$
  OUTPUT-GIFTS$(H, k-1, x - w_k)$
Else if $H(k, x) = H(k-1, x)$
  Output OUTPUT-GIFTS$(H, k-1, x)$
Else if $H(k, x) = u_k$
  Output $g_k$

**Problem -4. Detecting Costly Cycles** [18 points]

You are given a weighted, directed graph $G = (V, E)$, in which each edge has a weight between 0 and 1, i.e. for all $(i, j) \in E,\ 0 \le w(i, j) \le 1$. We say a directed cycle with $c$ edges is *costly* if the sum of the weights of the edges on the cycle is more than $c - 1$.

**(a) [10 points]** Give an $O(V^3)$ algorithm to find the minimum-cost directed cycle in $G$. (Assume the graph $G$ contains no self-loops, so a directed cycle contains at least two edges.)

**Solution:** Run FLOYD-WARSHALL to find all pairs shortest paths. Then for every $i, j \in V$, find the shortest path from $i$ to $j$ (i.e. $d_{ij}^{(n)}$) and the shortest path from $j$ to $i$ (i.e. $d_{ji}^{(n)}$). These two paths make up the minimum cost cycle that goes through vertices $i$ and $j$. Thus, if we find the minimum-cost cycle that goes through $i, j$ for all $i, j \in V$, then the minimum of these is the minimum-cost cycle in the graph. Below is pseudo-code for finding the cost of the minimum-cost cycle in a directed graph $G$.

MIN-COST-CYCLE($G$)
Run FLOYD-WARSHALL on $G$
$c \leftarrow \infty$
For all pairs $i \neq j \in V$
    If $d_{ij}^{(n)} + d_{ji}^{(n)} < c$
        $c = d_{ij}^{(n)} + d_{ji}^{(n)}$
Return $c$

Common errors included running FLOYD-WARSHALL and then finding the minimum $d_{ii}^{(n)}$ for all $i \in V$. This does not work if you use FLOYD-WARSHALL as implemented in CLRS because $d_{ii}^{(1)}$ is initialized to 0 and since $G$ contains no negative cycles, $d_{ii}^{(n)}$ will be 0 for all $i \in V$. This approach could work if you run FLOYD-WARSHALL and initialize $d_{ii}^{(1)} = \infty$ for all $i \in V$. However, for full credit for this answer, you needed to argue why this implementation of FLOYD-WARSHALL is correct.

Another common incorrect solution was to run DFS and "find all cycles", sort them according to cost and return the minimum. Note that there may be exponentially many cycles, so any algorithm that finds all cycles is not efficient.

**(b) [8 points]** Give an $O(V^3)$ algorithm to determine whether $G$ contains a *costly* cycle. Argue that your algorithm is correct and analyze its running time. (**Hint:** Use part **(a)** on a modified graph.)

**Solution:** Consider the graph $G'$ in which every edge has weight $w'(i,j) = 1 - w(i,j)$. If any cycle in $G$ with $c$ edges has value greater than $c-1$, then the same cycle in $G'$ has weight less than 1. Therefore, we simply need to find the minimum-cost cycle in $G'$. If the minimum-cost cycle is less than 1, then $G$ contains a costly cycle. Finding a minimum-cost cycle in $G'$ can be done in $O(V^3)$ time using part **(a)**
Correctness follows from the following claim. A cycle in $G$ is costly iff the corresponding cycle in $G'$ has cost less than 1. In other words, consider some cycle $C$ which contains a subset of $c$ edges in $G$. Then we have:

$$\sum_{ij \in C} w_{ij} > c - 1 \iff \sum_{ij \in C} (1 - w_{ij}) = c - \sum_{ij \in C} w_{ij} < c - (c-1) = 1.$$

Most people used the correct modification of the graph, since that was a hint given in problem **1(l)**. However, many people incorrectly argued correctness by claiming that a minimum-cost cycle in $G'$ corresponds to a *maximum*-cost cycle in $G$. This is not true: a minimum-cost cycle in $G'$ corresponds to a cycle in $G$ which has the smallest difference between its cost and its cardinality, but it might not be the maximum-cost cycle.