Introduction to Algorithms | April 14, 2004
Massachusetts Institute of Technology | 6.046J/18.410J
Professors Erik Demaine and Shafi Goldwasser | Practice Quiz 2

Practice Quiz 2

- 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. Each part is worth **4 points**.

(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 $O(\lg n)$.

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

(c) T F The expected amount of space used by a skip list on n elements is $O(n)$.

Solution: True. Each element appears in $2 = O(1)$ rows in expectation, and each occurrence requires $O(1)$ space for the element and pointers. By linearity of expectation, the space taken by all elements is $O(n)$ in expectation. Alternatively, we expect each level to have half as many elements as the one below it, for space $O(n + n/2 + n/4 + \cdots) = O(2n) = O(n)$ by geometric summation.
It is not correct to answer "false" because the skip list has $O(\log n)$ levels (w/ high prob.), and each level has $O(n)$ elements, for a total of $O(n \log n)$ space. While this bound is true, it is not the tightest possible (i.e., it is too loose of an approximation of the actual space used).

(d) T F The height of a randomly built binary search tree is $O(\log n)$ with high probability; therefore in randomized quicksort, every element is involved in $O(\log n)$ comparisons with high probability.

Solution: False. In randomized quicksort, the first pivot is compared to *every* other element, and there are $n - 1 = \Omega(n)$ of them. Therefore every element has at least a $1/n$ chance of being involved in $\Omega(n)$ comparisons.
Alternately, the root of the randomly-built binary search tree is compared to every other element. (More generally, an element is compared with every element in its subtree, as well as all its ancestors.) These comparisons are the same ones done by randomized quicksort.
Many solutions regurgitated the fact that there is a connection between the comparisons performed by quicksort and those performed in building a random BST. This is true, but irrelevant. The *height* of a node in the BST has nothing to do with the number of elements it is compared to in the corresponding quicksort.

(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.

Spring 2004: See CLRS pg. 408-409

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 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 $G = \{w(s,b) = -3, w(s,c) = 2, w(b,c) = 4\}$.

(g) 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.

(h) 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 $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = 2 = O(1)$.

(i) 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.

(j) 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

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 Ω(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 ≥ 1 is not necessarily connected!).

FFT

(e) Snowball Throwing

Several 6.046 students hold a team snowball throwing contest. Each student throws a snowball with a distance in the range from 0 to $10n$. Let M be the set of distances thrown by males and F be the set of distances thrown by females. You may assume that the distance thrown by each student is unique and is an integer. Define a team score to be the combination of one male and one female throw.

Give an $O(n \log n)$ algorithm to determine every possible team score, as well as how many teams could achieve a particular score. This multi-set of values is called a *cartesian sum* and is defined as:

$$C = \{m + f : m \in M \text{ and } f \in F\}$$

Solution:
Represent M and F as polynomials of degree $10n$ as follows:

$$M(x) = x^{a_1} + x^{a_2} + \ldots + x^{a_n}, F(x) = x^{b_1} + x^{b_2} + \ldots + x^{b_n}$$

Multiply M and F in time $\Theta(n \log n)$ to obtain a coefficient representation $c_0, c_1, \ldots, c_{2n}$. In other words $C(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{2n} x^{2n}$. Each pair a_i, b_j will account for one term $x^{a_i} x^{b_j} = x^{a_i + b_j} = x^k$. Therefore, c_k will be the number of such pairs $a_i + b_j = k$.

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($\mathcal{G}, c$)
1  For $1 \leq x \leq c$
2      $H(0, x) = 0$
3  For $1 \leq k \leq n$
4      For $1 \leq x \leq c$
5              If $x - w_k < 0$
6                  $H(k, x) = H(k - 1, x)$
7              If $x - w_k \geq 0$
8                  $H(k, x) = \max\{H(k - 1, x - w_k) \cdot u_k, H(k - 1, x), u_k\}$
9  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($H, k, x$)
1  If $k = 0$
2      Output $\emptyset$
3  Else If $H(k, x) = H(k - 1, x - w_k) \cdot u_k$
4      Output $g_k$
5      OUTPUT-GIFTS($H, k - 1, x - w_k$)
6  Else if $H(k, x) = H(k - 1, x)$
7      Output OUTPUT-GIFTS($H, k - 1, x$)
8  Else if $H(k, x) = u_k$
9      Output $g_k$
```

Problem 4. Reliable distribution

A communication network consists of a set V of nodes and a set $E \subset V \times V$ of directed edges (communication links). Each edge $e \in E$ has a ***weight*** $w(e) \geq 0$ representing the cost of using e. A ***distribution*** from a given source $s \in V$ is a set of directed $|V| - 1$ paths from s to each of the other $|V| - 1$ vertices in $V - \{s\}$. The ***cost*** of a distribution is the sum of the weights of its constituent paths. (Thus, some edges may be counted more than once in the cost of the distribution.)

(a) Give an efficient algorithm to determine the cheapest distribution from a given source $s \in V$. You may assume all nodes in V are reachable from s.

Solution: This problem can be modelled as a single-source shortest paths problem. A distribution is a tree of paths from s to every other vertex in the graph. Since the cost of a distribution is the sum of the lengths of its paths, a minimum cost distribution is a set of shortest paths. Since the edge weights are non-negative, we can use Dijkstra's algorithm. The running time of Dijkstra's algorithm can be improved from $O(|V|^2)$ to $O((|V| + |E|) \lg |V|)$ using a binary heap. Since we assumed that the graph is connected, the running time is $O(|E| \lg |V|)$. If we use a Fibonacci heap, the running time is $O(|V| \lg |V| + |E|)$.
In addition, we need to return the minimum cost distribution. Dijkstra's algorithm as given in CLR computes backpointers $\pi[v]$ to represent the shortest paths. We can use these to represent the distribution; when asked for the shortest path from s to v, we trace the backpointers from v to s and return the traversed edges (in reverse order).

(b) One of the edges in the communication network may fail, but we don't know which one. Give an efficient algorithm to determine the maximum amount by which the cost of the cheapest distribution from s might increase if an adversary removes an edge from E. (The cost is infinite if the adversary can make a vertex unreachable from s.)

Solution: If an edge is removed from the graph, it is possible that the cost of the minimum cost distribution on the resulting graph may be more than the cost of original chepeast distribution. That is, let D be the minimum cost distribution found in part (a), and let $C(D)$ be its cost. If we remove edge e from the graph, let $C(e)$ denote the cost of the minimum cost distribution for the new graph (with edge set $E - \{e\}$). We need to compute

$$\max_{e \in E} C(e) - C(D).$$

First recall that cost of a distribution is $\sum_{v \in V} d[v]$, where the $d[v]$ are the distance values returned by Dijkstra.

The straightforward brute-force approach to solve this problem is compute $C(e)$ by deleting e from the graph and rerunning the algorithm from (a). However, note that if the deleted edge $e \notin D$ then $C(e) = C(D)$, since the removal of e does not affect the distribution D. So we only need to find

$$\max_{e \in D} C(e) - C(D).$$

Since the edges in D are a set of shortest paths, they form a tree, and a tree has $|V| - 1$ edges.

To compute $C(e)$ for an edge $e \in D$, we can delete e and then rerun Dijkstra's algorithm on the resulting graph. It is important to note that removing e may make some vertices unreachable from s. To check this, we remove e, rerun Dijkstra, and then check if any of the $d'[v]$ distances are ∞. If so, then $C(e) = \infty$ and we should halt the algorithm and return ∞ as the maximum possible increase. If not, then $C(e) = \sum_v d'[v]$.

The running time of this solution is the cost of $|V| - 1$ calls to Dijkstra's algorithm. Using Fibonacci heaps, this is $O(|V|^2 \lg |V| + |V||E|)$.