INSTRUCTIONS:

- This a closed book exam, no calculators, but you may refer to TWO prepared 8”x11” sheets of notes.
- The questions are divided into two parts. The first part are mechanical and meant to be “quickies” and answers must be written on the question sheet. Do them quickly so that you have more time for the second part.
- All algorithms must be described at two levels: first, give a descriptive explanation in which you explain your data representation or variable names and general strategy. Second, provide a pseudo-code that is C-like or Java-like.
- Please write ONLY on the right-hand side of 2-sided pages. 4 points will be deducted for non-compliance.

PART A (50 Points)

**Q1** (2 Points each)
(a) The theory of $NP$-completeness is centered around two sets, $P$ and $NP$. Formally, what are the elements of these two sets?
(b) In what sense is an element in $P \cup NP$ regarded as a “problem”?
(c) (EXTRA CREDIT) How can the “problems” in the sense of (b) be related to the ordinary problems we encounter in this course, especially in regards to computability in polynomial time.

**SOLUTION**
(a) Each element in $L \in P \cup NP$ is a language, i.e., a set of strings over some alphabet.
(b) We view $L$ as a decision problem, where for each input, we just have to decide one of two possible values. Turing machines are well-suited for such problems!
(c) We can usually transform our problems into a corresponding decision problem such that original problem is tractable iff the decision problem is tractable. Note: tractable can be taken to mean solvable by a Turing machine in polynomial time.

**Comments:**

**Q2** (5 Points each) No justifications needed.
State the $\Theta$-order bound of these sums.
(a) $S_a(n) = \sum_{i=1}^{n} i(i+1)(i+2)$
(b) $S_b(n) = \sum_{i=1}^{n} \log(i) \log(i^2)$

**SOLUTION**
(a) This is polynomial-type and each term is $\Theta(i^3)$. So $S_a(n) = \Theta(n^4)$.
(b) This is polynomial-type $S_b(n) = \Theta(n \log^2 n)$.

**Comments:**

**Q3** (6 points)
Suppose $T(n) = 3T(n/4) + 2T(n/3) + n$. Give good upper and lower bounds on $T$ based on the Master Theorem. You must briefly indicate how you derive your bounds (one or two lines suffice).

**SOLUTION**
$T(n) = \Omega(n^\alpha)$ where $\alpha = \log_3 5$ since $T(n) \geq 5T(n/3) + n$. $T(n) = O(n^\beta)$ where $\beta = \log_4 5$ since $T(n) \leq 5T(n/4) + n$.

**Comments:**

© Chee Yap

Page 1

December 14, 2011
(Q4) (4 Points)
We gave an amortized analysis of incrementing binary counters. Joe thought: why don’t we allow decrements on our counters? Convince Joe that it is not possible to get $O(1)$ amortized cost if we allow decrements as well.

**SOLUTION** If you do a sequence of $n$ alternating increments and decrements so that the counter value alternates between $2^n$ and $2^n - 1$, you will have a total cost of $\Theta(n \log n)$, i.e., the amortized cost per operation is $\Theta(\log n)$, not $O(1)$.

**Comments:**

(Q5) (4 Point)
I incremented my counter from 31 to 129. What is the exact cost?

**SOLUTION** From homework, we know that to increment the counter value from $m$ to $n$, the total cost is $2(n - m) + \Phi(n) - \Phi(m)$. Note that $32 = (01111)_2$ and $129 = (10000011)_2$.

Thus, the answer is $2(129 - 31) + \Phi(129) - \Phi(31) = 2(88) + 2 - 5 = 173$.

**Comments:**

(Q6) (4 Points each)
Let $X_n = (01a)_n$ and $Y_n = (10a)_n$.
(a) Show the matrix $M$ for computing $L(X_3, Y_3)$.
(b) We want to recover the set $LCS(X_3, Y_3)$ (you may annotate the matrix from (a) first). Since this set grows very quickly with $n$, we want you to represent the set in (b) as a DAG (directed acyclic graph). Draw such a graph. [TRY TO SHOW ANY STRUCTURE THAT MIGHT BE PRESENT IN SUCH A GRAPH]
(c) How many strings are there in $LCS(X_3, Y_3)$?
(d) Describe any special properties of the DAG representation of $LCS(X_n, Y_n)$ for arbitrary $n$. In particular, how many nodes does it have?

**SOLUTION** You might recall from homework that $L(X_n, Y_n) = 2n$.
(a) The last entry should be 6. **MATRIX...**
(b) The graph should represent 20 strings in $LCS(X_3, Y_3)$. **FIGURE...**
(c) 18.
(d) Our DAG has a unique source with $n(n+1)^2$ nodes and $n^2$ edges. The nodes are partitioned into $2n + 1$ levels, and edges only go from level $i$ to level $i - 1$. There is a unique source at level $n$ and all maximal paths from source to sink has length $2n$. The number of maximal paths is equal to $\lambda(X_n, Y_n)$.

**Comments:**

(Q7) (5 Points each)
We use the strings $X_n, Y_n$ from the previous question.
(a) Compute the alignment cost of $X_3, Y_2$ (not $Y_3$) under the following cost function:

$$\Delta(x, y) = \begin{cases} 
3 & \text{if } x = * \text{ or } y = * \\
-2 & \text{else if } x = y \\
1 & \text{else if } x \neq y
\end{cases}$$

Show the matrix for your work.
(b) Produce one optimal alignment for $(X_3, Y_2)$.
(c) Describe the properties of the DAG representation of the set of all optimal alignments of any two strings $X, Y$ (this is analogous to part (d) in the previous question).

**SOLUTION** (a) $A(X_3, Y_2) = 3$.
(b) One optimal alignment is $X_* = 01a, 01a, 01a, Y_* = *1*, 0a1, 0* a$.
(c) This is analogous to part(d) in the previous problem, but the DAG we construct is no longer layered. The optimal alignment path are no longer all of the same length, and may vary from $\min\{m, n\}$ to $m + n$.

**Comments:**

© Chee Yap  December 14, 2011
(Q8) (5 Point)

Show a counter example to the triangular inequality $A_\Delta(X, Z) \leq A_\Delta(X, Y) + A_\Delta(Y, Z)$ when $\Delta$ is an arbitrary alignment cost.

(b) (EXTRA CREDIT, SKIP AT FIRST) Sketch an argument that the triangular inequality holds when the alignment cost is non-negative.

SOLUTION  
(a) Counter example: Let the gap penalty be 3, and for $x, y \in \Sigma$, $\Delta(x, y) = 1$ if $x \neq y$ and $\Delta(x, y) = -2$ if $x = y$. Let $X = ab, Y = bb$ and $Z = ba$. Then we have $A(ab, ba) = 2, A(ab, bb) = -2$, and $A(bb, ba) = -2$.

Comments:

PART B (50 Points)

(P1) (25 Points) Huffman Coding

Let $f : \Sigma \to \mathbb{N}$ be a frequency function, and let $T$ be a 4-ary (Huffman) code tree for $f$ (similar to the 3-ary one in homework). Thus, each leaf of $T$ is associated with a symbol $b \in \Sigma$, and also a weight $f(b)$. The internal nodes of $T$ has a weight equal to the sum of the weights of its children. The COST of $T$ is the sum of all the weights of the internal nodes.

(a) Give a short inductive proof of the following fact: Suppose $T$ is any 4-ary tree on $n \geq 1$ leaves, and let $N_d$ be the number of nodes with $d$ children ($d = 0, 1, 2, 3, 4$). Thus, $n = N_0$. Give a short inductive proof for the following formula: $n = 1 + N_2 + 2N_3 + 3N_4$.

(b) Show that if $T$ is an optimal code tree, then $N_1 = 0$ and $3N_2 + 2N_3 \leq 4$, and every non-full internal node has only leaves as children. Moreover, we can always transform $T$ into $T'$ such that $N'_1 = 0$ and $N'_2 + N'_3 \leq 1$.

(c) Suppose $r = (n - 1) \mod 3$. So $r \in \{0, 1, 2\}$. Show how $N'_2, N'_3$ in part (b) is determined by $r$.

(d) Describe an algorithm to construct an optimal code tree from a frequency function $f$.

(e) Show the optimal 4-ary Huffman tree for the input string hello world!. Please state the cost of this optimal tree.
SOLUTION (a) We use induction on \( n \). For \( n = 1 \), the formula holds (we don’t have to prove this). For \( n > 1 \), there is at least one internal node. Look at the internal node \( u \) of greatest depth. Its children are all leaves. Let \( u \) have \( d \geq 1 \) children. If we delete the children of \( u \), we obtain a tree \( T' \) with \( n' \) leaves. Clearly, \( N'_d = N_d - 1 \) and \( N'_0 = N_0 - d + 1 = n' \). Moreover, \( N'_c = N_c \) for all \( c \) not equal to \( d \) or \( 0 \). By inductive hypothesis, \( n' = n - d + 1 = 3N'_4 + 2N'_3 + N'_2 + 1 \). For instance, suppose \( d = 3 \). Then \( n' = n - 2 + 3N_4 + 2(N_3 - 1) + N_2 + 1 \). This implies \( n = 3N_4 + 2N_3 + N_2 + 1 \), as we wanted to show. The case where \( d = 4, 2, 1 \) are similar.

(b) Standard argument, as in 3-ary or binary case. Surely \( N_1 > 0 \) is suboptimal. Likewise a non-full internal node with a non-leaf is suboptimal. Why is the inequality \( 3N_2 + 3N_4 \leq 4 \) true? This inequality implies that \( (N_2, N_3) \) has only four possibilities: \( (N_2, N_3) \in \{(0, 0), (0, 1), (1, 0), (0, 2)\} \). It is easy to see that in any other case outside these four possibilities, the tree \( T \) is suboptimal. Finally, we want to transform \( T \) to some optimal \( T' \) such that \( N'_2 + N'_3 \leq 1 \), i.e., \( (N'_2, N'_3) \in \{(0, 0), (0, 1), (1, 0)\} \). In other words, we want to avoid the case \( (N_2, N_3) = (0, 2) \). This is easy: we can replace the two internal nodes of degree 3 by one one degree 4 and one of degree 2.

(c) The equation \( n - 1 = N_2 + 2N_3 + 3N_4 \) implies that \( r = N_2 + 2N_3 \pmod{3} \). But \( N_2 + N_3 \leq 1 \) implies \( r = N_2 + 2N_3 \). So, if \( r = 0 \), this implies \( N_2 = N_3 = 0 \). If \( r = 0 \) then \( N_2 = N_3 = 0 \). If \( r = 1 \) then \( N_2 = 1, N_3 = 0 \). If \( r = 2 \) then \( N_2 = 0, N_3 = 1 \).

(d) The algorithm is now exactly as in standard Huffman algorithm: Let \( n = |\Sigma| \), and let \( Q \) be a priority queue comprising 4-ary Huffman trees. Initially, \( Q \) contains \( n \) trees, each with just one leaf. First check if \( r = (n - 1) \pmod{3} = 0 \). If not, we take the smallest \( r + 1 \) elements from \( Q \), and merge them, and reinsert into \( Q \). Now enter an iterative loop, until \( Q \) has one tree left. In this loop, we extract 4 minimal elements, and merge them and reinsert into \( Q \). The output is the single tree left at the end.

(e) Note that \( n = 9 \) and so \( r = 8 \pmod{3} = 2 \). So we first create a subtree \( T_1 \) with weight 3, and with 3 leaves: \( h, e, \Box \). Next we create a subtree \( T_2 \) with with weight 4, and with 4 leaves: \( w, r, d, \Box \). Finally, we create the tree \( T \) with weight 12, and with four children: the leaf \( \ell \) of weight 3, the leaf \( o \) of weight 2, and \( T_1 \) and \( T_2 \). The cost of \( T \) is the sum of the weights of all its internal nodes: \( 12 + 3 + 4 = 19 \).

Comments:

(P2) (25 Points) Convex Hull

Let \( X \subseteq \mathbb{R}^2 \) be a finite set of \( n \) points, assumed to be generic position, i.e., for any two distinct points \( p, q \in X \), \( p.x \neq q.x \) and \( p.y \neq q.y \). Let \( N, S, E, W \) denote the Northernmost, Southernmost, Easternmost and Westernmost points on the convex hull of \( X \).

![Figure 1: NE-hull of X](image)

(a) Note that these four points need not be distinct (for instance, when \( |X| < 4 \)). But assuming \( X \) has \( n \geq 4 \) distinct points, what is smallest possible value for \( |\{N, S, E, W\}| \)?

(b) Give a linear time algorithm to compute these four points. Please read our introductory instructions about how to present algorithms.

(c) Suppose \( N \neq E \), and we want to compute the \( NE-hull \), i.e., the portion of the convex hull from \( N \) to \( E \) as you go around the convex hull in a clockwise manner. See Figure 1. Clearly, the \( NE-hull \) is determined by
the subset of $X$ contained in the interior of the $\Delta(N, A, E)$ as shown in the Figure where $A = (E.x, N.y)$. Implement the predicate $\text{CandidateNE}(q, N, E)$ that returns true iff point $q$ lies in the interior of $\Delta(N, A, E)$. You may use make use of any predicate you know.

(d) Consider the set $X' = X \cap \Delta(N, A, E) = \{p_1, \ldots, p_m\}$. Suppose we first sort $X'$ using the $x$-coordinates of points, say: $p_1 <_x p_2 <_x \cdots <_x p_m$. Describe an efficient algorithm to compute the $NE$-hull of a sorted $X'$. REMARK: In practice, we expect $X'$ to be a small subset of $X$, so sorting $X'$ is much cheaper than sorting $X$.

(e) Give a complexity analysis of your algorithm in (d). HINT: you should be able to exploit amortization.

SOLUTION (a) $|\{N, S, E, W\}|$ can be as small as 2.

(b) We first explain how to compute the point $N$. We maintain a candidate point $c \in X$. Assume $X$ is stored in an array $X[1..n]$ storing the points $p_1, \ldots, p_n$ in this order. Initially, $c = p_1$, and we iterate through the array, comparing successive points $p_i (i = 2, \ldots, n)$ with the current candidate $c$. If $p_i.y > c.y$, then we update $c = p_i$. At the end, we output $c$ as the value of $N$.

The nice thing is that we can use the same loop to compute all the four compass points. Pseudo code:

\begin{verbatim}
NSEW(X, n):
    N ← S ← E ← W ← X[1]
    For i = 2, \ldots, n
        If (N.y < X[i].y), then N ← X[i]
        If (S.y > X[i].y), then S ← X[i]
        If (E.x < X[i].x), then E ← X[i]
        If (W.x > X[i].x), then W ← X[i]
    Return(N, S, E, W)
\end{verbatim}

This is clearly a linear time algorithm.

(c) Make use of the LeftTurn geometric predicate:

\begin{verbatim}
CandidateNE(q, N, E):
    If (q.x < N.x) or (q.y < E.y) Return FALSE.
    If (LeftTurn(E, N, q) = 1) Return TRUE.
    Return FALSE.
\end{verbatim}

Comments:

1 The interior of the triangle refers to those points that are “strictly” inside the triangle, not on the boundary.
(d) This is basically the same the upper hull algorithm in one of the homework. Let $NE[1..n]$ denote an array to contain the $NE$-hull. Let $1 \leq k \leq n$ be the index such the current candidate $NE$-hull is found in $NE[1..k]$. We initialize $NE[1] = N$. Let array $X'[1..m]$ store the sorted set $X'$. For each $p_i$ in $X'$, while $k \geq 2$ and $LeftTurn(p_i, X[k], X[k-1]) \leq 0$, we decrement $k$. On termination of this while-loop, we increment $k$ and add $p_i$ to the $NE$-hull.

$$NEhull(X', m)$$

$\triangleright$ $X'$ is an array of $m \geq 2$ points.

Initialize an array $NE[1..m]$ to contain the $NE$-hull.

$k \leftarrow 1$, $NE[1] \leftarrow X'[1]$

For $i = 2, \ldots, n$

(A) While $(k \geq 2)$ and $(LeftTurn(X'[i], X[k], X[k-1]) \leq 0)$

$k \leftarrow k - 1$

(B) $k \leftarrow Return NE[k]\leftarrow X[i]$

(e) The complexity is $O(m)$, by an amortization argument. The array $NE[1..k]$ is acting like a stack with $k$ elements. The while loop (Line A) amounts to a generalize pop, and the instruction after the while loop (Line B) is a push. One of our homework problems shows that each push-pop operation has an amortized cost of $O(1)$.

Comments: