Massachusetts Institute of Technology — Handout 17
6.046J/18.410J: Introduction to Algorithms — March 7, 2002
Professors Michel Goemans and Piotr Indyk

Problem Set 3

This problem set is due in class on *Thursday, March 21, 2002.*

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material and will be useful in solving the assessed problems. You are responsible for material covered by the exercises. Each problem is to be done on a separate sheet (or sheets) of paper. Mark the top of each sheet with your name, 6.046J/18.410J, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

Exercise 3-1. Do Exercise 12.1-5 in CLRS page 256.

Exercise 3-2. Do Exercise 17.1-2 in CLRS page 409.

Exercise 3-3. Do Exercise 18.2-3 in CLRS page 447.

Problem 3-1. Inversible Traversibles

Consider the three types of Binary Search Tree (BST) walks: preorder, inorder, and postorder. The code for an inorder walk is given on CLRS page 255. Below is the code for preorder and postorder walks:

```
PREORDER-TREE-WALK(x)
if x ≠ NIL
   then  print key[x]
         PREORDER-TREE-WALK(left[x])
         PREORDER-TREE-WALK(right[x])
```

```
POSTORDER-TREE-WALK(x)
if x ≠ NIL
   then  POSTORDER-TREE-WALK(left[x])
         POSTORDER-TREE-WALK(right[x])
         print key[x]
```

(a) Suppose A, an array of n distinct integers, is the output of an **inorder** treewalk of a BST T with n nodes. More precisely, if INORDER-TREE-WALK($root[T]$) is called, $A[1]$ contains the first key printed, $A[2]$ contains the second key printed, etc., for $A[1 \dots n]$.

Is T the only BST that could have generated A as its inorder walk? If so, give an algorithm to construct T given A. If not, give an example showing two trees that generate the same inorder walk. Remember that not all binary trees are BSTs; BSTs all have the **binary-search-tree** property (CLRS pg. 254) .

(b) Do the same for **preorder** walks (answer the question from (a), with all instances of the word "inorder" replaced by "preorder").

(c) Do the same for **postorder** walks (answer the question from (a), with all instances of the word "inorder" replaced by "postorder").

Problem 3-2. Dot-Com Office Manager

You are the office manager for `Iamaka.com`, an internet company feeling the volatile effects of the new post-dot-com economy. Your company goes through endless cycles of hirings and lay-offs, so you are constantly having to move people around the office and either expand or consolidate the space you are renting.

For any n, the rental company can rent out square offices of dimension $n \times n$ that can hold an $n \times n$ grid of cubicles, with enough space for n^2 employees. They only allow companies to maintain square offices.

When someone is hired, you are responsible for assigning them a cubicle. When someone gets fired, you must clear their cubicle. You are also allowed to switch any two employees' cubicles, or move an employee to an empty cubicle. All of these operations are disturbing to normal company operations, so you'd like to avoid doing them when you can.

You design a computer program that maintains an $n \times n$ matrix M representing the square office. Each entry $M(i,j), 0 \leq i,j < n$, holds the name of the employee in cubicle (i,j) in the grid. You have the following operations at your disposal:

- ASSIGN(x,i,j) sets $M(i,j) = x$, and costs \$1.
- REMOVE(i,j) sets $M(i,j) = \langle\text{empty}\rangle$ and costs \$1.
- SWITCH(i_1,j_1,i_2,j_2) switches the values of $M(i_1,j_1)$ and $M(i_2,j_2)$, and costs \$1.
- EXPAND(n') expands the office M from size $n \times n$ to a new office M' of size $n' \times n'$. In the expanded office, $M'(i,j) = M(i,j)$ if $i,j < n$, and $M(i,j) = \langle\text{empty}\rangle$ otherwise. This operation costs $\$(n'^2 - n^2)$, the number of new seats created.
- DOWNSIZE(n') shrinks the office M from size $n \times n$ to a an office M' of size $n' \times n'$. If this operation is applied, all cubicles in the removed portion of the office must be empty. More precisely, for all i,j where $i \geq n'$ or $j \geq n'$, $M(i,j)$ must be $\langle\text{empty}\rangle$. In the new office, $M'(i,j) = M(i,j)$ if $i,j < n'$ This operation costs $\$(n^2 - n'^2)$, the number of cubicles removed.

(a) It is the year 1999, and your business is booming. People are getting hired all the time, and no one is getting fired. Stock options are given to someone who goes out and gets a foosball table for the employees. The company pays for a 3 week white-water rafting trip for the management team. The CEO buys a small country.

Design a HIRE(X) procedure. Your procedure should assign a single new employee x to an empty cubicle. You may use any of the operations above. Start with an empty 0×0 office (the management team has a nice non-square office on the next floor with beautiful carpeting). Show that for any m, m calls to HIRE costs a total of $\$(\Theta(m))$.

(b) It is the year 2001, and the bubble has burst. The foosball table is sold for scrapmetal. Subsidized scuba lessons are no longer given. All the scientists run back to academia to teach classes again.

Design a LAY-OFF(X) procedure. This procedure should remove x from the office entirely. When it is called, you may assume that x is somewhere in the office. The CEO also requires that the office be at least $1/16$ full at all times, or else it seems empty to potential clients. You may use any of the operations above. Start with a full $n \times n$ office, where n is a power of 2. Show that for any m, m calls to LAY-OFF(X) costs a total of $\$(\Theta(m))$.

(c) Now it is the present time, and the future is uncertain. Starting with an empty office, use the potential method to show that any sequence of m calls to LAY-OFF(X) or HIRE(X) costs a total of $\$(\Theta(m))$.

Problem 3-3. Amortized weight-balanced trees

Do Problem 17-3 in CLRS, page 427.

Problem 3-4. Error-Correcting Codes

Flo is an international spy. She must send critical information to base in the form of a k-bit binary string. The problem is that she is deep in enemy territory and her transmitter is weak; every bit she sends to base camp has a chance of being flipped as it goes through the air. For example, if she wanted to send `010`, it might be received as `011`, if the third bit is flipped.

Due to the unreliability of the communication channel, Flo needs to use an *error-correcting code*. Instead of sending the k bits of information, she will send 2^k bits that represent the k bits. These 2^k bits will be sent through the channel instead. Since we would like to be robust against error, we want to use our extra bits to build *redundancy* into the data. For convenience, we define $n = 2^k$.

The two elements that go into an error-correcting code are the *encoder* and the *decoder*. The encoder is a function $\text{ENCODER}(x) : \{0,1\}^k \rightarrow \{0,1\}^n$ that converts k-bit strings to n-bit *code words*. The decoder $\text{DECODER}(y) : \{0,1\}^n \rightarrow \{0,1\}^k$ converts any n-bit string to a k-bit information string.

The communication from Flo to base camp works as follows. Suppose x is the k-bit information string Flo wishes to send. Flo encodes x by setting $y = \text{ENCODER}(x)$, and transmits the

n-bit string y over the channel. The channel corrupts y, so base camp receives y', an n-bit string where y' differs from y in some of the bits. The computers at base camp then apply the decoding function and produce $x' = \text{DECODER}(y')$. Hopefully, if everything worked out, $x' = x$.

In this problem, we will use the amazing properties of universal hash families to design the functions ENCODER and DECODER so that they have some desirable properties.

First we need a few definitions. The *Hamming distance* of two codewords is the number of bits on which they differ. For example, $(0, 1, 1, 0, 0, 1, 1, 0)$ and $(0, 1, 1, 0, 1, 0, 0, 1)$ have a Hamming distance of 4, since 4 of their bits are different. For two binary sequences α and β, we will denote their Hamming distance by $\delta(\alpha, \beta)$.

Let A be the set of all k-bit strings, so $|A| = 2^k = n$. For example, if $k = 3$, $A = \{(0,0,0), (0,0,1), (0,1,0), \ldots, (1,1,0), (1,1,1)\}$. For a particular k-bit vector a, let $\text{INCREMENT}(a)$ increment a by one, as if a was a binary counter. For example, if $a = (0, 1, 0)$, $\text{INCREMENT}(a)$ would change it to $(0, 1, 1)$; calling $\text{INCREMENT}(a)$ again would change a to $(1, 0, 0)$. Let $h_a(x)$ be a function computed as follows:

$$h_a(x) = \sum_{i=0}^{k-1} a_i x_i \mod 2$$

Consider the family of functions $\mathcal{H} = \{h_a : a \in A\}$. This is the same as the universal hash family on page 22 of the L7 notes, setting $m = 2$ and $r = k - 1$. This universal hash family $\mathcal{H}$ is not so valuable for hashing, since it maps to only two buckets. However, it's great for encoding! Define ENCODER as follows:

$\text{ENCODER}(x)$
$a \leftarrow (0, 0, 0, \ldots, 0)$
for $j \leftarrow 0$ **to** $n - 1$
    $y_j \leftarrow h_a(x)$
    $\text{INCREMENT}(a)$
return y

For example, if $k = 3$, $n = 8$, $x = (0, 1, 1)$, $\text{ENCODER}(x)$ returns codeword $y = (\ h_{(0,0,0)}(x),\ h_{(0,0,1)}(x),\ h_{(0,1,0)}(x),\ h_{(0,1,1)}(x),\ h_{(1,0,0)}(x),\ h_{(1,0,1)}(x),\ h_{(1,1,0)}(x),\ h_{(1,1,1)}(x)\ ) = (0, 1, 1, 0, 0, 1, 1, 0)$.

(a) For the case of $k = 3$, $n = 8$ in the example above, compute the output of $\text{ENCODER}(x)$ for each $x \in A$. This should result in 8 different codewords, with 8 bits in each codeword. Feel free to split this computation between study group members, or write a computer program to do it. Now compare pairs of codewords. What is the Hamming distance of each pair of code words you just computed?

(b) Using the properties of universal hash families, show that for any two distinct k-bit information strings v, w, the n-bit codewords produced by $\text{ENCODER}(v)$ and $\text{ENCODER}(w)$ have a Hamming distance of exactly $n/2$.

(c) Let a, b be two vectors from A that differ on exactly one bit. Let's say that they differ on bit i, so $a_i \neq b_i$. Show that $h_a(x) - h_b(x) \mod 2 = x_i$.

(d) Using the observation from part (c), design a decoder $\textsc{Decoder}(y')$ that runs in $\Theta(nk)$ time, such that $\textsc{Decoder}(y') = x$ as long as $\delta(y, y') < n/4$. Be sure to argue your running time, and show why your algorithm exhibits this property.