Suggested Solutions to Midterm Problems

(Compiled on February 27, 2002)

1. Find an open Gray code of length $\lceil \log_2 13 \rceil$ $(= 4)$ for 13 objects. Show how the Gray code is constructed systematically from Gray codes of smaller lengths.

   Solution. Let $(c_1, c_2, \ldots, c_n)^R$ denote the list $c_n, c_{n-1}, \ldots, c_1$.

   Code of length 1 for 2 objects: $0, 1$.

   Code of length 2 for 2 objects: $00, 01$.

   11 has not been used and differs from 01 by 1 bit.

   Code of length 2 for 3 objects: $00, 01, \mathbf{11}$ (which is open).

   Code #1 of length 3 for 3 objects: $000, 001, 011$.

   Code #2 of length 3 for 3 objects: $100, 101, 111$.

   Code of length 3 for 6 objects: $000, 001, 011, (100, 101, 111)^R$.

   Code #1 of length 4 for 6 objects: $0000, 0001, 0011, 0111, 0101, 0100$.

   Code #2 of length 4 for 6 objects: $1000, 1001, 1011, 1111, 1101, 1100$.

   Code of length 4 for 12 objects:

   $0000, 0001, 0011, 0111, 0101, 0100,\ (1000, 1001, 1011, 1111, 1101, 1100)^R$

   $= 0000, 0001, 0011, 0111, 0101, 0100, 1100, 1101, 1111, 1011, 1001, 1000$.

   1010 has not been used and differs from 1000 by 1 bit.

   Code of length 4 for 13 objects:

   $0000, 0001, 0011, 0111, 0101, 0100, 1100, 1101, 1111, 1011, 1001, 1000, \mathbf{1010}$ (which is open). □

2. Let $a_1, a_2, \cdots, a_n$ be positive real numbers such that $a_1 a_2 \cdots a_n = 1$. Prove *by induction* that $(1+a_1)(1+a_2)\cdots(1+a_n) \geq 2^n$. (Hint: In the inductive step, try introducing a new variable that replaces two chosen numbers from the sequence.)

   Solution. The proof is by induction on n.

   Base case ($n = 1$): $a_1 = 1$. So, $(1 + a_1) = 2 \geq 2^1$.

   Inductive step ($n > 1$): In any sequence $a_1, a_2, \cdots, a_n$ ($n > 1$) of positive real numbers where $a_1 a_2 \cdots a_n = 1$, there must exist two numbers a_i and a_j such that $a_i \geq 1$ and $a_j \leq 1$. Without loss of generality, we assume that the two numbers are a_{n-1} and a_n (this can always be achieved by swapping numbers in the sequence). As $(1 - a_{n-1})(1 - a_n) \leq 0$, it follows that $a_{n-1} + a_n \geq 1 + a_{n-1}a_n$. Let a'_{n-1} be the number equal to $a_{n-1}a_n$ (which is also a positive real number) so that $a_1 a_2 \cdots a_{n-2} a'_{n-1} = a_1 a_2 \cdots a_{n-2} a_{n-1} a_n = 1$.

   $(1+a_1)(1+a_2)\cdots(1+a_{n-2})(1+a_{n-1})(1+a_n) = (1+a_1)(1+a_2)\cdots(1+a_{n-2})(1+a_{n-1}+a_n+a_{n-1}a_n) \geq (1+a_1)(1+a_2)\cdots(1+a_{n-2})((1+a_{n-1}a_n)+(1+a_{n-1}a_n)) = 2(1+a_1)(1+$

$a_2)\cdots(1+a_{n-2})(1+a_{n-1}a_n) = 2(1+a_1)(1+a_2)\cdots(1+a_{n-2})(1+a'_{n-1})$, which from the induction hypothesis is $\geq 2 \times 2^{n-1} = 2^n$. □

3. Below is an algorithm for solving a variant of the Towers of Hanoi puzzle with an additional fourth peg D; `Towers_Hanoi` is an algorithm for the original puzzle.

```
Algorithm Four_Towers_Hanoi(A,B,C,D,n);
begin
   if n<=2 then
      Towers_Hanoi(A,B,C,n);
   else
      Four_Towers_Hanoi(A,D,B,C,n-2);
      Towers_Hanoi(A,B,C,2);
      Four_Towers_Hanoi(D,B,C,A,n-2);
end;
```

Let $T(n)$ denote the number of moves needed for n disks. Write a recurrence relation for $T(n)$ and solve it.

Solution. `Towers_Hanoi(A,B,C,1)` takes 1 move, while `Towers_Hanoi(A,B,C,2)` takes 3 moves. A recurrence relation for $T(n)$ is the following:

$$\begin{array}{l} T(1) = 1 \\ T(2) = 3 \\ T(n) = 2T(n-2)+3, \text{for } n \geq 3 \end{array}$$

We solve the recurrence relation by considering odd and even n's separately.

When n (≥ 3) is odd,

$$\begin{array}{rl} T(n) = & 2T(n-2)+3 \\ 2T(n-2) = & 2(2T(n-4)+3) = 2^2T(n-4)+2\times 3 \\ 2^2T(n-4) = & 2^2(2T(n-6)+3) = 2^3T(n-6)+2^2\times 3 \\ \cdots & \cdots \\ 2^{\frac{n-3}{2}}T(3) = & 2^{\frac{n-3}{2}}(2T(1)+3) = 2^{\frac{n-1}{2}}+2^{\frac{n-3}{2}}\times 3 \\ \hline T(n) = & 2^{\frac{n-1}{2}}+3\times(2^{\frac{n-1}{2}}-1) \\ = & 2^{\frac{n+3}{2}}-3 \end{array}$$

When n (≥ 3) is even,

$$\begin{array}{rl} T(n) = & 2T(n-2)+3 \\ 2T(n-2) = & 2(2T(n-4)+3) = 2^2T(n-4)+2\times 3 \\ 2^2T(n-4) = & 2^2(2T(n-6)+3) = 2^3T(n-6)+2^2\times 3 \\ \cdots & \cdots \\ 2^{\frac{n-4}{2}}T(4) = & 2^{\frac{n-4}{2}}(2T(2)+3) = 3\times 2^{\frac{n-2}{2}}+2^{\frac{n-4}{2}}\times 3 \\ \hline T(n) = & 3\times 2^{\frac{n-2}{2}}+3\times(2^{\frac{n-2}{2}}-1) \\ = & 3\times 2^{\frac{n}{2}}-3 \end{array}$$

□

4. Show all intermediate and the final AVL trees formed by inserting the numbers 1, 7, 2, 6, 3, 5, and 4 (in this order). If a rotation is performed during an insertion, please also show the tree before the rotation.

   Solution. See the attached. □

5. The Knapsack Problem is defined as follows: Given a set S of n items, where the ith item has an integer size $S[i]$, and an integer K, find a subset of the items whose sizes sum to exactly K or determine that no such subset exists.

   Below is an algorithm for determining whether a solution to the problem exists.

```
Algorithm Knapsack ($(S, K)$);
begin
    $P[0,0].exist := true$;
    for $k := 1$ to $K$ do
        $P[0,k].exist := false$;
    for $i := 1$ to $n$ do
        for $k := 0$ to $K$ do
            $P[i,k].exist := false$;
            if $P[i-1,k].exist$ then
                $P[i,k].exist := true$;
                $P[i,k].belong := false$
            else if $k - S[i] \geq 0$ then
                    if $P[i-1,k-S[i]].exist$ then
                        $P[i,k].exist := true$;
                        $P[i,k].belong := true$
end
```

   (a) Modify the algorithm to solve a variation of the knapsack problem where each item has an unlimited supply. In your algorithm, please change the type of $P[i,k].belong$ into integer and use it to record the number of copies of item i needed.

   Solution. It suffices to modify the last five lines before "end" as follows:

```
                $P[i,k].belong := 0$;
            else $P[i,k].belong := 0$;
                $j := 1$;
                while $k - S[i] \times j \geq 0$ do
                        if $P[i-1,k-S[i] \times j].exist$ then
                $P[i,k].exist := true$;
                $P[i,k].belong := j$;
                break;
            $j := j+1$;
```

□

(b) Design an algorithm to recover the solution recorded in the array P of the algorithm in (a).

Solution.

```
Procedure Print_Solution $(S, P, n, K)$;
begin
    if $\neg P[n,K].exist$ then
        print "no solution"
    else $i := n$;
         $k := K$;
         while $k > 0$ do
             if $P[i,k].belong > 0$ then
                 print $i, P[i,k].belong$;
                 $k := k - S[i] \times P[i,k].belong$;
             $i := i - 1$
end
```

□

6. Given as input two sorted arrays A and B, each of n numbers (in an increasing order), and another number x, design an algorithm with running time $O(n)$ to determine whether there exist an element in A and an element in B whose sum is exactly x. (Hint: Recall the ideas of the $O(n)$ soluton to the Celebrity Problem discussed in class.)

Solution. The basic idea is the following: If $A[1]+B[n] < x$, then $A[1]$ cannot be the number in A we are looking for, as $A[1]+B[j]$ will be smaller than x for any $j < n$. On the other hand, if $A[1]+B[n] > x$, then $B[n]$ cannot be the number in B we are looking for. In either case, we eliminated one element from either array.

```
Algorithm Find_Sum $(A, B, n, x)$;
begin
    $i := 1$;
    $j := n$;
    while $i \leq n$ and $j \geq 1$ do
        if $A[i] + B[j] = x$ then
            break;
        if $A[i] + A[j] < x$ then
            $i := i + 1$
        else $j := j - 1$;
    if $i \leq n$ and $j \geq 1$ then
        print "yes"
    else print "no"
end
```

The while loop will be executed at most $2n - 1$ times, hence the running time of the algorithm is $O(n)$. $\square$

7. Apply the quicksort algorithm to the following array. Show the contents of the array after each partition operation.

5	1	8	11	2	12	7	3	6	10	4	9

Solution.

5	1	8	11	2	12	7	3	6	10	4	9
2	1	4	3	**5**	12	7	11	6	10	8	9
1	**2**	4	3	**5**	12	7	11	6	10	8	9
1	**2**	3	**4**	**5**	12	7	11	6	10	8	9
1	**2**	3	**4**	**5**	9	7	11	6	10	8	**12**
1	**2**	3	**4**	**5**	6	7	8	**9**	10	11	**12**
1	**2**	3	**4**	**5**	**6**	7	8	**9**	10	11	**12**
1	**2**	3	**4**	**5**	**6**	**7**	8	**9**	10	11	**12**
1	**2**	3	**4**	**5**	**6**	**7**	8	**9**	**10**	11	**12**

$\square$

8. Below is a variation of the n-coins problem.

> You are given a set of n coins $\{c_1, c_2, \ldots, c_n\}$, among which at least $n - 1$ are identical "true" coins and at most one coin is "false". A false coin is *lighter* or *heavier* than a true coin. Also, you are given a balance scale, which you may use to compare the total weight of any m coins with that of any other m coins. The problem is to find the "false" coin, or show that there is no such coin, by making some sequence of comparisons using the balance scale.

Show that in the worst case it is impossible to solve the n-coins problem with k comparisons (for any n and k) if $n > \frac{(3^k-1)}{2}$. (Hint: Think about decision trees and how many possible outcomes there can be for the problem.)

Solution. Each use of the balance scale may produce three possible results. Solutions to the n-coins problem fall within the model of decision trees where each internal node has three branches. Any such tree of height k can contain at most 3^k leaves, representing at most 3^k different outcomes.

For n coins, there are $2n+1$ possible outcomes: (a) one of the n coins is lighter (n possibilities), (b) one of the n coins is heavier (another n possibilities), and (c) none of the coins is false (one possibility). Therefore, 3^k must be greater than or equal to $2n+1$ for a solution with k comparisons to exist. In other words, no solution with k comparisons exists if $2n+1 > 3^k$, i.e., if $n > \frac{(3^k-1)}{2}$. □

9. Draw a Huffman tree for a text that contains eight characters A, B, C, D, E, F, G, and H with frequencies 8, 2, 5, 6, 14, 3, 2, and 4, respectively.

   Solution. See the attached. □