Suggested Solutions to Midterm Problems

(Compiled on May 8, 2002)

1. Find an open Gray code of length $\lceil \log_2 15 \rceil$ $(= 4)$ for 15 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: $000, 001, 011, (100, 101, 111)^R = 000, 001, 011, 111, 101, 100$.
   110 has not been used and differs from 100 by 1 bit.
   Code of length 3 for 7 objects: $000, 001, 011, 111, 101, 100, \mathbf{110}$ (which is open).
   Code #1 of length 4 for 7 objects: $0000, 0001, 0011, 0111, 0101, 0100, 0110$.
   Code #2 of length 4 for 7 objects: $1000, 1001, 1011, 1111, 1101, 1100, 1110$.
   Code of length 4 for 14 objects:
   $0000, 0001, 0011, 0111, 0101, 0100, 0110,\ (1000, 1001, 1011, 1111, 1101, 1100, 1110)^R$
   $= 0000, 0001, 0011, 0111, 0101, 0100, 0110, 1110, 1100, 1101, 1111, 1011, 1001, 1000$.
   1010 has not been used and differs from 1000 by 1 bit.
   Code of length 4 for 15 objects:
   $0000, 0001, 0011, 0111, 0101, 0100, 0110,\ (1000, 1001, 1011, 1111, 1101, 1100, 1110)^R$
   $= 0000, 0001, 0011, 0111, 0101, 0100, 0110, 1110, 1100, 1101, 1111, 1011, 1001, 1000, \mathbf{1010}$ (which is open). $\square$

2. What is wrong with the following proof?

   Claim: In any non-empty set of horses, all horses are of the same color.

   Proof: By induction on the size n of a set.
   Base case: $n = 1$, there is just one horse and it has the same color as its own.
   Inductive step: $n = k + 1, k \geq 1$. Consider any set H of size $k + 1$. Remove one horse h from H to get H_1 of size k. From the induction hypothesis, all horses in H_1 are of the same color. Put the removed horse h back and remove a different horse from H to get H_2 of size k. Again, from the induction hypothesis, all horses in H_2 are of the same color. H_2 contains

horse h and some other horses from H_1. It follows that horse h has the same color as those in H_1 and, therefore, all horses in H are of the same color.

Solution. The inductive step is not valid when $k = 1$. In this case, H_1 contains only one horse and so does H_2; in particular, H_2 contains horse h and *no other horses* from H_1. Therefore, it is incorrect to infer that horse h has the same color as those in H_1 and thereafter conclude that all horses in H are of the same color. $\square$

3. Below is a theorem from Manber's book:

   > For all constants $c > 0$ and $a > 1$, and for all monotonically increasing functions $f(n)$, we have $(f(n))^c = O(a^{f(n)})$.

   Prove, by using the above theorem, that $n^2(\log n)^4 = O(n^{2.5})$.

   Solution. If we are able to show that $(\log n)^4 = O(n^{.5})$, then $n^2(\log n)^4 = O(n^2{\cdot}n^{.5}) = O(n^{2.5})$.
   Applying the theorem with $f(n) = \log n$, $c = 4$, and $a = 2^{.5}$, we have $(\log n)^4 = O((2^{.5})^{\log n}) = O(2^{.5 \log n}) = O(2^{\log n^{.5}}) = O(n^{.5})$.
   (Note: As usual, we have assumed the base of logarithm is 2. The same result can still be obtained even if a different base is used.) $\square$

4. Show all intermediate and the final AVL trees formed by inserting the numbers 2, 4, 5, 8, 7, 6, 3, and 1 (in this order). Please use the following ordering convention: the key of an internal node is larger than that of its left child and smaller than that of its right child. If a rotation is performed during an insertion, please also show the tree before the rotation.

   Solution. Please see the attached. $\square$

5. Suppose that you are given an algorithm as a *black box* (you cannot see how it is designed) that has the following properties: If you input any sequence of real numbers and an integer k, the algorithm will answer "yes" or "no," indicating whether there is a subset of the numbers whose sum is exactly k. Show how to use this black box to find the subset whose sum is k, if it exists. You should use the black box $O(n)$ times (where n is the size of the sequence).

   Solution. Let `Find_Subset` denote the given algorithm, which takes as input an array of real numbers, the size of the array (these two together representing the sequence of real numbers), and an integer.

```
Algorithm Print_Subset(S,n,k);
begin
   if Find_Subset(S,n,k)="no" then
      print "No suitable subset"; halt;
   print "Below is a suitable subset:";
   sum := 0.0;
```

```
    i := 1;
    while sum<k do
       this := S[i];
       S[i] := 0;
       if Find_Subset(S,n,k)="no" then
          print this;
          sum := sum + this;
          S[i] := this;
       i := i + 1;
end
```

□

6. 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
   $\quad P[0,0].exist := true;$
   $\quad$**for** $k := 1$ **to** K **do**
   $\quad\quad P[0,k].exist := false;$
   $\quad$**for** $i := 1$ **to** n **do**
   $\quad\quad$**for** $k := 0$ **to** K **do**
   $\quad\quad\quad P[i,k].exist := false;$
   $\quad\quad\quad$**if** $P[i-1,k].exist$ **then**
   $\quad\quad\quad\quad P[i,k].exist := true;$
   $\quad\quad\quad\quad P[i,k].belong := false$
   $\quad\quad\quad$**else if** $k - S[i] \geq 0$ **then**
   $\quad\quad\quad\quad\quad$**if** $P[i-1,k-S[i]].exist$ **then**
   $\quad\quad\quad\quad\quad\quad P[i,k].exist := true;$
   $\quad\quad\quad\quad\quad\quad P[i,k].belong := true$
   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. Insert "$P[0,0].belong := 0;$" after "$P[0,0].exist := true;$" and modify the last five lines before "end" as follows:

$P[i,k].belong := 0$
else if $k - S[i] \geq 0$ **then**
if $P[i, k - S[i]].exist$ **then**
$P[i,k].exist := true$;
$P[i,k].belong := P[i, k - S[i]].belong + 1$

□

(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

□

7. 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 $i \le n$ and $j \ge 1$ **do**
  if $A[i] + B[j] = x$ **then**
    break;
  if $A[i] + A[j] < x$ **then**
    $i := i + 1$
  else $j := j - 1$;
if $i \le n$ and $j \ge 1$ **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 $O(n)$. □

8. Rearrange the following array into a (max) heap using the bottom-up approach.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	2	5	11	10	14	7	6	8	1	13	4	15	12	9

Show the result after each element is added to the part of array that already satisfies the heap property.

Solution.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	2	5	11	10	14	7	6	8	1	13	4	15	12	9
3	2	5	11	10	14	(**12**)	6	8	1	13	4	15	(**7**)	9
3	2	5	11	10	(**15**)	12	6	8	1	13	4	(**14**)	7	9
3	2	5	11	(**13**)	15	12	6	8	1	(**10**)	4	14	7	9
3	2	5	11	13	15	12	6	8	1	10	4	14	7	9
3	2	(**15**)	11	13	(**14**)	12	6	8	1	10	4	(**5**)	7	9
3	(**13**)	15	11	(**10**)	14	12	6	8	1	(**2**)	4	5	7	9
(**15**)	13	(**14**)	11	10	(**5**)	12	6	8	1	2	4	(**3**)	7	9

□

9. Draw a Huffman tree for a text with the following frequency distribution: $A:9$, $B:3$, $C:6$, $D:5$, $E:16$, $F:4$, $G:2$, and $H:1$.

Solution. Please see the attached. □

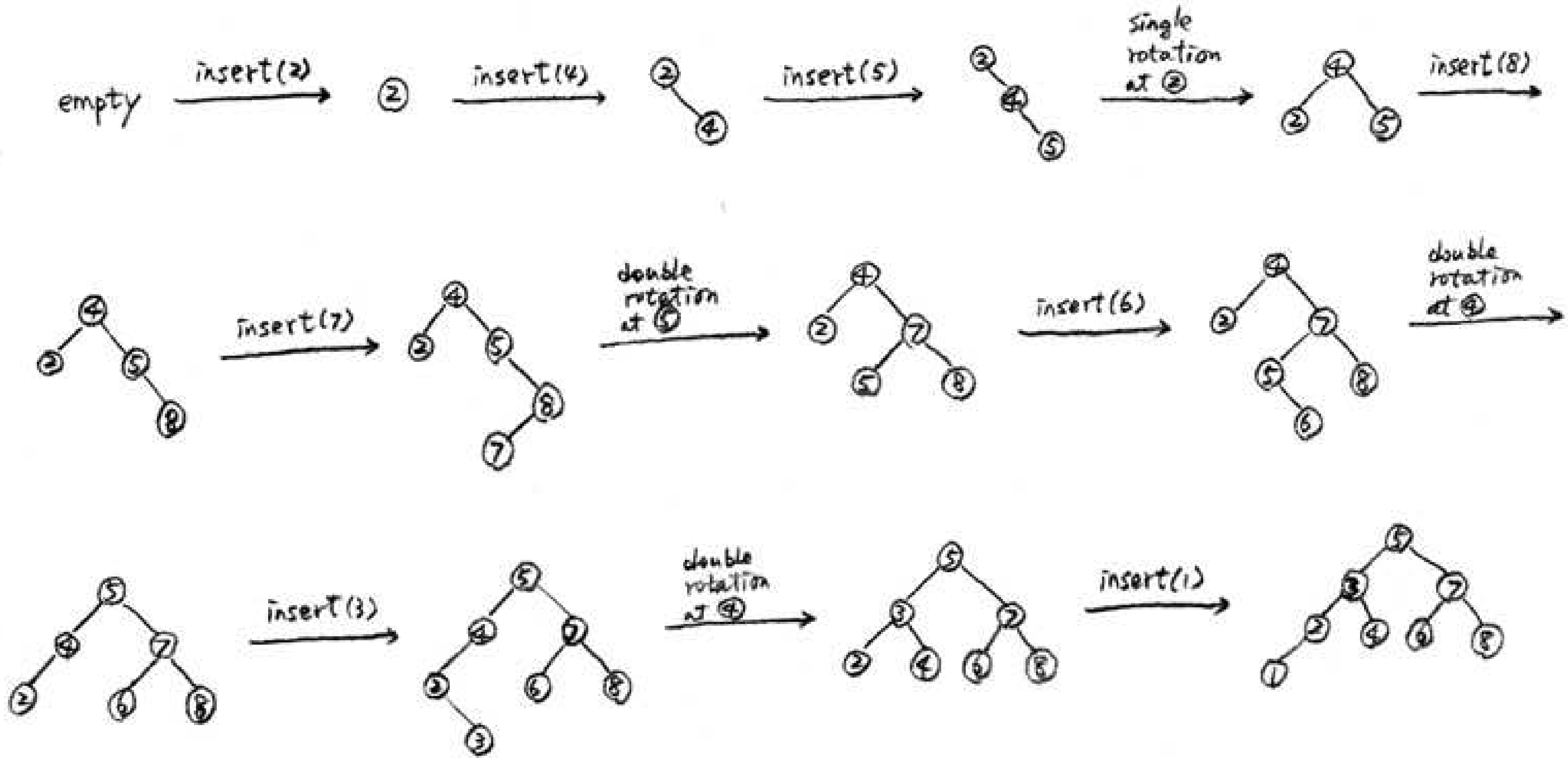
10. Compute the *next* table as in the KMP algorithm for the string *aababaabaab*. Please show the details of how *next*[11] is computed using *next*[1..10].

Solution.

1	2	3	4	5	6	7	8	9	10	11
a	a	b	a	b	a	a	b	a	a	b
-1	0	1	0	1	0	1	2	3	4	2

$next[11] = 2$: $B_{11-1} = B_{10} = a$. $B_{next[11-1]+1} = B_{4+1} = B_5 = b \neq a$; $B_{next[5]+1} = B_{1+1} = B_2 = a = B_{10}$. So, $next[11] = 2$. $\square$

4.



empty
insert(2)
insert(4)
insert(5)
single rotation at 2
insert(8)
insert(7)
double rotation at 5
insert(6)
double rotation at 4
insert(3)
double rotation at 4
insert(1)

9.