

# JOINT ENTRANCE EXAM IN COMPUTER SCIENCE 2001

May 25<sup>th</sup> 2001

**You may solve two problems at the most.** If the examinee has solved all three problems, the *best* solution will be excluded from evaluation.

---

## PROBLEM 1

A group of friends, Heidi, Kate, Larry, Mike, Nora, Patty, Rick and Sam are spending the evening together. They have enough beverages, but are overcome by hunger suddenly. They decide to order some pizzas.

Heidi wants a pizza with bell peppers and pineapple. Kate also wants a pizza with pineapple but also with tuna. Larry likes bell peppers and wants salami, as well. Mike wants a pizza with garlic and pineapple. Nora wants a pizza with ham and mushrooms, and so does Patty. Rick wants ham and pineapple in his pizza. Sam wants salami and bell peppers.

Pizzas can be bought in two sizes. The smaller one can have three toppings at the most, and gives two slices. The larger one takes six toppings at the most, and gives four slices. The small pizza costs FIM 40 and the large one FIM 70. The number of toppings on the pizza has no bearing on the price.

Each of the friends is content with one slice, and their pizza may contain other toppings in addition to the ones they prefer.

- a) The friends want to order their pizzas as cheaply as possible. How should they order? (There can be leftovers.) (5 points)
- b) Give clear and logical reasons for why you answered as you did in a). (10 points)
- c) Present a general model for how to conclude which pizzas should be ordered. The sizes and prices of the pizzas are the ones presented above. The number of people and their wishes may vary from that presented above, but each of them orders one slice with at least two toppings. The number of toppings to choose from may be larger than presented above. (10 points)

The evaluation of the solution to this problem will be affected by the price of the order and the clarity and logic of the solution.

---

## PROBLEM 2

Write a program that reads a set of positive integers, finds the pair of integers that are the closest in value to one another (i.e. such integers  $a$  and  $b$  that the absolute value of the difference  $a - b$  is less or the same as the absolute value of the difference  $c - d$  for any pair of integers  $c$  and  $d$  in the set) and outputs it. You may assume that there are no two numbers of the same value, and there are at least 2 and at most 100 integers in the set.

*Example:* The program is given the numbers 2 35 7 43 9 10 57 34 5 52 13 and 135. It should output the numbers 35 and 34 or 9 and 10.

---

### **PROBLEM 3** (based on the attached material)

(1)

- (a) What is the root node of the adjoining tree?
- (b) Which are inner nodes in the adjoining tree?
- (c) Which are leaf nodes on the adjoining tree?

*Figure 1*

(2)

- (a) Which images of (a) – (e) in Figure 2 are binary trees? Explain why the other images of (a) – (e) in Figure 2 are not binary trees.
- (b) Which images of (a) – (e) in Figure 2 are binary search trees? Explain why the other binary trees are not binary search trees.

*Figure 2.*

(3) Draw all the possible search trees containing keys 1, 2 and 3 (each key must appear exactly once in each tree).

(4) Create the binary search trees (a) and (b) by adding to an empty tree the keys in the given order at points (a) and (b) (use the key insertion method presented in the appendix). Draw each step clearly on paper. (At point (a), for example, you have to draw four separate binary search trees; first the tree with key number 4, then the tree with keys 4 and 3, etc.)

- (a) The keys 4, 3, 2 and 1.
  - (b) The keys 5, 2, 4, 8, 10, 9, 3, 7, 1 and 6.
-

## Material (the questions in problem 3 are based on this material)

A *binary tree* is a data structure with several different applications. *Binary search trees*, for example, as the name says, are used to search for information in trees.

A binary tree is specified as follows. A binary tree either does not contain any nodes, or it consists of a *root node* and two separate groups of nodes: the *left* and the *right* subtree. The left and right subtrees are also binary trees. If a binary tree does not contain any nodes, it is an *empty tree*. Binary trees have a finite number of nodes.

If a node  $x$  in a binary tree has a non-empty left subtree  $t$ , the root node of subtree  $t$  is called the *left child* of node  $x$ . Correspondingly, the root node of a non-empty right subtree is the *right child* of node  $x$ . If one of the subtrees is empty, node  $x$  *lacks* a child.

Nodes without any children are called *leaf nodes*. If node  $x$  has a child  $y$ , the node  $x$  is the *parent* of child  $y$ . The root node is the only node that has no parents. Each node has one parent at the most. Nodes that are not leaf nodes or root nodes are *inner nodes*.

When drawing binary trees you should pay attention to the fact that if a binary tree node has only one child node, the child node should be clearly placed in either the left or the right subtree. Two trees are not the same, even in case their only difference is the position of the child node in either the left or the right subtree.

Figure 1 (a) shows a binary tree. Its root node contains the number 6. The nodes containing the numbers 2, 5 and 8 are leaf nodes. The rest of the nodes are inner nodes. Node 3 is the parent of nodes 2 and 5, and the child of node 6. The information in a node is called the *key*; the keys 2, 3, 5, 6, 7 and 8 are saved in this tree. As seen before, nodes are referred to by their keys. Thus, speaking of node 3 and node 5, for example, refers to the node in which the key 3 has been saved and the node in which the key 5 has been saved.

*Figure 1.*

The trees in figures 1 (b) and 1 (c) are also binary trees. Please note that the trees in (a) and (b) are not the same.

A tree is a binary search tree if, for all its nodes  $x$ , it is true that the node keys in the left subtree are less than the key  $x$ . Correspondingly, the keys in the right subtree of node  $x$  should be greater than the key  $x$ . In figures 1 (a) and (c), the trees are binary search trees.

The program described below searches for the key  $a$  in a tree. It outputs the node containing key  $a$  from the tree containing root node  $s$ . In addition, the parent  $y$  of node  $a$  is returned. If the key  $a$  is not found in the tree, the empty node  $x$  and the node  $y$  where the search for node  $a$  terminated, are returned. Node  $y$  may also be empty if the tree is empty or if the key is in the root.

---

**Searching for key  $a$  in a tree with the root node  $s$ .**

- (1) Put  $y :=$  empty.
  - (2) Put  $x := s$ .
  - (3) Repeat lines (4) – (8) until node  $x$  ? empty and  $a$  ? node  $x$
  - (4) Put  $y := x$ .
  - (5) If  $a <$  node  $x$ , then
  - (6) put  $x :=$  the root node of the left subtree of node  $x$ ,
  - (7) else
  - (8) put  $x :=$  the root node of the right subtree of node  $x$ .
  - (9) Return node  $x$  and its parent  $y$ .
- 

Assume that we are searching for key 5 in the binary search tree in figure 1 (a). The tree root node  $s$  contains the key 6. To begin with, the node  $x$  is given the value  $s$ , which is 6, and node  $y$  is empty. Since node  $x$  is not empty and the keys are not the same, assign the variable  $x$  on line (6) the root node of the left subtree (node 3).

Now node  $x$  on line (3) is not empty, and the keys are not the same, so variable  $x$  on line (8) is assigned the value node 5. Since the keys are the same, lines (4) – (8) are not repeated. Thus node  $x$  (whose key is the same as what was searched for) and its parent, node 3 (the value of variable  $y$ ), are returned on line (9).

If the key 4 would be searched for, the value of variable  $x$  on lines (4) – (8) would eventually be the root node of the left subtree. Since node 5 has no subtree, variable  $x$  would have the value empty. The variable  $y$  would have the value node 5. These nodes  $x$  and  $y$  would also be given as results (i.e. there is no key 4 in the tree and the search would end at node 5).

The following presents adding a key to a binary tree. The idea is to find a suitable point where the node containing the key may be added. After that, the node is added as either a root or as the left or right child. To find a suitable point, the key search presented above may be used.

---

**Inserting the node containing key  $a$  into the binary tree  $t$ .**

- (1) Find key  $a$  in tree  $t$  with the method presented above. Let  $x$  be the node given by the method and  $y$  its parent.
  - (2) If node  $y$  is empty, then
  - (3) give tree  $t$  node  $a$  as root
  - (4) else if node  $a <$  node  $y$ , then
  - (5) add node  $a$  as the left child to node  $y$
  - (6) else
  - (7) add node  $a$  as the right child to node  $y$ .
- 

Note that node  $x$  is not needed to run the end of the program. It is included because the program used for searching for keys returns two nodes ( $x$  and  $y$ ).

The binary tree in figure 1 (a), for example, has been created by adding the keys in the order 6, 7, 8, 3, 2 and 5 to an empty tree. Another possible order would be 6, 3, 7, 2, 5 and 8. If a node with the key 4 is added to the tree, node 5 (the value of variable  $y$ ) is first searched for. After this, node 4 is given as the left child of node 5 on lines (2) – (7).