British Informatics Olympiad Final

12–14 April, 1996

Sponsored by Data Connection

Trees, Forests and Heaps of leaves

Trees are one of the most common structures to be found in computing. They are frequently used in programming, as well as more mundane everyday usage such as directory structures. We will begin with a few definitions and explanations - just to remind you of the terminology.

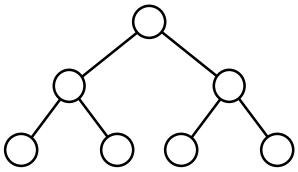


Figure (i)

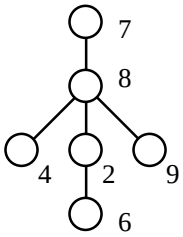


Figure (ii)

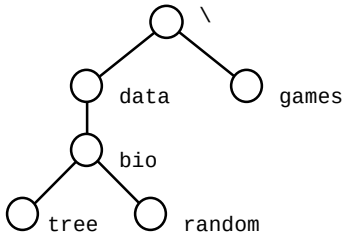


Figure (iii)

These three diagrams are examples of **tree**s. The circles are known as **node**s, and the lines between them, which indicate how and which nodes are connected, are called **edge**s. Nodes can be labelled (the value of a node is its label): (i) shows unlabelled nodes, (ii) shows nodes labelled with numbers, and (iii) is an example of nodes labelled with directory names.

You will notice the diagrams, as with all trees, have no cycles. In other words you cannot start at a node, move along different edges (but never the same one twice) and get back to your starting place.

For this question all our trees will be rooted. This means we distinguish one of the nodes from the others and call it the **root**. We will always draw this node at the top of the tree. In diagram (ii) 7 is the root node, and in diagram (iii) it is \. When we start an algorithm we will always start from the root.

The nodes connected directly to the root are its **children**, and the root is their **parent**. In general the parent of node A is the first node you encounter when travelling from A to the root - there can only ever be one way to do this without re-using edges, so the parent of a node is unique. If A is the parent of B then B is the child of A. All nodes have a single parent, except the root which has none. We conventionally draw the children of a node below it.

For example, in diagram (ii) the children of 8 are 4, 2 and 9; the parent of 6 is 2; and 4 has no children. In diagram (iii) the only child of *data* is *bio*; both *random* and *tree* have the parent *bio*; and \ has no parent.

We can split our tree into **level**s. Level 0 contains only the root, level 1 contains root's children, level 2 contains root's children's children etc... In general level *n* contains the children of all the nodes in level *n-1*. The **depth** of a tree is the highest number level that exists. The depths of the three diagrams are 2, 3 and 3 respectively.

1: Binary Trees

A **binary tree** is a tree where each node contains at most 2 children. Children are defined to be either **left-children** or **right-children** (with a node having no more than one of each). Diagram (iv) shows a binary tree where children are drawn staggered left or right to indicate what type of child they are.

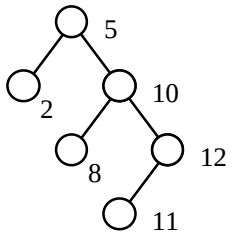


Figure (iv)

Q1.1
a) Draw a binary tree with 8 nodes which has the least possible depth.
b) Draw a binary tree with 8 nodes which has the largest possible depth.

A **binary search tree** (of numbers) has the additional constraint that any left-child of a node must have a value ≤ the node, and any right-child has a value ≥ the node. Diagram (iv) is a binary search tree.

Q1.2
a) Suppose both your trees for question 1.1 were binary search trees. If we are looking to see if a given number is in the tree, the best case would be when the required number is at the root - this takes a single comparison. What is the largest number of comparisons that may be required to find an element for both trees?
b) What would the largest number of comparisons if question 1.1 had asked for trees with 15 nodes?

2: Binary Heaps

A tree is **heap-ordered** if all the children of each node always have a value ≥ to that of their parent. A binary tree is **complete** if it is completely filled on all levels, except possibly the lowest which is filled up as far as possible from the left. Diagram (v) shows a tree that is complete but not heap-ordered. Diagram (vi) shows a tree that is heap-ordered but not complete.

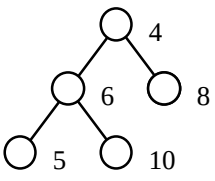


Figure (v)

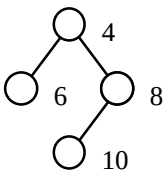


Figure (vi)

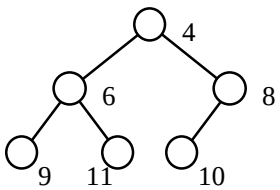


Figure (vii)

A **binary heap** is a heap-ordered, complete binary tree. Diagram (vii) shows a binary heap.

Q2.1
Draw a binary heap where the nodes have the values 1, 2, 4, 5, 7 and 11.

Q2.2
In general, how many operations (comparisons) might it take in the worst case to find a) the minimum element and b) the maximum one?

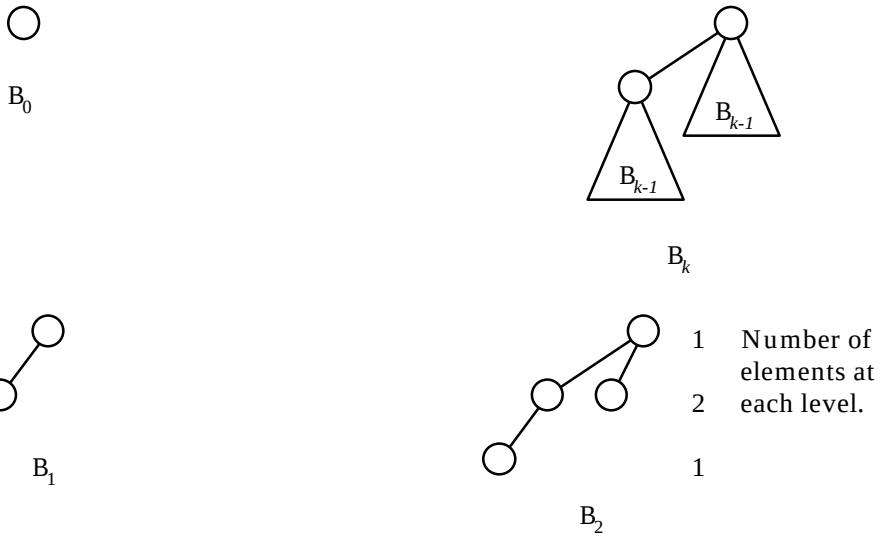
Q2.3
a) How would you add a node to a binary heap to produce another binary heap?
b) Add a node labelled 7 to the binary heap in diagram (vii)., using your method from part (a).

It is an important characteristic of heaps that we can arrange any n elements into a heap in linear time. In other words there is some fixed constant C, and we never require more than $C \times n$ operations to generate the heap.

Q2.4
a) How would you add two binary heaps together, producing another binary heap? The two heaps may be different sizes.
b) Give an estimate (or a sensible upper bound) on the number of operations this would take in the worst case.

3: Binomial Heaps

We can define **binomial trees** recursively as follows. B_0 is a single node, and B_k is made up of two B_{k-1} binomial trees with the root of one being the leftmost child of the root of the other.



We say B_k has degree k.

Q3.1
Draw B_4, indicating the number of nodes at each level (as shown with B_2 above).

Q3.2
How many nodes are there in B_n?

A set of trees is called a **forest**. A **binomial heap** is a forest of binomial trees under the two conditions:

1) There is at most one binomial tree of each degree.
2) Each binomial tree is heap-ordered.

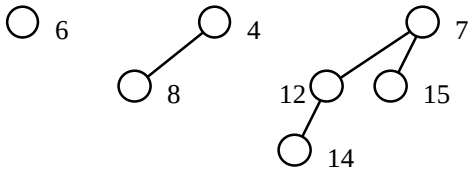


Figure (viii)

Q3.3

a) How would you go about finding the minimum element of a binomial heap?
b) Is this faster or slower than finding the minimum of the binary heaps in part 2?

Q3.4

a) How would you join together two heap ordered binomial trees of the same degree (to produce a binomial tree)?
b) What do you get by joining diagrams (ix) and (x) together?

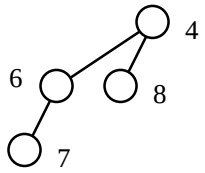


Figure (ix)

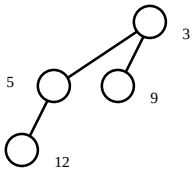


Figure (x)

Q3.5

a) How would you join together two binomial heaps to produce another binomial heap?
b) How does this compare to joining two balanced binary heaps together?

British Informatics Olympiad Final

12–14 April 1996

Sponsored by Data Connection

Random Numbers

Suppose you are given a file which you are told contains random bytes. How would you go about testing this? You are free to mention anything you feel would be relevant, such as algorithms and time/space considerations. There is no need to go into details on any statistical tests, other than to mention where (and on what) you would use one.

British Informatics Olympiad Final

12–14 April, 1996

Sponsored by Data Connection

Connecting Towns

Given a list of roads directly connecting pairs of towns, you are required to find how many disjoint sets of towns there are. Two towns are in the same set if it is possible to reach one from the other by one road or a combination of roads. Two towns are in disjoint sets if it is not possible to get between the towns on the given roads.

For example there are two disjoint sets, {AAAAA,BBBBB,CCCCC} and {DDDDD,EEEEE} in the following map :

AAAAA ----- BBBBB ----- CCCCC

DDDDD ----- EEEEE

Your input data will consist of a list of roads (given by the two towns they connect), terminated with the word "XXXXX". All towns will consist of five letters. There will be no list of the towns given - you should assume the only towns of interest are those which occur in the roads list. No two towns share the same name, and no town will be called "XXXXX".

There will be no more than 250 towns in any list, though the list may contain several million roads. Note roads can be travelled in both directions, and may be given either way round. Due to the carelessness of mapmakers, town planning committees and the Pentium floating point bug, roads may be listed more than once.

Your output should consist of the number of towns, followed by the number of disjoint sets.

First Example

```
AAAAA BBBBB
CCCCC BBBBB
DDDDD EEEEE
XXXXX
```

5 2

Second Example

```
WURRP AMIER
IHFLC NARST
RCOQA DNATC
NARST GDRIF
XXXXX
```

7 3

British Informatics Olympiad Final

12–14 April 1996

Sponsored by Data Connection

Selection — Part One

You are to write a program which given a square (*n* by *n*) array and a number *x*, will try to find *n* occurrences of *x* so that there is one (and only one) chosen *x* in each row / column.

The input data will be a single number *n* indicating the size of the array, the number *x* you are to look for, followed by the array itself. The numbers in the array and *x* will be between 0 and 32, while *n* will be between 1 and 32.

The first *n* numbers given for the array represent its top row (going from (1,1) to (*n*,1)) and so on. Line-breaks should not be treated as indicating that the next number is for the next row, since the potentially large size of the arrays means arbitrary line breaks may be added.

If the problem is solvable you should print out a list of your chosen points. If there is no solution you should print 'Impossible'.

Example

```
5
10
4  10 8  10 15
10 1  2  13 2
9  16 22 10 12
4  10 4  2  10
7  10 10 1  18
```

(1,2)
(2,1)
(3,5)
(4,3)
(5,4)

British Informatics Olympiad Final

12–14 April 1996

Sponsored by Data Connection

Selection — Part Two

There are *n* jobs available in an organisation and coincidentally *n* candidates. Each candidate has a measure of suitability for each job. The 'optimal assignment' problem is to fit the candidates to the jobs in the best possible way. Your task is to write a program to solve this.

The input data will be a single number *n* indicating the number of candidates/jobs, followed by an *n* by *n* array. You can think of the columns representing the candidates, the rows the jobs, and the numbers a measure of suitability. Once again *n* will be between 1 and 32, and the elements in the array between 0 and 32.

Each candidate should be assigned to one (and only one) job. The value of assigning all these candidates is the sum of the suitabilities for each assignment. An assignment is optimal if it is not possible to assign the jobs and get a lower overall value.

You should output the overall value of your optimal assignment, followed by the assignment itself.

Example

```
5
4  10 8  10 15
10 1  2  13 2
9  16 22 10 12
4  10 4  2  10
7  10 10 1  18
```

```
22
(1,1)
(2,2)
(3,4)
(4,5)
(5,3)
```

[Note: In general you will not be able to give every candidate their optimal job. In the given example candidates 2, 3 and 5 all prefer job 2...]