Breadth-first tree traversal scheme:

Step 0. Insert the tree root into the queue.

While quee is not empty do:

Step 1. Remove the first node from the queue and process it.

Step 2. Insert all children of the removed node into the queue.

**1.** The tree depicted in the image is subjected to the BFS algorithm. At some point in time the contents of the queue is the following (we suppose that the queue head is on the left):

a) BGA

b) GA

c) AG

d) AEG

e) GEA

f) AE

**Solution**  b), f).

**2.** The tree depicted in the image is subjected to the BFS algorithm. At some point in time the contents of the queue is the following (we suppose that the queue head is on the left):



a) D

b) DF

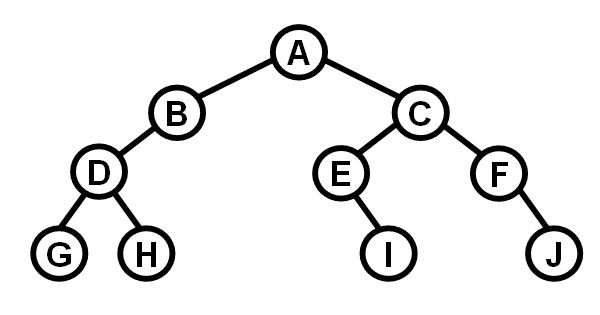
c) FB

d) BGE

e) BAC

f) ABCD

**Solution**  a). If random order of children is allowed then a), b), c), d).

**3.** Apply the BFS algorithm on the given tree and list the queue contents before the start of each Step 1 in the scheme above.

**Solution**

A

BC

CD

DE F

E FGH

FGHI

GHIJ

HIJ

IJ

J

**4.** We have to reconstruct the shape the tree which was traversed by the BFS algorithm.

There is the list of contents of the queue. Each element of the list (= queue contents) was printed immediately before the Step 1 in the BFS scheme above.

A

BC

C

DE

EFG

FG

GHI

HI

I

**Solution**

\_\_\_A\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

B \_\_\_\_\_\_\_\_\_\_C\_\_\_

\_\_\_D\_\_\_\_\_ E

F \_\_\_G\_\_\_

H I

**5.** Write down a general algorithm which will reconstruct the tree and which input is the sequence of queue contents defined in the previous problem. The tree is supposed to be regular.

**Solution**  First of all, the tree must be regular. There is no mechanism which will distinguish between left and right children in the queue if we do not explicitly store that information in the queue. Without the left/right information it is not possible in general to restore the tree correctly.

At the beginning, the first node in the queue is the tree root. Next, it is enough to identify the children of each node. The children of a given node are inserted into the queue immediately after the node had been removed from the queue. To identify the children of the node X we have to find a line which begins with X. Then we remove from the next line all symbols which appear in both these lines. What remains in the next line are the children of X.

**6.** Consider the previous problem and suppose the following:

-- The processed tree to be reconstructed is a BST.

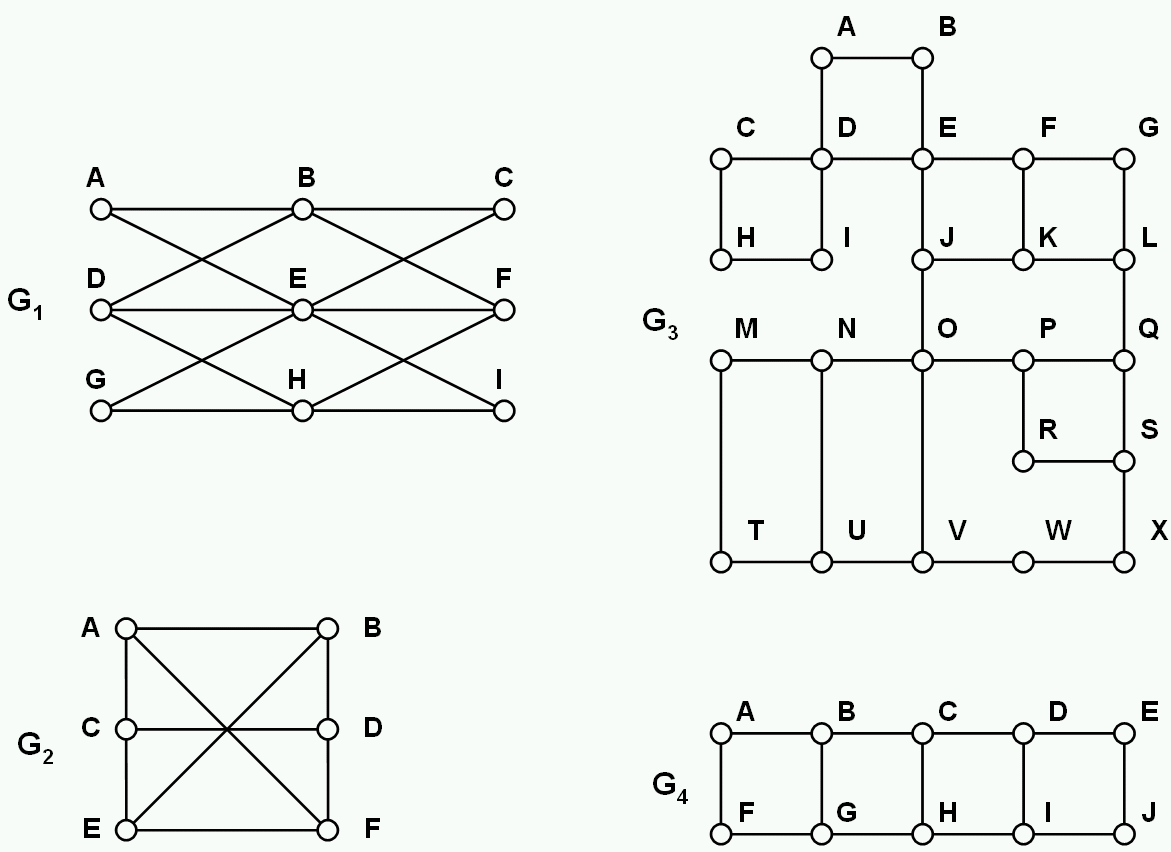
-- The queue stores the keys of the nodes.

Again, write down an algorithm which reconstructs this BST.

**Solution**  The algorithm differs form the previous one only in one part: After the identification of the node's chidren, it has to be decided which child is the left one and which child is right one. That depends on key values:

Smaller key value than the parent's key value indicates left child, bigger value indicates right child.

Graph examples:



In each of the following problems 7.-9., produce the list of visited nodes. The order of the nodes in the list should be the same as the order in which they were processed by the search/traversal algorithm.

**7.** Perform BFS on graphs G1, G2, G3, G4. Each time start in node A. In each node of each graph always process the neighbouring nodes in alphabetical order.

For example, in node B in G2 process the neighbours in order A, D, E.

**Solution**

G1: ABECDFGIH

G2: ABCFDE

G3: ABDECIFJHGKOLNPVMQRUWSTX

G4: ABFCGDHEIJ

**8.** Perform DFS (Depth-first search) on graphs G1, G2, G3, G4. Each time start in node B. In each node of each graph always process the neighbouring nodes in alphabetical order.

**Solution**

G1: BAECDHFGI

G2: BACDFE

G3: BADCHIEFGLKJONMTUVWXSQPR

G4: BAFGHCDEJI

**9.** Perform DFS (Depth-first search) on graphs G1, G2, G3, G4. Each time start in any node you like. The order in which the neighbours are processed is arbitrary. The additional condition is that each two neigbouring nodes in the list of visited nodes should also be neighbours in the graph. In other words, the search tree produced by DFS should contain no branchings and it should be one simple path.

**Solution**

G1: Impossible

G2: Start in A:

ABDCEF, ABDFEC, ABECDF, ABEFDC

ACDBEF, ACDFEB, ACEBDF, ACEFDB

AFDBCE, AFDCEB, AFEBDC, AFECDB

Start in any other node leads analogously to 12 more possibilities associated with that node.

G3: All possibilities must either start or end in the upper left square CDHI, namely either in node C or in node I

All possibilities starting in C must begin with sequence CHIDABE. All possible extensions to these starts are listed bellow

JONMTUVWXSRPQLGFK

JONMTUVWXSRPQLKFG

JKFGLQPONMTUVWXSR

JKFGLQPRSXWVONMTU

JKFGLQPRSXWVONUTM

JKFGLQPRSXWVUTMNO

JKFGLQSRPONMTUVWX

JKFGLQSXWVUTMNOPR

FGLKJONMTUVWXSQPR

FGLKJONMTUVWXSRPQ

FGLQPRSXWVUTMNOJK

Those are also extensions to the possibilities starting in I which begin with IHCDABE. Considering also the reversed sequences (ending in C or I) we have 44 possibilities in total.

G4: We investigate only start in A, B and C. Start in E, F, J is analogous to start in A. Start in D, G, I is analogous to start in B. Start in H is analogous to start in C.

AFGBCDEJIH

AFGBCHIDEJ

AFGBCHIJED

ABCDEJIHGF

AFGHIJEDCB

BAFGHCDEJI

BAFGHCDIJE

BAFGHIJEDC

BCDEJIHGFA

CBAFGHIJED

CDEJIHGFAB

Taking in account all possibilitiees and their combinations mentioned above we come to the number

5∙4 + 4∙4 + 2∙2 = 40 possibilities.