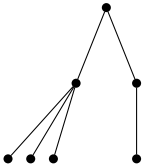
Lecture 33: Trees, queues and stacks

To search a graph G systematically, it helps to have a spanning tree T, together with an ordering of the vertices of T.

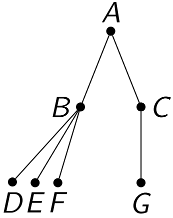
Breadth first ordering

The easiest ordering to understand is called *breadth first*, because it orders vertices "across" the tree in "levels."

Level 0 is a given "root" vertex.
Level 1 is the vertices one edge away from the root.
Level 2 are the vertices two edges away from the root,
. . . and so on.



Example.
A, B, C, D, E, F, G
is a breadth first ordering of



Queues

Breadth first ordering amounts to putting vertices in a *queue* - a list processed on a "first come, first served" or "first in, first out" basis.

- The root vertex is first in the queue (hence first out).
- Vertices adjacent to the head vertex v in the queue go to the tail of the queue (hence they come out after v), if they are not already in it.
- The head vertex v does not come out of the queue until all vertices adjacent to v have gone in.

Breadth first algorithm

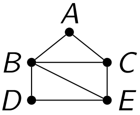
For any connected graph G, this algorithm not only orders the vertices of G in a queue Q, it also builds a spanning tree T of G by attaching each vertex v to a "predecessor" among the adjacent vertices of v already in T. An arbitrary vertex is chosen as the root V_0 of T.

1. Initially, $T =$ tree with just one vertex V_0, $Q =$ the queue containing only V_0.
2. While Q is nonempty
    2.1 Let V be the vertex at the head of Q
    2.2 If there is an edge $e = VW$ in G where W is not in T
        2.2.1 Add e and W to T
        2.2.2 Insert W in Q (at the tail).
    2.3 Else remove V from Q.

Remarks

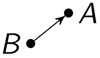
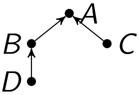
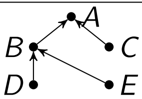
1. If the graph G is not connected, the algorithm gives a spanning tree of the connected component containing the root vertex A, the part of G containing all vertices connected to A.
2. Thus we can recognise whether G is connected by seeing whether all its vertices are included when the algorithm terminates.
3. Being able to recognise connectedness enables us, e.g., to recognise bridges.

Example.

If $G =$ 

with root vertex A.

Then Q and T grow as shown on the right:

Step	Q	T
1	A	
2	AB	
3	ABC	
4	BC	
5	BCD	
6	$BCDE$	
7	CDE	
8	DE	
9	E	

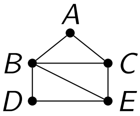
Depth first algorithm

This is the same except it has a *stack* S instead of a queue Q. S is "last in, first out," so we insert and remove vertices from the same end of S (called the top of the stack).

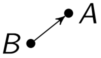
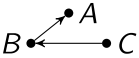
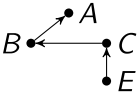
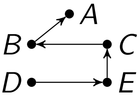
1. Initially, $T =$ tree with just one vertex V_0, $S =$ the stack containing only V_0.
2. While S is nonempty
   - 2.1 Let V be the vertex at the top of S
   - 2.2 If there is an edge $e = VW$ in G where W is not in T
     - 2.2.1 Add e and W to T
     - 2.2.2 Insert W in S (at the top).
   - 2.3 Else remove V from S.

Remark. The breadth first and depth first algorithms give two ways to construct a spanning tree of a connected graph.

Example. We use the same G, and take the top of S to be its right hand end.

$G =$ 

with root vertex A.

Step	S	T
1	A	
2	AB	
3	ABC	
4	$ABCE$	
4	$ABCED$	
6	$ABCE$	
7	ABC	
8	AB	
9	A	

Questions

33.1 The following list gives the state, at successive stages, of either a queue or a stack.

A
AB
ABC
BC
BCD
CD
D

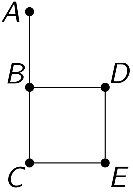
Which is it: a queue or a stack?

ANS: It is a queue. Notice that items are entering at the right hand end and leaving at the left hand end, which is characteristic of a queue.

In a stack, things would enter and leave at the same end.

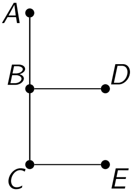
Questions

33.2 Construct a breadth first spanning tree for the graph



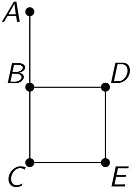
The queue follows the steps shown, producing the spanning tree shown far right.

A
AB
B
BC
BCD
CD
CDE
DE
E



Questions

33.3 Construct a depth first spanning tree for the graph



The stack trace is as shown, producing the spanning tree shown far right.

A
AB
ABC
ABCE
ABCED
ABCE
ABC
AB
A