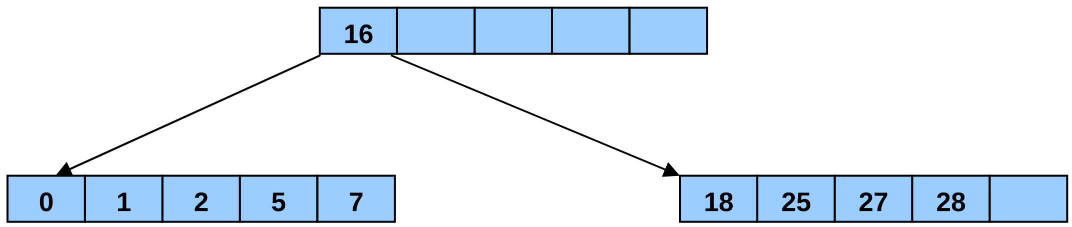
COP 3530 – Final Exam Review Problems – Summer 2005

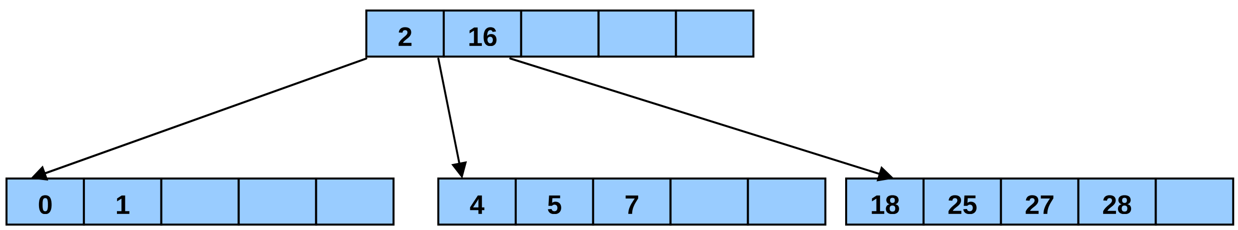
SOLUTION KEY

1. *Shown below is a B-Tree of order 6 at some point in time. Show the tree at each time instance after each "access" to the tree in the following "access sequence" occurs.*

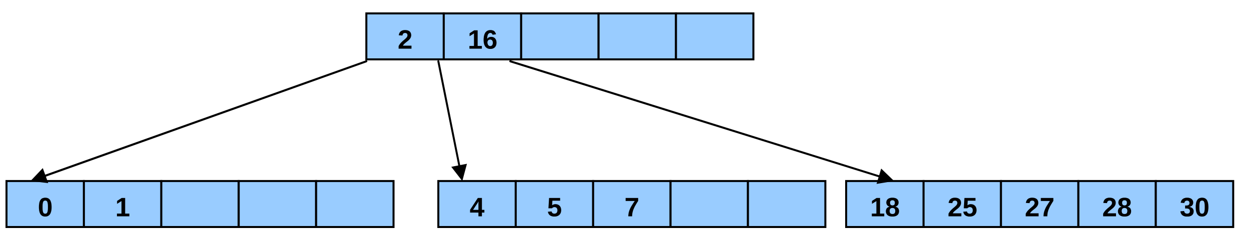
Access sequence: (insert 4), (insert 30), (insert 24), (delete 18), (delete 24)



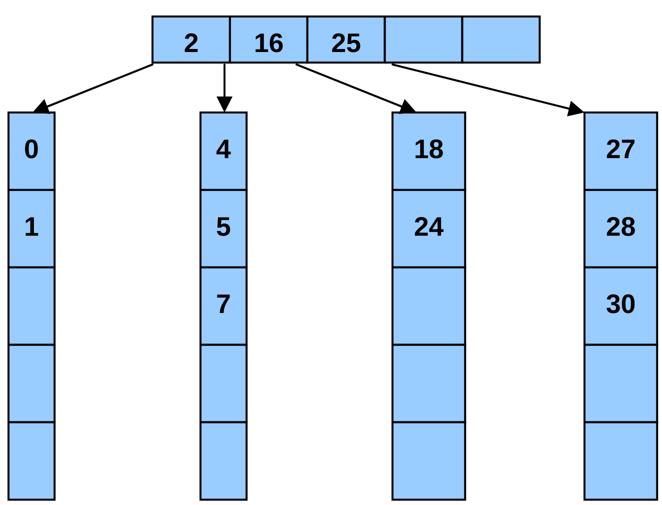
(a) Initial B-tree



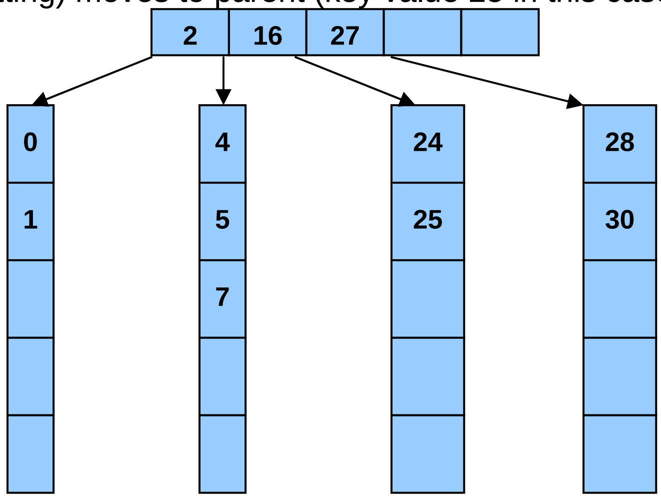
(b) B-tree after inserting 4 into full leaf node. Node splits and highest key in old leaf (after splitting equally) moves to parent (key value 2 in this case).



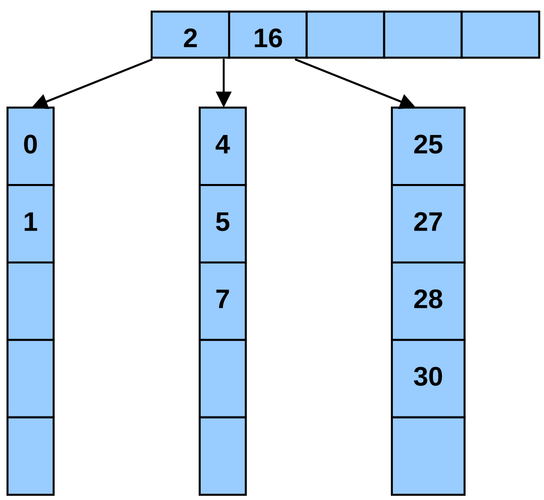
(c) B-tree after inserting 30 into leaf with room. No splitting occurs, but node is now full. (Nodes are vertical to fit all four nodes on the same level on the paper.)



(d) B-tree after inserting 24 into full leaf node. Node splits and highest key in old node (after splitting) moves to parent (key value 25 in this case).

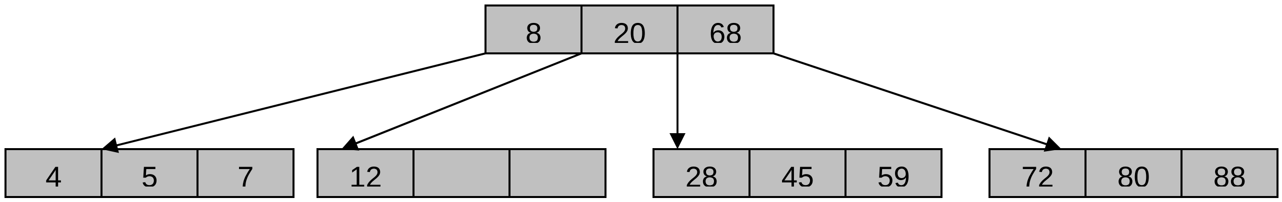


(e) B-tree after deletion of key value 18. Deletion causes an underflow. Since the sibling contains more than $\lceil m/2 \rceil - 1 = 2$, key values, then all keys from the underflowing leaf and the sibling are redistributed between them by moving the separator key (25 in this case) from the parent into the underflowing leaf and moving one key from the sibling to the parent to become the new separator key (27 in this case).

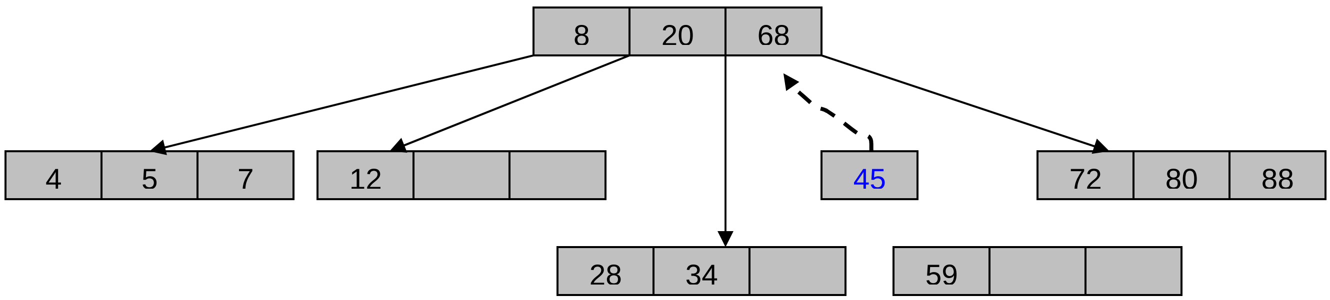


(f) B-tree after deletion of key value 24. Deletion causes underflow since number of key values in this node becomes less than $\lceil m/2 \rceil - 1 = 2$, however, we cannot “steal” a key value from the sibling in this case since that would cause the sibling to underflow. So the underflowing node, the separator key, and the sibling are merged.

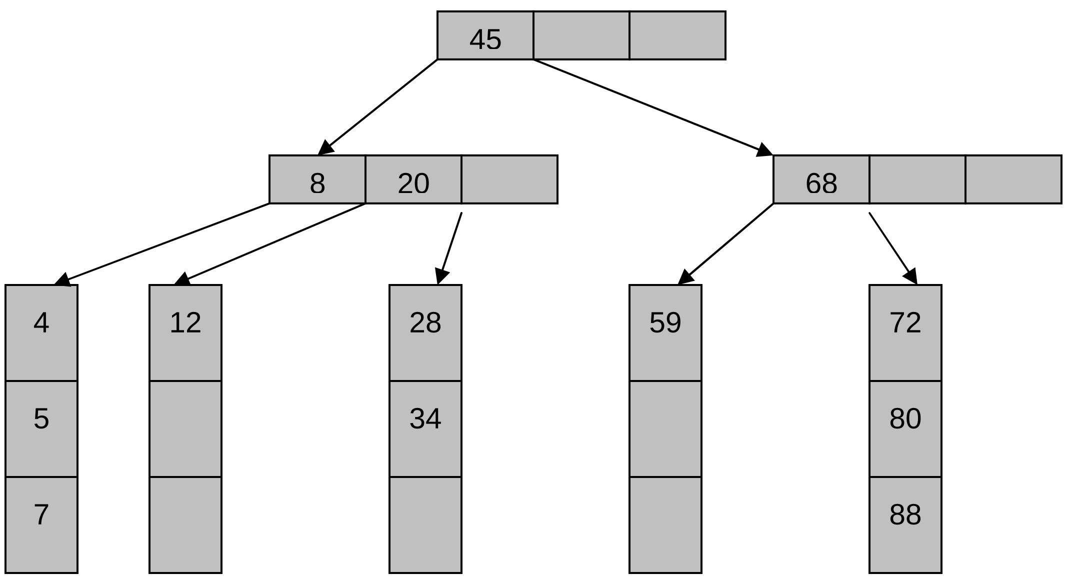
2. *For the 2-4 tree shown below, shown the change to the tree that occurs when the key value 34 is inserted into this tree. Explain what happened to the tree when this insert occurred.*



(a) Initial 2-4 tree.

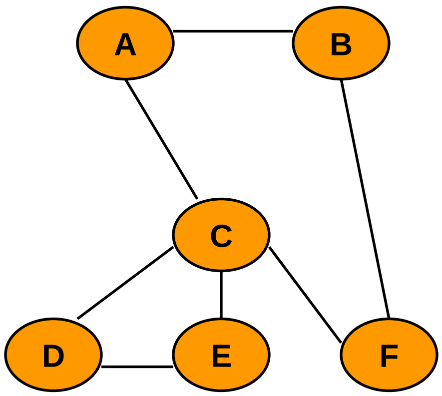


(b) Insertion of key value 34 causes node to overflow and require splitting. The split forces the third key value (which is 45 in this case) to move "up" the tree to the parent.

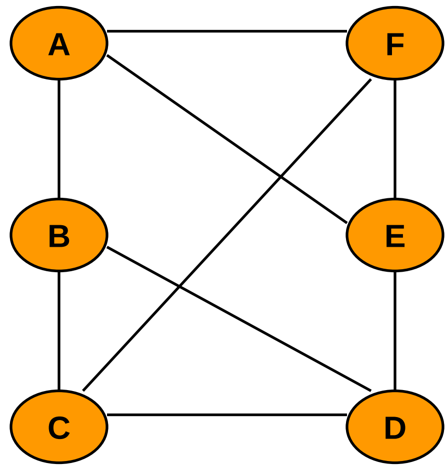


(c) Migration of key value 45 to the parent node causes a further splitting of the parent node since it too overflows. Since this new overflowing node is the current root, a new root node must be created and once again the third key from the overflowing node moves up into this new root node. In this case, it happens to be the key value 45 again that moves up. The rightmost child of the new root was the original highest key in the old root. The leftmost pointer in this node must point to the new node created from the original split at the previous level.

3. *For each graph shown below, (a) determine if the graph has an Euler circuit and (b) if it does produce one for the graph.*



(a)



(b)

Graph (a) contains an Euler circuit. Graph (b) does not since more than two vertices have an odd degree. For graph (a) one possible Euler circuit is:
A, C, D, E, C, F, B, A.

4. *For the graph shown below, produce the minimum spanning tree for the graph using Prim's algorithm. Assume that the starting node for the spanning tree is node D. Repeat the same problem using Kruskal's and Baruvka's algorithms.*

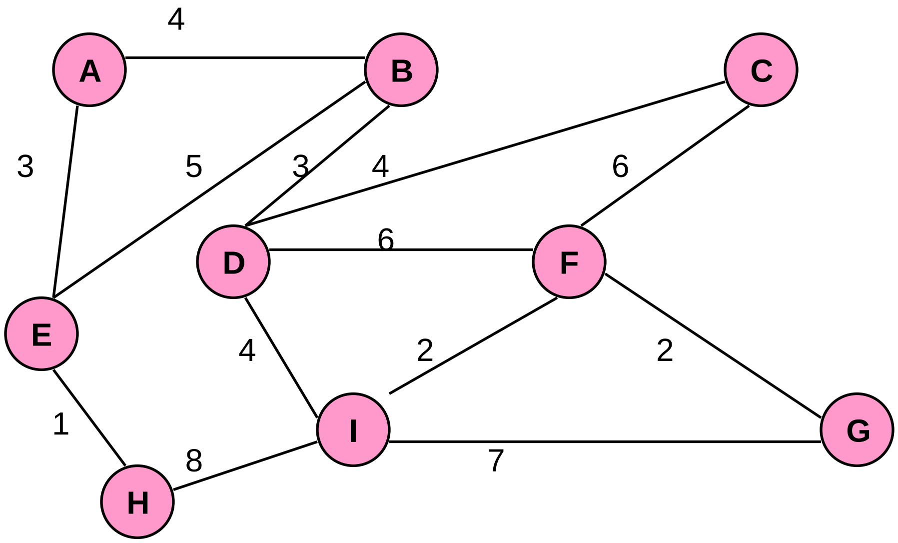


Illustration of Prim's Technique given below.

vertex	visited	minimum weight	vertex causing change to min weight
A	F	∞	
B	F	∞	
C	F	∞	
D	T	0	
E	F	∞	
F	F	∞	
G	F	∞	
H	F	∞	
I	F	∞	

Initial table

vertex	visited	minimum weight	vertex causing change to min weight
A	F	∞	
B	F	3	D
C	F	4	D
D	T	0	
E	F	∞	
F	F	6	D
G	F	∞	
H	F	∞	
I	F	4	D

Table after iteration 1 – consideration of node D

vertex	visited	minimum weight	vertex causing change to min weight
A	F	4	B
B	T	3	D
C	F	4	D
D	T	0	
E	F	5	B
F	F	6	D
G	F	∞	
H	F	∞	
I	F	4	D

Table after iteration 2 – consideration of node B

vertex	visited	minimum weight	vertex causing change to min weight
A	T	4	B
B	T	3	D
C	F	4	D
D	T	0	
E	F	3	A
F	F	6	D
G	F	∞	
H	F	∞	
I	F	4	D

Table after iteration 3 – consideration of node A

vertex	visited	minimum weight	vertex causing change to min weight
A	T	4	B
B	T	3	D
C	F	4	D
D	T	0	
E	T	3	A
F	F	6	D
G	F	∞	
H	F	1	E
I	F	4	D

Table after iteration 4 – consideration of node E

vertex	visited	minimum weight	vertex causing change to min weight
A	T	4	B
B	T	3	D
C	F	4	D
D	T	0	
E	T	3	A
F	F	6	D
G	F	∞	
H	T	1	E
I	F	4	D

Table after iteration 5 – consideration of node H

vertex	visited	minimum weight	vertex causing change to min weight
A	T	4	B
B	T	3	D
C	T	4	D
D	T	0	
E	T	3	A
F	F	6	D
G	F	∞	
H	T	1	E
I	F	4	D

Table after iteration 6 – consideration of node C

vertex	visited	minimum weight	vertex causing change to min weight
A	T	4	B
B	T	3	D
C	T	4	D
D	T	0	
E	T	3	A
F	F	2	I
G	F	7	I
H	T	1	E
I	T	4	D

Table after iteration 7 – consideration of node I

vertex	visited	minimum weight	vertex causing change to min weight
A	T	4	B
B	T	3	D
C	T	4	D
D	T	0	
E	T	3	A
F	T	2	I
G	F	2	F
H	T	1	E
I	T	4	D

Table after iteration 8 – consideration of node F

vertex	visited	minimum weight	vertex causing change to min weight
A	T	4	B
B	T	3	D
C	T	4	D
D	T	0	
E	T	3	A
F	T	2	I
G	T	2	F
H	T	1	E
I	T	4	D

Final table – after consideration of node G

Minimum spanning tree consists of the following:

edge(A,B), cost 4
edge(B,D), cost 3
edge(C,D), cost 4
edge(E,A), cost 3
edge(F,I), cost 2
edge(G,F), cost 2
edge(H,E), cost 1
edge(I,D), cost 4

The minimum spanning tree is shown below:

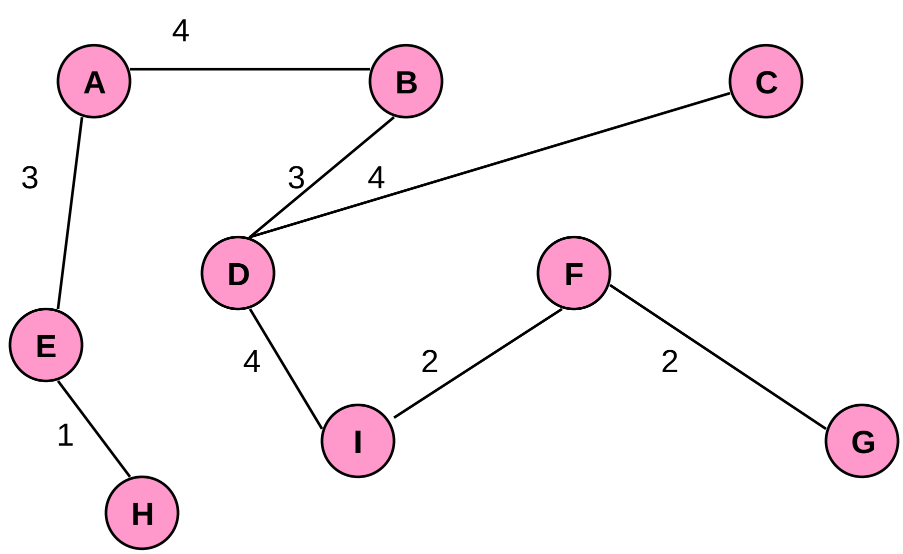
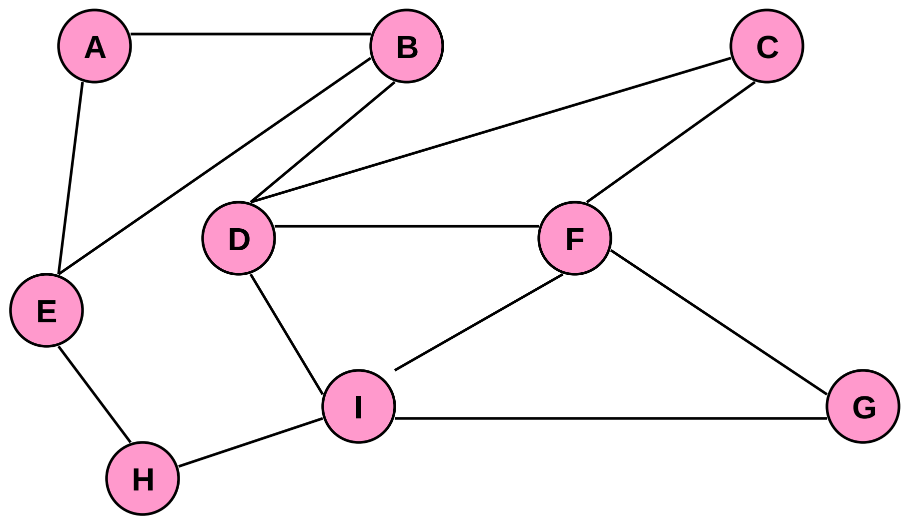
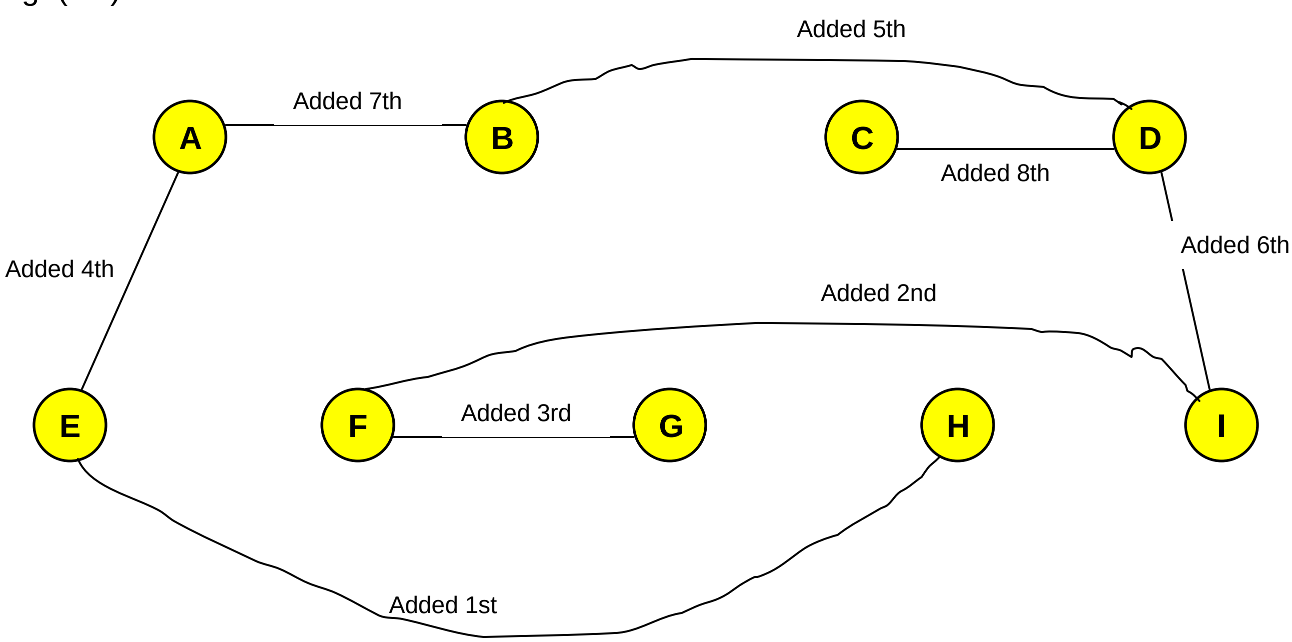


Illustration of Kruskal’s Technique:

Edges ordered as:
edge(E-H) = 1
edge(F-I) = 2
edge(F-G) = 2
edge(A-E) = 3
edge(B-D) = 3
edge(D-I) = 4
edge(A-B) = 4
edge(D-C) = 4
edge(B-E) = 5
edge(D-F) = 6
edge(C-F) = 6
edge(G-I) = 7
edge(H-I) = 8





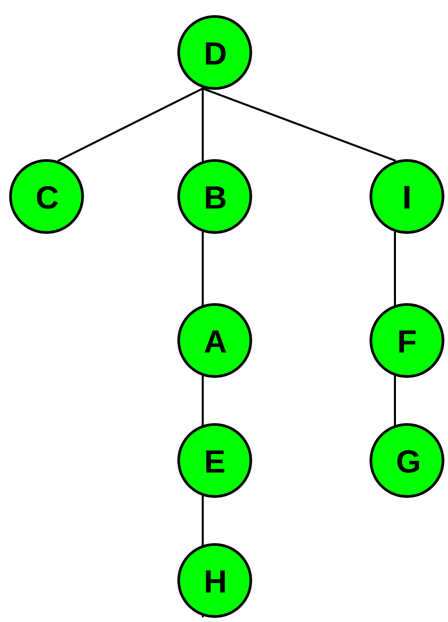


Illustration of Baruvka's Algorithm:

A: edges a-b = 4, a-e = 3
select edge a-e, since e not in tree
B: edges b-a = 4, b-e = 5, b-d = 3
select edge b-d, since d not in tree
C: edges c-d = 4, c-f = 6
select c-d since d is in tree
D: edges d-b = 3, d-c = 4, d-f = 6, d-i = 4
can't select d-b since b is in tree
select d-i since i not in tree
E: edges e-a = 3, e-b = 5, e-h = 1
select e-h since h not in tree
F: edges f-c = 2, f-d = 6, f-i = 2, f-g = 2
select f-i since i not in tree
G: edges g-f = 2, g-i = 7
select g-f since f not in tree
H: already in tree
I: already in tree