ECS 60 **Preparation for Midterm #2 Key**

Topics: ADTs, heaps, and graphs.

1. (30 points) Programs Problem. This question will attempt to determine whether you wrote the code for programming assignments 3 and 4. There is no need for you to see sample questions if you actually participated in writing the code.

2. (30 points) Timetest3 Problem. This question will address what you learned writing your paper for timetest3.cpp. Questions could be of the form where you are given a description of a new File5.dat, and you are asked for the expected result for given ADTs compared to one of the original data files. Questions could also involve determining whether you clearly understand the source(s) of the anomalous behavior of the different ADTs.

3. (40 points) ADT Problem. Web search engines permit keyword searches. Assume that there are 100,000 keywords, 10 million document addresses, and that each document address is stored in exactly one place on a disk. There may be many pointers to that location on the disk, but there are no pointers from it. A document may have more than one keyword associated with it. The search engine companies must keep their information appearing current by ensuring that newly inserted documents are the first reported from a search.

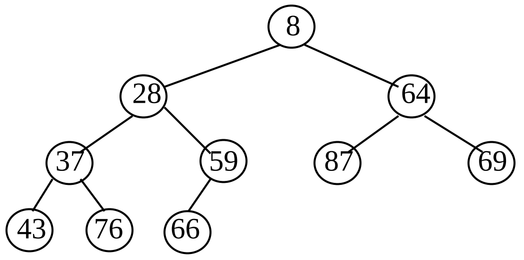
   Describe the data structure(s) that you would use. Explain how you would insert and delete a document, including how many disk accesses would be involved. We do not expect you to come up with a perfect solution to this problem in 15 minutes.

   One solution would be to use a hash table for keyword search. Since there are only 100,00 keywords this table could be kept in RAM at all times. Associated with each keyword is a linked list of groups of document address pointers. The groups (with their next pointer) would be the size of a disk block.

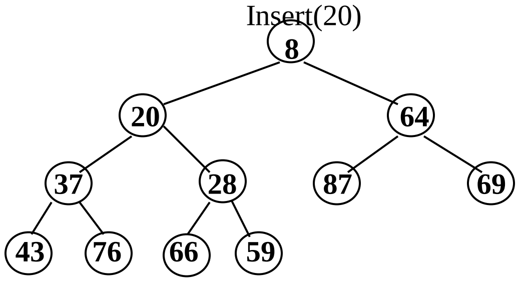
   Insertion would involve noting where the address is stored, and then inserting a pointer to that address for every keyword associated with the document. The insertion would be at the front of the linked list so it would usually involve one disk access to read the first disk block and one disk access to write a block. Occasionally a new disk block would have to be added to the front of the linked list. Since there would be no need to write the old first block, the number of disk writes remains only one. Thus, there would be a pair of disk accesses for every keyword insertion associated with a document.

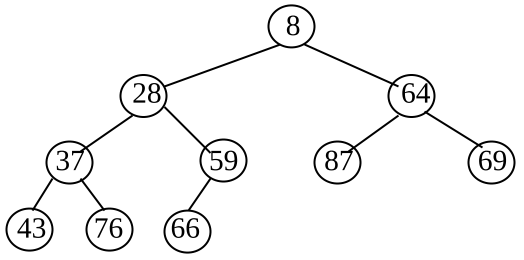
   Searching would involve hashing the given keyword to find the front of the linked list of blocks, and then just reading the addresses of the documents from the front of the linked list. Note that since the most recent are at the front, they will be supplied first. This involves one disk access for each group of pointers, and one disk access for each address. (Certainly a less than ideal situation.)

4. (15 points) Heap Problem. There will be one problem dealing insert, deleteMin, or BuildHeap operations with a priority queue. For each of the following binary heaps, show the state of heap after operation specified.

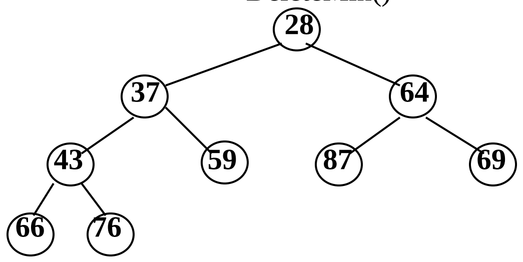


Insert(20)





DeleteMin()



5. Graph Problems There will be two graph problem using the following formats.

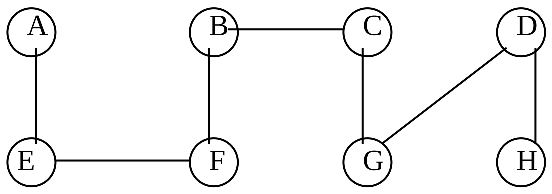
a) Minimum Spanning Tree. Fill in the following tables using Kruskal's algorithm to find the minimum spanning tree of the graph. Use union-by-size, but not path compression. For a union of sets of the same size, the new root should be the old root with the lower array index. Stop the process when you have a minimum spanning tree. Weights are to the left of internal edges.

Edge	Weight	Action
(A, B)	37	**reject**
(A, E)	10	**accept**
(B, C)	41	**accept**
(B, E)	22	**reject**
(B, F)	3	**accept**
(C, D)	34	**reject**
(C, F)	45	
(C, G)	7	**accept**
(D, G)	25	**accept**
(D, H)	12	**accept**
(E, F)	17	**accept**
(F, G)	44	
(G,H)	29	**reject**

Find-Union Array

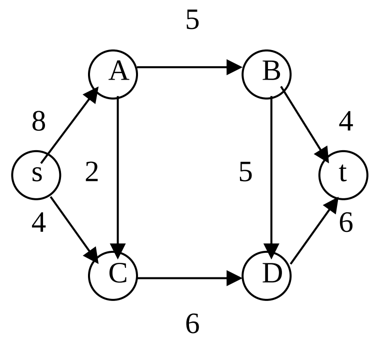
A	B	C	D	E	F	G	H
0	1	2	3	4	5	6	7
-8	**0**	**0**	**2**	**0**	**1**	**2**	**3**

Based on your tables, draw the minimum spanning tree of the graph. (Don't bother with weights.)

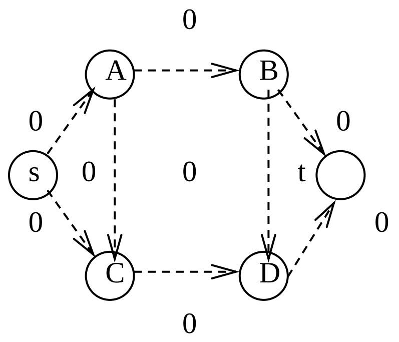


b) Network Flow. Choosing the augmenting path that allows the largest increase in flow, find the maximum network flow for the following graph, G. Internal flows are to the left of their respective edges. Your score will be based on your completion of the augmented path line, G_f and G_r, including their initial states and back-flow edges where appropriate.

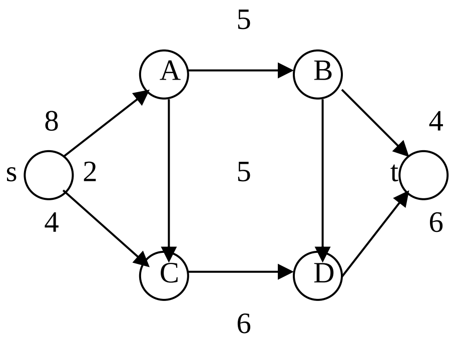
G



G_f

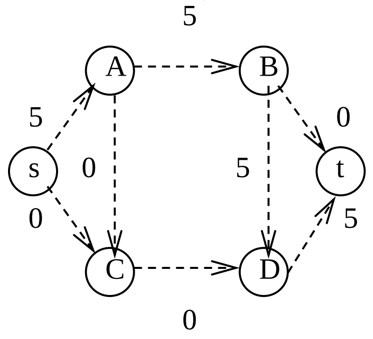


G_r

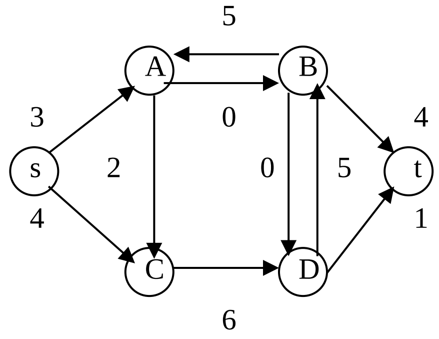


1st augmented Path: s, A, B, D, t (5)

G_f

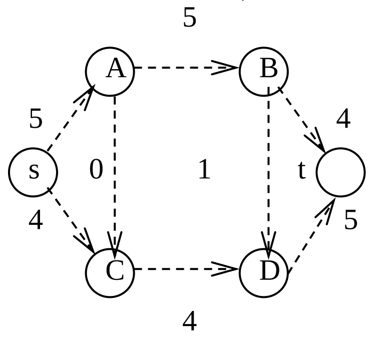


G_r

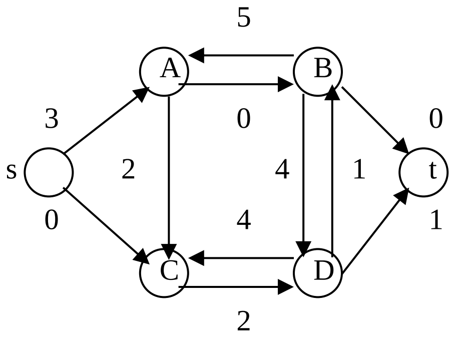


2nd augmented Path: s, C, D, B, t (4)

G_f

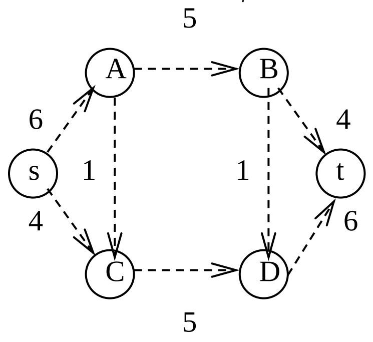


G_r

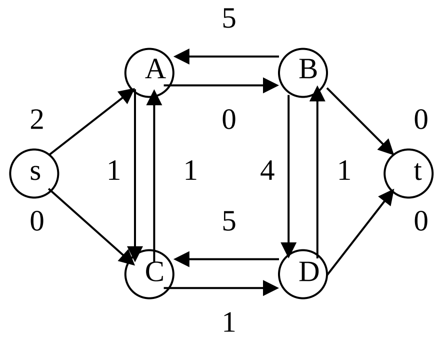


3rd augmented Path: s, A, C, D, t (1)

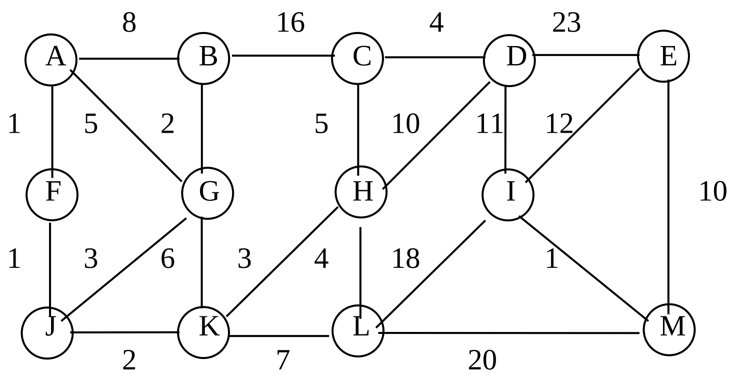
G_f



G_r

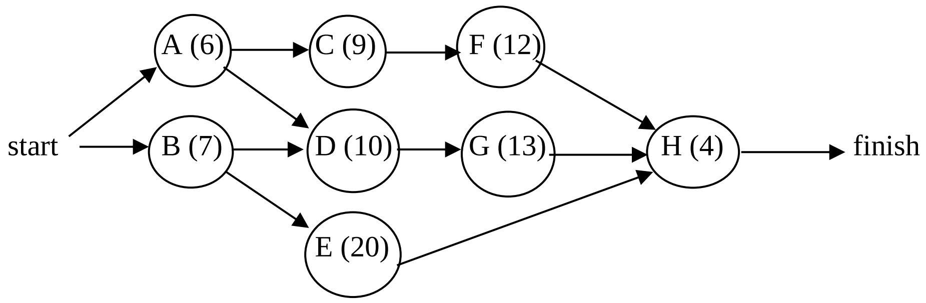


c) Shortest Path. Fill in the table using Dijkstra's algorithm to determine the shortest paths from vertex B The weight of each interior edge is to the left of the edge.

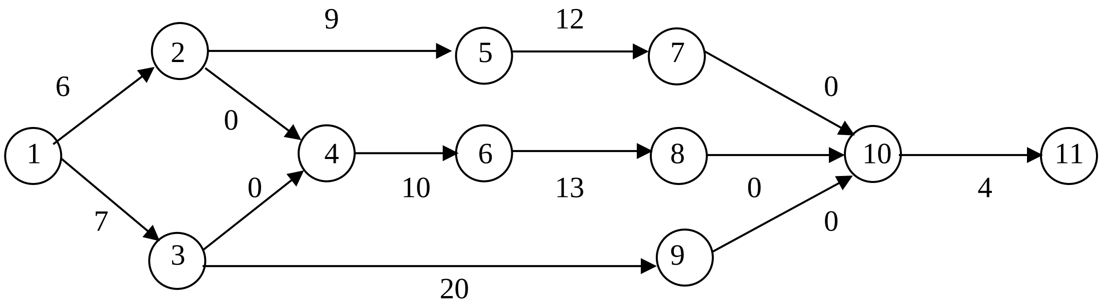


Vertex	known	d_v	p_v
A	T	7	G
B	T	0	--
C	T	15	H
D	T	19	C
E	T	41	M
F	T	6	J
G	T	2	B
H	T	10	K
I	T	30	D
J	T	5	G
K	T	7	J
L	T	14	K
M	T	31	I

d) Event Node Graph. For the following Activity-node graph:



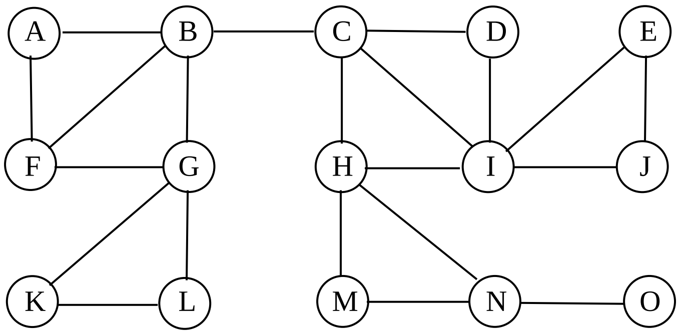
Draw the corresponding Event-node graph using numbered nodes (start numbering at left please).



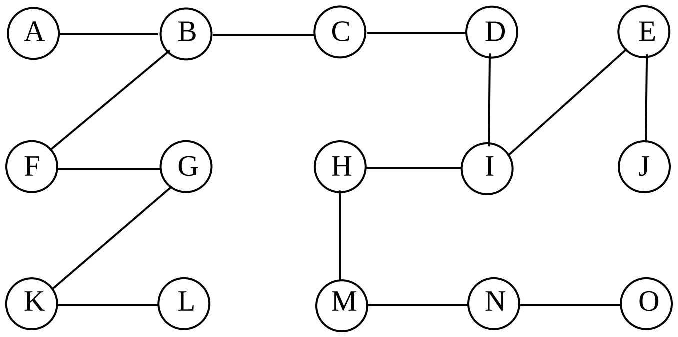
Based on your graph fill in the values for the following table. There may be more columns than needed.

Nodes:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Earliest completion time	**0**	**6**	**7**	**7**	**15**	**17**	**27**	**30**	**27**	**30**	**34**				
p_v	**1**	**1**	**1**	**3**	**2**	**4**	**5**	**6**	**3**	**8**	**10**				

e) Articulation Points. Using DFS, starting at A, searching in alphabetical order whenever multiple vertices may be processed, give each vertex a number followed by its low. Then list the articulation points



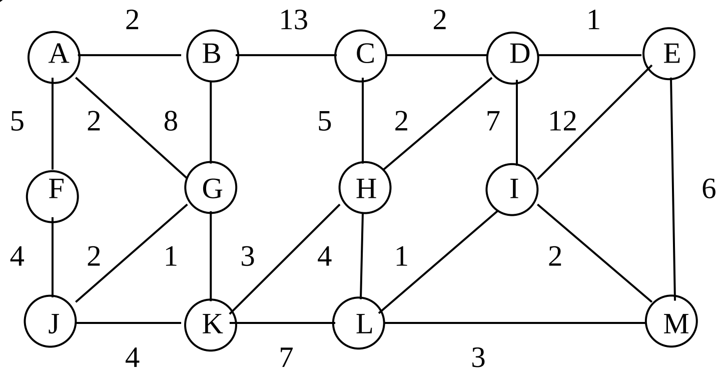
(5 points) Show DFS tree(s) below. You need not show the back edges, but may if you wish.



2 points for each completely correct column

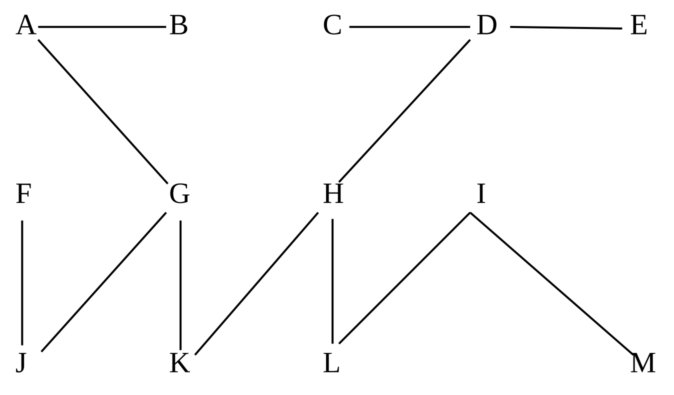
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Vertex number (5 points)	**1**	**2**	**3**	**4**	**6**	**12**	**13**	**8**	**5**	**7**	**14**	**15**	**9**	**10**	**11**
Low (5 points)	**1**	**1**	**3**	**3**	**5**	**1**	**2**	**3**	**3**	**5**	**13**	**13**	**8**	**8**	**11**
Articulation Point Y or N (5 points)	**N**	**Y**	**Y**	**N**	**N**	**N**	**Y**	**Y**	**Y**	**N**	**N**	**N**	**N**	**Y**	**N**

f) Minimum Spanning Tree. Fill in the table using Prim's algorithm to determine the shortest paths from vertex E. The weight of each interior edge is to the left of the edge. **(1 point for each d_v, and p_v, 4 points for all known as T.)**



Vertex	**known**	**d_v**	**p_v**
A	**T**	**2**	**G**
B	**T**	**2**	**A**
C	**T**	**2**	**D**
D	**T**	**1**	**E**
E	**T**	**0**	**-**
F	**T**	**4**	**J**
G	**T**	**1**	**K**
H	**T**	**2**	**D**
I	**T**	**1**	**L**
J	**T**	**2**	**G**
K	**T**	**3**	**H**
L	**T**	**4**	**H**
M	**T**	**2**	**I**

Based on your tables, draw the minimum spanning tree of the graph. (Don't bother with weights.)


A
B
C
D
E
F
G
H
I
J
K
L
M