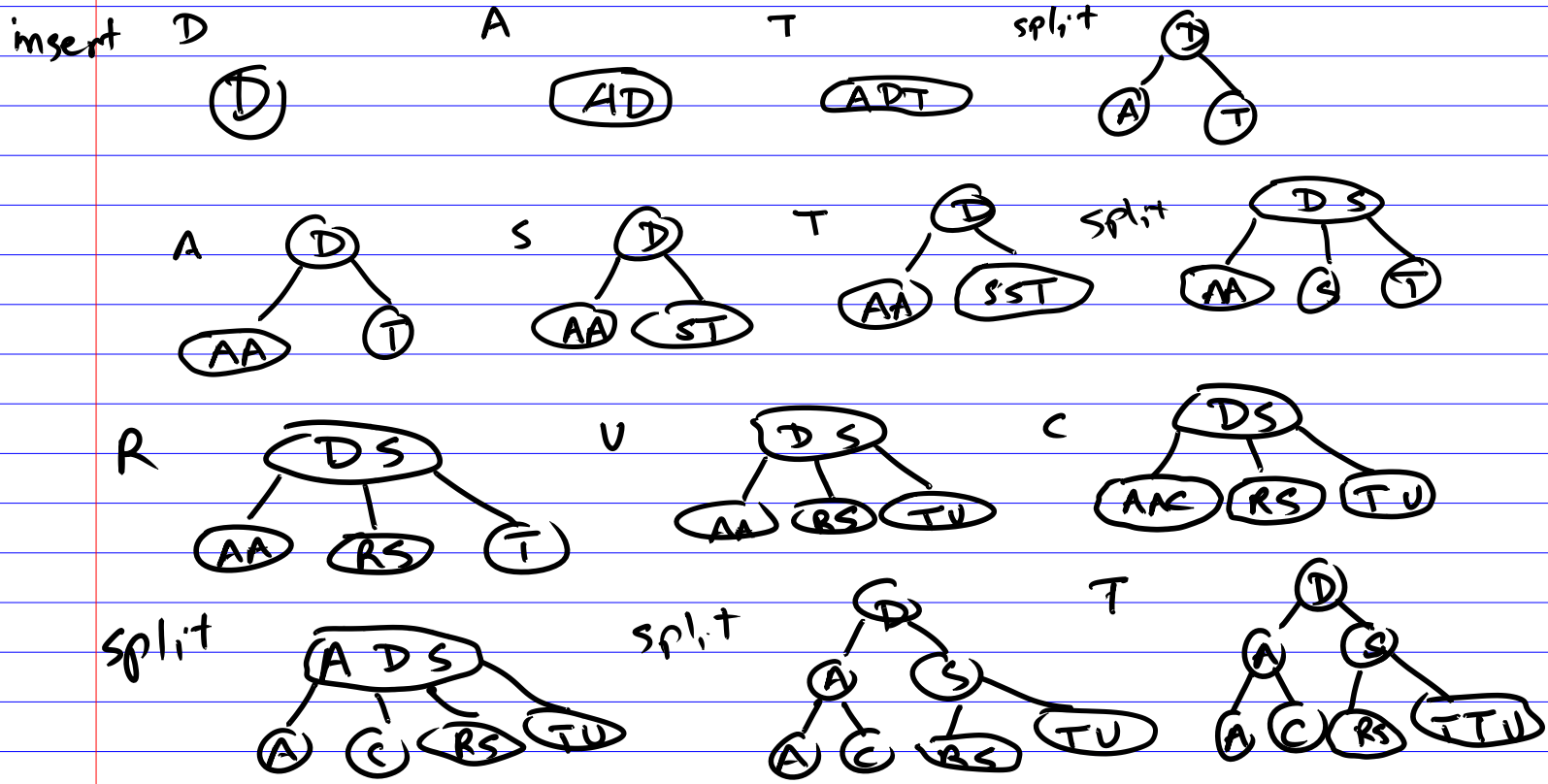
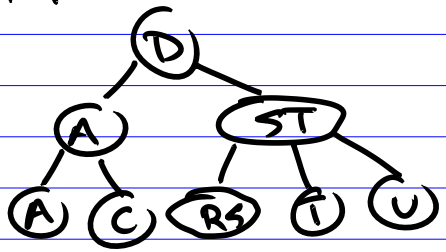
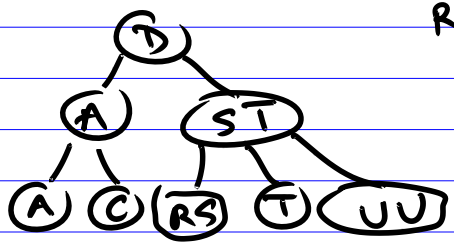
Q1a) Insert into a 2-3 tree.



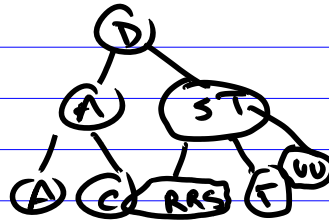
split



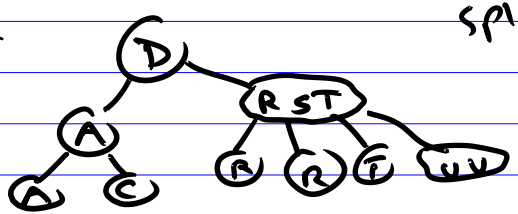
U



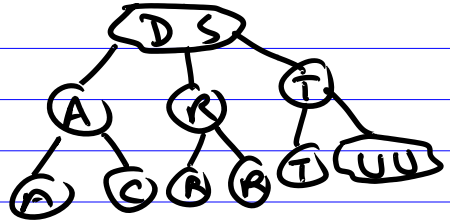
R



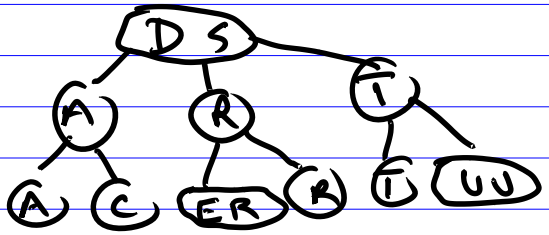
split



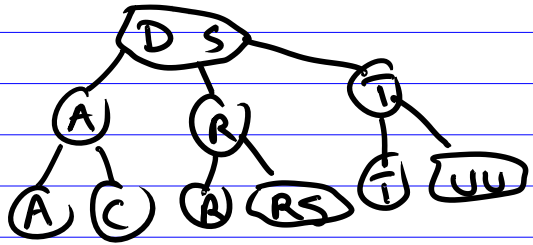
split



E



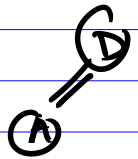
S



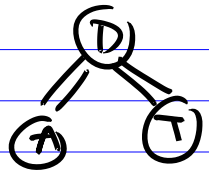
Q 1b)

Insert D

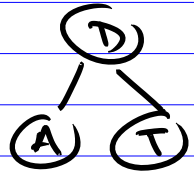
A



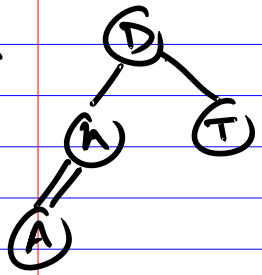
T



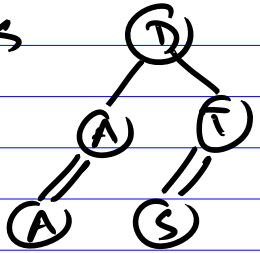
split



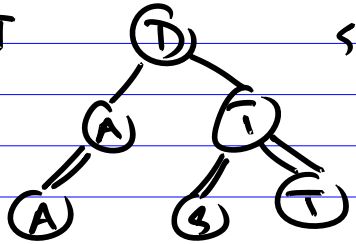
A



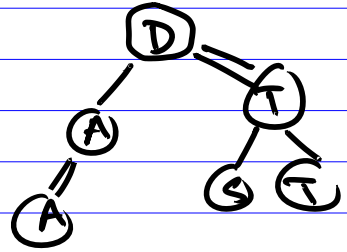
S



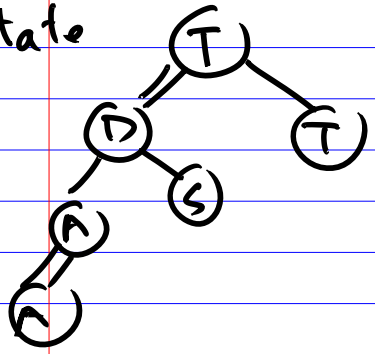
T



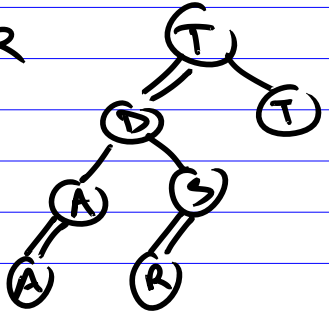
split



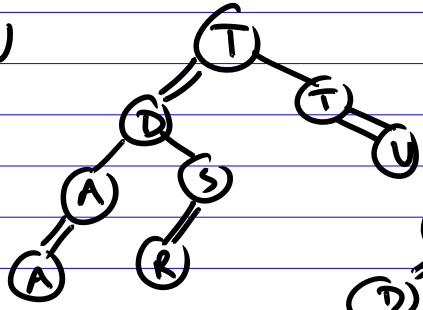
rotate



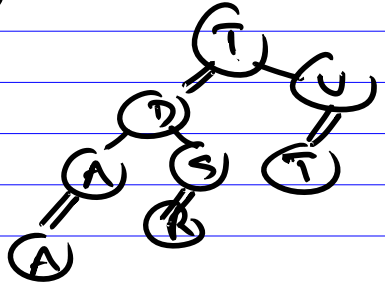
R



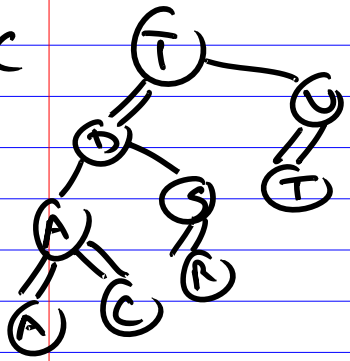
U



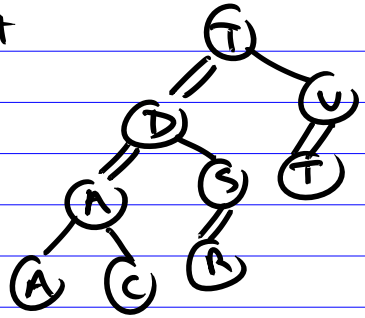
rotate



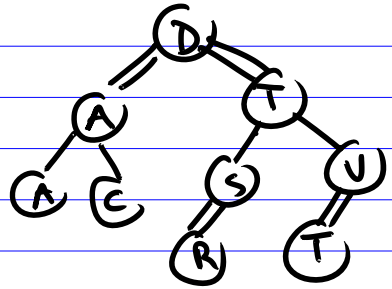
C



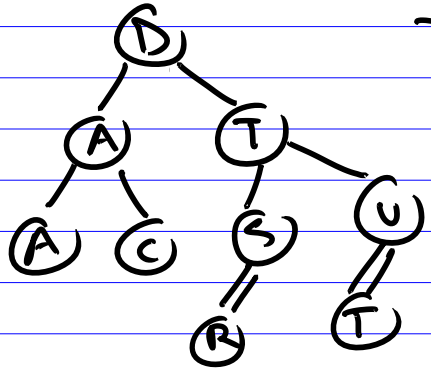
split



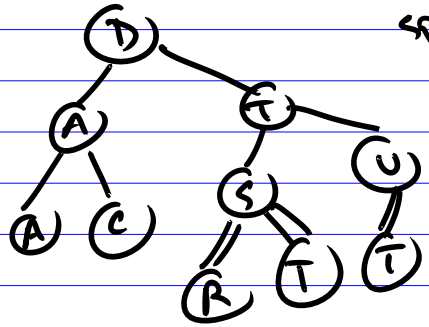
rotate



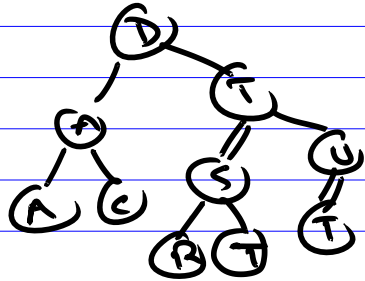
split



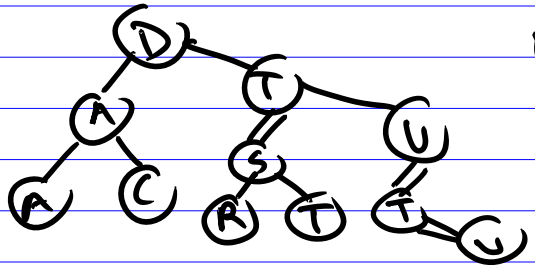
T



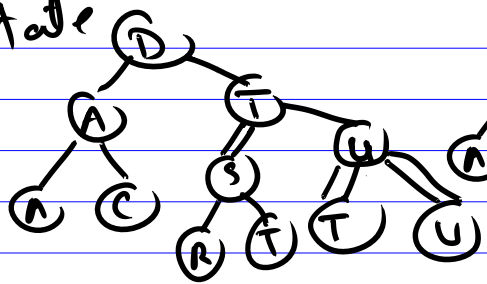
split



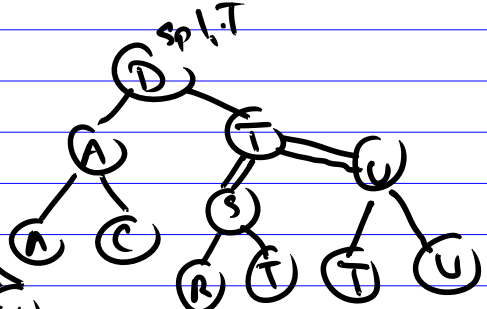
U



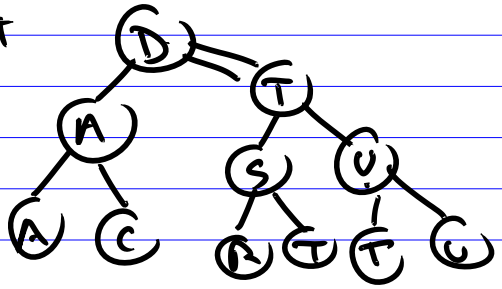
rotate



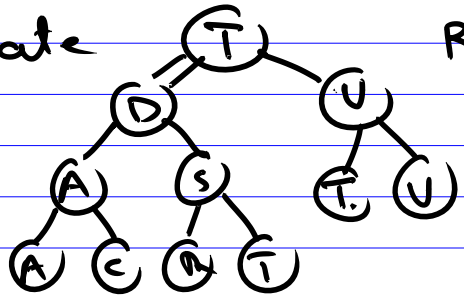
split



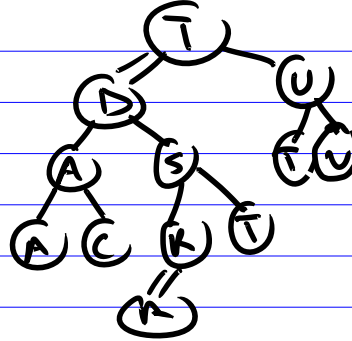
split



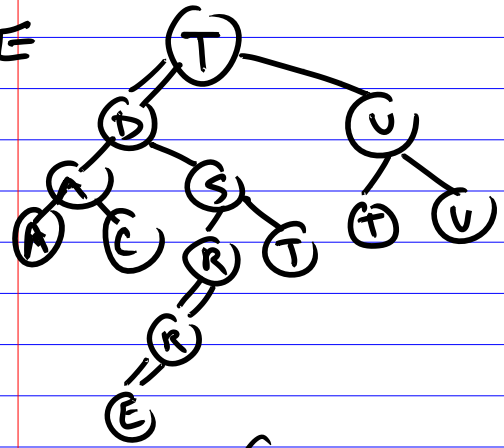
rotate



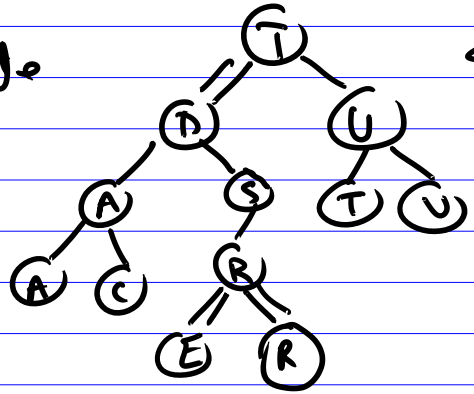
R



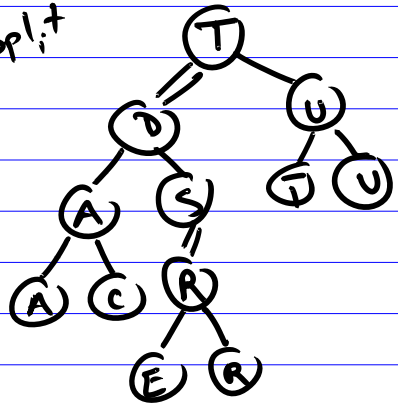
E



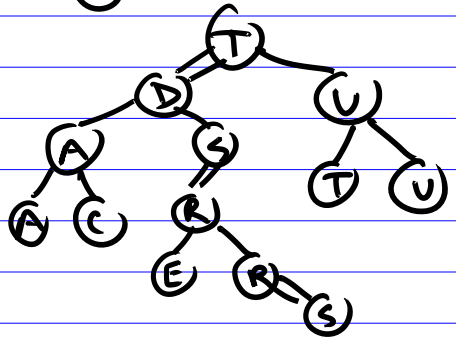
rotate



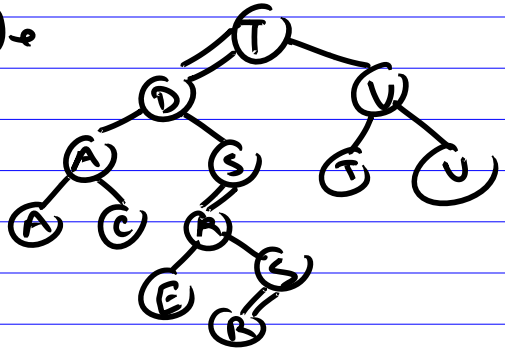
split



S

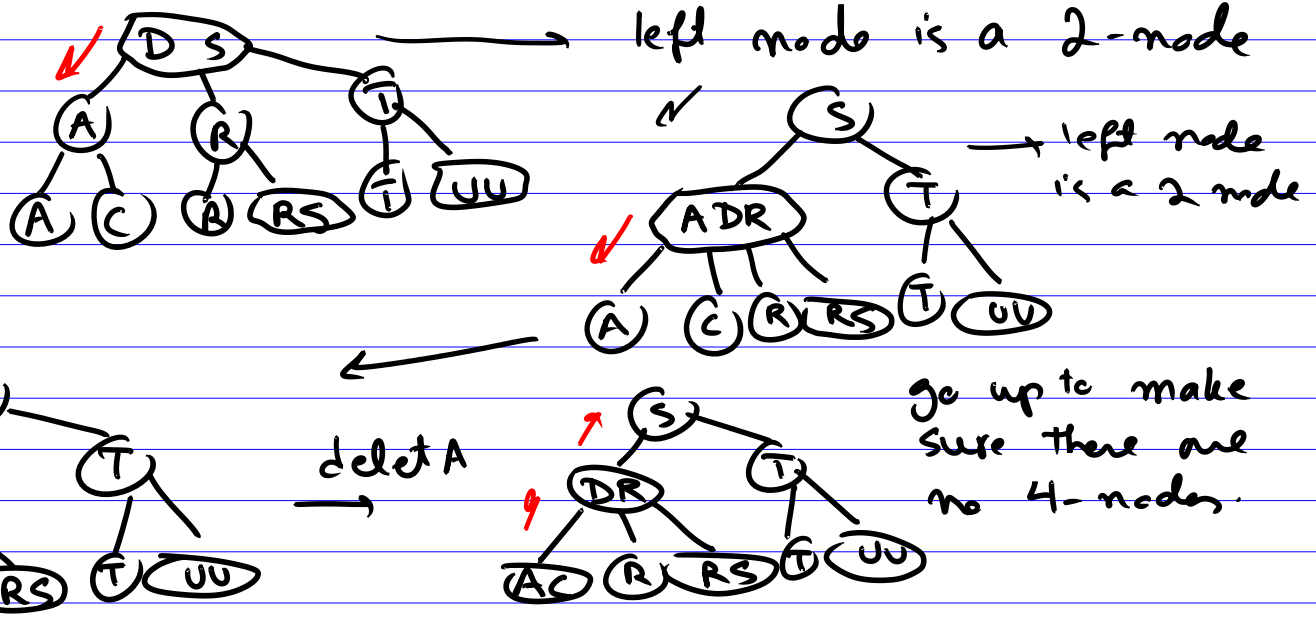


rotate

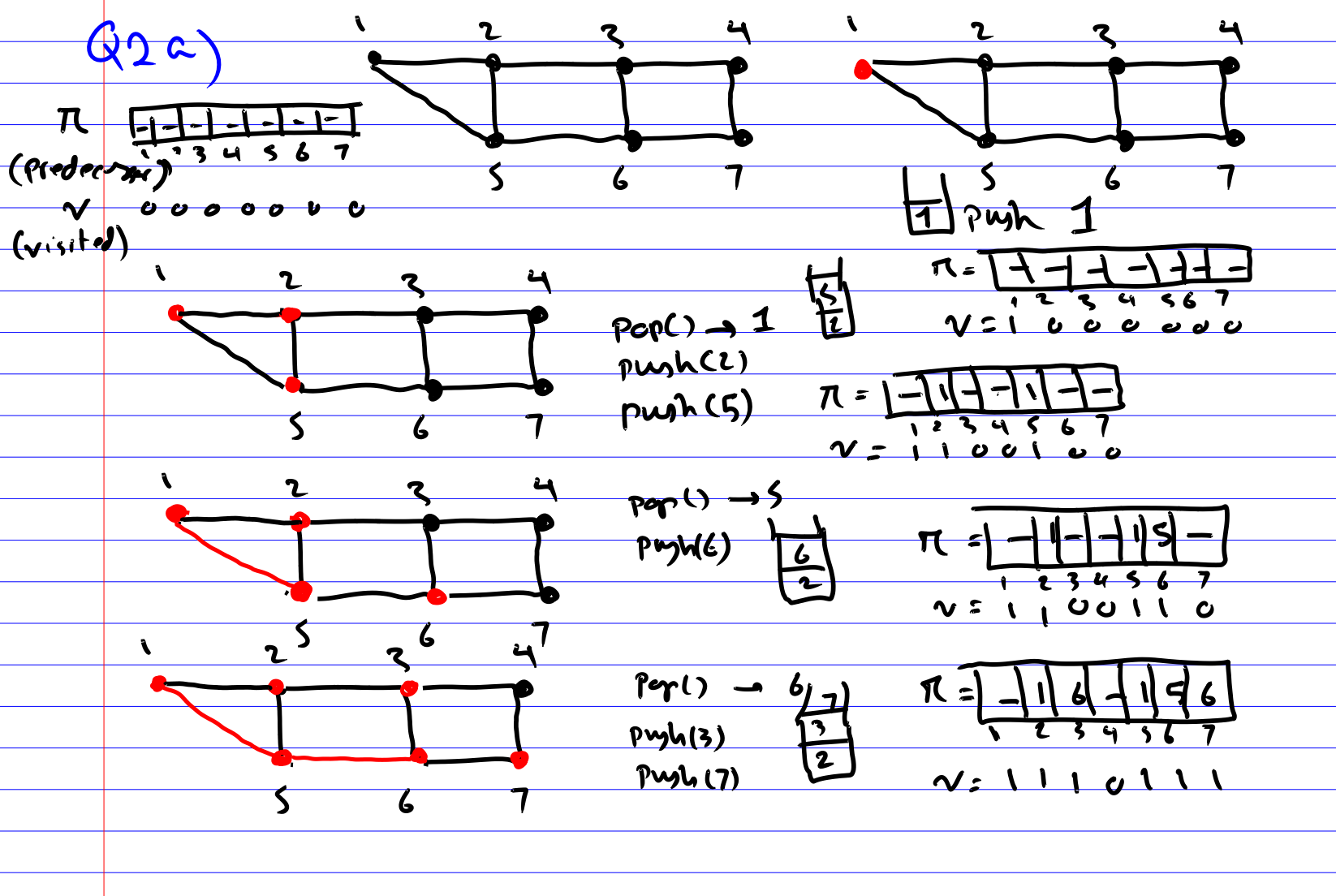


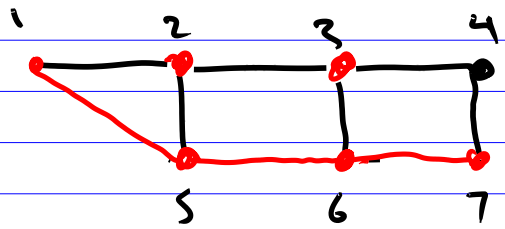
Note: When the key we are inserting is equal to the key in the node, we could go either left or right. This decision will change the structure of the tree from that point forward.

Q1c)



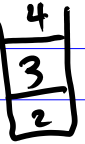
Q2 a)


π
(Predecessor)
1 2 3 4 5 6 7
v
(visited)
0 0 0 0 0 0 0
1 2 3 4
5 6 7
push 1
pop() → 1
push(2)
push(5)
pop() → 5
push(6)
pop() → 6
push(3)
push(7)
v = 1 0 0 0 0 0 0
v = 1 1 0 0 1 0 0
v = 1 1 0 0 1 1 0
v = 1 1 1 0 1 1 1



Pop() → 7

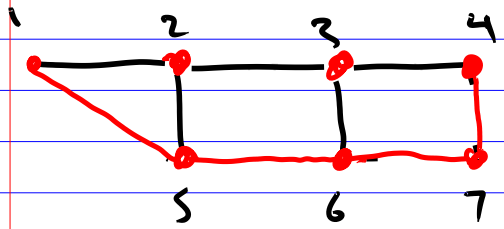
Push(4)



π =

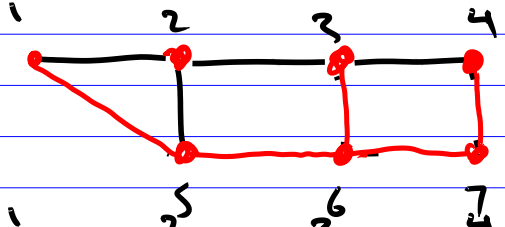
–	1	6	7	1	5	6
1	2	3	4	5	6	7

v = 1 1 1 1 1 1 1



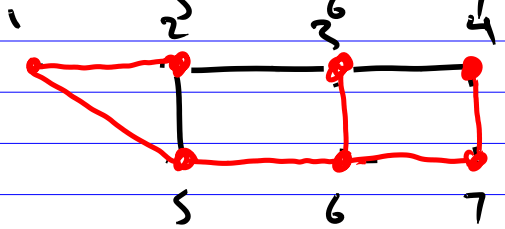
Pop() → 4

π, v unchanged



Pop() → 3

π, v unchanged



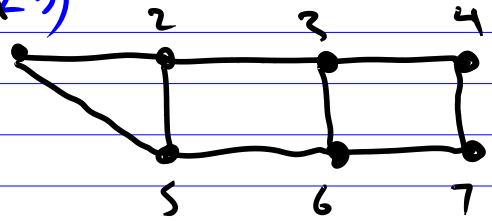
Pop() → 2

π, v unchanged

stack empty, v is all ones

Done.

Q2b)

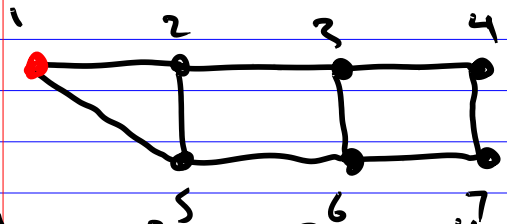


π (Predecessor)

1	2	3	4	5	6	7
-	-	-	-	-	-	-

V (visited)

1	2	3	4	5	6	7
0	0	0	0	0	0	0



addQ(1)

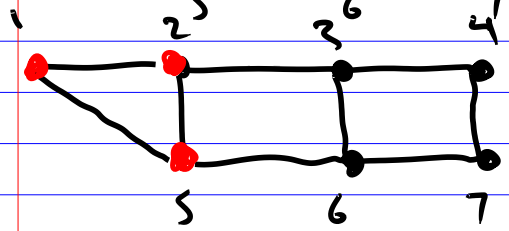
→ 1 →

π =

1	2	3	4	5	6	7
-	-	-	-	-	-	-

V =

1	2	3	4	5	6	7
1	0	0	0	0	0	0



deleteQ() → 1

addQ(2)

addQ(5)

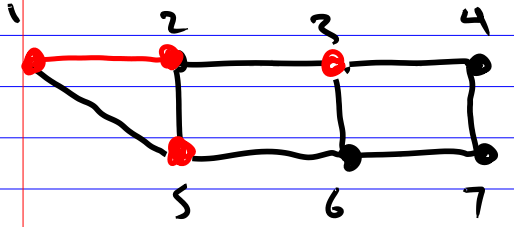
→ 5 | 2 →

π =

1	2	3	4	5	6	7
-	1	-	-	1	-	-

V =

1	2	3	4	5	6	7
1	1	0	0	1	0	0



deleteQ() → 2

addQ(3)

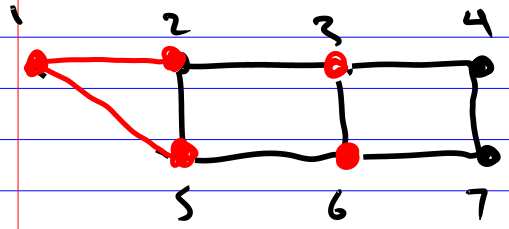
→ 3 | 5 →

π =

1	2	3	4	5	6	7
-	1	2	-	1	-	-

V =

1	2	3	4	5	6	7
1	1	1	0	1	0	0



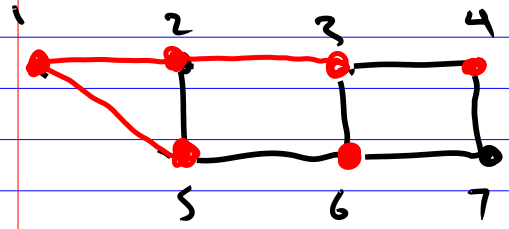
deleteQ() → 5

addQ(6)

→ 6 | 3 →

π = | – | 1 | 2 | – | 1 | 5 | – |
1 2 3 4 5 6 7

V = 1 1 1 0 1 1 0



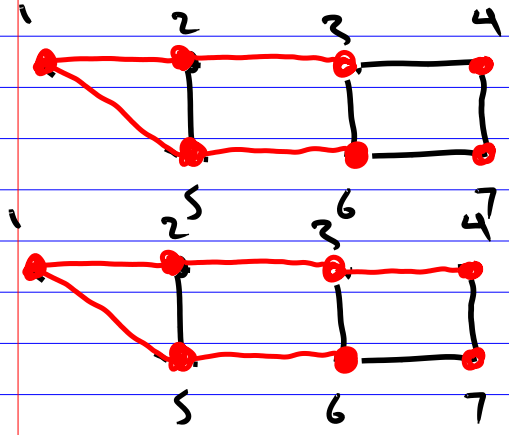
deleteQ() → 3

addQ(4)

→ 4 | 6 →

π = | – | 1 | 2 | 3 | 1 | 5 | – |
1 2 3 4 5 6 7

V = 1 1 1 1 1 1 0



deleteQ() → 6

addQ(7)

→ 7 | 4 →

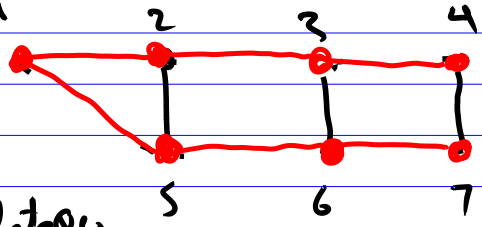
deleteQ() → 4

7

π, V unchanged

π = | – | 1 | 2 | 3 | 1 | 5 | 6 |
1 2 3 4 5 6 7

V = 1 1 1 1 1 1 1



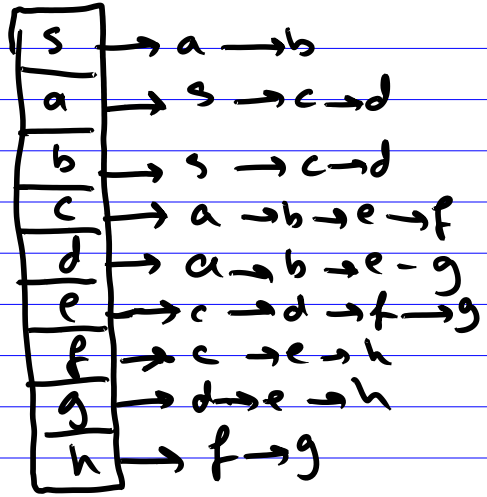
deleteQ() → 7

π, V unchanged

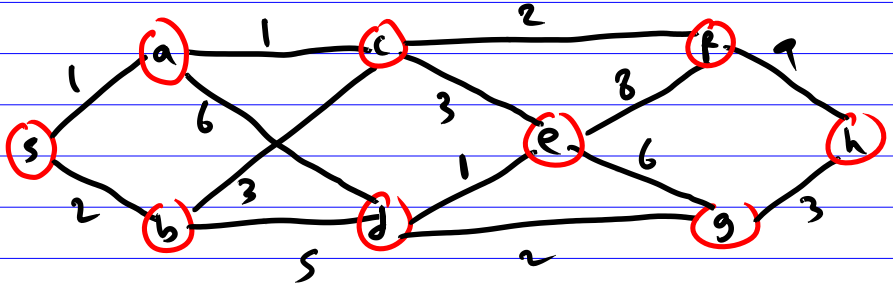
Q empty, π above, done! ←

→

Q 3a)



Q3b)



keep track of edges in the forest.

edges = 0

- Sort edges in increasing weight

sa	ac	de	sb	cf	dg	bc	ce	gh	bd	ad	eg	ef	fh
1	1	1	2	2	2	3	3	3	5	6	6	8	9

- make each node a single-node tree.

The set representation of the spanning forest:

s(-1) a(-1) b(-1) c(-1) d(-1) e(-1) f(-1) g(-1) h(-1)

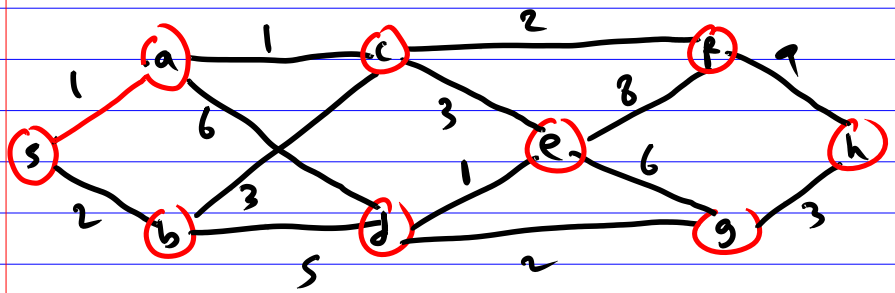
-1	-1	-1	-1	-1	-1	-1	-1	-1
s	a	b	c	d	e	f	g	h

1) scan edge sa : Find(s) Find(a), different → union

add sa

s(-2) a(s) b(-1) c(-1) d(-1) e(-1) f(-1) g(-1) h(-1)

-2	s	-1	-1	-1	-1	-1	-1	-1
s	a	b	c	d	e	f	g	h



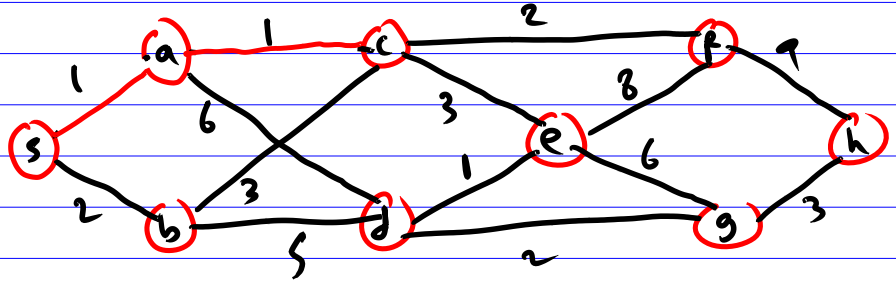
edges=1

2) ~~sa~~ ac de sb cf dg bc ce gh bd ad eg ef fh
1 1 2 2 2 3 3 3 5 6 6 8 9
↑

Check ac : find(a) find(c), different → union, add(ac)

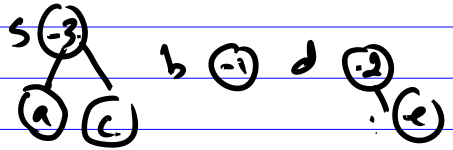
b -1 d -1 e -1 f -1 g -1 h -1

s	a	b	c	d	e	f	g	h
-3	s	-1	s	-1	-1	-1	-1	-1

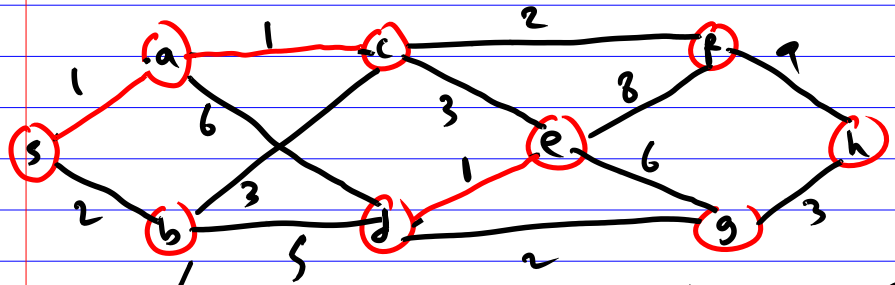


edges = 2

3) ~~sa~~ ~~ac~~ de sb cf dg bc ce gh bd ad eg ef fh
1 2 2 2 3 3 3 5 6 6 8 9
↑ check de find(d) find(e) different, union, add de

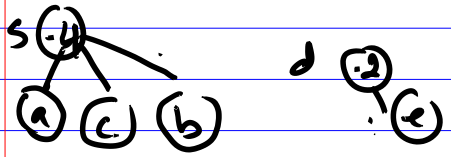


s	a	b	c	d	e	f	g	h
-3	s	-1	s	-2	d	-1	-1	-1

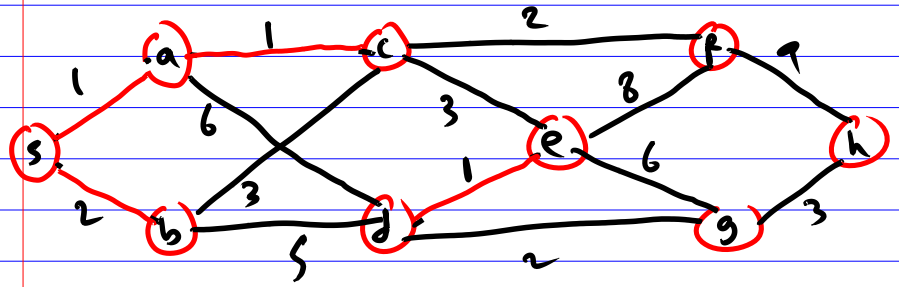


edges = 3

4) ~~sa~~ ~~ac~~ ~~de~~ sb cf dg bc ce gh bd ad eg ef fh
2 2 2 3 3 3 5 6 6 8 9
↑
check sb: Find(s) Find(b) different, add sb



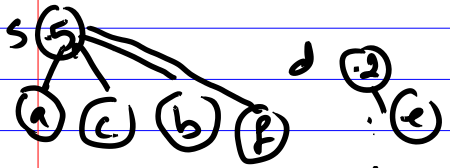
s	a	b	c	d	e	f	g	h
-4	s	s	s	-2	d	-1	-1	-1



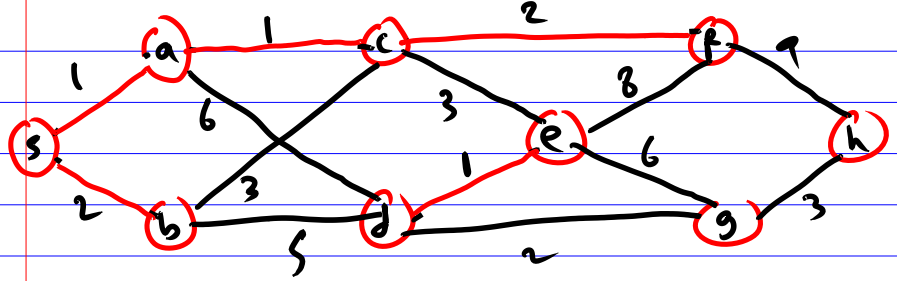
edges = 4

5) ~~sa~~ ~~ac~~ ~~de~~ ~~sb~~ cf dg bc ce gh bd ad eg ef fh
2 2 3 3 3 5 6 6 8 9

↑ check cf, find(c) find(f), diff set, union, add cf



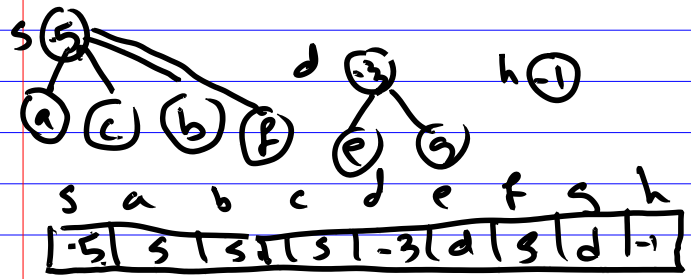
s	a	b	c	d	e	f	g	h
-5	s	s	s	-2	d	s	-1	-1

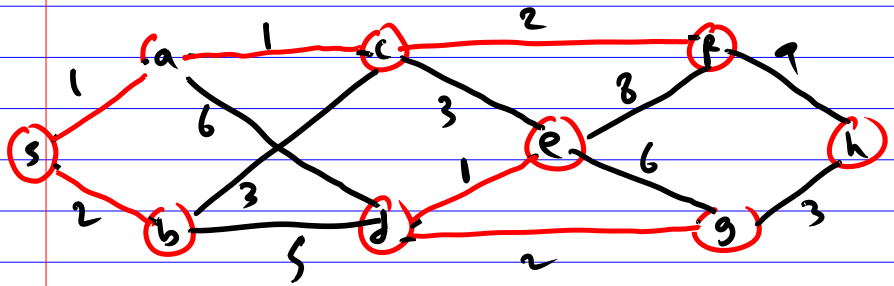


edges = 5

6) ~~sa~~ ~~ac~~ ~~de~~ ~~sb~~ ~~cf~~ dg bc ce gh bd ad eg ef fh
~~1~~ ~~2~~ ~~2~~ 2 3 3 3 5 6 6 8 9

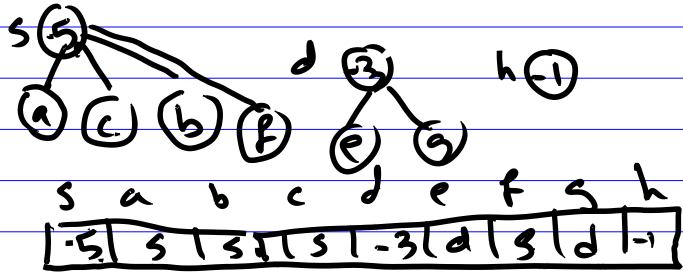
↑ check dg, find(d) → e find(g) → g
union, add dg.



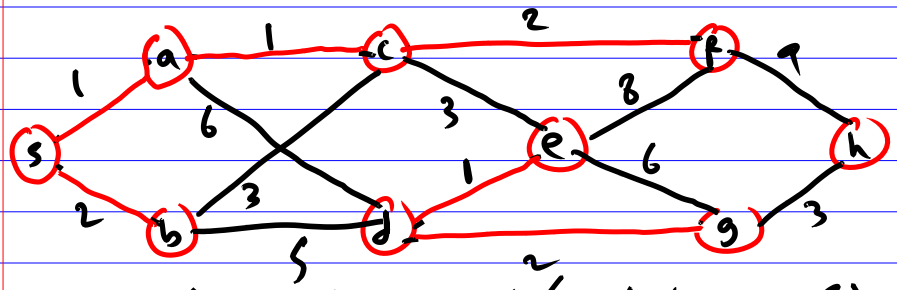


edges = 6

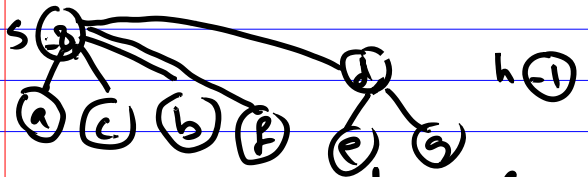
7) ~~sa~~ ~~ac~~ ~~de~~ ~~sb~~ ~~cf~~ ~~dg~~ bc ce gh bd ad eg ef fh
1 2 2 3 3 3 5 6 6 8 9
↑ check bc, find(b), find(c)
The same, do nothing



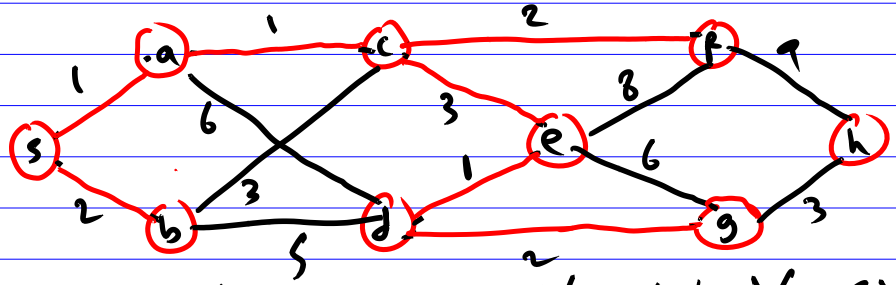
edges = 6



8) ~~sa~~ ~~ac~~ ~~de~~ ~~sb~~ ~~cf~~ ~~dg~~ ~~bc~~ ce gh bd ad eg ef fh
2 3 3 5 6 6 8 9
9
check ce, find(c)→s, find(e)→d
different, union, add ce



s	a	b	c	d	e	f	g	h
-8	s	s	s	s	s	d	s	d	-1

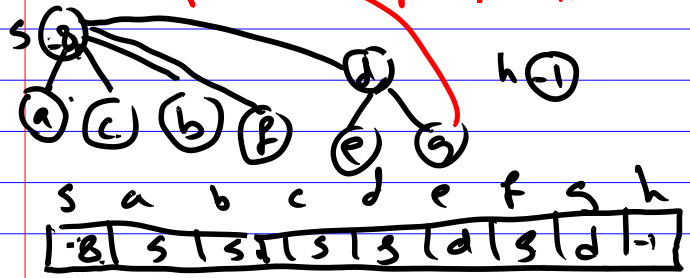


edges = 7

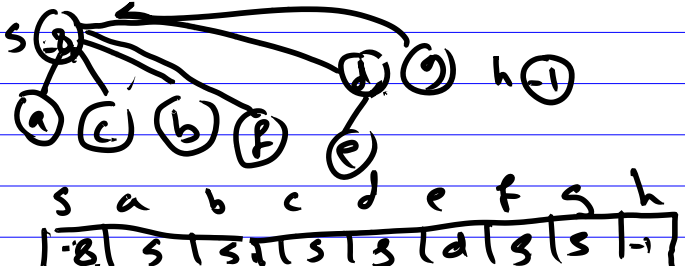
9) sa ac de sb cf dg bc cf gh bd ad eg ef fh

1 1 2 3 3 3 5 6 6 8 9

∴ check gh, find(g) → s find(h) → h
union, add gh.

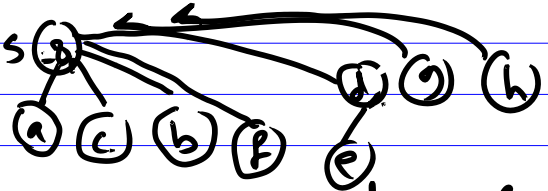


s	a	b	c	d	e	f	g	h
-8	s	s	s	s	s	d	s	d	-1

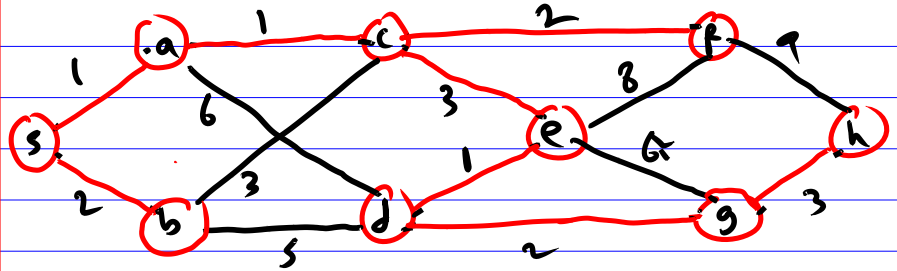


s	a	b	c	d	e	f	g	h
-8	s	s	s	s	s	d	s	s	-1

union:



s	a	b	c	d	e	f	g	h
g	s	s	s	s	d	s	s	s

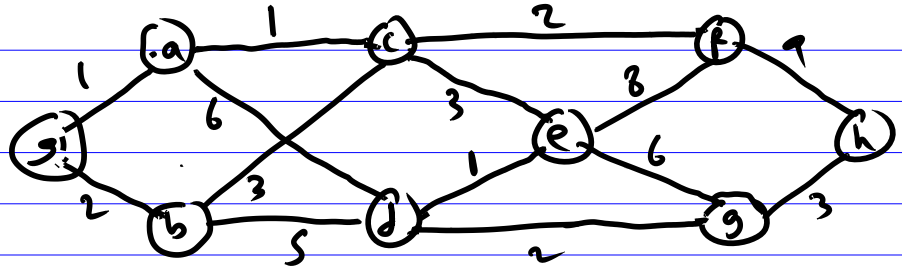


edges = 8

Since edges = 8
# of nodes = 9

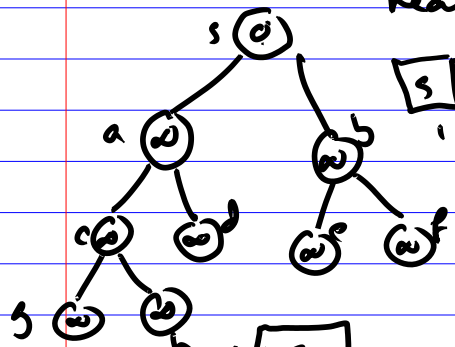
edges = # of nodes − 1
done!

Q3c)



Start with (s), the first node as the single node

Initialize the arrays and the heap:



heap

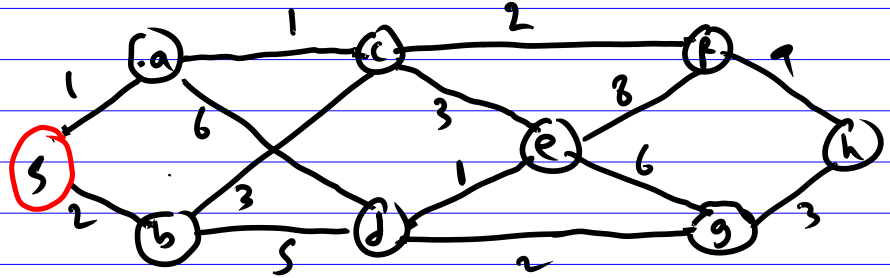
s	a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8	9

	s	a	b	c	d	e	f	g	h
in-heap	1	1	1	1	1	1	1	1	1
key	0	∞	∞	∞	∞	∞	∞	∞	∞
pred	–	–	–	–	–	–	–	–	–
Index	1	2	3	4	5	6	7	8	9

1	s	→ a → b
2	a	→ s → c → d
3	b	→ s → c → d
4	c	→ a → b → e → f
5	d	→ a → b → e → g
6	e	→ c → d → f → g
7	f	→ c → e → h
8	g	→ d → e → h
9	h	→ f → g

We follow vertices as listed in the adjacency list and edges also in the order in the list of each edge.

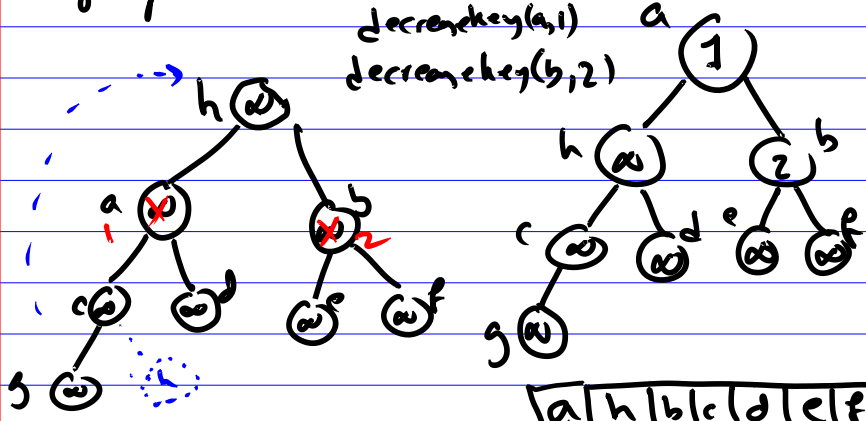
1) Delete heap ( ) ⟶ s, decrease key(a) decrease key(b)



key(a) = min(1, ∞) = 1
key(b) = min(2, ∞) = 2
pred(a) = s
pred(b) = s

heap after delete min

decreasekey(a,1)
decreasekey(b,2)



	s	a	b	c	d	e	f	g	h
in-heap	0	1	1	1	1	1	1	1	1
key	0	1	2	∞	∞	∞	∞	∞	∞
pred	–	s	s	–	–	–	–	–	–
Index	–	1	3	4	5	6	7	8	2

h	a	b	c	d	e	f	g	s
1	2	3	4	5	6	7	8	9

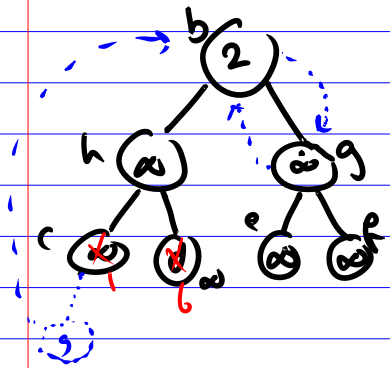
a	h	b	c	d	e	f	g	s
1	2	3	4	5	6	7	8	9

2) DeleteMin() → a (pred(a), a) edge added to tree

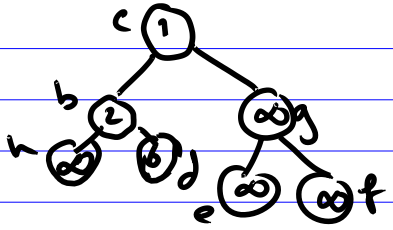
scan adj list(a) and decreasekey:

decreasekey(c), decreasekey(d)

key(c) = min(∞, 2) = key(d) = min(∞, 6) = 6

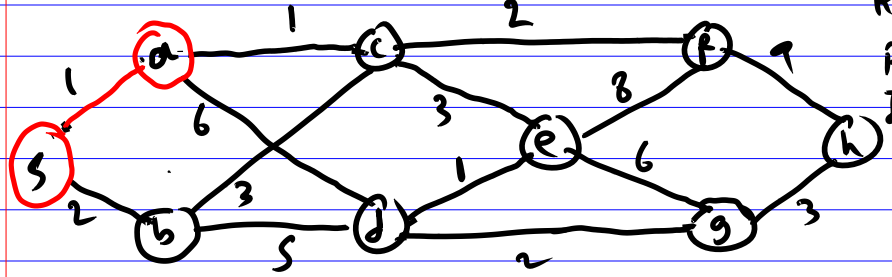


⟹ decreasekey(c, 1) = 1 decreasekey(d, 6) = 6



b	h	d	c	d	e	f		
1	2	3	4	5	6	7	8	9

c	b	g	h	d	e	f
1	2	3	4	5	6	7



	s	a	b	c	d	e	f	g	h
in-heap	0	0	1	1	1	1	1	1	1
key	0	1	2	1	6	∞	∞	∞	∞
pred	–	s	s	a	a	–	–	–	–
Index	–	–	2	1	5	6	7	8	2

3) deleteMin() → c (pred(c), c) = (a, c) → to tree

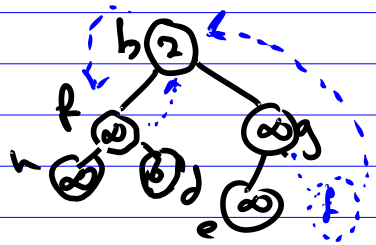
scan list(c) b, e, f

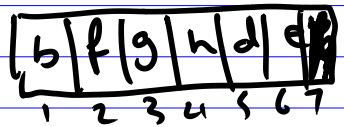
key(b) = min(2, 3) = 2 don't change

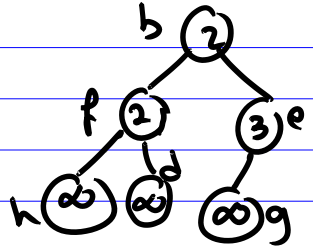
key(e) = min(∞, 3) = 3 decreaseKey(e, 3)

key(f) = min(∞, 2) = 2 decreaseKey(f, 2)

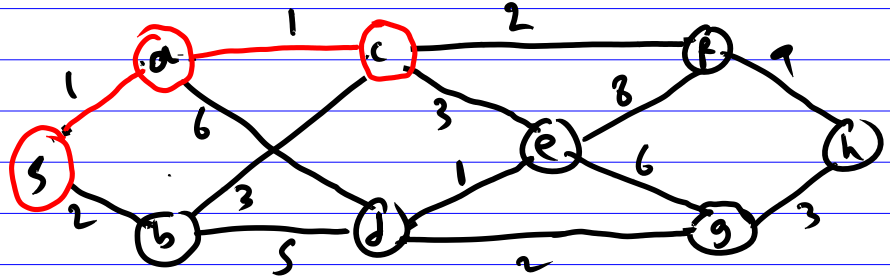
decreaseKey(e, 3) , decreaseKey(f, 3)







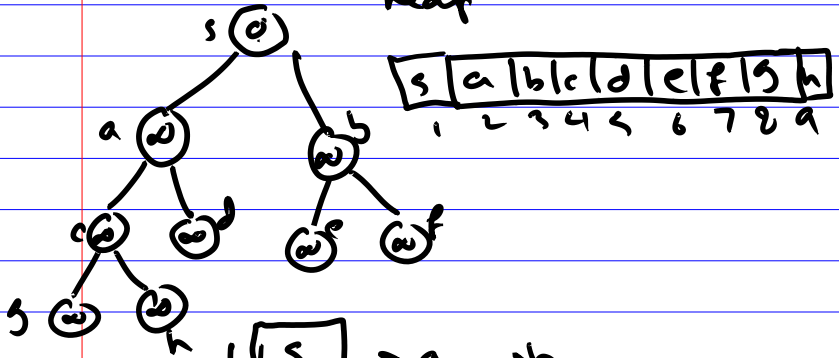
	s	a	b	c	d	e	f	g	h
in-heap	0	0	1	0	1	1	1	1	1
key	0	1	2	1	6	3	2	∞	∞
pred	–	s	s	a	a	c	c	–	–
Index	–	–	!	–	5	3	2	8	2



The rest is similar, you should be able to do it yourself....

(Q3d) Start with (s), the first node as the single node

Initialize the arrays and the heap:



heap

s	a	b	c	d	e	f	g	h
1	2	3	4	5	6	7	8	9

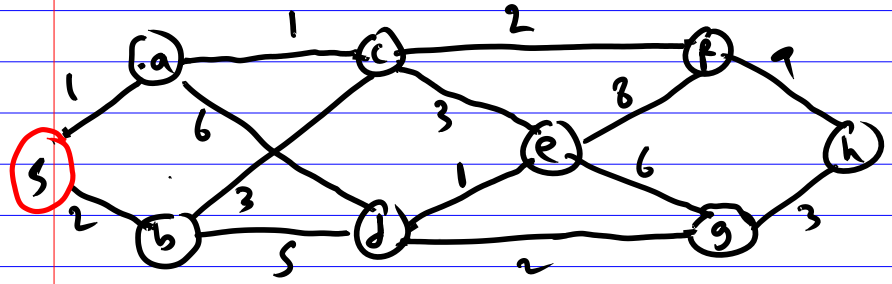
	s	a	b	c	d	e	f	g	h
in-heap	1	1	1	1	1	1	1	1	1
keys	0	∞	∞	∞	∞	∞	∞	∞	∞
pred	–	–	–	–	–	–	–	–	–
Index	1	2	3	4	5	6	7	8	9

1	s	→ a → b
2	a	→ s → c → d
3	b	→ s → c → d
4	c	→ a → b → e → f
5	d	→ a → b → e → g
6	e	→ c → d → f → g
7	f	→ c → e → h
8	g	→ d → e → h
9	h	→ f → g

We follow vertices as listed in the adjacency list and edges also in the order in the list of each edge.

Dijkstra is exactly as Prim, except for updating keys.

1) Delete heap ( ) → s, decrease key(a) decrease key(b)



key(a) = min(1, ∞) = 1

key(b) = min(2, ∞) = 2

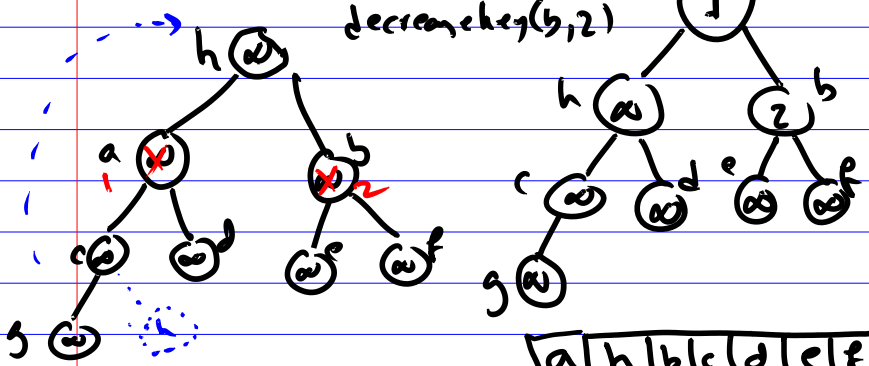
pred(a) = s

pred(b) = s

heap after delete min

decreasekey(a,1)

decreasekey(b,2)



	s	a	b	c	d	e	f	g	h
in-heap	0	1	1	1	1	1	1	1	1
key	0	1	2	∞	∞	∞	∞	∞	∞
pred	–	s	s	–	–	–	–	–	–
Index	–	1	3	4	5	6	7	8	2

h	a	b	c	d	e	f	g	s
1	2	3	4	5	6	7	8	9

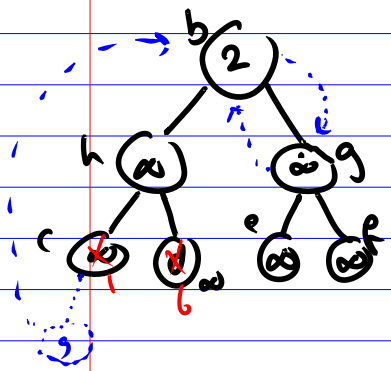
a	h	b	c	d	e	f	g	s
1	2	3	4	5	6	7	8	9

2) DeleteMin() → a (pred(a), a) edge added to tree

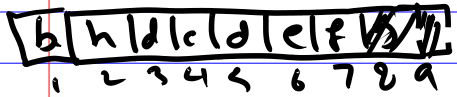
scan adj list(a) and decreasekey:

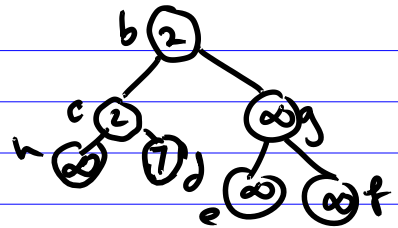
decreasekey(c), decreasekey(d)

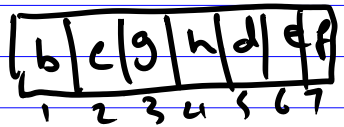
key(c) = min(∞, key(a)+6) = 7 key(d) = min(∞, key(a)+1) = 2

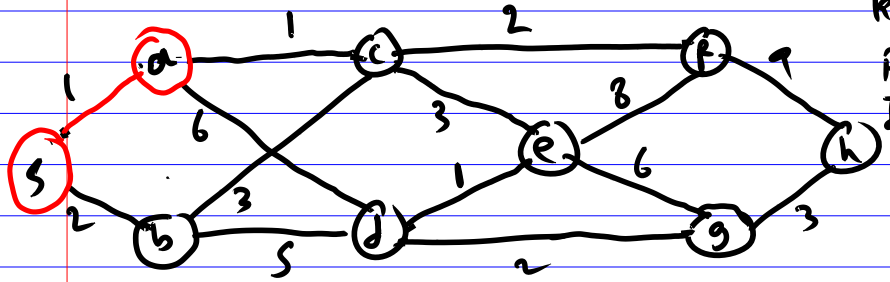


⟹ decreasekey(c, 1) = 1 decreasekey(d, 6) = 6









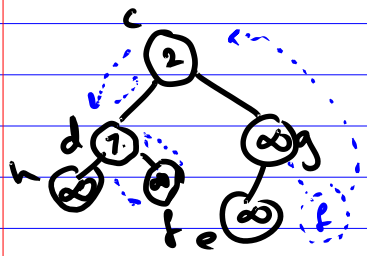
	s	a	b	c	d	e	f	g	h
in-heap	0	0	1	1	1	1	1	1	1
key	0	1	2	2	7	∞	∞	∞	∞
pred	–	s	s	a	a	–	–	–	–
Index	–	–	1	2	5	6	7	8	2

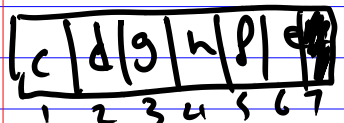
3) deleteMin() → b (pred(b), b) = (s, b) → to tree

scan list(b) → c, d

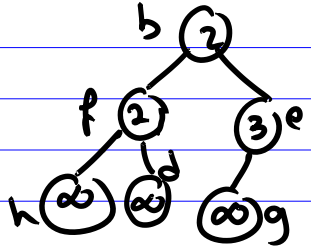
key(c) = min(2, key(b)+3) = 2 don't change

key(d) = min(7, key(b)+5) = 7 don't change

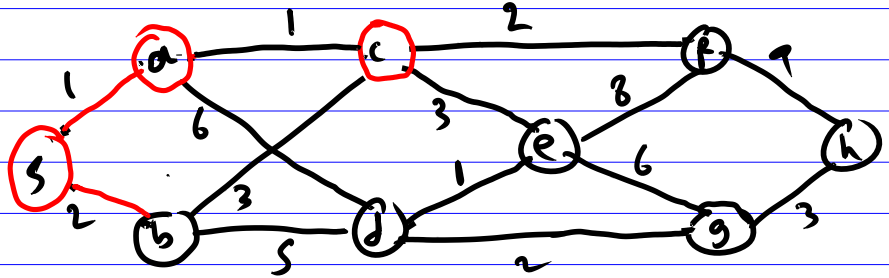




new decrease_key() operation



	s	a	b	c	d	e	f	g	h
in_heap	0	0	0	1	1	1	1	1	1
key	0	1	2	1	6	3	2	∞	∞
pred	–	s	s	a	a	c	c	–	–
Index	–	–	–	–	5	3	2	8	8



The rest is similar, you should be able to do it yourself....

Q4)

	run 0		run 1		run 2	
	l	π	l	π	l	π
a b	∞	-	2	a		
b c	∞	-	1	b		
b e	∞	-	10	b		
c d	∞	-	2	c		
d e	∞	-	-1	d		
e d	∞	-	2	c		
e f	∞	-	0	e		
f a	0	-	-1	f		
f e	∞	-	-1	d		

π = predecessor

We could continue, but since the source, a, now has a negative label indicates we found a negative cycle containing a. In fact following predecessors a, f, e, d, c, b, a

$\overset{-1}{\leftarrow} \overset{1}{\leftarrow} \overset{-3}{\leftarrow} \overset{1}{\leftarrow} \overset{-1}{\leftarrow} \overset{2}{\leftarrow} = -1$