Discrete Maths – Assignments Solutions

Assignment 1

1) For any sets A and B, define A Δ B, the symmetric difference of A and B to be the set (A − B) ∪ (B − A). Prove that intersection ∩ distributes over Δ : A ∩ (B Δ C) = (A ∩ B) Δ (A ∩ C) for all sets A, B, and C using the proof methods employed in lecture. Venn diagrams, truth tables, or diagrams for simplifying statements in Boolean algebra such as Veitch diagrams are NOT acceptable and will not be awarded any credit.

$$\begin{aligned} A \cap (B \,\Delta\, C) &= A \cap ((B - C) \cup (C - B)) \\ &= [A \cap (B - C)] \cup [A \cap (C - B)] \end{aligned}$$

$$\begin{aligned} (A \cap B) \,\Delta\, (A \cap C) &= [(A \cap B) - (A \cap C)] \cup [(A \cap C) - (A \cap B)] \\ &= [A \cap (B - C)] \cup [A \cap (C - B)] \end{aligned}$$

Q.E.D

2) Let R be the set of real numbers. For x, y ∈ R, x ∼ y iff x − y ∈ Q, i.e., if the difference x − y is a rational number. Determine whether or not the relation ∼ is :

a) Reflexive

b) Symmetric

c) Anti-Symmetric

d) Transitive

e) An Equivalence Relation

f) A Partial Order

a) If ∼ is a reflexive relation that implies for all x ∈ R, x ∼ x. In other words meaning that for all x ∈ R, $x - x$ ∈ Q. This is true since x – x = 0 ∈ Q.

b) If ∼ is a symmetric relation that implies for all x,y ∈ R, x ∼ y = y ∼ x. This true since x – y = q ∈ Q and since y – x = -q ∈ Q.

c) If ∼ is an anti-symmetric relation that implies for all x,y ∈ R, if x ∼ y and y ∼ x then y = x. This is not true, take for example x = 1 and y = 0, x ∼ y = x – y = 1 – 0 = 1 ∈ Q. However, y ∼ x = 0 – 1 = - 1 ∈ Q. Clearly x != y.

d) If $\sim$ is a transitive relation that implies for all x,y,z $\in R$, if x $\sim y$ and y $\sim z$ then x $\sim z$. This is true, because for any x,y,z $\in R$ if x – y = p and y – z = q and p,q $\in Q$ then x – z = (x – y) - (x – z) = p – q $\in Q$.

e) $\sim$ is an equivalence relation because it is reflexive, symmetric and transitive.

f) $\sim$ is not a partial order as it fails to be anti-symmetric as shown above.

6) Let $f : [-2, 0] \rightarrow [0, 1]$ be the function defined by $f(x) = 1 / x^2 + 6x + 9$ for all $x \in [-2, 0]$.

Determine whether or not this function is injective and whether or not it is surjective. Justify your answers. Recall that $[-2, 0]$ is the set of all real numbers between −2 and 0 with the end-points of −2 and 0 included in the set.

a) f is an <u>injective</u> function as there does not exist an x, y $\in$ [-2, 0] s.t f(x) = f(y) = p. Each input in the domain maps to a unique output in the codomain.

b) f is not a <u>surjective</u> function as for all x $\in$ [-2, 0], y $\in$ [0, 1] there does not exist an f(x) = y. For example there does not exist an x $\in$ [-2, 0] s.t f(x) = 0.

7) Let $A = \{(x, y) \in R^2 \mid 2x - 3y = 0\}$ with the operation of addition given by

$$(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$$

a) Is (A, +) a semigroup? Justify your answer.

b) Is (A, +) a monoid? Justify your answer.

c) Is (A, +) a group? Justify your answer.

a) *(A, +)* is a semigroup as it is a set endowed with an associative binary operation (addition) since $(x_1, y_1) + (x_2, y_2) = (x_2, y_2) + (x_1, y_1)$. Also, this can be associated over 2x – 3y = 0. Namely, $2(x_1, x_2) + 3(y_1, y_2) = 0$.

b) *(A, +)* is a monoid as it has an associated identity element which is $(x_1, y_1) = (0, 0)$. This is an identity element as for any $(x_1, y_1) \in R^2$ $(x_1, y_1) + (0, 0) = (x_1, y_1)$.

c) *(A, +)* is a group since for all $(x_1, y_1) \in R^2$ there exists an $(x_1, y_1)^{-1}$ namely in the form of

$(-x_1, -y_1)$.

Assignment 2

Question 1

a) Describe the formal language over the alphabet {a, b, c} generated by the context-free grammar whose non-terminals are S and A , whose start symbol is S , and whose production rules are the following:

(1) <S> → a <A>

(2) <A> → b <S>

(3) <S> → c

L = { $(ab)^n$ c | $n \in N,\ n \geq 0$ }

Words generated under L are of the following form:

- $(ab)^n$ <S>
- $(ab)^n$ a <A>
- $(ab)^n$ c

b) Is this grammar regular? Justify your answer.

Yes, this grammar is regular as all of the above production rules satisfy the criteria necessary for a grammar to be regular. In other words, all of the production rules are of the form:

a) <A> → b <B>

b) <A> → b

c) <A> → ε

From our production rules given above, (1) is of type (a), (2) is of type (a) and (3) is of type (b) thus satisfying that it is a regular grammar.

c) Is this grammar innormal form? If it is not in normal form, then modify it to make it be in normal form. Explain why it generates the same language after it has been modified.

No, this grammar is not in normal form. To modify it to be in normal form the grammar would have to look like this:

L = { $(ab)^n c \mid n \in N,\ n \geq 0$ }

1) *<S> → a <A>*
2) *<A> → b <S>*
3) *<S> → c <B>*
4) *<B> → ε*

As before, words generated under this grammar are of the following form:

- $(ab)^n$ <S>
- $(ab)^n$ a <A>
- $(ab)^n$ c <B>
- $(ab)^n$ c

d) Write down a regular expression that gives the language from part (a) and justify your answer.

This can be generated under the following regular expression:

(ab)*c

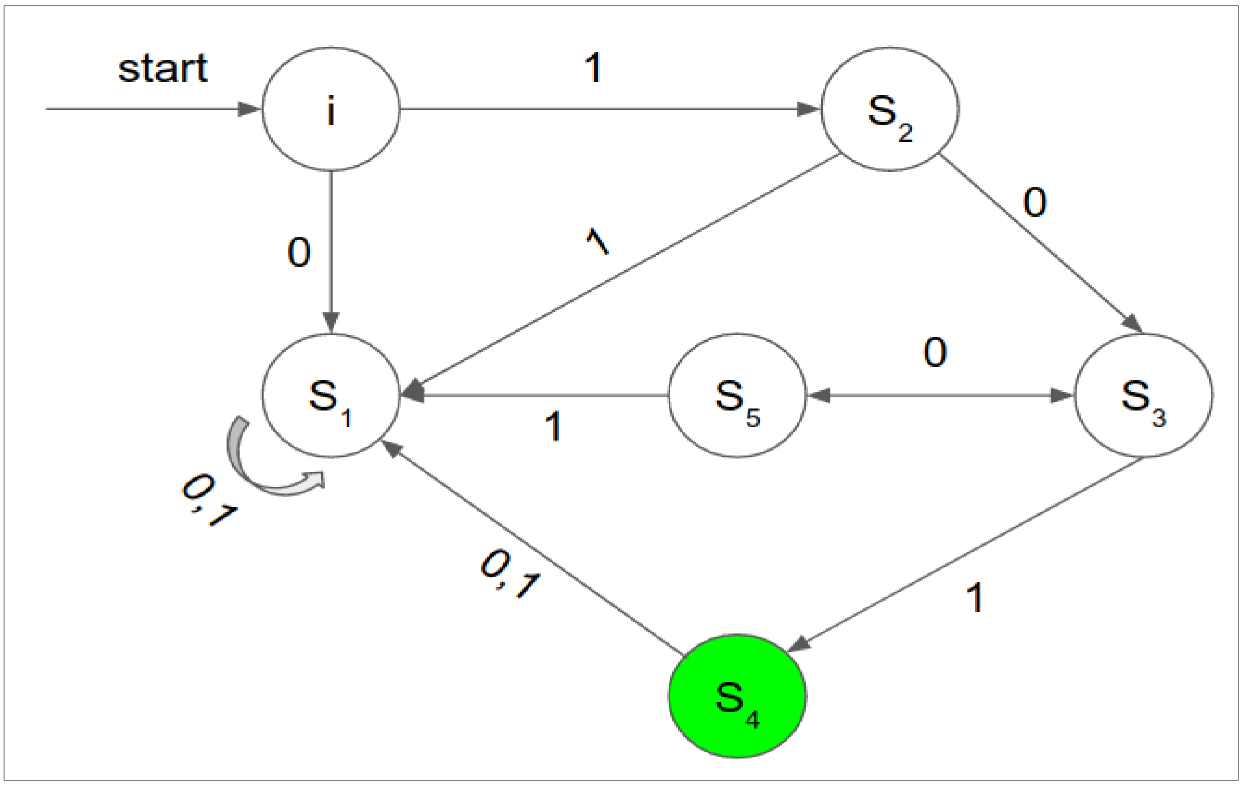
This expression gives the language K = { $(ab)^n c : n \geq 0$ } which is exactly our language L.

Question 2

Let L be the language consisting of all the strings of the form $10^{2m+1}1$ for m a natural number, m ≥ 0, i.e.

{101, 10001, 1000001, 100000001, . . . }

a) Draw a deterministic finite state acceptor that accepts the language L. Carefully label all the states including the start state and the finishing states as well as all the transitions. Make sure you justify it accepts all strings in the language L and no others.



A finite state acceptor (S, A, i, t, F) consists of:

- A finite set S of states
- A finite set A that is the input alphabet
- A starting state i ∈ S
- A transition mapping t : S × A → S
- A set F of finishing states, where F ⊆ S

- Accepting states: S_4
- Non accepting states: I, S_1, S_2, S_3, S_5
- Start states: I

- S = {i, S_1, S_2, S_3, S_4, S_5}
- F = {S_4}
- A = {0, 1}
- t : S × A → S

- $t(i, 0) = S_1$
- $t(i, 1) = S_2$

- $t(S_1 , 0) = S_1$
- $t(S_1 , 1) = S_1$

- $t(S_2 , 0) = S_3$
- $t(S_2 , 1) = S_1$
-
- $t(S_3 , 0) = S_5$
- $t(S_3 , 1) = S_4$

- $t(S_4 , 0) = S_1$
- $t(S_4 , 1) = S_1$

- $t(S_5 , 0) = S_3$
- $t(S_5 , 1) = S_1$

It can be shown that this FSA only accepts all strings in the language L and no others as if the string does not start with 1 it immediately transitions to an endless reject state (S_1).

States S_2 , S_3 , and S_5 ensure that there are 2m + 1 zero's following from the previous 1 value. If this is not the case, the FSA will transition once again to the endless reject state (S_1).

<u>Test 10001:</u>

String	1	0	0	0	1
State	S_2	S_3	S_5	S_3	S_4
Output	NO	NO	NO	NO	YES

<u>Test 01011:</u>

String	0	1	0	1	1
State	S_1	S_1	S_1	S_1	S_1
Output	NO	NO	NO	NO	NO

It can be therefore taken that the Finite State Acceptor as shown above accepts all strings in the language L and no other strings.

b) Devise a regular grammar in normal form that generates the language L. Be sure to specify the start symbol, the non-terminals and all the production rules.

$L = \{10^{2m+1}\,1 \mid m \text{ a natural number}, m \geq 0\}$

{101, 10001, 1000001, 100000001, . . . }

Start Symbol = <S>

Non-Terminals = <A>, <B>, <C>, <D>, <E>

Production Rules:

1. <S> → 1 <A>
2. <A> → 0 <B>
3. <A> → 0 <D>
4. <B> → 1 <C>
5. <C> → ε
6. <D> → 0 <E>
7. <E> → 0 <A>
8. <E> → 0 <B>

101	10001	10000001
1. → 1	1. → 1	1. → 1
2. → 10	3. → 10	3. → 10
4. → 101	6. → 100	6. → 100
5. → 101ε	8. → 1000	7. → 1000
	4. → 10001	3. → 10000
	5. → 10001ε	6. → 100000
		8. → 1000000
		4. → 10000001
		5. → 10000001ε

Question 3

Let M be the language

{0101, 001001, 00010001, 0000100001, . . . }

$M = \{ (0^m 1)^m \mid m \in N,\ m \geq 1 \}$

(a) Use the Pumping Lemma to show this language is not regular.

<u>The Pumping Lemma</u>

If A is a Regular Language, then A has a Pumping Length ‘P’ such that any string ‘S’ where $|S| \geq P$ may be divided into 3 parts S = xyz such that the following conditions are true

1. $x y^i z \in A$ for every $i \geq 0$
2. $|y| \geq 0$
3. $|xy| < P$

<u>To prove that a language is NOT REGULAR using the Pumping Lemma:</u>

1. Assume that A is regular
2. A must then have a Pumping Length P
3. All strings longer than P can be pumped $|S| \geq P$
4. Now find a string ‘S’ in A such that $|S| \geq P$
5. Divide S into x y and z
6. Show that $x y^i z \,!\!\in A$ for some i
7. Then consider all ways that S can be divided into x y and z
8. Show that none of these satisfy all 3 pumping conditions at the same time
9. S cannot be Pumped == CONTRADICTION

<u>Proof:</u>

Assume M is a regular language, therefore it must have a pumping length p such that a string $S = (0^P 1)^P$ and S is a word under the language M.

1. Divide such a string S int [x] [y] and [z]
2. Let the pumping length P = 7

   Therefore, S = 0000000100000001

3. Explore the different possibilities of breaking up S into [x] [y] and [z]

Case 1 (The y is right of 1):

$$\overset{x}{[00000001]}\ \overset{y}{[0000]}\ \overset{z}{[0001]}$$

Case 2 (The y contains 1):

$$\overset{x}{[0000000]}\ \overset{y}{[1000]}\ \overset{z}{[00001]}$$

Case 3 (The y is left of 1):

$$\overset{x}{[000]}\ \overset{y}{[0000]}\ \overset{z}{[100000001]}$$

Show that x y^i z !∈ M for some I
Let i = 2, therefore test S = x y^2 z

Case 1: $S = \overset{x}{[00000001]}\overset{y^2}{[00000000]}\overset{z}{[0001]}$

This S !∈ M since 7 x 0's and 1 x 1 followed by 11 x 0's and 1 x 1 which does not satisfy
$M = \{ (0^m1)^m \}$ and therefore S !∈ M.

Case 2: $S = \overset{x}{[0000000]}\overset{y^2}{[10001000]}\overset{z}{[00001]}$

This S !∈ M since 7 x 0's and followed by $(1000)^2$ and then further followed by 0001 which does not satisfy $M = \{ (0^m1)^m \}$ and therefore S !∈ M.

Case 3: $S = \overset{x}{[000]}\overset{y^2}{[00000000]}\overset{z}{[100000001]}$

This S !∈ M since 11 x 0's and 1 x 1 followed by 7 x 0's and 1 x 1 which does not satisfy
$M = \{ (0^m1)^m \}$ and therefore S !∈ M.

****REVERT BACK TO CONDITIONS 1-3 AND SHOW HOW THEY ARE NOT PROVED****

Condition 1 proven false above.

Condition 2 is true.

Condition 3 - $|xy| < P$ – is false in all of the above cases.

Therefore using the Pumping Lemma it is not possible for all strings under L to satisfy the conditions of the Pumping Lemma, therefore the language L is not regular.

(a) Write down the production rules of a context-free grammar that generates exactly M.

M = { $(0^m1)^m$ | m ∈ N, m ≥ 1 }

{0101, 001001, 00010001, 0000100001, . . . }

Production Rules:

1. <S> → 0 <A> 01
2. <A> → 0 <A> 0
3. <A> → 1

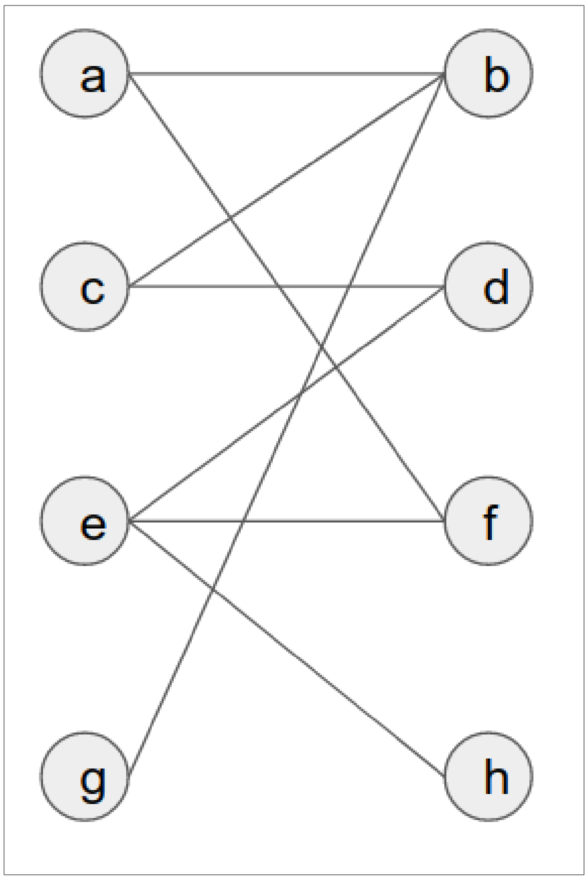
All words generated under this grammar are of the following form:

- 0 <A> 01
- 0^n <A> 0^n1
- 0^n10^n1

Question 4

Let (V, E) be the graph with vertices a, b, c, d, e, f, g, and h, and edges ab, bc, cd, de, ef, af, bg, and eh.

a) Draw this graph.



b) Write down the graphs incidence table & incidence matrix

	AB	BC	CD	DE	EF	AF	BG	EH
A	1					1		
B	1	1					1	
C		1	1					
D			1	1				
E				1	1			1
F					1	1		
G							1	
H								1

c) Write down the graphs adjacency table & adjacency matrix

	A	B	C	D	E	F	G	H
A	0	1				1		
B	1	0	1				1	
C		1	0	1				
D			1	0	1			
E				1	0	1		1
F	1				1	0		
G		1					0	
H					1			0

d) Is this graph complete?

No, this graph is not complete. As for in order for a graph to be complete there must exist the maximum number of edges possible for the given vertices. This is not the case for the graph (V, E) for example there does not exist the edge fg.

e) Is this graph bipartite?

Yes this graph is bipartite. As for in order for a graph to be bipartite there must exist two distjoint subsets A = (V_1), B = (V_2) such that A U B = (V) and that for every edge ab an element of (E), a must be an element of V_1 and b must be an element of V_2.

Let A = (V_1 , E_1)
V_1 = { a, c, e, g}

Let B = (V_2 , E_2)
V_2 = { b, d, f, h}

Take for example the edge ef, e $\in V_1$ and f $\in V_2$.

e) Is this graph regular?

No this graph is not regular. As for a graph to be regular the deg(v) must be equal for all v an element of V, where deg(v) is the degree of a given vertex.

This is not the case for the given graph. For example deg(e) = 3 whilst deg(h) = 1.

f) Does this graph have a regular subgraph?

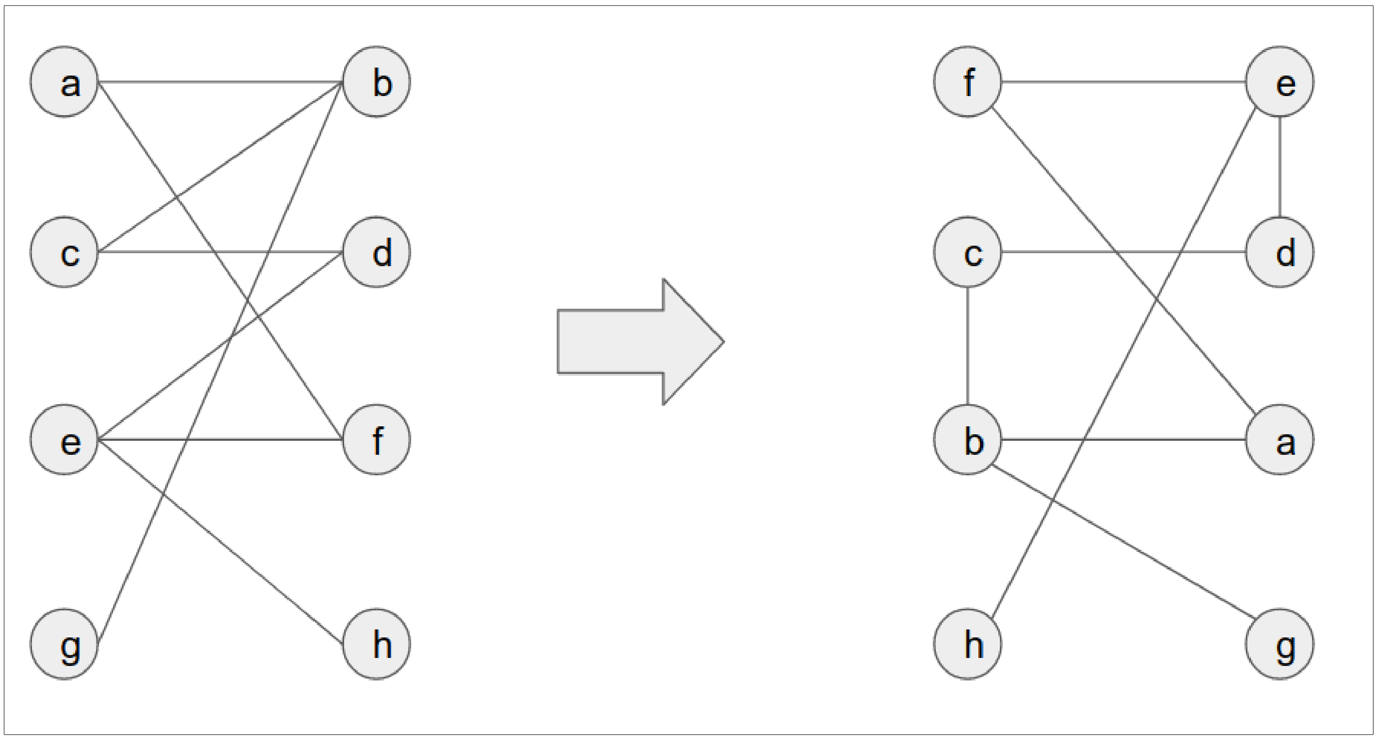
Yes, this graph does contain a regular subgraph.

Let the subgraph be (V', E') s.t

V' = { a, b, c, d, e, f }
E' = { ab, af, bc, cd, ef, de}

This is a regular subgraph since for all v an element of V', deg(v) = 2.

g) Give an example of an isomorphism φ from the graph (V, E) to itself satisfying that φ (b) = e.



φ (a) = f
φ (b) = e
φ (c) = c
φ (d) = d
φ (e) = b
φ (f) = a
φ (g) = h
φ (h) = g

Assignment 3

Question 1

Let (V, E) be the graph with vertices a, b, c, d, e, f, g, and h, and edges ab, bc, cd, de, ef, af, bg, and eh.

a) Is this graph connected? Justify your answer.

Yes this graph is as for a graph to be connected there must exist a path in the graph from every u to every v.

This is true for the graph (V, E) as for all v an element of V there exists a path $e_1, \dots , e_i$. In other words, there exists a route through the graph from all v_1 to v_2.

b) Does this graph have an Eulerian trail? Justify your answer.

An Eulerian trail is a trail that traverses every edge of that graph. In other words, an Eulerian trail is a walk that traverses every edge of the graph exactly once.

Trail ⇒ an **edge** is traversed **at most once**.
Eulerian ⇒ **every edge** is traversed..

No, there does not exist an Eulerian trail within the graph (V, E). As there is no possible path that traverses **every edge** of the graph **at most once**.

This can also be shown as four vertices have odd degree instead of two, deg b = deg e = 3 and deg g = deg h = 1. Therefore there is no possible Eulerian trail.

c) Does this graph have an Eulerian circuit? Justify your answer.

An Eulerian circuit is a circuit that traverses every edge of that graph. In other words, an Eulerian circuit is a circuit that traverses every edge of the graph exactly once.

For an Eulerian circuit to exist all vertices must have an even degree. As shown above four vertices have odd degrees, namely: b, e, g, h.

d) Does this graph have an Hamiltonian circuit? Justify your answer.

A Hamiltonian circuit is a circuit that passes through every vertice of that graph at most once.

This can not exist within the graph (V, E) as there exists two pendant vertices, g and h. If we had a Hamiltonian circuit, it would have to pass through the two vertices adjacent to the pendant vertices, b and e, twice, which is not allowed under the definition of a Hamiltonian circuit.

e) Is this graph a tree?

The graph (V, E) is not a tree as there exists a circuit *abcdef.*

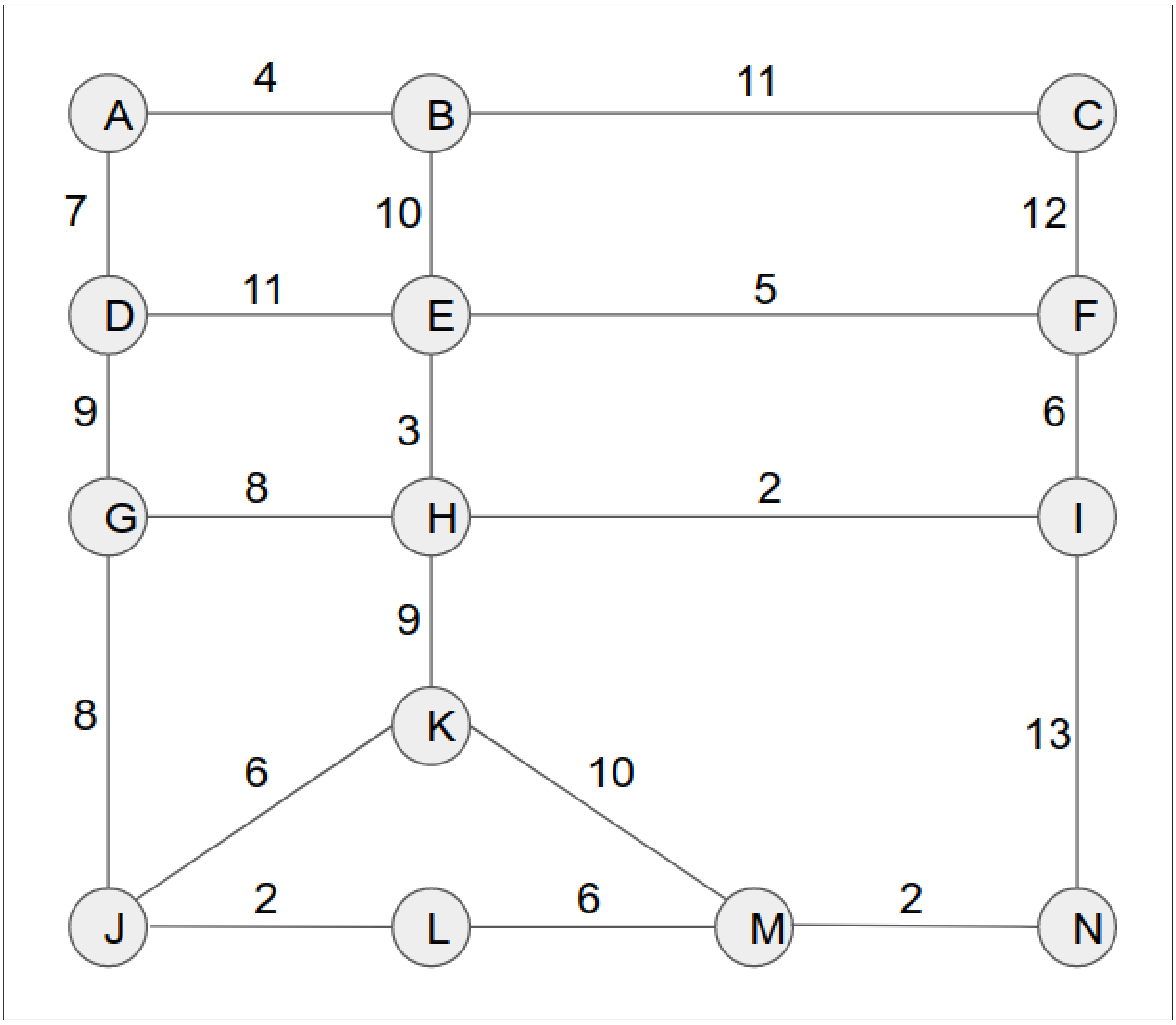
Another reason that (V, E) is also not a tree is the fact that in order for a graph to be a tree the number of edges must equal the number of vertices -1. In other words, #E = #V -1. This is not the case for (V, E) as #E = 8 and #V = 8. Therefore, #E != #V -1.

Question 2

Consider the connected graph with vertices A, B, C, D, E, F , G, H, I, J, K, L, M and N and with edges, listed with associated costs, in the following table:

HI	JL	MN	EH	AB	EF	FI	LM	JK	AD
2	2	2	3	4	5	6	6	6	7
GH	GJ	HK	DG	KM	BE	DE	BC	CF	IN
8	8	9	9	10	10	11	11	12	13

a) Draw the graph and label each edge with its cost

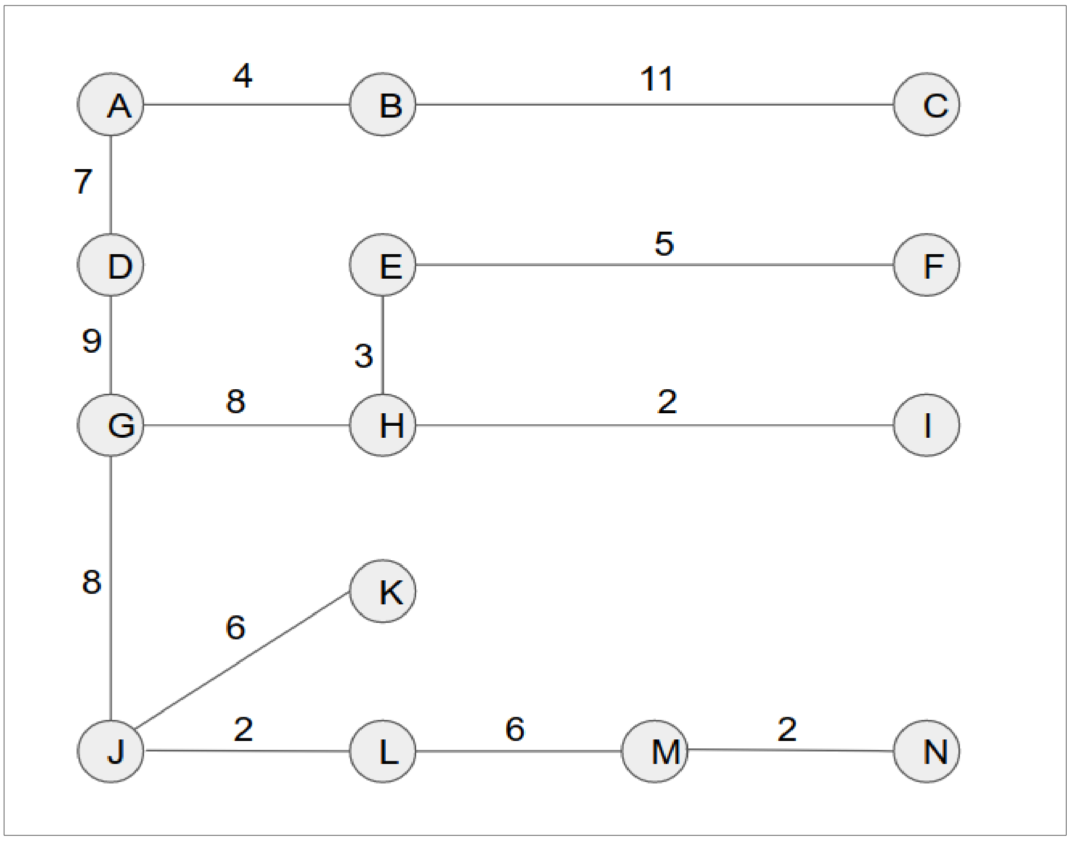


b) Determine the minimum spanning tree generated by Kruskals Algorithm, where that algorithm is applied with the queue specified in the table above. For each step of the algorithm list the edge that is added and draw the graph.

HI	JL	MN	EH	AB	EF	FI	LM	JK	AD
2	2	2	3	4	5	6	6	6	7
GH	GJ	HK	DG	KM	BE	DE	BC	CF	IN
8	8	9	9	10	10	11	11	12	13

Edges are added in the following order:

- HI, JL, MN, EH, AB, EF
- Skip FI (creates circuit)
- LM, JK, AD, GH, GJ
- Skip HK (creates circuit)
- DG
- Skip KM (creates circuit)
- Skip BE (creates circuit)
- Skip DE (creates circuit)
- BC
- Skip CF (creates circuit)
- Skip IN (creates circuit)



b) Determine the minimum spanning tree generated by Prims Algorithm, starting from the vertex F, where that algorithm is applied with the queue specified in the table above. For each step of the algorithm list the edge that is added and draw the graph.

HI	JL	MN	EH	AB	EF	FI	LM	JK	AD
2	2	2	3	4	5	6	6	6	7
GH	GJ	HK	DG	KM	BE	DE	BC	CF	IN
8	8	9	9	10	10	11	11	12	13

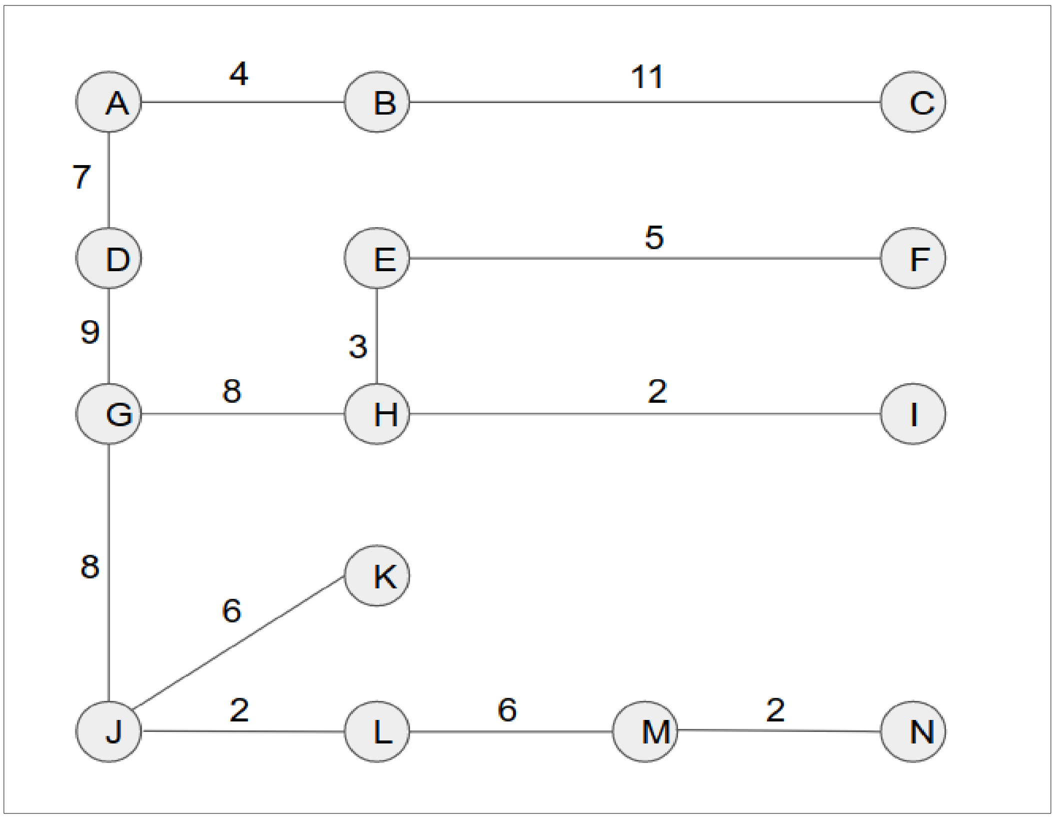
Starting with vertex F:

- Check edges adjacent to F
- Choose edge with least cost and mark it as settled (e.g EF)
- Ensure added edge does not create circuit

- Check edges adjacent to E and F
- Choose edge with least cost and mark it as settled (e.g EH)
- Ensure added edge does not create circuit
- Repeat until all vertices have been marked

Edges added in the following order:

- EF, EH, HI, GH, GJ, JL, LM, MN, JK, DG, AD, AB, and BC



Assignment 4

Question 1

a) Is {(x, sin(πx)) | x ∈ R} ∩ {Q × Z} finite, countably infinite, or uncountably infinite? Justify your answer.

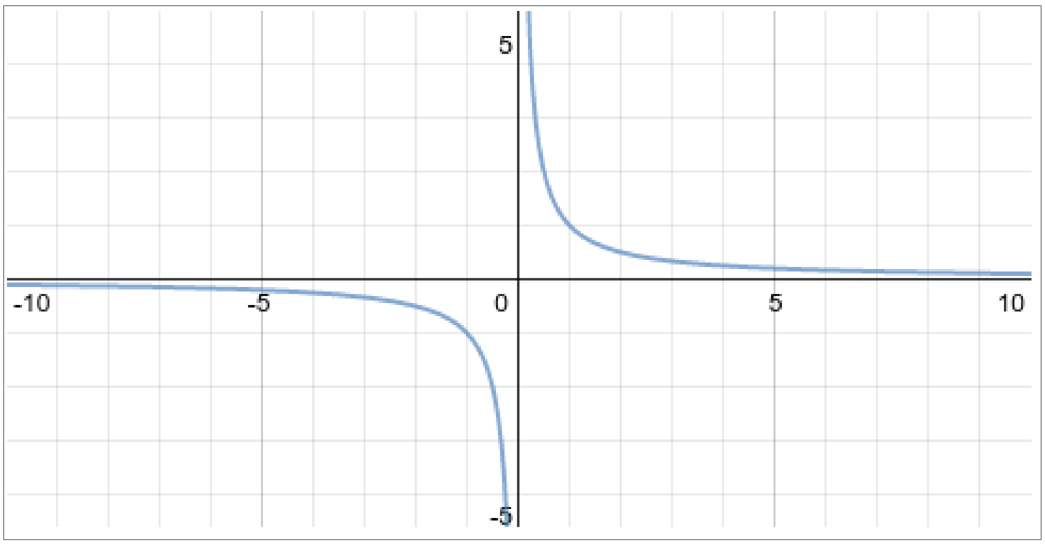
This set is countably infinite as both Q and Z are countably infinite sets, as a result the Cartesian product of these two sets also produces a set that is countably infinite.

Although the set *{(x, sin(πx)) x ∈ R}* in itself is uncountably infinite, it's intersection with the set {Q × Z} produces a set that is within the bounds of {Q × Z} which as discussed above is countably infinite.

(b) Is {(x, y) ∈ R | xy = 1} finite, countably infinite, or uncountably infinite? Justify your answer.

This set R itself is an uncountably infinite set as a result of the set of all irrational numbers being a subset of R.

The function *xy = 1* can be re-written in the form of *x = 1/y* or *y=1/x*. When these functions are graphed, the following graph is produced.



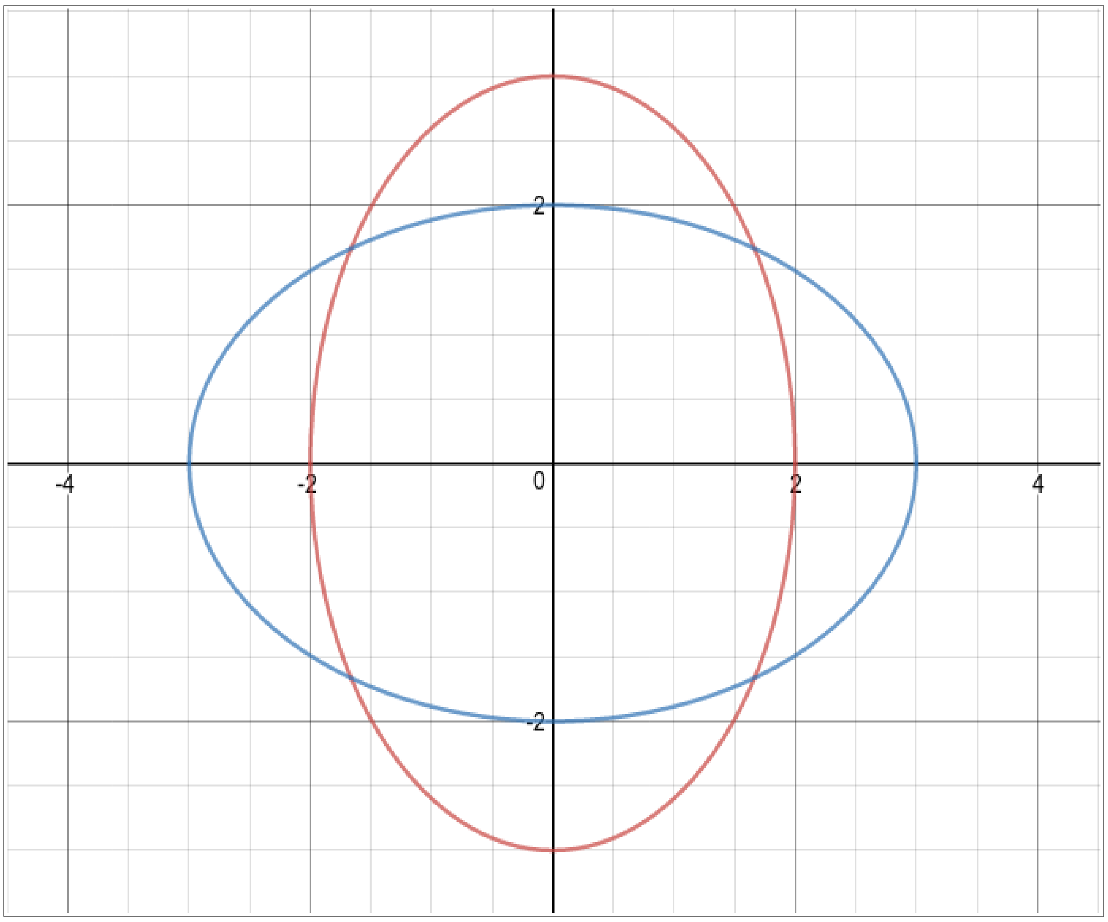
There are infinite amounts of x or y's as defined under the Real Numbers Set (R) that satisfy the equations above. However, as can be seen by the graph this is an uncountably infinite set as the graph tends towards both axes thus yielding an uncountably infinite set.

(c) c) Is {(x, y) ∈ R 2 | x 2 /4 + y 2 /9 = 1} ∩ {(x, y) ∈ R 2 | x 2 /9 + y 2 /4 = 1} finite,countably infinite, or uncountably infinite? Justify your answer.

The set R 2 itself is uncountably infinite. Since, as discussed above, the set R is uncountably infinite this yields that the Cartesian product R 2 is also uncountably infinite.

However, the set {(x, y) ∈ R 2 | x 2 /4 + y 2 /9 = 1} and the set {(x, y) ∈ R 2 | x 2 /9+ y 2 /4 = 1} are the sets containing the co-ordinates of two ellipses, the first of which being with a vertical major axis and the second of which being with a horizontal major axis.

The intersection of these two sets is namely the set containing the co-ordinates of the intersections of the two ellipses which is a finite set containing four elements. This can be illustrated in the following graph.



(c) Let A = {0, 1}. Is (0 ∗ ◦ 1 ∗) ∩ {A ∗ ◦ 11 ◦ A ∗ } finite, countably infinite,or uncountably infinite? Justify your answer.

(0* ◦ 1*) describes a word containing an infinite amount of 0's concatenated with an infinite amount of 1's e.g 0001111. This set itself is countably infinite as it is a language given under the regular expression (0* ◦ 1*) and the set of all regular languages over a finite alphabet is countably infinite.

{A*} is the set of all words over A. Since A is a finite alphabet, the set {A*} is a countably infinite set.

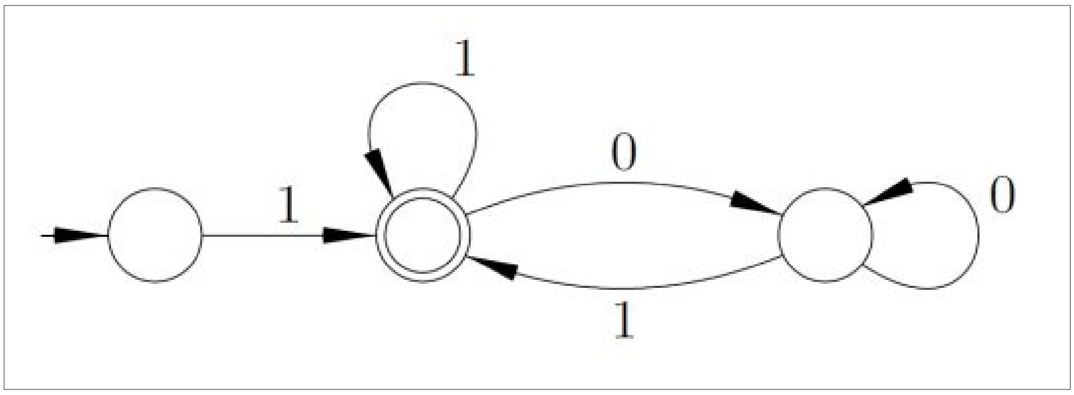
{A ∗ ◦ 11 ◦ A ∗ } is also a countably infinite set, similar to what was discussed above it is a language defined under a regular expression and a regular language over a finite alphabet is countably infinite.

As a result the intersection of (0* ◦ 1*) and {A ∗ ◦ 11 ◦ A ∗ } yields a set that is also countably infinite since the intersection of two countably infinite sets is itself a countably infinite set.

Question 2

a) Let L be the language consisting of the binary representations of all odd natural numbers. Write down the algorithm of a Turing machine that recognizes L. Process the following strings according to your algorithm: empty string, 0, 11, and 100.

Binary representations of odd natural numbers can be described under the following expression $1 \cup 1(0 \cup 1) * 1$. An example of a finite state acceptor that accepts binary representations of odd natural numbers can be found below 1 . This representation does not account for leading zeros.



This can be described in the form of a Turing Machine M which performs the following algorithm:

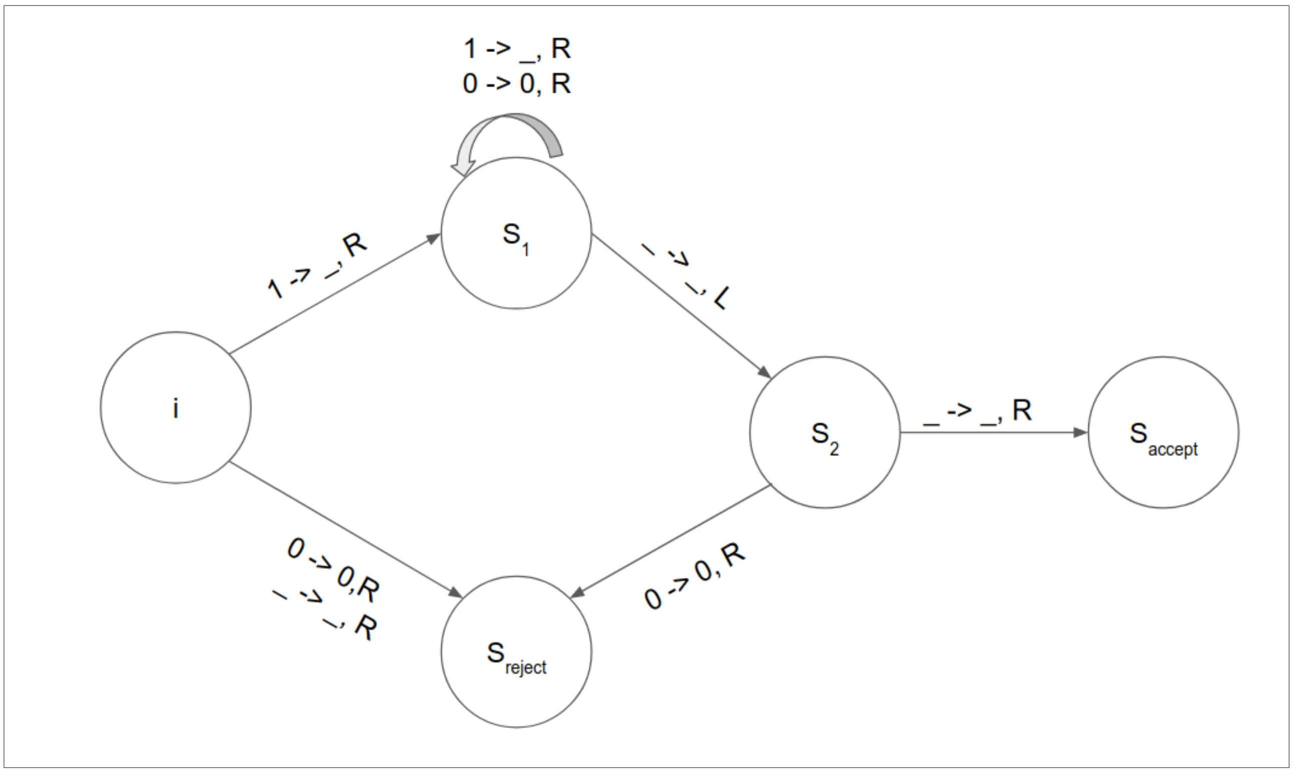
1. If anything other than 1 is in the first cell REJECT.
2. Move to the end of the string (the next _ character), then move left.
3. If 1 is in the current cell ACCEPT. Else REJECT.

Processing Empty Word	Processing '0'	Processing '11'	Processing '100'
[$\overset{\downarrow}{\_}$] REJECT - Step 1	[$\overset{\downarrow}{0}$] REJECT - Step 1	[$\overset{\downarrow}{1}$][1][_] Step 1	[$\overset{\downarrow}{1}$][0][0][_] Step 1
		[1][1][$\overset{\downarrow}{\_}$] Step 2	[1][0][0][$\overset{\downarrow}{\_}$] Step 2
		[1][$\overset{\downarrow}{1}$][_] Step 2	[1][0][$\overset{\downarrow}{0}$][_] Step 2
		[1][$\overset{\downarrow}{1}$][_] ACCEPT Step 3	[1][0][$\overset{\downarrow}{0}$][_] REJECT Step 3

(b) Write down the transition diagram of the Turing machine from part (a) carefully labelling the initial state, the accept state, the reject state, and all the transitions specified in your algorithm

The Turing Machine above can be described under the 7-tuple (S, A, Ã, t, i, S_{accept}, S_{reject}):

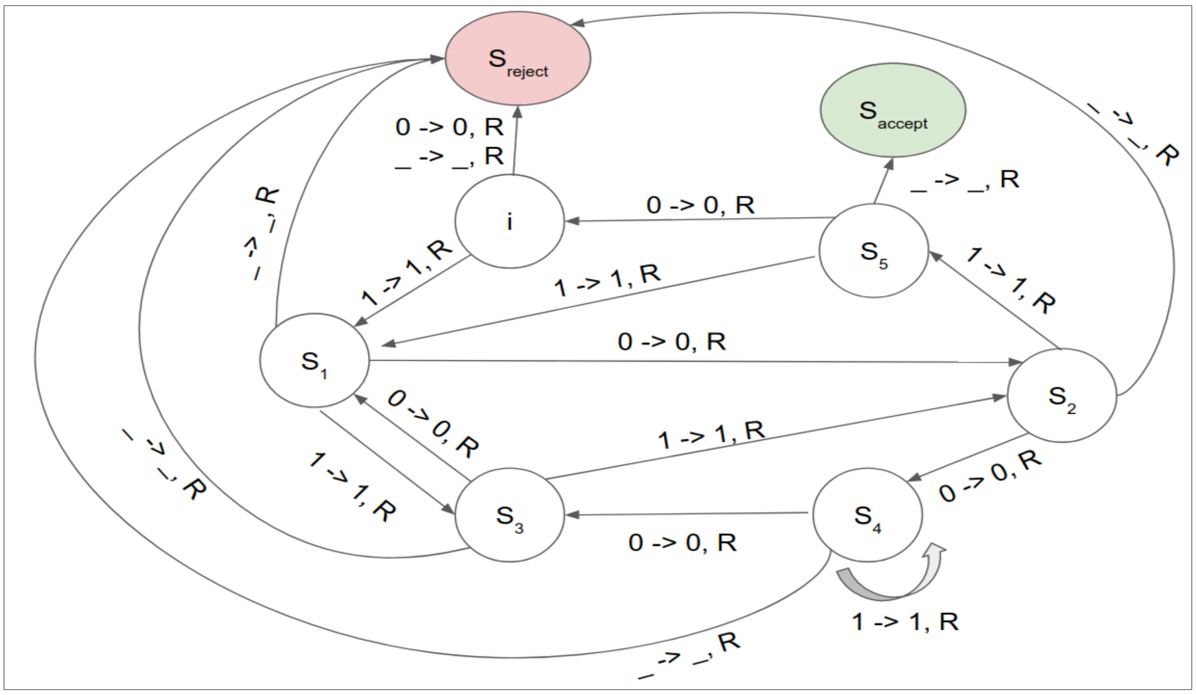
S ={S_1, S_2}
A = {0, 1}
Ã = { _ }
t = t : S x Ã ⟶ S x Ã x {L, R}
I = S_i
S_{accept} = S_{accept}
S_{reject} = S_{reject}



(b) Write down the algorithm of an enumerator that prints out EXACTLY ONCE every odd natural number divisible by 5. (Hint: Order the words on the input tape.)

-The Turing Machine as described above needs to be somewhat changed as more states needed to be added. The new and updated Turing Machine M is described under the following model:

S ={S_1, S_2, S_3, S_4, S_5}
A = {0, 1}
Ã = { _ }
t = t : S x Ã ⟶ S x Ã x {L, R}
I = S_i
S_{accept} = S_{accept}
S_{reject} = S_{reject}



Algorithm of Enumerator:

Let L be the language L = $1 \cup 1(0 \cup 1)*1$. Let M be the above Turing Machine that recognizes all words under L that are binary representations of odd numbers divisible by 5. We construct an enumerator E that outputs these words.

Let A be the alphabet of L{0, 1}. A* has an enumeration as a sequence A* = {w_1, w_2, … }.

1. Repeat the following for k = 1, 2, 3, ...
2. Run M, passing it the input word w_k
3. If any computations accept, print out the corresponding w_k.

Every string accepted by M (every binary representation of every odd natural number) will appear on the list of E.