Discrete Mathematics - Theorems

Theorem 1

If A has n elements, P(A) has 2^n elements.

Theorem 2

For any equivalence relation R on a set A, its equivalence class forms a partition on the set A. That is:

1. Every element of A has an equivalence class: $\forall x \in A, \exists y \in A \ s.t. \ x \in [y]$
2. All elements related by R belong to the same equivalence class: $xRy \Leftrightarrow [x] = [y]$
3. If two elements are not related by R, they belong to disjoint equivalence classes: $\neg(xRy) \Leftrightarrow [x] \cap [y]$

Theorem 3

Let f: A->B be a function. f^{-1} exists ⇔ f: A->B is bijective.

Proposition 4

Let A, B be sets and let f: A->B be a function. Assume A is finite. Then,

f is injective ⇔ |f(A)| = |A|

(where | | means cardinality)

Corollary 4.1

Let A, B be finite sets such that #(A) = #(B). Let f: A->B be a function. Then,

f is injective ⇔ f is bijective.

Corollary 4.2 - The Pigeonhole Principle

Let A, B be finite sets, and let f: A-> B be a function. If #(B) < #(A), then

$\exists a, a' \in A \; with \; a \neq a' \; s.t. f(a) = f(a')$

Proposition 5

Set A is finite ⇔ ∀ f: A-> A, an injective function is also bijective.

(Relies on corollary 4.1)

Theorem 6

Let (A, *) be a semigroup.

$\forall a \in A, a^m * a^n = a^{m+n}, \forall m, n \in N^*$

Induction

Theorem 7

Let (A, *) be a semigroup.

$\forall a \in A, (a^m)^n = a^{mn}, \forall m, n \in N^*$

Induction

Theorem 8

A binary operation on a set cannot have >1 identity element.

That is, if ∃ identity element, it is unique.

Proof by contradiction.

Theorem 9

Let (A, *) be a monoid and let a ∈A.

$a^m * a^n = a^{m+n}, \forall m, n \in N$

(Relies on Theorem 6, just prove for 0)

Theorem 10

Let (A, *) be a monoid and let a ∈A.

$(a^m)^n = a^{mn}, \forall m, n \in N$

(Relies on Theorem 7, just prove for 0)

Theorem 11

Let (A,*) be a monoid. If a ∈ A has an inverse, then that inverse is unique.

Proof by contradiction.

Theorem 12

Let (A, *) be a monoid. Let a, b be invertible elements of A.

Then, a*b is also invertible and $(a*b)^{-1} = b^{-1} * a^{-1}$

Theorem 13

Let (A, *) be a monoid and let a ∈A be invertible. Then,

$a^m * a^n = a^{m+n}, \forall m, n$

(Relies on Theorem 9, then proof by cases)

Theorem 14

Let (A, *) be a monoid and let a ∈A be invertible. Then,

$(a^m)^n = a^{mn}, \forall m, n \in Z$

(Relies on Theorem 10, then proof by cases)

Theorem 15

Let (A, *) and (B, *) both be semigroups, monoids or groups.

The inverse ($f^{-1} : B \to A$) of any isomorphism($f : A \to B$) is also an isomorphism.

(Relies on theorem 3, then shows that it is homomorphic.)

Theorem 16

Let A be a finite set. (A*,○) is a monoid with identity element ε.

Theorem 17

Let A be a finite set.

a. If L_1 and L_2 are languages over A,
   - $L_1 \cup L_2$ is a language over A - (closed under union)
   - $L_1 \cap L_2$ is a language over A. - (closed under intersection)
   - The concatenation of L_1 and L_2 given by - (closed under concatenation)
     - $L_1 \circ L_2 = \{w_1 \circ w_2 \in A \mid w_1 \in L_1, w_2 \in L_2\}$ is a language over A.

b. Let L be a language over A. Define $L^1=L$. Inductively, for any n >= 1,
   - $L^n = L \circ L^{n-1}$ is also a language over A.
   - $L^* = \{\varepsilon\} \cup L^1 \cup L^2 \cup L^3 \cup \ldots$ = sum of all L^ns for n->∞ is a language over A (closed under Kleene star)

Theorem 18

The collection of regular languages C is also closed under:

a. Intersection (∩)
b. Complement (/)

(Proof not in notes)

Theorem 19

A language L over some alphabet A is a regular language

⇔ L is recognised by a deterministic FSA with input alphabet A

⇔ L is recognised by a non-deterministic FSA with input alphabet A

Theorem 20 - Myhill-Nerode Theorem

Let L be a language over alphabet A.

If set L/≡ of equivalence classes in L is infinite, then L is NOT a regular language.

(where if x ≡y then x and y place the FSA into the same state)

(Notes only give sketch of the proof)

Lemma (no number cause I messed up)

Any language generated by a regular grammar may be generated by a regular grammar in normal form.

Theorem 21

A language L is regular

⇔ L is recognised by a FSA

⇔ L is generated by a regular grammar.

(Builds off of theorem 19)

Theorem 22

A language L is regular

⇔ some regular expression describes it.

Theorem 23

A language L is regular

⇔ L is recognised by a FSA

⇔ L is generated by a regular grammar

⇔ L is given by a regular expression,

(Builds off theorem 21 and 22)

The Pumping Lemma

If L is a regular language, then there is a number p (the pumping length) where if w is any word in L of length at least p, then w=xuy for words x, u, y satisfying

i. $u \neq \varepsilon$ ($|u| > 0$)
ii. $|xu| <= p$
iii. $xu^n y \in L$, $\forall n >= 0$

(complex but apparently natural consequence of pigeonhole principle)

Theorem 24

Let (V,E) be a graph. Then,

$$\sum_v degv = 2 \times \#(E)$$

Where $\sum_v degv$ is the sum of the degrees of all the vertices of the graph and #(E) is the number of edges in the graph.

Corollary 24.1

$\sum_v degv$ is an even integer.

Corollary 24.2

In any graph, the number of vertices of odd degree must be even.

(Proof by contradiction)

Corollary 24.3

Let (V,E) be a k-regular graph.

Then,

k*#(V) = 2*#(E)

Corollary 24.4

A complete bipartite graph $K_{p,q}$ is regular

⇔ p=q

Theorem 25

Let (V,E) be an undirected graph and let u,v ∈ V.

∃ path between u and v in the graph

⇔ ∃ walk in the graph between u and v

Corollary 25.1

An undirected graph (V,E) is connected

$\Leftrightarrow \forall u, v \in V, \exists$ a walk in the graph between u and v.

Lemma for splitting an undirected graph into components

Let (V,E) be an undirected graph. We define a relation ~ on the set of vertices V where a,b $\in$ V satisfy a~b IFF $\exists$ walk in the graph from a to b.

~ is an equivalence relation.

Lemma 26

The vertices and edges of any walk in an undirected graph are all contained in a single component of that graph.

Lemma 27

Each component of an undirected graph is connected.

Theorem 28

If (V,E) has no isolated or pendant vertices, then (V,E) contains at least 1 simple circuit.

(Proof by maximality)

Theorem 29

Let (V,E) be an undirected graph, and let u,v $\in$ V be vertices s.t e!=v and $\exists$ at least 2 distinct paths in (V,E) from u to v.

Then, the graph contains at least 1 simple circuit.

Theorem 30

Let (V,E) be a graph and let $v_0v_1v_2\cdots v_n$ be a trail in (V,E).

Let $v \in V$ be a vertex, then the number of edges of the trail incident to v is even, except when the trail is not closed and the trail starts or finishes at v, in which case the number of edges of the trail incident to v is odd.

Corollary 30.1

Let v be a vertex of the graph.

Given any circuit in the graph, the number of edges incident to v traversed by that circuit is even.

Corollary 30.2

If a graph admits an Eulerian circuit, then the degree of every vertex of that graph must be even.

(Uses corollary 30.1)

Corollary 30.3

If a graph admits an Eulerian trail that is not a circuit, then the degrees of exactly 2 vertices of the graph must be odd, and the degrees of the remaining vertices must be even. The vertices with odd degrees are exactly the initial and final vertices of the Eulerian trail.

Theorem 31 - Euler's Theorem

A non-trivial connected graph contains an Eulerian circuit if the degree of every vertex of the graph is even.

(Relies on lemmas 31A, 31B, 31C, 31D)

Lemma 31A

Let vw be the edge of a graph in which the degree of every vertex is even.

Then, ∃ circuit of the graph that traverses vw.

(Relies on theorem 30)

Lemma 31B

Let (V,E) be a connected graph such that $\forall v \in V$, degree of v is even.

Let some circuit $v_0v_1v_2 \cdots v_mv_0$ be given.

Suppose for some i (0 <= i <= m-1), some, but not all edges of the graph which are incident to v_i are traversed by $v_0v_1v_2 \cdots v_mv_0$.

Then, ∃ another circuit in (V,E) passing through v_i that does not traverse any edge traversed by $v_0v_1v_2 \cdots v_mv_0$.

(Relies on corollary 30.1 and lemma 31A)

Lemma 31C

Suppose a graph contains a circuit of length m and a circuit of length n.

Suppose that no edge of the graph is traversed by both circuits, and at least one vertex of the graph is common to both circuits.

Then, the graph contains a circuit of length m+n.

Lemma 31D

Let (V,E) be a connected graph and let some trail in the graph be given.

Suppose that no vertex of the graph has the property that not all of the edges of the graph incident to that vertex are traversed by the trail.

Then, the given trail is a Eulerian trail.

(Proof by contradiction)

Theorem 32

A nontrivial connected graph has a Eulerian circuit

⇔ the degree of each of its vertices is even.

(Relies on corollary 30.2 and theorem 31)

Corollary 32.1

Supposed that a connected graph has exactly 2 vertices whose degree is odd.

∃ an Eulerian trail in the graph joining the two vertices of odd degrees.

Theorem 33

Every complete graph K_n for n >= 3 has a Hamiltonian circuit.

Theorem 34

Every forest contains at least one pendant vertex.

(Relies on theorem 28)

Theorem 35

A non-trivial tree contains at least one pendant vertex.

(Proof by contradiction AND theorem 34)

Theorem 36

Let (V, E) be a tree.

Then,

#(E) = #(V) - 1

(Proof by induction)

Theorem 37

Let (V,E) be a tree.

$\forall v, w \in V \ s.t. \ v \neq w$, ∃! path in (V,E) from v to w.

(Partial proof by contradiction)

Theorem 38

Every connected graph contains a spanning tree.

(Partial proof by contradiction)

Corollary 38.1

Let (V,E) be a connected graph with #(V) vertices and #(E) edges.

If #(E) = #(V) -1,

Then (V,E) is a tree.

Proposition 39

Let (V,E) be a connected graph with the associated cost function $c : E \to \mathbf{R}$.

Kruskal's algorithm yields a spanning tree of (V,E).

Proposition 40

Let (V,E) be a connected graph with the associated cost function $c : E \to \mathbf{R}$.

Kruskal's algorithm yields a minimal spanning tree of (V,E).

(relies on proposition 39)

Theorem 41

Every infinite subset of a countably infinite set is itself countably infinite.

Theorem 42

Let $\{A_n\}_{n=1,2,\cdots}$ be a sequence of countably infinite sets.

Let $\mathcal{S} = \bigcup_{n=1}^{\infty} A_n$.

Then, $\mathcal{S}$ is countably infinite.

(Snaking method to get enumeration)

Corollary 42.1

Suppose an indexing set $\mathbf{I}$ is countable and $\forall i \in \mathbf{I},\ A_i$ is countable.

Then,

$T = \bigcup_{i \in \mathbf{I}} A_i$ is countable.

Corollary 42.2

Let A be a countably infinite set.

Let $A^n = \underbrace{A \times A \times \cdot \times A}_{n\ times}$.

Then, A^n is countably infinite.

(Induction)

Corollary 42.3

$\mathbb{N}^n$ is countably infinite $\forall n \geq 1$.

(Relies on corollary 42.2)

Corollary 42.4

$\mathbb{Z}^n$ is countably infinite $\forall n \geq 1$

(Relies on corollary 42.2)

Corollary 42.5

$\mathbb{Q}$ is countably infinite.

(Proof)

$$\mathbb{Q} = \{\frac{p}{q} | q \neq 0,\ p, q \in \mathbb{Z},\ \underbrace{(p,q) = 1}_{no\ common\ factors}\ \}$$

We can also represent $\mathbb{Q}$ as $\{(p,q) | q \neq 0,\ p, q \in \mathbb{Z}\} / \sim \subseteq \mathbb{Z}^2$,

Where $(p_1, q_1) \sim (p_2, q_2) \Leftrightarrow \frac{p_1}{q_1} = \frac{p_2}{q_2} \Leftrightarrow p_1 q_2 = p_2 q_1$ by cross-multiplication.

We also know $\mathbb{Z} \subseteq \mathbb{Q}$ (when q = 1).

Therefore, $\mathbb{Q}$ is sandwiched between $\mathbb{Z} = \mathbb{Z}^1$ and $\mathbb{Z}^2$, both of which are countably infinite

$\Rightarrow$ $\mathbb{Q}$ is countably infinite.

Theorem 43

Let A be the set of all sequences $s = \{x_1, x_2, \cdots\} = \{x_n\}_{n=1,2,3,\cdots}$ such that $x_n \in \{0, 1\} \forall n \geq 1$.

Then A is uncountably infinite.

(Proof by contradiction)

Assume A is countably infinite

$\Leftrightarrow A = \{s_1, s_2, \cdots\}\ where\ s_j = \{\ x_n^j\}_{n=1,2,\cdots}\ for\ x_n^j = 0\ or\ x_n^j = 1$

We now construct a sequence s_0 of 0s and 1s which cannot be in the enumeration.

Let s_0 be such that:

$$x_j^0 = \begin{cases} 1, & if\ x_j^j = 0 \\ 0, & if\ x_j^j = 1 \end{cases}$$

That is - s_0 differs from each s_j in the j^{th} element.

$\Rightarrow$ $s_0 \notin \{s_1, s_2, \cdots\}$, but s_0 is a sequence of 0s and 1 $\Rightarrow$ $s_0 \in A$ $\Rightarrow\Leftarrow$

(q.e.d.)

Corollary 43.1

Power set $\mathcal{P}(\mathbb{N})$ of $\mathbb{N}$ is uncountably infinite.

(Proof)

$\mathbb{N} \sim J$, so we can write $\mathbb{N} = \{x_1, x_2, x_3, \cdots\}$. For each subset of $\mathbb{N}$, for each i, we can include x_i or not include it. We can represent including it as 1 and not including it as a 0 (similar to how it's done in theorem 1). Then, each subset of $\mathbb{N}$ can be represented as a unique string of 0s and 1s. There is then a one-to-one correspondence between all the subsets of $\mathbb{N}$ and sequences of 0s and 1s. Therefore, $\mathcal{P}(\mathbb{N}) \sim A$, where A is the set of all sequences of 0s and 1s.

We showed in theorem 43 that A is uncountably infinite.

$\Rightarrow$ $\mathcal{P}(\mathbb{N})$ is uncountably infinite.

Theorem 44

$\mathbb{R}$ is uncountably infinite.

(Relies on propositions 44.1 and 44.2)

Proposition 44.1

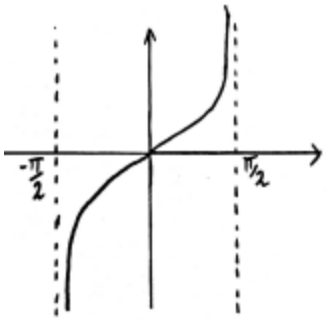
$\mathbb{R}$ is in bijective correspondence with the interval (0,1).

(Proof)

We want bijection $f : \mathbb{R} \to (0, 1)$.

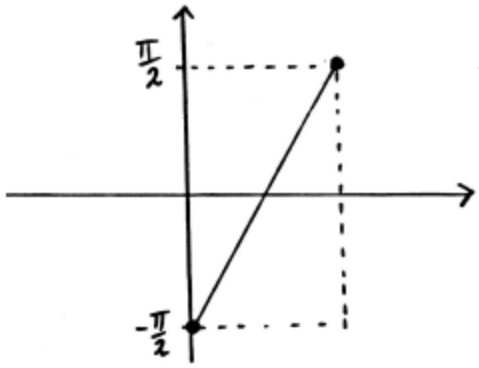
Recall from trigonometry that

$\tan : (-\frac{\pi}{2}, \frac{\pi}{2}) \to \mathbb{R}$ is a bijection. tan:



We now show $(0, 1) \sim (-\frac{\pi}{2}, \frac{\pi}{2})$ by a linear function which is a bijection.

Let $g(x) = \pi x - \frac{\pi}{2}$. g(x):



As the composition of 2 bijections is itself a bijection

$\Rightarrow$ tan($g(x)$) = tan($\pi x - \frac{\pi}{2}$) is a bijection from (0,1) to $\mathbb{R}$.

We find its inverse to get the map we want.

$f : \mathbb{R} \to (0, 1)$

$f(x) = (tan(\pi x - \frac{\pi}{2}))^{-1}$

This is also a bijection as the inverse of a bijection is a bijection. So, we have a bijective correspondence from $\mathbb{R}$ to (0,1).

Proposition 44.2

(0,1) is uncountably infinite

(Proof)

We want to give the binary expansion of each $x \in$ (0,1) - we want to associate to each x a $0.x_1x_2x_3\cdots$ where after the radix $\{x_1, x_2, x_3, \cdots\}$ is a sequence of 0s and 1s.

So, in general,

$$0.x_1x_2x_3\cdots = 0 + \frac{1}{2}x_1 + \frac{1}{4}x_2 + \frac{1}{8}x_3 + \cdots = 0 + \sum_{n=1}^{\infty} \frac{1}{2^n}x_n$$

Recall that $\sum_{n=1}^{\infty} \frac{1}{2^n} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 1$

Therefore, we can simplify our INCOMPLETE

(Relies on proposition 44.1)

Theorem 45

If A is a finite alphabet, then the set of all words over A (A^*) is countably infinite.

(Relies on corollary 42.1)

Corollary 45.1

If A is a finite alphabet, then the set of all languages over A is uncountably infinite.

(also relies on corollary 43.1)

Corollary 45.2

The set of all programs in any programming language is countably infinite.

Corollary 45.3

(Completed after theorem 61)

Given a finite alphabet A, the set of all Turing recognisable languages over A is countably infinite.

Theorem 46

The set of all regular languages over a finite alphabet is countably infinite.

(Relies on theorem 22 and theorem 45)

Theorem 47

Every multitype Turing machine has an equivalent single tape Turing machine.

Corollary 47.1

A language L is Turing-recognisable

⇔ some multitype Turing machine recognises L.

Theorem 48

Every non-deterministic Turing machine has an equivalent deterministic Turing machine.

Corollary 48.1

A language L is Turing-recognisable

⇔ some non-deterministic Turing machine recognises L.

Theorem 49

A language L is Turing-recognisable

⇔ some enumerator enumerates (outputs) L.

Theorem 50 - Acceptance Problem for DFA

To test whether a given DFA, B, accepts a given string.

Written as a language:

$$L_{DFA} = \{<B, w> \mid B \text{ is a DFA that accepts input string } w\}$$

L_{DFA} is a Turing decidable language.

Theorem 51 - Acceptance Problem for NFA

To test whether a given NFA, B, accepts a given string.

Written as a language:

$$L_{NFA} = \{<B, w> \mid B \text{ is a NFA that accepts input string } w\}$$

L_{NFA} is a Turing decidable language.

(This is just a corollary of theorem 50 thanks to theorem 19)

Theorem 52 - Acceptance Problem for regular expressions

To test whether a given regular expression, R, generates a string w.

Written as a language:

$$L_{REX} = \{<B, w> \mid B \text{ is a regular expression that generates string } w\}$$

L_{REX} is Turing decidable.

(Relies on theorem 51 as there is an algorithm to turn any regular expression into an NFA)

Theorem 53 - Emptiness testing for language of FSA

Given DFA B, test if L(B) is empty.

Written as a language:

E_{DFA} = {<B> | B is a DFA and L(B) = $\varnothing$ }

E_{DFA} is Turing decidable.

Theorem 54 - Equality of DFAs' languages

Given DFAs B_1, B_2, test if L(B_1) = L(B_2).

Written as a language:

EQ_{DFA} = {< B_1, B_2 > | B_1, B_2 are DFAs and L(B_1) = L(B_2) }

EQ_{DFA} is Turing decidable.

(Symmetric difference and theorem 53)

Theorem 55 - Acceptance Problem for CFG

To test whether a given CFG , G, generates string w.

Written as a language:

L_{CFG} = {<G,w> | G is a context free grammar and w is a string }

L_{CFG} is Turing decidable.

(Full proof not given, uses Chomsky normal form)

Theorem 56 - Emptiness testing for CFG

Given CFG G, test if L(G) is empty.

Written as a language:

E_{CFG} = {<G> | G is a CFG and L(G) = $\varnothing$}

E_{CFG} is Turing decidable.

Proposition 57 - Equivalence problem for CFGs

Given 2 CFGs G_1, G_2, determine whether they generate the same language - $L(G_1) = L(G_2)$

Written as a language:

EQ_{CFG} = {< G_1, G_2 > | G_1, G_2 are CFGs and $L(G_1) = L(G_2)$}

EQ_{CFG} is NOT a Turing decidable language.

(Proven by reducibility)

Proposition 58 - Emptiness testing for Turing machines

Given Turing machine M, test if L(M) is empty.

Written as a language:

E_{TM} = {<M> | M is a Turing machine and L(M) = $\varnothing$}

E_{TM} is NOT Turing decidable.

Proposition 59 - Equivalence testing for Turing machines

EQ_{TM} is NOT Turing decidable.

(relies on proposition 58)

Theorem 60

Let

$REGULAR_{TM}$ = {<M> | M is a Turing machine and L(M) is a recognisable language }

$REGULAR_{TM}$ is NOT Turing decidable.

(Proven using reducibility)

Theorem 61 - Rice's Theorem

Any property of the languages recognised by Turing machines is NOT Turing decidable.

Proposition 62

Let A be a finite alphabet.

Not all languages over A are Turing-recognisable.

(Relies on corollaries 45.1 and 45.3)

Proposition 63

Let

L_{TM} = {<M,w> | M is a Turing machine and M accepts w }

L_{TM} is Turing-recognisable.

Proposition 64

Let

$$L_{TM} = \{<M,w> \mid M \text{ is a Turing machine and } M \text{ accepts } w\}$$

L_{TM} is NOT Turing-decidable.

(Proof by contradiction)

Theorem 65

A language L is Turing-decidable

⇔ L is Turing-recognisable AND co-Turing recognisable.

Corollary 65.1

$\overline{L_{TM}}$ (complement of L) is not Turing-recognisable.

(Relies on proposition 64 and 65)

Discrete Mathematics - Definitions

Here only ones I forget are included

1. Relation between A and B

A subset of cartesian product AxB.

2. Relation on A

A subset of cartesian product AxA.

3. Binary relation

A relation involving only 2 elements.

4. Reflexive relation

A relation is reflexive $\Leftrightarrow \forall x \in A, xRx$

5. Symmetric relation

A relation is symmetric $\Leftrightarrow \forall x, y \in A, xRy \rightarrow yRx$

6. Transitive relation

A relation is transitive $\Leftrightarrow \forall x, y, z \in A, xRy \wedge yRz \rightarrow xRz$

7. Equivalence relation

An equivalence relation is reflexive, symmetric and transitive.

8. Partition

Let A be a set. A partition of A is a collection of non-empty sets any two of which are disjoint such that their union is A.

$\lambda = \{A_\alpha | \alpha \in I\} s.t. \forall \alpha, \alpha' \in I \; satisfying \alpha \neq \alpha', A_\alpha \cap A_{\alpha'} = \varnothing \; and \; the \; union \; of \; all \; A_\alpha s = A$

9. Antisymmetric relation

A relation is antisymmetric $\Leftrightarrow \forall x, y \in A \; s.t. \; xRy \wedge yRx \rightarrow x = y$

10. Partial order

A relation on set A is a partial order if it is reflexive, antisymmetric, and transitive.

11. Injective

A function f: A->B is injective if:

f(x) = f(y) => x=y.

Each input has exactly 1 unique output.

Use horizontal line test.

12. Surjective

A function f: A->B is surjective if:

$\forall$ z$\in$B, $\exists$ x$\in$A s.t. f(x)=z

If range of f is all of B.

13. Bijective

A function f is bijective if it is both injective and surjective.

14. Binary operation

* is a binary operation on A if $\forall$ x,y $\in$A, x*y$\in$A

15.Commutative

Binary operation * on set A for which

$\forall$ x,y$\in$A, x*y = y*x

16. Associative

Binary operation * on set A for which

$\forall x,y,z \in A, (x*y)*z = x*(y*z)$

17. Semigroup

A set endowed with an associative binary operation.

(eg: (M , *), set of nxn matrices with entries in R with matrix multiplication as operation)

18. Identity element

Let (A, *) be a semigroup. Element $e \in A$ is an identity element for binary operation * if

$e*x = x*e = x, \forall x \in A$

Let $a \in A$. Then we define

$a^0 = e$, identity element

19. Monoid

A semigroup (A, *) where * has the identity element e.

(eg: (M , *), set of nxn matrices with entries in R with matrix multiplication as operation, with e = I)

20. Abelian

A monoid (A, *)/group (A, *, e) is Abelian (commutative) if * is commutative.

(eg: (N, +))

21. Inverse

Let (A,*) be a monoid with identity element e, and let $x \in A$.

Element y of A is inverse of x if:

$x*y = y*x = e$

If $x \in A$ has an inverse, it is invertible.

22. Group

A monoid in which every element is invertible. (A, *, e)

Overall, set A endowed with binary operation * satisfying:

- * is associative.
- There exists an identity element e of A.
- Every element in A is invertible.

23. Homomorphism

Let (A, *) and (B, *) be semigroups, monoids, or groups.

Function f: A->B is homomorphism if

$$f(x*y) = f(x) * f(y) \quad \forall x,y \in A$$

(behaves well w.r.t. the binary operation)

24. Isomorphism

Let (A, *) and (B, *) be semigroups, monoids, or groups.

Function f: A->B is an isomorphism if it is both a bijection and a homomorphism.

25. Word

$\forall n \in \mathbf{N}^*$, word of length n in alphabet A as any string of the form:

26. A*

Let A^n be the set of all words of length n over alphabet A.

Let A^+ = union of all A^ns=$A^1 \cup A^2 \cup A^3$ - set of all words of positive length over alphabet A.

Let A^0={ ε } where ε is the empty word.

Then, $A^* = A^0 \cup A^+$

27. Language

Let A be a finite set. A language over A is a subset of A*.

28. Formal language

A language with a finite set of rules/algorithm that generates EXACTLY L.

29. Grammar

A grammar is a set of production rules for strings in a language generated given

- A: alphabet
- <s>: start symbol
- Set of production rules

30. Production rules

Set of production rules is a subset of the Cartesian product (V\A)xV*. (where V is the set of terminals and non-terminals.

31. Context free grammar

A context free grammar $(V, A, \langle s \rangle, P)$ consists of:

- Finite set V
- Subset A of V
- An element <s>: of V\A
- **Finite** subset P of Cartesian product (V\A)xV*

32. Context sensitive grammar

Where only certain replacements of<T> by w are allowed, which are governed by the syntax of language L.

33. Ambiguous

A grammar is ambiguous if it generates the same string in more than 1 way.

34. Directly yields

Let w' and w'' be words over alphabet V.

w' directly yields w'' if exists words u, v over alphabet V and exists production rule <T> -> w of the grammar such that

w' = u<T>v

w'' = uwv

Where either u/v can be the empty word.

Written as

w' => w''

35. Yields

Let w' and w'' be words over alphabet V.

w' yields w'' if exists either w' = w'' or exist words $w_0,w_1,...,w$ over alphabet V such that

w_0 = w'

w =w''

and

w_{i-1} => w_i for all 1<= i <= n

Denoted by

w' =*> w''

36. Context-free language

Let (V,A,<s>,P) be a context-free grammar.

The language generated by this grammar is the subset L of A* defined by

L={w∈A* | <s> =*> w}

This is called a context-free language.

37. Phrase structure grammar

A phrase structure grammar (V, A, <s>, P):

- Finite set V
- Subset A of V
- An element <s>: of V\A
- **Finite** subset P of Cartesian product (V*\A*)xV*

That is, a production rule in a phrase structure grammar r->w has a LHS which may contain 1+ non-terminals.

38. Language generated by a phrase structure grammar

Let (V,A,<s>,P) be a phrase structure grammar.

The language generated by this grammar is the subset L of A^* defined by

$$L=\{w \in A^* \mid <s> =*> w\}$$

39. Regular language

Let A be a finite set.

Let A^* be the set of words over alphabet A.

A subset L of A^* is called a regular language over alphabet A if

$L = L_i$ for some finite sequence $L_1, L_2, \ldots, L$ of subsets of A^* with the property that $\forall i, 1 <= i <= m, L_i$ satisfies one of the following:

i. L_i is a finite set.

ii. $L_i = L^*$ for some $1 <= j < i$ (Kleene star on one of previous L s)

iii. $L_i = L \circ L$ for some $j,k, 1 <= j,k < i$ (concat on two previous languages)

iv. $L_i = L \cup L$ for some $j,k, 1 <= j,k < i$

40. Finite State Acceptor (FSA)

An FSA (S, A, i, t, F) consists of

- Finite set S of states
- Finite set A (input alphabet)
- Starting state $i \in S$
- Transition mapping t: (SxA)-> S
- Set F of finishing states

41. Acceptance (word)

Let (S, A, i, t, F) be a FSA, and let A^* denote the words over alphabet A.

A word $a_1a_2\ldots a$ of length n over alphabet A is recognised or accepted if

$\exists$ set of states $s_0, s_1, \ldots, s \in A$ such that

- $s_0 = i$
- $s \in F$
- $s_i = t(s_{i-1}, a_i)$ for all $i, 1 <= i <= n$

42. Acceptance (language)

Let (S, A, i, t, F) be a FSA.

A language L over alphabet A is recognised or accepted by the FSA if L is the set consisting of all worls recognised by the FSA.

43. Deterministic FSA

Every state has exactly one transition for each input:

$$\forall(s,a) \in S \times A, \exists! t(s,a) \in S$$

(this mapping is a function)

44. Non-deterministic FSA

An input can lead to 1, >1, or <1 transitions for a given state.

Some (s,a)∈SxA might be associated with >= 1 element of S.

(this mapping is not a function).

45. Regular grammar

A context free grammar (V, A, <s>, P) where every production rule in P is in one of the following forms:

i. <A> -> b<B>
ii. <A> -> b
iii. <A> -> ε

where <A>, <B> are non-terminals, b is a terminal, and ε is the empty word.

A regular grammar is in **normal form** if all of its production rules are of type (i) and (iii).

46. Regular expression

Let A be an alphabet.

- ∅, ε , and all elements of A are regular expressions.
- If w and w' are regular expressions, then
  i. w○w'
  ii. w∪w'
  iii. w* (and w'*)

Are also regular expressions.

47. Undirected graph

Consists of

- A finite set of points V called vertices
- A finite set E of edges joining 2 distinct vertices of the graph.

48. Incident

If v is a vertex of some graph and e is an edge of that graph,

and if e = vv' for v'=another vertex, then

vertex v is incident to edge e, and edge e is incident to the vertex v.

49. Adjacent

Let (V,E) be an undirected graph.

Two vertices A, B ∈ V, A!=B are called adjacent if ∃ edge AB ∈ E.

50. Incidence matrix

Let (V,E) be an undirected graph with m vertices and n edges.

Let the vertices be ordered as

$$v_1, v_2, \cdots, v_m.$$

Let the edges be ordered as

$$e_1, e_2, \cdots, e_n$$

The incidence matrix for this graph is given by

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Where the entry a_{ij} in row i and column j has value 1 if the ith vertex is incident to the jth edge and has the value 0 otherwise.

51. Adjacency matrix

Let (V,E) be an undirected graph with m vertices, ordered as

$$v_1, v_2, \cdots, v_m.$$

The adjacency matrix for this graph is given by

$$\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \cdots & \cdots & & \cdots \\ b_{m1} & b_{m2} & \cdots & b_{mm} \end{pmatrix}$$

Where the entry b_{ij} = 1 if v_i and v_j are adjacent to each other, and 0 otherwise.

52. Complete graph

A graph (V,E) is called complete if

$$\forall v, v' \in V \ s.t. \ v \neq v', vv' \in E$$

(highest possible number of edges)

A complete graph with n vertices is denoted by K_n, and has $\frac{n(n-1)}{2}$ edges per graph.

53. Bipartite graphs

A graph (V,E) is bipartite if ∃ subsets V_1 and V_2 such that

i. $V_1 \cup V_2 = V$
ii. $V_1 \cap V_2 = \varnothing$
iii. Every edge in E is of the form vw with $v \in V_1$ and $w \in V_2$

Having a complete subgraph or a cycle means a graph cannot be bipartite.

54. Complete bipartite graphs

A bipartite graph is called complete if $\forall v \in V_1, \forall w \in V_2, \exists vw \in E$.

Denoted by K_{pq} where V_1 has p elements and V_2 has q elements.

55. Isomorphism of a graph

An isomorphism between two graphs (V,E) and (V',E') is a bijective function φ:V->V' satisfying that $\forall$ a,b $\in$ V with a != b,

The edge ab $\in$ E $\Leftrightarrow$ the edge ϕ (a) ϕ (b) $\in$ E'

(If there is an edge, that edge is mapped, and there is a one-to-one correspondence of vertices) (there is an inverse of this as it is a bijection)

56. Subgraph

Let (V,E) and (V',E') be graphs.

Graph (V',E') is a subgraph of (V,E) if

V' $\subseteq$ V AND

E' $\subseteq$ E

57. Vertex degrees

Let (V,E) be a graph. The degree of a vertex v $\in$ V is the number of edges of the graph which are incident to v.

58. Isolated vertex

Vertex of degree 0.

59. Pendant vertex

Vertex of degree 1.

60. k-regular

A graph is called k-regular for some non-negative integer k if every vertex of the graph has degree = k.

61. Walk

Let (V,E) be a graph.

A walk $v_0v_1v_2\cdots v_n$ of length n in the graph from vertex a to vertex b is determined by a finite sequence $v_0, v_1, v_2, \cdots, v_n$ of vertices of the graph such that

- v_0 = a
- v_n = b
- $v_{i-1}v_i$ is an edge of the graph for i = 1 ,2, 3, ...

62. Traverse

A walk $v_0v_1v_2\cdots v_n$ traverses edges $v_{i-1}v_i$.

63. Pass through

A walk $v_0v_1v_2\cdots v_n$ passes through vertices $v_0, v_1, v_2, \cdots, v_n$.

64. Trail

Let (V,E) be a graph. A trail $v_0v_1v_2\cdots v_n$ from some vertex a to some vertex b is a walk of length n from a to b with the property that edges $v_{i-1}v_i$ are distinct for i=1,2,...n.

(A trail is a walk on the graph which traverses edges of the graph at MOST once.)

65. Path

Let (V,E) be a graph. A path $v_0v_1v_2\cdots v_n$ from some vertex a to some vertex b is a walk of length n from a to b with the property that vertices $v_0, v_1, v_2, \cdots, v_n$ are distinct.

(A path is a walk in the graph which passes through each vertex in the graph at most once)

(All paths are trails)

66. Trivial

A walk, trail, or path that consists of a single vertex $v \in V$ and has length 0 is trivial.

Otherwise it is non-trivial.

67. Connected

An undirected graph (V,E) is called connected if $\forall u, v \in V$ vertices, $\exists$ path in the graph to get from u to v.

68. Components

Components are subgraphs (V_i, E_j) of (V,E) defined by the equivalence classes formed by ~.

69. Closed walk

Let (V,E) be a graph.

A walk $v_0 v_1 v_2 \cdots v_n$ in (V, E) is called closed if $v_0 = v_n$.

70. Circuit (cycle)

Let (V,E) be a graph.

A circuit is a closed trail in (V, E).

(A closed walk with no repeated edges, passing through at least two vertices)

71.Simple circuit (cycle)

A circuit is called simple if the vertices $v_0, v_1, \cdots, v_{n-1}$ are distinct.

72. Eulerian trail

Eulerian trail in a graph is a trail which traverses every edge of the graph.

(walk which traverses every (Eulerian) edge of the graph exactly once(trail))

73. Eulerian circuit

Eulerian circuit in a graph is the circuit that traverses every edge of the graph.

74. Hamiltonian path

A Hamiltonian path in a graph is a path that passes exactly once through every vertex of a graph.

75. Hamiltonian circuit

A Hamiltonian circuit in a graph is a simple circuit that passes through every vertex of the graph.

76. Acyclic

A graph is acyclic if it contains no circuits (cycles).

77. Forest

Acyclic graph.

78. Tree

Connected forest.

(Connected acyclic graph)

79. Spanning Trees

A spanning tree in a graph (V,E) is a subgraph of the graph (V,E) which is a tree and contains every vertex in V.

80. Cost function

Let (V, E) be an undirected graph.

A cost function:

$$c : E \to \mathbf{R}$$

on the set E of edges of the graph is a function that assigns to each edge e of the graph, a real number $c(e)$.

81. Cost on S

Let $c : E \to \mathbf{R}$ be a cost function on set E of edges of a graph (V,E).

Given any subset $\mathbf{S} \subset E$, the cost on S is:

$$c(\mathbf{S}) = \sum_{e \in \mathbf{S}} c(e)$$ = the sum of costs of all elements of $\mathbf{S}$.

82. Minimal spanning tree

Let (V,E) be a connected graph with cost function $c : E \to \mathbf{R}$.

A spanning tree (V, E_M) is called minimal w.r.t. the cost function if

$$\forall (V, E_T)\ spanning\ tree\ of\ (V, E),\ c(E_M) \leq c(E_T)$$

83. Kruskal tree

Let (V,E) be a connected graph with the associated cost function $c : E \to \mathbf{R}$.

Let (V, E') be the minimal spanning tree of (V,E) produced by Kruskal's algorithm.

(V, E') is then called the Kruskal tree.

84. Prim spanning tree

The minimal spanning tree yielded by Prim's algorithm is called the Prim spanning tree.

85. Visited vertices

Let (V_i, E_i) be the subgraph at the end of step i of Prim's algorithm.

All vertices in V_i are called visited vertices.

If (V,E) is the original graph, all vertices V \ V_i are called unvisited vertices.

86. Directed graph (digraph)

Digraph (V, E) consists of a finite set V together with a subset E of VxV. (NOT of V_2)

The elements of V are the vertices of the digraph, and the elements of E are the edges.

(Note - pairs in VxV are ordered, and loops are allowed)

Let (v,w) ∈ E be the edge of digraph (V,E).

Initial vertex

v is the initial vertex of the edge.

Terminal vertex

w is the terminal vertex of the edge.

Adjacent from/to

Vertex w is adjacent from vertex v.

Vertex v is adjacent to vertex w.

Incident from/to

Edge (v,w) is incident from vertex v.

Edge (v,w) is incident to vertex w.

87. J

J ={1,2,...,n}

88. A~J

A set A has n elements

⇔ ∃ bijection f: A -> J .

89. A~B

A~B ⇔ ∃ f: A -> B, which is a bijection and a relation on sets.

This is an equivalence relation.

90. [J]

[J] is an equivalence class for all sets A of size n.

91. Finite set

A set A is finite if

$$A \sim J_n \ for \ some \ n \in \mathbf{N}^*$$

OR

$$A = \varnothing$$

92. Infinite set

A set A is infinite if it is not finite.

93. J

J = $\mathbf{N}^*$ = {1, 2, 3, ...} = counting numbers

94. Countably infinite

Set A is countably infinite if A~J.

OR

A set A is countably infinite if its elements can be arranged in a sequence $\{x_1, x_2, \cdots\}$.

(Called countable in maths)

95. Uncountably infinite

Set A is uncountably infinite if A is neither finite nor countably infinite.

(Called uncountable in maths)

96. Countable

Set A is countable if A is finite or A is countably infinite.

(Called at most countable in maths)

97. Sequence

A set of elements $x_1, x_2, \cdots$ indexed by J.

$$\exists\ f : J \to \{x_1, x_2, \cdots\}\ s.t.\ f(n) = x_n\ \forall n \in J$$

98. [N]

Equivalence class of countably infinite sets.

$\mathbf{Z} \in [\mathbf{N}]$

99. [R]

Equivalence class of uncountably infinite sets.

100. Turing machine

7-tuple $(\mathcal{S}, A, \tilde{A}, i, t, S_{accept}, S_{reject})$

Where $\mathcal{S}, A, \tilde{A}$ are finite sets and

a. $\mathcal{S}$ is the set of states
b. A is the input alphabet, not containing $\sqcup$
c. $\tilde{A}$ is the tape alphabet, $\sqcup \in \tilde{A},\ A \subseteq \tilde{A}$
d. $t : \mathcal{S} \times \tilde{A} \to \mathcal{S} \times \tilde{A} \times \{L, R\}$ is the transition mapping
e. i is the initial state of the machine
f. $S_{accept} \in \mathcal{S}$ is the accept state
g. $S_{reject} \in \mathcal{S}$ is the reject state, and $S_{accept} \neq S_{reject}$

101. Configuration

Setting of the

1. State of the machine
2. Tape contents
3. Location of tape head

Represented by uS_iv, where u,v are strings in tape alphabet $\tilde{A}$ and S_i is the current state. The tape contents are uv, and the tape head is located on the first symbol of v.

Initial configuration

(with input u)

εiu - machine is in initial state i with head at leftmost position on the tape.

Accepting configuration

$uS_{accept}v \ \ for \ u, v \in \tilde{A}^*$

Rejecting configuration

$uS_{reject}v \ \ for \ u, v \in \tilde{A}^*$

Halting configuration

Yields no further configurations - there are no transitions out of their states. Includes accepting and rejecting configurations.

102. Yields

Let C_1, C_2 be two configurations of a given Turing machine. C_1 yields C_2 if the Turing machine goes from C_1 to C_2 in one step.

103. Accepts (Turing machine)

A Turing machine M accepts input $w \in A^*$ if

$\exists$ sequences of configurations $c_1, c_2, \cdots$, such that

1. C_1 is the start configuration with input w ($C_1 = \varepsilon w$).
2. Each C_i yields C_{i+1} for i=1,2,..., k-1.
3. C_k is an accepting configuration.

104. Recognition (by a Turing machine)

Let M be a Turing machine.

L(M) = { $w \in A^*$ | M accepts w} is the language recognised by M.

105. Turing-recognisable

A language $L \subset A^*$ is called Turing-recognisable if

$exists$ M a Turing machine that recognises L (L = L(M)).

(AKA recursively enumerable)

106. Decider

A Turing machine that either enters an accept state or a reject state $\forall$ inputs $\in$ A.

107. Decide

A decider that recognises some language $L \in A^*$ is said to decide that language.

108. Turing-decidable

A language $L \subset A^*$ is called Turing-decidable if $\exists$ Turing machine M that decides L.

109. Equivalent (Turing machine)

We call two Turing machines $M_1 \ and \ M_2$ equivalent if

$L(M_1) = L(M_2)$ - they recognise the same language,

110. Multitype Turing machine

Turing machine with several tapes each with its own tape head. Initially input is on the first tape and the rest are blank.

If k=number of tapes, and $A^n = \underbrace{A \times A \times \cdot \times A}_{n \ times}$ then the transition mapping is defined as:

$t : \mathcal{S} \times \tilde{A}^k \ \rightarrow \ \tilde{A}^k \times \{L, R, N\}^k$ where N is no movement.

111. Non-deterministic Turing machine

A Turing machine where the machine may proceed in different ways from each state.

The transition mapping of this type is given by:

$t : \mathcal{S} \times \tilde{A} \rightarrow \mathcal{P}(\mathcal{S} \times \tilde{A} \times \{L, R\})$

112. Enumerator

A Turing machine with an attached printer.

It prints out the language it accepts (L) as a sequence of strings. Can print out the strings of the language in any order possibly with repetitions.

113. Complement (language)

Given finite alphabet A and language L $\subset$ A*, the complement $\overline{L}$ of L as

$\overline{L}$ = A*\L (all words over A that are not in L)

114. Co-Turing recognisable

A language L is co-Turing recognisable if its complement is Turing recognisable.

Discrete Mathematics - Algorithms

Kruskal's Algorithm

Let (V,E) be a connected graph with an associated cost function $c : E \to \mathbf{R}$.

1. Start with (V,∅), the subgraph of (V,E) consisting of all vertices and no edges.
2. List all edges in E in a queue so that the cost of edges is non-decreasing in the queue.

   (if $e, e' \in E$, and if c(e) < c(e'), then e precedes e' in the queue)

3. Algorithm:
   - Take edges successively from the front of the queue.
   - Determine whether or not the addition of that edge to the current subgraph will create a cycle.
   - If a cycle would be created, discard the edge.
   - Otherwise, add it to the subgraph.
   - Continue step 3 until the queue is empty.

Prim's Algorithm

Let (V,E) be a connected graph with an associated cost function $c : E \to \mathbf{R}$.

1. Start by choosing some vertex $v \in V$. Our starting subgraph is ({v}, ∅).
2. List all edges in E in a queue so that the cost of edges is non-decreasing in the queue.

   (if $e, e' \in E$, and if c(e) < c(e'), then e precedes e' in the queue)

3. Algorithm:
   - Identify the first edge in the queue which has one vertex included in the current subgraph and one edge not included in the current subgraph.
   - Add that edge to the current subgraph as well as the new vertex.
   - Since the subgraph we started with was a tree, the resulting subgraph is also a tree.
   - Continue step 3 until it is not possible to proceed further (having added all the vertices in V)