Problem 1 :

(a) Show that the logical equivalence relation, $\equiv$, is an equivalence relation on F.

① reflexive:

For ∀ formulas ϕ ∈ F,

that is v(ϕ) = v (ϕ) for all truth assignments v;

-->relation ≡ is reflexive

② sysmetric:

For ∀ ϕ, ψ ∈ F,

if (ϕ, ψ) ∈ ≡ , that is v(ϕ) = v (ψ) for all truth assignments v;

Then we can prove v (ψ) = v(ϕ), that is (ψ, ϕ) ∈ ≡

--> relation ≡ is sysmetric

③ transitive:

For ∀ ϕ, ψ, Ω ∈ F, if (ϕ, ψ) ∈ ≡ and (ψ, Ω) ∈ ≡;

that is v(ϕ) = v (ψ) and v(ψ) = v (Ω) for all truth assignments v;

Then we can prove v (ϕ) = v(Ω), that is (ϕ, Ω) ∈ ≡

--> relation ≡ is transitive

(b) List four elements in $[\bot]$, the equivalence class of $\bot$.

v(⊥) = False;

[⊥] is the equivalence class of ⊥, that is : [⊥] = { ϕ : ϕ ∈ F and ⊥≡ ϕ }

that is v(ϕ) = v(⊥) = False, here are some example ϕ_1, ϕ_2, ϕ_3, ϕ_4:

ϕ_1 : (⊥ ∧ ⊥) cause v(⊥ ∧ ⊥) = v(⊥) && v(⊥) = False;

ϕ_2 : ($\perp \wedge$ T) cause v($\perp \wedge$T) = v($\perp$) && v(T) = False;

ϕ_3 : ($\perp \wedge \phi$) cause v($\perp \wedge \phi$) = v($\perp$) && v(ϕ) = False; no matter what v(ϕ)is;

ϕ_4 : ($\perp \wedge \psi$) cause v($\perp \wedge \psi$) = v($\perp$) && v(ψ) = False; no matter what v(ψ)is;

(c) For all $\varphi, \varphi', \psi, \psi' \in F$ with $\varphi \equiv \varphi'$ and $\psi \equiv \psi'$; show that:

(i) $\neg\varphi \equiv \neg\varphi'$
(ii) $\varphi \wedge \psi \equiv \varphi' \wedge \psi'$
(iii) $\varphi \vee \psi \equiv \varphi' \vee \psi'$

$\phi \equiv \phi'$ and $\psi \equiv \psi'$ means that for all truth assignments v: we have

v(ϕ) = v(ϕ') and v(ψ) = v(ψ')

i :

ϕ	ϕ'	$\neg\phi$	$\neg\phi'$
T	T	F	F
F	F	T	T

So $\neg\phi \equiv \neg\phi'$

ii :

ϕ	ψ	ϕ'	Ψ'	$\phi \wedge \psi$	$\phi' \wedge \psi'$
T	T	T	T	T	T
T	F	T	F	F	F
F	T	F	T	F	F
F	F	F	F	F	F

v($\phi \wedge \psi$) = v($\phi' \wedge \psi'$) for all truth assignments;

so $\phi \wedge \psi \equiv \phi' \wedge \psi'$

iii :

ϕ	ψ	ϕ′	Ψ′	ϕ ∨ ψ	ϕ′ ∨ ψ′
T	T	T	T	T	T
T	F	T	F	T	T
F	T	F	T	T	T
F	F	F	F	F	F

v(ϕ ∨ ψ) = v(ϕ′ ∨ ψ′) for all truth assignments;

so ϕ ∨ ψ ≡ ϕ′ ∨ ψ′

Let us define $F_{\equiv}$ to be the set of **equivalence classes** of F under $\equiv$. That is,

$$F_{\equiv} := \{[\varphi] : \varphi \in F\}.$$

Part (c) above shows that the following operations are *well-defined*[1] on $F_{\equiv}$:

- $[\varphi] \wedge [\psi]$ defined to be $[\varphi \wedge \psi]$
- $[\varphi] \vee [\psi]$ defined to be $[\varphi \vee \psi]$
- $[\varphi]'$ defined to be $[\neg\varphi]$

(d) Show that $F_{\equiv}$ together with the operations defined above forms a Boolean Algebra. Note: you will have to give a suitable definition of a zero element and a one element in $F_{\equiv}$. (12 marks)

According to this question:

We define a zero element : 0 = [⊥] and a one element : 1 = [⊤];

That is : our defined boolean algebra is : ($F_{\equiv}$, ∧, ∨, ′, 0, 1):

identity:

[ϕ] ∨ 0 = [ϕ] ∨ [⊥] = [ϕ ∨⊥] = [ϕ∨False] = [ϕ];

[ϕ] ∧ 1 = [ϕ] ∧ [⊤] = [ϕ ∧⊤] = [ϕ∧True] = [ϕ];

complementation:

$[\phi] \vee [\phi]' = [\phi] \vee [\neg\phi] = [\phi \vee \neg\phi] = [\top] = [\text{True}]$;

$[\phi] \wedge [\phi]' = [\phi] \wedge [\neg\phi] = [\phi \wedge \neg\phi] = [\bot] = [\text{False}]$;

commutative:

$[\phi] \vee [\psi] = [\phi \vee \psi] = [\psi \vee \phi] = [\psi] \vee [\phi]$

$[\phi] \wedge [\psi] = [\phi \wedge \psi] = [\psi \wedge \phi] = [\psi] \wedge [\phi]$

associative:

$([\phi] \vee [\psi]) \vee [\Omega] = [\phi \vee \psi] \vee [\Omega] = [(\phi \vee \psi) \vee \Omega] = [\phi \vee (\psi \vee \Omega)] = [\phi] \vee [\psi \vee \Omega]$

$= [\phi] \vee ([\psi] \vee [\Omega])$

$([\phi] \wedge [\psi]) \wedge [\Omega] = [\phi \wedge \psi] \wedge [\Omega] = [(\phi \wedge \psi) \wedge \Omega] = [\phi \wedge (\psi \wedge \Omega)] = [\phi] \wedge [\psi \wedge \Omega]$

$= [\phi] \wedge ([\psi] \wedge [\Omega])$

distributive:

$[\phi] \vee ([\psi] \wedge [\Omega]) = [\phi] \vee [\psi \wedge \Omega] = [\phi \vee (\psi \wedge \Omega)] = [(\phi \wedge \psi) \vee (\phi \wedge \Omega)]$

$= [(\phi \wedge \psi)] \vee [(\phi \wedge \Omega)] = ([\phi] \wedge [\psi]) \vee ([\phi] \wedge [\Omega])$

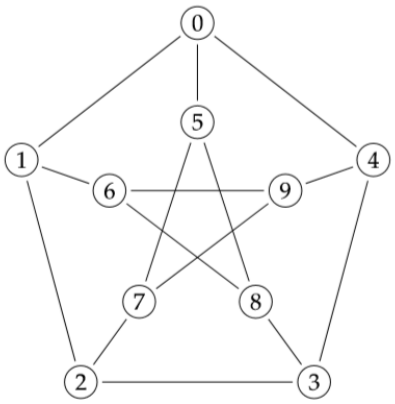
$[\phi] \wedge ([\psi] \vee [\Omega]) = [\phi] \wedge [\psi \vee \Omega] = [\phi \wedge (\psi \vee \Omega)] = [(\phi \vee \psi) \wedge (\phi \vee \Omega)]$

$= [(\phi \vee \psi)] \wedge [(\phi \vee \Omega)] = ([\phi] \vee [\psi]) \wedge ([\phi] \vee [\Omega])$

All rules are satisfable for structure ($F\equiv$, $\wedge$, $\vee$, $'$, 0, 1), so $F\equiv$ together with the operations defined above forms a Boolean Algebra

Problem 2 :

This is the Petersen graph:



(a) Give an argument to show that the Petersen graph does not contain a subdivision of K_5.

Proof :

For graph K_5 , we know :

(1) K_5 is a complete graph, every vertice connects with the other. So every vertice of K_5 has degree 4. That is : deg(v) = 4 for $v \in K_5$;

(2) an subdivsion of K_5 must contain at least 5 vertices whose degree are equal to 4.

Then we look at petersen graph : for $V_1 \sim V_9$, deg(v) = 3, there is no point such that deg(v) = 4

So perterson graph does not contain a subdivsion of K_5.

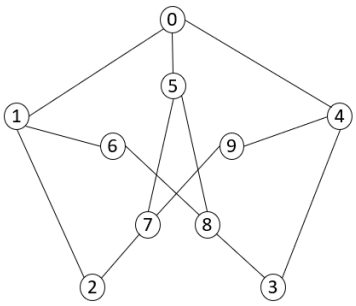
(b) Show that the Petersen graph contains a subdivision of $K_{3,3}$.

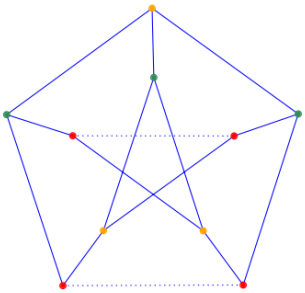
We start with G:

(1) remove edge (6, 9)

(2) remove edge (2, 3)

then we have :





And we have partition of vertices set (1, 4, 5) and (0, 7, 8)

(1, 4, 5) corresponding the green points;

(0, 7, 8) corresponding the yellow points;

The red points are points which are added to making a subdivsion of $K_{3,3}$.

By (1) and (2), we get a conclusion that Petersen graph contains a subdivision of $K_{3,3}$.

Problem 3 :

Harry would like to take each of the following subjects: Defence against the Dark Arts; Potions; Herbology; Transfiguration; and Charms. Unfortunately some of the classes clash, meaning Harry cannot take them both. The list of clashes are:

- Defence against the Dark Arts clashes with Potions and Charms
- Potions also clashes with Herbology
- Herbology also clashes with Transfiguration, and
- Transfiguration also clashes with Charms.

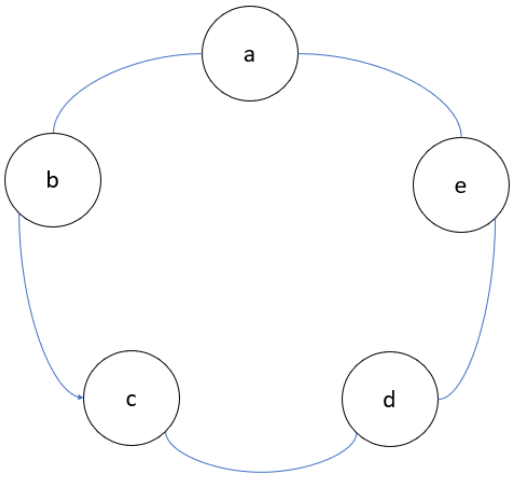
Harry would like to know the **maximum** number of classes he can take.

(a) Model this as a graph problem. Remember to:

(i) Clearly define the vertices and edges of your graph. (4 marks)

(ii) State the associated graph problem that you need to solve. (2 marks)

i : we define a undirected graph G = (V, E) to simulate this scenario :



vertices V = {a, b, c, d, e} : each of subjects

that is : vertice a : Defence against the DarkArts

vertice b : Potions

vertice c : Herbology

vertice d : Transfiguration

vertice e : Charms

edges : E = {(a, b), (b, c), (c, d), (d, e), (e, a)} each pair of subjects which clash with each other

that is : (a, b) : subject a and subject b are conflict with each other;

the same situations for (b, c), (c, d), (d, e), (e, a)

ii : here is the statement:

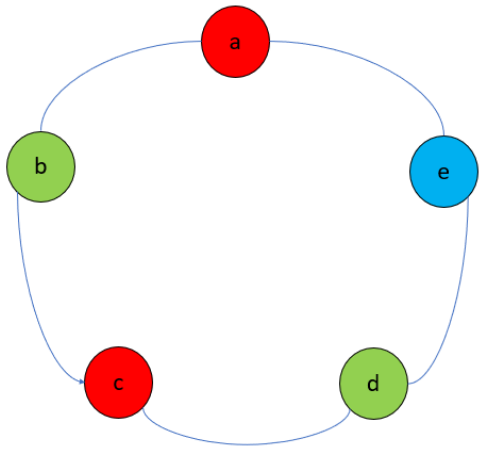
Harry would like to take subjects as many as possible and we know the clashed relationship between each pair of subjects. Each pair of vertices that are connected by an edge are conflict with each other.

And we can assign a color to each vertex and the vertices connected by an edge have different colours.The meaning of the same color is that we can take these vertices(subjects) at the same time, and the vertices connected by an edge have different colours means we cannot take subjects which are conflict with each other.

So to solve this problem, we need to find the least number of colours we need to take.

(b) Give the solution to the graph problem corresponding to this scenario; and solve Harry's problem.

As we can see :



the chromatic number of above graph G = (E, V) is :

$\chi(G) = 3$

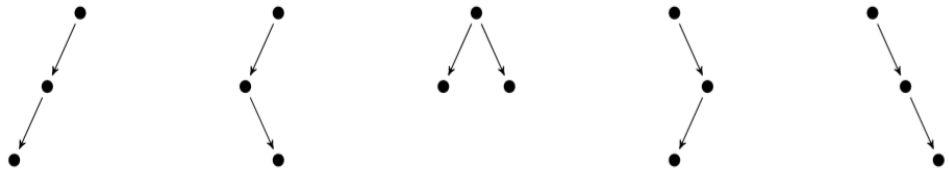
this means the color of vertice e cannot be red or green, we can at most take two subjects of them:

the maximum # of classes he can take = $\chi(G) - 1 = 2$

Problem 4 :

Recall from Assignment 2 the definition of a binary tree data structure: either an empty tree, or a node with two children that are trees.

Let $T(n)$ denote the number of binary trees with n nodes. For example $T(3) = 5$ because there are five binary trees with three nodes:



(a) Using the recursive definition of a binary tree structure, or otherwise, derive a recurrence equation for $T(n)$. (8 marks)

A **full binary tree** is a non-empty binary tree where every node has either two non-empty children (i.e. is a fully-internal node) or two empty children (i.e. is a leaf).

Specially We define T(0) = 1 for the convenience of calculation.

For a binary tree with n nodes, we can seperate n nodes to 2 parts :

A root node with :

Left part : a binary tree with (n-1) nodes;

Right part : a binary tree with 0 nodes or;

Left part : a binary tree with (n-2) nodes;

Right part : a binary tree with 1 nodes or;

……

Left part : a binary tree with 1 nodes;

Right part : a binary tree with (n-2) nodes or;

Left part : a binary tree with 0 nodes;

Right part : a binary tree with (n-1) nodes;

Until all devision possibilities are listed;

So we have :

Basic step :

T(1) = T(0)*T(0) = 1

Recursive step :

T(2) = T(1)*T(0) + T(0)*T(1)

T(3) = T(2)*T(0) + T(1)*T(1) + T(0)*T(2)

T(4) = T(3)*T(0) + T(2)*T(1) + T(1)*T(2) + T(0)*T(3)

T(5) = T(4)*T(0) + T(3)*T(1) + T(2)*T(2) + T(1)*T(3) + T(0)*T(4)

……

So we get an general result of this question :

T(n) = T(n-1)*T(0) + T(n-2)*T(1) + ……+ T(1)*T(n-2) + T(0)*T(n-1);

T(n) = $\sum_{i=0}^{n-1} \mathrm{T}(\mathrm{n}-1-\mathrm{i}) * \mathrm{T}(\mathrm{i})$

A **full binary tree** is a non-empty binary tree where every node has either two non-empty children (i.e. is a fully-internal node) or two empty children (i.e. is a leaf).

(b) Using observations from Assignment 2, or otherwise, explain why a full binary tree must have an odd number of nodes. (4 marks)

As we know from Assignment 2, the number of a binary tree(which is not empty) is :

count(T) = count(T.left_child) + count(T.right_child) + 1

= (1 + count(T.left_child.left_child) + count(T.left_child.right_child)) + (1 + count(T.right_child.left_child) + count(T.right_child.right_child)) + 1

As we can see in the process of recursing, (count(T.left_child) + count(T.right_child)) is an even number for a full binary tree.

--> so we have count(T) is an odd number.

That is a full binary tree must have an odd number of nodes.

(c) Let $B(n)$ denote the number of full binary trees with n nodes. Derive an expression for $B(n)$, involving of $T(n')$ where $n' \leq n$. *Hint: Relate the internal nodes of a full binary tree to* $T(n)$. (6 marks)

Firstly, we list some cases when we calculate T(n) and B(n):

# of nodes	T(n)	B(n)
1	T(1)=1	1
2	T(2)=2	0
3	T(3)=5	1
4	T(4)=14	0
5	T(5)=42	2
.....	.....	.....

We find a latent pattern of T(n) and B(n):

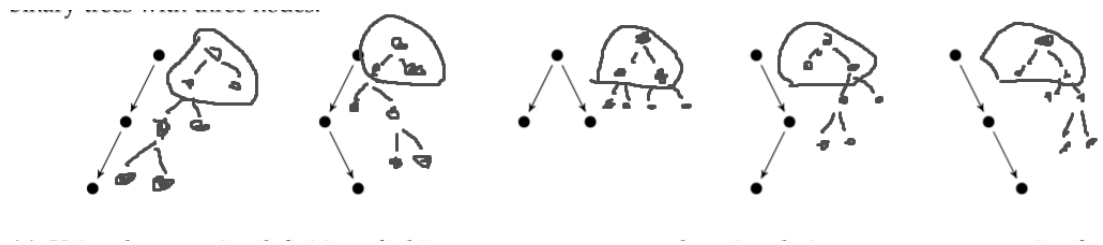
B(1) = T(0);

B(3) = T(1);

B(5) = T(2);

B(7) = T(3);

.....



We can regard the most top nodes together as a root for a full binary tree. And in order to keep the structure of full binary tree, every time we add two nodes into either the left side or the right side. This is the process of B(1), B(3), B(5), B(7),....;

And it is the same as the way we find T(n). the difference between this two methods is that we consider root a single node when we find T(n).

So the relation between B(n) and T(n') is :

B(2i+1) = T(i) and i $\in$ N

A well-formed formula is in **Negated normal form** if it consists of just $\wedge$, $\vee$, and literals (i.e. propositional variables or negations of propositional variables). That is, a formula that results after two steps of the process for transforming a formula into a logically equivalent one. For example, $p \vee (\neg q \wedge \neg r)$ is in negated normal form; but $p \vee \neg(q \vee r)$ is not.

Let $F(n)$ denote the number of well-formed, negated normal form formulas[2] there are that use precisely n propositional variables exactly one time each. So $F(1) = 2$, $F(2) = 16$, and $F(4) = 15360$.

(d) Using your answer for part (c), give an expression for $F(n)$. (4 marks)

Due to this property of Negated normal form, we can establish a full binary tree structure to simulate Negated normal form.

That is, for all internal nodes : they represent either $\wedge$ or $\vee$;

For all leaves nodes : they represent either propositional variables or negations of propositional variables;

We simulate Negated normal form with full binary tree. For a full binary tree with n nodes, there are B(2n-1) different kinds of full-binary structures.

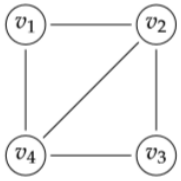
And for every internal node in every structure, it has two choice : $\wedge$ or $\vee$. for every leave node in every structure, it has n! choice to choose the

corresponding propositional variables, and every propositional variables has origin literal and negation literal.

$F(n) = B(2n-1) * 2^{n-1} * n! * 2 = B(2n-1) * 2^{2n-1} * n!$

Problem 5 :

Consider the following graph:



and consider the following process:

- Initially, start at v_1.
- At each time step, choose one of the vertices adjacent to your current location uniformly at random, and move there.

Let $p_1(n)$, $p_2(n)$, $p_3(n)$, $p_4(n)$ be the probability your location after n time steps is v_1, v_2, v_3, or v_4 respectively. So $p_1(0) = 1$ and $p_2(0) = p_3(0) = p_4(0) = 0$.

(a) Express $p_1(n+1)$, $p_2(n+1)$, $p_3(n+1)$, and $p_4(n+1)$ in terms of $p_1(n)$, $p_2(n)$, $p_3(n)$, and $p_4(n)$. (6 marks)

For $P_1(n+1)$, it can only happen when your location after n time steps is v_2 or v_4, because if your location is v_3 or v_1 after n time steps, it cannot reach v_1 at next step according to the rule;

Same reason for $P_2(n+1)$, $P_3(n+1)$ and $P_4(n+1)$;

Each verice k have uniformly-random possibility to reach corresponding adjacent vertices, so we have :

$P_1(n+1) = P_2(n) * 1/3 + P_4(n) * 1/3 = 1/3 * (P_2(n) + P_4(n))$;

$P_2(n+1) = P_1(n) * 1/2 + P_3(n) * 1/2 + P_4(n) * 1/3$;

$P_3(n+1) = P_2(n) * 1/3 + P_4(n) * 1/3 = 1/3 * (P_2(n) + P_4(n))$

$P_4(n+1) = P_1(n) * 1/2 + P_3(n) * 1/2 + P_2(n) * 1/3$;

(b) As n gets larger, each $p_i(n)$ converges to a single value (called the steady state) which can be determined by setting $p_i(n+1) = p_i(n)$ in the above equations. Determine the steady state probabilities for all vertices. (8 marks)

We get 5 equations from (a):

$$\begin{cases} \mathrm{P1(n+1)} = \frac{1}{3} * \left(\mathrm{P2(n)} + \mathrm{P4(n)}\right) = \mathrm{P1(n)} \\ \mathrm{P2(n+1)} = \mathrm{P1(n)} * \frac{1}{2} + \mathrm{P3(n)} * \frac{1}{2} + \mathrm{P4(n)} * \frac{1}{3} = \mathrm{P2(n)} \\ \mathrm{P3(n+1)} = \frac{1}{3} * \left(\mathrm{P2(n)} + \mathrm{P4(n)}\right) = \mathrm{P3(n)} \\ \mathrm{P4(n+1)} = \mathrm{P1(n)} * \frac{1}{2} + \mathrm{P3(n)} * \frac{1}{2} + \mathrm{P2(n)} * \frac{1}{3} = \mathrm{P4(n)} \\ \mathrm{P1(n)} + \mathrm{P2(n)} + \mathrm{P3(n)} + \mathrm{P4(n)} = 1 \end{cases}$$

Calculate 5 equations, we can get every probablity with :

$P_1(n) = P_3(n) = 1/5;$

$P_2(n) = P_4(n) = 3/10;$

(c) The distance between any two vertices is the length of the shortest path between them. What is your expected distance from v_1 in the steady state? (4 marks)

According to the question, we let $|v_i\ v_j|$ represents the distance between vertice v_i and v_j :

Expected_distance = $|v_1\ v_1|* P_1(n) + |v_2\ v_1|* P_2(n) + |v_3\ v_1|* P_3(n) + |v_4\ v_1|* P_4(n) = 0*(1/5) + 1*(3/10) + 2*(1/5) + 1*(3/10) = 1$