**Problem 1 :**



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



v(⊥) = False;

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

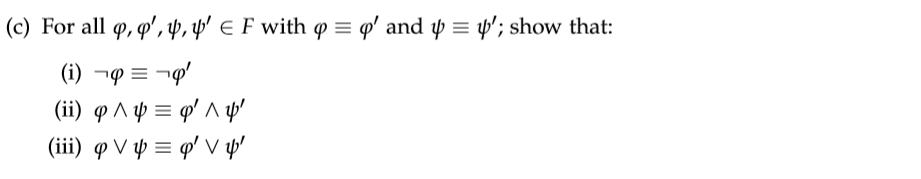
that is v(ϕ) = v(⊥) = False, here are some example ϕ1, ϕ2, ϕ3, ϕ4:

ϕ1 : (⊥∧⊥) cause v(⊥∧⊥) = v(⊥) && v(⊥) = False;

ϕ2 : (⊥∧ T) cause v(⊥∧T) = v(⊥) && v(T) = False;

ϕ3 : (⊥∧ ϕ) cause v(⊥∧ϕ) = v(⊥) && v(ϕ) = False; no matter what v(ϕ)is;

ϕ4 : (⊥∧ ψ) cause v(⊥∧ψ) = v(⊥) && v(ψ) = False; no matter what v(ψ)is;



ϕ ≡ ϕ’ and ψ ≡ ψ’ means that for all truth assignments v: we have

v(ϕ) = v(ϕ’) and v(ψ) = v(ψ’)

i :

|  |  |  |  |
| --- | --- | --- | --- |
| ϕ | ϕ’ | ¬ϕ | ¬ϕ’ |
| T | T | F | F |
| F | F | T | T |

So ¬ϕ ≡¬ϕ’

ii :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ϕ | ψ | ϕ’ | Ψ’ | ϕ∧ψ | ϕ'∧ψ’ |
| 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(ϕ∧ψ) = v(ϕ’∧ψ’) for all truth assignments;

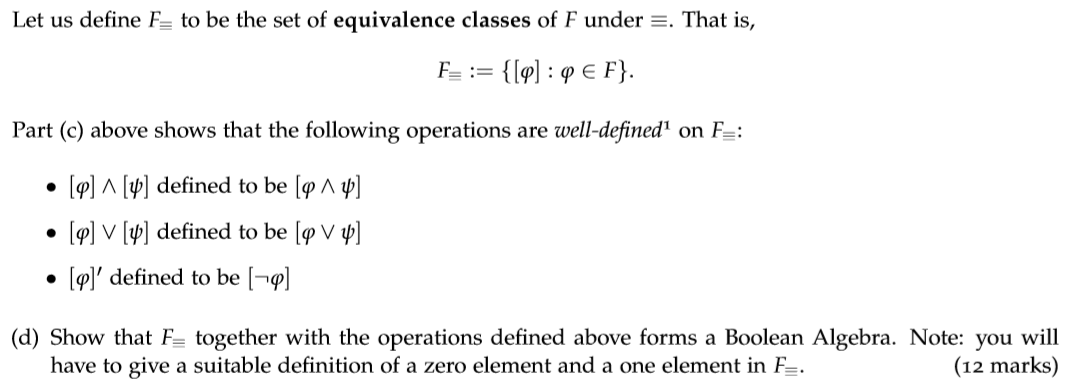
so ϕ∧ψ ≡ ϕ’∧ψ’

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’



According to this question:

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

That is : our defined boolean algebra is : (F≡, ∧, ∨, ‘, 0, 1):

identity:

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

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

complementation:

[ϕ] ∨ [ϕ]’ = [ϕ] ∨ [¬ϕ] = [ϕ ∨ ¬ϕ] = [Т] = [True];

[ϕ] ∧ [ϕ]’ = [ϕ] ∧ [¬ϕ] = [ϕ ∧ ¬ϕ] = [⊥] = [False];

commutative:

[ϕ] ∨ [ψ] = [ϕ∨ψ] = [ψ∨ϕ] = [ψ] ∨ [ϕ]

[ϕ] ∧ [ψ] = [ϕ∧ψ] = [ψ∧ϕ] = [ψ] ∧ [ϕ]

associative:

([ϕ] ∨ [ψ]) ∨ [**Ω**] = [ϕ∨ψ] ∨ [**Ω**] = [(ϕ∨ψ)∨**Ω**] = [ϕ∨(ψ∨**Ω**)] = [ϕ] ∨ [ψ∨**Ω**] = [ϕ] ∨ ([ψ] ∨ [**Ω**])

([ϕ] ∧ [ψ]) ∧ [**Ω**] = [ϕ∧ψ] ∧ [**Ω**] = [(ϕ∧ψ)∧**Ω**] = [ϕ∧(ψ∧**Ω**)] = [ϕ] ∧ [ψ∧**Ω**] = [ϕ] ∧ ([ψ] ∧ [**Ω**])

distributive:

[ϕ] ∨ ([ψ] ∧ [**Ω**]) = [ϕ] ∨ [ψ∧**Ω**] = [ϕ∨(ψ∧**Ω**)] = [(ϕ∧ψ)∨(ϕ∧**Ω**)]

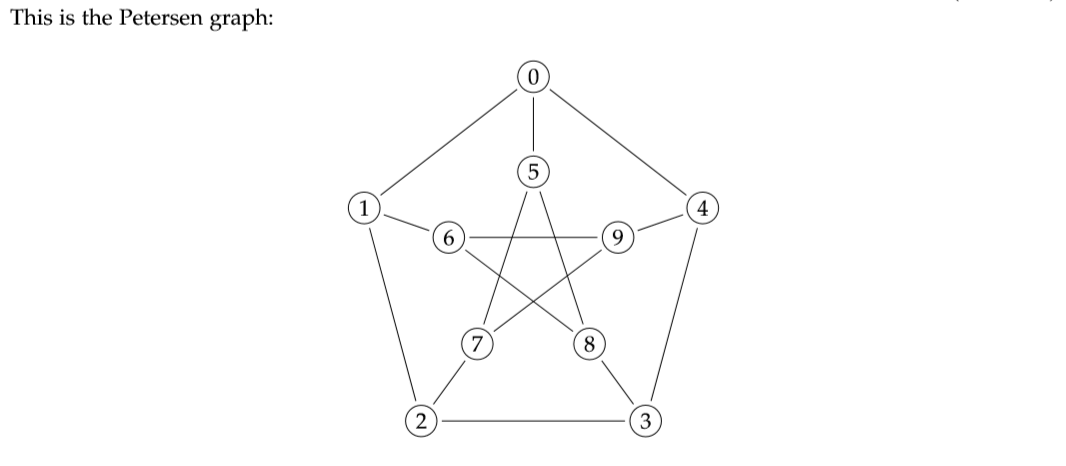
= [(ϕ∧ψ)] ∨ [(ϕ∧**Ω**)] = ([ϕ] ∧[ψ]) ∨ ([ϕ] ∧[**Ω**])

[ϕ] ∧ ([ψ] ∨ [**Ω**]) = [ϕ] ∧ [ψ∨**Ω**] = [ϕ∧(ψ∨**Ω**)] = [(ϕ∨ψ) ∧(ϕ∨**Ω**)]

= [(ϕ∨ψ)] ∧ [(ϕ∨**Ω**)] = ([ϕ] ∨[ψ]) ∧ ([ϕ] ∨[**Ω**])

All rules are satisfable for structure (F≡, ∧, ∨, ‘, 0, 1), so F≡ together with the operations deﬁned above forms a Boolean Algebra

**Problem 2 :**





Proof :

For graph K5 , we know :

(1) K5 is a complete graph, every vertice connects with the other. So every vertice of K5 has degree 4. That is : deg(v) = 4 for v ∈ K5;

(2) an subdivsion of K5 must contain at least 5 vertices whose degree are equal to 4.

Then we look at petersen graph : for V1 ~ V9, deg(v) = 3, there is no point such that deg(v) = 4

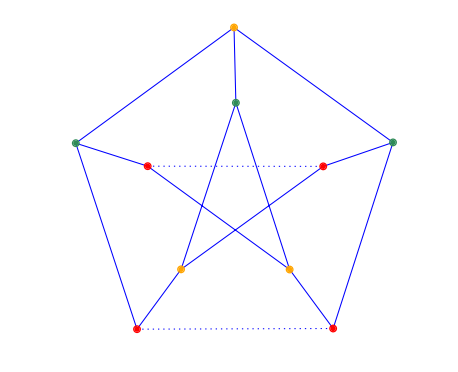
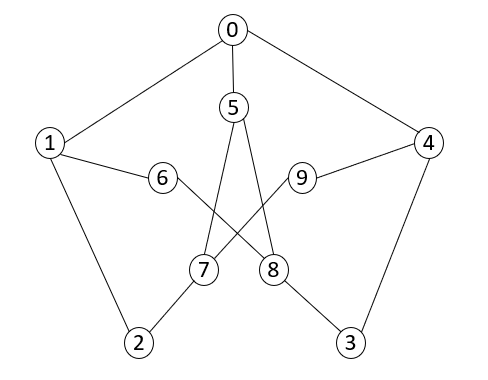
So perterson graph does not contain a subdivsion of K5.



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)

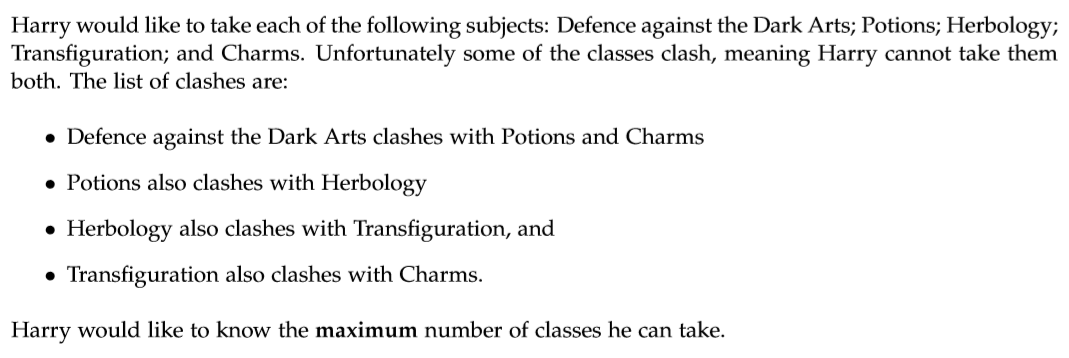
(1, 4, 5) corresponding the green points;

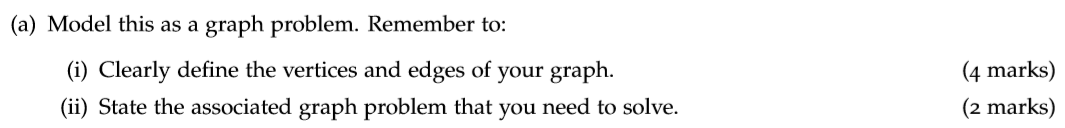
(0, 7, 8) corresponding the yellow points;

The red points are points which are added to making a subdivsion of K3,3.

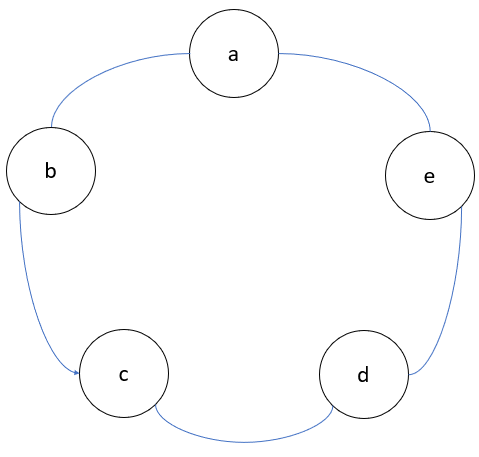
By (1) and (2), we get a conclusion that Petersen graph contains a subdivision of K3,3.

**Problem 3 :**





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:

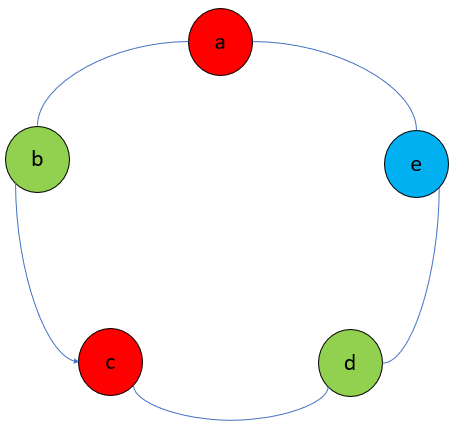
Harry would like to take subjects as many as possible and we know the clashed relationship between each pair of subjects. Each pair of vertices that are connected by an edge are conflict with each other.

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.



As we can see :



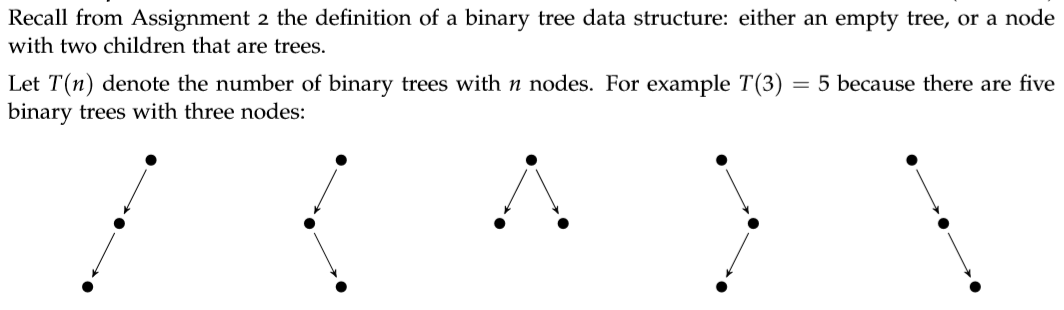
the chromatic number of above graph G = (E, V) is :

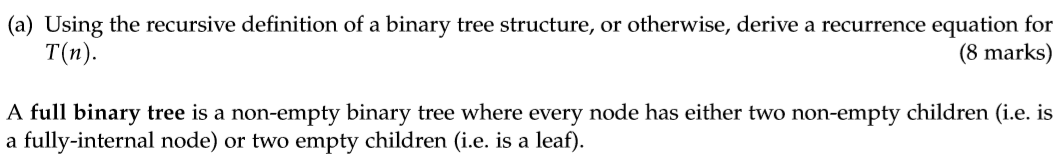
χ(G) = 3

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 = χ(G) – 1 = 2

**Problem 4 :**





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)

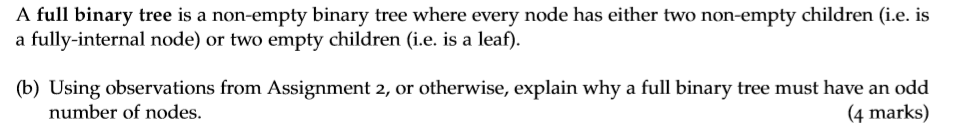
T(5) = T(4)\*T(0) + T(3)\*T(1) + T(2)\*T(2) + T(1)\*T(3) + T(0)\*T(4)

……

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) =



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

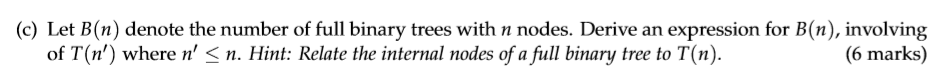
count(T) = count(T.left\_child) + count(T.right\_child) + 1

= (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.



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):

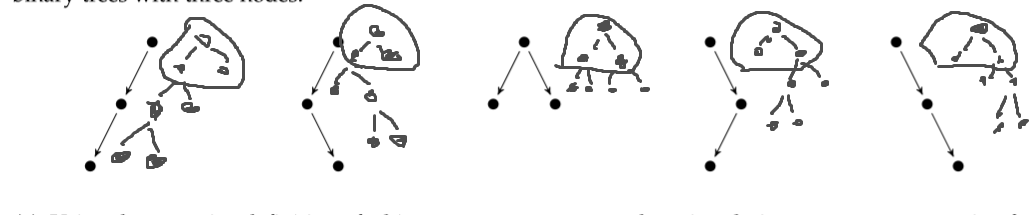
B(1) = T(0);

B(3) = T(1);

B(5) = T(2);

B(7) = T(3);

……

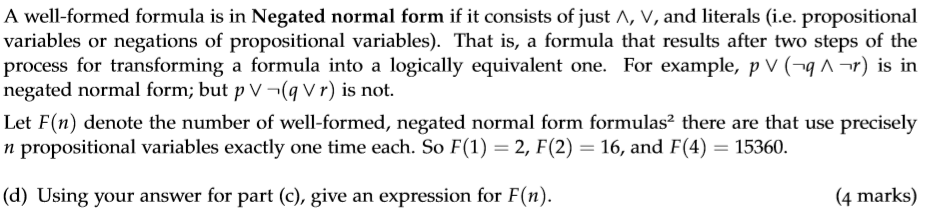


We can regard the most top nodes together as a root for a full binary tree. And in order to keep the structure of full binary tree, every time we add two nodes into either the left side or the right side. This is the process of B(1), B(3), B(5), B(7),….;

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 ∈ N



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 ∧ or ∨;

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

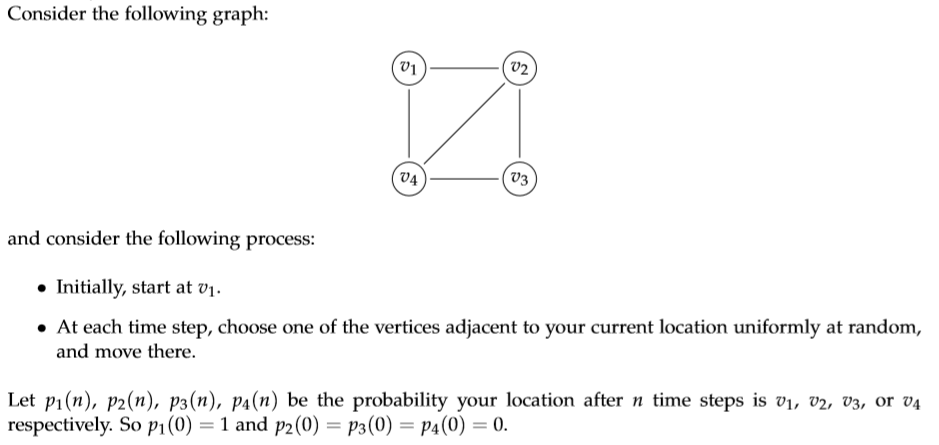
We simulate Negated normal form with full binary tree. For a full binary tree with n nodes, there are B(2n-1) different kinds of full-binary structures.

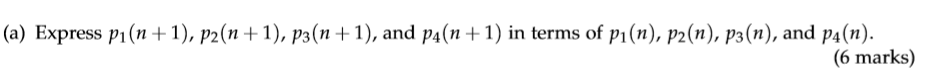
And for every internal node in every structure, it has two choice : ∧ or ∨.

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) \* 2n-1 \* n! \* 2 = B(2n-1) \* 22n-1 \* n!

**Problem 5 :**





For P1(n+1), it can only happen when your location after n time steps is v2 or v4, because if your location is v3 or v1 after n time steps, it cannot reach v1 at next step according to the rule;

Same reason for P2(n+1), P3(n+1) and P4(n+1);

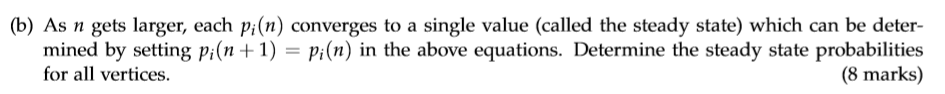
Each verice k have uniformly-random possibility to reach corresponding adjacent vertices, so we have :

P1(n+1) = P2(n) \* 1/3 + P4(n) \* 1/3 = 1/3 \* (P2(n) + P4(n));

P2(n+1) = P1(n) \* 1/2 + P3(n) \* 1/2 + P4(n) \* 1/3;

P3(n+1) = P2(n) \* 1/3 + P4(n) \* 1/3 = 1/3 \* (P2(n) + P4(n))

P4(n+1) = P1(n) \* 1/2 + P3(n) \* 1/2 + P2(n) \* 1/3;

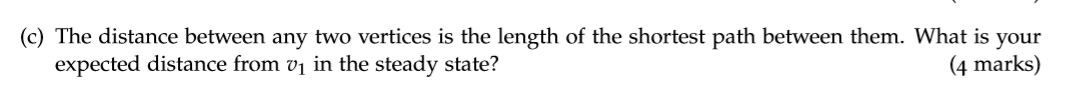


We get 5 equations from (a):

Calculate 5 equations, we can get every probablity with :

P1(n) = P3(n) = 1/5;

P2(n) = P4(n) = 3/10;



According to the question, we let |vi vj| represents the distance between vertice vi and vj :

Expected\_distance = |v1 v1|\* P1(n) + |v2 v1|\* P2(n) + |v3 v1|\* P3(n) + |v4 v1|\* P4(n) = 0\*(1/5) + 1\*(3/10) + 2\*(1/5) + 1\*(3/10) = 1