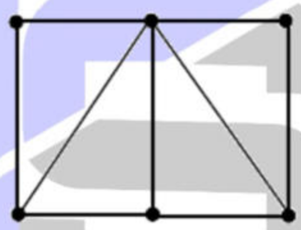
www.ignousite.com
Course Code : MCS-033
Course Title : Advanced Discrete Mathematics
Assignment Number : MCA(III)/033/Assignment/2020-21
Maximum Marks : 100
Weightage : 25%
Last Dates for Submission : 31st October, 2020 (For July, 2020 Session)
: 15th April, 2021 (For January, 2021 Session)

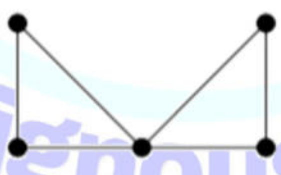
Q1. Is a Hamiltonian graph Eulerian ? Is a Eulerian graph Hamiltonian ? Show with the help of a suitable example.

Ans. There's a big difference between Hamiltonian graph and Euler graph. Hamiltonian is the one in which each vertex is visited exactly once except the starting and ending vertex(need to remember) and Euler allows vertex to be repeated more than once but each edge should be visited exactly once without any repetition.

If a graph has a Hamiltonian circuit, then the graph is called a Hamiltonian graph. Important: An Eulerian circuit traverses every edge in a graph exactly once, but may repeat vertices, while a Hamiltonian circuit visits each vertex in a graph exactly once but may repeat edges. Example: Following graph is Hamiltonian and non-Eulerian



It is not the case that every Eulerian graph is also Hamiltonian. It is required that a Hamiltonian cycle visits each vertex of the graph exactly once and that an Eulerian circuit traverses each edge exactly once without regard to how many times a given vertex is visited. Take as an example the following graph:



It's easy to find an Eulerian circuit, but there is no Hamiltonian cycle because the center vertex is the only way one can get from the left triangle to the right.

Q2. (a) Solve $a_{n+1}+1 = 5a_n$ for n≥ 0, a_0= 2 by Substitution method.

Ans.

Ans 2 (a) Given :-

$$a_{n+1} + 1 = 5a_n \quad , \quad a_0 = 2$$

we successively apply the recurrence formula:-

$$a_{n+1} + 1 = 5a_n$$

$$a_n = \frac{a_{n+1} + 1}{5}$$

$$a_n = \frac{a_n + 1}{5}$$

$$= \frac{a_{n-1} + 1}{5}$$

$$= \frac{a_{n-2} + 1}{5}$$

$$= \frac{a_{n-3} + 1}{5}$$

$$\vdots$$

$$\frac{(n+1)!\, a_0}{5}$$

$$2) \quad \frac{2}{5}(n+1)!$$

(b) Solve the recurrence by using iterative approach :

$a_n = a_{n-1} + 2_n + 3$, $a_0 = 4$.

Ans.

(b) $a_n = a_{n-1} + 2n + 3 \qquad a_0 = 4$

we successively apply the recurrence relation :-

$a_n = a_{n-1} + 2n + 3 + 0 = a_{n-1} + 2n + 3$

$a_{n-1} = a_{n-2} + 2n - 2 + 3 = a_{n-2} + 2n + 1$

$a_{n-2} = a_{n-3} + 2n - 4 + 3 = a_{n-3} + 2n - 1$

$a_{n-3} = a_{n-4} + 2n - 6 + 3 = a_{n-4} + 2n - 3$

. . .

$= a_{n-n} + 2(n-n) + 3$

$= a_0 + 3$

Q3. Define a recurrence relation. Describe the following problems with the help of examples which can be solved through Divide and Conquer technique and Show its recurrence relation.

(i) Binary Search

(ii) Merge Sort

Solve these recurrence relations with a substitution method.

Ans. Recurrence relation: A recurrence relation is an equation that defines a sequence based on a rule that gives the next term as a function of the previous term(s).

The simplest form of a recurrence relation is the case where the next term depends only on the immediately previous term. If we denote the n^{th} term in the sequence by x_n, such a recurrence relation is of the form

$$x_{n+1}=f(x_n)$$

for some function ff. One such example is $x_n+1=2-x_n/2$.

A recurrence relation can also be higher order, where the term x_{n+1} could depend not only on the previous term x_n but also on earlier terms such as x_{n-1}, x_{n-2}, etc. A second order recurrence relation depends just on x_n and x_{n-1} and is of the form

$$x_{n+1}=f(x_n,x_{n-1})$$

For some function *f* with two inputs. For example, the recurrence relation $x_{n+1}=x_n+x_{n-1}$ can generate the Fibonacci numbers.

To generate sequence basd on a recurrence relation, one must start with some initial values. For a first order recursion $x_{n+1}=f(x_n)$, one just needs to start with an initial value x_0 and can generate all remaining terms using the recurrence relation. For a second order recursion $x_{n+1}=f(x_n,x_{n-1})$, one needs to begin with two values x_0 and x_1. Higher order recurrence relations require correspondingly more initial values.

The divide-and-conquer technique involves taking a large-scale problem and dividing it into similar sub-problems of a smaller scale, and recursively solving each of these sub-problems. Generally, a problem is divided into sub-problems repeatedly until the resulting sub-problems are very easy to solve.

This type of algorithm is so called because it divides a problem into several levels of sub-problems, and conquers the problem by combining the solutions at the various levels to form the overall solution to the problem.

Divide-and-conquer algorithms:
1. Dividing the problem into smaller sub-problems
2. Solving those sub-problems
3. Combining the solutions for those smaller subproblems
to solve the original problem
Recurrences are used to analyze the computational
complexity of divide-and-conquer algorithms.

Divide-and-conquer recurrence

- Assume a divide-and-conquer algorithm divides a problem of size n into a sub-problems.
- Assume each sub-problem is of size n/b.
- Assume f(n) extra operations are required to combine the solutions of sub-problems into a solution of the original problem.

- Let T(n) be the number of operations required to solve the problem of size n.

$T(n) = a\, T(n/b) + f(n)$

In order to make the recurrence well defined T(n/b) term will actually be either $T(\lceil n/b \rceil)$ or $T(\lfloor n/b \rfloor)$.

The recurrence will also have to have initial conditions. (e.g. T(1) or T(0))

(i) Binary Search : Binary search is one such divide and conquer algorithm to assist with the given problem; note that a sorted array should be used in this case too. This search algorithm recursively divides the array into two sub-arrays that may contain the search term. It discards one of the sub-array by utilising the fact that items are sorted. It continues halving the sub-arrays until it finds the search term or it narrows down the list to a single item. Since binary search discards the sub-array it's pseudo Divide & Conquer algorithm. What makes binary search efficient is the fact that if it doesn't find the search term in each iteration, it just reduces the array/list to it's half for the next iteration. The time complexity of binary search is O(log n), where n is the number of elements in an array. If the search term is at the centre of the array, it's considered to be the best case since the element is found instantly in a go. Hence the best case complexity will be O(1).

Now, consider the above-mentioned time complexities. Complexities like O(1) and O(n)are very intuitive to understand:

1. O(1) : refers to an operation where the value/the element is accessed directly

2. O(n) : refers to a (set of) where the element can only be accessed by traversing a set of n elements, like in linear search.

But what does O(log n) really mean? It may seem difficult to understand but let's visualize it using a simple example of binary search, while searching for a number in a sorted array which will take the worst-case time complexity:

$T(n) = T(\lfloor n/2 \rfloor) + 2$

$T(1) = 2$

(ii) Merge Sort : Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.

Idea:

- Divide the unsorted list into N sublists, each containing 1 element.
- Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. N will now convert into N/2 lists of size 2.
- Repeat the process till a single sorted list of obtained.

While comparing two sublists for merging, the first element of both lists is taken into consideration. While sorting in ascending order, the element that is of a lesser value becomes a new element of the sorted list. This procedure is repeated until both the smaller sublists are empty and the new combined sublist comprises all the elements of both the sublists.

Merge sort:

The merge sort algorithm splits a list with n elements into two list with $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ elements. (The list with 1 element is considered sorted.)

It uses less than n comparison to merge two sorted lists of $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$ elements.

Find a recurrence T(n) that represents the number of operations required to solve the problem of size n.

(The merge sort may have less operations than T(n))

Solution:

The problem of size n can be reduced into two problems of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$.

n comparisons are required to find the original solution from those sub-problems.

$T(1) = 0 \qquad T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$

Q4. To multiply two n-digit numbers, one must do normally n^2 digit-times-digit multiplications. Use a divide and conquer algorithm to propose an algorithm when n is a power of 2.

Ans. With divide-and-conquer multiplication, we split each of the numbers into two halves, each with n/2 digits. I'll call the two numbers we're trying to multiply a and b, with the two halves of a being aL (the left or upper half) and aR (the right or lower half) and the two halves of b being b_L and b_R.

Basically, we can multiply these two numbers as follows.

$$
\begin{array}{rccc}
 & & a_L & a_R \\
\times & & b_L & b_R \\
\hline
 & & a_L b_R & a_R b_R \\
+ & a_L b_L & a_R b_L & \\
\hline
 & a_L b_L & (a_L b_R + a_R b_L) & a_R b_R
\end{array}
$$

That image is just a picture of the idea, but more formally, the derivation works as follows.

$$
\begin{aligned}
ab &= (a_L\ 10^{n/2} + a_R)\ (b_L\ 10^{n/2} + b_R) \\
&= a_L\ b_L\ 10^n + a_L\ b_R\ 10^{n/2} + a_R\ b_L\ 10^{n/2} + a_R\ b_R \\
&= a_L\ b_L\ 10^n + (a_L\ b_R + a_R\ b_L)\ 10^{n/2} + a_R\ b_R
\end{aligned}
$$

Thus, in order to multiply a pair of n-digit numbers, we can recursively multiply four pairs of n/2-digit numbers. The rest of the operations involved are all O(n) operations. (Multiplying by 10^n may look like a multiplication (and hence not O(n)), but really it's just a matter of appending n zeroes onto the number, which takes O(n) time.) That's fine as far as it goes, but it turns out that it's not far enough: If you write down the recurrence and solve it, it turns out to solve to $O(n^2)$, which is what we had from grade school. And this algorithm is much more complicated. Not a very encouraging result.

But there turns out to be a very clever approach, we permits us to reduce the number of n/2-digit multiplications from four to three! This clever idea yields a better result.

Q5. Find a recurrence relation and initial conditions for 4, 14, 44, 134, 404, …

Ans.

We are going to try to solve these recurrencence relations. By this we mean something very similar to solving differential equations.

We want to find a function of n which satisfies the recurrence relation as well as the initial condition

4, 14, 144, 134, 404

2) 10, 130, -10, 270

Initial condition is 2.

Thus checking that the solution is correct.

Q6. Prove/show the followings:

- **the sum of the degrees of the vertices of G is twice the number of edges**
- **If W is a u-v walk joining two distinct vertices u and v, then there is a path joining u and v contained in the walk using the principles of mathematical induction**
- **A connected graph G is Eulerian if and only if the degree of each of its vertices is even.**
- **If G is a connected planar (p,q)-graph, then the number r of the regions of G is given by r = q - p +2**

Ans.

Ans (6): (1)

The Number of edges = 24

Degree of each vertex = 4

let the number of vertices in the graph = n

Using Handshaking theorem =)

Sum of degree of all vertices = 2 × No. of edges

substituting the value, we get =)

$n \times 4 = 2 \times 24$

$n = 2 \times 6$

$n = 12$

Thus, Number of vertices in the graph = 12.

(2) let G be a graph and let u and v be two of its vertices. prove that if there is a walk from u to v. then there is also a path from u to v.

Using mathematical induction :-

walk :- A walk in a graph G = (V, E) is a sequence of vertices $v_1, v_2, v_3 \ldots v_k$

Path :- let w be a walk between u and v.

if $|W| = 1$ then W is just the edges uv and it is a u-v path.

Induction step :-

Now assume the statement is true for all u-v walk of smaller size than W. If all the vertices in W are distinct, then W is u-v path and we are done. Otherwise W has a repeated vertex say x,

let W' be the walk obtained by suppressing the section of W. b/w two repetition.

obviously, W' is u-v walk of smaller length than W.

By Induction hypothesis, W' has u-v path which mean that W has u-v path.

(iii) The degree of a vertex is the number of edges incident with that vertex.

So, let G be a graph that has an Eulerian circuit. Every time we arrive at a vertex during our traversal of G, we enter one edge and exit another. Thus there must be an even number of edges at every vertex. Therefore every vertex of G has even degree.

(iv) We prove it by induction on q.

If $q = 0$ then $p = 1$ obviously $r = 1$

and results is follows.

Inductive step :- Assume that the Euler theorem holds for all connected graphs with fewer than q $(q \geq 1)$ edges

let G be a connected plane graph with q edges.

If G is a tree. then $p = q + 1$ and $r = 1$

If G is not tree, then it has an enclosed face. the edges of the face form a cycle.

$H = G - e$

Since, $q(H) = q - 1$

$p(H) - q(H) + r(H) = 2$, Here $p(H) = p$, $r(H) = r - 1$

The result follows.

Q7. Show the followings:

- **Show that for a subgraph H of a graph G, Δ(H) ≤Δ (G)**
- **Show that $Km,n$is not Hamiltonian when m + n is odd**

Ans.

⑦A(i) Let H be a subgroup of G, then N(H) is also a subgroup of G.

By definition, $N(H) = \{x \in G : Hx = xH\}$.

Now for any $x \in H$, $Hx = xH$

$\Rightarrow x \in N(H) \Rightarrow H \subset N(H)$

Since H and N(H) are both subgroup of G, so H is a subgroup of N(H).

Also, $Hx = xH$ for all $x \in N(H)$

$\Rightarrow$ H is a normal subgroup of N(H)

(ii) Let H be a normal subgroup of K, where K itself is a subgroup of G.

Now, $Hx = xH$ for all $x \in K \subset G$.

The above result can be expressed as:

any $x \in K \Rightarrow Hx = xH$, $x \in G$

$\Rightarrow x \in N(H) \Rightarrow K \subset N(H)$

Hence N(H) is the largest subgroup of G in which H is ~~m~~ normal.

(7) (B) Let the set of vertices of $K_{m,n}$ be $V = V_0 \cup V_1$, where $|V_0| = m$, $|V_1| = n$, and all edges are between V_0 and V_1. A path in $K_{m,n}$ must alternate between vertices in V_0 and vertices in V_1. A circuit necessarily has $2k$ vertices for some positive integer k; k of these vertices are in V_0, and the other k are in V_1. Thus if $m \neq n$ it is impossible for a circuit in $K_{m,n}$ to hit every vertex, and therefore $K_{m,n}$ can have a Hamilton circuit only if $m = n$. Conversely, it's easy to show by induction that $K_{m,m}$ has a Hamilton circuit for for all $m \geq 2$.

A Hamilon path in $K_{m,n}$ that cannot be extended to a Hamilton circuit must have both ends in V_0 or both ends in V_1. Suppose that both ends are in V_0. Then the path has $2k$ edges and $2k+1$ vertices for some k; moreover, $k+1$ of the vertices are in V_0, and k are in V_1. But this is a Hamilton path, so it reaches every vertex exactly once, and therefore $m = k+1$ and $n = k$, i.e., $m = n+1$.

$n = m+1$, Hamiltonian circuit
$m \geq 1$.

Q8. Define homogeneous recurrence relation. Write the first order and second order homogeneous recurrence relations with constant coefficients giving an example for each.Solve the following recurrence relation:

$a_n + a_{n-1} - 6a_{n-2} = 0$ for n ≥ 2 given that $a0=-1, a_1= 8$

Ans.

Suppose we want to solve the recurrence relation expressed as combination of the two previous term, such as

$$a_n + a_{n-1} - 6a_{n-2} = 0$$

Now iterative is too complicated. but think just for a second what would happen if we did iterate.

Multiple a previous iteration by 6. Perhaps the solution will take the form r^n for some constant r

$$r^n + r^{n-1} - 6r^{n-2} = 0$$

Now,

$$r^{n-2}(r^2 + r - 6) = 0$$

$$r^{n-2} = 0$$

$$r^2 + r - 6 = 0$$

$$r^2 + 3r - 2r - 6 = 0$$

$$(r+3)(r-2) = 0$$

$$r = -3, 2$$

$a_n = (-3)^n$ is a solution to the recurrence relation.

$a_n = 2^n$ which one is correct?

$a_n = (-3)^n + 2^n$

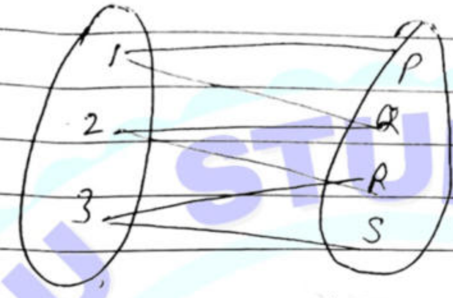
$a_n = 7(-3)^n + 4.3^n$

$a_n = a(-2)^n + b.3^n$ is a solution.

Q9. (a) Find chromatic number of bipartite graph Km, n.

Ans.

(a)



Chromatic number = 2

The total chromatic number of a Complete bipartite graph is either

$\Delta + 1$ or $\Delta + 2$.

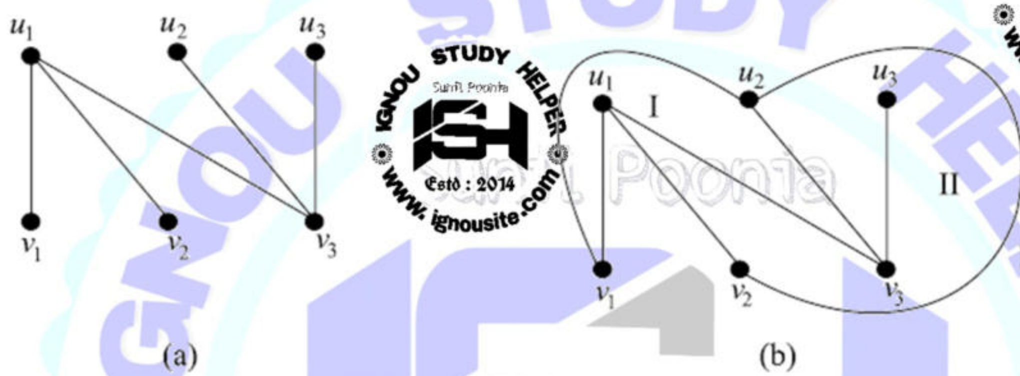
More specifically, if $m \neq n$ then

$\chi'' K(m,n) = \Delta + 1 = \max(m,n) + 1$.

(b) Show that K3, 3 is non-planar.

Ans.

The complete bipartite graph has six vertices and nine edges. Let the vertices be u1, u2, u3, v1, v2, v3. We have edges from every ui to each vi, 1 _ i _ 3. First we take the edges from u1 to each v1, v2 and v3. Then we take the edges between u2 to each v1, v2 and v3. Thus we get three regions namely I, II and III. Finally we have to draw the edges between u3 to each v1, v2 and v3. We can draw the edge between u3 and v3 inside the region II without any crossover, Figure 6.4(b). But the edges between u3 and v1, and u3 and v2 drawn in any region have a crossover with the previous edges. Thus the graph cannot be embedded in a plane. Hence K3,3 is nonplanar.



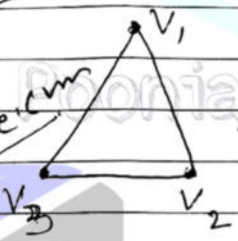
(c) Construct a 5-regular graph on 10 vertices.

Ans.

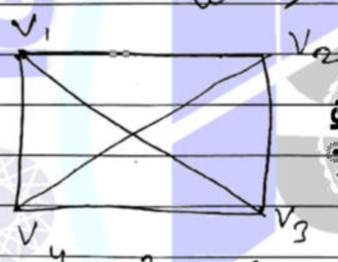
(7) (c)

Regular graph → A graph in which degree of each vertex is equal. let K is called K - regular graph.

1 regular

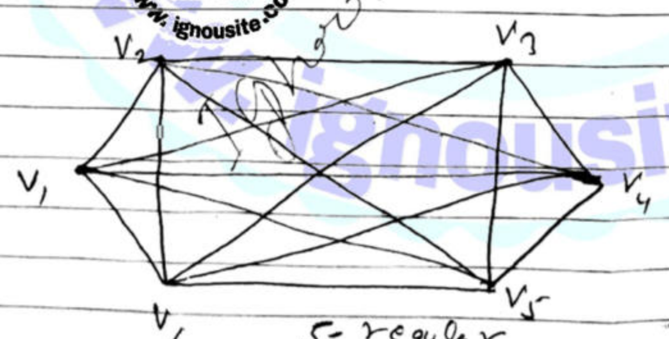


2-regular



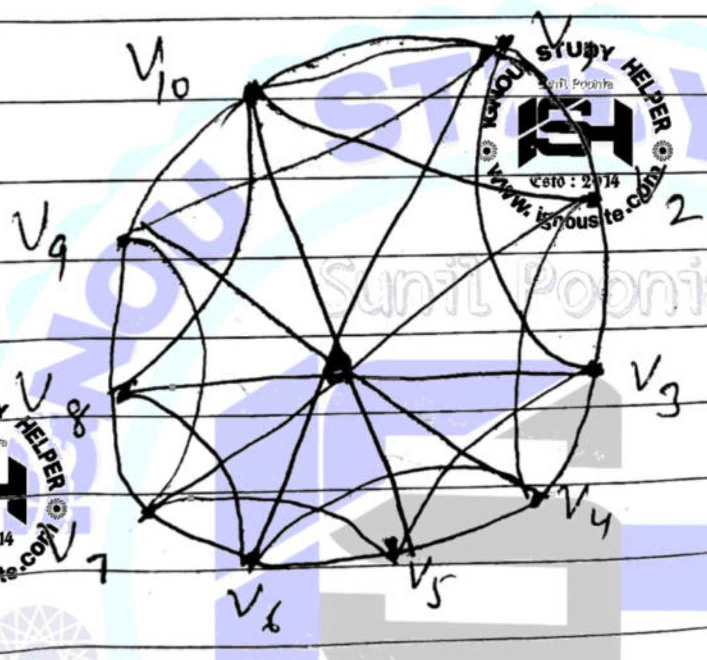
3. regular

4-regular



5- regular

5-regular graph with 10 vertices →



Q10. Solve the recurrence $a_n = a_{n-1} + 2$; $a_0 = 3$

Ans.

Given $a_n = a_{n-1} + 2, \quad a_0 = 3$

we successively apply recurrence formula :-

$$a_n = a_{n-1} + 2$$

$$\Rightarrow (a_{n-2} + 2) + 2 = a_{n-2} + 4$$

$$\text{or } a_{n-2} + 2.2$$

$$\Rightarrow (a_{n-3} + 2) + 2.2 \Rightarrow a_{n-3} + 6 = a_{n-3} + 2.3$$

$$\vdots$$

$$(a_{n-n} + n) + n.n \Rightarrow a_0 + 2n$$

$$= 1 + 2n$$