Q.2. Comment on P = NP. **3M**

Ans. P.6-7, Q.6

Q.3. Explain polynomial reduction. **3M**

Ans. P.6-11, Q.16

Q.4. How can polynomial reduction be used for showing NP-completeness of a problem? **3M**

Ans. P.6-11, Q.16

WINTER - 13 (CS)

Q.1. Explain the following terms :

(i) Non-deterministic algorithm.

(ii) Decision and optimization problem.

(iii) NP-hard problem.

(iv) NP-complete problem. **8M**

Ans. P.6-6, 8, 9, Q.3, 9 & 12

Q.2. Explain polynomial reduction. Explain how polynomial reduction can be used to solve NP-C problems. **6M**

Ans. P.6-11, Q.16

SUMMER - 14 (CT)

Q.1. Write short notes on :

(i) P and NP problems. **4M**

(ii) NP-complete and NP-hard problems. **3M**

(iii) Deterministic and Non-deterministic algorithms. **3M**

(iv) Decision and optimization problems. **3M**

Ans. P.6-6, 8, 9, Q.2, 3, 9 & 12

SUMMER - 14 (CS)

Q.1. Prove that P $\subseteq$ NP. **3M**

Ans. P.6-7, Q.7

Q.2. Write algorithm for Non-deterministic sorting. **4M**

Ans. P.6-9, Q.11

SOLVED QUESTION BANK

[Sequence given as per syllabus]

INTRODUCTION

NP-completeness is an indication of a difficult situation; it shows that many important problems cannot be solved fast enough.

NP-complete problems possess characteristics such as linear lower bound known as polynomial time algorithm.

The algorithmic gap for the problem class NP-complete is extremely very large. A growing number of problems in diverse fields are found to be members of the class NP-C.

The fields covered are as wide as combinatorics, graph theory, game theory, logic, management and economics.

Around 1971, Cook and Levin developed the idea of NP-completeness. Soon after, Karp showed 24 NP-complete problems of all shapes and colors.

NP-HARD AND NP-COMPLETE PROBLEMS- BASIC CONCEPTS

Q.1. Explain the term polynomial time. *CT : S-11(1M)*

OR Give the basic concepts of NP-hard and NP-complete problem.

OR Explain in detail the relationship between P, NP, NP-complete and NP-hard with the help of diagram. *CS : S-12(7M)*

Ans.

- There are many problems for which no polynomial time algorithm is known. Some of these problems are travelling salesman, optimal graph coloring knapsack, hamiltonian cycle.
- These problems belongs to an interesting class of problems called the "NP-complete" problems, whose status is unknown.
- The NP-complete problems are tractable i.e. require a super polynomial time. The reason is that a polynomial time algorithm to

solve any one of the NP-complete problems would automatically provides us with a polynomial time algorithm for all of them.

Polynomial time algorithm :

- Polynomial time is a reduction which is computable by a deterministic turing machine in polynomial time.
- Algorithms with worst case running time of $O(n^k)$, where k is a constant, are called tractable or super polynomial.
- An algorithm is a polynomial time algorithm if there exists a polynomial P(n) such that the algorithm can solve any instance of size n in a time $O(P(n))$.
- Problem requiring $\Omega(n^{35})$ time to solve are essentially intractable for large n.
- Undecidable problems are halting problem of turning machine.

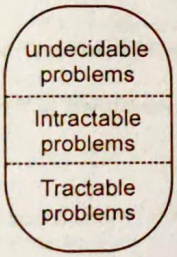


Fig.(a) Classification of problems

- The advantages in considering the class of polynomial time algorithms is that all reasonable deterministic single processor model of computation can be simulated on each other with at most a polynomial slowdown.

Classification of problems :

- The subject of computational complexity theory is dedicated to classify problems by how hard they are.
- There are many classification some of the most common and useful are the following :

(1) **P :**

- "P stands for polynomial" problems that can be solved in polynomial time.
- More specifically, they are the problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem.

(2) **NP :**

- "NP stand for non-deterministic polynomial time". Where non-deterministic is just a fancy way of talking about guessing a solution.
- A problem is in NP if we can quickly test whether a solution is correct.

(3) **PSPACE :**

Problems that can be solved using a reasonable amount of memory without regard to how much time the solution takes is known as PSPACE problems.

(4) **EXPTIME :**

- These are the problems that can be solved in exponential time.
- This class contains most problems we are likely to run into including everything in the previous three classes.

(5) **Undecidable :**

- For some problems, we can prove that there is no algorithm that always solves them, no matter how much time or space is allowed. These are called the undecidable problem.

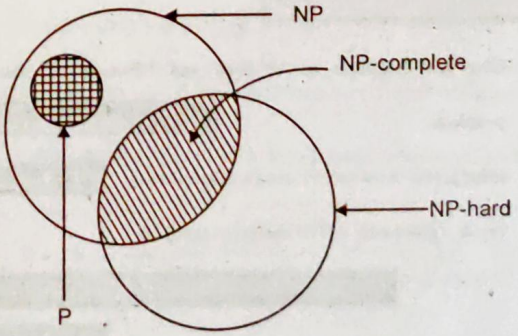


Fig.(b) Relationship among P, NP, NP-complete and NP-hard

- From this figure it is easy to see that there are NP-hard problems that are not NP-complete.
- Only a decision problem can be NP-complete. However an optimization problem may be NP-hard.

Q.2. Explain P and NP with example. *CT : S-10(3M), S-14(4M)* *CS : W-11(3M), S-13(1M)*

Ans. P :

- We define P as a class of decision problem that are solvable by algorithm that run in time polynomial in the length of the input i.e. a decision question if there exists an exponent k and an algorithm for the question that runs in time $O(n^k)$ where n is the length of input.

Class P :

- The set of all polynomially solvable problems comes under class P.
- P is closed under addition, multiplication and composition.
- P is independent of particular formal modes of computation.
- Any problem not in P is hard.
- Any problem in P does not necessarily have an efficient algorithm.

NP :

- The class NP consists of set of all problems that can be solved if we always guess correctly what computation path we should follow.
- It includes problems with exponential algorithm but has not proved that they cannot have polynomial time algorithm.

NP-HARD AND NP-COMPLETE

Q.3. Give the definition of NP-hard and NP-complete class of problems. *CT : S-06,13(3M), W-13(4M)*

OR What are NP-hard and NP-complete problems? *CT : W-06(2M)*

OR Write a short note on NP complete problem.

CT : S-07,W-07,09(7M), W-08,10,(2M), S-11(1M) *CS : W-13(2M)*

OR Write a short note on NP-hard problem.

CT : S-07, W-07,09(6M), S-08(2M) W-08,10(2M), S-11(1M), CS : W-13(2M)

OR Explain NP-complete and NP-hard.

CT : S-10(3M), CS : W-11(3M)

OR Explain following terms :

(i) NP-hard problem.

(ii) NP-complete problem.

CT : S-09, W-12(6M), S-14(3M)

OR Define the following terms with suitable examples : NP-C and NP-H. Explain the methodology to derive a solution for NP-H domain. *CS : S-13(7M)*

Ans. NP-hard :

- If a language L satisfies property 2 of NP-complete, but not necessarily property 1, we say that L is NP-hard.
- It means that if there is a problem X such that every problem in NP reduces to X, then X is atleast as hard as every problem in NP. We say that X is hard for NP or NP-hard.
- Once we have one problem that is NP-hard, we can use it to find others, for example by reducing X to Y, to show that Y is also NP-hard.
- Consider a problem X in NP-hard if there is an NP complete problem Y that can be polynomially turing reduced to it : $Y \leq_T^P X$. By definitions of polynomial reductions any polynomial time algorithm for X would translate into one for Y. Since Y is NP accepted belief.
- Therefore no NP-hard problem can be solved in polynomial time in the worst case under the assumption that $P \neq NP$.

NP-complete :

- If X is in NP and is also NP-hard then we say that X is complete for NP or is NP-complete.
- If any NP-hard problem is in P that every problem reduces to it is also in P. There are pro-blems that are NP-hard but not known to be in NP. So we usually restrict ourselves to NP-complete problems.
- All NP-complete problems have the same hardness.
- If any NP-complete problem is in P, then P = NP.

- Polynomial time reductions provide a formal means for showing that one problem is atleast as hard as another, to within polynomial time factor. That is if $L_1 \leq_P L_2$, then L_1 is not more than a polynomial factor harder than L_2, which is why the "less than or equal to" notation for reduction NP-mnemonic.
- We can now define the set of NP-complete languages, which are hardest problem in NP as follows :

A language $L \subseteq \{0,1\}^*$ is NP-complete if it satisfies the following two properties :

(i) $L \in NP$;

(ii) For every $L' \in NP$, $L' \leq P^L$

- If a language L satisfies property 2, but not necessarily property 1, we say that L is NP-hard.
- We use the notation $L \in NPC$ to denote that L is NP-complete.

Q.4. Explain the necessity of the study of the theory of NP-completeness. ***CT : W-06(3M)***

Ans.

- The NP-complete problems are quite common knowing that these are hard to solve, one may do well to stop trying to solve them exactly and reconcile to an almost correct or estimated solution which are as follows :

(i) Use of a heuristic.

(ii) Solve the problem approximately.

(iii) Accept the exponential time behavior.

(iv) Select a better abstraction.

- NP-complete problems are the hardest problems in NP, if anyone finds a polynomial time algorithm for even one NP-complete problem.
- For better abstraction there is necessity of the study of the theory of NP-completeness because of the ability to quickly verify solutions to a problem.

Q.5. Differentiate between NP-Hard and NP-Complete.

CT : W-11(4M), CS : S-12(2M)

Ans.

Sr.No.	NP-hard	NP-complete
(1)	The input in NP-hard problem is undefined.	The input in NP-complete problem is not fixed.
(2)	The process is undefined.	The process is not fixed.
(3)	Time is unknown.	Time is not fixed.
(4)	Not all NP-hard problems are NP-complete.	All NP-complete problems are NP-hard.
(5)	NP-hard problem that is in a certain sense are atleast as difficult to solve as any other NP-problem.	NP-complete problems are toughest problems that are in NP.

Q.6. Comment on P = NP. ***CT : S-06,13, W-13(3M), S-12(4M)***

OR Justify the statement "Satisfiability is in P if and only if P = NP".

CS : S-12(6M)

Ans.

- One of the most important problems in computer science is whether P = NP or $P \neq NP$? observe that $P \subseteq NP$.
- Given a problem $A \in P$ and a certificate, to verify the validity of a yes-input we can simply solve A in polynomial time since $(A \in P)$. It implies $A \in NP$.
- If any NP-complete language can be decided in polynomial time, then NP = P.
- If any language in NP cannot be decided in polynomial time, then all NP-complete languages cannot.

Q.7. Prove that $P \subseteq NP$. ***CS : W-10, 12, S-14(3M)***

Ans. $P \subseteq NP$:

- There is a polynomial time algorithm B that decides L.

- Define A(x , y) = B (x) for any x belongs to {0, 1}* and for any certificate y. Algorithm A (x, y) ignores y and accepts those x belongs to L.
- Polynomial time algorithms are defined as fixed input and process with fixed time, hence it is justified that P ⊆ NP

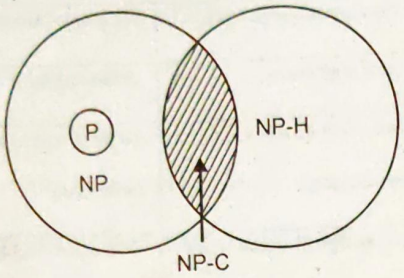


Fig. P⊆ NP

Q.8. Differentiate between P and NP. *CS : W-10(3M)*

Ans.

Sr. No.	P	NP
(1)	In polynomial time the input is fixed.	In NP the input is not fixed.
(2)	The process is fixed.	The process is not fixed.
(3)	Time is fixed.	Time is not fixed.
(4)	P problems can be efficiently solve.	NP problems which, given a proposed solution, we can efficiently check if it works.
(5)	**Example :** Prime number is relatively easy problem to solve.	**Example :** Travelling salesman problem.

DETERMINISTIC AND NON-DETERMINISTIC ALGORITHMS

Q.9. Write short note on deterministic and Non-deterministic problem. *CT : W-08, (2M), W-10, S-14(3M), S-11(1M)* *CS : W-13(2M)*

Ans.

- The algorithm where result is uniquely defined is termed as the deterministic algorithms. Such algorithms agree with the way programs are executed on a computer.
- In a theoretical framework we can remove this restriction on the outcome of every operation. We can allow algorithms to contain operations whose outcomes are not uniquely defined but limited to specified set of possibilities.
- The machine executing such operations is allowed to choose anyone of the outcomes subject to a termination condition.
- This leads to the concepts of a non-deterministic algorithm.
- To specify such algorithms, we introduce three new functions :

(i) Choose (s) : arbitrarily chooses one of the elements of set s.

(ii) failure( ) : signals a unsuccessful completion

(iii) success( ) : signals a successful completion.

- A non-deterministic algorithm terminates un-successfully if and only if there exists no set of choices leading to a success signal.
- The computing time for choice, success and failure are taken to be O(t).

Q.10. Differentiate between deterministic and non-deterministic polynomial time algorithm. *CT : W-11(5M)*

Ans.

Sr. No.	Deterministic polynomial time algorithm	Non-deterministic polynomial time algorithm
(1)	The algorithm where result is uniquely defined is termed as the deterministic algorithm.	The algorithm which terminates unsuccessfully if and only if there exists no set of choices leading to a success signal is known as non-deterministic algorithm.

(2)	In deterministic algorithm given input will always produce the same output.	Algorithm is non-deterministic if there are more than one path the algorithm take. Due to this one cannot determine the next state of the machine.
(3)	**Example :** Mathematical function is deterministic.	**Example :** Random function is non-deterministic.
(4)	It has a unique value.	It has different values.
(5)	It has a single outcome.	It has multiple outcomes.

Q.11. Write an algorithm to implement non-deterministic sorting algorithm. Find out the complexity of the same.

CS : S-11, W-11 (4M)

OR **Write algorithm for Non-deterministic sorting.**

CS : W-12 (3M), S-14(4M)

Ans. Non-deterministic sorting :

- Select n position from the given array.
- Create a temporary array using the elements present at the selected position.
- Check the temporary array for sorted sequence.

Algorithm :

```
Algorithm ND-sorting (a, n)
  {
    for i=1 to n do
      {
        j=select (a, n);
        a[i] = a[j];
      }
      for i=1 to n–1 do
      {
        if (b[i] > b[i+1]) then
            failure( );
      }
      if (i = n)
        success( );
  }
```

complexity : $O(n)$

DECISION AND OPTIMIZATION PROBLEMS

Q.12. Write short note on decision and optimization problems.

CT : W-06, S-08(2M), S-14(3M)

OR **Explain the terms decision and optimization problems.**

CT : W-08, S-11(2M), CS : W-13(2M)

Ans. Decision problem :

- Any problem for which the answer is either zero or one or equivalently either true or false is called as decision problem.

  Example : "Find a Hamiltonian cycle in graph G" is not decision problem, but Is graph G is Hamiltonian?" is a decision problem.
- The class NP-corresponds to the decision problems, that have an efficient proof system, which means that each yes instance must have atleast one certificate whose validity can be verified quickly.
- The theory of NP-completeness is concerned with the notion of polynomially verifiable properties.
- Intuitively, a decision problem X is polynomial time verifiable if some one could show that x is X wherever this is so.
- Given this example (x in X) one should be able to verify in polynomial time that indeed x in X. However in fact x not in X, then one should not falsely show that x in X.

Optimization problem :

- Any problem that involves identification of an optimal value of given cost function known as optimization problem.
- An optimization algorithm is used to solve an optimization problem.
- The problem in which some value must be minimized or maximized, optimization problems can be recost in terms of decision problem by placing a bound on value to be optimized.

Example : In recosting the shorted path problems as the decision.

Q.13. Differentiate between decision problem and optimization problems. *CT : W-11(5M), CS : S-12(3M)*

Ans.

Sr. No.	Decision problem	Optimization problem
(1)	Any problem for which the answer is either zero or one/ true or false is called as decision problem.	Any problem that involves identification of an optimal value of given cost function is known as optimization problems.
(2)	Decision problem asks us to check if something is true i.e. 'Yes' or 'No'.	An optimization problem asks us to find, among all feasible solutions, are that maximizes or minimizes a given objective.
(3)	**Example :** Prime number.	**Example :** Single source shortest path.
(4)	If we provide evidence that a decision problem is hard, we also provide evidence that its related optimization problems is hard.	If an optimization problem is easy then its related decision problem is easy as well.
(5)	Decision problem is easier than optimization problem.	Optimization problem is harder than decision problem.

Q.14. What is efficient certifications?

Ans. Efficient certifications :

- Certification algorithm means "Design an algorithm that check whether proposed solution is a yes instance"
- Any algorithm C is an efficient certifier to X if :

(i) C is a polynomial time algorithm that takes two input s and t.

(ii) There exists a polynomial P( ). So that for every string s, $s \in X$ such that there exist a string t such that $|t| \leq P(|s|)$ and C(s, t) = yes .

Example :

COMPOSITE - Given integer S, is composite?

Observation - s is composite.

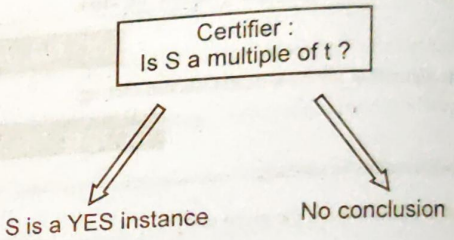
- It is implied that there exists an integer $1 < t < s$ such that s is a multiple of t.
- Yes instance : $s = 437, 669$

Certificate : $t = 541$ or 809

- No instance : $s = 437, 677$

No witness can fool verifier into saying yes.

input s :	Certificate t :
437, 669	541



- Therefore now we have conclusion that COMPOSITE $\in$ NP .

COOK'S THEOREM

Q.15. Write short note on Cook's theorem. *CT : S-08(3M)*

OR State and explain Cook's theorem. Also explain its importance. *CT : S-10(8M), S-12(4M)*

Ans. Cook's theorem :

- Cook's theorem proves that satisfiability is NP-complete by reducing all non-deterministic turning machine to SAT.
- Each turing machine has access to a two way infinite tape (read/ write) and a finite state control, which serves as a program.

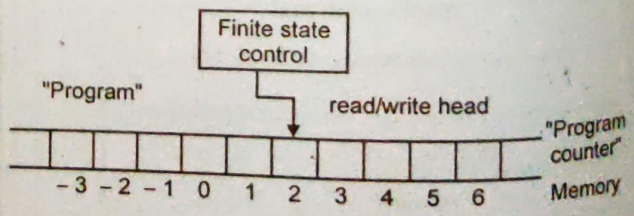


Fig.(a)

A program for a non-deterministic TM is :

(a) Space on the tape for guessing a solution and certificate to permit verification.

(b) A finite set of tape symbols.

(c) A finite set of states $\emptyset$ for the machine, including the start state q_0 and final state Z_{yes}, Z_{no}.

(d) A transition function, which takes the current machine state and current tape symbol and returns the new state, symbol and head position.

- We know a problem is in NP if we have a NDTM program to solve it in worst case time $p^{[n]}$, where P is a polynomial and n is the size of the input.

"Cook's theorem satisfiability is NP-complete."

Proof :

- We must show that any problem in NP is atleast as hard as SAT.
- Any problem in NP has a non deterministic TM program which solves it in polynomial time, specifically P(n).
- We will take this program and create from it an instance of satisfiability such that it is satisfiable if and only if the input string was in the language.

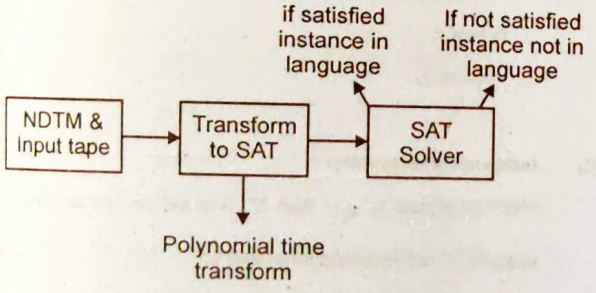


Fig.(b)

- If a polynomial time transform exist, then SAT must be NP-complete, since a polynomial solution to SAT gives a polynomial time algorithm to anything in NP.
- Since a polynomial time algorithm for SAT would imply a polynomial time algorithm for everything in NP. SAT is NP-hard since we can guess a solution to SAT, it is in NP and thus NP-complete. Therefore, now Cook's theorem has been proved.

POLYNOMIAL REDUCTION

Q.16. Explain polynomial reduction.

CT : S-06,13, W-13(3M), W-10(4M)

OR How polynomial reduction can be used for showing NP-completeness of a problem? **CT : S-06,13(4M), W-13(3M)**

OR Explain clearly polynomial reduction and how it can be used to show that a problem is NP-complete.

CT : W-06, S-08,12(6M), W-08(5M), S-09,11, W-12(7M)

CS : W-13(6M)

Ans.

- Let A and B be two problems, we say that A is polynomially turning reducible to B if there exists an algorithm for sorting A in a time that would be polynomial if we could solve arbitrary instance of problem B at unit cost. This is denoted as

$$A_T = <^P_B$$

- Note that if $A_T = <^P_B$ and $B_T = <^P_A$ then $A = =^P B_T$ is called polynomial turning equivalent and polynomial reductions are transitive if :

$$A_T = <^P_B \text{ and } B_T = <^P_A$$

$$A_T = <^P_C$$

- A problem A can be reduced to another problem B if any instance of A can be "rephrased" as an instance of B, the solution to the instance of B provides a solution to the instance of A.

Polynomial reduction in NP-complete problem :

- Let L and L' be two languages and L' $\in$ NP-complete. If $L' \leq P^L$. Then L is NP-hard. If in addition L is NP, then L is NP-complete.
- Let f be the polynomial time function that reduces L_1 to L_2 and A, a polynomial time algorithm that decides L_2. Then for any x, run A on y = f(x). If it accepts y, then $x \in L_1$; otherwise $x \notin L_1$. Since the decision on x is done in polynomial time $L_1 \in P$.

PROBLEMS BASED ON GRAPH

Q.17. Explain the following NP problems and relationship between them.

(i) Clique

(ii) Graph partitioned into triangle.

(iii) Independent set problem. **CS : W-10(7M)**

OR Explain the following NP problems with respect to graph :

(i) Clique

(ii) Graph partitioned into triangle.

(iii) Independent set problem.

Write an algorithm for clique and modify it to solve the remaining problem. **CS : S-11, W-11(9M)**

OR Write an algorithm to find clique from the given graph. Explain how the algorithm can be modified for graph partitioned into triangle problem. **CS : W-12 (7M)**

Ans. There are two ways to solve the given problem.

(1) Solve the given problem from scratch.

(2) Find a problem of similar category and reduce the given problem to the problem which is already solved.

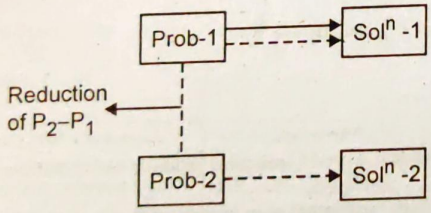


Fig. (a)

Problems based on graph are as follows :

(1) Clique

(2) Independent set problem.

(3) Graph partitioned into triangle (GPT).

(1) Clique algorithm :

It is a part of graph which is considered as complete

$G(V, E) \rightarrow [V_1 \subseteq V]$ such that $G(V_1)$ = complete

Assumption :

G represents a graph of n vertices and it is required to find out clique of k vertices

$n = 7, k = 3$ (size of clique)

Condition :

The vertices selected should represent complete graph <connectivity>.

Method :

Using select function k vertices from the graph and generate a set representing k vertices

$s = < 2, 4, 2 >$

After selection test the connectivity for the vertices present in s.

```
Algorithm clique (G, n : k, s)
  {
    s =NULL;
    for i=1 to k do
    {
      j=select(G, n);
      if(j∉s)then
        s=sU{j};
    }
  for each pair of vertices (i, j) && (i=j) or
    (G[i, j] ≠ 0)
  failure( );
  success( );
  {
```

(2) Independent set problem :

If G = (V, E) and $V' \subseteq V$ then V' is an independent set if no two nodes in V' are connected by an edge is E.

Given a graph G (V, E) ISP will find averted V is such that

(i) The vertices present in V_1 are not connected with each other.

(ii) The vertices present in V_1 are independently connected to all the remaining vertices of the graph.

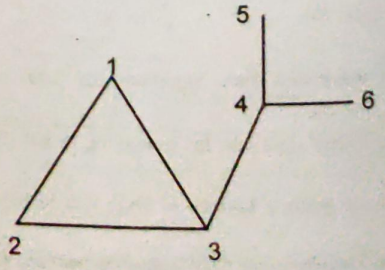


Fig. (b)

Modify ⇒ ISP (1, 2, 3)

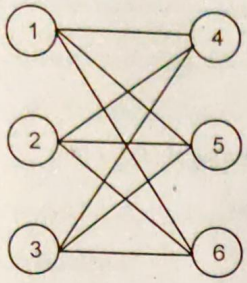


Fig. (c)

```
Algorithm ISP (G, n; k, s)
{
    s = NULL;
    for i=1 to k do
    {
        j=select(G, n);
        if(j∉s)then
          s=sU{j};
    }
for each pair of vertices (i, j) such that
    {
        (i ∈s) and j ∈(v – s) OR G[i, j] = 0) OR #
        failure( );
        success( );
                        [# '. For each pair of vertices (k, L) ∈ s
          such that (G [k, L] = L) and (k ≠ L)]
    }
}
```

(3) **Graph partitioned into triangle :**

Given a graph GPT will find a subset V, which belongs to V

$[V_1]$ = No. of vertices present = (3)

G (V_1) = complete

The problem of graph partitioning arises in a diversity of applications, from circuit layout to program analysis of image segmentation.

Input : An undirected graph G = (V, E) with non-negating.

Output : A partition of the vertices into two groups A and B, each of size atleast $\propto |V|$.

Goal : Minimize the capacity of the cut (A, B).

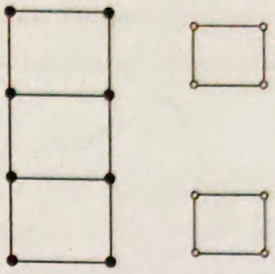


Fig. An instance of graph partitioning with the optimal partition $\propto = 1/2$. Vertices on one side of the cut are shaded

COMPUTATIONAL GEOMETRY

Q.18. Explain the computational geometry. Write the algorithm for finding the convex hull with its complexity. ***CS : S-12 (8M)***

Ans. Computational geometry :

- Computational geometry is a term claimed by a number of different groups.
- The term was coined perhaps first by Marvin Minsky in his book "Perceptrons" which was about pattern recognition and it is also been used often to describe algorithms for manipulating curves and surface in solid modeling.
- In computer science computational geometry is the study of algorithms to solve problems stated in terms of geometry.
- The input to computational geometry is a description of a set of geometric object such as a set of points, a set of line segments or the vertices of a polygon in counterclockwise order.
- The output is often a response to a query about the objects, such as whether any of the lines intersect of a new geometric object such as the convex hull of the set of points.
- One of the goals of computational geometry is to provide the basic geometric tools needed from which application areas can build their programs.

Polygon :

- Polygon is just a collection of line segments, forming a cycle and not crossing each other.
- We can represent it as a sequence of points, each of which is just a pair of coordinates.

Convex polygon :

- A figure is convex if every line segments drawn between any two points inside the figure lies entirely inside the figure.

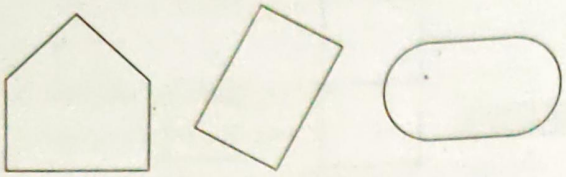


Fig. Polygon : Convex polygon, convex hulls

Convex Hull :

- A convex hull is an important structure in geometry that can be used in the construction of many other geometric structures.
- The convex hull of a set s of points in the plane is defined to be the smallest convex polygon containing all the points of s.

Algorithm for finding the convex hull :

- There are two algorithms that compute the convex hull of a set of n points.
- Both algorithms output the vertices of the convex hull in counterclockwise order.
- The first known as Graham's scan runs in O (n log n) time.
- The second called Jarvis's march, runs in O(nh) time, where h is the number of vertices of the convex hull.

GRAHA SCAN (Q) :

(1) Let P_0 the point in Q with the minimum y-coordinate, or the leftmost such point in case of a tie.

(2) Let $< P_1, P_2, ... P_m >$ be the remaining points in Q, sorted by polar angle in counter clockwise order around P_0.

(3) top[s] $\leftarrow$ 0.

(4) PUSH (P_0, s)

(5) PUSH (P_1, s)

(6) PUSH (P_2, s)

(7) For $i \leftarrow 3$ to m

(8) do while the angle formed by points NEXT-TO-TOP(S), TOP(S), and P_i makes a nonleft turn

(9) do POP(s)

(10) PUSH (s, P_i)

(11) return s

Complexity :

$$T(n) = O(n) + O(n \log n) + O(1) + O(n)$$

$$= O(n \log n)$$

Where $n = |Q|$

Q.19. Give strengths and limitations of computational geometry.

Ans.

Strengths of computational geometry :

(1) Development of geometric tools.

(2) Emphasis on provable efficiency.

(3) Emphasis on correctness /robustness.

(4) Linkage to discrete combinational geometry.

Limitations of computational geometry :

(1) Emphasis on discrete geometry.

(2) Emphasis on flat objects.

(3) Emphasis on low-dimensional spaces.

RANDOMIZED ALGORITHM

Q.20. Explain randomized algorithm and its types.

Ans. Randomized algorithm :

- Algorithms where some of the actions are dependent on chance are generally termed as probabilistic or randomized algorithms.
- For many problems randomized algorithm run faster than the known best deterministic algorithms.
- These algorithms are simpler to describe and implement than the deterministic algorithms of comparable performance.
- The result of randomized algorithm may not always be hundred percent correct. Types of randomized algorithm are shown in figure.

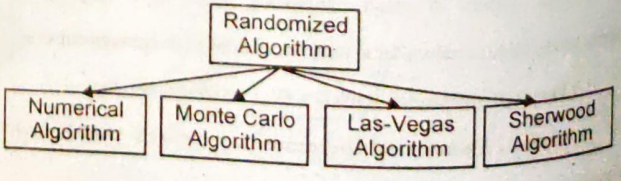


Fig. Classification of randomized algorithm

(1) **Numerical algorithms :**

- For some problem, finding the exact solution requires a long time, randomness can be used to find an approximate numerical solution of such problem.
- It is difficult to find the average length of a queue in closed form in a complex system. But an approximate value of the average queue length can be estimated through simulation using randomness and this result will improve, the more time we run the simulation.

(2) **Monte-Carlo-Algorithm :**

- There are some problems for which we need to find out the exact answer (yes/no). An approximate answer is meaningless far such type of problems.

Example : In decision problems, the answer is either 'yes' or 'no' there is no scope for an approximate answer for these problems.

- The probabilistic algorithms that are used to solve these kinds of problems are called Monte-Carlo algorithms.
- A Monte Carlo algorithm always gives answer but the answer is not necessarily correct.

(3) Las Vegas algorithms :

Las Vegas algorithm may or may not find an answer but whenever it find answer the answer is correct.

(4) Sherwood algorithms :

- These algorithms are used when we have a deterministic algorithm that shows different performances for the worst case and the average case.
- The Sherwood algorithms reduce the gap in the time performance of the worst case behaviour and average case behaviour.
- The Sherwood algorithms always give an answer and the answer is correct.

POINTS TO REMEMBER :

(1) ***NP-complete :*** *Informally, a problem is in the class NPC and we refer to it as being NP-complete if it is in NP and is as "hard" as any problem in NP. A language* $L \subseteq \{0, 1\}^*$ *is NP-complete if*

- $L \in NP$
- $L' \leq P^L$ *for every* $L' \in NP$.

(2) ***NP-hard :*** *If a language L satisfies property 2 of NP-complete, but not necessarily property 1, we say that L is NP-hard.*

(3) ***Polynomial time algorithms :*** *An algorithm with integer injects* $a_1, a_2, \ldots a_k$ *is a polynomial in* $lg\, a_1, lg\, a_2, lg\, a_3, \ldots lg\, a_k$ *that is binary encoded inputs.*

(4) *The algorithm where result is uniquely defined is termed as the deterministic algorithm.*

(5) *The algorithm that contains operations whose outcomes are not uniquely defined but limited to specified set of possibilities is termed as non-deterministic algorithm.*

(6) *Any problem for which the answer is either zero or one or equivalently either true or false is called as decision problem.*

(7) *Any problem that involves identification of an optimal value of given cost function is known as optimization problem.*

(8) *In computer science computational geometry is the study of algorithms to solve problems states in terms of geometry.*

(9) *Algorithms where some of the actions are dependent on chance are generally termed as randomized algorithms.*