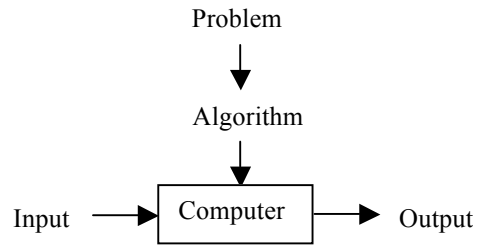
UNIT-I / PART-A	
1	**Define Algorithm.** ***(June 06,07,May 13)*** An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
2	**Define order of an algorithm.** ***(June 07)*** The order of an algorithm is a standard notation of an algorithm that has been developed to represent function that bound the computing time for algorithms. The order of an algorithm is a way of defining its efficiency. It is usually referred as Big O notation.
3	**Write about Notion of Algorithm.** 
4	**What are the characteristics of an algorithm?** ✓ Input ✓ Output ✓ Definiteness ✓ Finiteness ✓ Effectiveness.
5	**What are the different criteria used to improve the effectiveness of algorithm?** ***(June 06,07)*** ✓ Input – Zero or more quantities are externally supplied. ✓ Output – At least one quantity is produced. ✓ Definiteness – Each instruction is clear and unambiguous. ✓ Finiteness – if we trace out the instruction of an algorithm, then for all cases, the algorithm terminates after a finite number of steps. ✓ Effectiveness – Every instruction must be very basic.
6	**What are the features of efficient algorithm*?(Dec 07)*** ✓ Free of ambiguity ✓ Efficient in execution time ✓ Concise and compact ✓ Completeness ✓ Definiteness ✓ Finiteness
7	**What are the steps involved in designing and analyzing an algorithm?** ✓ Understand the problem ✓ Decide on ❖ Computational Device ❖ Exact Vs Approximate Algorithms ❖ Data Structures ❖ Algorithm Design Techniques ✓ Design an algorithms ✓ Prove Correctness ✓ Analyze the Algorithm ✓ Code the Algorithm
8	**What is an algorithm design techniques?** ***(Dec 06)*** An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.

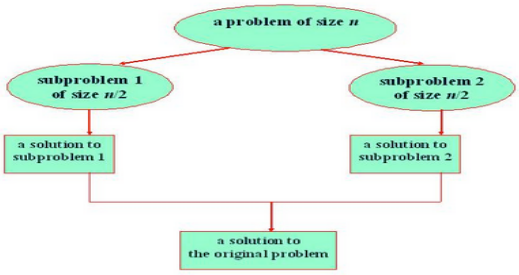
9	**Define Pseudo Code** Pseudo Code is a mixture of a natural language and programming language - like constructs such as functions, loops, and decision making statements. A pseudo is usually more precise than a natural language.
10	**Mention different algorithm design techniques.** ✓ Methods of specifying an algorithm ✓ Proving an algorithms correctness ✓ Analyzing an algorithm ✓ Coding an algorithm
11	**Mention the technique for proving correctness of an algorithm.** A common technique for proving correctness is to use mathematical induction because algorithms iterations provide a natural sequence of steps needed for such proof.
12	**What are the kinds of algorithm efficiency?** ✓ Time efficiency ✓ Space efficiency
13	**Define time efficiency.** Time efficiency indicates how fast the algorithm runs.
14	**What is space efficiency?** Space efficiency indicates how much extra memory the algorithm needs.
15	**Mention the desirable characteristics of an algorithm.** ✓ Simplicity ✓ Generality ✓ Optimality
16	**Mention the most important problem types.** ***(Dec 07,Apr 08)*** ✓ Sorting ✓ Searching ✓ String processing ✓ Graph problems ✓ Combinatorial problems ✓ Geometric problems ✓ Numerical problems
17	**Mention the sorting problems.** The sorting problems ask us to rearrange the items of a given list in ascending order.
18	**Mention the two properties of sorting algorithms.** ✓ A sorting algorithm is called stable if it preserves the relative order of any two equal elements in its input. ✓ An algorithm is said to be in place if it does not require extra memory.
19	**Mention the searching problems.** The searching problem deals with finding a given value, called a search key, in a given set.
20	**State travelling salesman problem (TSP).** The travelling salesman problem is the problem of finding the shortest tour through n cities that visits every exactly once.
21	**State graph-coloring problem.** The graph-coloring problem asks us to assign the smallest number of colors to vertices of a graph so that no two adjacent vertices are the same color.
22	**Give examples of combinational problems.** ✓ Travelling salesman problem ✓ Graph-coloring problem
23	**State combinatorial problems.** These are the problems that ask(explicitly or implicitly) to find a combinatorial object-such as a permutation, a combination, or a subset-that satisfies certain constraints and has some desired property (maximizes a value or minimizes a cost)

24	**State geometric algorithms.** Geometric algorithms deal with geometric objects such as points, lines, and polygons.
25	**Give examples of computational geometry.** ✓ Closet –pair problem ✓ Convex-hull problem
26	**State Closet –pair problem.** Given n points in the points in the plane, find the closet pair among them.
27	**State Convex-hull problem** To find the smallest convex polygon that would include all the points of a given set.
28	**Define basic operation in algorithm.** To identify the most important operation of the algorithm called the basic operation, the operation contributing the most to the total running time, and compute the number of times the basic operation is executed.
29	**What is the purpose of Instruction Space?** Instruction space is the space needed to store the compiled version of the program instructions.
30	**What is the purpose of Data Space?** Data space is the space needed to store all the constant and variable values.
31	**What is the component of Data Space?** ✓ Space needed by constants and simple variables. ✓ Space needed by component variables such as array. This category includes space needed by structures, and dynamically allocated memory.
32	**What is the purpose of Environment Stack Space?** The Environment stack space is used to save the information needed to resume execution of partially Completed methods. Examples: If F1 invokes F2, then we must save a pointer to the Instruction of F1 to be executed when F2 terminates.
33	**What are the factors that determine the amount of Instruction Space Needed?** ✓ The compiler used to compile the program into machine code. ✓ The compiler options in effect at the time of compilation. ✓ The target computer.
34	**Write down formula for Space Complexity Calculation?** ***(Dec 13)*** S (P) = c +Sp (instance characteristics). Where c is a constant that denotes the fixed part of the space requirements and Sp denotes the variable component.
35	**Write down formula for Time Complexity Calculation?** The Time T (P) taken by a program P is the sum of the compile time and the run time. T(p)=Compile time +Run time
36	**Write the methods used for Estimating Run Time?** ✓ Identify one or more key operations and determine the number of times these are performed .It is referred as operation count ✓ Determine the total number of steps executed by the program. It is referred as step count
37	**Define Program Step?** A program step is loosely defined to be a syntactically or semantically meaningful segment of a program for which the execution time is independent of the instance characteristics.
38	**Define Operation Count?** It is the method to estimate the time complexity of a program by selecting one or more operations, such as add, multiply and compare and to determine how many of each is done. The success of this method depends on our ability to identify the operations that contribute most to the time complexity.
39	**Define Step Count?** Step count determines the total no of steps executed by the program.
40	**Write the reason for determining operation count and step count.** Two important reasons to determine operation and step counts are ✓ To compare the time complexities of two programs that compute the same function ✓ To predict the growth in run time as the instance characteristic change.

41	**Define Asymptotic Notations.** The notation that will enable us to make meaningful statements about the time and space complexities of a program. This notation is called asymptotic notation.
42	**Mention the various Asymptotic Notations.*(Dec 06)*** ✓ Big Oh notation, ✓ Theta notation ✓ Omega notation ✓ Little Oh notation.
43	**Define Big 'O' Notation. *(June 06,May 13)*** f(n)= O(g(n)) iff positive constants c and n0 exist such that f(n)<= cg(n) for all n, n ≥ n_0. The definition states that the function f is at most c times the function g except possibly when n is smaller than n_0.
44	**When is the function f(n) said to be O(g(n))?** f(n)= O(g(n)) iff positive constants c and n_0 exist such that f(n)<= cg(n) for all n, n ≥ n_0.
45	**Define big'O' ratio theorem?** Let f(n) and g(n) be such that lim f(n)/g(n) exists . n->∝ f(n)=O(g(n)) iff lim f(n)/g(n)<= c for some finite constant c. n->∝
46	**Define Omega Notation. *(June 07)*** f(n)= Ω(g(n)) iff positive constants c and n0 exist such that f(n) ≥ cg(n) for all n, n ≥ n_0 . The definition states that the function f is at least c times the function g except possibly when n is smaller than n0. Here c is some positive constant. Thus g is a lower bound on the value of f for all suitably large n.
47	**Define Omega Ratio Theorem.** Let f(n) and g(n) be such that lim g(n)/f(n) exists . n->∝ f(n)= Ω (g(n)) iff lim gn)/f(n)<= c for some finite constant c. n->∝
48	**When is the function f(n) said to be Ω(g(n))?** f(n)= Ω(g(n)) iff positive constants c and n0 exist such that f(n) ≥ cg(n) for all n, n ≥ n_0 .
49	**Define Theta Notation. *(Dec 14)*** f(n)= Θ(g(n)) iff positive constants c1 and c2 and an n0 exist such that c1g(n)<=f(n)<= c2g(n) for all n, n ≥ n0 . The definition states that the function f lies between c1 times the function g and c2 times the function g except possibly when n is smaller than n0. Here c1 and c2 are positive constants. Thus g is both a lower and upper bound on the value of f for all suitably large n.
50	**When is the function f(n) said to be Θ (g(n))?** f(n)= Θ(g(n)) iff positive constants c1 and c2 and an n_0 exist such that c1g(n)<=f(n)<= c2g(n) for all n, n ≥ n_0 .
51	**Define Theta Ratio Theorem.** Let f(n) and g(n) be such that lim f(n)/g(n) and lim g(n)/f(n) exists . n->∝ n->∝ f(n)= Θ (g(n)) iff lim f(n)/g(n)<= c and lim g(n)/f(n) <=c for some finite constant c. n->∝ n->∝
52	**List the properties of asymptotic notations. *(June 06)*** ✓ Sum rule : O (f(n)) + O (g(n)) = O (max {f(n),g(n)}) ✓ Transitivity rule : f(n) = O(g(n)) and g(n) = O(h(n)) then f(n) = O(h(n)) ✓ Reflexivity: f(n) = O(f(n)) ✓ Any constant value is equivalent to O(1). ✓ Polynomial Rule: Let f(n) be any polynomial in n, with k being the highest exponent. f(n) = O(nk)

53	**Write the basic asymptotic efficiency classes.** | Class | Name | Example | |---|---|---| | 1 | constant | access array element | | $\log n$ | logarithmic | binary search | | n | linear | find median | | $n \log n$ | "n-log-n" | mergesort | | n^2 | quadratic | insertion sort | | n^3 | cubic | matrix multiplication | | a^n | exponential | generating all subsets | | $n!$ | factorial | generating all permutations |
54	**What is worst-case efficiency?** The worst-case efficiency of an algorithm is its efficiency for the worst-case input of size n, which is an input or inputs of size n for which the algorithm runs the longest among all possible inputs of that size.
55	**What is best-case efficiency?** The best-case efficiency of an algorithm is its efficiency for the best-case input of size n, which is an input or inputs for which the algorithm runs the fastest among all possible inputs of that size.
56	**What is average case efficiency?** The average case efficiency of an algorithm is its efficiency for an average case input of size n. It provides information about an algorithm behavior on a "typical" or "random" input.
57	**What are the classifications of algorithm?** ✓ Recursive algorithm ✓ Non-recursive algorithm
58	**What is recursive algorithm?** An algorithm is said to be recursive if the same algorithm is invoked in the body. An algorithm that calls itself is direct recursive. Algorithm A is said to be indeed recursive if it calls another algorithm, which in turn calls A.
	UNIT-I / PART-B
1	Describe briefly the notions of complexity of an algorithm. ***(June 07)***
2	Explain the fundamentals of algorithmic problem solving. (OR) Discuss briefly the sequence of steps in designing and analyzing an algorithm. ***(Dec 06)***
3	Explain the various asymptotic notations of an algorithm in detail. ***(Dec 07)***
4	Elaborate on how space and time complexities are calculated. Give Examples. **(*May 13)***
5	Explain the general framework for analyzing the efficiency of algorithm. (or) Discuss the fundamentals of analysis framework. ***(Dec 06, 07)***
6	Explain some of the important problem types used in the design of algorithm with an example. ***(Dec 06, 07)***
7	a) List out the properties of asymptotic notations or big O***.(Dec 14)*** b) Give a short account of basic efficiency classes ***(Dec 06)***
8	Explain the following terms ***(June 07)*** ✓ Searching ✓ Asymptotic notations ✓ Analysis framework ✓ Algorithm efficiency
9	Explain in detail about mathematical analysis of non recursive algorithm with suitable examples. ***(June 07)***
10	a) Discuss the general plan for analyzing the efficiency of non recursive algorithm. ***(Dec 07)*** b) Write an algorithm to find the number of binary digits in the binary representation of a positive decimal integer and analyze its efficiency. ***(Dec 07)***
11	Design a non recursive algorithm for computing the product of two n*n matrices and also find time efficiency of algorithm***.(Dec 06)***
12	Design a recursive algorithm to compute the factorial function F (n) =n! and derive the recurrence relation***.(Dec 06)***
13	Design a recursive algorithm to find the number of moves in tower of Hanoi problem and find the time of complexity. ***(Dec 013)***
14	a) Solve the recurrence equation of the Fibonacci Series. T(n) = T(n-1) +T(n-2) subject to T(0)=0, T(1) = 1**.**

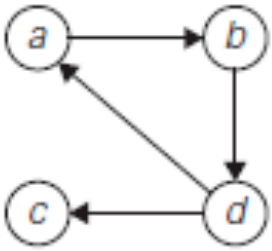
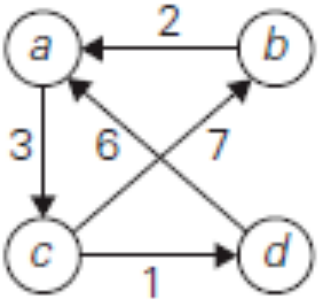
	(June 07,May 08) b) Write the general plan for analyzing time efficiency of non recursive algorithms and find the time complexity of element uniqueness problem. ***(Dec 07)***
UNIT-II / PART-A	
1	**What is Brute Force?** Brute Force is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved.
2	**What would be the most straightforward method for solving the sorting problem?** Selection sort and bubble sort are the two better algorithms and it implements brute-force approach more clearly.
3	**What are the advantages and disadvantages of brute force approach?** **Advantages:** ✓ applicable to a wide variety of problems ✓ for important problems it yields reasonable algorithms of at least some practical value with no limitation on instance size ✓ It can solve few instances of a problem with acceptable speed ✓ simple to design and may be useful to solve small problem instances **Disadvantage:** ✓ inefficient in general case
4	**When is sorting method said to be stable?** A sorting method is said to be stable when it has minimum number of swaps (less than 'n' number of comparisons) i.e. if the two data items of matching value are guaranteed not to be rearranged with respect to each other when the algorithm progresses.
5	**List out some of the stable and unstable sorting techniques.** **Stable sorting techniques includes** ✓ Bubble sort ✓ Insertion sort ✓ Selection sort ✓ Merge sort **Unstable sorting techniques includes** ✓ Shell sort ✓ Quick sort ✓ Radix sort ✓ Heap sort
6	**Define unstable sort.** A sorting method is said to be Unstable when it has maximum number of swaps (greater than 'n' number of comparisons) i.e. if the two data items of matching value are guaranteed not to be rearranged with respect to each other when the algorithm progresses.
7	**Define Closet –pair problem.** Given n points in the points in the plane, find the closet pair among them. Brute force approach
8	**Define Convex.** A set of points (finite or infinite) on the plane is called convex if for any two points P and Q in the set, the entire line segment with the end points at P and Q belongs to the set.
9	**Define Convex-Hull.** The Convex-Hull of a set S of points is the smallest convex set containing S. The smallest requirements means that the convex-hull of S must be a sub set of any convex set containing S.
10	**Define Convex-Hull Problem.** The problem of constructing the convex-hull for a given set S of n points.
11	**Define Extreme points.** The extreme point of a convex set is a point of set S that is not a middle point of any line segment with end points in the set.
	Define Exhaustive search.

12	Exhaustive search is simply a brute-force approach to combinatorial problems. It suggests generating each and every element of the problem's domain, selecting those of them that satisfy all the constraints, and then finding a desired element.
13	**Define travelling salesman problem.** Travelling salesman problem finds the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it is started.
14	**Define Hamiltonian circuit.** It is defined as a cycle that passes through all the vertices of a graph exactly once.
15	**Define Knapsack problem.*(Dec 14)*** Given n items of known weights $w_1 \ldots w_n$ and values $v_1 \ldots v_n$ and knapsack of capacity W. The aims is to find the most valuable subset if the items that fit into the knapsack. The exhaustive search approach to knapsack problem leads to generating all the subsets of the set of n items given, computing the total weight of each subset to identify feasible subsets and finding a subset of the largest value among them.
16	**Define assignment problem.** There are n people who need to be assigned to execute n jobs, one person per job. The cost that would accrue if the i^{th} person is assigned to the j^{th} job is a known quantity c [i,j] for each pair I, j=1,…,n. The problem is to find an assignment with the smallest total cost.
17	**Define merge sort.** The merge sort algorithm divides a given array A[0..n-1] by dividing it into two halves A[0.. $\lfloor n/2 \rfloor$ -1] and A[$\lfloor n/2 \rfloor$ -..n-1], sorting each of them recursively, and then merging the two smaller sorted arrays into a single sorted one. If the list has even length, split the list into two equal sub lists. If the list has odd length, divide the list in two by making the first sub list one entry greater than the second sub list. Then spilt both the sub lists in to two and go on until each of the sub lists are of size one. Finally start merging the individual sub lists to obtain a sorted list.
18	**Define divide and conquer design technique.** ***(May 13)*** ✓ A problem's instance is divided into several smaller instances of the same problem, ideally of about the same size. ✓ The smaller instances are solved ✓ If necessary, the solutions obtained for the smaller instances are combined to get a solution to the original instance. 
19	**Define quick sort.** Quick sort employs a divide-and-conquer strategy. It starts by picking an element from the list to be the "pivot." It then reorders the list so that all elements with values less than the pivot come before the pivot, and all elements with values greater than the pivot come after it (a process often called "partitioning"). It then sorts the sub-lists to the left and the right of the pivot using the same strategy, continuing this process recursively until the whole list is sorted.
20	**What is a pivot element?** The pivot element is the chosen number which is used to divide the unsorted data into two halves. The lower half contains less than value of the chosen number i.e. pivot element. The upper half contains greater than value of the chosen number i.e. pivot element. So the chosen number is now sorted.
21	**What is median-of-three –portioning method?** The median-of-three-portioning method is employed in quick sort to find the pivot value. This is done by randomly choosing three elements in the list and finding the median of the elements gives the pivot value.
22	**Define Binary Search.** ***(Dec 07)*** Binary search is a efficient algorithm for searching in a sorted array. It works by comparing a search key K with the

	array's middle element A[m]. If they match, the algorithm stops; otherwise, the same operation is repeated recursively for the first half of the array if K<a[m] and for the second half if K>A[m].
23	**Define Stassen's matrix multiplication.** ***(Dec 07)*** Strassen showed that 2x2 matrix multiplications can be accomplished in 7 multiplication and 18 additions or subtractions. (2log27 =22.807). This reduce can be done by Divide and Conquer Approach.
24	**How many multiplications are performed in two n-digit multiplication?** There is a divide-and-conquer algorithm for multiplying two n-digit integers that requires about n1.585 one-digit multiplications.
	UNIT-II / PART-B
1	Explain how brute force approach is applied to solve closest-Pair and convex-Hull problem.
2	Explain the concept of travelling salesman problem, knapsack problem and assignment problem using exhaustive search.
3	Explain the concept of knapsack problem using exhaustive search.
4	Explain the concept of assignment problem using exhaustive search.
5	Explain in detail about merge sort. Illustrate the algorithm with a numeric example***.(Dec 06)***
6	Apply merge sort to sort the list E, X, A, M, P, L, E in alphabetical order.
7	Write an algorithm for merge sorting. Show the intermediate steps when the numbers 310, 285, 179, 652, 351, 423, 861, 254, 450, 520 are sorted using Merge Sort***.(Dec 14)***
8	Devise an algorithm to sort the following elements using Merge sort technique 286, 45,278,368,475,389,656,788,503,126. ***(Dec 13)***
9	Distinguish between quick sort and merge sort, and arrange the following numbers in increasing order using merge sort.(18,29,68,32,43,37,87,24,47,50). ***(May 13)***
10	Explain quick sort using divide and conquer method.
11	Sort the following set of elements using quick sort. 12,24,8,71,4,23,6 ***(June 06)***
12	Write an algorithm for binary search using divide and conquer and analyze the time complexity***.(June 06,Dec 14)***
13	Apply binary search algorithm to find the key value 32 in the following list of elements. 1,7,12, 16, 18, 21,24,32,34.
14	Explain binary search algorithm in detail. Using Binary Search, search for the elements 151, -14, 9 in the given set of sorted elements 15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151.
15	Write Strassen's matrix multiplication algorithm. Is there any time efficiency improvement compared to ordinary matrix multiplication?
16	Apply Strassen's algorithm to compute $\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} * \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
17	Explain how divide and conquer method is applied in multiplication of large integers?
18	Compute 2101*1130 by applying the divide and conquer algorithm.
19	Explain how divide and conquer method is applied to solve closest-Pair and convex-Hull problem.
	UNIT-III / PART-A
1	**Define dynamic programming.** Dynamic programming is a technique for solving problems with overlapping sub problems. Rather than solving overlapping sub problems again and again, dynamic programming suggests solving each of the smaller sub problems only once and recording the results in a table from which a solution to the original problem can then be obtained.
2	**What does dynamic programming have in common with divide and conquer?** Both the techniques are based on dividing a problems instance into smaller instances of the same problem.
3	**What is the difference between dynamic programming with divide and conquer method?** Divide and conquer divides an instance into smaller instances with no intersections whereas dynamic programming deals with problems in which smaller instances overlap. Consequently divide and conquer algorithm do not explicitly store solutions to smaller instances and dynamic programming algorithms do.
4	**Define binomial coefficient.** A binomial coefficient C (n, k) can be defined as the coefficient of X^k in the expansion of $(1 + X)^n$. A binomial

	coefficient C (n, k) also gives the number of ways, disregarding order that k objects can be chosen from among n objects; more formally, the number of k-element subsets or k-combinations of an n-element set.
5	**Write the equation for calculating binomial coefficient.** The value of C (n, k) can recursively calculated using following standard formula for Binomial coefficients that k objects can be chosen from among n objects. C(n, k) = C(n-1, k-1) + C(n-1, k) C(n, 0) = C(n, n) = 1
6	**Which algorithm is used to find out the transitive closure of a matrix?** Warshall algorithm is used to find the transitive closure of the given digraph
7	**What is the Purpose of the Floyd algorithm?** The Floyd's algorithm is used to find the shortest distance between every pair of vertices in a graph.
8	**What are the conditions involved in the Floyd's algorithm?** ✓ Construct the adjacency matrix. ✓ Set the diagonal elements to zero ✓ $A_k[i,j]=\min\begin{cases} A_{k-1}[i,j] \\ A_{k-1}[i,k] \text{and } A_{k-1}[k,j] \end{cases}$
9	**Mention the methods for generating transitive closure of digraph.** ✓ Depth First Search (DFS) ✓ Breadth First Search (BFS)
10	**Define principle of optimality.*(Dec 14)*** It states that an optimal sequence of decisions has the property that whenever the initial stage or decisions must constitute an optimal sequence with regard to stage resulting from the first decision
11	**Define Optimal Binary Search Tree (OBST). *(June 06)*** Dynamic programming can be used for constructing an optimal binary search tree for a given set of keys and known probabilities of searching for them. If probabilities of searching for an element of a set are known, the average number of comparisons in a search will have smallest possible value in an OBST.
12	**Define knapsack problem using dynamic programming.** Designing a dynamic programming algorithm for the knapsack problem: given n items of known weights $w_1 . . . w_n$ and values $v_1, . . . , v_n$ and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack. We assume here that all the weights and the knapsack capacity are positive integers; the item values do not have to be integers.
13	**Define Memory function techniques. (*Dec 06*)** The memory function technique seeks to combine strengths of the top-down and bottom-up approaches to solving problems with overlapping sub problems. It does this by solving, in the top-down fashion but only once, just necessary sub problems of a given problem and recording their solutions in a table.
14	**State greedy technique.** The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.
15	**Define feasible solution. *(Dec 13)*** Any subset that satisfies problem's constraints called feasible solution.
16	**Define objective function.** To find a feasible solution that either maximizes or minimizes a given objective function.
17	**Define optimal solution. *(June 07,Dec 13)*** It has to be the best choice among all feasible solution available on that step.
18	**Mention the two classic algorithms for the minimum spanning tree problem.** ✓ Prim's algorithm ✓ Kruskal's algorithm
19	**Define spanning tree.** A spanning tree of a connected graph is its connected acyclic sub graph that contains all the vertices of the graph
21	**Define minimum spanning tree. *(June 06,07)***

	A minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight.
22	**Define weight of a tree.** Weight of a tree is defined as the sum of the weights on all its edges.
23	**Define minimum spanning tree problem.** A minimum spanning tree problem is the problem of finding a minimum spanning tree for a given weighted connected graph.
24	**State two obstacles for constructing minimum spanning tree using exhaustive-search approach.** ✓ The number spanning tree grows exponentially with the graph size ✓ Generating all spanning trees for a given graph is not easy; in fact, it is more difficult than finding a minimum spanning tree for a weighted graph by using one of several efficient algorithms available for this problem.
25	**Write the concept of Prim's spanning tree.** Prim's algorithm constructs a minimum spanning tree through a sequence of expanding sub trees. The initial sub tree in such a sequence consists of a single vertex selected arbitrarily from the set V of the graph's vertices. On each iteration, we expand the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree. The algorithm stops after all the graph's vertices have been included in the tree being constructed
26	**Does prim's algorithm always yield a minimum spanning tree?** Yes. prim's algorithm always yield a minimum spanning tree
27	**How efficient is prim's algorithm?** It depends on the data structures chosen for the graph itself and for the priority queue of the set $V-V_T$ whose vertex priorities are the distances to the nearest tree vertices.
28	**How priority queues are implemented?** We can also implement the priority queues as a min-heap.
29	**Define min-heap.** A min-heap is a complete binary tree in which every element is less than or equal to its children.
30	**Write the concept of kruskal's algorithm**. Kruskal's algorithm looks at a minimum spanning tree for a weighted connected graph G=(V,E) as an acyclic sub graph with |V|-1 edges for which the sum of the edge weights is the smallest. Consequently, the algorithm constructs a minimum spanning tree as an expanding sequence of sub graphs, which are always acyclic but are not necessarily connected on the intermediate stages of the algorithm. The algorithm begins by sorting the graph's edges in non decreasing order of their weights. Then, starting with the empty sub graph, it scans this sorted list, adding the next edge on the list to the current sub graph if such an inclusion does not create a cycle and simply skipping the edge otherwise.
31	**What is the purpose of Dijikstra's Algorithm?** Dijikstra's algorithm is used to find the shortest path between sources to every vertex. This algorithm is applicable to undirected and directed graphs with nonnegative weights only.
32	**Write the concept of Dijikstra's Algorithm?** Dijkstra's algorithm finds the shortest paths to a graph's vertices in order of their distance from a given source. First, it finds the shortest path from the source to a vertex nearest to it, then to a second nearest, and so on. In general, before its ith iteration commences, the algorithm has already identified the shortest paths to $i - 1$ other vertices nearest to the source. These vertices, the source, and the edges of the shortest paths leading to them from the source form a sub tree Ti of the given graph .Since all the edge weights are nonnegative, the next vertex nearest to the source can be found among the vertices adjacent to the vertices of Ti . The set of vertices adjacent to the vertices in Ti can be referred to as "fringe vertices"; they are the candidates from which Dijkstra's algorithm selects the next vertex nearest to the source. To identify the ith nearest vertex, the algorithm computes, for every fringe vertex u, the sum of the distance to the nearest tree vertex v and the length dv of the shortest path from the source to v and then selects the vertex with the smallest such sum.
33	**Define codeword.** Encode a text that comprises symbols from some n-symbol alphabet by assigning to each of the text's symbols some sequence of bits called the codeword.
34	**Define fixed-length encoding.**

	Assigning each character a bit string of the same length m called pixel length encoding (m ≥ log2 n).
35	**Define Variable – length encoding.** Assigning code words of different lengths to different characters called Variable-length encoding.
36	**State Huffman's algorithm.** ✓ S Initialize n one-node trees and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree's root to indicate the tree's weight. (More generally, the weight of a tree will be equal to the sum of the frequencies in the tree's leaves.) ✓ Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight make them the left and right sub tree of a new tree and record the sum of their weights in the root of the new tree as its weight.
37	**Define Huffman tree.** A tree constructed using Huffman algorithm is called a Huffman tree. A Huffman tree is a binary tree that minimizes the weighted path length from the root to the leaves of predefined weights
38	**Define Huffman code. (*Dec 06*)** A Huffman code is an optimal prefix-free variable-length encoding scheme that assigns bit strings to symbols based on their frequencies in a given text. This is accomplished by a greedy construction of a binary tree whose leaves represent the alphabet symbols and whose edges are labeled with 0's and 1's.
	UNIT-III / PART-B
1	Explain how dynamic programming technique is applied to compute binomial coefficient.
2	Explain Warshall's algorithm to find the transitive closure with an example.
3	Explain the pseudo code for Warshall's algorithm and apply for the following diagraph. 
4	Write Floyd's algorithm for the all-pairs shortest path problem and explain with an example.
5	Write and explain Floyd's algorithm for the all-pairs shortest path problem. Using this find the length of the shortest path between all pairs of vertices of the following graph. ***(Dec 06,07)*** 
6	Explain an algorithm to find optimal binary search tree with example***.(Dec 07,14)***
7	Develop the optimal binary search tree for the following instance and explain the algorithm in detail. Key: A, B, C, D Probability: 0.1, 0.2, 0.4, 0.3
8	Explain knapsack problem using dynamic programming with an example.
9	Apply the dynamic programming algorithm to solve the following instance of the knapsack problem and explain in detail.

	item	weight	value
	1	3	$25
	2	2	$20
	3	1	$15
	4	4	$40
	5	5	$50

Knapsack capacity W=6

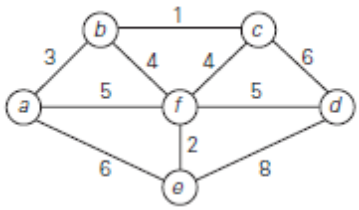
10 Write an memory function algorithm to solve the following knapsack problem. ***(Dec 07)***

item	weight	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

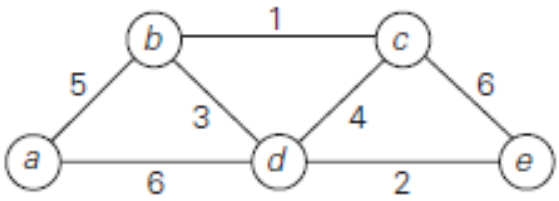
Knapsack capacity W= 5

11 Explain the method for finding the minimum spanning tree for a connected graph using Prim's algorithm with an example. ***(June 06,Dec 14)***

12 Explain the pseudo code for prim's algorithm and apply the same to minimum spanning tree for the following graph. ***(Dec 07)***

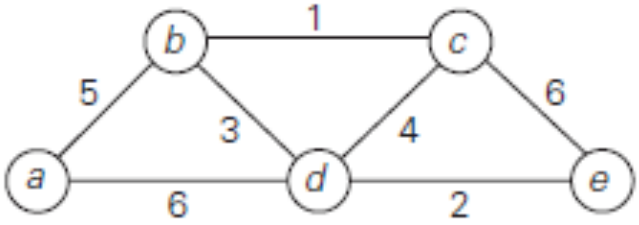


13 Explain the pseudo code for prim's algorithm and apply the same to minimum spanning tree for the following graph.

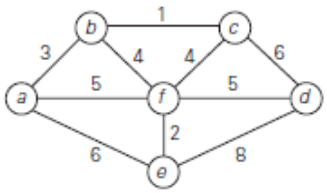


14 Construct a minimum spanning tree using Kruskal's algorithm with your own example***.(Dec 06,07)***

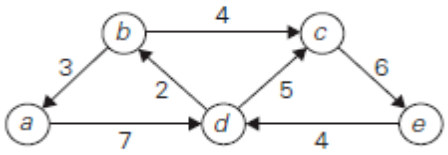
15 Apply Kruskal's algorithm to find a minimum spanning tree of the following graph. ***(Dec 07)***



16 Apply Kruskal's algorithm to find a minimum spanning tree of the following graph.



17 How will find the shortest path between two given vertices using Dijikstra's algorithm? Explain the pseudo code with an example. ***(June 06,07)***

18	Solve the following instance of the single source shortest path problem with vertex a as the source. 
19	Explain the construction of huffman coding tree with an example.
20	Consideer the five character alphabet {A,B,C,D,_} with the following occurence probabilities and construct huffman tree. <table><tr><td>Character</td><td>A</td><td>B</td><td>C</td><td>D</td><td>_</td></tr><tr><td>Probability</td><td>0.35</td><td>0.1</td><td>0.2</td><td>0.15</td><td></td></tr></table>

UNIT-IV / PART-A

1	**What is the purpose of iterative improvement technique?** The iterative-improvement technique involves finding a solution to an optimization problem by generating a sequence of feasible solutions with improving values of the problem's objective function. Each subsequent solution in such a sequence typically involves a small, localized change in the previous feasible solution. When no such change improves the value of the objective function, the algorithm returns the last feasible solution as optimal and stops.
2	**List out the problems that can be solved by iterative improvement algorithms.** ✓ Linear programming ✓ Maximizing the flow in a network, ✓ Matching the maximum possible number of vertices in a graph.
3	**Define simplex method.** The simple method is the classic method for solving the general linear programming problem. It works by generating a sequence of adjacent extreme points of the problem's feasible region with improving values of the objective function.
4	**Define feasible solution.** Any point (x, y) that satisfies all the constraints of the problem
5	**Define feasible region.** Feasible region is the set of all its feasible points.
6	**Define extreme point theorem.** Any linear programming problem with a nonempty bounded feasible region has an optimal solution; moreover, an optimal solution can always be found at an extreme point of the problem's feasible region.
7	**Define basic feasible solution.** If all the coordinates of a basic solution are nonnegative, the basic solution is called a basic feasible solution.
8	**Define simplex tableau.** Simplex method progresses through a series of adjacent extreme points (basic feasible solutions) with increasing values of the objective function. Each such point can be represented by a simplex tableau, a table storing the information about the basic feasible solution corresponding to the extreme point.
9	**Define source vertex.** It contains exactly one vertex with no entering edges; this vertex is called the source and assumed to be numbered 1.
10	**What is the need of maximum-flow problem?** The maximum-flow problem asks to find the maximum flow possible in a network, a weighted directed graph with a source and a sink.
11	**Define sink vertex.** It contains exactly one vertex with no leaving edges; this vertex is called the sink and assumed to be numbered n.
12	**Define edge capacity.** The weight u_{ij} of each directed edge (i, j) is a positive integer, called the edge capacity. E.x; This number represents

	the upper bound on the amount of the material that can be sent from i to j through a link represented by this edge.
13	**Define value of the flow.** The quantity, the total outflow from the source or equivalently, the total inflow into the sink-is called the value of the flow. It is denoted it by v. It is this quantity that we will want to maximize over all possible flows in a network
14	**Define flow.** A (feasible) flow is an assignment of real numbers x_{ij} to edges (i, j) of a given network that satisfy flow-conservation constraints and the capacity constraints $0 \le x_{ij} \le u_{ij}$ for every edge (i, j) ∈ E.
15	**Write the optimization function for maximum-flow problem.** The maximum-flow problem can be stated formally as the following optimization problem: $\text{maximize} \quad v = \sum_{j:(1,j)\in E} x_{1j}$ $\text{subject to} \quad \sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = 0 \quad \text{for } i = 2, 3, \ldots, n-1$ $0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i, j) \in E.$
16	**Define Ford-Fulkerson method.** The Ford-Fulkerson method is a classic template for solving the maximum flow problem by the iterative-improvement approach. The shortest augmenting-path method implements this idea by labeling network vertices in the breadth-first search manner.
17	**Define forward edge.** A undirected graph in which any two consecutive vertices i, j are either connected by a directed edge from i to j with some positive unused capacity $r_{ij}= u_{ij}- x_{ij.}$ Edges of this kind are called forward edges because their tail is listed before their head in the vertex list 1→. . . i →j . . .→n defining the path.
18	**Define backward edge.** A undirected graph in which any two consecutive vertices i, j are either connected by a directed edge from j to i with some positive flow x_{ji} . Edges of this kind are called backward edges because their tail is listed after their head in the path list 1→. . . i ←j . . .→n.
19	**Define Max-Flow Min-Cut theorem.** Max-Flow Min-Cut theorem is defined as the value of a maximum flow in a network is equal to the capacity of its minimum cut.
20	**What you mean by matching in a graph.** A matching in a graph is a subset of its edges with the property that no two edges share a vertex.
21	**Define maximum-matching problem.** The maximum-matching problem is the problem of finding a maximum matching (matching with the largest number of edges) in a given graph.
22	**What is mean by bipartite graph? (or) Define 2-colorable in bipartite graph.** In a bipartite graph, all the vertices can be partitioned into two disjoint sets V and U, not necessarily of the same size, so that every edge connects a vertex in one of these sets to a vertex in the other set. In other words, a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also said to be 2-colorable.
23	**What do you mean by maximum weight matching?** The problem of maximizing the sum of the weights on edges connecting matched pairs of vertices. This problem is called maximum-weight matching.
24	**When marriage matching problem is said to stable?** A marriage matching M is called stable if there is no blocking pair for it.
25	**What is mean by stable marriage problem?** Consider a set Y = {m_1, m_2, . . . , m_n} of n men and a set X = {w_1, w_2, . . . , w_n} of n women. Each man has a preference list ordering the women as potential marriage partners with no ties allowed. Similarly, each woman has a preference list of the men, also with no ties.

26	**Define marriage matching.** A marriage matching M is a set of n (m, w) pairs whose members are selected from disjoint n-element sets Y and X in a one-one fashion, i.e., each man m from Y is paired with exactly one woman w from X and vice versa.
27	**Define blocking pair.** A pair (m, w), where m ∈ Y, w ∈ X, is said to be a blocking pair for a marriage matching M if man m and woman w are not matched in M but they prefer each other to their mates in M.
28	**When marriage matching problem is said to unstable?** A marriage matching M is called stable if there is a blocking pair for it.
29	**Write the algorithm for stable marriage problem.** **Input**: A set of n men and a set of n women along with rankings of the women by each man and rankings of the men by each woman with no ties allowed in the rankings **Output**: A stable marriage matching ✓ Start with all the men and women being free. ✓ While there are free men, arbitrarily select one of them and do the following: **Proposal**: The selected free man m proposes to w, the next woman on his preference list (who is the highest-ranked woman who has not rejected him before). **Response:** If w is free, she accepts the proposal to be matched with m. If she is not free, she compares m with her current mate. If she prefers m to him, she accepts m's proposal, making her former mate free; otherwise, she simply rejects m's proposal, leaving m free. ✓ Return the set of n matched pairs.
30	**Prove that the stable marriage algorithm terminates after no more than n^2 iterations with a stable marriage output.** The algorithm starts with n men having the total of n2 women on their ranking lists. On each iteration, one man makes a proposal to a woman. This reduces the total number of women to whom the men can still propose in the future because no man proposes to the same woman more than once. Hence, the algorithm must stop after no more than n2 iterations.
31	**Prove that the stable marriage algorithm always yields a gender-optimal stable matching.** To prove that a man (woman)-optimal matching is unique for a given set of participant preferences. Therefore the algorithm's output does not depend on the order in which the free men (women) make their proposals.
	UNIT-IV / PART-B
1	Explain geomtric interpretation of linear programming with an example. Consider the following linear programming problem in two variables: $$\begin{aligned} \text{maximize} \quad & 3x + 5y \\ \text{subject to} \quad & x + y \le 4 \\ & x + 3y \le 6 \\ & x \ge 0, \quad y \ge 0. \end{aligned} \qquad \textbf{(10.2)}$$ By definition, a ***feasible solution*** to this problem is any point (x, y) that satisfies all the constraints of the problem; the problem's ***feasible region*** is the set of all its feasible points. It is instructive to sketch the feasible region in the Cartesian plane. Recall that any equation $ax + by = c$, where coefficients a and b are not both equal to zero, defines a straight line. Such a line divides the plane into two half-planes: for all the points in one of them, $ax + by < c$, while for all the points in the other, $ax + by > c$. (It is easy to determine which of the two half-planes is which: take any point (x_0, y_0) not on the line $ax + by = c$ and check which of

the two inequalities hold, $ax_0 + by_0 > c$ or $ax_0 + by_0 < c$.) In particular, the set of points defined by inequality $x + y \leq 4$ comprises the points on and below the line $x + y = 4$, and the set of points defined by inequality $x + 3y \leq 6$ comprises the points on and below the line $x + 3y = 6$. Since the points of the feasible region must satisfy all the constraints of the problem, the feasible region is obtained by the intersection of these two half-planes and the first quadrant of the Cartesian plane defined by the nonnegativity constraints $x \geq 0$, $y \geq 0$ (see Figure 10.1). Thus, the feasible region for problem (10.2) is the convex polygon with the vertices (0, 0), (4, 0), (0, 2), and (3, 1). (The last point, which is the point of intersection of the lines $x + y = 4$ and $x + 3y = 6$, is obtained by solving the system of these two linear equations.) Our task is to find an ***optimal solution***, a point in the feasible region with the largest value of the ***objective function*** $z = 3x + 5y$.

Are there feasible solutions for which the value of the objective function equals, say, 20? The points (x, y) for which the objective function $z = 3x + 5y$ is equal to 20 form the line $3x + 5y = 20$. Since this line does not have common points

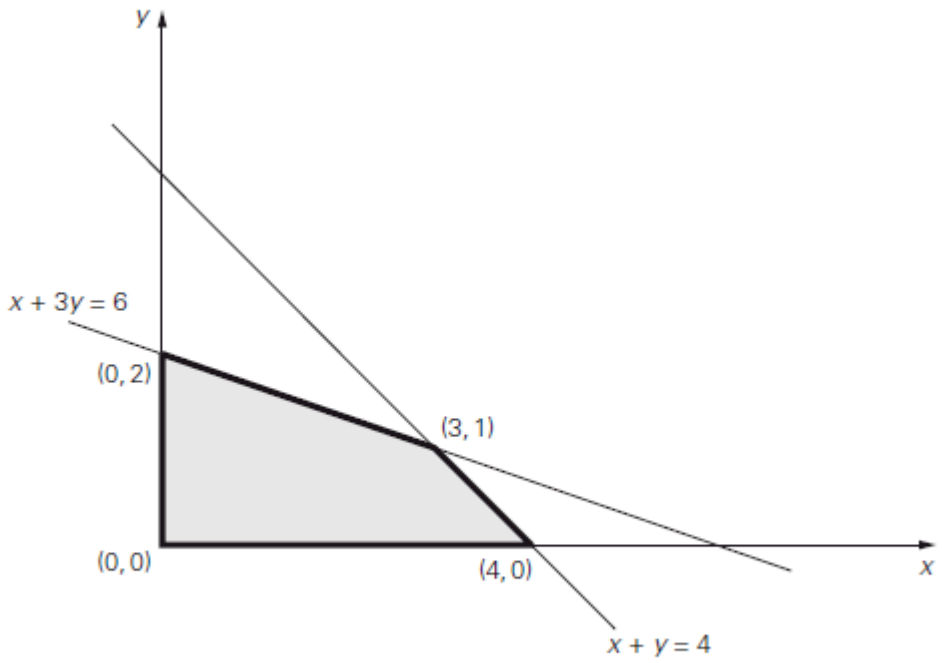


FIGURE 10.1 Feasible region of problem (10.2).

with the feasible region—see Figure 10.2—the answer to the posed question is no. On the other hand, there are infinitely many feasible points for which the objective function is equal to, say, 10: they are the intersection points of the line $3x + 5y = 10$ with the feasible region. Note that the lines $3x + 5y = 20$ and $3x + 5y = 10$ have the same slope, as would any line defined by equation $3x + 5y = z$ where z is some constant. Such lines are called ***level lines*** of the objective function. Thus, our problem can be restated as finding the largest value of the parameter z for which the level line $3x + 5y = z$ has a common point with the feasible region.

We can find this line either by shifting, say, the line $3x + 5y = 20$ south-west (without changing its slope!) toward the feasible region until it hits the region for the first time or by shifting, say, the line $3x + 5y = 10$ north-east until it hits the feasible region for the last time. Either way, it will happen at the point (3, 1) with the corresponding z value $3 \cdot 3 + 5 \cdot 1 = 14$. This means that the optimal solution to the linear programming problem in question is $x = 3$, $y = 1$, with the maximal value of the objective function equal to 14.

Note that if we had to maximize $z = 3x + 3y$ as the objective function in problem (10.2), the level line $3x + 3y = z$ for the largest value of z would coincide with the boundary line segment that has the same slope as the level lines (draw this line in Figure 10.2). Consequently, all the points of the line segment between vertices (3, 1) and (4, 0), including the vertices themselves, would be optimal solutions, yielding, of course, the same maximal value of the objective function.

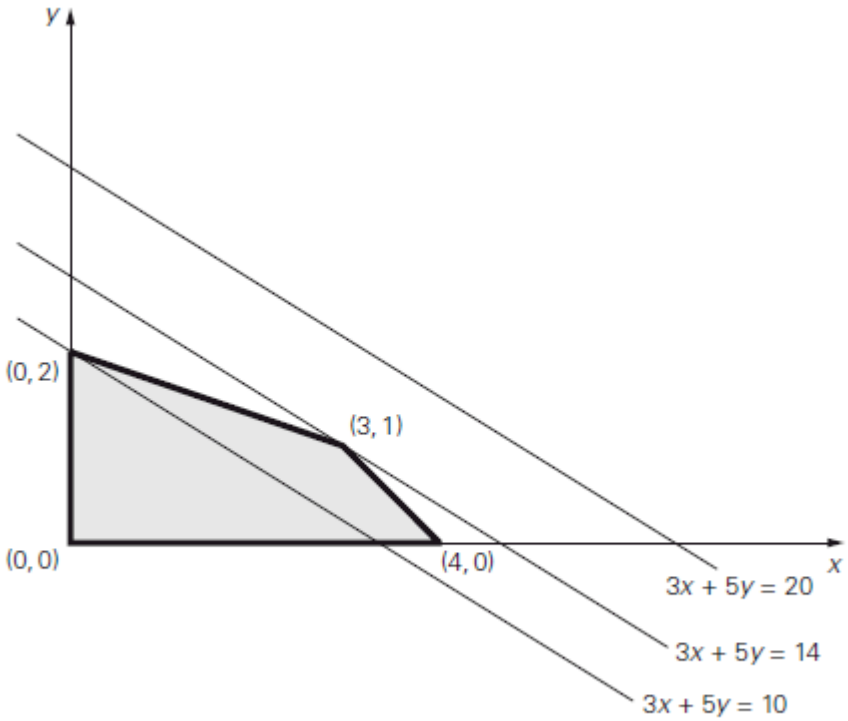


FIGURE 10.2 Solving a two-dimensional linear programming problem geometrically.

Does every linear programming problem have an optimal solution that can be found at a vertex of its feasible region? Without appropriate qualifications, the answer to this question is no. To begin with, the feasible region of a linear programming problem can be empty. For example, if the constraints include two contradictory requirements, such as $x + y \leq 1$ and $x + y \geq 2$, there can be no points in the problem's feasible region. Linear programming problems with the empty feasible region are called ***infeasible***. Obviously, infeasible problems do not have optimal solutions.

2 Explain the steps to solve simplex method using linear programming with an example.

Linear Programming (LP) Problems

A linear programming problem in two unknowns x and y is one in which we are to find the maximum or minimum value of a linear expression ax + by called the objective function, subject to a number of linear constraints of the form

$$cx + dy \leq e \text{ or } cx + dy \geq e$$

The largest or smallest value of the objective function is called the optimal value, and a pair of values of x and y that gives the optimal value constitutes an optimal solution.

A linear programming problem has to satisfy three requirements in order to be classified as a standard problem:

Standard Problems

1. The objective function is to be maximized.
2. All variables must be nonnegative.
3. Each constraint inequality must have the form:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq k,$$

where $a_1, a_2, \ldots, a_n$ are constants, $x_1, x_2, \ldots, x_n$ are the variables, and k is a positive constant.

Simplex Method:

A linear-programming algorithm that can solve problems having more than two decision variables. The simplex technique involves generating a series of solutions in tabular form, called tableaus. By inspecting the bottom row of each tableau, one can immediately tell if it represents the optimal solution. Each tableau corresponds to a corner point of the feasible solution space. The first tableau corresponds to the origin. Subsequent tableaus are developed by shifting to an adjacent corner point in the direction that yields the highest (smallest) rate of profit (cost). This process continues as long as a positive (negative) rate of profit (cost) exists.

There are three main steps in the simplex procedure:

1. Setting up the simplex tableau (which is a matrix).
2. Pivoting (this step may need to be done several times).
3. Reading the final solution.

The simplex method in tabular form

Step1: Initialization (Convert to a system of linear equations)

- Transform all the constraints to equality by introducing slack, surplus, and artificial variables as follows:

Constraint type	Variable to be added
≤	+ slack (s)
≥	- Surplus (s) + artificial (A)
=	+ Artificial (A)

- The first step towards using the simplex algorithm is to convert the structural constraints into equalities by adding so called slack variables. The following structural constraints,

$$a + b \leq c_1;$$

$\mathbf{a + b \leq c_2;}$

turn into the following equalities or slack equations when slack variables, s1 and s2 are added to the first and second constraint, respectively:

$\mathbf{a + b + s1 = c_1;}$

$\mathbf{a + b + s2 = c_2;}$

- The slack variables will always be **nonnegative (zero or positive)** when solving linear programming problems. In a linear programming problem with two variables, the slack variables are always nonnegative in the corner points of the feasible region.

Step 2: Construct the initial simplex tableau

Basic variable	X_1	...	X_n	S_1	.....	S_n	A_1	....	A_n	RHS
S	Coefficient of the constraints									b_1
⋮										⋮
A										b_m
Z	Objective function coefficient In different signs									Z value

- A basic solution is an augmented corner point solution.
- A basic solution has the following properties:

1. Each variable is designated as either a nonbasic variable or a basic variable.
2. The number of basic variables equals the number of functional constraints. Therefore, the number of nonbasic variables equals the total number of variables minus the number of functional constraints.
3. The nonbasic variables are set equal to zero.
4. The values of the basic variables are obtained as simultaneous solution of the system of equations (functional constraints in augmented form). The set of basic variables are called "basis"
5. If the basic variables satisfy the nonnegativity constraints, the basic solution is a Basic Feasible (BF) solution.

Step 3: Test for optimality

Case 1: Maximization problem

The current Basic Feasible solution is optimal if every coefficient in the objective function row is nonnegative

Case 2: Minimization problem

The current Basic Feasible solution is optimal if every coefficient in the objective function row is nonpositive .

Step 4: Select the pivot column or Entering Variable

- Determine the entering basic variable by selecting the variable (automatically a nonbasic variable) with the most negative value (in case of maximization) or with the most positive (in case of minimization) in the last row (Z-row).
- Put a box around the column below this variable, and call it the "pivot column". If all the numbers on the left-hand side of the bottom row are zero or positive, then we are done, and the basic solution is the optimal solution.

Step 5: Select the pivot row or Leaving Variable

- Determine the leaving basic variable by applying the minimum ratio test as following:
  1. Pick out each coefficient in the pivot column that is strictly positive (>0)
  2. Divide each of these coefficients into the right hand side entry for the same row

3. Identify the row that has the smallest of these ratios

4. The basic variable for that row is the leaving variable, so replace that variable by the entering variable in the basic variable column of the next simplex tableau. Put a box around this row and call it the "pivot row"

Step 6: Perform the pivot operation(Forming the next tableau)

Solve for the new Basic Feasible solution by using elementary row operations (multiply or divide a row by a nonzero constant; add or subtract a multiple of one row to another row) to construct a new simplex tableau, and then return to the optimality test. The specific elementary row operations are:

1. Divide the pivot row by the "pivot number" (the number in the intersection of the pivot row and pivot column)
2. For each other row that has a negative coefficient in the pivot column, add to this row the product of the absolute value of this coefficient and the new pivot row.
3. For each other row that has a positive coefficient in the pivot column, subtract from this row the product of the absolute value of this coefficient and the new pivot row.

Pivoting

1. *Divide the pivot row by the pivot element* so the pivot element becomes 1.
2. Use the *pivot row* and row operations to get 0's in the rest of the pivot column.

- When pivoting is done, and we set the non basic variables to zero, we obtain a solution called a basic feasible solution to the linear programming problem.
- Using the smallest quotient rule to choose the pivot element in a given matrix ensures that the slack variables remain non-negative. In a linear programming problem with two variables, the basic feasible solution corresponds to a corner point of the feasible region.

Example

Solve the following problem using the simplex method

- Maximize $Z = 3X_1 + 5X_2$

Subject to $X_1 \leq 4$

$2X_2 \leq 12$

$3X_1 + 2X_2 \leq 18$

$X_1, X_2 \geq 0$

Solution

(All constraints are ≤) so add slack variables

Step1: Initialization (Convert to a system of linear equations)

Standard form

Maximize Z, Subject to

$$Z - 3X_1 - 5X_2 = 0$$
$$X_1 + S_1 = 4$$
$$2X_2 + S_2 = 12$$
$$3X_1 + 2X_2 + S_3 = 18$$
$$X_1, X_2, S_1, S_2, S_3 \geq 0$$

Sometimes it is called the augmented form of the problem because the original form has been augmented by some supplementary variables needed to apply the simplex method.

Step 2: Construct the initial simplex tableau

2. Initial tableau

Entering variable

Basic variable	X_1	X_2	S_1	S_2	S_3	RHS
S_1	1	0	1	0	0	4
S_2	0	2	0	1	0	12
S_3	3	2	0	0	1	18
Z	-3	-5	0	0	0	0

Leaving variable; Pivot column; Pivot number; Pivot row

- The basic feasible solution at the initial tableau is (0, 0, 4, 12, 18)
  Where: $X_1 = 0$, $X_2 = 0$, $S_1 = 4$, $S_2 = 12$, $S_3 = 18$, and $Z = 0$
  Where S_1, S_2, and S_3 are basic variables, X_1 and X_2 are nonbasic variables

Step 3: Test for optimality

- The solution at the initial tableau is associated to the origin point at which all the decision variables are zero.
- By investigating the last row of the initial tableau, we find that there are some negative numbers. Therefore, the current solution is not optimal

Step 4: Select the pivot column or Entering Variable

- Determine the entering variable by selecting the variable with the most negative in the last row.
- From the initial tableau, in the last row (Z row), the coefficient of X_1 is -3 and the coefficient of X_2 is -5; therefore, the most negative is -5. Consequently, X_2 is the entering variable.
- X_2 is surrounded by a box and it is called the pivot column.

Step 5: Select the pivot row or Leaving Variable

- Determining the leaving variable by using the minimum ratio test as following:

Basic variable	Entering variable X_2 (1)	RHS (2)	Ratio (2)÷(1)
S_1	0	4	None
S_2 Leaving	**2**	**12**	**6 Smallest ratio**
S3	2	18	9

Step 6: Perform the pivot operation(Forming the next tableau)

Solving for the new BF solution by using the eliminatory row operations as following

New pivot row = old pivot row ÷ pivot number

Basic variable	X_1	X_2	S_1	S_2	S_3	RHS
S_1						
X_2	0	1	0	1/2	0	6
S_3						
Z						

Note that X_2 becomes in the basic variables list instead of S_2

For the other row apply this rule:

New row = old row – the coefficient of this row in the pivot column (new pivot row)

For S_1

1 0 1 0 0 4

-

0 (0 1 0 1/2 0 6)

1 0 1 0 0 4

For S_3

3 2 0 0 1 18

-

2 (0 1 0 1/2 0 6)

3 0 0 -1 1 6

for Z

-3 -5 0 0 0 0

-

-5(0 1 0 1/2 0 6)

-3 0 0 5/2 0 30

Substitute this values in the table

Test for optimality

This solution is not optimal, since there is a negative numbers in the last row

Basic variable	X_1	X_2	S_1	S_2	S_3	RHS
S_1	1	0	1	0	0	4
X_2	0	1	0	1/2	0	6
S_3	3	0	0	-1	1	6
Z	-3	0	0	5/2	0	30

The most negative value; therefore, X_1 is the entering variable

The smallest ratio is 6/3 =2; therefore, S_3 is the leaving variable

- Apply the same rules from step 3 to 5 we will obtain this solution:

Basic variable	X_1	X_2	S_1	S_2	S_3	RHS
S_1	0	0	1	1/3	-1/3	2
X_2	0	1	0	1/2	0	6
X1	1	0	0	-1/3	1/3	2
Z	0	0	0	3/2	1	36

This solution is optimal. Since there is no negative solution in the last row.

Basic variables are $X_1 = 2$, $X_2 = 6$ and $S_1 = 2$
Nonbasic variables are $S_2 = S_3 = 0$
Z = 36

Special cases of linear programming

- Infeasible solution
- Multiple solution (infinitely many solution)
- Unbounded solution
- Degenerated solution

Notes on the Simplex tableau

1. In any Simplex tableau, the intersection of any basic variable with itself is always one and the rest of the column is zeroes.
2. In any simplex tableau, the objective function row (Z row) is always in terms of the nonbasic variables. This means that under any basic variable (in any tableau) there is a zero in the Z row. For the non basic there is no condition (it can take any value in this row).
3. If there is a zero under one or more nonbasic variables in the last tableau (optimal solution tableau), then there is a multiple optimal solution.

	4. When determining the leaving variable of any tableau, if there is no positive ratio (all the entries in the pivot column are negative and zeroes), then the solution is unbounded. 5. If there is a tie (more than one variables have the same most negative or positive) in determining the entering variable, choose any variable to be the entering one. 6. If there is a tie in determining the leaving variable, choose any one to be the leaving variable. In this case a zero will appear in RHS column; therefore, a "cycle" will occur, this means that the value of the objective function will be the same for several iterations. 7. A Solution that has a basic variable with zero value is called a "degenerate solution". 8. If there is no Artificial variables in the problem, there is no room for "infeasible solution"
3	Solve the following problem using simplex method: maximize $Z = 3x + 5y$ subject to $x + y \leq 4$ $x + 3y \leq 6$ $x \geq 0, y \geq 0.$

Maximize $z = 3x + 5y$ subject to $x + y \leq 4$, $x + 3y \leq 6$

$x \geq 0, y \geq 0$

Solution:

Step 1 : Insert Slack Variables s_1, s_2 & express the given LPP into its Standard form.

$$x + y + s_1 = 4$$
$$x + 3y + s_2 = 6.$$

Step 2: Rewrite the objective function $z = 3x + 5y$ to match the format of slack equations.

$$-3x - 5y + z = 0.$$

Step 3 : Initial simplex tableau

Basic Variable	x	y	s_1	s_2	RHS	
s_1	1	1	1	0	4	s_1
s_2	1	3	0	1	6	s_2
z	-3	-5	0	0	0	← objective function

s_1, s_2 = Basic Variables x, y = Non-basic variables

Since there are 2 equations with 4 variables, the initial feasible solution is obtained by equating $(4-2) = 2$ variables to zero. $[x=0, y=0 \quad S_1=4, S_2=6]$

∴ The basic feasible solution can be written as

$(x, y, S_1, S_2) = (0, 0, 4, 6)$

Step 4: Find the pivot element

	x	y	S_1	S_2	I	RHS	Basic variable	
	1	1	1	0	1	4	S_1	$4/1 = 4$
	1	3	0	1	1	6	S_2	$6/3 = 2$
	-3	-5	0	0	1	0	Z	

Entering Variable

↑ Entering Variable.

Here '-5' is the most negative, the corresponding basic variable 'y' enters the basic. The column to the 'y' is called the pivot column. (↑) / entering Variable.

Leaving Variable:

$\theta_{S_1} = 4/1 = 4 \qquad \theta_{S_2} = 6/3 = 2.$

$\theta = \min\{4, 2\} = 2$. which corresponds to S_2.

Basic Variables	x	y	s_1	s_2	I	RHS	
s_1	1	1	1	0	1	4	s_1
s_2	1	3 (pivot element)	0	1	1	6	← s_2 Leaving Variable
z	-3	-5	0	0	1	0	

↑ Entering Variable.

Step 5: Pivot Operation: (Solving for the new Basic Feasible Solution by using the elimination row operations as following)

New pivot equation = old pivot equation ÷ pivot element

or row

$$\overleftarrow{row}_{new} = [1 \quad 3 \quad 0 \quad 1 \quad 6] \div 3$$

$$\overleftarrow{row}_{new} = [1/3 \quad 1 \quad 0 \quad 1/3 \quad 2]$$

For s_1

New s_1 (row_1) equation = old s_1 equation − [corresponding column coefficient] × New pivot equation.

$$= \text{old } s_1 \text{ equation} - (1) \times \overleftarrow{row}_{new}$$

$$= 1 \quad 1 \quad 1 \quad 0 \quad 4$$

$$- \quad 1/3 \quad -1 \quad 0 \quad -1/3 \quad -2$$

$$2/3 \quad 0 \quad 1 \quad -1/3 \quad 2$$

For z

New R_{new_3} equation = $[-3 \quad -5 \quad 0 \quad 0 \quad 0] - (-5) \times R_{1new}$

$= -3 \quad -5 \quad 0 \quad 0 \quad 0$

$\quad 5/3 \quad 5 \quad 0 \quad 5/3 \quad 10$

$-4/3 \quad 0 \quad 0 \quad 5/3 \quad 10$

Basic Variable	x	y	S_1	S_2	RHS	
S_1	2/3	0	1	−1/3	2	S_1 2/2/3 = 3 ∴ S_1 is the leaving variable (smallest ratio)
y	1/3	1	0	1/3	2	← y 2/1/3 = 6. [Replacing S_2 leaving Variable by entering Variable] y.
z	−4/3	0	0	5/3	10	10/5/3 = −15/2

↑ most negative value

Tableau represents the basic feasible solution (0,2,2,0) with an increased value of the objective function which is equal to 10. If is not optimal (The last row contain negative value)

Entering Variable: −4/3.

Leaving Variable : (3, 6, −15/2)

Smallest Non-negative value is 3. Hence departing

Variable is s_1. Replace s_1 by x.

Basic Variable	x	y	s_1	s_2	R.H.S
x	$2/3$ (pivot element)	0	1	$-1/3$	2
y	$1/3$	1	0	$1/3$	2
Z	$-4/3$	0	0	$5/3$	10

↑ Entering Variable.

Replace leaving Variable x by entering Variable x.

$$\overleftarrow{\text{row}}_{\text{new}} = \left[\frac{2}{3} \quad 0 \quad 1 \quad -\frac{1}{3} \quad 2\right] \div \frac{2}{3}$$

$$\overleftarrow{\text{row}}_{\text{new}} = 1 \quad 0 \quad \frac{3}{2} \quad -\frac{1}{2} \quad 3$$

$$\text{New row}_2 = \left[\frac{1}{3} \quad 1 \quad 0 \quad \frac{1}{3} \quad 2\right] - \frac{1}{3}\left[1 \quad 0 \quad \frac{3}{2} \quad -\frac{1}{2} \quad 3\right]$$

$$= \left[\frac{1}{3}-\frac{1}{3} \quad 1-0 \quad 0-\frac{1}{2} \quad \frac{1}{3}+\frac{1}{6} \quad 2-1\right]$$

$$= \left[0 \quad 1 \quad -\frac{1}{2} \quad \frac{1}{2} \quad 1\right]$$

$$\text{New row}_3 = \left[-\frac{4}{3} \quad 0 \quad 0 \quad \frac{5}{3} \quad 10\right] - \left(-\frac{4}{3}\right)$$

$$\left[1 \quad 0 \quad \frac{3}{2} \quad -\frac{1}{2} \quad 3\right]$$

=	$-4/3$	0	0	$5/3$	10.
	$+4/3$	0	2	$-2/3$	4
	0	0	2	1	14.

Basic Variables	x	y	S_1	S_2	RHS	
x	1	0	3/2	$-1/2$	2.	x
y	0	1	$-1/2$	$1/2$	1	y.
z	0	0	2	1	14	← objective funct.

This tableau represents the basic feasible solution (3,1,0,0).

It is optimal because all the entries in the objective row of tableau are non-negative.

∴ The solution to linear programming problem is Z reaches a maximum value of 14 for $x=3$ & $y=1$.

The optimal solution will be $(x, y, S_1, S_2) = (3,1,0,0)$

$Z = 3x + 5y = 3(3) + 5(1) = 14$

$Z = 14$

Basic Variables are $x=3$, $y=1$
Non-basic variables $S_1 = S_2 = 0$

4	Use simplex method to solve the LPP maximize Z = 4x + 10y subject to $2x + y \leq 50$ $2x + 5y \leq 100$ $2x + 3y \leq 90$ $x \geq 0, y \geq 0.$

Maximize $Z = 4x + 10y$ subject to $2x + y \leq 50$, $2x + 5y \leq 100$, $2x + 3y \leq 90$ & $x, y \geq 0$

Step 1: Insert Slack Variables.

Convert the inequality constraints into equations by introducing the slack variables s_1, s_2 and s_3 & express the given LPP into its standard form.

$$2x + y + s_1 = 50$$
$$2x + 5y + s_2 = 100$$
$$2x + 3y + s_3 = 90$$

Step 2: Rewrite the objective function

The given objective function is $Z = 4x + 10y$ Rewrite the objective function to match the format of the slack equations and add the corresponding equation to the bottom of the slack equations.

$$2x + y + s_1 = 50$$
$$2x + 5y + s_2 = 100$$
$$2x + 3y + s_3 = 90$$
$$-4x - 10y + Z = 0$$

Step 3: Initial Simplex Tableau.

Write the equations in the form

$$-c_1x_1 - c_2x_2 - c_3x_3 \ldots - c_nx_n + (0)s_1 + (0)s_2 + \ldots + z = 0$$

$2x$	+	y	+	s_1			=	50
$2x$	+	$5y$	+		s_2		=	100
$2x$	+	$3y$	+			s_3	=	90

The augmented matrix is called initial simplex tableau. [It is augmented form of problem because the original form has been augmented by some supplementary variables needed to apply the simplex method]

Basic Variable	x	y	s_1	s_2	s_3	RHS	
s_1	2	1	1	0	0	50	s_1
s_2	2	5	0	1	0	100	s_2
s_3	2	3	0	0	1	90	s_3
z	-4	-10	0	0	0	0	← Bottom row represents objective function.

*) Basic Variables are s_1, s_2 and s_3 & Non-basic Variables are x & y.

*) If $x=0$ & $y=0$ then the initial basic

feasible solution is $S_1=50$, $S_2=100$, $S_3=90$ ($x=0, y=0$, non-basic)

Since there are 3 equations with 5 variables, the initial feasible solution is obtained by equating $(5-3)=2$ variables to zero.

∴ The basic feasible solution can be written as $(x, y, s_1, s_2, s_3) = (0, 0, 50, 100, 90)$.

Step 4: Find the pivot element.

*) The most negative indicator in the last row of the tableau determines the pivot column.

Entering Variable :- smallest negative entry in the bottom row of the table.

'-10' is the most negative, the corresponding basic variable 'y' enters the basis. The column corresponding to this 'y' is called the key column / pivot column. (↑)

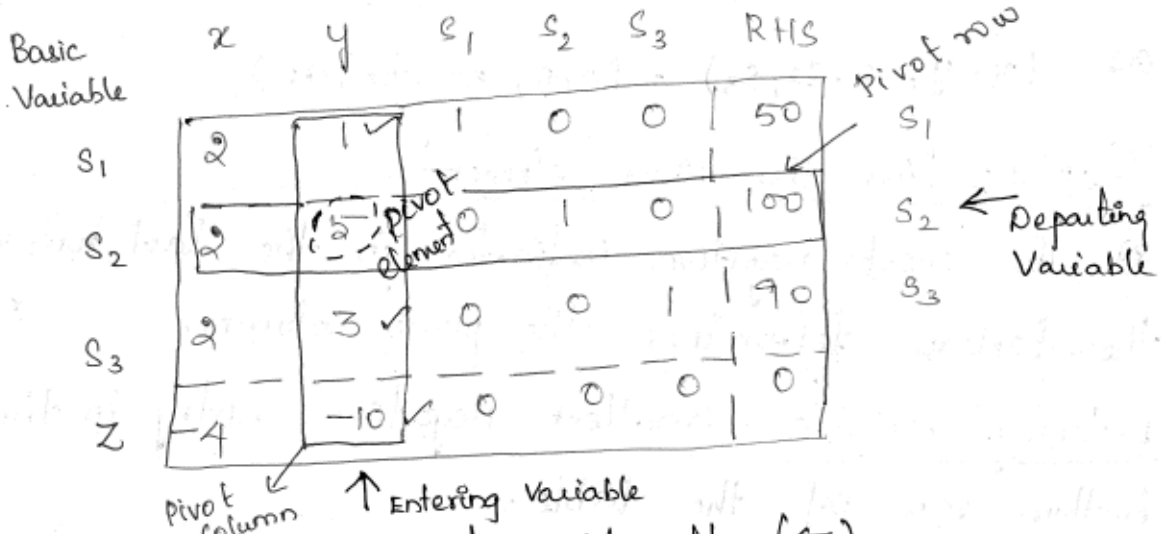
Basic variable	x	y	s_1	s_2	s_3	RHS	
s_1	2	1	1	0	0	50	s_1 50/1
s_2	2	5	0	1	0	100	s_2 100/5
s_3	2	3	0	0	1	90	s_3 90/3
Z	-4	-10	0	0	0	0	

↑ entering variable

Departing / Leaving Variable (Apply minimum ratio test)

*) Smallest negative ratio of RHS/a_{ij} in the column determined by entering variable.

*) Compute the θ-ratio by dividing the row's last entry by the entry in the pivot column.



To find departing Variable (←)

$\theta_{S_1} = 50/1 = 50$ $\theta_{S_2} = 100/5 = 20$ $\theta_{S_3} = 90/3 = 30$

$\theta = \min\{50, 20, 30\} = 20$ which corresponds to θ of S_2.

[Identify the row with smallest

∴ The leaving Variable is the basic Variable S_2 which corresponds to the minimum ratio $\theta = 20$. The leaving Variable row is called the

pivot row / key row / pivot equation (←) & '5' is the pivot element.

Step 5: Pivot Operation eliminating row operations

New pivot equation = old pivot equation ÷ pivot element

$$\overleftarrow{row}_{new} = (2 \quad 5 \quad 0 \quad 1 \quad 0 \quad 100) \div 5$$

$$\boxed{\overleftarrow{row}_{new} = \frac{2}{5} \quad 1 \quad 0 \quad \frac{1}{5} \quad 0 \quad 20.}$$

Replace each of the other rows, including the objective row, by the difference

$$\boxed{row - c.\overleftarrow{row}_{new}}$$

where c is the row's entry in the pivot column.

For S_1 row_1

New (S_1) equation = old S_1 equation − [Corresponding Column Coefficient] × New pivot equation

$$= row_1 - 1 \times \overleftarrow{row}_{new}$$

$$= \begin{Bmatrix} 2 & 1 & 1 & 0 & 0 & 50 \\ -\frac{2}{5} & -1 & 0 & -\frac{1}{5} & 0 & -20. \end{Bmatrix}$$

Here c = 1.

$$\frac{8}{5} \quad 0 \quad 1 \quad -\frac{1}{5} \quad 0 \quad 30$$

For S_3

New (S_3) equation = 2 3 0 0 1 90

(−) $-\frac{6}{5}$ −3 0 $-\frac{3}{5}$ 0 −60

$\left[3\times\frac{2}{5}\quad 3\times 1\quad 3\times 0\quad 3\times\frac{1}{5}\quad 3\times 0\quad 3\times 20\right]$

Here c = 3

$\frac{4}{5}$ 0 0 $-\frac{3}{5}$ 1 30

For z

New row4 equation = −4 −10 0 0 0 0

(−) −4 −10 0 −2 0 −200

c × new row

$\left[-10\times\frac{2}{5}\quad -10\times 1\quad -10\times 0\quad -10\times\frac{1}{5}\quad -10\times 0\quad -10\times 20\right]$

Here c = −10

4 10 0 2 0 200

0 0 0 2 0 200

Basic Variable	x	y	S_1	S_2	S_3	RHS	
S_1	$\frac{8}{5}$	0	1	$-\frac{1}{5}$	0	30	S_1
y	$\frac{2}{5}$	1	0	$\frac{1}{5}$	0	20	y ← Replacing departing Variable
S_3	$\frac{4}{5}$	0	0	$-\frac{3}{5}$	1	30	S_3
Z	0	0	0	2	0	200	

↑ S_2 by y. (i.e. entering Variable)

If no negative indicators are present in the last row, the maximum of the objective function has been reached

∴ The solution to linear programming problem is z reaches a maximum value of 200 for $x = 0$ and $y = 20$.

[Note: Slack Variables are not part of the final solution, just a tool to help us to solve the problem].

The optimal solution will be

$(x, y, s_1, s_2, s_3) = (0, 20, 30, 0, 30)$

$Z = 4x + 10y = 4(0) + 10(20) = 200.$

Basic Variables are $s_1 = 30$, $y = 20$, $s_3 = 30$.

Non-basic Variables are $x = 0$, $s_2 = 0$

$Z = 200$

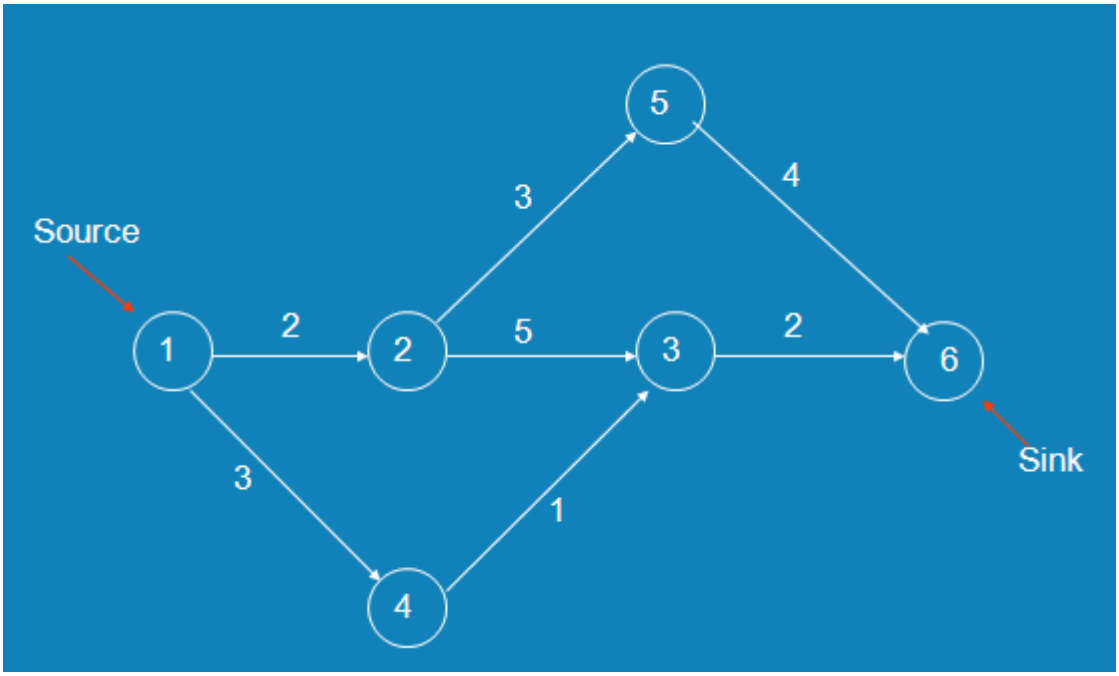
5	Explain the maximum flow problem with an example. **Maximum Flow Problem**

Problem of maximizing the flow of a material through a transportation network (e.g., pipeline system, communications or transportation networks)

Formally represented by a connected weighted digraph with *n* vertices numbered from 1 to *n* with the following properties:

- **contains exactly one vertex with no entering edges, called the *source* (numbered 1)**
- **contains exactly one vertex with no leaving edges, called the *sink* (numbered *n*)**
- **has positive integer weight u_{ij} on each directed edge (*i*,*j*), called the *edge capacity*, indicating the upper bound on the amount of the material that can be sent from *i* to *j* through this edge**

Example of Flow Network



Definition of a Flow

A *flow* is an assignment of real numbers x_{ij} to edges (i,j) of a given network that satisfy the following:

- *flow-conservation requirements*
  The total amount of material entering an intermediate vertex must be equal to the total amount of the material leaving the vertex

$$\sum_{j:(j,i)\in E} x_{ji} = \sum_{j:(i,j)\in E} x_{ij} \quad \text{for } i = 2,3,\ldots,n-1$$

- *capacity constraints*

$$0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i,j) \in E$$

Flow value and Maximum Flow Problem

Since no material can be lost or added to by going through intermediate vertices of the network, the total amount of the material leaving the source must end up at the sink:

$$\sum_{j:(1,j)\in E} x_{1j} = \sum_{j:(j,n)\in E} x_{jn}$$

The *value* of the flow is defined as the total outflow from the source (= the total inflow into the sink).

The *maximum flow problem* is to find a flow of the largest value (maximum flow) for a given network.

Maximum-Flow Problem as LP problem

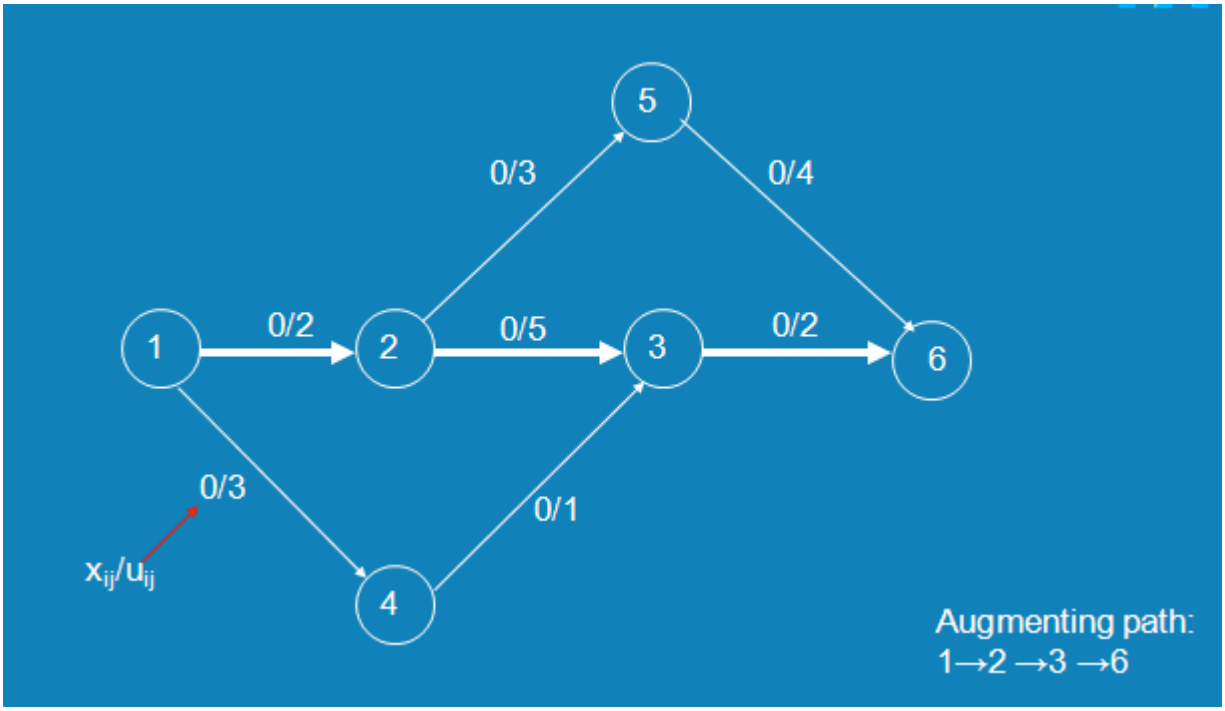
Maximize $v = \sum_{j:(1,j)\in E} x_{1j}$

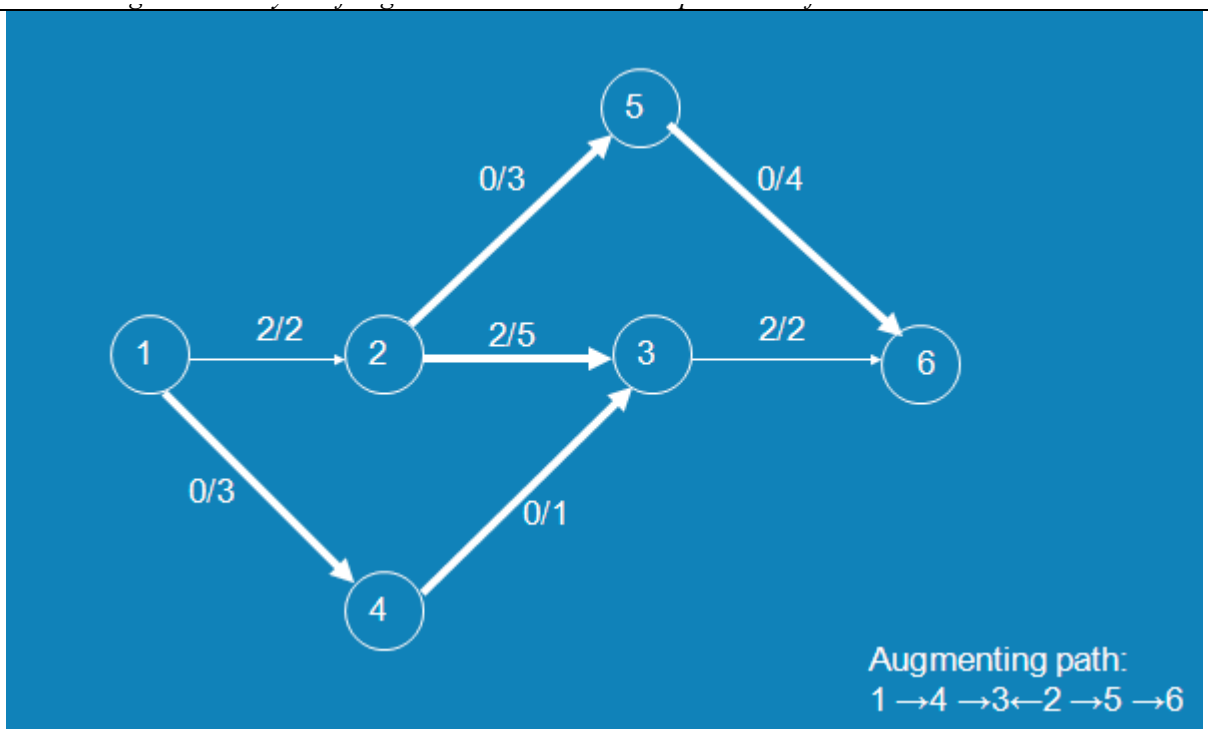
subject to

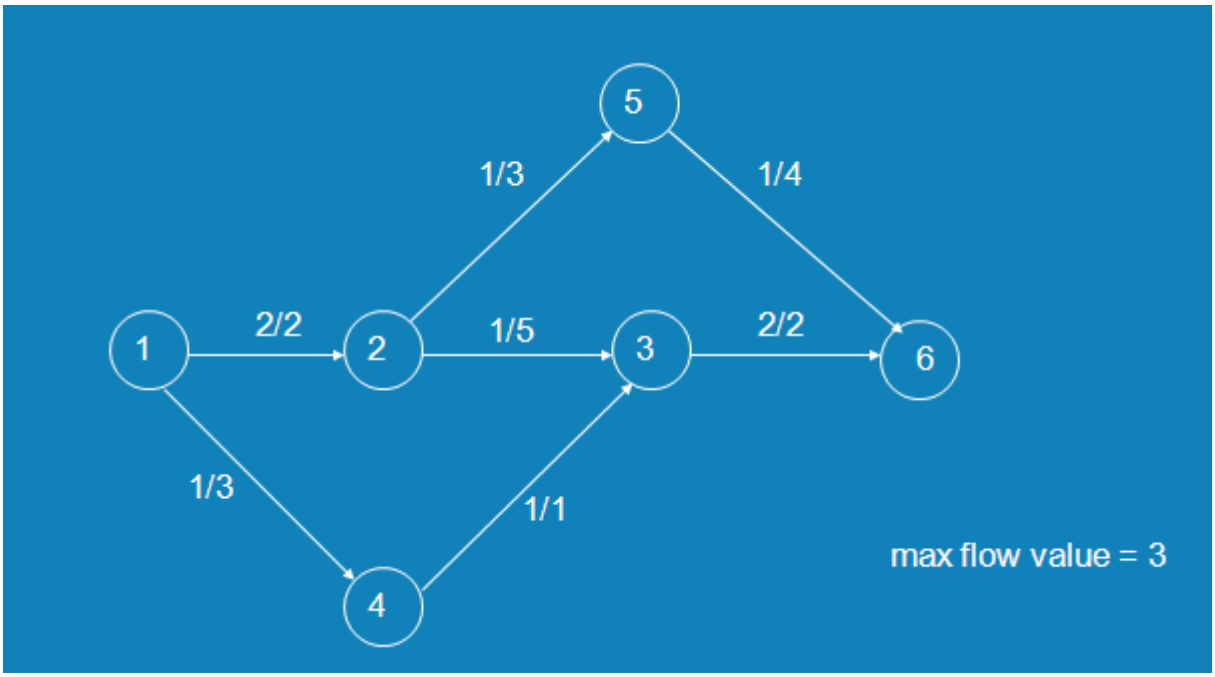
$$\sum_{j:(j,i)\in E} x_{ji} - \sum_{j:(i,j)\in E} x_{ij} = 0 \quad \text{for } i = 2, 3, \ldots, n-1$$

$$0 \le x_{ij} \le u_{ij} \quad \text{for every edge } (i,j) \in E$$

Example 1







Finding a flow-augmenting path

To find a flow-augmenting path for a flow x, consider paths from source to sink in the underlying undirected graph in which any two consecutive vertices i,j are either:

- connected by a directed edge (i to j) with some positive unused capacity $r_{ij} = u_{ij} - x_{ij}$
  - known as *forward edge* ($\rightarrow$)

OR

- connected by a directed edge (j to i) with positive flow x_{ji}
  - known as *backward edge* ($\leftarrow$)

If a flow-augmenting path is found, the current flow can be increased by r units by increasing x_{ij} by r on each forward edge and decreasing x_{ji} by r on each backward edge, where

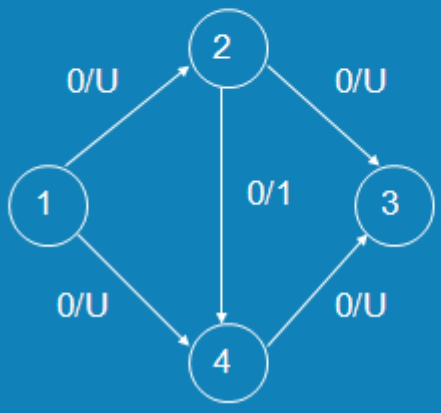
$r = \min \{r_{ij}$ on all forward edges, x_{ji} on all backward edges$\}$

- Assuming the edge capacities are integers, r is a positive integer
- On each iteration, the flow value increases by at least 1
- Maximum value is bounded by the sum of the capacities of the edges leaving the source; hence the augmenting-path method has to stop after a finite number of iterations
- The final flow is always maximum, its value doesn't depend on a sequence of augmenting paths used

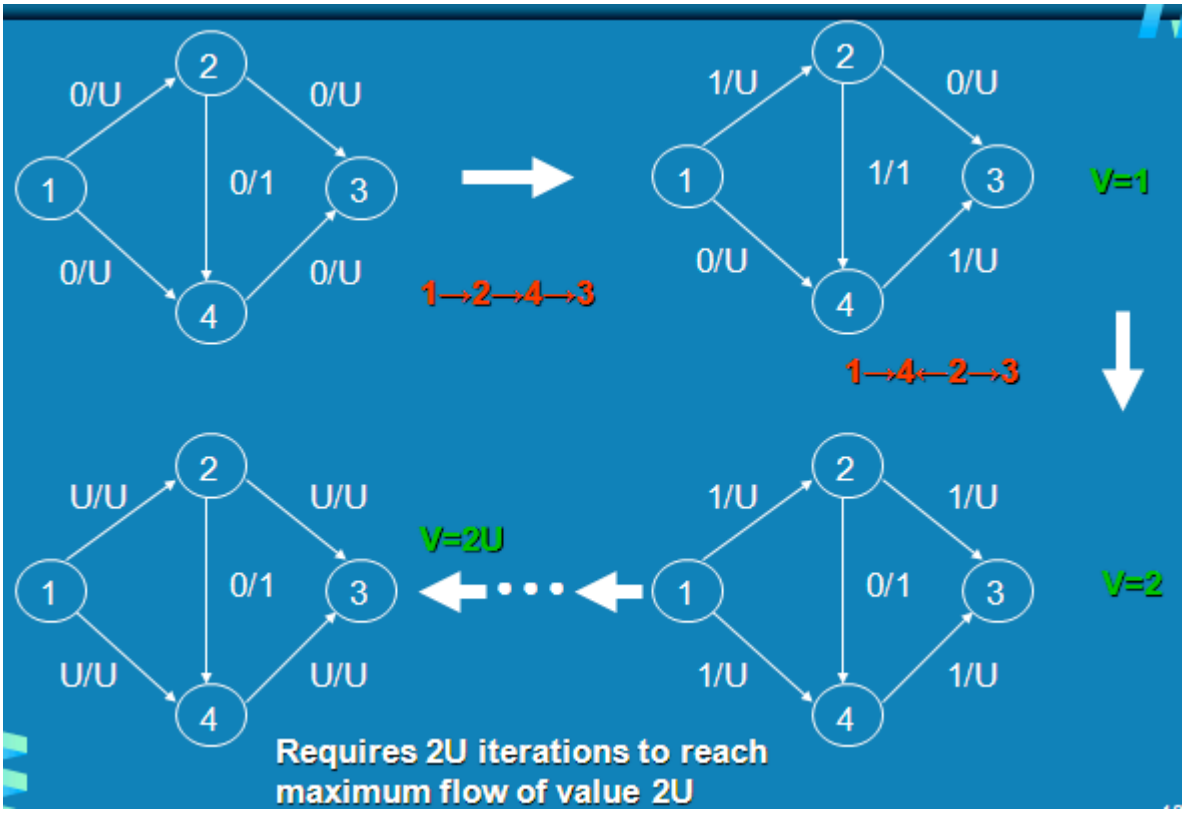
Performance degeneration of the method

- The augmenting-path method doesn't prescribe a specific way for generating flow-augmenting paths
- Selecting a bad sequence of augmenting paths could impact the method's efficiency

Example 2



U = large positive integer



6	Write the pseudo code for Shortest Augmenting Path method and explain in detail.

Generate augmenting path with the least number of edges by BFS as follows.

Starting at the source, perform BFS traversal by marking new (unlabeled) vertices with two labels:

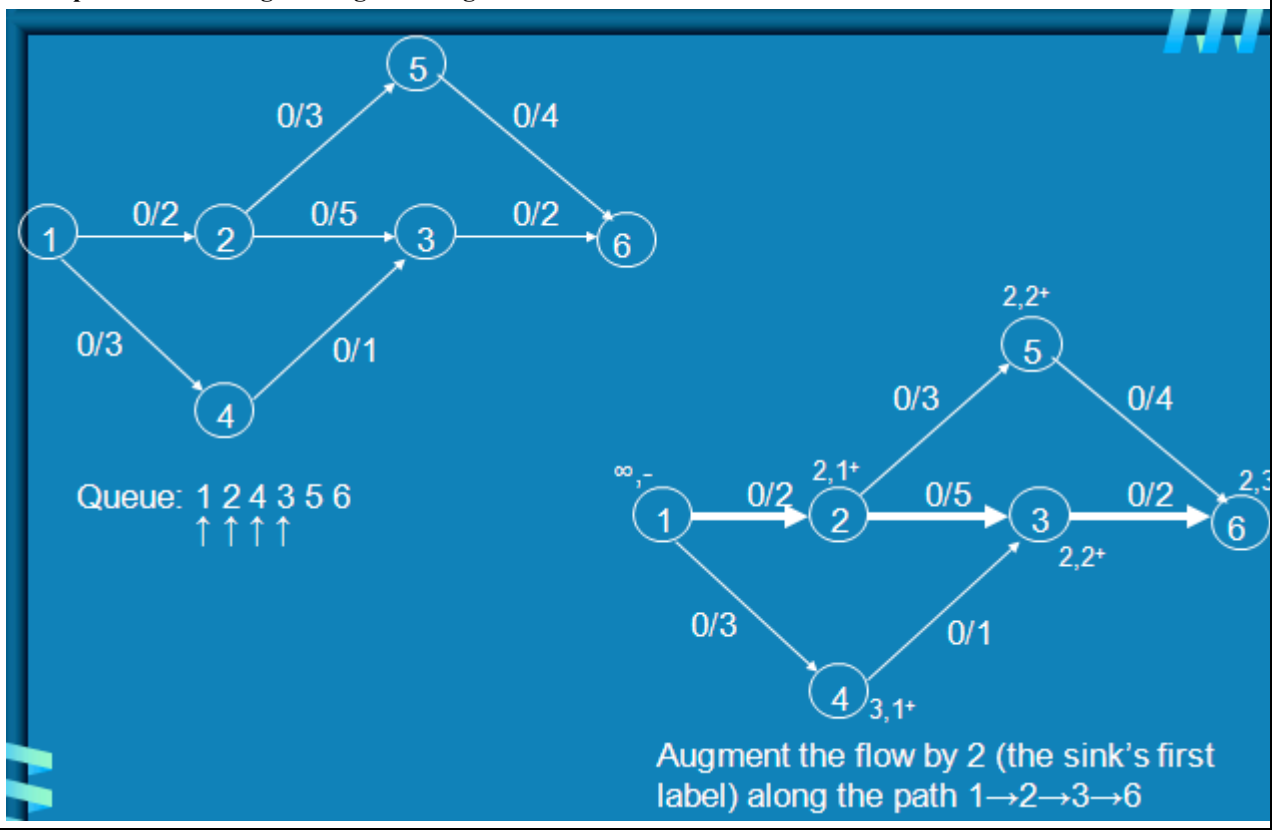
- **first label – indicates the amount of additional flow that can be brought from the source to the vertex being labeled**
- **second label – indicates the vertex from which the vertex being labeled was reached, with "+" or "–" added to the second label to indicate whether the vertex was reached via a forward or backward edge**

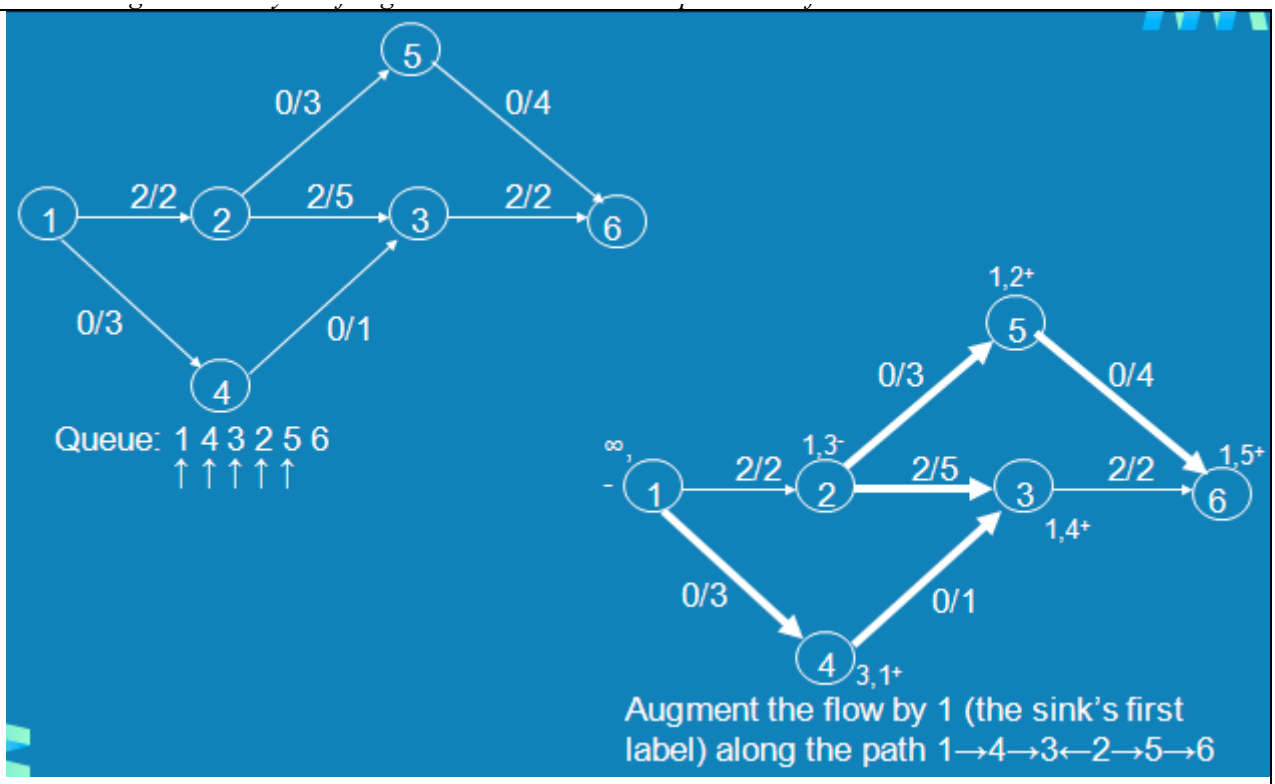
Vertex labeling

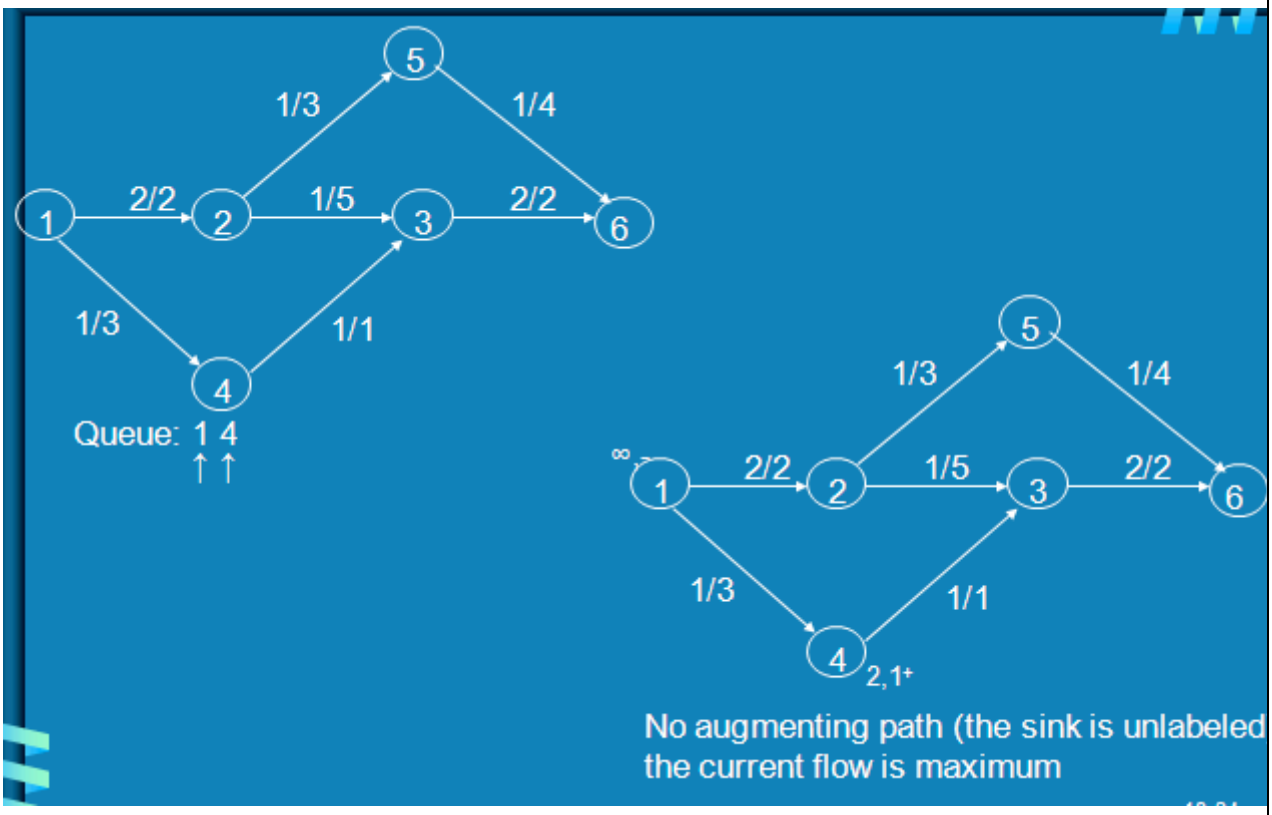
- **The source is always labeled with ∞,-**
- **All other vertices are labeled as follows:**
  - **If unlabeled vertex *j* is connected to the front vertex *i* of the traversal queue by a directed edge from *i* to *j* with positive unused capacity $r_{ij} = u_{ij} - x_{ij}$ (forward edge), vertex *j* is labeled with l_j, i^+, where $l_j = \min\{l_i, r_{ij}\}$**
  - **If unlabeled vertex *j* is connected to the front vertex *i* of the traversal queue by a directed edge from *j* to *i* with positive flow x_{ji} (backward edge), vertex *j* is labeled l_j, i^-, where $l_j = \min\{l_i, x_{ji}\}$**

- If the sink ends up being labeled, the current flow can be augmented by the amount indicated by the sink's first label
- The augmentation of the current flow is performed along the augmenting path traced by following the vertex second labels from sink to source; the current flow quantities are increased on the forward edges and decreased on the backward edges of this path
- If the sink remains unlabeled after the traversal queue becomes empty, the algorithm returns the current flow as maximum and stops

Example: Shortest-Augmenting-Path Algorithm




5
0/3
0/4
1
2/2
2
2/5
3
2/2
6
0/3
0/1
4
Queue: 1 4 3 2 5 6
1,2+
5
0/3
0/4
∞,
-
1
2/2
1,3-
2
2/5
3
2/2
1,5+
6
1,4+
0/3
0/1
4
3,1+
Augment the flow by 1 (the sink's first label) along the path 1→4→3←2→5→6


5
1/3
1/4
1
2/2
2
1/5
3
2/2
6
1/3
1/1
4
Queue: 1 4
5
1/3
1/4
∞,
-
1
2/2
2
1/5
3
2/2
6
1/3
1/1
4
2,1+
No augmenting path (the sink is unlabeled
the current flow is maximum

Shortest-augmenting-path algorithm

Input: A network with single source 1, single sink n, and
positive integer capacities u_{ij} on its edges (i, j)

Output: A maximum flow x

assign $x_{ij} = 0$ to every edge (i, j) in the network
label the source with $\infty, -$ and add the source to the empty queue Q
while not $Empty(Q)$ **do**
  $i \leftarrow Front(Q)$; $Dequeue(Q)$
  for every edge from i to j **do** //forward edges
    if j is unlabeled
      $r_{ij} \leftarrow u_{ij} - x_{ij}$
      if $r_{ij} > 0$
        $l_j \leftarrow \min\{l_i, r_{ij}\}$; label j with l_j, i^+
        $Enqueue(Q, j)$
  for every edge from j to i **do** //backward edges
    if j is unlabeled
      if $x_{ji} > 0$
        $l_j \leftarrow \min\{l_i, x_{ji}\}$; label j with l_j, i^-
        $Enqueue(Q, j)$
  if the sink has been labeled
    //augment along the augmenting path found
    $j \leftarrow n$ //start at the sink and move backwards using second labels
    while $j \neq 1$ //the source hasn't been reached
      if the second label of vertex j is i^+
        $x_{ij} \leftarrow x_{ij} + l_n$
      else //the second label of vertex j is i^-
        $x_{ji} \leftarrow x_{ji} - l_n$
      $j \leftarrow i$
    erase all vertex labels except the ones of the source
    reinitialize Q with the source
return x //the current flow is maximum

7 Explain maximum matching problem in bipartite graphs.

In many situations we are faced with a problem of pairing elements of two sets. The traditional example is boys and girls for a dance, but you can easily think of more serious applications. It is convenient to represent elements of two given sets by vertices of a graph, with edges between vertices that can be paired. A ***matching*** in a graph is a subset of its edges with the property that no two edges share a vertex. A ***maximum matching***—more precisely, a ***maximum cardinality matching***—is a matching with the largest number of edges.

The maximum-matching problem is the problem of finding a maximum matching in a given graph. For an arbitrary graph, this is a rather difficult problen

a ***bipartite graph***, all the vertices can be partitioned into two disjoint sets V and U, not necessarily of the same size, so that every edge connects a vertex in one of these sets to a vertex in the other set. In other words, a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also said to be ***2-colorable***. The graph in Figure 10.8 is bipartite. It is not difficult to prove that a graph is bipartite if and only if it does

not have a cycle of an odd length.

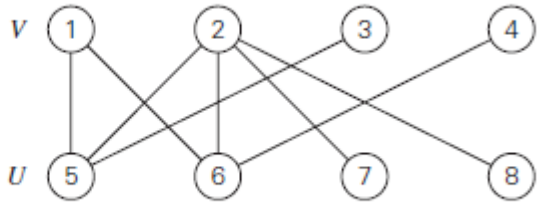


FIGURE 10.8 Example of a bipartite graph.

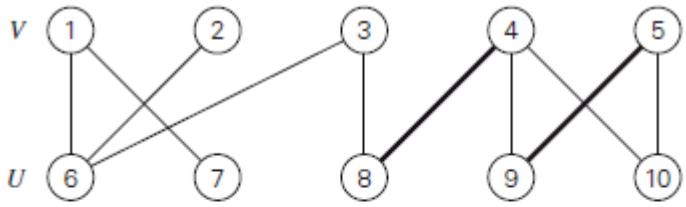
the vertex set of a given bipartite graph has been already partitioned into sets V and U as required by the definition (see Problem 8 in Exercises 3.5).

Let us apply the iterative-improvement technique to the maximum-cardinality-matching problem. Let M be a matching in a bipartite graph $G = \langle V, U, E \rangle$. How can we improve it, i.e., find a new matching with more edges? Obviously, if every vertex in either V or U is ***matched*** (has a ***mate***), i.e., serves as an endpoint of an edge in M, this cannot be done and M is a maximum matching. Therefore, to have a chance at improving the current matching, both V and U must contain ***unmatched*** (also called ***free***) ***vertices***, i.e., vertices that are not incident to any edge in M. For example, for the matching $M_a = \{(4, 8), (5, 9)\}$ in the graph in Figure 10.9a, vertices 1, 2, 3, 6, 7, and 10 are free, and vertices 4, 5, 8, and 9 are matched.

Another obvious observation is that we can immediately increase a current matching by adding an edge between two free vertices. For example, adding (1, 6) to the matching $M_a = \{(4, 8), (5, 9)\}$ in the graph in Figure 10.9a yields a larger matching $M_b = \{(1, 6), (4, 8), (5, 9)\}$ (Figure 10.9b). Let us now try to find a matching larger than M_b by matching vertex 2. The only way to do this would be to include the edge (2, 6) in a new matching. This inclusion requires removal of (1, 6), which can be compensated by inclusion of (1, 7) in the new matching. This new matching $M_c = \{(1, 7), (2, 6), (4, 8), (5, 9)\}$ is shown in Figure 10.9c.

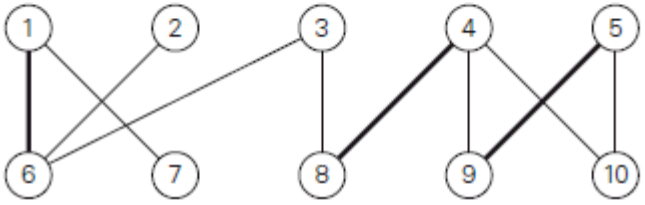
In general, we increase the size of a current matching M by constructing a simple path from a free vertex in V to a free vertex in U whose edges are alternately in $E - M$ and in M. That is, the first edge of the path does not belong to M, the second one does, and so on, until the last edge that does not belong to M. Such a path is called ***augmenting*** with respect to the matching M. For example, the path 2, 6, 1, 7 is an augmenting path with respect to the matching M_b in Figure 10.9b. Since the length of an augmenting path is always odd, adding to the matching M the path's edges in the odd-numbered positions and deleting from it the path's edges in the even-numbered positions yields a matching with one more edge than in M. Such a matching adjustment is called ***augmentation***. Thus, in Figure 10.9, the matching M_b was obtained by augmentation of the matching M_a along the augmenting path 1, 6, and the matching M_c was obtained by augmentation of the matching M_b along the augmenting path 2, 6, 1, 7. Moving further, 3, 8, 4, 9, 5, 10 is an augmenting path for the matching M_c (Figure 10.9c). After adding to M_c the edges (3, 8), (4, 9), and (5, 10) and deleting (4, 8) and (5, 9), we obtain the matching $M_d = \{(1, 7), (2, 6), (3, 8), (4, 9), (5, 10)\}$ shown in Figure 10.9d. The matching M_d is not only a maximum matching but also ***perfect***, i.e., a matching that matches all the vertices of the graph.

Before we discuss an algorithm for finding an augmenting path, let us settle the issue of what nonexistence of such a path means. According to the theorem discovered by the French mathematician Claude Berge, it means the current matching is maximal.



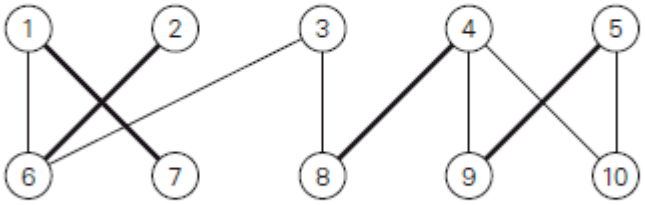
(a)

Augmenting path: 1, 6



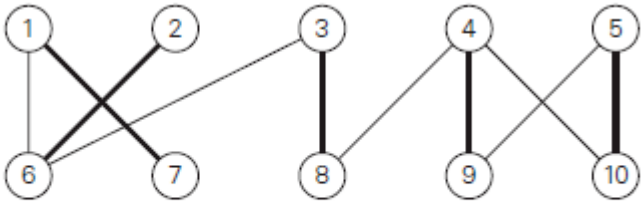
(b)

Augmenting path: 2, 6, 1, 7



(c)

Augmenting path: 3, 8, 4, 9, 5, 10



(d)

Maximum matching

8	Write the pseudo code for maximum bipartite matching.

ALGORITHM *MaximumBipartiteMatching(G)*

```
//Finds a maximum matching in a bipartite graph by a BFS-like traversal
//Input: A bipartite graph G = ⟨V, U, E⟩
//Output: A maximum-cardinality matching M in the input graph
initialize set M of edges with some valid matching (e.g., the empty set)
initialize queue Q with all the free vertices in V (in any order)
while not Empty(Q) do
    w ← Front(Q);  Dequeue(Q)
    if w ∈ V
        for every vertex u adjacent to w do
            if u is free
                //augment
                M ← M ∪ (w, u)
                v ← w
                while v is labeled do
                    u ← vertex indicated by v's label;  M ← M − (v, u)
                    v ← vertex indicated by u's label;  M ← M ∪ (v, u)
                remove all vertex labels
                reinitialize Q with all free vertices in V
                break  //exit the for loop
            else  //u is matched
                if (w, u) ∉ M and u is unlabeled
                    label u with w
                    Enqueue(Q, u)
    else  //w ∈ U (and matched)
        label the mate v of w with w
        Enqueue(Q, v)
return M  //current matching is maximum
```

9 Explain stable marriage algorithm with suitable example.

Stable Marriage Problem

There is a set Y = $\{m_1,\ldots,m_n\}$ of *n* men and a set X = $\{w_1,\ldots,w_n\}$ of *n* women. Each man has a ranking list of the women, and each woman has a ranking list of the men (with no ties in these lists).

A *marriage matching* M is a set of *n* pairs (m_i, w_j).

A pair (*m*, *w*) is said to be a *blocking pair* for matching M if man *m* and woman *w* are not matched in M but prefer each other to their mates in M.

A marriage matching M is called *stable* if there is no blocking pair for it; otherwise, it's called *unstable*.

The *stable marriage problem* is to find a stable marriage matching for men's and women's given preferences.

men's preferences

	1st	2nd	3rd
Bob:	Lea	Ann	Sue
Jim:	Lea	Sue	Ann
Tom:	Sue	Lea	Ann

(a)

women's preferences

	1st	2nd	3rd
Ann:	Jim	Tom	Bob
Lea:	Tom	Bob	Jim
Sue:	Jim	Tom	Bob

(b)

ranking matrix

	Ann	Lea	Sue
Bob	[2,3]	1,2	3,3
Jim	3,1	[1,3]	2,1
Tom	3,2	2,1	[1,2]

(c)

FIGURE 10.11 Data for an instance of the stable marriage problem. (a) Men's preference lists; (b) women's preference lists. (c) Ranking matrix (with the boxed cells composing an unstable matching).

A pair (m, w), where $m \in Y$, $w \in X$, is said to be a ***blocking pair*** for a marriage matching M if man m and woman w are not matched in M but they prefer each other to their mates in M. For example, (Bob, Lea) is a blocking pair for the marriage matching $M = \{$(Bob, Ann), (Jim, Lea), (Tom, Sue)$\}$ (Figure 10.11c) because they are not matched in M while Bob prefers Lea to Ann and Lea prefers Bob to Jim. A marriage matching M is called ***stable*** if there is no blocking pair for it; otherwise, M is called ***unstable***. According to this definition, the marriage matching in Figure 10.11c is unstable because Bob and Lea can drop their designated mates to join in a union they both prefer. The ***stable marriage problem*** is to find a stable marriage matching for men's and women's given preferences.

Surprisingly, this problem always has a solution. (Can you find it for the instance in Figure 10.11?) It can be found by the following algorithm.

Stable marriage algorithm

Input: A set of n men and a set of n women along with rankings of the women by each man and rankings of the men by each woman with no ties allowed in the rankings

Output: A stable marriage matching

Step 0 Start with all the men and women being free.

Step 1 While there are free men, arbitrarily select one of them and do the following:

Proposal The selected free man m proposes to w, the next woman on his preference list (who is the highest-ranked woman who has not rejected him before).

Response If w is free, she accepts the proposal to be matched with m. If she is not free, she compares m with her current mate. If she prefers m to him, she accepts m's proposal, making her former mate free; otherwise, she simply rejects m's proposal, leaving m free.

Step 2 Return the set of n matched pairs.

Before we analyze this algorithm, it is useful to trace it on some input.

		Ann	Lea	Sue	
Free men: Bob, Jim, Tom	Bob	2, 3	[1,2]	3, 3	Bob proposed to Lea
	Jim	3, 1	1, 3	2, 1	Lea accepted
	Tom	3, 2	2, 1	1, 2	
		Ann	Lea	Sue	
Free men: Jim, Tom	Bob	2, 3	[1,2]	3, 3	Jim proposed to Lea
	Jim	3, 1	1, 3	2, 1	Lea rejected
	Tom	3, 2	2, 1	1, 2	
		Ann	Lea	Sue	
Free men: Jim, Tom	Bob	2, 3	[1,2]	3, 3	Jim proposed to Sue
	Jim	3, 1	1, 3	[2,1]	Sue accepted
	Tom	3, 2	2, 1	1, 2	
		Ann	Lea	Sue	
Free men: Tom	Bob	2, 3	[1,2]	3, 3	Tom proposed to Sue
	Jim	3, 1	1, 3	[2,1]	Sue rejected
	Tom	3, 2	2, 1	1, 2	
		Ann	Lea	Sue	
Free men: Tom	Bob	2, 3	1, 2	3, 3	Tom proposed to Lea
	Jim	3, 1	1, 3	[2,1]	Lea replaced Bob with Tom
	Tom	3, 2	[2,1]	1, 2	
		Ann	Lea	Sue	
Free men: Bob	Bob	[2,3]	1, 2	3, 3	Bob proposed to Ann
	Jim	3, 1	1, 3	[2,1]	Ann accepted
	Tom	3, 2	[2,1]	1, 2	

Application of the stable marriage algorithm. An accepted proposal is indicated by a boxed cell; a rejected proposal is shown by an underlined cell.

Let us discuss properties of the stable marriage algorithm.

THEOREM The stable marriage algorithm terminates after no more than n^2 iterations with a stable marriage output.

PROOF The algorithm starts with n men having the total of n^2 women on their ranking lists. On each iteration, one man makes a proposal to a woman. This reduces the total number of women to whom the men can still propose in the future because no man proposes to the same woman more than once. Hence, the algorithm must stop after no more than n^2 iterations.

Let us now prove that the final matching M is a stable marriage matching. Since the algorithm stops after all the n men are one-one matched to the n women, the only thing that needs to be proved is the stability of M. Suppose, on the contrary, that M is unstable. Then there exists a blocking pair of a man m and a woman w who are unmatched in M and such that both m and w prefer each other to the persons they are matched with in M. Since m proposes to every woman on his ranking list in decreasing order of preference and w precedes m's match in M, m must have proposed to w on some iteration. Whether w refused m's proposal or accepted it but replaced him on a subsequent iteration with a higher-ranked match, w's mate in M must be higher on w's preference list than m because the rankings of the men matched to a given woman may only improve on each iteration of the algorithm. This contradicts the assumption that w prefers m to her final match in M. ■

The stable marriage algorithm has a notable shortcoming. It is not "gender neutral." In the form presented above, it favors men's preferences over women's

preferences. We can easily see this by tracing the algorithm on the following instance of the problem:

	woman 1	woman 2
man 1	1, 2	2, 1
man 2	2, 1	1, 2

The algorithm obviously yields the stable matching $M = \{$(man 1, woman 1), (man 2, woman 2)$\}$. In this matching, both men are matched to their first choices, which is not the case for the women. One can prove that the algorithm always yields a stable matching that is ***man-optimal***: it assigns to each man the highest-ranked woman possible under any stable marriage. Of course, this gender bias can be reversed, but not eliminated, by reversing the roles played by men and women in the algorithm, i.e., by making women propose and men accept or reject their proposals.

There is another important corollary to the fact that the stable marriage algorithm always yields a gender-optimal stable matching. It is easy to prove that a man (woman)-optimal matching is unique for a given set of participant preferences. Therefore the algorithm's output does not depend on the order in which the free men (women) make their proposals. Consequently, we can use any data structure we might prefer—e.g., a queue or a stack—for representing this set with no impact on the algorithm's outcome.

10 Explain the following

a. Blocking pair

A pair (m, w), where $m \in Y$, $w \in X$, is said to be a ***blocking pair*** for a marriage matching M if man m and woman w are not matched in M but they prefer each other to their mates in M. For example, (Bob, Lea) is a blocking pair for the marriage matching $M = \{$(Bob, Ann), (Jim, Lea), (Tom, Sue)$\}$ (Figure 10.11c) because they are not matched in M while Bob prefers Lea to Ann and Lea prefers Bob to Jim.

b. Stable marriage problem

The ***stable marriage problem***
is to find a stable marriage matching for men's and women's given preferences.

c. Man-optimal and d. Woman-optimal

		woman 1	woman 2
	man 1	1, 2	2, 1
	man 2	2, 1	1, 2

The algorithm obviously yields the stable matching M = {(man 1, woman 1), (man 2, woman 2)}. In this matching, both men are matched to their first choices, which is not the case for the women. One can prove that the algorithm always yields a stable matching that is ***man-optimal***: it assigns to each man the highest-ranked woman possible under any stable marriage. Of course, this gender bias can be reversed, but not eliminated, by reversing the roles played by men and women in the algorithm, i.e., by making women propose and men accept or reject their proposals.

The stable matching produced by the algorithm is always *man-optimal*: each man gets the highest rank woman on his list under any stable marriage. One can obtain the *woman-optimal* matching by making women propose to men

11 Solve the instance of the stable marriage problem given by the ranking matrix and find the stable and unstable matching.

Ranking matrix

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Instance of the Stable Marriage Problem

An instance of the stable marriage problem can be specified either by two sets of preference lists or by a ranking matrix, as in the example below.

men's preferences

	1st	2nd	3rd
Bob:	Lea	Ann	Sue
Jim:	Lea	Sue	Ann
Tom:	Sue	Lea	Ann

women's preferences

	1st	2nd	3rd
Ann:	Jim	Tom	Bob
Lea:	Tom	Bob	Jim
Sue:	Jim	Tom	Bob

ranking matrix

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

{(Bob, Ann) (Jim, Lea) (Tom, Sue)} is unstable

{(Bob, Ann) (Jim, Sue) (Tom, Lea)} is stable

Stable Marriage Algorithm

Step 0 Start with all the men and women being free

Step 1 While there are free men, arbitrarily select one of them and do the following:

Proposal The selected free man *m* proposes to *w*, the next woman on his preference list

Response If *w* is free, she accepts the proposal to be matched with *m*. If she is not free, she compares *m* with her current mate. If she prefers *m* to him, she accepts *m*'s proposal, making her former mate free; otherwise, she simply rejects *m*'s proposal, leaving *m* free

Step 2 Return the set of *n* matched pairs

Free men:
Bob, Jim, Tom

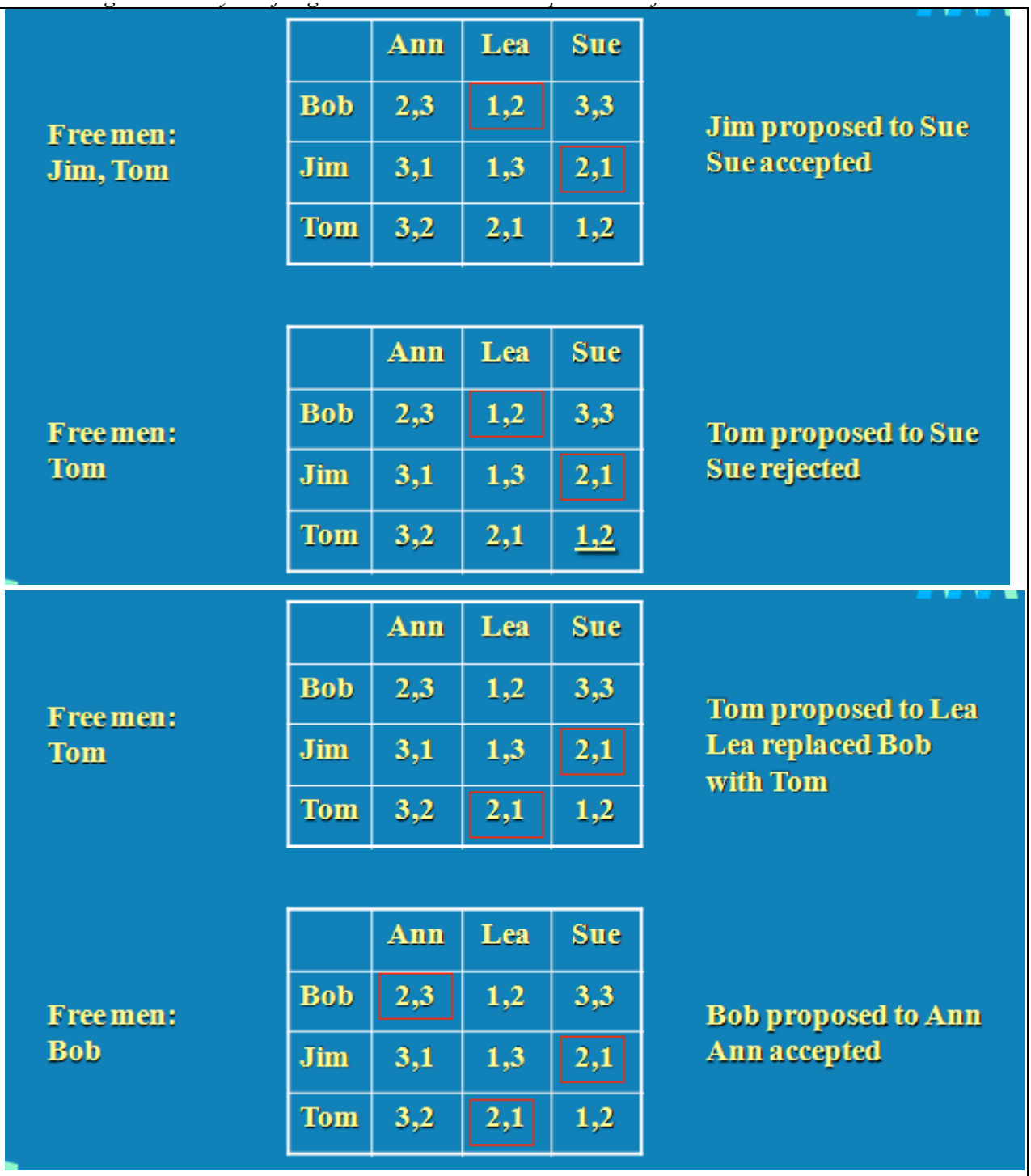
	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

Bob proposed to Lea
Lea accepted

Free men:
Jim, Tom

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

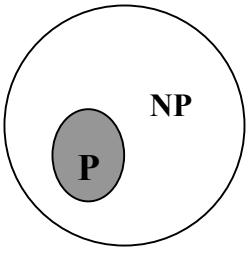
Jim proposed to Lea
Lea rejected

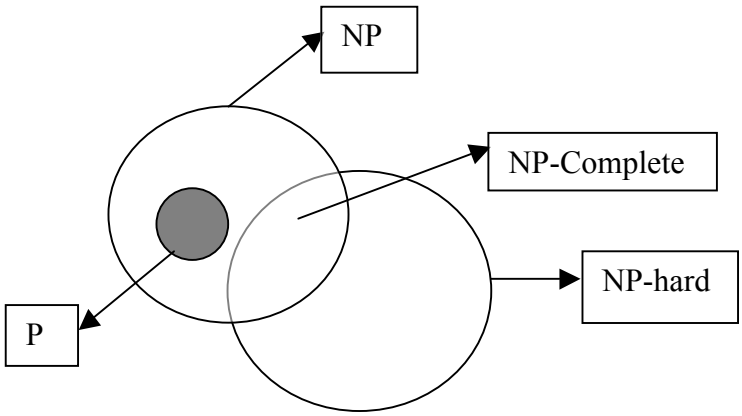


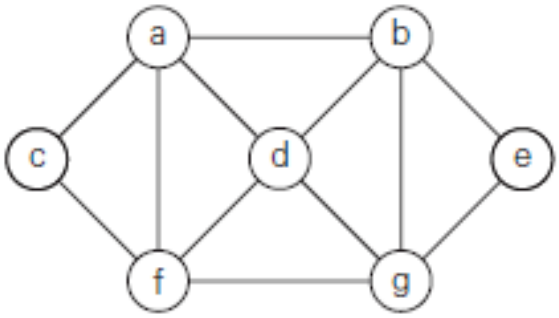
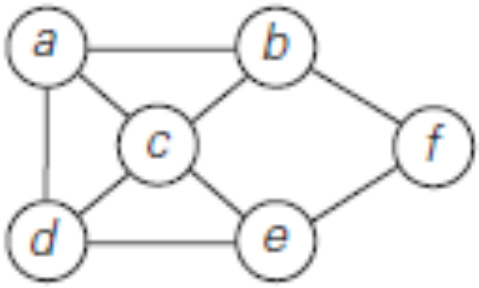
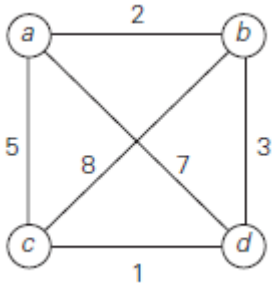
UNIT-V / PART-A

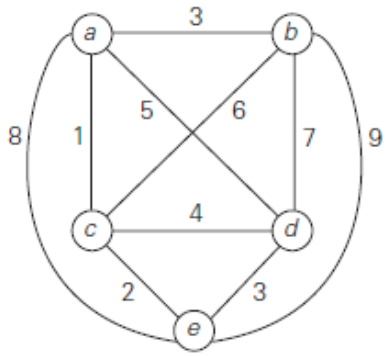
1	**Define trivial lower bound of a class.** The simplest method of obtaining a lower-bound class is based on counting the number of items in the problem's input that must be processed and the number of output items that need to be produced. Since any algorithm must at least "read" all the items it needs to process and "write" all its outputs, such a count yields a trivial lower bound.
2	**Define decision tree.** A decision tree is a flowchart-like structure in which internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. In decision analysis a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated. A decision tree consists of 3 types of nodes:

	✓ Decision nodes - commonly represented by squares ✓ Chance nodes - represented by circles ✓ End nodes - represented by triangles
3	**Define tournament tree.** A tournament tree is a complete binary tree reflecting results of a "knockout tournament": its leaves represent n players entering the tournament, and each internal node represents a winner of a match played by the players represented by the node's children. Hence, the winner of the tournament is represented by the root of the tree.
4	**Define tractable problems.** Problems that can be solved in polynomial time are called tractable.
5	**Define intractable problems.** Problems that cannot be solved in polynomial time are called intractable.
6	**On what basis problems are classified?** Problems are classified into two types based on time complexity. They are ✓ Polynomial (P) Problem. ✓ Non-Polynomial (NP) Problem
7	**Define Polynomial (P) problem.** Class P is a class of decision problems that can be solved in polynomial time by (deterministic) algorithms. This class of problems is called polynomial.
8	**Define Non Polynomial (NP) problem.** ***(Dec 06)*** Class NP is the class of decision problems that can be solved by nondeterministic polynomial algorithms. This class of problems is called nondeterministic polynomial
9	**Give the classification of Non Polynomial problem.** Non-Polynomial (NP) problems are classified into two types. They are: ✓ NP-Hard ✓ NP-Complete
10	**Give some examples of Polynomial problem.** ✓ Selection sort ✓ Bubble Sort ✓ String Editing ✓ Factorial, etc.
11	**Give some examples of Non-Polynomial problem.(*Dec 06)*** ✓ Travelling Salesman Problem, ✓ Knapsack Problem.
12	**Define Deterministic algorithm** It has a property that the result of every operation is uniquely defined, and then this type of algorithm is referred to as Deterministic algorithm. Such an algorithm agrees with the way programs are executed on a computer. The machine executing such algorithm is refereed as Deterministic machine.
13	**Define Non-Deterministic algorithm.** It has a property that the result of every operation is not uniquely defined, but subject to set of possibilities, then this type of algorithms is referred to as Non-deterministic algorithm. The possible functions are ✓ Choice (S)-It arbitrarily chooses one of the value of set S ✓ Failure ()-signals an Unsuccessful completion ✓ Success ()-signals an Successful completion
14	**Give an example for Class P problem.** ✓ Graph Coloring.
15	**Give an example for Class NP problem.** ✓ Traveling Salesman Problem. ✓ Knapsack problem
16	**Define decision problem.**

	Any problem for which the answer is either zero or one is called a decision problem. An algorithm for a decision problem is termed a Decision algorithm.
17	**Define optimization Problem.** Any problem that involves the identification of an optimal (maximum or minimum) value of a given cost function is known as an optimization problem. An Optimization algorithm is used to solve an optimization problem.
18	**When a decision problem is said to be polynomially reducible?** A decision problem D1 is said to be polynomially reducible to a decision problem D2, if there exists a function t that transforms instances of D1 to instances of D2 such that: ✓ t maps all yes instances of D1 to yes instances of D2 and all no instances of D1 to no instances of D2 ✓ It is computable by a polynomial time algorithm
19	**When decision problem D is said to be NP-complete?** A decision problem D is said to be NP-complete if: ✓ it belongs to class NP ✓ every problem in NP is polynomially reducible to D
20	**Define backtracking.** ***(June 06)*** The principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as follows. If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component. If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.
21	**Define state space tree.** ***(Dec 06)*** It is convenient to implement this kind of processing by constructing a tree of choices being made, called the state-space tree. Its root represents an initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of a solution; the nodes of the second level represent the choices for the second component, and so on.
22	**When a node in a state space tree is said to promising and non promising?** A node in a state-space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise, it is called non promising. Leaves represent either non promising dead ends or complete solutions found by the algorithm.
23	**Define n-queens problem.** The problem is to place n queens on an n × n chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.
24	**Mention the relation between P and NP.** 

25	**Mention the relation between P, NP, NP-Hard and NP Complete Problem.** 
26	**Define Hamiltonian circuit.*(June 06, Dec14)*** Let G= (V, E) the € connected graph, with n vertices. A Hamiltonian cycle is a round trip path along n edges of G that visits every vertex once and returns to its starting position.
27	**What is meant by sum of subset problem?** The sum of subset problem is to find a subset of a given set A = $\{a_1, \ldots, a_n\}$ of n positive integers whose sum is equal to a given positive integer d.
28	**Define assignment problem.** Assignment problem is the problem of assigning n people to n jobs so that the total cost of the assignment is as small as possible.
29	**Define branch and bound method.** ✓ Branch and bound is an algorithm that enhances the idea of generating a state space tree with idea of estimating the best value obtainable from a current node of the decision tree ✓ If such an estimate is not superior to the best solution seen up to that point in the processing, the node is eliminated from further consideration.
30	**Define NP hard problems.** If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time.
31	**Define NP complete problems.(*Dec 06)*** A problem that is NP-complete has the property that it can be solved in polynomial time iff if all other NP-complete problems can also be solved in polynomial time.
32	**Define cost of tour. *(Dec 14)*** Cost of a tour is defined as the sum of the cost of the edges on the tour.
33	**Define Live and Dead nodes. *(Dec 14)*** ✓ Live node → a node which has been generated and all of whose children have not yet been generated. ✓ E-node → the live node whose children are currently being generated. ✓ Dead node → a generated node which is not to be expanded further or all of whose children have been generated.
	UNIT-V / PART-B
1	Explain decision tree with an example.
2	What is backtracking? Write the template of general backtracking algorithm and explain in detail with suitable example. ***(June 07)***
3	Explain n-queen's problem. Draw a portion of the state space tree and perform backtracking search for a solution to 4-queens problem. ***(June 06,Dec07,14)***
4	Explain how backtracking method is applied to solve subset sum problem with suitable example***.(June 0,Dec 07)***
5	Write a pseudo code for backtracking algorithm and apply backtracking to solve the following instances of the subset sum problem: S= {1, 3, 4,5} d=11 and d=8. ***(Dec 06)***
6	Explain in detail about Hamiltonian circuit problem with suitable example.

7	Explain Hamiltonian circuit in a graph. Use backtracking to get a Hamiltonian circuit of following the graph.. 
8	Explain Hamiltonian circuit in a graph. Use backtracking to get a Hamiltonian circuit of following the graph.***(Dec 07)*** 
9	Explain assignment problem using branch and bound method.
10	Solve the following assignment problem using branch and bound technique. Explain in detail how branch and bound technique is useful for solving assignment problems. $C = \begin{matrix} \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} & \begin{matrix} \text{person } a \\ \text{person } b \\ \text{person } c \\ \text{person } d \end{matrix} \end{matrix}$
11	Explain knapsack problem using branch and bound technique. ***(Dec 07)***
12	Explain travelling salesman problem using branch and bound technique.
13	Explain the concept of P, NP, NP hard and NP complete.
14	Solve the following instance of the travelling salesman problem by branch and bound method and explain in detail. 
15	Solve the following instance of the travelling salesman problem by branch and bound method and explain in detail. ***(Dec 07)***



16 Apply the branch and bound algorithm to solve the following knapsack problem and explain in detail.

item	weight	value
1	10	$100
2	7	$63
3	8	$56
4	4	$12

The knapsack capacity W is 16.

17 Explain in detail about approximation algorithms for NP hard problems. How all you quantify the accuracy of an approximation algorithm?

18 Explain approximation algorithm for travelling salesman problem with an example.

19 Explain approximation algorithm for knapsack problem.

20 Explain in detail about branch and bound technique with an example. ***(June 07)***

21 Apply the branch and bound algorithm to solve the following knapsack problem and explain in detail. ***(Dec 06)***

Item	Weight	value
1	4	$40
2	7	$42
3	5	$25
4	3	$12

The knapsack capacity W is 10.

22 Apply the branch and bound algorithm to solve the following knapsack problem and explain in detail. ***(Dec 06)***

Item	Weight	value
1	2	1
2	3	2
3	4	5

The knapsack capacity W is 6.