Assignment - 1.

Q.1) What is Algorithm?

→ Algorithm is a step-by-step process to perform some action or solve a problem.

Q.2) Define Properties of algorithms.

→ The Properties of algorithms are as follows -

i) input - In this algorithm uses values from a specific set in { }.

ii) output - For each input the algorithm produces a values from specific task. Every input has an output.

iii) Precision - In this steps are precisely defined.

iv) correctness - Input is defined for that output is correct and desired.

v) Finiteness - In this Output after finite number of steps for each input.

vi) Determination - The Result should be guaranteed.

vii) Generality - Procedure applies to all problems not a special subset.

Q.3) Define Time complexity.

→ It defines that how much time it consumes to execute the program is known as Time Complexity.

Q.4) Define Space Complexity.

→ It defines that the amount of memory space required for an algorithm or program during the execution is known as space complexity.

Q.5) What is Binary Search.

→ Binary Search is an efficient algorithm for searching in a sorted array.

Q.6) Sort the following numbers using quick sort.

→ 50 31 71 38 77 81 12 33.

Quick Sort Algorithm –

pivot

50 31 71 | 38 | 77 81 12 33

pivot =< pivot >= pivot.

12 31 33 | 38 | 77 81 50 71

pivot pivot

12 31 33 38 | 50 77 81 71

pivot

pivot

12 31 33 38 50 77 71 81

Sorted –

12 31 33 38 50 71 77 81.

Q.7) Explain Binary Search and divide and conquer with example.

→ Binary search is a searching algorithm used in a sorted array. It repeatedly divides the search interval in half, efficiently narrowing down the search space. Divide and conquer is a problem-solving technique that splits a complex problem into smaller, similiar subproblem. Then it combines the solutions of these problems to deduce the final answer.

eg.

0	1	2	3	4	5	6	7	8	9
5	7	9	13	32	33	42	54	56	88

key = 33

$$mid = \frac{(start + end)}{2} = \frac{0+9}{2} = \frac{9}{2} = 4 \rightarrow \text{mid value}$$

search key > A[mid] skip - 0 → 4

33 > 32 greater = mid + 1 (R.S)

greater

start = mid + 1

5	6	7	8	9
33	42	54	56	88

$$mid = \frac{5+9}{2} = \frac{14}{2} = 7 \rightarrow \text{mid value.}$$

33 < A[mid] skip - 7 → 9

33 < 7 lesser = mid - 1 (L.S).

less

classmate
Date
Page

start = mid + 1

5	6
33	42

$mid = \frac{5+6}{2} = \frac{11}{2} = 5 \rightarrow$ mid value

$33 = A[mid]$

$33 = 33$

It is found at index 5.

8) Simulate merge on data sequence.

77 22 33 44 11 55 66

→ 77 22 33 44 11 55 66

77 22 33 44   11 55 66

77 22   33 44   11 55   66

77   22   33   44   11   55

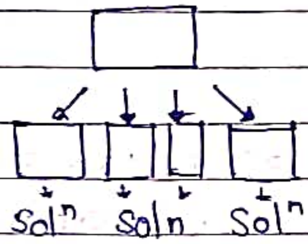
22 77   33 44   11 55

22 33 44 77   11 55 66

11 22 33 44 55 66 77

Q.9) Explain Dynamic Programming with example. (chain matrix).

→ It is a strategy for designing algorithm which is used when problems, breaks down into recursing small problems.



values.

In such problems there can be many solution. Each solution has a values.

ex. chain matrix multiplication.

A = 10 × 30
B = 30 × 5
C = 5 × 60
D = 60 × 8

m[1,1]	m[2,2]	m[3,3]	m[4,4]
A	B	C	D

m[1,2]	m[2,3]	m[3,4]
A · B	B · C	C · D
10×30 30×5	30×5 5×60	5×60 60×8
10 × 30 × 5	30×5×60	5×60×8
1500	9000	2400

m [1, 3]
A· (B·C)
10×30 30×5 5×60
m[1,1] + m[2,3] + 10×30×60
cost of A + cost of B·C + cost of A·(B·C).
= 0 + 9000 + 18000
= 27000

(A·B)·C
m[1,2] + m(3,3) + 10 × 5 × 60
= 1500 + 0 + 3000
= 4500

m[1, 3] = 27000

m[2,4]
B·(C·D)
30×5 5×60 60×8
m[2,2] + m[3,4] + 30×5×8
= 0 + 2400 + 1200
= 3600

(B·C)·D
30×5 5×60 60×8
m[2,3] + m[4,4] + 30×60×8
= 9000 + 0 + 1440
= 10,440

m[1,4]
= (A·B·C·D)
= (A·B)·(C·D)
= (A·B·C)·D

$$= \min \{ m(1,1) + m(2,4) + 10 \times 30 \times 8,$$
$$m(1,2) + m(3,4) + 10 \times 5 \times 8,$$
$$m(1,3) + m(4,4) + 10 \times 6 \times 8 \}$$

$$= \min \{ 0 + 3600 + 2400,$$
$$1500 + 2400 + 400,$$
$$27000 + 0 + 4800 \}$$

$$= \min \{ 6000, 4300, 31800 \}$$

$$= \underline{31800}$$

PLACE function determines the position of the queen in $O(n)$ time. This function is called n times.

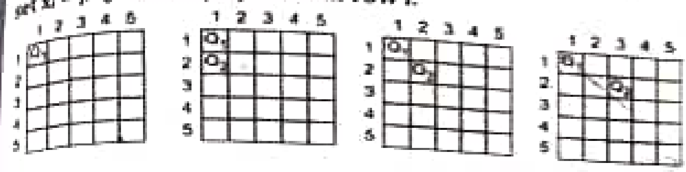
Thus, the recurrence of n-Queen problem is defined as, $T(n) = n^*T(n-1) + n^2$. Solution to recurrence would be $O(n!)$.

Ex. 6.1.1 : Find all possible solutions for five queen problem using backtracking approach.

Soln. :

Solution of N queen problem is represented using n-tuple $X = [x_1, x_2, x_3, \ldots, x_n]$. Each $x_i = 1, 2, \ldots, n$.

If queen Q_i can be placed successfully in column j, then set $x_i = j$. Q_i is always placed in row i.

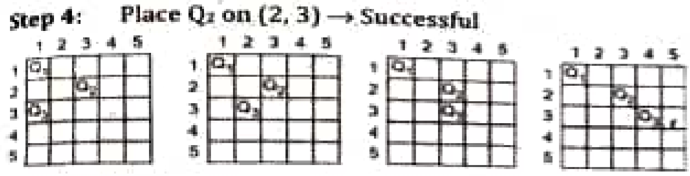


Step 1: Place Q_1 on (1, 1) → Successful

Step 2: Place Q_2 on (2, 1) → Fail → Backtrack

Step 3: Place Q_2 on (2, 2) → Fail → Backtrack

Step 4: Place Q_2 on (2, 3) → Successful

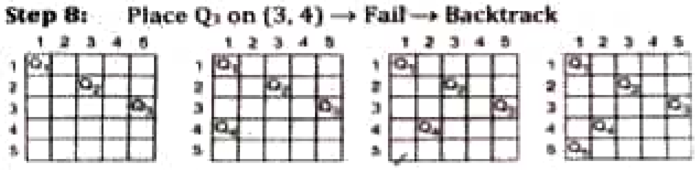


Step 5: Place Q_3 on (3, 1) → Fail → Backtrack

Step 6: Place Q_3 on (3, 2) → Fail → Backtrack

Step 7: Place Q_3 on (3, 3) → Fail → Backtrack

Step 8: Place Q_3 on (3, 4) → Fail → Backtrack

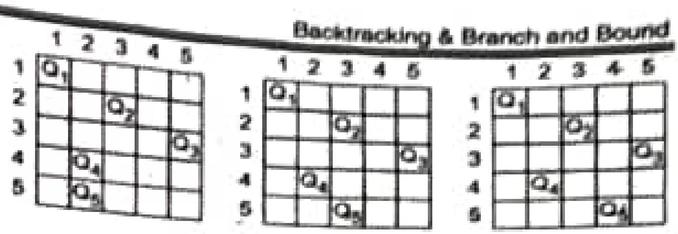


Step 9: Place Q_3 on (3, 5) → Successful

Step 10: Place Q_4 on (4, 1) → Fail → Backtrack

Step 11: Place Q_4 on (4, 2) → Successful

Step 12: Place Q_5 on (5, 1) → Fail → Backtrack



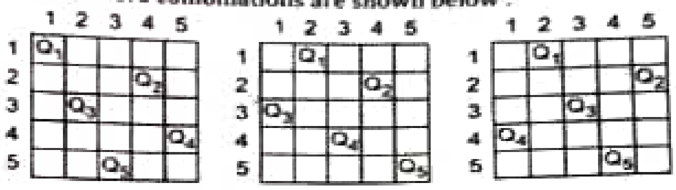
Step 13: Place Q_5 on (5, 2) → Fail → Backtrack

Step 14: Place Q_5 on (5, 3) → Fail → Backtrack

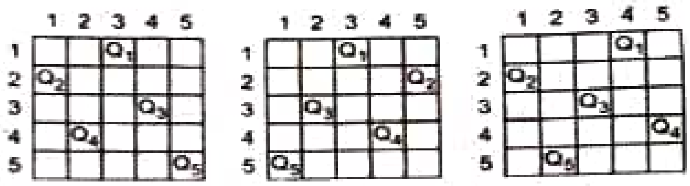
Step 15: Place Q_5 on (5, 4) → Successful

Thus, the solution of this instance is {1, 3, 5, 2, 4}.

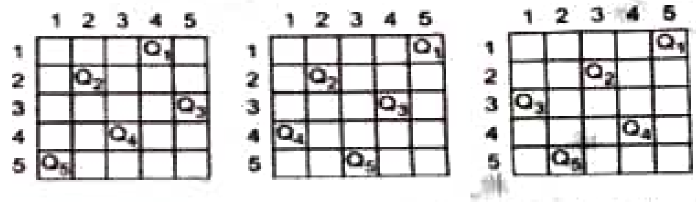
Few more combinations are shown below :



Solution: {1, 4, 2, 5, 3} Solution: {2, 4, 1, 3, 5} Solution: {2, 5, 3, 1, 4}



Solution: {3, 1, 4, 2, 5} Solution: {3, 5, 2, 4, 1} Solution: {4, 1, 3, 5, 2}



Solution :{4, 2, 5, 3, 1} Solution: {5, 2, 4, 1, 3} Solution: {5, 3, 1, 4, 2}

Ex. 6.1.2 : Current configuration is (6, 4, 7, 1) for 8-queens problem. Find answer tuple.

Soln. :

- Tuple (6, 4, 7, 1) indicates the first queen is in the 6th column, the second queen is in the 4th column, the third queen is in the 7th column and forth queen is in the 1st column. Given arrangement of queen is, UNIT-5

ASSIGNMENT-2

Q1. What is network flow?

In combinatorial optimization, network flow problems are a class of computational problems in which the input is a flow network (a graph with numerical capacities on its edges), and the goal is to construct a flow, numerical values on each edge that respect the capacity constraints and that have incoming flow equal to ...

Q2. Define Residual Networks?

Residual Networks: The Residual Network consists of an edge that can admit more net flow. Suppose we have a flow network G = (V, E) with source s and sink t. Let f be a flow in G, and examine a pair of vertices u, v ∈ V. The sum of additional net flow we can push from u to v before exceeding the capacity c (u, v) is the residual capacity of (u, v) given by $c_f(u, v) = c\,(u, v) - f\,(u, v).$

When the net flow f (u, v) is negative, the residual capacity c_f (u,v) is greater than the capacity c (u, v).

Q3. Define Augmenting path.?

Augmenting Path**:** Given a flow network G = (V, E) and a flow f, an **augmenting path** p is a simple path from s to t in the residual networkG_f. By the solution of the residual network, each edge (u, v) on an augmenting path admits some additional positive net flow from u to v without violating the capacity constraint on the edge.

Let G = (V, E) be a flow network with flow f. The **residual capacity** of an augmenting path p is

$$C_f(p) = \min\{C_f(u, v): (u, v) \text{is on } p\}$$

The residual capacity is the maximal amount of flow that can be pushed through the augmenting path. If there is an augmenting path, then each edge on it has a positive capacity. We will use this fact to compute a maximum flow in a flow network.

Q4. Which algorithm is use to solve maximum flow networks?

Ford-Fulkerson algorithm

Ford-Fulkerson algorithm is a greedy approach for calculating the maximum possible flow in a network or a graph. A term, flow network, is used to describe a network of vertices and edges with a source (S) and a sink (T).

Q5.What is maximum flow and minimum flow?

In computer science and optimization theory, the max-flow min-cut theorem states that in a flow network, **the maximum amount of flow passing from the source to the sink is**

equal to the total weight of the edges in a minimum cut, i.e., the smallest total weight of the edges which if removed would disconnect the source ...

10 marks

Q6. Write the steps for maximum flow ?

Finding the maximum flow in a network involves using algorithms like Ford-Fulkerson or Edmonds-Karp. Here are the general steps for finding the maximum flow in a flow network:

1. **Initialize Flow:** Set the flow in all edges to 0.
2. **Choose an Augmenting Path:**
   - Use a suitable algorithm (e.g., DFS, BFS) to find an augmenting path from the source to the sink in the residual graph.
   - An augmenting path is a path in which all edges have residual capacity greater than 0.
3. **Determine Maximum Flow:**
   - Determine the maximum flow that can be sent along the augmenting path. This is the minimum residual capacity of the edges in the path.
4. **Augment Flow:**
   - Update the flow along the augmenting path by adding the determined maximum flow to each edge in the forward direction and subtracting it in the reverse direction.
5. **Update Residual Graph:**
   - Update the residual capacities of the edges in the residual graph based on the new flow.
   - For each forward edge with flow f, subtract f from its residual capacity, and for each reverse edge with flow f, add f to its residual capacity.
6. **Repeat Steps 2-5:**
   - Repeat steps 2-5 until no more augmenting paths can be found in the residual graph.
7. **Calculate Maximum Flow:**
   - Once there are no more augmenting paths, the flow in the network is now at its maximum.
   - Sum up all the flows leaving the source (or entering the sink) to get the maximum flow.
8. **Termination:**
   - Ensure that the algorithm terminates. In some cases, if capacities are not integers, you may need to use algorithms like Edmonds-Karp to guarantee termination.
9. **Optimization:**
   - To optimize the algorithm, consider using advanced data structures or path selection strategies. The choice of the augmenting path selection method can impact the efficiency and termination of the algorithm.

Q7.Write the steps for fordfulkerson algorithm ?

The Ford-Fulkerson algorithm is a method for finding the maximum flow in a flow network. Here are the basic steps of the algorithm:

1. **Initialize Flow:** Set the flow in all edges to 0.
2. **Find Augmenting Paths:**
   - Start with an empty flow.
   - Find an augmenting path from the source to the sink in the residual graph. An augmenting path is a path in which all edges have residual capacity greater than 0.
   - You can use depth-first search (DFS) or breadth-first search (BFS) to find augmenting paths.

3. **Augment Flow:**
   - Once you find an augmenting path, determine the maximum flow that can be sent along this path. This is the minimum residual capacity of the edges in the path.
   - Update the flow along the augmenting path by adding the determined maximum flow to each edge in the forward direction and subtracting it in the reverse direction.
4. **Update Residual Graph:**
   - Update the residual capacities of the edges in the residual graph based on the new flow.
   - For each forward edge with flow f, subtract f from its residual capacity, and for each reverse edge with flow f, add f to its residual capacity.
5. **Repeat Steps 2-4:**
   - Repeat steps 2-4 until no more augmenting paths can be found in the residual graph.
6. **Calculate Maximum Flow:**
   - Once there are no more augmenting paths, the flow in the network is now at its maximum.
   - Sum up all the flows leaving the source (or entering the sink) to get the maximum flow.

It's important to note that the algorithm may not terminate if capacities are not integers. In such cases, you may need to use algorithms like Edmonds-Karp, which guarantees termination by always choosing the shortest augmenting path in terms of the number of edges.

Q8. Difference between fordfulkerson and maximum flow?

"Maximum flow" refers to the overall concept and problem of finding the maximum amount of flow that can be sent from a source node to a sink node in a flow network. "Ford-Fulkerson" is one of the algorithms designed to solve the maximum flow problem.

Here's the difference between the two:

1. **Maximum Flow:**
   - **Definition:** Maximum flow is a concept in graph theory and network flow problems that represents the maximum amount of flow that can be sent from a designated source node to a designated sink node in a flow network.
   - **Objective:** The objective is to determine the optimal distribution of flow along the edges of the network to maximize the flow from the source to the sink, while respecting the capacities of the edges.
2. **Ford-Fulkerson:**
   - **Definition:** Ford-Fulkerson is an algorithmic approach to find the maximum flow in a flow network.
   - **Algorithmic Approach:** Ford-Fulkerson is a generic method for solving the maximum flow problem. It doesn't prescribe a specific rule for choosing augmenting paths, which are paths in the residual graph used to increase the flow.
   - **Termination Issue:** The basic Ford-Fulkerson algorithm doesn't guarantee termination when capacities have fractional values, so variations like Edmonds-Karp address this by always choosing the shortest augmenting path.

In summary, "maximum flow" is the overarching problem, and "Ford-Fulkerson" is one of the algorithms used to solve this problem. There are other algorithms, such as Edmonds-Karp, that are modifications of Ford-Fulkerson with specific rules for choosing augmenting paths to ensure termination.

Q9.Write steps for maximum cut?

Finding a maximum cut in an undirected graph involves partitioning the vertices into two disjoint sets such that the number of edges between the two sets is maximized. Here are the basic steps for finding a maximum cut in a graph:

1. **Initialize Partitions:**
    - Start with an arbitrary partition of the vertices into two sets, A and B.
2. **While Improvements Can Be Made:**
    - Repeat the following steps as long as improvements can be made to the cut:
3. **Compute Cut Size:**
    - Calculate the size of the cut by counting the number of edges crossing the partition (i.e., having one endpoint in set A and the other in set B).
4. **Identify Improving Edges:**
    - Identify the edges that, if moved from one set to the other, would increase the cut size.
5. **Move Edges:**
    - Choose one or more improving edges and move their vertices from one set to the other. This increases the cut size.
6. **Repeat Steps 3-5:**
    - Repeatedly compute the cut size and move improving edges until no further improvements can be made.
7. **Output Result:**
    - The final partition of the vertices into sets A and B represents the maximum cut in the graph.

It's important to note that finding an exact solution to the maximum cut problem is NP-hard. Various heuristic and approximation algorithms are often employed to find near-optimal solutions in a reasonable amount of time. Additionally, there are more sophisticated algorithms, such as the spectral partitioning method, that use linear algebraic techniques to find good approximations of maximum cuts.

Unit - 2

Assignment - 3

Page No.
Date

Q.1. Define disjoint Set ?

→ These are Sets of data Structure ~~of data~~ which Supports 3 operations make Set, Union and find Set. It can be define as the subsets where there is no common element between the two Sets.
Ex :- We have 2 subsets S1 and S2
S1 Contains the element 1, 2, 3, 4
S2 Contains 5, 6, 7, 8
There is no common element between 2 sets.

Q.2. Define Union

→ Union operation is to take 2 different Sets and merge them into 1 Set. Union of Set A and B is defined to be the Set of all those elements which belong to A or B or both and is denoted by A∪B
Let A = {1, 2, 3} B = {3, 4, 5, 6}
A∪B = {1, 2, 3, 4, 5, 6}

Q.3. Define Union by Rank

→ To ensure that when we combine two trees, we try to keep the overall depth of the resulting tree small. This technique used to optimize, the union operation by ensuring that the smaller tree is always attached to the root of the larger tree. This approach prevents the trees

Page No.
Date

from becoming imbalanced, which would lead to inefficient find operations.

Q.4. What are applications of disjoint set.

→ (1) Hashing functions, union find algorithms.
(2) Stack operations.
(3) Heap operations, cycle detection in graph
(4) pushing & popping values.
(5) Job sequencing problem solving.
(6) kruskals algorithm, computer networks.

Q.5. Define lower bound theory.

→ Lower bound theory says that no algorithm can do the job in fewer than L(n) time units for arbitory inputs. Calculation of minimum time that is required to execute an algorithm is known as a lower bound theory. It uses a number of methods to find out the lower bound.
(1) Comparison trees
(2) decision tree.

Q.6. Explain disjoint set with one example.

→ These are the sets of data structure which supports 3 operations make set, union and find set.

.

Page No.
Date

(1) Make Set is the Operation to create set with only one element.

(2) Union Operation is to take 2 different sets and merge them into set

(3) Find Set is an Operation to return an identity of set which is usually an element in set which acts as ~~rep~~ representative of that set

Characteristics –

(i) It keeps a set partitioned into disjoint subsets.

(ii) It allows the efficient union of two subsets.

(iii) It makes it possible to quickly determine of given element belongs to which subset

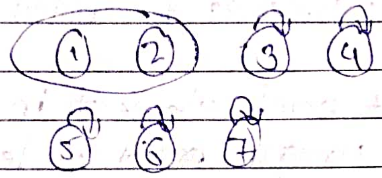
- In the disjoint set each element in a set is represented by a unique root node.
- In the disjoint set two elements belong to the same set if they share the same root node.
- The root node of an element can be found by following the parent pointers until a node is reached that has itself as its parent.

Ex :- Let assume we have 9 nodes initially we will store them in the form of trees. where each tree corresponds to one set and root of the tree will be parent / leader of set

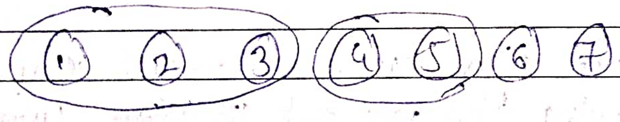
Following Union queries.

Union (1, 2)
Union (2, 3)
Union (4, 5)
union (6, 7)
Union (5, 6)
Union (2, 6)

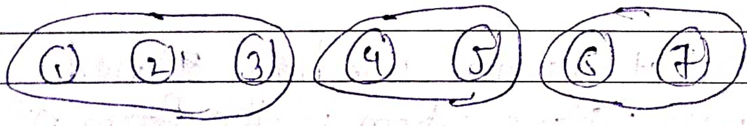
In first query union (1, 2) we need to join two sets i.e. into one.



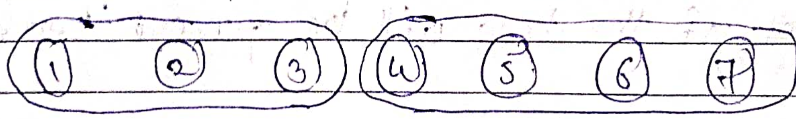
union 4, 5



union 6, 7

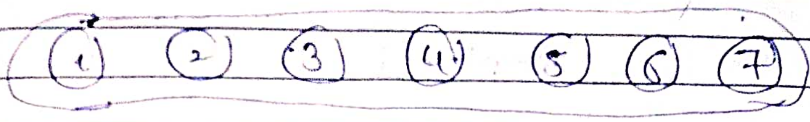


union 5, 6



Page No.
Date

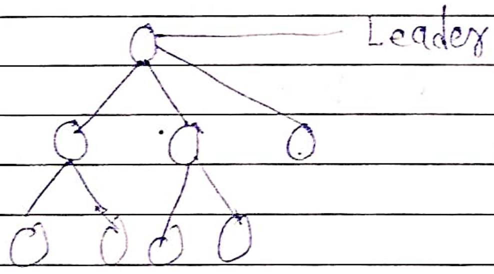
union 2, 6



Q.7. Explain Union by Rank with example.

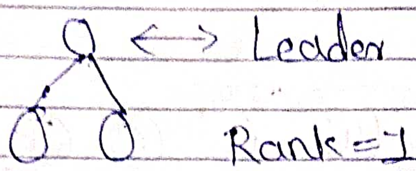
→ We need a new array of integers called rank [] The size of this array is the same as the parent array Parent [].
If i is a representative of a Set rank [i] is the height of the tree representing the set Now recall that in the union operation it doesn't matter which of the two trees is moved under the other. Now what we want to do is minimize the height of the resulting tree. If we are uniting two trees (or sets) lets call them left and right then it all depends on the rank of Left & rank of right.

Ex :-
Given 2 trees : Tree 1

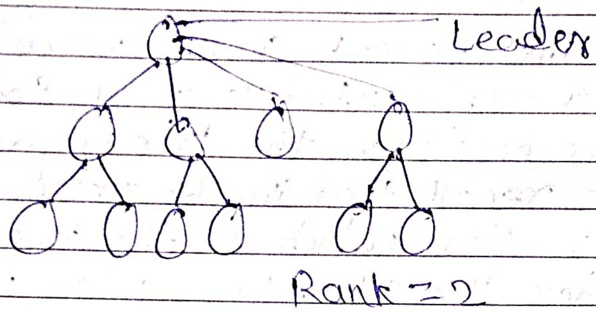


Rank = 2

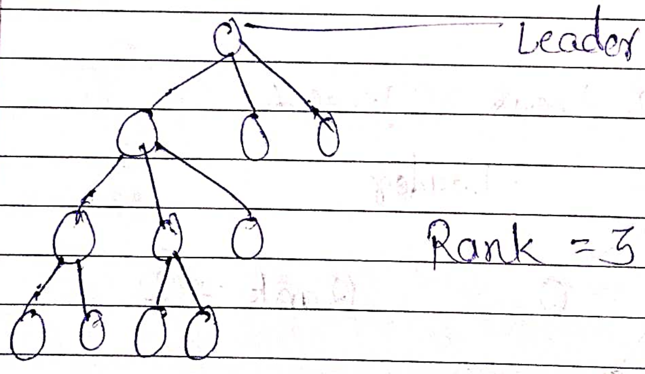
Tree - 2



If we combine their rank then rank [tree 1] = 2 and rank [tree 2] = 1 combined.



We can see that since tree 2 is attached to trees so rank of the combined tree is same as tree 1. If we combine tree 1 & 2 checking their rank then it would be



Page No.
Date

Q.8 Describe briefly lower Bound theory

→ Lower bound theory concept is based upon the calculation of minimum time that is required to execute an algorithm.
It used to calculate minimum number of comparis required to execute an algorithm.
According to lower bound theory for lower bound L(n) of an algorithm.
Once lower bound is calculated then we can compare it with actual complexity of algorithm & if their order are same then we declare it has optimal (best)
The Technique used for lower bound theory

Q.9 Difference Between Structure and Union

Structure	Union
(1) The struct keyword is used to define a structure.	(1) The union keyword is used to define union.
(2) Each variable member occupied a unique memory space.	(2) Variables members share the memory space of the largest size variable.
(3) Changing the value of a member will not affect other variable members	(3) changing the value of one member will also affect other variables members.

(4) Each variable member will be assessed at a time.	(4) Only one variable member will be assessed at a time.
(5) It is used to store different data type values.	(5). It is used for storing one at a time from different data type values.
(6) We can initialize multiple variables of a structure at a time	(6) In union, only the first data member can be initialized.

Q.10. Explain techniques used for lower bound theory

→ (1) Comparisan trees :-

In a comparison sort we use only comparisons between elements to gain order information about an input sequence (a_1; a_2.... a_n)

Given a_i, a_j from (a_1, a_2.... a_n) we perform one of the comparison.

$a_i < a_j$ less than

$a_i < a_j$ less than or equal to

$a_i < a_j$ greater than

$a_i \geq a_j$ greater than or equal to

$a_i = a_j$ equal to

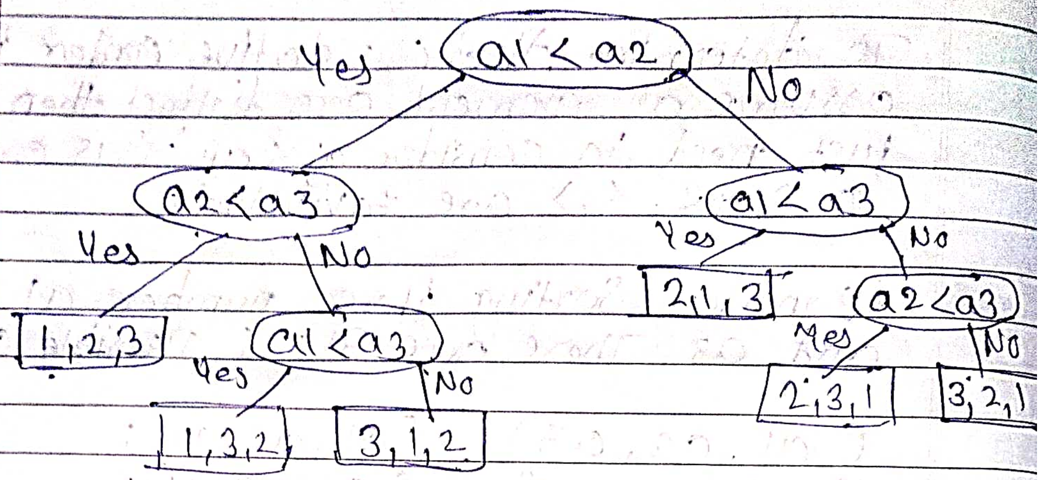
To determine their relative order if we assume all element are distinct then we just need to consider $a_i < a_j$ '=' is excluded ($\geq$, $\leq$, $<$, $>$) are equivalent.

Consider Sorting three numbers a1, a2 and a3. There are 3! = 6 Possible Combination

(a1, a2, a3), (a1, a3, a2);
(a2, a1, a3), (a2, a3, a1);
(a3, a1, a2), (a3, a2, a1)

The Comparison based algorithm defines a decision tree

(2) Decision tree :- it is a fully binary tree that shows the comparisons between elements that are executed by an appropriate sorting algorithm operating on an input of a given size. Control, data movement, and all other condition of the algorithm are ignored. In a decision tree three will an array of length n.



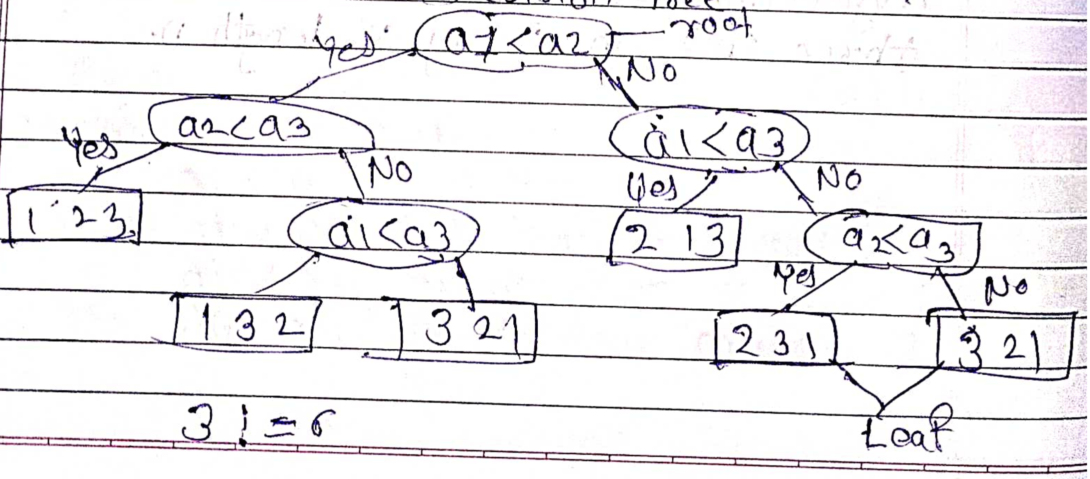
Right Subtree will be False condition [ai > aj]

(1) Comparison trees – are the computational model useful for determining decision tree

(2) Decision tree – (1) full binary tree that shows comparison between elements that are executed by String.

(2) There will be array of length n so total leaves will be n! (total no. of comparison)

Decision tree



3! = 6

Page No.
Date

Left Subtree will be true $a_i < a_j$
right Subtree will be false $(a_i > a_j)$

.

Unit - 3

Assignment - 4

Page No.	
Date	

Q.1. List various String matching algorithm

→ (i) The Navie String matching Algorithm
(ii) The Rabin - Karp - Algorithm
(iii) Finite Automata.
(iv) The Knuth - Morries - Pratt Algorithm
(v) The Beyer - Moore Algorithm

Q.2. Explain in short naive string machine algorith

→ • This is simple and efficient brute force approach.
• It Comperes the first character of pattern with searchable texet
• If a match is found, Paxiters is both strings are advanced.
• If a match is not found, the pointer to texet is uncemented & pointer of the pattern & is reset
• This process is repeated till the end of the texet.

Q.3. Define Pattern matching.

→ • Pattern matching is widely used in computer science and many other fields
• Pattern matching algorithm are used to Search for patterns within a large texet or data set.

.

- Pattern machine algorithms are used to find patterns within a bigger it of data or text.
- Pattern matching algorithm are important because they allow us to search for patterns an large data it quickly.

Q.4. Define string matching with finite automata.

→ • The string matching automata as very useful tool which is used in string matching algorithm.

- It examins early character in the text exactly once reports all the valid shifts an O(n) time

Finite Automata

- Q is a finite set of states
- $q_0 \in Q$ is the start state.
- $A \subset Q$ is a notable set of accepting state
- Σ is a finite input alphabet,
- δ is a function from $Q \times \Sigma$ unto Q called the transition function of M

Q.5. Construct prefix table with given pattern ababaca

→

a	b	a	b	a	c	a
0	0	1	1	2	0	1
	0	0	1	2	0	1

Page No.
Date

(1) a b
a b 0

(2) a b a
a a
ab ba 0

(3) ab ab
a b 0
ab ab 1
aba bab 0

(4) ab aba
a b 1
aba aba 0
abab baba 0

(5) ababa c
a c 0
ab ac 0
aba bac 0
abab abac 0
ababa babac 0

(6) ababaca
a b 1
ab ca 0
aba aca 0
abab baca 0
ababa abaca 0

Page No.
Date

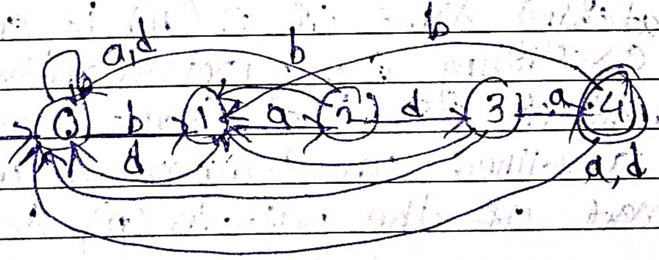
Q6. Write and explain string matching with finite automata with one example.

→ String Matching :-
- The string matching automation as a very useful tool which is used in string matching algorithm
- The goal of string matching as to find the raction of specific texet pattern within the larger body of texet.

Finite Automata :

- A finite Automata :-

(i) A finite automata in as a 5-tuple (Q, q0, A, Σ δ), where
(ii) Q is a finite set of states.
(iii) q0 ∈ Q is the start state.
(iv) A ⊆ Q is a notable set of accepting states.
(v) Σ is a finite input alphabet
(vi) δ is a function from Q × Σ into Q. Called the transition function of M
(vii) Example :- Construction string matching with finiter automata of transition Table (boda)



Page No.
Date

	a	b	d
0	0	1	0
1	2	1	0
2	1	0	3
3	4	1	0
4	0	1	0

(1) b → a bb = 1
b → d bd = 0

(2) ba → a baa = 0
ba → b bab = 1

(3) bad → b badb = 1
bad → d badd = 0

(4) bada → a badaa = 0
bada → b badab = 1
bada → d badad = 0

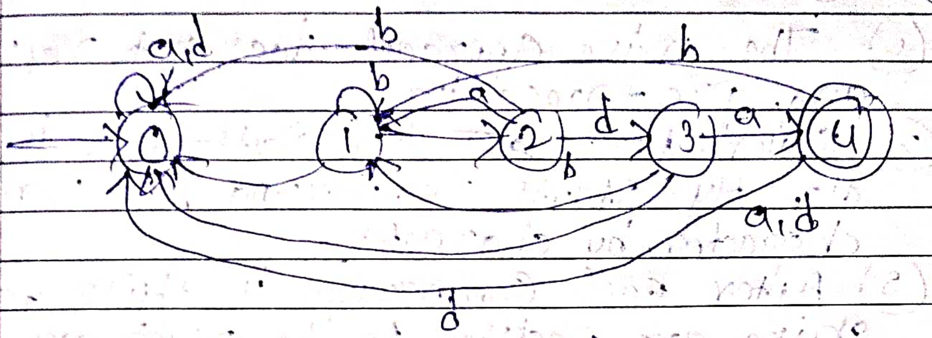
Q.7 Explain and write knuth-morris-pratt algorithm with one example.

→ • Knuth - Morris - Pratt introduce a linear time algorithm for the string ~~linear~~ matching problem.

• A matching time of O(n) is achieved by avoiding comparison with an element of (S) that have previously been involved in comparison with some element of the pattern (P) to be matched

Page No.
Date

(8) Construct prefix table with given pattern (bada) and draw transition diagram.



	a	b	d
0	0	1	0
1	2	1	0
2	1	0	3
3	4	1	0
4	0	1	0

Pefic & construction

(1) b → a bb = 1
b → d bd = 0

(2) ba → a baa = a
ba → b bab = 1

(3) bada → a badaa = a
bada → b badab = 1
bada → d badad = 0

Page No.
Date

Q.9. Explain in brief Naive - string matching algorithm with example.

→ (1) The naive approach does not require any pre-processing.

(2) Given text ~~a~~ T and Pattern P. It ~~shifts~~ directly starts comparsing both strings character by character.

(3) After each comparsion it shifts pattern string one position to the right example. Q.7.

Q.10. Describe various String matching Algorithm.

→ (1) The Naive String Matching Algorithm

- This is simple and efficient brute force approach
- It compares the first character of pattern with searchable text i.e. back tracking on the string (S) never occurs.

example :-

Step I :- T[1] ≠ P[1], & ~~a~~ advance text pointer. i.e. if

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I/S

↕

A	N	D
1	2	3

Step 2 :- T [2] ≠ P [1], so advance ~~text~~ pointer
i.e. i++.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	L	A	N	I	N	G	A	N	D	A	N	A	L	Y	S	I

1	2	3
A	N	D

.

Unit - 4

Assignment - 5

Page No.

Date

Q.1: What is Graph?

→ A graph is a unique data structure in programming that consists of finite set of need or vertices and a set of edge that connect these vertice to them. In simple terms, a graph

2) What are the types of graph.

— undirected graph
Directed graph
weighted graph
cyclic Graph.
Acylic Graph
Connected Graph.
Discconected Graph.
Complete Graph

3) What is BFS.

→ BFS is an alogorithm for traversing or Searching tree or graph data Structure. It Starts at given nodes and explores all at the neighbour nodes at the present depth perior to moving on to nodes at the next depth level.

.

Page No.
Date

4. Define tree edge, forward edge, Cross edge, back edge.

→ Tree : An edge that is part of the tree Structure formed by DFS traversal. It Connects a node to a newly discovered node.

Forward edge : An edge that Connects a nodes to a descendant in the DFS tree but is not a tree edge.

Cross edge :- An edge that Connects nodes ~~or~~ across different branches of the DFS tree

Back Edge :- An edge that connects a node to one of its ancestors in the DFS tree forming a cycle.

Q.5. What is DFS

→ DFS (Depth First Search) is an algorithm for traversing or searching tree or graph data Structures. It starts at a given nodes (root) and explores as far as possible along each branch before backtracking.

Page No.
Date

Q.6 Write algorithm for BFS with one example.

→ The steps involved in the BFS algorithm to explore a graph are given as follows

Step 1 :- SET STATUS = 1 (ready state) for each node in G

Step 2 :- Enqueue the starting node A and set its STATUS = 2 (waiting state)

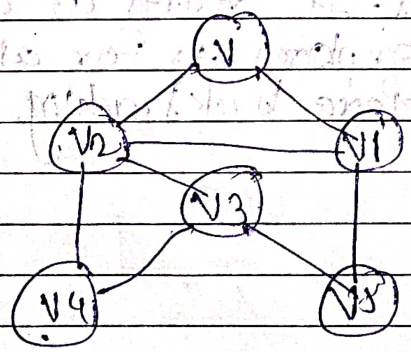
Step 3 :- Repeat step 4 and 5 until Queue is empty

Step 4 :- Dequeue a node N. Process it an

Step 5 :- Enqueue all the neighbours of N that are in the ready

State (where

~~Step~~ Step 6 :- Exit



Page No.
Date

Step 1 :- | V | | | | | |

Step 2 :- | | V2 | V1 | | | |

Step 3 :- | | | V1 | V4 | V3 | |

Step 4 :- | | | | V4 | V3 | V5 |

Step 5 :- | | | | | V3 | V5 | Delete V4

Step 6 :- | | | | | | V5 | Delete V3

Step 7 :- | | | | | | | Delete V5

Q.7. Write algorithm for DFS with one example.

Step 1 :- Set status := 1 (ready state) for each node in G.

Step 2 :- Push the starting node A on the stack & set its status = 2 (waiting state).

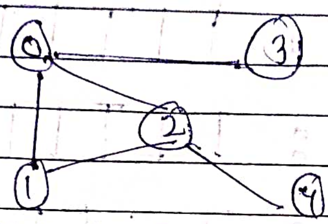
Step 3 :- Repeat step 4 and 5 until stack is empty.

Step 4 :- Pop the top node N & process it and set its status = 3 processed state.

Step 5 :- Push on the Stack all the neighbours of N that are in the ready state. (whose status = 1) and set these status = 2 (waiting state)
[End of loop]

Step 6 :- Exit

Ex :-



Step 1 :- Select Starting Point 0 push

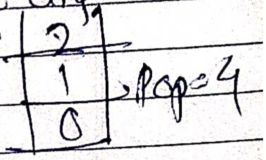
Step 2 :- push adjecent element of 0 unto Stack push 1

Step 3 :- Push 2

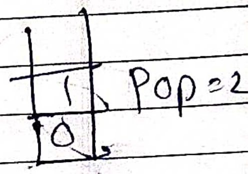
Step 4 :- Push 4 because 0, 1, 2, are

Page No.
Date

Step 5 :- Since 4 does not have any adjecent element well Pop 4 from Stack



Step 6 :- Same as above pop 2



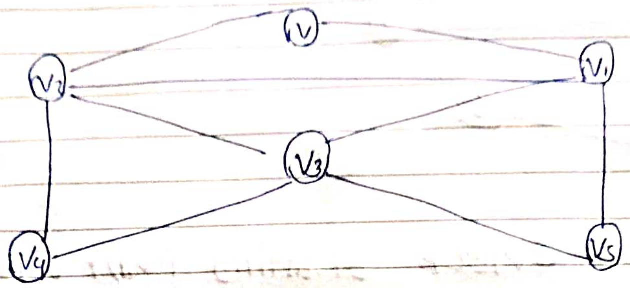
Step 7 :- Pop 1

.

8 Difference between BFS and DFS.

BFS	DFS
• BFS stands for Breadth First search	DFS stands for Depth First search.
• It a vertex-based technique to find the shortest path in a graph.	It is an edge-based technique because the vertices along the edge are explored first from the starting to the en
• BFS is a traversal technique in which all the nodes of the same level are explored first and then we move to the next level.	DFS is also a traversal technique in which tra traversal is started fr the root node and expl the nodes as far as possible until .
• Queue data structure is used for the BFS traversal	Stack data structure is used for the BFS traversal.
• BFS does not use the backtracking concept	DFS uses backtracking traverse all the unvisi nodes
• BFS is slower than DFS	DFS is faster than B

9 Determine BFS.



Step 1 :

V					

Step 2 :

	V2	V1				

Step 3 :

		V1	V4	V3		

Step 4 :

				V4	V3	V5

Step 5 :

					V3	V5

There is no element present adjacent to V4
Delete V4

Step 6 :

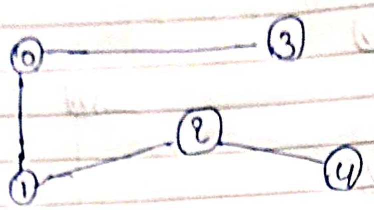
					V5

Delete V3

Step 7 :

Delete V5

10 Determine DFS



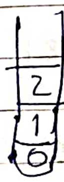
Step 1 : Select starting Point 0 push 0

Step 2 : push adjacent element of 0 unto
Stack push 1

Step 3 : push 2

Step 4 : Push 4 because 0, 1, 2, are

Step 5 : Since 4 does not have any
adjacent element so we will pop
4 from stack



Pop 4

Step 6 : Same as above Pop 2

Pop = 2

Step 7 :- Pop 1