

Vidarbha Youth Welfare Society's
Prof. Ram Meghe Institute of Technology & Research
Badnera, Amravati (M.S.) 444701



Practical Record
Semester VI
(Subject code:- 6IT07)

Subject: - Design and Analysis of Algorithm Lab

(Academic Year: - 2021 - 2022)

Name of Student:- _____

Roll No: - _____ **Section:-** _____

Department of Information Technology

Phone: (0721) - 2580402/2681246

Fax No. 0721 – 2681337

Website : www.mitra.ac.in

Sant Gadge Baba Amaravti University

Vidarbha Youth Welfare Society's
Prof. Ram Meghe Institute of Technology & Research
Badnera, Amravati (M.S.) 444701



Practical Record
Semester VI
(Subject code: - 6IT07)
Subject: - Design and Analysis of Algorithm Lab
Sant Gadge Baba Amravati University Amravati

Department of Information Technology

Phone: (0721) - 2580402/2681246
Fax No. 0721 – 2681337
Website : www.mitra.ac.in

Vidarbha Youth Welfare Society's
Prof. Ram Meghe Institute of Technology & Research
Badnera, Amravati (M.S.) 444701



CERTIFICATE

This is to certify that Mr. /Miss _____
Enrollment No. _____ Roll No _____ Section _____
of B.E. (IT.) SEM-VI has satisfactorily completed the term work of the subject **6IT07**
Design and Analysis of Algorithm Lab prescribed by Sant Gadge Baba University
Amravati during the academic term 2021-22

Date:

Signature of the faculty
Department of Information Technology

Index

Sr. No	Title	Page No.
1	Vision and Mission of the Institute and Program	i
2	Program educational objective and program outcomes	ii
3	Syllabus(Theory+ Practical) Course outcomes, Mapping with POs	iii
4	Assessment Format i.e ACIPV(Guidelines)	v
5	Aim of the Lab Manual,	v
6	List of experiment	vi
7	a .Title/Aim	
	b .M/C specification/Software used	
	c. Theory	
	d. Procedure/Steps	
	e. Program Code	
	f. Output/Result	
	g. Conclusion.	
	h. Quiz/Objective type question/Viva-voce	

➤ **1.Mission & Vision statement of the Institute:**

VISION

To become a pace-setting
Centre of excellence believing in three
Universal values namely
Synergy, Trust and Passion,
With zeal to serve the Nation
In the global scenario

MISSION

To dedicate ourselves
to the highest standard of technical education
& research in core & emerging engineering
disciplines and strive for the overall personality
development of students so as to nurture
not only quintessential technocrats
but also responsible citizens

➤ **Mission & Vision statement of the Department of Information Technology:**

VISION

Cater to global need in comprehensive manner
by applying judicious mix of technology
Comprising of hardware and software for saving time,
material resources while imbibing ethical values.

MISSION

To become leading education center
by inspiring the students to become strategic technologist
in all walks of life by making them innovative and research oriented,
to raise their ability to provide appropriate solutions to global needs professionally

2. Program Education Objectives (PEOs)

- PEO1. Preparation:** To prepare students become generalist engineers for successful career in IT industry, to excel in higher studies & research and to promote entrepreneurial thinking among students.
- PEO2. Core Competence:** To provide student graduates with solid foundation in mathematical, scientific, computing, core information technologies required to develop problem solving ability.
- PEO3. Breadth:** To analyze, design and develop, efficient and cost effective IT solutions using multidisciplinary approaches for the benefit of society.
- PEO4. Professionalism:** To inculcate the value systems, leadership and team work, good communication skills to bring holistic development of personality.
- PEO5. Learning Environment:** To provide students with excellent academic environment in order to imbibe self-learning, lifelong learning, and innovation for successful professional career.

➤ Program Outcomes (PO's)

On completion of the course a graduate of Information Technology program will be able to

- PO1.** Acquire and apply the knowledge of mathematics, science and engineering, Information Technology in solving complex problems.
- PO2.** Identify, formulate, review literature and analyze engineering problems to reach substantiated conclusions.
- PO3.** Design a system or process to meet the desired needs with appropriate consideration for economic, public health and safety, social, cultural and environmental issues.
- PO4.** Investigation of complex problems through literature review, indulge in research and methods to design new experiments, analyze, and interpret data to draw valid conclusions.
- PO5.** Use suitable IT techniques, skills and modern tools necessary for computing practices as an IT professional with an awareness of limitations.
- PO6.** Apply contextual knowledge to address societal, legal, cultural, health and safety issues applicable to IT professional practices.
- PO7.** Comprehend the impact of IT solutions on society and environment for sustainable development.
- PO8.** Understand and adapt professional and ethical responsibilities at work place.
- PO9.** Function effectively as an individual, as a member or leader in diverse teams and in a multidisciplinary environment.
- PO10.** Communicate effectively about engineering problems and solutions with engineering community & society at large in both verbal & written form.
- PO11.** Apply the knowledge of engineering, finance and management principles to manage projects in multidisciplinary environments.
- PO12.** Engage in lifelong learning of IT technologies to cope up with the rapid changes in technology.

➤ Program specific program outcome (PSOs)

- PO13.** Apply core aspects of Information Technology, Networking, Internet of Things and Security to solve real world problems for betterment of society.
- PO14.** Develop analyze and find IT solutions through programming paradigm, Android Application Development, Web Designing and Cloud Computing.

3. Design and Analysis of Algorithm (6IT07)

Course Number and Title	:	Design and Analysis of Algorithm (6IT07)
Instructor In-charge	:	Dr. A. S. Alvi Prof. M. S. Deshmukh
Course Type	:	Lab
Compulsory/ Elective	:	Compulsory
Teaching Methods	:	Lecture : 4 Hrs/ week. Laboratory: 8 Hrs / week Discussion : Office hour discussion, Debate, Quiz.
Course Assessment	:	Exams : 2 Class Tests Semester end examination by SGBAU
Grading Policy	:	20 % homework, class test and viva-voce 80% Semester end examinations 25 Internal Marks+25 External Exam and Viva

Course Learning Objectives:

1. To teach paradigms and approaches used to analyze and design algorithms and to appreciate the impact of algorithm design in practice.
2. To make students understand how the worst-case time complexity of an algorithm is defined, how asymptotic notation is used to provide a rough classification of algorithms.
3. To explain different computational models (e.g., divide-and- conquer), order notation and various complexity measures.
4. Study of various advanced design and analysis techniques such as greedy algorithms, dynamic programming
5. Synthesize efficient algorithms in Common Engineering situations.

Upon completion of this course, students will able to:

Sr. No	Specific Course Outcomes
CON 607.1	Analyze worst-case running times of algorithms using asymptotic analysis.
CON 607.2	Describe the divide-and-conquer paradigm and explain when an algorithmic design situation calls for it.
CON 607.3	Describe the dynamic-programming paradigm and explain when an algorithmic design situation calls for it.
CON 607.4	Describe the greedy paradigm and explain when an algorithmic design situation calls for it.
CON 607.5	Able to understand the concept of Backtracking, Polynomial Time & Non Polynomial Time Algorithms.

4. Assessment Sheet

Name of Experiment							
Roll no.	Student Name	A (5marks)	C (5marks)	I (5marks)	P (5marks)	V (5marks)	Total (25Marks)

(A-Attendance, C- Competency, I-Innovation, P-Presentation, V-Viva)

Guidelines for Awarding Internal Marks for Practical:

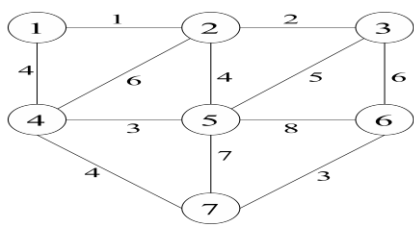
Each experiment/ practical carries 25 marks. The student shall be evaluated for 25 marks as per the following guidelines. At the end of the semester, internal assessment marks for practical shall be the average of marks scored in all the experiments.

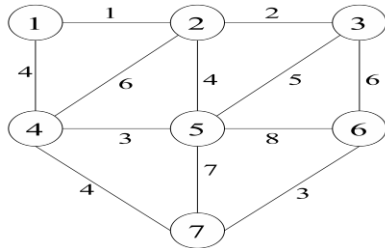
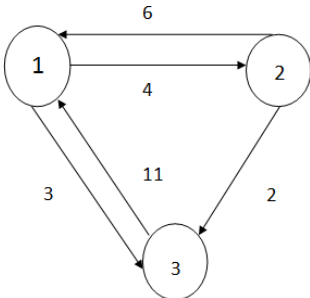
- a. **Attendance (5 marks):** These 5 marks will be given to the regularity of a student. If the student is present on the scheduled time, he/ she will be awarded 5 marks. Otherwise will be given 2.5 marks for his attendance.
- b. **Competency (5 marks):** Here the basic aim is to check whether the student has developed the skill to write down the code on his own and debug. If a student executes the practical on the scheduled day within the allocated time, he will be awarded full marks. Otherwise, marks will be given according to the level of completion/ execution of practical
- c. **Innovation (5 marks):** Here the basic objective is to explore the innovative ideas from the students with respect to the corresponding practical or how innovatively he interprets the aim of the practical. It is expected that the students must be aware about the scope of practical precisely such that in future, they could implement the task in various applications.
- d. **Performance/Participation in-group activity (5 marks):** These marks will be given on how a student is actively participating the group. If the student is performing the practical as the part of a group he must participate actively.
- e. **Viva-Voce (5 marks):** These 5 marks will be totally based on the performance in viva-voce. There shall be viva-voce on the completion of each practical in the same practical session. The student shall get the record checked before next practical.

5. Aim of the Lab manual:

This manual is designed for the final year (6th Semester) students of Information Technology for the Design and Analysis of Algorithm Lab. The main aim of this Manual is to provide guidelines to students for proper executions of lab sessions. It has been prepared as per university curriculum & covers various aspects of DAA.

6. List of Experiment**Department of Information Technology****Subject: - Design and Analysis of Algorithm.****Semester: - VII****University Code: - 6IT07hj**

Sr. No.	List of Practical	Page. No.	Date	Remark
1	To study various algorithm designing strategies.			
2	Implement C programs to perform recursive calls using following searching algorithms. a. Linear Search when the list is given. b. Binary Search when the given list is not sorted.			
3	Study and Implement Divide and Conquer strategy using Merge sort Algorithm and determine the complexity of algorithm. DATA- {23,12,3,5,89,1,24}			
4	Write a C program for implementing (n X n) matrix multiplication using Strassen's matrix multiplication algorithm.			
5	Explain the knapsack algorithm to find an optimal solution of getting maximum profit and implement using C program.			
6	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm and implement using C. 			

7	<p>Implement Prim's algorithm to find Minimum Cost Spanning Tree of a undirected graph using C program.</p> 			
8	<p>Develop a C program to implement Floyd's algorithm which will produce shortest distance between all vertex pairs of a weighted graph.</p> 			
9	<p>Study an algorithm Tower of Hanoi where the aim is to move the entire stack to another rod for n=3 and understand the concept of recursion.</p>			
10	<p>Print all the nodes reachable from a given starting node in a digraph using Breadth First Search.</p>			
11	<p>Print all the nodes reachable from a given starting node in a digraph using Depth First Search.</p>			

Practical No. 01

Aim: To study various algorithm designing strategies.

Theory:

The earlier example might have convinced you that the strategy adopted by an algorithm is important in deciding its efficiency. There are a number of classical strategies which are well studied and used extensively.

Some of the well-known ones are:

1. Divide-and-conquer
2. Brute-force
3. Dynamic programming
4. Greedy algorithms
5. Exotic methods like simulated annealing, genetic algorithms, artificial neural networks, and so on, which are used in special and complex cases.

❖ Divide and Conquer

Divide and conquer algorithm suggests splitting the inputs into distinct subsets. These sub problems must be solved and then a method must be found to combine sub solutions into a solution of the whole. If the sub problems are still relatively large, then the divide and conquer strategy can be possibly be reapplied. Often the sub problems resulting from a divide and conquer design are of the same type as the original problem. For those cases the reapplication of the divide and conquer principle is naturally expressed by a recursive algorithm. This algorithm technique is the basis of efficient algorithms for all kinds of the problems, such as quick sort, merge sort and discrete Fourier transform. Its application to numerical algorithms is commonly known as binary splitting.

Divide-and-conquer algorithm works as follows:

- Divide and conquer algorithm are divided into several smaller instances of the same problem and same size.
- The smaller instances are solved by recursively. Sometimes, a different algorithm is applied when instances become small enough.
- The smaller instances are combined and to get a solution to the original problem.
 - o No necessary to combine in some cases.

- Divide-and-conquer technique is ideally suited for parallel computers, in which each sub problem can be solved simultaneously by its own processors.
- Common case: Dividing a problem into two smaller problems

❖ **Brute-force**

Definition: An algorithm that inefficiently solves a problem, often by trying every one of a wide range of possible solutions.

A brute force algorithm simply tries all possibilities until a satisfactory solution is found such an algorithm can be:

- Optimizing: Find the best solution. This may require finding all solutions, or if a value for the best solution is known, it may stop when any best solution is found
- Satisfying: Stop as soon as a solution is found that is good enough Brute force algorithm is require exponential time and used in various heuristics and optimizations can be used Heuristic: A rule of thumb that helps you decide which possibilities to look at first. Optimization: In this case, to eliminate certain possibilities without fully exploring them.

❖ **Greedy Algorithm**

The greedy algorithm is perhaps the most straightforward design technique. It can be applied to a wide variety of problem. Most though not all of these problems have n inputs and require us to obtain a subset that satisfies some constraints. Any need to find a feasible solution that either maximizes or minimizes a given objective function. A feasible solution that does this is called an optimal solution. Note: greedy algorithm avoid backtracking and exponential time $O(2^n)$ Greedy algorithms work in phases. In each phase, a decision is made that appears to be good, without regard for future consequences E.g. Kruskal's MST algorithm, Dijkstra's algorithm.

Greedy algorithm work in phases. In each phase, a decision is made that appears to be good, without regard for future consequences. Generally, this means that some local optimum is chosen. Greedy algorithm to find minimum spanning tree. Want to find set of edges.

Type of Greedy Algorithm:

There are three type of greedy algorithms

- Pure Greedy Algorithms
- Orthogonal Greedy Algorithms
- Relaxed Greedy Algorithms

❖ Dynamic programming

Dynamic programming is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions. Dynamic programming is a similar to divide and conquer algorithm. It is express solution of a problem in terms of solutions to sub problems. The Key difference is between dynamic programming and divide and conquer is that while sub problems in divide and conquer are independent, sub problems in dynamic programming may themselves share sub problems. This means that if these were treated as independent sub problems, the complexity would be higher. Dynamic programming is typically used to solve optimization problems. In bioinformatics, the most common use of dynamic programming is in sequence matching and alignment.

- To begin, the word programming is used by mathematicians to describe a set of rules, which must be followed to solve a problem.
- Thus, linear programming describes sets of rules which must be solved a linear problem.
- In our context, the adjective dynamic describes how the set of rules works.
- In this course, a number of examples of recursive algorithms are seen.
- The run time of these algorithms may be found by solving the recurrence relation itself.

❖ Backtracking Algorithm:

Backtracking algorithm represents one of the most general techniques. Many problems which deal with searching for a set of solutions or which ask for an optimal solution satisfying some constraints can be solved using the backtracking formulation. Many of the problems being solved using backtracking require that all the solutions satisfy a complex set of constraints. For any problem these constraints can be divided into two categories explicit and implicit.

- View the problem as a sequence of decisions
- Systematically considers all possible outcomes for each decision
- Backtracking algorithms are like the brute-force algorithms
- However, they are distinguished by the way in which the space of possible solutions is explored
- Sometimes a backtracking algorithm can detect that an exhaustive search is not needed

❖ **Branch and Bound Algorithm**

The general idea of B&B is a BFS-like search for the optimal solution, but not all nodes get expanded (i.e., their children generated). Rather, a carefully selected criterion determines which node to expand and when, and another criterion tells the algorithm when an optimal solution has been found.

Branch and Bound (B&B) is the most widely used tool for solving large scale NP-hard combinatorial optimization problems. The following table-1 summarizes these techniques with some common problems that follows these techniques with their running time

Conclusion:

Viva Questions:

1. What are strategies of algorithm design?
2. What are the 4 types of algorithm?
3. What are key features of an algorithm?
4. What do you mean by algorithmic strategy?
5. What are the 3 algorithm analysis techniques?
6. What are the simple strategies for developing algorithms?
7. What is the role of algorithm in problem solving?
8. What are the building blocks of algorithm?
9. Where are algorithms used in real life?
10. What are five things algorithms must have?

Signature
Department of Information
Technology

Practical No. 02

Aim: Implement C programs to perform recursive calls using following searching algorithms.

- a. Linear Search when the list is given.
- b. Binary Search when the given list is not sorted.

Software Requirement: - Ubuntu, gcc compiler.

Theory:

a. Linear Search: Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection. The complexity of linear search algorithm, Worst-case performance is $O(n)$, Best-case performance is $O(1)$ and Average performance is $O(n)$.

Algorithm:

Procedure linear_search (list, value)

```
for each item in the list if match item == value
return the item's location end if
end for
end procedure
```

Program:

Output:-

b. Binary search: Binary search is most important and easiest technique of searching. In binary search, if the array is present in sorting order the first find out mid by using low and high. If the search element is greater than the mid then search element is present in second half and if element is smaller than mid then it is present in first half. Otherwise the element is not present. The list is not in sorted order then first applies the logic of sorting for sort the list. Binary search has a best case efficiency is $O(1)$ and worst case & average case efficiency is $O(\log n)$.

Algorithm:

procedure binarysearch(low, high, mid)

```
low=0, high=n-1;
while(low<=high)
mid=(low+high)/2; if(s>a[mid]) then low=mid+1;
else if(s<a[mid]) high=mid-1;
else if(s==a[mid])
```

Program:

Output:-

Conclusion: -

Viva Questions:

1. How many types of searching & which are they?
2. Explain binary search algorithm.
3. Compare binary search algorithm with linear search algorithm and decide which one is better.
4. There are any necessary conditions for binary search algorithm means given array is in sorted order or not?
5. What is the complexity of linear search algorithm?
6. What is the complexity of binary search algorithm?

Signature
Department of Information
Technology

Practical No: 03

Aim: Study and Implement Divide and Conquer strategy using Merge sort algorithm and determine the complexity of algorithm.

Software Requirement: - Ubuntu, gcc compiler.

Theory:

Merge Sort is based on a simple "Divide and conquer" approach. The principle behind it is simple: We want to sort an entire list, but if this entire list is sorted, then each half of this list must also be sorted. It is an easier problem to sort a smaller list than a larger list, so we sort each of the smaller lists. Once these two lists are sorted, we run a merge algorithm which combines the two smaller lists into one larger list. These smaller lists are then sorted using the same principle until our smaller lists are of size 1 or 2, in which case we can simply swap the two items or return the item itself. The different complexity of merge sort, Worst-case performance, Best-case performance and Average performance is **$O(n \log n)$** .

Algorithm:

```
function merge_sort(m)
if length(m)  $\leq$  1
return m
var list left, right, result
var integer middle = length(m) / 2
for each x in m up to middle add x to left
for each x in m after middle add x to right
left = merge_sort(left)
right = merge_sort(right)
result = merge(left, right)
return result
```


Program:

Output:

Conclusion: -

Viva question:

1. What are desirable conditions for merge sort?
2. How you compare quicksort and merge sort?
3. Can Mergesort be implemented recursively?
4. How you analyze the performance of merge sort complexity?
5. What is the complexity of merge sort algorithm?

Signature
Department of Information
Technology

Practical No: 04

Aim: Write a C program for implementing (n X n) matrix multiplication using Strassen matrix multiplication algorithms.

Software Requirement: - Ubuntu, gcc compiler.

Theory:

Strassen Matrix Multiplication.

To further illustrate the concept of D&C let us take an example from classical mathematics—Strassen Matrix Multiplication method. Matrix multiplication is a fundamental problem and it arises in almost all branches of science and engineering. For example, high-energy physicists multiply very large matrices.

Let X , Y , and Z be $n \times n$ matrices, where $Z = X \cdot Y$. There are n rows, n columns, and n^2 entries in each of the matrices. A fundamental numerical operation is the multiplication of 2 matrices.

The standard method of matrix multiplication of two $n \times n$ matrices takes, $T(n) = O(n^3)$.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

The following algorithm multiplies $n \times n$ matrices A and B :

```
// Initialize C.
for i = 1 to n
  for j = 1 to n
    for k = 1 to n
      C[i, j] = A[i, k] * B[k, j];
```

We can use a Divide and Conquer solution to solve matrix multiplication by separating a matrix into 4 quadrants:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

- Then we know have:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \quad C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

- If $C=AB$, then we have the following:

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

Strassen's algorithm showed how two matrices can be multiplied using only 7 multiplications and 18 additions:

Consider calculating the following 7 products:

$$\begin{aligned} q1 &= (a_{11} + a_{22}) * (b_{11} + b_{22}) \\ q2 &= (a_{21} + a_{22}) * b_{11} \\ q3 &= a_{11} * (b_{12} - b_{22}) \\ q4 &= a_{22} * (b_{21} - b_{11}) \\ q5 &= (a_{11} + a_{12}) * b_{22} \\ q6 &= (a_{21} - a_{11}) * (b_{11} + b_{12}) \\ q7 &= (a_{12} - a_{22}) * (b_{21} + b_{22}) \end{aligned}$$

It turns out that

$$\begin{aligned} c_{11} &= q1 + q4 - q5 + q7 \\ c_{12} &= q3 + q5 \\ c_{21} &= q2 + q4 \\ c_{22} &= q1 + q3 - q2 + q6 \end{aligned}$$

Algorithm:-

```

for i ← 1 to p do
    for j ← 1 to r do
        C[i,j] ← 0
        for k ← 1 to q do
            C[i, j] ← C[i, j] + A[i, k] * B[k, j]

```

Program:-

Output:-

Conclusion: -

Standard Viva Questions-

- 1) Explain matrix multiplication algorithm.
- 2) What is the complexity of 2×2 simple matrixes?
- 3) Explain Strassen's matrix multiplication.
- 4) Distinguish simple matrix multiplication and Strassen's matrix multiplication.

Signature
Department of Information
Technology

Practical No: 05

Aim: - Explain the knapsack algorithm to find an optimal solution of getting maximum profit and implement using C program.

Software Requirement: - Ubuntu, gcc compiler.

Theory:-

Given a set of items, each with a weight and a value, determine which items you should pick to maximize the value while keeping the overall weight smaller than the limit of your knapsack. A simple solution is to consider all subsets of items and calculate the total weight and value of all subsets. Consider the only subsets whose total weight is smaller than W . From all such subsets, pick the maximum value subset. The knapsack problem arises whenever there is resource allocation with financial constraints. Given a fixed budget, how do you select what things you should buy? Everything has a cost and value, so we seek the most value for a given cost. The term *knapsack problem* invokes the image of the backpacker who is constrained by a fixed-size knapsack and so must fill it only with the most useful items. The complexity of knapsack algorithm is $O(n)$.

Algorithm:-

Let i be the highest-numbered item in an optimal solution S for W pounds. Then $S' = S - \{i\}$ is an optimal solution for $W - w_i$ pounds and the value to the solution S is V_i plus the value of the subproblem.

We can express this fact in the following formula: define $c[i, w]$ to be the solution for items $1, 2, \dots, i$ and maximum weight w . Then

$$c[i, w] = \begin{array}{ll} 0 & \text{if } i = 0 \text{ or } w = 0 \\ c[i-1, w] & \text{if } w_i \geq 0 \\ \max [v_i + c[i-1, w-w_i], c[i-1, w]] & \text{if } i > 0 \text{ and } w \geq w_i \end{array}$$

Program:-

Output:-

Conclusion:-

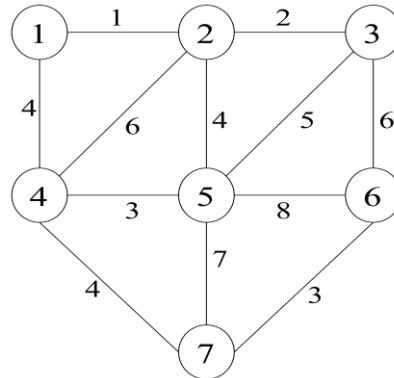
Standard Viva Questions-

1. What is knapsack algorithm?
2. What is the use of knapsack problem?
3. What are different methods of selecting objects?
4. Which method gives more efficient output?
5. Why any algorithm called as efficient algorithm?
6. What is the complexity of Knapsack algorithm?

Signature
Department of Information
Technology

Practical No: 06

Aim: - Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm and implement using C.



Software Requirement: - Ubuntu, gcc compiler.

Theory:-

The set T of edges is initially empty. As the algorithm progress, edges are added to T . So as long as a solution is not found, the partial graph formed by the nodes of G and the edges in T consist of several connected components. The elements of T included in a given connected component form a minimum spanning tree for nodes in this component. At the end of the algorithm only one connected component remains, so T is then a minimum spanning tree for all the nodes of G . The complexity is $O(E \log V)$.

Algorithm:-

```

function Kruskal( $G = (N, A)$ : graph; length:  $A - R^+$ ): set of edges
{initialization}
Sort  $A$  by increasing length
 $n \leftarrow$  the number of nodes in  $N$ 
 $T \leftarrow \emptyset$  {will contain the edges of the minimum spanning tree}
initialize  $n$  sets, each containing a different element of  $N$ 
{greedy loop}
  repeat
     $e \leftarrow \{u, v\} \leftarrow$  shortest edge not yet considered
     $ucomp \leftarrow find(u)$ 
     $vcomp \leftarrow find(v)$ 
    if  $ucomp \neq vcomp$  then
      merge( $ucomp, vcomp$ )
       $T \leftarrow T \cup \{e\}$ 
    until  $T$  contains  $n - 1$  edges
  return  $T$ 
  
```

Program:-

Output:-

Conclusion: -

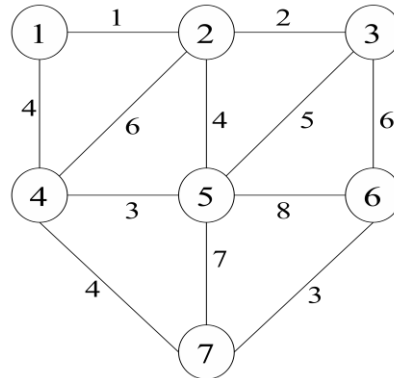
Standard Viva Questions-

1. What is the purpose of Kruskal's algorithms?
2. Explain the Kruskal's algorithm.
3. How the Kruskal's algorithm defines greedy method?
4. Distinguish Kruskal's and prim's algorithm.
5. If the direct path from one node to another node not exists in the graph that time what's the values from first node to that second node.
6. What is the complexity of Krushkal's algorithm?

Signature
Department of Information
Technology

Practical No: 07

Aim: - Implement Prim's algorithm to find Minimum Cost Spanning Tree of an undirected graph using C program.



Software Requirement: - Ubuntu, gcc compiler.

Theory:-

In kruskal's algorithm, the selection function chooses edges in increasing order of length without worrying too much about their connection to previously chosen edges, except that we are careful never to form a cycle. The result is to forest of trees that grows somewhat haphazardly, until finally all the components of the forest merge into a single tree. In prim's algorithm, on the other hand, the minimum spanning tree grows in a natural way, starting from an arbitrary root. At each stage we add a new branch to the tree already constructed; the algorithm stops when all the nodes have been reached. The complexity is $O(E + \log V)$.

Algorithm:-

```

function Prim (G= < N,A >:graph; length: A  $\rightarrow$   $R^+$  ) ;
set of edges {initialiazation}
  T  $\leftarrow \emptyset$ 
  B  $\leftarrow$  { an arbitrary member of N}
  while B  $\neq$  N do
    find e={u,v} of minimum length such that
      u  $\in$  B and v  $\in$  N\B
    T  $\leftarrow$  T  $\cup$  {e}
    B  $\leftarrow$  B  $\cup$  {v}
  return T
  
```

Program:-

Output:-

Conclusion: -

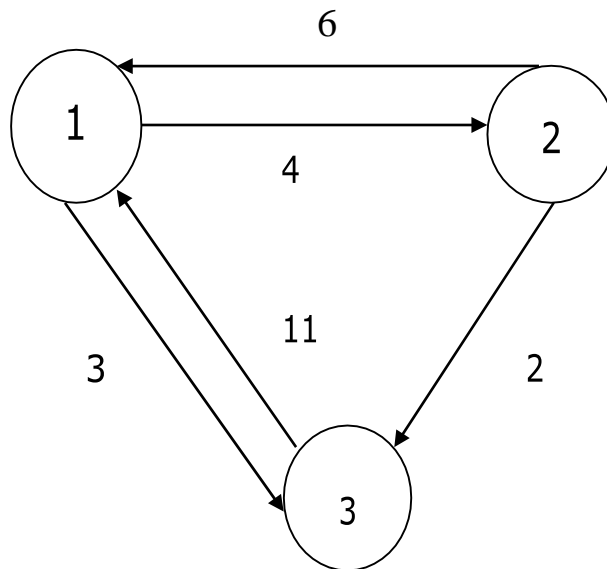
Standard Viva Questions-

1. What is the purpose of prim's algorithms?
2. Explain the prim's algorithm.
3. Distinguish prims and kruskal's algorithm.
4. If the direct path from one node to another node not exists in the graph that time what's the values from first node to that second node.
5. What is the complexity of Prim's algorithm?

Signature
Department of Information
Technology

Practical No: 08

Aim: Develop a C program to implement Floyd's algorithm which will produce shortest distance between all vertex pairs of a weighted graph.



Software Required: Ubuntu, gcc Compiler

Theory:

Floyd's algorithm is one of the shortest path algorithms in graph theory. A weighted graph is taken as input in the form of weight-matrix. Weight-matrix is a vertex by vertex matrix where, $w(i,j)$ = weight between vertices v_i and v_j . If there is no edge between v_i and v_j then $w(i,j)$ = Infinity (a very large number i.e 9999 used as input for giving infinity in the actual program). Diagonal elements are always zero for a simple graph. The output of the algorithm is also a matrix of same order as the input, where we can find shortest distance between all vertex pairs. Three for loops are used in the program and total number of iterations are same as the number of vertices of the graph. For example - if number of vertices are 5 then we get the output matrix after 5th iteration. At iteration k , the algorithm must check for each pair of node (i,j) whether or not there exists a path from i to j passing through the node k that is greater than the present optimal path. The complexity of Floyd's algorithm is $O(n^3)$.

Algorithm:

```
Algo_floyd(w[size][size], n)
/*The input graph and no. of vertices n passed as parameters*/
{
  for(k=1 to n) /*k represents table no.*/
  {
    for(i=1 to n) /*i represents row no. within a table*/
    {
      for(j=1 to n) /*j represents column no. within a row*/
      {
        If( $w[i][j] > (w[i][k] + w[k][j])$ )
        /*Minimum is to be selected*/
        /*w[i][j] denotes distance between ith vertex vi and jth vertex vj*/
         $w[i][j] = (w[i][k] + w[k][j]);$ 
      }
    }
  }
}
/*End of Algo_floyd*/
```

Program:-

Output:

Conclusion:

Standard Viva Questions-

1. What is the purpose of Floyd algorithm?
2. Explain the Floyd algorithm with example?
3. Which strategy is used for Floyd algorithm?
4. What is the time complexity of algorithm?

Signature

Department of Information
Technology

Practical No. 09

Aim: Study an algorithm Tower of Hanoi where the aim is to move the entire stack to another rod for $n=3$ and understand the concept of recursion.

Software Requirement: - Ubuntu, gcc compiler.

Theory:-

Tower of Hanoi is an example of recursion that can be used as tool in developing algorithm to solve a particular problem. Suppose three pegs labeled A, B and C are given and suppose on peg A there are placed a finite number n of disks of decreasing size. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary. The rules are as follows:

- a. Only one disk may be moved at a time. Specifically only the top disks on any peg may be moved to any other peg.
- b. At no time can a larger disk be placed on a smaller disk.

The complexity of Tower of Hanoi problem is $O(2^n)$.

Algorithm:

```
START
Procedure Hanoi(disk, source, dest, aux)
  IF disk == 1, THEN
    move disk from source to dest ELSE
    Hanoi(disk - 1, source, aux, dest) // Step 1 move disk from source to dest //
    Step 2 Hanoi(disk - 1, aux, dest, source) // Step 3
  END IF
END Procedure STOP
```

Program:

Output:

Conclusion:

Viva Questions:

1. Explain recursion with example.
2. How tower of Hanoi problem is implemented?
3. What are the conditions to be followed for implementing tower of Hanoi problem?
4. What is the complexity of Tower of Hanoi algorithm?

Signature
Department of Information
Technology

Practical No. 10

Aim: - Print all the nodes reachable from a given starting node in a digraph using Breadth First Search.

Software Requirement: - Ubuntu, gcc compiler.

Theory:-

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbor nodes first, before moving to the next level neighbors. Breadth First Search algorithm(BFS) traverses a graph in a breadth wards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration. BFS has the same efficiency as DFS: it is $\Theta(V^2)$ for Adjacency matrix representation and $\Theta(V+E)$ for Adjacency linked list representation.

Algorithm:-

```
local queue Q;
for each  $v$  in  $V$  do
 $d[v] \leftarrow \infty$ ;
end
 $d[s] \leftarrow 0$ ;
ENQUEUE( $Q, s$ );
while not empty( $Q$ ) do
 $v \leftarrow$  DEQUEUE( $Q$ );
for each  $u$  in ADJACENT[ $v$ ] do
if  $d[u] = \infty$  then
 $d[u] \leftarrow d[v] + 1$ ;
ENQUEUE( $Q, u$ );
end
end
end
```

Program:-

Output:-

Conclusion: -

Standard Viva Questions-

1. Explain traversals of graph.
2. Explain breadth First Search algorithm.
3. What is mean by backtracking?
4. How the backtracking is applicable on breadth first search?

Signature
Department of Information
Technology

Practical No. 11

Aim: - Print all the nodes reachable from a given starting node in a digraph using Depth First Search.

Software Requirement: - Ubuntu, gcc compiler.

Theory:-

Let $G \langle N, A \rangle$ be an undirected graph all of whose nodes we wish to visit. Depth first traversal of the graph, choose any node $v \in N$ as the starting point. Mark this node to show it has been visited. Next, if there is a node adjacent to v that has not yet been visited, choose this node as a new starting point and call the depth first search procedure recursively. On return from the recursive call, if there is another node adjacent to v that has been visited, choose this node as the next starting point, and call the procedure recursively once again, and so on. When the entire nodes adjacent to v are marked, the search starting at v is finished. If there remain any nodes of G that have not been visited, choose any one of them as a new starting point, and call the procedure yet again. Continue thus until all the nodes of G are marked.

DFS is typically used to traverse an entire graph, and takes time $\Theta(|V| + |E|)$. For the adjacency matrix representation, the traversal time efficiency is in $\Theta(|V|^2)$ and for the adjacency linked list representation, it is in $\Theta(|V| + |E|)$, where $|V|$ and $|E|$ are the number of graph's vertices and edges respectively.

Algorithm:-

procedure dfsearch(G)

for each $v \in N$ do mark[v] \leftarrow not visited
for each $v \in N$ do
if mark[v] \neq visited then dfs(v)

procedure dfs(v)

```
{
node v has not previously been visited
}
mark[v]  $\leftarrow$  visited
for each node w adjacent to v do if mark[w]  $\neq$  visited
then dfs(w)
```

Program:-

Output:-

Conclusion: -

Standard Viva Questions-

1. Explain traversals of graph.
2. Explain Depth First Search algorithm.
3. What is mean by backtracking?
4. How the backtracking is applicable on depth first search?

Signature
Department of Information
Technology

