SECTION 1 DESIGN AND ANALYSIS OF ALGORITHMS LAB

Session 1: Implementation of Simple Algorithms

Introduction

The focus of the section is to implement small problems such as Euclid's algorithm for GCD, polynomial expression evaluation through Horner' method, algorithms for evaluation of exponentiation and simple sorting algorithms. Selection sort begins by finding the least element in the array which is moved **to** the first position. Then the second least element is searched for and moved the second position in the array. This process continues until the entire array is sorted.

Objectives

The main objectives of the algorithms are to :

- Implement Euclid's algorithm to find GCD
- Implement Horner's method to evaluate polynomial expression
- Compare the performance of Horner's method with brute force method
- Implement simple sorting algorithms

Implement multiplication of two matrices

Problems for Implementation

Q1. Implement Euclid's algorithm to find GCD (15265, 15)and calculate the number of times **mod** and assignment operations will be required.

Q2(i). Implement Horner's rule for evaluating polynomial P(x)= $6x^6+5x^5+4x^4-3x^3+2x^2+8x-7$ at x = 3. Calculate how many times (i) multiplications and addition operations will take (ii) how many times the loop will iterate

Q2(ii) Apply a brute force method to implement the above polynomial expression(Q2(i)) and compare it with Horner's method in terms of number of multiplication operations

Q3.Implement multiplication of two matrices A[4,4] and B[4,4]

and calculate (i) how many times the innermost and the outermost loops will run (ii) total number of multiplications and additions in computing the multiplication

Q4. Implement left to right and right to left binary exponentiation methods for the following problems:

(i) 4^{512} (ii) 3^{31}

Calculate the followings for problems (i) and (ii)

- How many times the loops will be executed?
- How many times the multiplication and additions operations will be executed?

Q5. Implement Bubble Sort algorithm for the following list of numbers:

55 25 15 40 60 35 17 65 75 10

Calculate (i) a number of exchange operations (ii) a number of times comparison operations (iii) a number of times the inner and outer loops will iterate?

Q6. Implement Selection Sort algorithm on a similar set of data as in Q5 and (i)perform similar operations on the above example (ii) make a comparison between the two algorithms in terms of best case and worst case complexities

Session 2: Fractional Knapsack Problem

Introduction

In Greedy algorithm, the solution that looks best at a moment is selected with the hope that it will lead to the optimal solution. This is one of the approaches to solve optimization problems. This is an optimization problem which we want to solve it through greedy technique. In this problem , a Knapsack (or bag) of some capacity is considered which is to be filled with objects. We are given some objects with their weights and associated profits. The problem is to fill the given Knapsack with objects such that the sum of the profit associated with the objects that are included in the Knapsack is maximum. The constraint in this problem is that the sum of the weight of the objects that are included in the Knapsack should be less than or equal to the capacity of the Knapsack. However, the objects can be included in fractions to the Knapsack because of which this problem is termed as Fractional Knapsack problem.

Objectives

The main objectives of the session are to:

- Implement Fractional Knapsack Problem
- Test the implementation on different problem instances
- Implement greedy technique in general

Problems for Implementation

Implement Fractional Knapsack algorithm and find out optimal result for the following problem instances:

Q1 (P_1, P_2, P3, P4, P5, P6, P_7) = (15, 5, 20, 8, 7, 20, 6)

(W_1, W_2, W3, W4,W5,W6,W7) = (3, 4, 6, 8, 2, 2, 3)

Maximum Knapsack Capacity = 18

Q2 (P_1, P_2, P_3, P_4, P_5) = (20, 30,40, 32, 55)

(W_1, W_2, W_3, W_4, W_5) = (5, 8, 10 12, 15)

Maximum Knapsack Capacity = 20

Q3 (P_1, P_2, P_3, P_4, P_5, P_6, P_7) = (12, 10, 8, 11, 14, 7, 9)

(W_1, W_2, W_3, W_4, W_5 , W_6, W_7) = (4, 6, 5, 7, 3, 1, 6)

Session 3 : Task Scheduling Algorithm

Introduction

A task scheduling problem is formulated as an optimization problem in which we need to determine the set of tasks from the given tasks that can be accomplished within their deadlines along with their order of scheduling such that the profit is maximum. So, this is a maximization optimization problem with a constraint that tasks must be completed within their specified deadlines.

Objectives

The main objectives of this session are to:

- Implement a task scheduling algorithm
- Test the algorithm on different problem instances
- Differentiate between a brute force approach and an efficient task scheduling algorithm

Problems for Implementation

Q1. Apply a brute force approach to schedule three jobs J1, J2 and J3 with service times as 5,8,12 respectively. The actual service time units are not relevant to the problems. Make all possible job schedules , calculate the total times spent in jobs by the system. Find the optimal schedule (total time). If there are N jobs , what would be the total number of job schedules?

Q2.Implement the task scheduling algorithm on your system to minimize the total amount of time spent in the system for the following problem:

Job	Service Time
1	5
2	10
3	7
4	8

Q3. Consider the following jobs, deadlines and profits. Implement the task scheduling algorithm with deadlines to maximize the total profits.

Jobs	Deadlines	Profits
1	3	50
2	4	20
3	5	70
4	3	15
5	2	10
6	1	47
7	1	60

Session 4: Huffman's Coding Algorithm

Introduction

Huffman coding is a greedy algorithm that is used to compress the data. Data can be sequence of characters and each character can occur multiple times in a data called as frequency of that character. Basically compression is a technique to reduce the size of the data. Huffman coding compress data by 70-80%. Huffman algorithm checks the frequency of each data, represent those data in the form of Huffman tree and build an optimal solution to represent each character as a binary string. Huffman coding is also known as variable length coding.

Objectives

The main objectives of this session are to:

- Implement Huffman's Coding algorithm.
- Test the implementation on different problem instances
- Construct the optimal binary prefix code

Problems for Implementation

Q1. Implement Huffman's coding algorithm and run on the problem instance of Q2.

Q2 Apply Huffman's algorithm to construct an optimal binary prefix code for the letters and its frequencies in the following table.

Letters	A	B	I	M	S	X	Z
Frequency	10	7	15	8	10	5	2

Show the complete steps.

Q3 What is an optimal binary tree and Huffman code for the following set of frequencies? Find out an average number of bits required per character. Show the complete steps.

A:15 B:25 C:5 D:7 E:10 F:13 G:9

Q4. A file contains only colon, spaces, new line character and digits in the following frequency:

colon (80), space (500), new line (110), commas (500), 0 (300), 1 (200), 2 (150), 3(60), 4 (180), 5 (240), 6 (170), 7 (200), 8 (202).

Using the Huffman's algorithm to construct an optimal binary prefix code.

Session 5: Divide and Conquer Technique

Introduction

In Divide and conquer approach, the original problem is divided into two or more sub-problems recursively, till it is small enough to be solved easily. Each sub-problem is some fraction of the original problem. Next, the solutions of the sub-problems are combined together to generate the solution of the original problem.

Objectives

The main objectives of this session are to:

- Write recurrence relation of a problem
- Implementations of Binary Search, Merge Sort and Quick Sort algorithms
- Draw a tree of recursive calls of recursive calls

Problems for Implementation

Q1 Implement a recursive binary search algorithm on your system to search for a number 100 in the following array of integers. Show the processes step by step:

10 35 40 45 50 55 60 65 70 100

Draw recursive calls to be made in this problem

Q2 Suppose that we are required to search for 512 million items in a list using binary search algorithm. What is the maximum number of searches needed to find a given item in the list or coming to the conclusion that the item is not found in the list.

Q3. Implement Merge Sort algorithm to sort the following list show the process step by step.

200 150 50 100 75 25 10 5

Draw a tree of recursive calls in this problem.

Q4 Implement Quick Sort's algorithm on your machine to do sorting of the following list of elements

12 20 22 16 25 18 8 10 6 15

Show step by step processes.

Q5. Examine the performance of Quick Sort algorithm implemented in the previous problem for the following list in terms of a number of comparisons, exchange operations and the number of times the loop will iterate?

6 8 10 12 15 16 18 20 22 25

Q6. Implement the Stassen's multiplication algorithm two matrices A and B of n* n, where n is a power of 2 on different problem instances and compare it(all the instances) in terms of a number of multiplications and additions required.

Session 6: Single Source Shortest Path Algorithm

Introduction

Dijkstra's algorithm solves the single-source shortest path problem when all edges have non-negative weights. It is a very similar to Prim's algorithm and always choose the path that are optimal right now but not for global optimizations

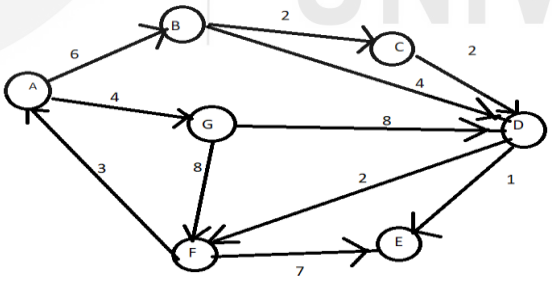
Objectives

Main objectives are to:

- Implement Dijkstra's algorithm to implement a single source shortest path using greedy technique
- Apply the implemented algorithm on different problem instances
- Represent a graph
- Find out the similarities in implementations of Prim's algorithm and Dijkstra's algorithm

Problems for Implementation

Implement Dijkstra's algorithm to find the single source shortest path algorithm from different sources to the rest of nodes in the following graph and show all the intermediate processes:



Q1 Find the shortest path from A to the rest of vertices

Q2 Find the shortest path from B to the rest of nodes

Q3. Find the shortest path from A to the rest of vertices but there are negative weights(-8) between G and F and (-3) between F and A

Session 7: Minimum Cost Spanning Tree

Introduction

A connected sub graph **S** of Graph **G**(**V**, **E**) is said to be spanning tree if and only if it contains all the vertices of the graph **G** and have minimum total weight of the edges of **G**. Spanning tree must be acyclic. A Spanning tree whose weight is minimum over all spanning trees is called a minimum spanning tree or MST

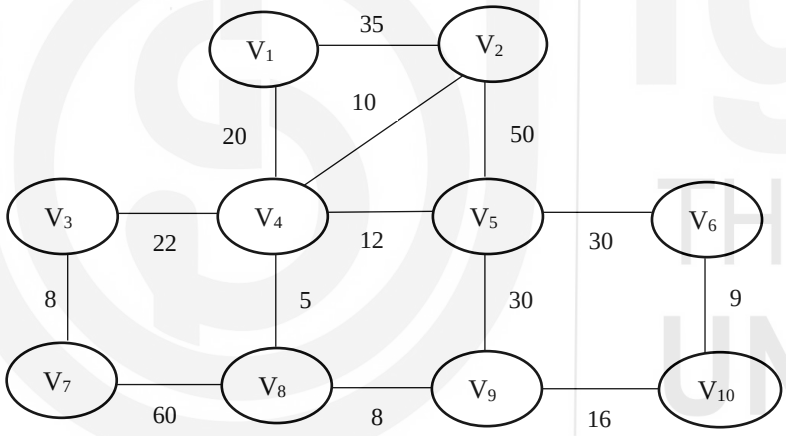
Objectives

The main objectives of the session are to:

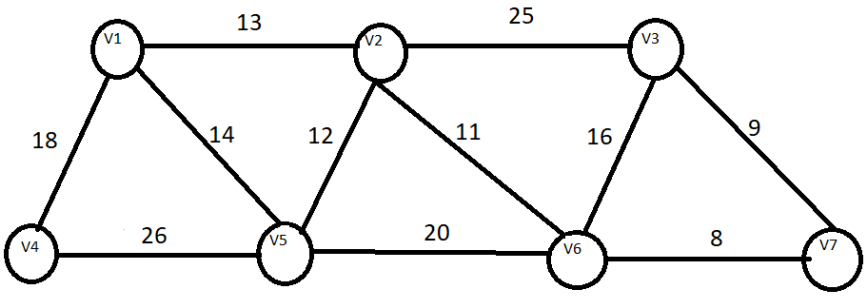
- Implement Prim's algorithm to find a minimum cost spanning tree
- Implement Kruskal's algorithm to find a minimum cost spanning tree
- Differentiate between the implementations of two algorithms
- Implementation of disjoint data structure

Problem for Implementation

Q1. Implement Prim's algorithm to find a minimum cost spanning tree (MCST) in the following graph. Show all the processes.



Q2. Implement Kruskal's algorithm to find a MCST for the following graph. Show all the processes.



Q3. Analyze the performance of both the algorithms on different problem instances and write a brief report

Session 8: Implementation of Binomial Coefficient Problem

Introduction

Binomial Coefficient is a part of permutation and combination. Computing binomial coefficients is non optimization problem but can be solved using dynamic programming. **Binomial Coefficient** is the coefficient in the Binomial Theorem which is an arithmetic expansion. It is denoted as c(n, k) which is equal to $\frac{n!}{k! \times (n-k)!}$where ! denotes factorial.

gives combinations to choose k elements from n-element set. We can compute n_{c_k} for any n and k using the recurrence relation as follows:

$$n_{C_k} = \begin{cases} 1, & if\ k=0\ or\ n=k \\ n-1C_{k-1} + n-1C_{k,} & for\ n>k>0 \end{cases}$$

It is possible to compute binomial coefficient in linear time $O(n \times k)$using Dynamic Programming.

Objectives

The main objectives of this section are to:

- understand the application of binomial coefficient
- define binomial coefficient recursively
- solve the binomial coefficient problem through divide and conquer and dynamic programming techniques
- do performance analysis of both techniques on different problem instances for large and small values of n and k

Problems for Implementation

Q1. Implement a binomial coefficient problem using Divide and Conquer technique

Q2. Implement a binomial coefficient problem using Dynamic Programming technique

Q3. Study the performance of both implementations using five problem instances in terms of the efficiency of both the approaches for large and small values of n and k in the problem instances.

Session 9: Floyd and Warshall's Algorithm for All Pair Shortest Path Problems

Introduction

Unlike the single source shortest path algorithm by Dijkstra's, Floyd Warshall's all pair shortest path algorithm computes the shortest path between any two vertices of a graph at one go. Floyd Warshall Algorithm uses the Dynamic Programming (DP) methodology. Unlike Greedy algorithms which always

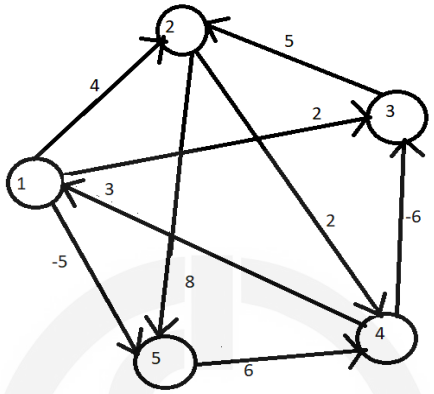
looks for local optimization, DP strives for global optimization that means DP does not rely on the intermediate(local) best results.

Main objectives of this session are to:

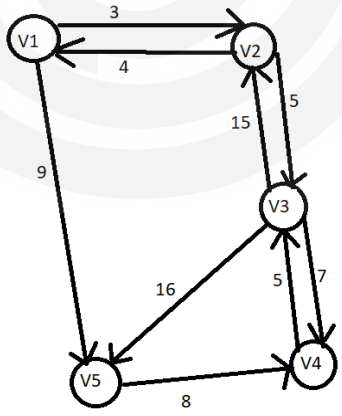
- Implement Floyd and Warshall's algorithm using dynamic programming
- Analyze the performance of the algorithm using different graph instances(large graphs, small graphs

Problems for Implementation

Q1. Apply the Floyd and Warshall's algorithm for the following graph. Show the matrix D^5 of the graph and find out the shortest path .



Q2. Apply the Floyd and Warshall's algorithm to compute the shortest path to the following graph. Compute D^5 matrix of the graph



Q3. Implement the all pair shortest path problems using different graphs

Session 10: Chained Matrix Multiplication

Introduction

The problem of matrix-chain-multiplication is the process of selecting the optimal pair of matrices in every step in such a way that the overall cost of multiplication would be minimal. If there are total N matrices in the sequence

then the total number of different ways of selecting matrix-pairs for multiplication will be ${}^{2n}C_n/(n+1)$. We need to find out the optimal order for multiplying n matrices.

Objectives

The main objectives of the sessions are to:

- List all the different orders in which we can multiply matrices
- Implement the chained matrix multiplications using dynamic programming technique

Problems for Implementation

Q1. List different orders for evaluating the product of A,B,C,D,E matrices.

Q2. Find the optimal order for evaluating the product A * B * C *D *E where

- A is (10 * 4)
- B is (4 *5)
- C is (5 * 20)
- D is (20 * 2)
- E is (2* 50)

Q3. Implement the chained matrix multiplication and print the optimal parentheses algorithms and study the performance of the algorithms on different problem instances.

Session 11: Optimal Binary Search Tree

Introduction

The problem of optimal binary search tree is, given a keys and their probabilities, we have to generate a binary search tree such that the searching cost should be minimum. Using dynamic programming we can find the optimal binary search tree without drawing each tree and calculating the cost of each tree. Thus, dynamic programming gives better, easiest and fastest method for trying out all possible binary search tree and picking up best one without drawing each sub tree

Objectives

The main objectives of this session are to:

- Determine the cost and structure of an optimal binary search tree
- Implement an optimal binary search tree and study the performance of the algorithm using different problem instances

Problems for Implementation

Determine the cost and structure of an optimal binary search tree for a set of n=7 keys with the following properties. Show the step by step processes.

Q1.

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

Q2.

I	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Q3 Implement the optimal binary search tree algorithm on your system and study the performance of the algorithm on different problem instances