ACADEMIC LAB MANUAL

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

VISION

Engineering the future of the nation by transforming the students to be skilled technocrats, innovative leaders and environmentally receptive citizens. The Vision of the department is to carve the youth as dynamic, competent, valued and knowledgeable professionals who shall lead the Nation to a better future.

MISSION

- To flourish the SRMS as the World Leader in Computer Science & Engineering through continuous research & development directed towards the betterment of the society.
- To establish the cooperative learning environment for facilitating the quality academics, state-of-the art research and remarkable development activities.
- To establish World Class resources especially Research & Development Laboratories, Value Addition courses etc. for the in-house up gradation & community services.
- To groom the students into Industry – Ready Professionals through a rigorous training in a self-disciplined environment.
- To groom the learned pool of faculty in accordance with the recent advancements in the field of Computer Science & Engineering.

PROGRAM EDUCATIONAL OBJECTIVES

- To encourage students to use their practical, computer and analytical skills to build industry ready engineers to solve multi-disciplinary sustainable projects.
- To keep abreast the students with the use of modern tools, equipments and software and inculcating the habit of lifelong learning.
- To foster team work and professional ethics among students towards devising feasible solutions to problems and project work.
- To augment the existing facilities: Library, Labs and efforts excel classroom teaching, thereby arousing curiosity, ultimately resulting in innovative ideas.
- To enhance technical skills of laboratory staff, provision to train the lab staff, encouraging staff to improve qualifications offering incentives.

PROGRAM SPECIFIC OUTCOMES

- Foundation of mathematical concepts: To use mathematical concepts to solve problem using suitable mathematical analysis, data structure and suitable algorithm.
- Foundation of Computer System: the ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.
- Foundations of Software development: the ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process. Familiarity and practical proficiency with a broad area of programming concepts and provide new ideas and innovations towards research.

KCS-551: DATABASE MANAGEMENT SYSTEM LAB

LIST OF PROGRAMS

1. Installing oracle/ MYSQL
2. Creating Entity-Relationship Diagram using case tools.
3. Writing SQL statements Using ORACLE /MYSQL:
    a) Writing basic SQL SELECT statements.
    b) Restricting and sorting data.
    c) Displaying data from multiple tables.
    d) Aggregating data using group function.
    e) Manipulating data.
    f) Creating and managing tables.
4. Normalization
5. Creating cursor
6. Creating procedure and functions
7. Creating packages and triggers
8. Design and implementation of payroll processing system
9. Design and implementation of Library Information System
10. Design and implementation of Student Information System
11. Automatic Backup of Files and Recovery of Files
12. Mini project (Design & Development of Data and Application) for following :
    a) Inventory Control System.
    b) Material Requirement Processing.
    c) Hospital Management System.
    d) Railway Reservation System.
    e) Personal Information System.
    f) Web Based User Identification System.
    g) Timetable Management System.
    h) Hotel Management System
13. Design and implementation of payroll processing system
14. Design and implementation of Library Information System
15. Design and implementation of Student Information System

KCS-552: COMPILER DESIGN LAB

LIST OF PROGRAMS

1. Design and implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and new lines
2. Implementation of Lexical Analyzer using Lex Tool
3. Generate YACC specification for a few syntactic categories.
    a) Program to recognize a valid arithmetic expression that uses operator +, – , * and /.
    b) Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.
    c) Implementation of Calculator using LEX and YACC
    d) Convert the BNF rules into YACC form and write code to generate abstract syntax tree
4. Write program to find ε – closure of all states of any given NFA with ε transition.
5. Write program to convert NFA with ε transition to NFA without ε transition.
6. Write program to convert NFA to DFA
7. Write program to minimize any given DFA.
8. Develop an operator precedence parser for a given language.
9. Write program to find Simulate First and Follow of any given grammar
10. Construct a recursive descent parser for an expression.
11. Construct a Shift Reduce Parser for a given language.
12. Write a program to perform loop unrolling
13. Write a program to perform constant propagation.
14. Implement Intermediate code generation for simple expressions.
15. Implement the back end of the compiler which takes the three address code and produces the 8086

assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

KCS-553: DESIGN AND ANALYSIS OF ALGORITHM LAB

LIST OF PROGRAMS

1. Program for Recursive Binary & Linear Search.
2. Program for Heap Sort.
3. Program for Merge Sort.
4. Program for Selection Sort.
5. Program for Insertion Sort.
6. Program for Quick Sort.
7. Knapsack Problem using Greedy Solution
8. Perform Travelling Salesman Problem
9. Find Minimum Spanning Tree using Kruskal's Algorithm
10. Implement N Queen Problem using Backtracking
11. Sort a given set of n integer elements using Quick Sort method and compute its time complexity.Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of thetime taken versus non graph sheet. The elements can be read from a file or can be generated using therandom number generator. Demonstrate using Java how the divide and-conquer method works alongwith its time complexity analysis: worst case, average case and best case.
12. Sort a given set of n integer elements using Merge Sort method and compute its time complexity.Run the program for varied values ofn> 5000, and record the time taken to sort. Plot a graph of thetime taken versus non graph sheet. The elements can be read from a file or can be generated using therandom number generator. Demonstrate how the divide and-conquer method works alongwith its time complexity analysis: worst case, average case and best case.
13.6. Implement , the 0/1 Knapsack problem using
    (a) Dynamic Programming method
    (b)Greedy method.
14. From a given vertex in a weighted connected graph, find shortest paths toother vertices usingDijkstra's algorithm.
15. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal'salgorithm. Use Union-Find algorithms in your program.
16.Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.
17. Write programs to
    (a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.
    (b) Implement Travelling Sales Person problem using Dynamic programming.
18. Design and implement to find a subset of a given setS = {Sl, S2,.....,Sn} of n positiveintegers whose SUM is equal to a given positive integer d. For example, if S ={1, 2, 5, 6, 8} and d= 9,there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instancedoesn't have a solution.
19. Design and implement to find all Hamiltonian Cycles in a connected undirected Graph Gof n vertices using backtracking principle.

KCS-551: DATABASE MANAGEMENT SYSTEM LAB

I. COURSE OUTCOME:

Course	Statement
C 307.1	Design and implement a database schema for a given problem-domain.
C307.2	Create and maintain tables using SQL.
C307.3	Able to perform data manipulation operation on database table using sql.
C307.4	Able to implement data definition operation on database.
C307.5	Able to perform data sorting on table using sql query.

II. PROGRAM OUTCOMES:

PO1	**Engineering Knowledge**: Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE.
PO2	**Problem Analysis**: Identify, Formulate, review research literature and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
PO3	**Design/Development of solutions**: Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural societal and environmental considerations
PO4	**Conduct Investigations of Complex problems**: Use research–based knowledge and research methods including design of experiments, analysis andinterpretationofdata,andsynthesisoftheinformationtoprovidevalid conclusions.
PO5	**Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO9	**Individual and Team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

III. PROGRAM SPECIFICOUTCOME

PS01: Foundation of mathematical concepts: To use mathematical concepts to solve problem using suitable mathematical analysis, data structure and suitable algorithm.

PSO2: Foundation of Computer System: the ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.

PSO-3: Provide effective and efficient real time solutions using acquired knowledge in various domains.

IV.MAPPING OF CO-PO ANDCO-PSO

Course Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C 307.1			3									
C307.2					**2**							
C307.3					**3**							
C307.4					**2**							
C307.5					**3**							
Average			**3**		**2**							

ASSIGNMENT NO -1

OBJECT: Create tables and specify the Queries in SQL.

THEORY & CONCEPTS:

SQL (Structured Query Language) is a nonprocedural language, you specify what you want, not how to get it. A block structured format of English key words is used in this Query language. It has the following components.

DDL (Data Definition Language)-

The SQL DDL provides command for defining relation schemas, deleting relations and modifying relation schema.

DML (DATA Manipulation Language)-

It includes commands to insert tuple into, delete tuple from and modify tuple in the database.

The SQL standard supports a variety of built in types, including-

- Char (n)- A fixed length character length string with user specified length .
- Varchar (n)- A variable character length string with user specified maximum length n.
- Int- An integer.
- Numeric (p, d)-A Fixed point number with user defined precision.
- Date- A calendar date containing a (four digit) year, month and day of the month.
- Number- Number is used to store numbers (fixed or floating point).

DDL statement for creating a table-

SQL>CREATE TABLE tablename
(Columnname1 datatype (size), columnname2 datatype (size)…);

Insertion of data into tables-

SQL> INSERT INTO tablename
(columnname, columnname, ……) Values (value1, value2);

Retrieving of data from the tables-

SQL> SELECT * FROM tablename1, tablename2,

The retrieving of specific columns from a table-

SQL> SELECT columnname1, columnname2,
FROM tablename1, tablename2

Elimination of duplicates from the select stateent-

SQL> SELECT DISTINCT columnname1, columnname2,
FROM tablename;

Selecting a data set from table data-

SQL> SELECT columnname1, columnname2,…
FROM tablename1, tablename2 …
WHERE searchcondition;

VIVA QUESTIONS

A) What is DDL and DML?

b) What are the different data types used in sql?

ASSIGNMENT NO -2

OBJECT: - Perform the below Operations on the table created in Lab 1**.**

THEORY AND CONCEPT

Problem1: Updating the content of a table:

In creation situation we may wish to change a value in table without changing all values in the tuple . For this purpose the update statement can be used.

SQL>UPDATE tablename
SET columnname = expression, columnname =expression……
Where columnname = expression;

Problem2: Deletion Operation:-

We can delete whole tuple (rows) or we can delete values on only particulars attributes.

Deletion of all rows

SQL>DELETE FROM tablename:

Deletion of specified number of rows

SQL>DELETE FROM tablename
WHERE columnname=searchcondition ;

Problem3: Renaming columns or using alias name: in select statement used with Expression Lists: - The default output column names can be renamed by the user if required (only for display purpose)

Syntax:

```
SQL>SELECT columnname          newname,
       Columnname              newname,
        FROM tablename;
```

Problem4: Add a new column to the table (single column)

```
SQL>ALTER TABLE tablename
          ADD columnname    column-definition;
```

Problem5:Add multiple columns in table

```
SQL>ALTER TABLE tablename ADD
          (
            Columname1      column-definition,
           Columnname2      column-definition,
           ...
            Columnname       column -definition
          );
```

***Problem*6: Modify column in table**

```
SQL>ALTER TABLE tablename
          MODIFY columnname      column -definition;
```

Modify multiple columns in table

```
SQL>ALTER TABLE tablename
          MODIFY (columnname1     column -definition,
                  columnname2     column -definition,
                          ...
                   columnname n       column -definition
                  );
```

VIVA QUESTIONS

a) How to delete all the rows in a table?
b) How will you view the structure of a table?

ASSIGNMENT NO -3

OBJECTIVE : PERFORM FOLLOWING OPERATION
THEORY AND CONCEPT

Problem1: Drop column in table
SQL>ALTER TABLE tablename
DROP COLUMN columnname;
Problem2: Rename column in table
SQL>ALTER TABLE tablename
RENAME COLUMN oldname TO newname;
Problem3: Rename table
SQL>ALTER TABLE oldtablename RENAME TO newname;

Problem 4: Drop A Table
SQL>DROP TABLE tablename;

VIVA QUESTIONS

a)How ALTER command works?
b)Is there any difference between dropping a table and deleting all the rows of a table?

ASSIGNMENT NO -4

OBJECT:- To Implement the restrictions on the table column and table row.
THEORY AND CONCEPT

Besides the cell name, cell length and cell data type there are other parameters i.e. other data constrains that can be passed to the DBA at creation time. The constraints can either be placed at column level or at the row level.
Column Level Constraints: If the constraints are defined along with the column definition, it is called a column level constraint.
Row Level Constraints: If the data constraint attached to a specify cell in a table reference the contents of another cell in the table then the user will have to use table level constraints.

Problem 1: NOT NULL
SQL>CREATE TABLE tablename
(
columnname1 datatype (size) NOT NULL,
columnname2 datatype (size) NOT NULL
);

Problem 2:-Primary Key: primary key is one or more columns is a table used to uniquely identify each row in the table. Primary key values must not be null and must be unique across the column. A multicolumn primary key is called composite primary key.
SQL>CREATE TABLE tablename
(columnname datatype (size) primary key,….);
Primary key as a table constraint (MULTICOLUMN/COMPOSITE KEY)
SQL> CREATE TABLE tablename
(
Columnname1 datatype (size),
Columnname2 datatype (size)…
CONSTRAINT constraintname PRIMARY KEY (columnname, columnname)
);
Problem 3:UNIQUE key:-A **UNIQUE** is similar to a primary key except that the purpose of a **UNIQUE** key is to ensure that information in the column for each record is **UNIQUE** as with telephone or devices license numbers. A table may have many **UNIQUE** keys.
SQL> CREATE TABLE tablename
(Columnname datatype (size) **UNIQUE**);
UNIQUE as table constraint:
SQL> CREATE TABLE tablename

(columnname datatype (size),
columnname datatype (size)…
CONSTRAINT constraintname **UNIQUE** (columnname, columnname)
);

Problem 4:Default value concept: At the time of cell creation a default value can be assigned to it. When the user is loading a record with values and leaves this cell empty, the DBA wil automatically load this cell with the default value specified. The data type of the default value should match the data type of the column

SQL> CREATE TABLE tablename
(columnname datatype (size) default value,….);

Problem 5:Foreign Key Concept: A foreign key is a way to enforce referential integrity within your Oracle database. A foreign key means that values in one table must also appear in another table.The referenced table is called the ***parent table*** while the table with the foreign key is called the ***child table***. The foreign key in the child table will generally reference a primary key in the parent table.A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement;

```
SQL> CREATE TABLE table_name
(
 column1 datatype null/not null,
 column2 datatype null/not null,
 …
 CONSTRAINT fk_column
  FOREIGN KEY (column1, column2, … column_n)
  REFERENCES parent_table (column1, column2, … column_n)
);
```

Example

```
SQL> CREATE TABLE        supplier
( supplier_id          numeric(10) not null,
 supplier_name       varchar2(50) not null,
 contact_name        varchar2(50),
 CONSTRAINT     supplier_pk          PRIMARY KEY (supplier_id)
);

SQL> CREATE  TABLE       products
( product_id           numeric(10) not null,
 supplier_id           numeric(10) not null,
 CONSTRAINT     fk_supplier
  FOREIGN KEY   (supplier_id)
  REFERENCES   supplier(supplier_id)
);
```

Problem 6:Using an ALTER TABLE statement

```
SQL>ALTER       TABLE   table_name
          ADD       CONSTRAINT      constraint_name
           FOREIGN KEY      (column1, column2, … column_n)
           REFERENCES       parent_table (column1, column2, … column_n);
```

Example

```
SQL>ALTER       TABLE products
          ADD       CONSTRAINT      fk_supplier
          FOREIGN KEY     (supplier_id)
          REFERENCES      supplier(supplier_id);
SQL>ALTER       TABLE   products
          ADD CONSTRAINT          fk_supplier
           FOREIGN KEY     (supplier_id, supplier_name)
           REFERENCES      supplier(supplier_id, supplier_name);
```

VIVA QUESTIONS

a) What is the use of select command?

b) How will you impose some constraint in rows and columns in a table.

ASSIGNMENT NO – 5

OBJECTIVE: To implement DDL statements.

Check Integrity Constraints: Use the check constraints when you need to enforce intergrity rules that can be evaluated based on a logical expression following are a few examples of appropriate check constraints. A **check constraint** allows you to specify a condition on each row in a table. A check constraint can NOT be defined on a SQL View.The check constraint defined on a table must refer to only columns in that table. It cannot refer to columns in other tables.A check constraint can NOT include a SQL Subquery.A check constraint can be defined in either a SQL CREATE TABLE statement or a SQL ALTER TABLE statement. This constraint ensures that the supplier_id field contains values between 100 and 9999.

```
SQL>CREATE TABLE suppliers
(  supplier_id numeric(4),
  supplier_name varchar2(50),
  CONSTRAINT check_supplier_id
  CHECK (supplier_id BETWEEN 100 and 9999)   );
```

This constraint ensures that the supplier_name column always contains uppercase characters.

```
SQL>CREATE TABLE suppliers
( supplier_id numeric(4),          supplier_name varchar2(50),
 CONSTRAINT check_supplier_name
 CHECK (supplier_name = upper(supplier_name))       );
```

Using an ALTER TABLE statement

```
SQL>ALTER TABLE table_name
ADD CONSTRAINT constraint_name CHECK (column_name condition) [DISABLE];
```

Ex:It ensures that the supplier_name field only contains the following values: IBM, Microsoft, or NVIDIA.

```
SQL>ALTER TABLE suppliers
ADD CONSTRAINT check_supplier_name
  CHECK (supplier_name IN ('IBM', 'Microsoft', 'NVIDIA'));
```

Drop a Check Constraint

```
SQL>ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

Disable/ Enable a Check Constraint

```
SQL>ALTER  TABLE  table_name
ENABLE/Disable CONSTRAINT constraint_name;
```

ORDER BY Clause: The Oracle ORDER BY clause is used to sort the records in your result set. The ORDER BY clause can only be used in SELECT statements.

```
SQL>SELECT expressions
FROM tables
WHERE conditions
ORDER BY expression [ ASC | DESC ];
```

Example:

```
SQL>SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'Microsoft'
ORDER BY supplier_city;--by default ASC
```

Sorting by relative position

You can also use the Oracle ORDER BY clause to sort by relative position in the result set, where the first field in the result set is 1. The next field is 2, and so on.

```
SQL>SELECT *FROM suppliers
WHERE supplier_name = 'Microsoft'
ORDER BY 1 DESC;
```

Using both ASC and DESC attributes

When sorting your result set using the Oracle ORDER BY clause, you can use the ASC and DESC attributes in a single SELECT statement.

```
SQL>SELECT supplier_city, supplier_state
FROM suppliers
WHERE supplier_name = 'Microsoft'
ORDER BY supplier_city DESC, supplier_state ASC;
```

VIVA QUESTIONS

a) How to set primary key in a table?

b) What is a foreign key?

ASSIGNMENT NO – 6

OBJECT: - To implement the concept of Joins

THEORY AND CONCEPT

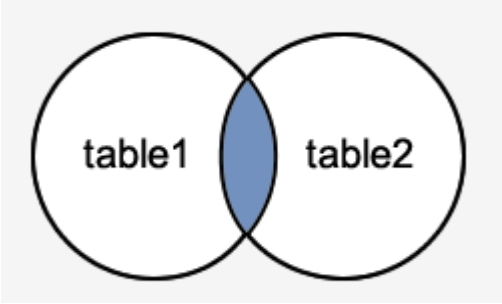
Oracle JOINS are used to retrieve data from multiple tables. An Oracle JOIN is performed whenever two or more tables are joined in a SQL statement.

There are 4 different types of Oracle joins:

- Oracle INNER JOIN (or sometimes called simple join)
- Oracle LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- Oracle RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)
- Oracle FULL OUTER JOIN (or sometimes called FULL JOIN)

INNER JOIN (simple join)

It is the most common type of join. Oracle INNER JOINS return all rows from multiple tables where the join condition is met



- The Oracle INNER JOIN would return the records where ***table1*** and ***table2*** intersect.

```
SQL>SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

We have a table called *suppliers* with two fields (supplier_id and supplier_name). It contains the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have another table called *orders* with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2003/05/12
500126	10001	2003/05/13
500127	10004	2003/05/14

If we run the Oracle SELECT statement (that contains an INNER JOIN) below:

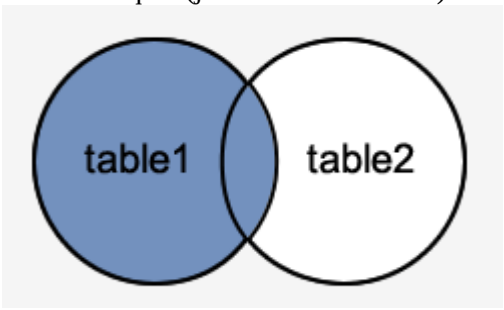
```
SQL>SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

supplier_id	name	order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13

LEFT OUTER JOIN

Another type of join is called an Oracle LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

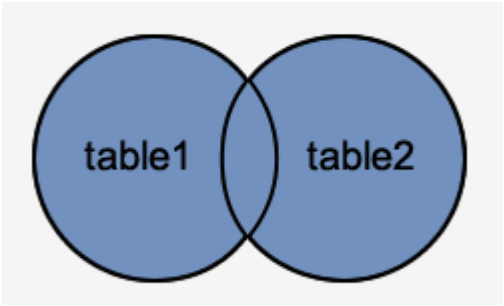


SQL>SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
LEFT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
Our result set would look like this:

supplier_id	supplier_name	order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13
10002	Microsoft	<null>
10003	NVIDIA	<null>

FULL OUTER JOIN

Another type of join is called an Oracle FULL OUTER JOIN. This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.



SQL>SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
FULL OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;

supplier_id	supplier_name	order_date
10000	IBM	2013/08/12
10001	Hewlett Packard	2013/08/13
10002	Microsoft	<null>
10003	NVIDIA	<null>
<null>	<null>	2013/08/14

VIVA QUESTIONS

a) What is natural join? How is it different from Cartesian product?
b) What do you mean by full outer join?

ASSIGNMENT NO -7

OBJECTIVE : Implement the concept of aggregate and scalar functions in SQL.

THEORY AND CONCEPT

```
SELECT AVG(salary)
FROM employee;
```

This statement will return a single result which contains the average value of everything returned in the salary column from the ***employee*** table.
Another example:

```
SELECT AVG(salary)

FROM employee

WHERE title = 'Programmer';
```

This statement will return the average salary for all employees whose title is equal to 'Programmer'
Example:

```
SELECT Count (*)

FROM employee;
```

VIVA QUESTIONS

a) What are the different aggregate functions in sql?
b) Name a few scalar functions supported in sql.

ASSIGNMENT NO -8

OBJECT:- To implement the concept of grouping of Data**.**

THEORY AND CONCEPT

GROUP BY Clause

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuple, we specify this wish in SQL using the group by clause. The attribute or attributes given in the group by clause are used to form group. Tuple with the same value on all attributes in the group by clause are placed in one group.

```
SQL>SELECT columnname, columnname
FROM   tablename
GROUP BY columnname;
```

At times it is useful to state a condition that applies to groups rather than to tuple. For example we might be interested in only those branches where the average account balance is more than 1200. This condition does not apply to a single tuple, rather it applies to each group constructed by the GROUP BY clause. To express such Query, we use the having clause of SQL. SQL applies predicates in the having may be used.

```
SQL>SELECT columnname, columnname
FROM   tablename
GROUP BY columnname;
HAVING searchcondition;
```

Complete Syntax is:

```
SQL>SELECT expression1, expression2, ... expression_n,
     aggregate_function (aggregate_expression)
FROM tables
WHERE conditions
GROUP BY expression1, expression2, ... expression_n;
```

HAVING Clause

The Oracle HAVING clause is used in combination with the GROUP BY clause to restrict the groups of returned rows to only those whose the condition is TRUE.

```
SQL>SELECT expression1, expression2, ... expression_n,
          FROM tables
          WHERE conditions…….
          GROUP BY expression1, expression2,…..
          HAVING having_condition;
```

```
SQL>SELECT department, SUM(sales) AS "Total sales"
FROM order_details
GROUP BY department
HAVING SUM(sales) > 25000;
```

VIVA QUESTIONS

.a)Can we use HAVING keyword without using GROUP BY clause?

b)How will you organize the data of a table in descending order of roll no?

ASSIGNMENT NO - 9

OBJECT:- To implement the concept of Sub Questionries**.**

THEORY AND CONCEPT

SUBQUESTIONRIES:- A subQuestionry is a form of an SQL statement that appears inside another SQL statement. It also termed as nested Questionry. The statement containing a subQuestionry called a parent statement. The rows returned bu the subQuestionry are use by the following statement.
It can be used by the following commands:

1.To insert records in the target table. 2. To create tables and insert records in this table.
3.To update records in the target table. 4. To create view.
5.To provide values for the condition in the WHERE , HAVING IN , SELECT,UPDATE, and DELETE statements.

Exam:- Creating clientmaster table from oldclient_master, table
Create table client_master
AS SELECT * FROM oldclient_master;

Using the Union, Intersect and Minus Clause:

- **UNION CLAUSE**: The user can put together multiple Questionries and combine their output using the union clause . The union clause merges the output of two or more Questionries into a single set of rows and column. The final output of union clause will be

Output: = Records only in Questionry one + records only in Questionry two + A single set of records with is common in the both Questionries.

Syntax:
```
SELECT columnname, columname
FROM tablename 1
UNION
SELECT columnname, columnname
From tablename2;
```

INTERSECT CLAUSE: The use can put together multiple Questionries and their output using the interest clause. The final output of the interest clause will be :

Output =A single set of records which are common in both Questionries

Syntax:
```
SELECT columnname, columnname
FROM tablename 1
INTERSECT
SELECT columnname, columnname
FROM tablename 2;
```

MINUS CLAUSE:- The user can put together multiple Questionries and combine their output = records only in Questionry one

Syntax:
```
SELECT columnname, columnname
FROM tablename ;
MINUS
SELECT columnname, columnname
FROM tablename ;
```

VIVA QUESTIONS

a) What do you mean by subquery?
b) What is co related sub query?

ASSIGNMENT NO – 10

OBJECT:- To implement the basics of PL/SQL.

THEORY AND CONCEPT

INTRODUCTION – PL/SQL bridges the gap between database technology and procedural programming languages. It can be thought of as a development tool that extends the facilities of Oracles SQL database language. Via PL/SQL you can insert, delete, update and retrieve table data as well as use procedural techniQuestions such as writing loops or branching to another block of code.

PL/SQL Block structure-

DECLARE
Declarations of memory variables used later
BEGIN
SQL executable statements for manipulating table data.
EXCEPTIONS
SQL and/or PL.SQL code to handle errors.
END;

Displaying user Messages on the screen – Any programming tool requires a method through which messages can be displayed to the user.

dbms_output is a package that includes a number of procedure and functions that accumulate information in a buffer so that it can be retrieved later. These functions can also be used to display message to the user.

put_line: put a piece of information in the buffer followed by a end of line marker. It can also be used to display message to the user.

Setting the server output on:

SQL>SET SERVER OUTPUT ON:

Example: Write the following code in the PL/SQL block to display message to user
DBMS_OUTPUT.PUT_LINE('Display user message');

Conditional control in PL/SQL-

Syntax:IF <condition> THEN
<Action>
ELSEIF<condition>
<Action>
ELSE
<Action>
ENDIF;

The WHILE LOOP:

Syntax:WHILE <condition>
LOOP
<Action>
END LOOP;

The FOR LOOP statement:

Syntax:FOR variable IN [REVERSE] start—end
LOOP
<Action>
END LOOP;

The GOTO statement: The goto statement allows you to change the flow of control within a PL/SQL Block.

Q1. WAP in PL/SQL for addition of two numbers.
Q2. WAP in PL/SQL for addition of 1 to 100 numbers.
Q3. WAP in PL/SQL to check the given number is even or odd.
Q4. WAP in PL/SQL to inverse a number, eg. Number 5639 when inverted must be display output 9365.
Q5. WAP in PL/SQL for changing the price of product 'P00001' to 4000 if the price is less than 4000 in product_master table. The change is recorded in the old_price_table along with product_no and the date on which the price was changed last.

VIVA QUESTIONS

a) Explain ACID properties?
b) What do you mean by grant and revoke?

ASSIGNMENT NO - 11

OBJECT:- To implement the concept of Cursor.

THEORY AND CONCEPT

CURSOR– We have seen how oracle executes an SQL statement. Oracle DBA uses a work area for its internal processing. This work area is private to SQL's operation and is called a **cursor.**
The data that is stored in the cursor is called the **Active Data set.** The size of the cursor in memory is the size required to hold the number of rows in the Active Data Set.

Explicit Cursor- You can explicitly declare a cursor to process the rows individually. A cursor declared by the user is called **Explicit Cursor.** For Questionries that return more than one row, You must declare a cursor explicitly.
The data that is stored in the cursor is called the **Active Data set.** The size of the cursor in memory is the size required to hold the number of rows in the Active

Why use an Explicit Cursor- Cursor can be used when the user wants to process data one row at a time.

Explicit Cursor Management- The steps involved in declaring a cursor and manipulating data in the active data set are:-

- Declare a cursor that specifies the SQL select statement that you want to process.
- Open the Cursor.
- Fetch the data from the cursor one row at a time.
- Close the cursor.

Explicit Cursor Attributes- Oracle provides certain attributes/ cursor variables to control the execution of the cursor. Whenever any cursor (explicit or implicit) is opened and used Oracle creates a set of four system variables via which Oracle keeps track of the 'Current' status of the cursor. You

- Declare a cursor that specifies the SQL select statement that you want to process.
- Open the Cursor.
- Fetch the data from the cursor one row at a time.
- Close the cursor.

How to Declare the Cursor:-

The General Syntex to create any perticular cursor is as follows:-
Cursor <Cursorname> is Sql Statement;

How to Open the Cursor:-

The General Syntex to Open any perticular cursor is as follows:-
Open Cursorname;

Fetching a record From the Cursor:-

The fetch statement retrieves the rows from the active set to the variables one at a time. Each time a fetch is executed. The focus of the DBA cursor advances to the next row in the Active set.
One can make use of any loop structute(Loop-End Loop along with While,For) to fetch the records from the cursor into variable one row at a time.
The General Syntex to Fetch the records from the cursor is as follows:-
Fetch cursorname into variable1, variable2, _____

Closing a Cursor:-
The General Syntex to Close the cursor is as follows:-
Close <cursorname>;

VIVA QUESTIONS

a) Explain ACID properties?
b) What do you mean by grant and revoke?

ASSIGNMENT NO -12

OBJECT: - To implement the concept of Trigger.

THEORY AND CONCEPT

Database Triggers:-

Database triggers are procedures that are stored in the database and are implicitly executed(fired) when the contents of a table are changed.

Use of Database Triggers:-

Database triggers support Oracle to provide a highly customized database management system. Some of the uses to which the database triggers can be put to customize management information in Oracle are as follows:-

- A Trigger can permit DML statements against a table anly if they are issued, during regular bussiness hours or on predetermined weekdays.
- A trigger can also be used to keep an audit trail of a table along with the operation performed and the time on which the operation was performed.
- It can be used to prevent invalid transactions.
- Enforce complex security authorizations.

How to apply DataBase Triggers:-

A trigger has three basic parts:-

1.A triggering event or ststement.

2.A triger restriction

3.A trigger action.

Types of Triggers:-

Using the various options , four types of triggers can be created:-

1.Before Statement Trigger:- Before executing the triggering statement, the trigger action is executed.

2.Before Row Trigger:- Before modifying the each row affected by the triggering statement and before appropriate integrity constraints, the trigger is executed if the trigger restriction either evaluated to TRUE or was not included.'

3.After Ststement Trigger:- After executing the triggering statement and applying any deferred integrity canstraints, the trigger action is executed.

4.After row Trigger:- After modifying each row affected by the triggering statement and possibly applying appropriate integrity constraints, the trigger action is executed for the current row if the trigger restriction either evaluates to TRUE or was not included.

Syntex For Creating Trigger:-

The syntex for Creating the Trigger is as follows:-

Create or replace Trigger<Triggername> {Before,After} {Delete, Insert, Update } On <Tablename> For Each row when Condition

Declare

<Variable declarations>;

<Constant Declarations>;

Begin

<PL/SQL> Subprogram Body;

Exception

Exception Pl/SQL block;

End;

How to Delete a Trigger:-

The syntex for Deleting the Trigger is as follows:-

Drop Trigger <Triggername>;

VIVA QUESTIONS

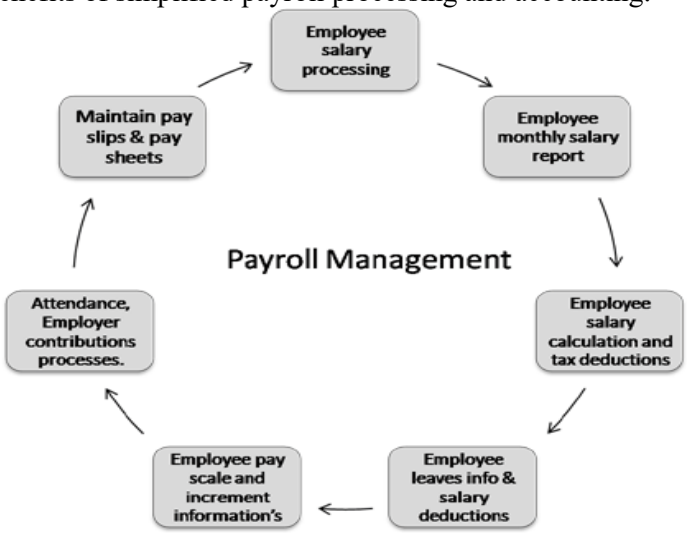
a) What is trigger?

b) What are the different types of trigger events?

ASSIGNMENT NO-13

OBJECT: - Design and implementation of payroll processing system

Payroll is a method of administering employees' salaries in an organization. The process consists of calculation of employee salaries and tax deductions, administrating employee benefits and payment of salaries. Payroll system can also be called as an accounts activity which commences the salary administration of employees in the organization. Our payroll management system is fully integrated with accounts and give's the benefits of simplified payroll processing and accounting.



Entities are:

COMPANY

CNAME	CID

BRANCH

BNAME	BRANCH#

EMPLOYEE

ENAME	EID	DOB	PHONENO	EDESIGNATION

EMPLOYER

NAME	ID	DESIGNATION	DEPARTMENT

SALARY

MEDICAL ALLOWANCE	BASIC	HRA	TA	DA	INCENTIVE

Relationships:

HAS

CID	BRANCH#

HEADED BY

BRANCH#	ID

EMPLOYS

EID	ID

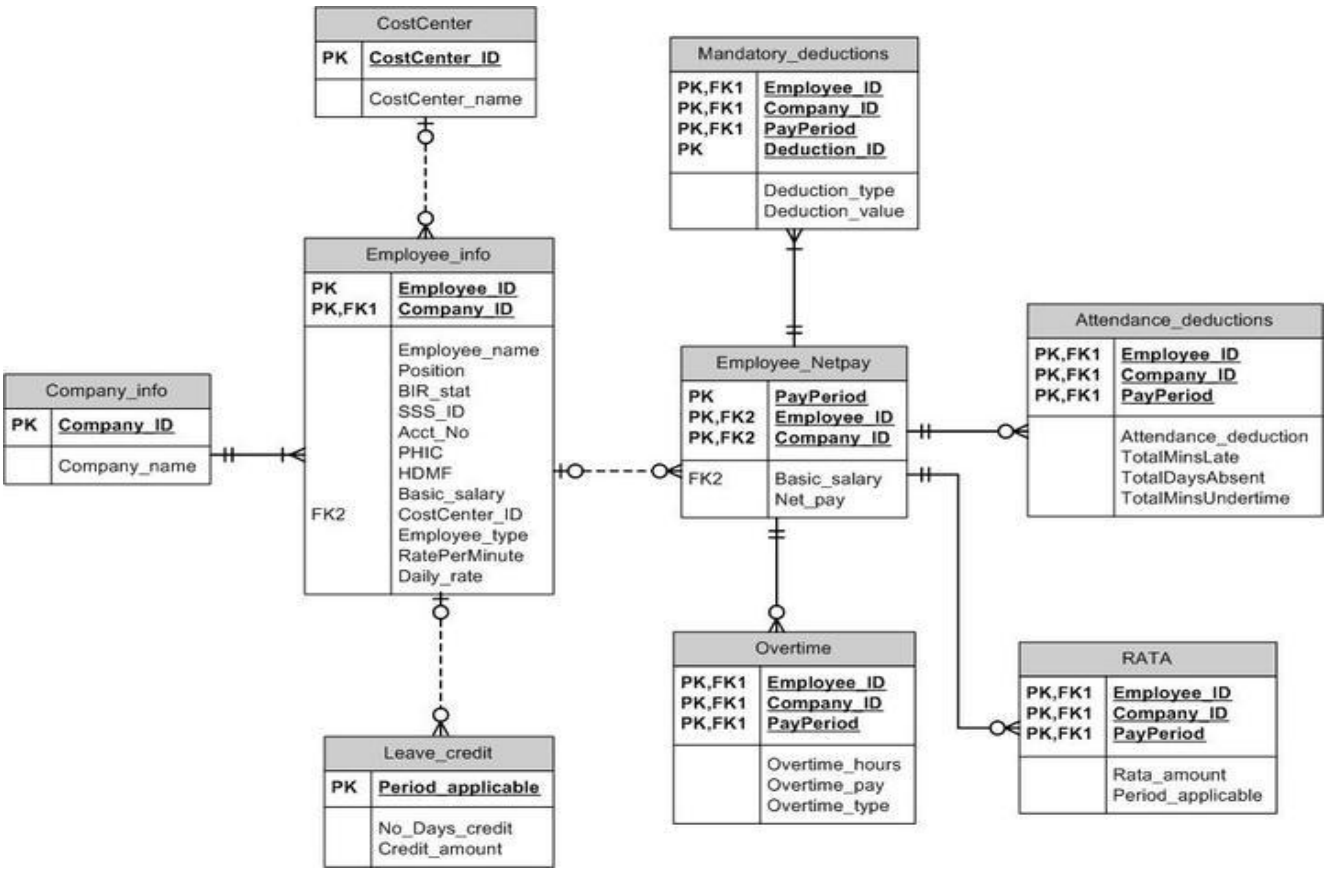
GETS

EID	BASIC

PAYS

ID	BASIC

Entity relationship diagram:



VIVA QUESTION

Q1: what is payroll system?
Q2: what is extended ER diagram?

ASSIGNMENT NO-14

OBJECT:- Design and implementation of Library Information System

Library Management System is an application which refers to library systems which are generally small or medium in size. It is used by librarian to manage the library using a computerized system where he/she can record various transactions like issue of books, return of books, addition of new books, addition of new students etc.

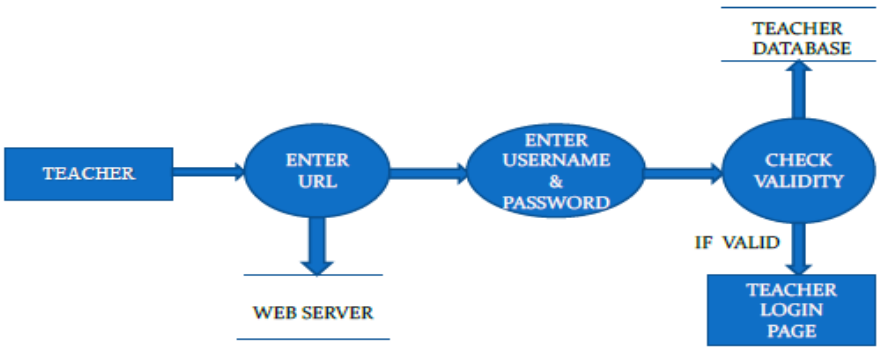
Books and student maintenance modules are also included in this system which would keep track of the students using the library and also a detailed description about the books a library contains. With this computerized system there will be no loss of book record or member record which generally happens when a non computerized system is used.

Aims and objectives:

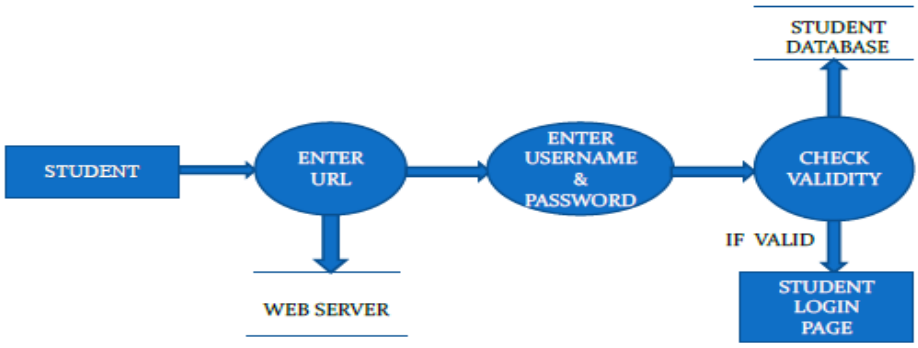
The aims and objectives are as follows:

- Online book issue
- Request column for librarian for providing new books
- A separate column for digital library
- Student login page where student can find books issued by him/her and date of return.
- A search column to search availability of books
- A teacher login page where teacher can add any events being organized in the college and important suggestions regarding books.
- Online notice board about the workshop.

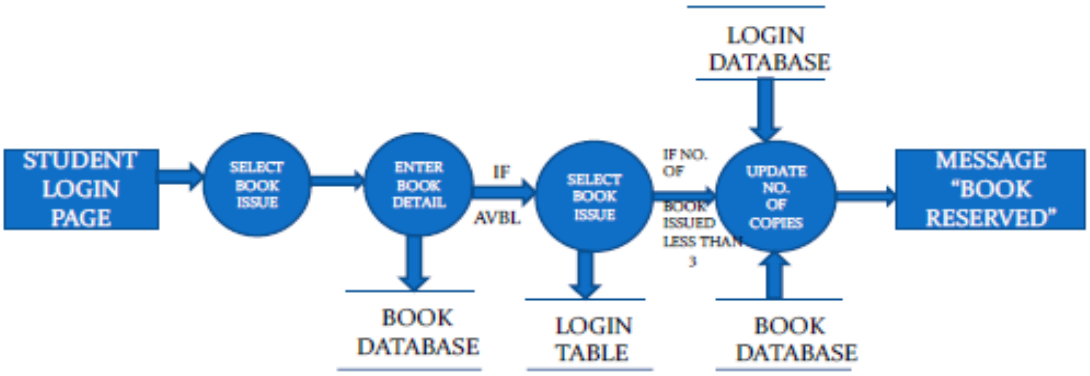
DATA FLOW DIAGRAM FOR TEACHER LOGIN



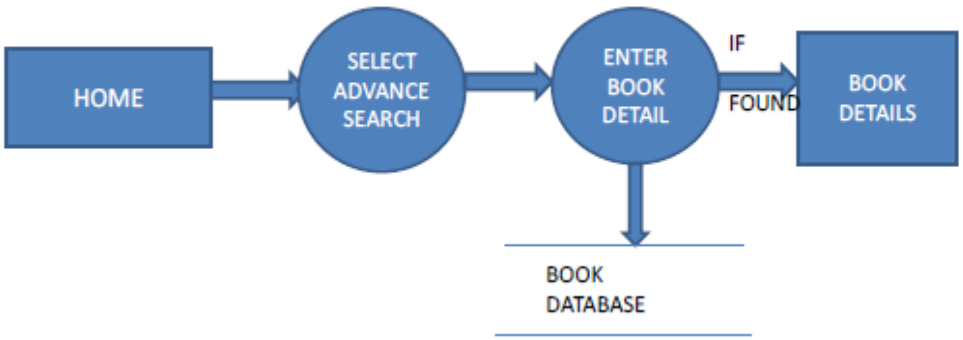
DATA FLOW DIAGRAM FOR STUDENT LOGIN



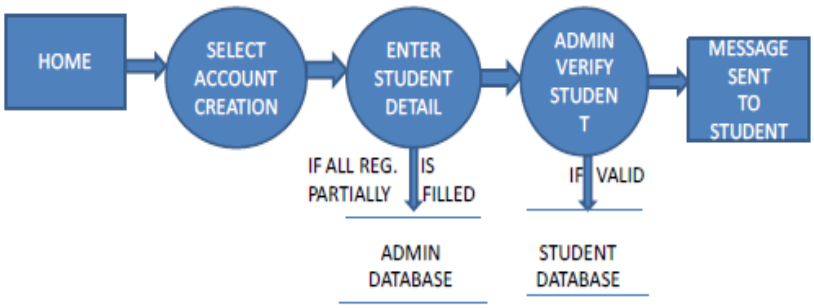
DATA FLOW DIAGRAM FOR BOOK ISSUE



DATA FLOW DIAGRAM FOR BOOK SEARCH



DATA FLOW DIAGRAM FOR ACCOUNT CREATION



VIVA QUESTION

Q1: What do you mean by Entity type?
Q2: Define the "integrity rules".

ASSIGNMENT NO- 15

OBJECT: -. Design and implementation of Student Information System

System modules: This software application is designed to register students from admission point; generate matriculation number for each registered student at the close of registration. Subsequently, student details can be modified with the appropriate access rights. The system comprises of the following modules:

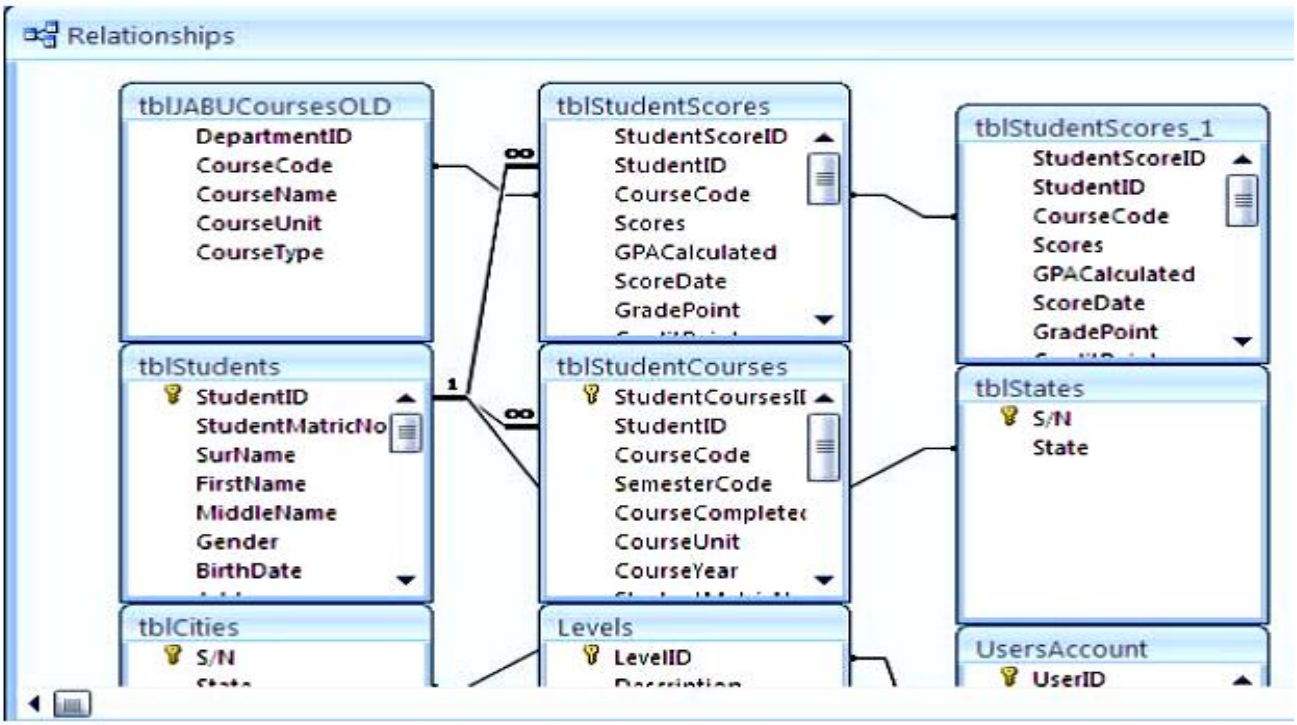
- Student Registration
- Course registration
- Attendance Monitoring
- Examination Records
- Enquiries
- Reports
- Systems Administration
- Help and Backup

Major factors in database design: In principle, there are only a few things that can be done with a database one can: view the data, find some data of interest, modify the data, add some data and delete some data. To achieve these, three major factors need to be considered in any database system:

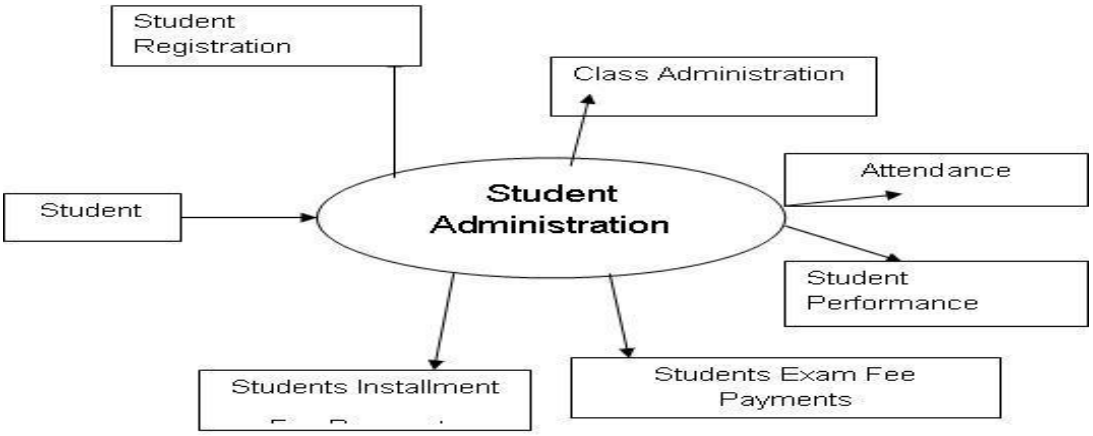
- creating the structure of the database
- entering data
- retrieving data

Relational structure: A relational database is so named not because it relates one file to another; but because it uses a mathematical construct called a relation. A relation is nothing more than a table containing rows (records) and columns (fields). The referral of one table to another via a common field is called a relation. Such groupings of tables are called relational databases (Microsoft Corporation, 1999). Relations can be linked together on the basis of a common field. Relational data structures are popular today because they are simple and adaptable. A student table may be related to student courses and scores tables' relations as shown in Figure by the student's matriculation number which is the common field joining the two relations. If a program were processing the course relation row by row, it could look up the student name in the student relation in order to produce a report. Another program could process the student relation and look up all the scores for each student.

Sample student tables relationship structure:

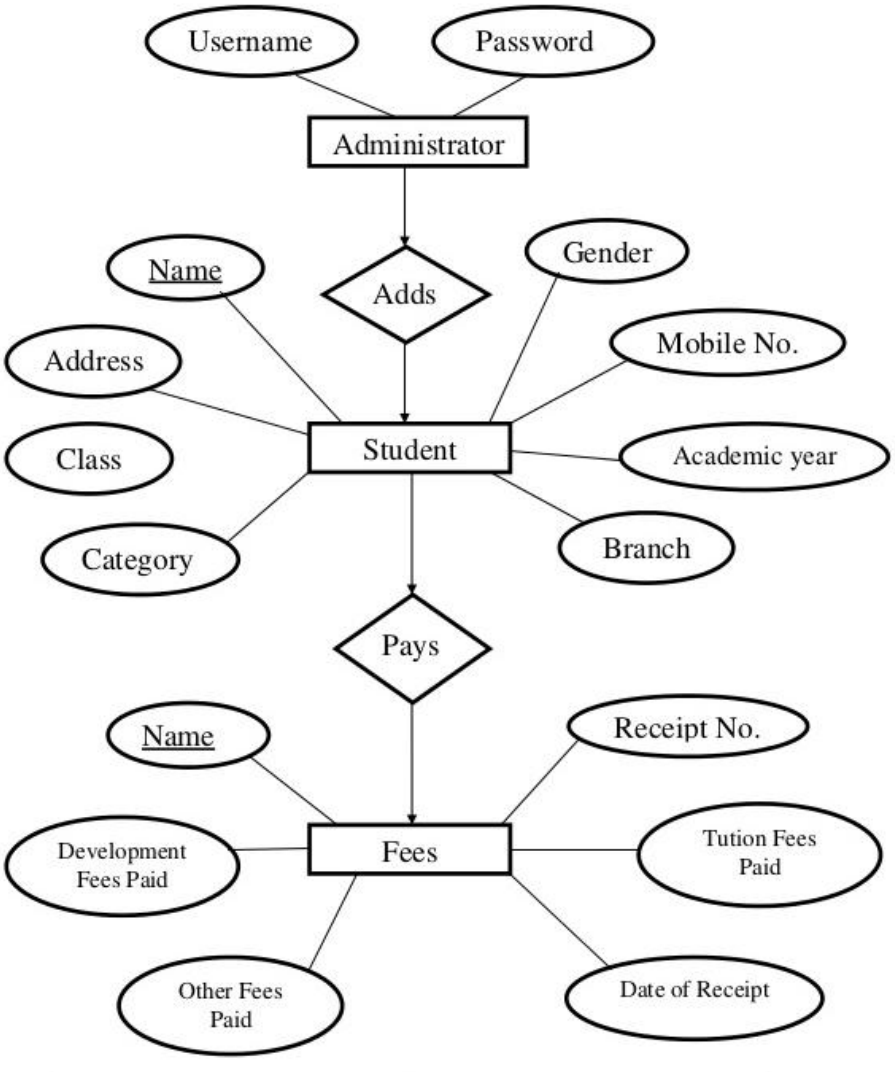


DFD for student information system:



ER Diagram:

6. ER Diagram



VIVA QUESTION

Q1: Enlist the various relationships of database.
Q2: what is the relationship between weak entity and identical relations

KCS-552: COMPILER DESIGN LAB

COURSE OUTCOMES:

Course Outcome (CO)		Bloom's Knowledge Level (KL)
At the end of course , the student will be able to:		
CO 1	Identify patterns, tokens & regular expressions for lexical analysis.	K_2, K_4
CO 2	Design Lexical analyser for given language using C and LEX /YACC tools	K_3, K_5
CO 3	Design and analyze top down and bottom up parsers.	K_4, K_5
CO 4	Generate the intermediate code	K_4, K_5
CO 5	Generate machine code from the intermediate code forms	K_3, K_4

ASSIGNMENT NO – 1

OBJECTIVE: Design and implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and new lines

EXPLANATION:- Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language

PROGRAM LOGIC:

1. Read the input Expression
2. Check whether input is alphabet or digits then store it as identifier
3. If the input is is operator store it as symbol
4. Check the input for keywords 1.4 PROCEDURE: Go to debug -> run or press CTRL + F9 to run the program

PROBLEMS

1. Write a program to recognize identifiers.
2. Write a program to recognize constants.

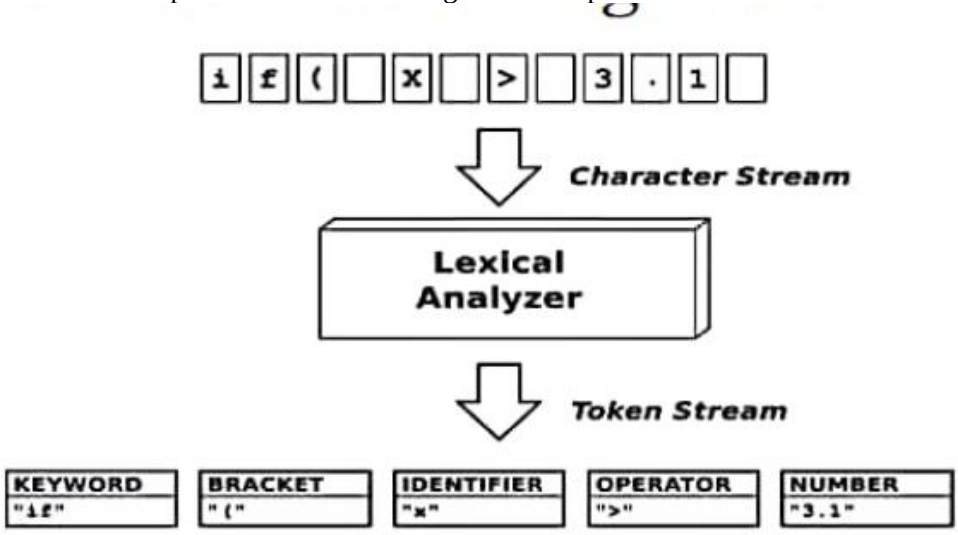
VIVA QUESTIONS

1. What is lexical analyzer?
2. Which compiler is used for lexical analyzer?

ASSIGNMENT NO – 2

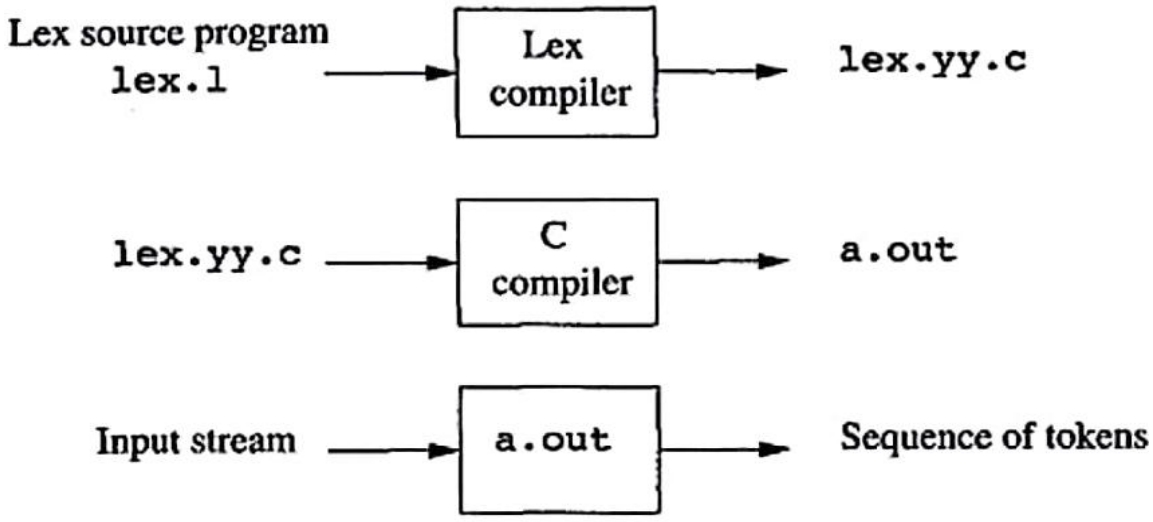
OBJECTIVE Implementation of Lexical Analyzer using Lex Tool

EXPLANATION: Lex Tool: It is a tool which generate **lexical analyser**Lexical analyser is a first phase of compiler which take input as source code and generate output as tokens



The input notation for the lex tool is referred to as the lex language and the tool itself is the **lex compiler.**

The lex compiler transforms the input patterns into transition diagram and generate code,in a file called lex.yy.c



PROBLEMS:

Write a program that defines auxiliary definitions and translation rules of C tokens?

VIVA QUESTIONS

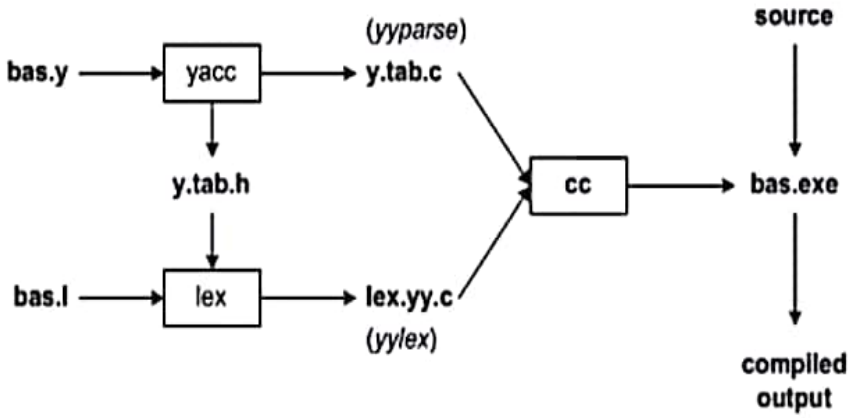
1. What is the output of Lexical analyzer?
2. What is LEX source Program?

ASSIGNMENT NO – 3

OBJECTIVE: Generate YACC specification for a few syntactic categories.

EXPLANATION: YACC stands for **Yet Another Compiler Compiler**.
YACC provides a tool to produce a parser for a given grammar. YACC is a program designed to compile a LALR (1) grammar. It is used to produce the source code of the syntactic analyzer of the language produced by LALR (1) grammar.The input of YACC is the rule or grammar and the output is a C program.

Basic Structure of YACC Compiler:



File Format of YACC:

Definitions
%%
Rules
%%
Supplementary Code

ALGORITHM:

Step1: Start the program.
Step2: Reading an expression .
Step3: Checking the validating of the given expression according to the rule using yacc.
Step4: Using expression rule print the result of the given values
Step5: Stop the program.

PROBLEMS:

a) Program to recognize a valid arithmetic expression that uses operator +, – , * and /.
b) Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.
c) Implementation of Calculator using LEX and YACC
d) Convert the BNF rules into YACC form and write code to generate abstract syntax tree

VIVA QUESTIONS:

1. What is the difference between yylex() and scanf().
2. What does the last section contain?

ASSIGNMENT NO – 4

OBJECTIVE: Write program to find ε – closure of all states of any given NFA with ε transition.

EXPLANATION:-NFAs with ε moves: Here one allows transitions labeled with ε, the empty string. This means that you can follow such an arrow without consuming any input symbols. The NFA on the left shows how useful epsilon moves are in recognizing regular expressions with the example a*b*c*: which is "zero of more as, followed by zero or more bs, followed by zero of more cs". (The diagram uses "eps" for ε.)
On the right is a DFA that recognizes the same language. Notice some new things about this DFA: three states are terminal ones, including the start state. There is an "error" state, with label err, and it is not terminal. If you transition to that state, you can't get out and you can't accept. This state could also have been labeled with the symbol for the empty set: Ø
Algorithm: ε-closure
Input: an NFA with ε-moves **N**, a set **P** of states.
Output: ε-closure(P), which is **P** along with all states
reachable from **P** by a sequence of ε-moves.
Method: use stack **S** to carry out a depth-first search

// **black** means in **ε-closure(P)**
color all states **white**
color all states of **P black**
push all states of **P** onto **S**
while (**S** not empty) {
  u = pop(S)
  for (each edge **(u,v)** labeled **ε**) {
    if (**v** has color **white**) {
      color **v black**
      push v
    }
  }
}
ε-closure(P) = black states
The subset algorithm for an NFA with ε-moves is the same as the one for ordinary NFAs, except that each time a subset is constructed, it must have the ε-closure algorithm (at the left) applied to it.
In the above case, the start state of the NFA is 0, so the start state of the DFA is not {0} but ε-closure({0}) = {0,1,2}.
Transitions on b from 0, 1, or 2 only go to 1. Then ε-closure({1}) = {1,2}..
There are no transitions on a from 1 or 2, so the set in this case is Ø, the empty set. The empty set is a subset that serves as an error state.

PROBLEMS: Write a program in C to find epsilon(ε) closure of a state to given by user.

VIVA QUESTIONS:

1. What do you mean by epsilon(ε) closure?
2. Both NFA and e-NFA recognize exactly the same languages?justify your answer.

ASSIGNMENT NO – 5

OBJECTIVE: Write program to convert NFA with ε transition to NFA without ε transition.

EXPLANATION:- Non-deterministic Finite Automata (NFA) is a finite automata having zero, one or more than one moves from a given state on a given input symbol. Epsilon NFA is the NFA which contains epsilon move(s)/Null move(s). To remove the epsilon move/Null move from epsilon-NFA and to convert it into NFA, we follow the steps mentioned below.

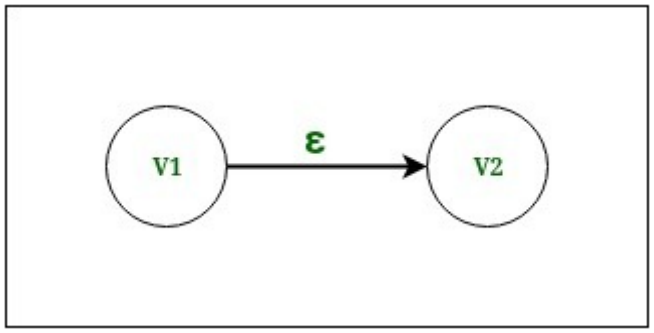
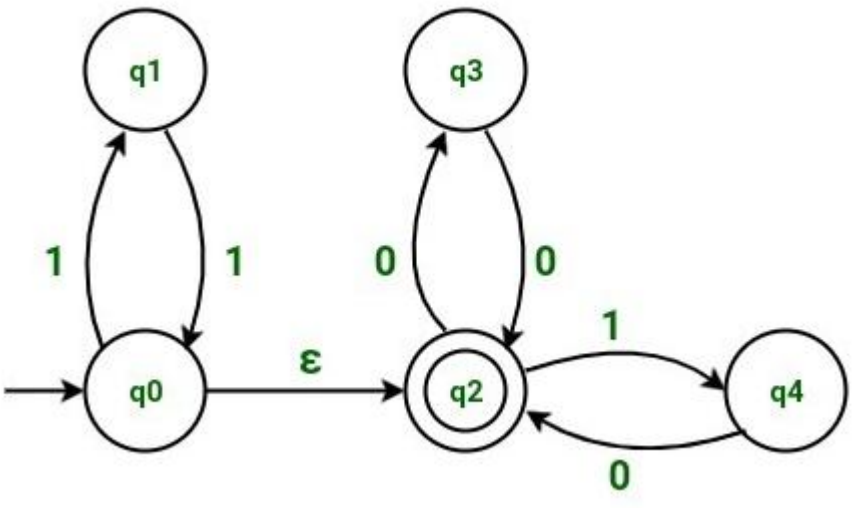


Figure – Vertex v1 and Vertex v2 having an epsilon move

Step-1:
Consider the two vertexes having the epsilon move. Here in *Fig.1* we have vertex v1 and vertex v2 having epsilon move from v1 to v2.
Step-2:
Now find all the moves to any other vertex that start from vertex v2 *(other than the epsilon move that is considering).*
After finding the moves, duplicate all the moves that start from vertex v2, with the same input to start from vertex v1 and remove the epsilon move from vertex v1 to vertex v2.
Step-3:
See that if the vertex v1 is a start state or not. If vertex v1 is a start state, then we will also make vertex v2 as a start state. If vertex v1 is not a start state, then there will not be any change.
Step-4:
See that if the vertex v2 is a final state or not.
If vertex v2 is a final state, then we will also make vertex v1 as a final state.
If vertex v2 is not a final state, then there will not be any change.
Repeat the steps*(from step 1 to step 4)* until all the epsilon moves are removed from the NFA.
Now, to explain this conversion, let us take an example.
Example: Convert epsilon-NFA to NFA.
Consider the example having states q0, q1, q2, q3, and q4.



In the above example, we have 5 states named as q0, q1, q2, q3 and q4. Initially, we have q0 as start state and q2 as final state. We have q1, q3 and q4 as intermediate states.
Transition table for the above NFA is:

STATES/INPUT	INPUT 0	INPUT 1	INPUT EPSILON
q0	–	q1	q2
q1	–	q0	–
q2	q3	q4	–
q3	q2	–	–
q4	q2	–	–

According to the transition table above,

- state q0 on getting input 1 goes to state q1.
- State q0 on getting input as a null move ***(i.e. an epsilon move)*** goes to state q2.
- State q1 on getting input 1 goes to state q0.
- Similarly, state q2 on getting input 0 goes to state q3, state q2 on getting input 1 goes to state q4.
- Similarly, state q3 on getting input 0 goes to state q2.
- Similarly, state q4 on getting input 0 goes to state q2.

We can see that we have an epsilon move from state q0 to state q2, which is to be removed.
To remove epsilon move from state q0 to state q1, we will follow the steps mentioned below.
Step-1:
Considering the epsilon move from state q0 to state q2. Consider the state q0 as vertex v1 and state q2 as vertex v2.

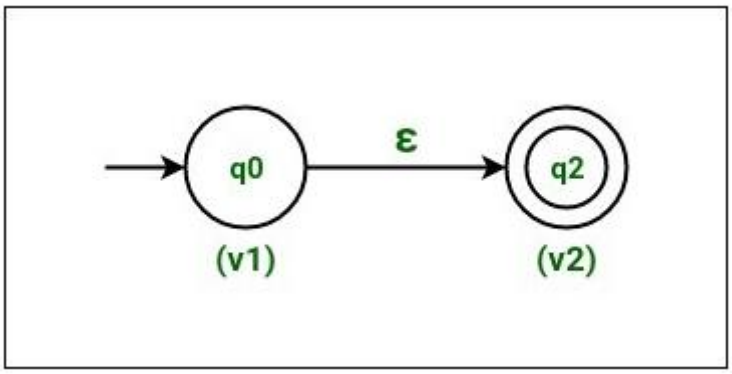


Figure – State q0 as vertex v1 and state q2 as vertex v2

Step-2:
Now find all the moves that starts from vertex v2 *(i.e. state q2)*.
After finding the moves, duplicate all the moves that start from vertex v2 *(i.e state q2)* with the same input to start from vertex v1 *(i.e. state q0)* and remove the epsilon move from vertex v1 *(i.e. state q0)* to vertex v2 *(i.e. state q2)*.
Since state q2 on getting input 0 goes to state q3. Hence on duplicating the move, we will have state q0 on getting input 0 also to go to state q3.
Similarly state q2 on getting input 1 goes to state q4. Hence on duplicating the move, we will have state q0 on getting input 1 also to go to state q4.

So, NFA after duplicating the moves is:

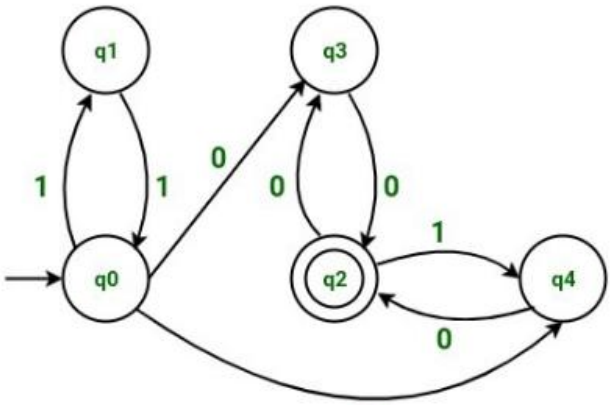


Figure – NFA on duplicating moves

Step-3:

Since vertex v1 *(i.e. state q0)* is a start state. Hence we will also make vertex v2 *(i.e. state q2)* as a start state.
Note that state q2 will also remain as a final state as we had initially.
NFA after making state q2 also as a start state is:

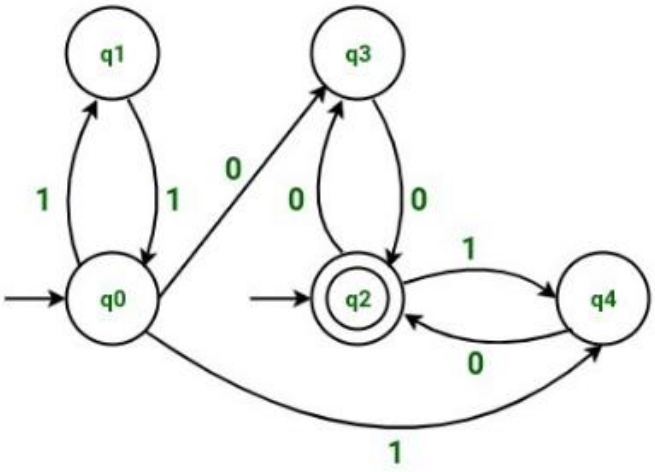
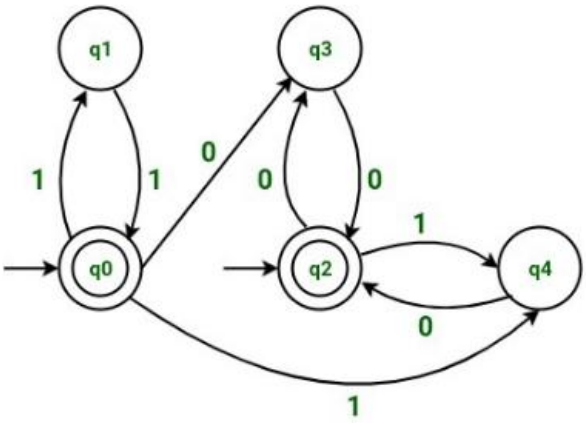


Figure – NFA after making state q2 as a start state

Step-4:

Since vertex v2 *(i.e. state q2)* is a final state. Hence we will also make vertex v1 *(i.e. state q0)* as a final state.
Note that state q0 will also remain as a start state as we had initially.
After making state q0 also as a final state, the resulting NFA is:



PROBLEMS:-Write a program in C to print convert NFA with ε transition to NFA without ε transition

VIVA QUESTIONS

1. What do you mean by NFA?
2. What do you mean by NFA without ε transition?

ASSIGNMENT NO – 6

OBJECTIVE :- Conversion from NFA to DFA

EXPLANATION:- A Deterministic finite automaton (DFA) can be seen as a special kind of NFA, in which for each state and alphabet, the transition function has exactly one state. Thus clearly every formal language that can be recognized by a DFA can be recognized by an NFA.

ALGORITHM

Input − An NFA
Output − An equivalent DFA
Step 1 − Create state table from the given NFA.
Step 2 − Create a blank state table under possible input alphabets for the equivalent DFA.
Step 3 − Mark the start state of the DFA by q0 (Same as the NDFA).
Step 4 − Find out the combination of States $\{Q_0, Q_1, \ldots, Q_n\}$ for each possible input alphabet.
Step 5 − Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.
Step 6 − The states which contain any of the final states of the NFA are the final states of the equivalent DFA.

PROBLEMS

Que. Write a program in C language to NFA to DFA.

VIVA QUESTIONS

1. Is every DFA is NFA or not? Give valid reason for your answer.
2. The power of NFA and DFA is same or differ, explain it with reason.

ASSIGNMENT NO – 7

OBJECTIVE: - To minimize any given DFA.

EXPLANATION:- DFA minimization stands for converting a given DFA to its equivalent DFA with minimum number of states.

Minimization of DFA

Suppose there is a DFA $D < Q, \Sigma, q0, \delta, F >$ which recognizes a language L. Then the minimized DFA $D < Q', \Sigma, q0, \delta', F' >$ can be constructed for language L as:

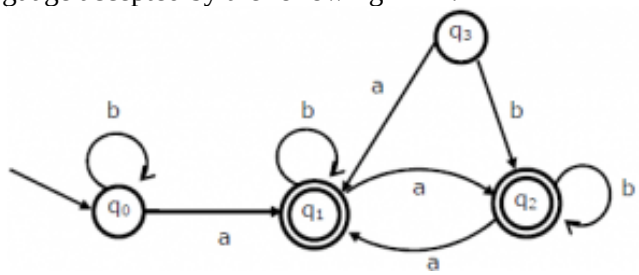
Step 1: We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P_0.
Step 2: Initialize k = 1
Step 3: Find P_k by partitioning the different sets of P_{k-1}. In each set of P_{k-1}, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in P_k.
Step 4: Stop when $P_k = P_{k-1}$ (No change in partition)
Step 5: All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in P_k.

PROBLEMS

Que. Write a program in C language to minimize any given DFA

VIVA QUESTIONS

1. What is the language accepted by the following DFA?



2. What is the reason by which the minimization of DFA required ?

ASSIGNMENT NO -8

OBJECTIVE:- To develop an operator precedence parser for a given language

EXPLANATION:- Bottom-up parsers for a large class of context-free grammars can be easily developed using operator grammars.

Operator grammars have the property that no production right side is empty or has two adjacent non terminals. This property enables the implementation of efficient operator-precedence parsers. These parser rely on the following three precedence relations:

Operator Precedence Parsing Algorithm

Initialize: Set ip to point to the first symbol of w$

Repeat: Let X be the top stack symbol, and a the symbol pointed to by ip

if $ is on the top of the stack and ip points to $ **then return**

else

Let a be the top terminal on the stack, and b the symbol pointed to by ip

if a <· b **or** a =· b **then**

push b onto the stack

advance ip to the next input symbol

else if a ·>b **then**

repeat

pop the stack

until the top stack terminal is related by <·

to the terminal most recently popped

else error()

end

Problems

Ques: WAP in C to print an operator precedence table for given operator precedence grammar:

E-> E+E
E-> E*E
E-> id

VIVA QUESTIONS

1. What is operator precedence parser? List the advantages and disadvantages.
2. Differentiate tokens, patterns, and lexeme.

ASSIGNMENT NO -9

OBJECTIVE:- To find First and Follow of any given grammar.

Algorithm to compute FIRST of any non-terminal element:

1. If X is terminal, FIRST(X) = {X}.
2. If $X \rightarrow \varepsilon$ is a production, then add ε to FIRST(X).
3. If X is a non-terminal, and $X \rightarrow Y_1\ Y_2 \ldots Y_k$ is a production, and ε is in all of $FIRST(Y_1)$, …, $FIRST(Y_k)$, then add ε to FIRST(X).
4. If X is a non-terminal, and $X \rightarrow Y_1\ Y_2 \ldots Y_k$ is a production, then add a to FIRST(X) if for some i, a is in $FIRST(Y_i)$, and ε is in all of $FIRST(Y_1)$, …, $FIRST(Y_{i-1})$.

Algorithms to compute FOLLOW of any non-terminal element:

- If $ is the input end-marker, and S is the start symbol, $ ∈ FOLLOW(S).
- If there is a production, $A \rightarrow \alpha B\beta$, then $(FIRST(\beta) - \varepsilon) \subseteq FOLLOW(B)$.
- If there is a production, $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B\beta$, where $\varepsilon \in FIRST(\beta)$, then $FOLLOW(A) \subseteq FOLLOW(B)$.

PROBLEMS

Ques.WAP to compute the FIRST() and FOLLOW of a grammar input by the user.

VIVA QUESTIONS

1. How the FIRST( ) of given Non Terminal is calculated.
2. How the FOLLOW( ) of given Non Terminal is calculated.

ASSIGNMENT NO -10

OBJECTIVE:- To implement recursive descent parser for an expression.

EXPLANATION:
Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammar associated with it (if not left factored) cannot avoid back-tracking. A form of recursive-descent parsing that does not require any back-tracking is known as **predictive parsing**.

PROBLEMS
Ques.WAP in C to implement Recursive descent parser an expression

VIVA QUESTIONS
1. Issues to be considered while applying the technique for code optimization.
2. What is a basic block?

ASSIGNMENT NO – 11

OBJECTIVE :- Implementation of shift –reduce parsing using 'C' .

EXPLANATION:- A Convenient way to implement a shift reduce parser is to use a stack to held grammar symbols and an input buffer to held the string 'W' to be parsed. We use the $ to mark the bottom of the stack and also the right end of the input initially the stack is empty and the string 'W' is on the input as follows.

	STACK / $	IMPUT / W$
Right – Sentetial Form	Handle	Ruducing Production
id1+id2+id3	id1	$->id
$+id2*id3	id2	$->id
$+$*id3	id3	$->id
$+$*$	$*$	$->$*$
$+$	$+$	$->$+$

PROBLEMS
Que. Write a program in C language to implement Shift-Reduce Parser.

VIVA QUESTIONS
1. How the top down parsing is carried out. What are the problems with top down parsing?
2. Mention the shift-reduce parsing techniques.

ASSIGNMENT NO – 12

OBJECTIVE :- To perform loop unrolling.

EXPLANATION:- Loop unrolling is a loop transformation technique that helps to optimize the execution time of a program. We basically remove or reduce iterations. Loop unrolling increases the program's speed by eliminating loop control instruction and loop test instructions.

Example:

```
while(i<10)                    while(i<10)
 {                              {
  x[i]=0;          →             x[i]=0;
   i++;                          i++;
   }                            x[i]=0;
                                 i++;
                                 }
```

In the first code segment the loop is executed 10 times while in second code the loop is executed 5 times.

PROBLEMS

Que. Write a program in C language to perform loop unrolling.

VIVA QUESTIONS

1. What are the advantages of loop unrolling?.
2. What are the disadvantages of loop unrolling?.

ASSIGNMENT NO – 13

OBJECTIVE :- To perform constant propagation.

EXPLANATION:-Constant propagation is the process of substituting the values of known constants in expressions at compile time.

Consider the following pseudo code:

```
int x = 14;
int y = 7 - x / 2;
return y * (28 / x + 2);
```

Propagating x yields:

```
int x = 14;
int y = 7 - 14 / 2;
return y * (28 / 14 + 2);
```

PROBLEMS

Que. Write a program in C language to perform constant propagation.

VIVA QUESTIONS

1. What is constant propagation?
2. Apply constant propagation on the following code:

```
int a = 30;
int b = 9 - (a / 5);
int c;

c = b * 4;
if (c > 10) {
   c = c - 10;
}
return c * (60 / a);
```

ASSIGNMENT NO – 14

OBJECTIVE:- Implementation of three address code using triples in C[Intermediate Code Generation].

EXPLANATION:- To avoid the entering temporary names into symbol table be might refer to a temporary value by the position of statement that computes it. It consist three fields op, arg1, arg2.
A:=b*-c+b*-c

	op	arg1	arg2
(0)	Uniminus	C	
(1)	*	B	(0)
(2)	uniminus	C	
(3)	*	B	(2)
(4)	+	(1)	(3)
(5)	assign	A	(4)

PROBLEM

Que. Implement the following expressions in triples using C language.

-(a+b)*(c+d)-(a+b+c)

VIVA QUESTIONS

1. What do you mean by machine dependent and machine independent optimization?
2. Mention the basic issues in parsing.

ASSIGNMENT NO – 15

OBJECTIVE :- To implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

EXPLANATION:- A compiler is a computer program that implements a programming language specification to "translate" programs, usually as a set of files which constitute the source code written in source language, into their equivalent machine readable instructions(the target language, often having a binary form known as object code). This translation process is called compilation. **BACK END:**

- Some local optimization
- Register allocation
- Peep-hole optimization
- Code generation
- Instruction scheduling

The main phases of the back end include the following:

- Analysis: This is the gathering of program information from the intermediate representation derived from the input; data-flow analysis is used to build use-define chains, together with dependence analysis, alias analysis, pointer analysis, escape analysis etc.
- Optimization: The intermediate language representation is transformed into functionally equivalent but faster (or smaller) forms. Popular optimizations are expansion, dead, constant, propagation, loop transformation, register allocation and even automatic parallelization.
- Code generation: The transformed language is translated into the output language, usually the native machine language of the system. This involves resource and storage decisions, such as deciding which variables to fit into registers and memory and the selection and scheduling of appropriate machine instructions along with their associated modes. Debug data may also need to be generated to facilitate debugging.

ALGORITHM:

1. Start the program
2. Open the source file and store the contents as quadruples.
3. Check for operators, in quadruples, if it is an arithmetic operator generator it or if assignment operator generates it, else perform unary minus on register C.
4. Write the generated code into output definition of the file in outp.c
5. Print the output.
6. Stop the program.

PROBLEMS

Que. Write a program in C language to implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

VIVA QUESTIONS

1. What is the back-end of a compiler? Discuss about it.
2. What is the front-end of a compiler? Discuss about it.

KCS-553: DESIGN AND ANALYSIS OF ALGORITHM LAB

Design and Analysis of Algorithm Lab (KCS-553)		
Course Outcome (CO)		**Bloom's Knowledge Level (KL)**
At the end of course , the student will be able to:		
CO 1	Implement algorithm to solve problems by iterative approach.	K_2, K_4
CO 2	Implement algorithm to solve problems by divide and conquer approach	K_3, K_5
CO 3	Implement algorithm to solve problems by Greedy algorithm approach.	K_4, K_5
CO 4	Implement algorithm to solve problems by Dynamic programming, backtracking, branch and bound approach.	K_4, K_5
CO 5	Implement algorithm to solve problems by branch and bound approach.	K_3, K_4

CO-PO MAPPING:

Course Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	3										
CO2	**1**	**3**										
CO3	**1**	**1**	2									
CO4	**1**	**3**	3	**3**	**2**							
CO5	**1**	**3**	3	**2**								

ASSIGNMENT NO – 1

1. write a program for implementing sequential search technique.

Theory and Concept

Sequential Search

The simplest algorithm to search a dictionary for a given key is to test successively against each element. This works correctly regardless of the order of the elements in the list. However, in the worst case all elements might have to be tested.

Linear Search (Array A, Value x)
Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
Asymptotic Notation
Since this algorithm compares every element to find the required one its complexity in all the cases remains order of n i.e. **O(n)** (where n is number of elements in the list) and its expected cost is also proportional to n provided that searching and comparing cost of all the elements is same.
Worst case time complexity – O(n)
Average case time complexity – O(n)

VIVA QUESTION

A) What is the best case, average case and worst case time complexity of Sequential search ?
B) The algorithm for sequential search is…
   a) for(i=0;i<n;i++)
      if (key != k (i))
      return (i);
      return (-1);
   b) for(i=0;i<n;i++)
      if (key == k (i))
      return (i);
      return (-1);\
   c) None of these
   d) All of these

ASSIGNMENT 2

2. write a program for implementing binary search technique.

Binary Search

In computer science, a **binary search tree** (**BST**) is a binary tree data structure which has the following properties:

- each node (item in the tree) has a value;
- a total order (linear order) is defined on these values;
- the left subtree of a node contains only values less than the node's value;
- the right subtree of a node contains only values greater than or equal to the node's value.

The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient.

Binary search trees can choose to allow or disallow duplicate values, depending on the implementation.

Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets, multisets, and associative arrays

Insertion

Insertion begins as a search would begin; if the root is not equal to the value, we search the left or right subtrees as before. Eventually, we will reach an external node and add the value as its right or left child, depending on the node's value. In other words, we examine the root and recursively insert the new node to the left subtree if the new value is less than the root, or the right subtree if the new value is greater than or equal to the root.

Here's how a typical binary search tree insertion might be performed in C++:

```
/* Inserts the node pointed to by "newNode" into the subtree rooted at "treeNode" */
 void InsertNode(Node *&treeNode, Node *newNode)
 {     if (treeNode == NULL)
       treeNode = newNode;
      else if (newNode->key < treeNode->key)
       InsertNode(treeNode->left, newNode);
      else
       InsertNode(treeNode->right, newNode);    }
```

The above "destructive" procedural variant modifies the tree in place. It uses only constant space, but the previous version of the tree is lost. Alternatively, as in the following Python example, we can reconstruct all ancestors of the inserted node; any reference to the original tree root remains valid, making the tree a persistent data structure:

```
def binary_tree_insert(node, key, value):
    if node is None:
        return TreeNode(None, key, value, None)
     if key == node.key:
        return TreeNode(node.left, key, value, node.right)
    if key < node.key:
        return TreeNode(binary_tree_insert(node.left, key, value), node.key, node.value, node.right)
    else:
        return TreeNode(node.left, node.key, node.value, binary_tree_insert(node.right, key, value))
```

The part that is rebuilt uses $\Theta(\log n)$ space in the average case and $\Omega(n)$ in the worst case.

In either version, this operation requires time proportional to the height of the tree in the worst case, which is $O(\log n)$ time in the average case over all trees, but $\Omega(n)$ time in the worst case.

Another way to explain insertion is that in order to insert a new node in the tree, its value is first compared with the value of the root. If its value is less than the root's, it is then compared with the value of the root's left child. If its value is greater, it is compared with the root's right child. This process continues, until the new node is compared with a leaf node, and then it is added as this node's right or left child, depending on its value.

There are other ways of inserting nodes into a binary tree, but this is the only way of inserting nodes at the leaves and at the same time preserving the BST structure.

Deletion

There are several cases to be considered:

- **Deleting a leaf:** Deleting a node with no children is easy, as we can simply remove it from the tree.
- **Deleting a node with one child:** Delete it and replace it with its child.

- **Deleting a node with two children:** Suppose the node to be deleted is called N. We replace the value of N with either its in-order successor (the left-most child of the right subtree) or the in-order predecessor (the right-most child of the left subtree).

Once we find either the in-order successor or predecessor, swap it with N, and then delete it. Since both the successor and the predecessor must have fewer than two children, either one can be deleted using the previous two cases. A good implementation avoids consistently using one of these nodes, however, because this can unbalance the tree.

Here is C++ sample code for a destructive version of deletion. (We assume the node to be deleted has already been located using search.)

```
void DeleteNode(Node * & node) {
    if (node->left == NULL) {       Node *temp = node;
                                    node = node->right;
                                    delete temp;
    } else if (node->right == NULL) {
       Node *temp = node;
       node = node->left;
       delete temp;
    } else {        // In-order predecessor (rightmost child of left subtree)
       // Node has two children - get max of left subtree
       Node **temp = &node->left; // get left node of the original node
        // find the rightmost child of the subtree of the left node
       while ((*temp)->right != NULL) {
          temp = &(*temp)->right;        }
        // copy the value from the in-order predecessor to the original node
       node->value = (*temp)->value;
        // then delete the predecessor
       DeleteNode(*temp);     } }
```

Traversal

Once the binary search tree has been created, its elements can be retrieved in order by recursively traversing the left subtree of the root node, accessing the node itself, then recursively traversing the right subtree of the node, continuing this pattern with each node in the tree as it's recursively accessed. The tree may also be traversed in pre-order or post-order traversals.

```
def traverse_binary_tree(treenode):
    if treenode is None: return
    left, nodevalue, right = treenode
    traverse_binary_tree(left)
    visit(nodevalue)
    traverse_binary_tree(right)
```

Traversal requires Ω(n) time, since it must visit every node. This algorithm is also O(n), and so it is asymptotically optimal.

VIVA QUESTION

A) Define threaded binary tree. Explain its common uses.

B) What is the worst case time complexity for search, insert and delete operations in a general Binary Search Tree?

ASSIGNMENT 3

3. Write a program for implementing bubble sort technique.

Theory and Concept

Sorting Techniques

The objective of the sorting algorithm is to rearrange the records so that their keys are ordered according to some well-defined ordering rule.

Problem: Given an array of n real number A[1.. n].

Objective: Sort the elements of A in ascending order of their values.

Internal Sort

If the file to be sorted will fit into memory or equivalently if it will fit into an array, then the sorting method is called internal. In this method, any record can be accessed easily.

External Sort

- Sorting files from tape or disk.
- In this method, an external sort algorithm must access records sequentially, or at least in the block.

Memory Requirement

1. Sort in place and use no extra memory except perhaps for a small stack or table.
2. Algorithm that use a linked-list representation and so use N extra words of memory for list pointers.
3. Algorithms that need enough extra memory space to hold another copy of the array to be sorted.

A Lower Bound for the Worst Case

The length of the longest path from the root to any of its leaves represents the worst-case number of comparisons the sorting algorithm performs. Consequently, the worst-case number of comparisons corresponds to the height of its tree. A lower bound on the height of the tree is

Bubble Sort

Bubble Sort is an elementary sorting algorithm. It works by repeatedly exchanging adjacent elements, if necessary. When no exchanges are required, the file is sorted.

```
SEQUENTIAL BUBBLESORT (A)
for i ← 1 to length [A] do
   for j ← length [A] downto i +1 do
      If A[j] < A[j-1] then
         Exchange A[j] ↔ A[j-1]
```

Memory Requirement

Clearly, bubble sort does not require extra memory.

Algorithm for Parallel Bubble Sort

```
PARALLEL BUBBLE SORT (A)
1.  For k = 0 to n-2
2.  If k is even then
3.      for i = 0 to (n/2)-1 do in parallel
4.          If A[2i] > A[2i+1] then
5.              Exchange A[2i] ↔ A[2i+1]
6.  Else
7.      for i = 0 to (n/2)-2 do in parallel
8.          If A[2i+1] > A[2i+2] then
9.              Exchange A[2i+1] ↔ A[2i+2]
10. Next k
```

VIVA QUESTION

A) What is worst case and average case of bubble sort algorithm?

B) While implementing the bubble sort algorithm, how many comparisons will performed in Pass 1?

ASSIGNMENT 4

4. Write a program for implementing Insertion Sort Technique.

Insertion Sort

If the first few objects are already sorted, an unsorted object can be inserted in the sorted set in proper place. This is called insertion sort. An algorithm consider the elements one at a time, inserting each in its suitable place among those already considered (keeping them sorted).

Insertion sort is an example of an incremental algorithm; it builds the sorted sequence one number at a time.

```
INSERTION_SORT (A)
1.  For j = 2 to length [A] do
2.      key = A[j]
3.      {Put A[j] into the sorted sequence A[1 . . j-1]
4.      i ← j -1
5.      while i > 0 and A[i] > key do
6.          A[i+1] = A[i]
7.              i = i-1
8.      A[i+1] = key
```

Analysis

Best-Case

The while-loop in line 5 executed only once for each j. This happens if given array A is already sorted.

$T(n) = an + b = O(n)$

It is a linear function of n.

Worst-Case

The worst-case occurs, when line 5 executed j times for each j. This can happens if array A starts out in reverse order

$T(n) = an^2 + bn + c = O(n^2)$

It is a quadratic function of n.

Stability

Since multiple keys with the same value are placed in the sorted array in the same order that they appear in the input array, Insertion sort is stable.

Extra Memory

This algorithm does not require extra memory.

- For Insertion sort we say the worst-case running time is $\theta(n^2)$, and the best-case running time is $\theta(n)$.
- Insertion sort use no extra memory it sort in place.
- The time of Insertion sort is depends on the original order of a input. It takes a time in $\Omega(n^2)$ in the worst-case, despite the fact that a time in order of n is sufficient to solve large instances in which the items are already sorted.

VIVA QUESTION

A) For insertion sort, the number of entries we must index through when there are n elements in the array is
n-1 entries

B) Total number of comparisons for insertion sort is
n(n-1)/2

ASSIGNMENT 5

5. Write a program for implementing selection sort technique.

THEORY AND CONCEPT OF SELECTION SORT

Selection sort is the most fundamental, simple and most importantly an in-place sorting algorithm.
This algorithm divides the input list into two sub arrays-

1. **A sub array of sorted elements** which is empty at the beginning and keep on increasing with each item added to it.
2. **An unsorted sub array of remaining elements**. This is equal to the input size in the beginning and its size reduces to zero as we reach the end of algorithm.

The basic idea is that in each iteration of this algorithm we pick an element (either largest or smallest, this depends on the sorting scenario) and appends it to the sorted element list, reducing the size of unsorted list by one.
The following example show the steps taken in a selection sort. The bold numbers are the numbers which have been moved to the front of the list after being found to be the lowest value on each iteration.

➔ 5 7 3 9 2 1
--> **1** 5 7 3 9 2
--> **1 2** 5 7 3 9
--> **1 2 3** 5 7 9
--> **1 2 3 5** 7 9
--> **1 2 3 5 7** 9
--> **1 2 3 5 7 9**

Procedure

```
void selectionSort(int a[], int size)
{
  int i, j, min;

  for (i = 0; i < size - 1; i++)
  {
    min = i;
    for (j = i+1; j < size; j++)
      if (a[j] < a[min])
        min = j;

    swap(a[i], a[min]);
  }
}
```

Performance

Selection sort is very easy to analyze since none of the loops depend on the data in the array. Selecting the lowest element requires scanning all *n* elements (this takes *n* - 1 comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining *n* - 1 elements and so on, for a total of $(n - 1) + (n - 2) + ... + 2 + 1 = \Theta(n^2)$ comparisons. Each of these scans requires one swap for a total of *n* - 1 swaps (the final element is already in place). Thus, the comparisons dominate the running time, which is $\Theta(n^2)$.

VIVA QUESTION

A) In a selectionsort of n elements, how many times is the swap function called in the complete execution of the algorithm?
B) In which cases are the time complexities same in selection sort?

a. Worst and Best
b. Best and Average
c. Worst and Average
d. Worst, average and Best

ASSIGNMENT 6

6. write a program for implementing quick sort technique.

THEORY AND CONCEPT

Quick Sort

The basic version of quick sort algorithm was invented by C. A. R. Hoare in 1960 and formally introduced quick sort in 1962. It is used on the principle of divide-and-conquer. Quick sort is an algorithm of choice in many situations because it is not difficult to implement, it is a good "general purpose" sort and it consumes relatively fewer resources during execution.

Good points

- It is in-place since it uses only a small auxiliary stack.
- It requires only n log(n) time to sort n items.
- It has an extremely short inner loop
- This algorithm has been subjected to a thorough mathematical analysis, a very precise statement can be made about performance issues.

Bad Points

- It is recursive. Especially if recursion is not available, the implementation is extremely complicated.
- It requires quadratic (i.e., n^2) time in the worst-case.
- It is fragile i.e., a simple mistake in the implementation can go unnoticed and cause it to perform badly.

Quick sort works by partitioning a given array A[p . . r] into two non-empty sub array A[p . . q] and A[q+1 . . r] such that every key in A[p . . q] is less than or equal to every key in A[q+1 . . r]. Then the two subarrays are sorted by recursive calls to Quick sort. The exact position of the partition depends on the given array and index q is computed as a part of the partitioning procedure.

QuickSort

1. If p < r then
2. q Partition (A, p, r)
3. Recursive call to Quick Sort (A, p, q)
4.
5. Recursive call to Quick Sort (A, q + r, r)

Note that to sort entire array, the initial call Quick Sort (A, 1, length[A])

As a first step, Quick Sort chooses as pivot one of the items in the array to be sorted. Then array is then partitioned on either side of the pivot. Elements that are less than or equal to pivot will move toward the left and elements that are greater than or equal to pivot will move toward the right.

Partitioning the Array

Partitioning procedure rearranges the subarrays in-place.

PARTITION (A, p, r)

1. x ← A[p]
2. i ← p-1
3. j ← r+1
4. while TRUE do
5. Repeat j ← j-1
6. until A[j] ≤ x
7. Repeat i ← i+1
8. until A[i] ≥ x
9. if i < j
10. then exchange A[i] ↔ A[j]
11. else return j

Partition selects the first key, A[p] as a pivot key about which the array will partitioned:

Keys ≤ A[p] will be moved towards the left .

Keys ≥ A[p] will be moved towards the right.

The running time of the partition procedure is $\Theta(n)$ where n = r - p +1 which is the number of keys in the array.

another argument that running time of partition on a subarray of size $\Theta(n)$ is as follows: pointer i and pointer j start at each end and move towards each other, conveying somewhere in the middle. the total

number of times that i can be incremented and j can be decremented is therefore o(n). associated with each increment or decrement there are o(1) comparisons and swaps. hence, the total time is o(n).
array of same elements
since all the elements are equal, the "less than or equal" teat in lines 6 and 8 in the partition (a, p, r) will always be true. this simply means that repeat loop all stop at once. intuitively, the first repeat loop moves j to the left; the second repeat loop moves i to the right. in this case, when all elements are equal, each repeat loop moves i and j towards the middle one space. they meet in the middle, so q= floor(p+r/2). therefore, when all elements in the array a[p . . r] have the same value equal to floor(p+r/2).

VIVA QUESTION

A) Suppose we have a O(n) time algorithm that finds median of an unsorted array.
Now consider a QuickSort implementation where we first find median using the above algorithm, then use median as pivot. What will be the worst case time complexity of this modified QuickSort.
(a) O(n^2 Logn)
(b) O(n^2)
(c) O(n Logn Logn)
(d) O(nLogn)
B) Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this:
2 5 1 7 9 12 11 10
Which statement is correct?
(A) The pivot could be either the 7 or the 9.
(B) The pivot could be the 7, but it is not the 9
(C) The pivot is not the 7, but it could be the 9
(D) Neither the 7 nor the 9 is the pivot.

ASSIGNMENT 7

7. Write a program for implementing merge sort technique.

Merge Sort

Merge-sort is based on the divide-and-conquer paradigm. The Merge-sort algorithm can be described in general terms as consisting of the following three steps:

1. Divide Step

   If given array A has zero or one element, return S; it is already sorted. Otherwise, divide A into two arrays, A_1 and A_2, each containing about half of the elements of A.

2. Recursion Step

   Recursively sort array A_1 and A_2.

3. Conquer Step

   Combine the elements back in A by merging the sorted arrays A_1 and A_2 into a sorted sequence.

We can visualize Merge-sort by means of binary tree where each node of the tree represents a recursive call and each external nodes represent individual elements of given array A. Such a tree is called Merge-sort tree. The heart of the Merge-sort algorithm is conquer step, which merge two sorted sequences into a single sorted sequence.

To begin, suppose that we have two sorted arrays $A_1[1], A_1[2], \ldots, A_1[M]$ and $A_2[1], A_2[2], \ldots, A_2[N]$. The following is a direct algorithm of the obvious strategy of successively choosing the smallest remaining elements from A_1 to A_2 and putting it in A.

MERGE (A_1, A_2, A)
i.← j 1
$A_1[m+1], A_2[n+1] \leftarrow$ INT_MAX
For k ←1 to m + n do
  if $A_1[i] < A_2[j]$
    then $A[k] \leftarrow A_1[i]$
      i ← i +1
    else
      $A[k] \leftarrow A_2[j]$
        j ← j + 1

Merge Sort Algorithm

MERGE_SORT (A)
$A_1[1 \ldots \lfloor n/2 \rfloor] \leftarrow A[1 \ldots \lfloor n/2 \rfloor]$
$A_2[1 \ldots \lfloor n/2 \rfloor] \leftarrow A[1 + \lfloor n/2 \rfloor \ldots n]$
Merge Sort (A_1)
Merge Sort (A_1)
Merge Sort (A_1, A_2, A)

VIVA QUESTION

A) Determine the running time of mergesort for
a. sorted input
b.reverse-ordered input
c.random input

B) Merge sort is based on the divide and conquer strategy.it consists of the following steps
   a. Divide,Recursive and Conquer
   b. Divide and Conquer
   c. Divide and Recursive
   d. None of the above

ASSIGNMENT 8

8. Write a program for implementing heap sort technique.

Heap Sort

Heap Sort Algorithm

The heap sort combines the best of both merge sort and insertion sort. Like merge sort, the worst case time of heap sort is O(n log n) and like insertion sort, heap sort sorts in-place. The heap sort algorithm starts by using procedure BUILD-HEAP to build a heap on the input array A[1 . . n]. Since the maximum element of the array stored at the root A[1], it can be put into its correct final position by exchanging it with A[n] (the last element in A). If we now discard node n from the heap than the remaining elements can be made into heap. Note that the new element at the root may violate the heap property. All that is needed to restores the heap property.

Heap Property -> A data structure in which all nodes of the heap is either greater than equal to its children or less than equal to its children.

Max-Heapify -> Process which satisfy max-heap property (A[parent[i]] >= A[i]). Here larger element is stored at root.

Min-Heapify -> Process which satisfy min-heap property (A[parent[i]] <= A[i]). Here smaller element is stored at root.

Pseudocode

```
MaxHeapify(A, i)
   l = left(i)
   r = right(i)
   if l <= heap-size[A] and A[l] > A[i]
      then largest = l
      else largest = i
   if r <= heap-size[A] and A[r] > A[largest]
      then largest = r
   if largest != i
      then swap A[i] with A[largest]
      MaxHeapify(A, largest)
end func

BuildMaxHeap(A)
   heap-size[A] = length[A]
   for i = |length[A]/2| downto 1
      do MaxHeapify(A, i)
end func

HeapSort(A)
   BuildMaxHeap(A)
   for i = length[A] downto 2
      do swap A[1] with A[i]
         heap-size[A] = heap-size[A] – 1
         MaxHeapify(A, 1)
end func
```

The HEAPSORT procedure takes time O(n lg n), since the call to BUILD_HEAP takes time O(n) and each of the n -1 calls to Heapify takes time O(lg n). Asymptotic Analysis of Heap Sort

Since in Heap Sort we first call Build-Max-Heap function which takes O(n) time then we call Max-Heapify function n time, where Max-Heapify function takes O(logn) time so, total time complexity of heap-sort comes out to be O(n+n*logn) which further evaluated to O(n*logn).

VIVA QUESTION

A) The condition applicable for max-heap is

a) A[Parent(i)]>=A[i]
b) A[Parent(i)]<=A[i]
c) A[Parent(i+1)]>A[i]
d) A[Parent(i+1)]

B) If a max heap is implemented using a partially filled array called data, and the array contains n elements (n > 0), where is the entry with the greatest value? (most appropriate answer)

a) data[0]
b) data[n-1]
c) data[n]
d) data[2*n + 1]
e) data[2*n + 2]

ASSIGNMENT NO – 9

9. Print all the nodes reachable from a given starting node in a digraph using BFS method.

Theory and Concept:

breadth first search:

```
unmark all vertices
choose some starting vertex x
mark x
list L = x
tree T = x
while L nonempty
choose some vertex v from front of list
visit v
for each unmarked neighbor w
   mark w
   add it to end of list
   add edge vw to T
```

VIVA QUESTION

A) What is the complexity of BFS algo

B) Find out BFS is compatible with queue or stack

ASSIGNMENT NO – 10

10. Check whether a given graph is connected or not using DFS method.

Depth first search

Depth first search is another way of traversing graphs, which is closely related to preorder traversal of a tree. Recall that preorder traversal simply visits each node before its children. It is most easy to program as a recursive routine:

```
preorder(node v)
{
visit(v);
for each child w of v
   preorder(w);
}
```

To turn this into a graph traversal algorithm, we basically replace "child" by "neighbor". But to prevent infinite loops, we only want to visit each vertex once. Just like in BFS we can use marks to keep track of the vertices that have already been visited, and not visit them again. Also, just like in BFS, we can use this search to build a spanning tree with certain useful properties.

```
dfs(vertex v)
{
visit(v);
for each neighbor w of v
   if w is unvisited
   {
   dfs(w);
   add edge vw to tree T
   }
}
```

VIVA QUESTION

A) What is the complexity of DFS algo.
B) Find out DFS is compatible with queue or stack

ASSIGNMENT NO – 11

11. Write a program for implementing Minimal Spanning Tree using Kruskal's Algorithm.

Theory and Concept:

Spanning Tree and Minimum Spanning Tree

Spanning Trees

A spanning tree of a graph is any tree that includes every vertex in the graph. Little more formally, a spanning tree of a graph G is a subgraph of G that is a tree and contains all the vertices of G. An edge of a spanning tree is called a branch; an edge in the graph that is not in the spanning tree is called a chord. We construct spanning tree whenever we want to find a simple, cheap and yet efficient way to connect a set of terminals (computers, cites, factories, etc.). Spanning trees are important because of following reasons.

- Spanning trees construct a sparse sub graph that tells a lot about the original graph.
- Spanning trees a very important in designing efficient routing algorithms.
- Some hard problems (e.g., Steiner tree problem and traveling salesman problem) can be solved approximately by using spanning trees.
- Spanning trees have wide applications in many areas, such as network design, etc.

Greedy Spanning Tree Algorithm

One of the most elegant Spanning Tree algorithms that I know of is as follows:

- Examine the edges in graph in any arbitrary sequence.
- Decide whether each edge will be included in the spanning tree.

Note that each time a step of the algorithm is performed, one edge is examined. If there are only a finite number of edges in the graph, the algorithm must halt after a finite number of steps. Thus, the time complexity of this algorithm is clearly O(n), where n is the number of edges in the graph.
Some important facts about spanning trees are as follows:

6. Any two vertices in a tree are connected by a unique path.
7. Let T be a spanning tree of a graph G, and let e be an edge of G not in T. The T + e contain a unique cycle.

Greediness

It is easy to see that this algorithm has the property that each edge is examined at most once. Algorithms, like this one, which examine each entity at most once and decide its fate once and for all during that examination is called greedy algorithms. The obvious advantage of greedy approach is that we do not have to spend time reexamining entities.
Consider the problem of finding a spanning tree with the smallest possible weight or the largest possible weight, respectively called a minimum spanning tree and a maximum spanning tree. It is easy to see that if a graph possesses a spanning tree, it must have a minimum spanning tree and also a maximum spanning tree. These spanning trees can be constructed by performing the spanning tree algorithm (e.g., above mentioned algorithm) with an appropriate ordering of the edges.

Minimum Spanning Tree Algorithm

Perform the Spanning Tree algorithm (above) by examining the edges is order of non decreasing weight (smallest first, largest last). If two or more edges have the same weight, order them arbitrarily.

Minimum Spanning Trees

A minimum spanning tree (MST) of a weighted graph G is a spanning tree of G whose edges sum is minimum weight. In other words, a MST is a tree formed from a subset of the edges in a given undirected graph, with two properties:

12. it spans the graph, i.e., it includes every vertex of the graph.
13. it is a minimum, i.e., the total weight of all the edges is as low as possible.

Let G=(V, E) be a connected, undirected graph where V is a set of vertices (nodes) and E is the set of edges. Each edge has a given non negative length.
Problem Find a subset T of the edges of G such that all the vertices remain connected when only the edges T are used, and the sum of the lengths of the edges in T is as small as possible.
Let G` = (V, T) be the partial graph formed by the vertices of G and the edges in T. [Note: A connected graph with n vertices must have at least n-1 edges AND more that n-1 edges implies at least one cycle]. So n-1 is the minimum number of edges in the T. Hence if G` is connected and T has more that n-1 edges, we can remove at least one of these edges without disconnecting (choose an edge that is part of cycle). This will decrease the total length of edges in T.
G` = (V, T) where T is a subset of E. Since connected graph of n nodes must have n-1 edges otherwise there exist at least one cycle. Hence if G` is connected and T has more that n-1 edges. Implies that it contains at least one cycle. Remove edge from T without disconnecting the G` (i.e., remove the edge that is part of the cycle). This will decrease the total length of the edges in T. Therefore, the new solution is preferable to the old one.
Thus, T with n vertices and more edges can be an optimal solution. It follow T must have n-1 edges and since G` is connected it must be a tree. The G` is called Minimum Spanning Tree (MST).

Kruskal's Algo

Edge first
1. Arrange all edges in a list (L)
in non-decreasing order
2. Select edges from L, and include
that in set T, avoid cycle.
3. Repeat 3 until T becomes a tree
that covers all vertices

VIVA QUESTION

A) Find the time complexity of Kruskal algorithm
B) Explain the procedure of Kruskal algo

ASSIGNMENT NO – 12

12. Write a program for implementing Minimal Spanning Tree using Prim's Algorithm.

Theory and Concept:

Spanning Tree and Minimum Spanning Tree

Spanning Trees

A spanning tree of a graph is any tree that includes every vertex in the graph. Little more formally, a spanning tree of a graph G is a subgraph of G that is a tree and contains all the vertices of G. An edge of a spanning tree is called a branch; an edge in the graph that is not in the spanning tree is called a chord. We construct spanning tree whenever we want to find a simple, cheap and yet efficient way to connect a set of terminals (computers, cites, factories, etc.). Spanning trees are important because of following reasons.

- Spanning trees construct a sparse sub graph that tells a lot about the original graph.
- Spanning trees a very important in designing efficient routing algorithms.
- Some hard problems (e.g., Steiner tree problem and traveling salesman problem) can be solved approximately by using spanning trees.
- Spanning trees have wide applications in many areas, such as network design, etc.

Greedy Spanning Tree Algorithm

One of the most elegant Spanning Tree algorithms that I know of is as follows:

- Examine the edges in graph in any arbitrary sequence.
- Decide whether each edge will be included in the spanning tree.

Note that each time a step of the algorithm is performed, one edge is examined. If there are only a finite number of edges in the graph, the algorithm must halt after a finite number of steps. Thus, the time complexity of this algorithm is clearly O(n), where n is the number of edges in the graph.

Some important facts about spanning trees are as follows:

8. Any two vertices in a tree are connected by a unique path.
9. Let T be a spanning tree of a graph G, and let e be an edge of G not in T. The T + e contain a unique cycle.

Greediness

It is easy to see that this algorithm has the property that each edge is examined at most once. Algorithms, like this one, which examine each entity at most once and decide its fate once and for all during that examination is called greedy algorithms. The obvious advantage of greedy approach is that we do not have to spend time reexamining entities.

Consider the problem of finding a spanning tree with the smallest possible weight or the largest possible weight, respectively called a minimum spanning tree and a maximum spanning tree. It is easy to see that if a graph possesses a spanning tree, it must have a minimum spanning tree and also a maximum spanning tree. These spanning trees can be constructed by performing the spanning tree algorithm (e.g., above mentioned algorithm) with an appropriate ordering of the edges.

Minimum Spanning Tree Algorithm

Perform the Spanning Tree algorithm (above) by examining the edges is order of non decreasing weight (smallest first, largest last). If two or more edges have the same weight, order them arbitrarily.

Minimum Spanning Trees

A minimum spanning tree (MST) of a weighted graph G is a spanning tree of G whose edges sum is minimum weight. In other words, a MST is a tree formed from a subset of the edges in a given undirected graph, with two properties:

14. it spans the graph, i.e., it includes every vertex of the graph.
15. it is a minimum, i.e., the total weight of all the edges is as low as possible.

Let G=(V, E) be a connected, undirected graph where V is a set of vertices (nodes) and E is the set of edges. Each edge has a given non negative length.

Problem- Find a subset T of the edges of G such that all the vertices remain connected when only the edges T are used, and the sum of the lengths of the edges in T is as small as possible.

Let G` = (V, T) be the partial graph formed by the vertices of G and the edges in T. [Note: A connected graph with n vertices must have at least n-1 edges AND more that n-1 edges implies at least one cycle]. So n-1 is the minimum number of edges in the T. Hence if G` is connected and T has more that n-1 edges, we can remove at least one of these edges without disconnecting (choose an edge that is part of cycle). This will decrease the total length of edges in T.

G` = (V, T) where T is a subset of E. Since connected graph of n nodes must have n-1 edges otherwise there exist at least one cycle. Hence if G` is connected and T has more that n-1 edges. Implies that it contains at least one cycle. Remove edge from T without disconnecting the G` (i.e., remove the edge that is part of the cycle). This will decrease the total length of the edges in T. Therefore, the new solution is preferable to the old one.
Thus, T with n vertices and more edges can be an optimal solution. It follow T must have n-1 edges and since G` is connected it must be a tree. The G` is called Minimum Spanning Tree (MST).

Prim's Algorithm
Start form any arbitrary vertex
Find the edge that has minimum
weight form all known vertices
Stop when the tree covers all
vertices

VIVA QUESTION

A) Find the time complexity of Prims algorithm
B) Explain the procedure of Prims algo

ASSIGNMENT NO – 13

13. Write a program for implementing Knapsack problem using Greedy Method.

Theory and Concept

Implementation Greedy algorithms

Greedy algorithms are simple and straightforward. They are shortsighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect these decisions may have in the future. They are easy to invent, easy to implement and most of the time quite efficient. Many problems cannot be solved correctly by greedy approach. Greedy algorithms are used to solve optimization problems

Greedy Approach

Greedy Algorithm works by making the decision that seems most promising at any moment; it never reconsiders this decision, whatever situation may arise later.

Problem Make a change of a given amount using the smallest possible number of coins.

Informal Algorithm

- Start with nothing.
- at every stage without passing the given amount.
    - add the largest to the coins already chosen.

Formal Algorithm

Make change for n units using the least possible number of coins.

```
MAKE-CHANGE (n)
 C ← {100, 25, 10, 5, 1} // constant.
 Sol ← {};// set that will hold the solution set.
Sum ← 0 sum of item in solution set
WHILE sum not = n
x = largest item in set C such that sum + x ≤ n
 IF no such item THEN
 RETURN "No Solution"
 S ← S {value of x}
 sum ← sum + x
 RETURN S
```

Characteristics and Features of Problems solved by Greedy Algorithms

To construct the solution in an optimal way, Algorithm maintains two sets. One contains chosen items and the other contains rejected items.

The greedy algorithm consists of four (4) functions.

1. A function that checks whether chosen set of items provide a solution.
2. A function that checks the feasibility of a set.
3. The selection function tells which of the candidates is the most promising.
4. An objective function, which does not appear explicitly, gives the value of a solution.

Structure Greedy Algorithm

1. Initially the set of chosen items is empty i.e., solution set.
2. At each step
    1. item will be added in a solution set by using selection function.
    2. IF the set would no longer be feasible
        1. reject items under consideration (and is never consider again).
    3. ELSE IF set is still feasible THEN
        1. add the current item.

Definitions of feasibility

A feasible set (of candidates) is promising if it can be extended to produce not merely a solution, but an optimal solution to the problem. In particular, the empty set is always promising why? (because an optimal solution always exists)

Unlike Dynamic Programming, which solves the subproblems bottom-up, a greedy strategy usually progresses in a top-down fashion, making one greedy choice after another, reducing each problem to a smaller one.

Greedy-Choice Property

The "greedy-choice property" and "optimal substructure" are two ingredients in the problem that lend to a greedy strategy.

Greedy-Choice Property

It says that a globally optimal solution can be arrived at by making a locally optimal choice.

VIVA QUESTION

A) Which of the following standard algorithms is not a Greedy algorithm?

(a) Dijkstra's shortest path algorithm
(b) Prim's algorithm
(c) Kruskal algorithm
(d) Huffman Coding
(e) Bellmen Ford Shortest path algorithm

B) A networking company uses a compression technique to encode the message before transmitting over the network. Suppose the message contains the following characters with their frequency:

haracter	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Note : Each character in input message takes 1 byte.

If the compression technique used is Huffman Coding, how many bits will be saved in the message?

(A)224
(B)800
(C)576
(D)324

ASSIGNMENT NO – 14

14. Write a program for implementing Job Sequencing Problem using Greedy Method.

Knapsack Problem

Statement

A thief robbing a store and can carry a maximal weight of w into their knapsack. There are n items and i^{th} item weigh w_i and is worth v_i dollars. What items should thief take?

There are two versions of problem

- Fractional knapsack problem
  The setup is same, but the thief can take fractions of items, meaning that the items can be broken into smaller pieces so that thief may decide to carry only a fraction of x_i of item i, where $0 \leq x_i \leq 1$.
  Exhibit greedy choice property.
    - Greedy algorithm exists.

  Exhibit optimal substructure property.
    - ?????
- 0-1 knapsack problem
  The setup is the same, but the items may not be broken into smaller pieces, so thief may decide either to take an item or to leave it (binary choice), but may not take a fraction of an item.
  Exhibit No greedy choice property.
    - No greedy algorithm exists.

  Exhibit optimal substructure property.
    - Only dynamic programming algorithm exists.

Greedy Solution to the Fractional Knapsack Problem

There are n items in a store. For i =1,2, . . . , n, item i has weight $w_i > 0$ and worth $v_i > 0$. Thief can carry a maximum weight of W pounds in a knapsack. In this version of a problem the items can be broken into smaller piece, so the thief may decide to carry only a fraction x_i of object i, where $0 \leq x_i \leq 1$. Item i contributes $x_i w_i$ to the total weight in the knapsack, and $x_i v_i$ to the value of the load.

In Symbol, the fraction knapsack problem can be stated as follows.

maximize $^{n}S_{i=1}\, x_i v_i$ subject to constraint $^{n}S_{i=1}\, x_i w_i \leq W$

It is clear that an optimal solution must fill the knapsack exactly, for otherwise we could add a fraction of one of the remaining objects and increase the value of the load. Thus in an optimal solution $^{n}S_{i=1}\, x_i w_i = W$.

```
Greedy-fractional-knapsack (w, v, W)
for i =1 to n
do x[i] =0
weight = 0
while weight < W
do i = best remaining item
 IF weight + w[i] ≤ W
then x[i] = 1
 weight = weight + w[i]
else
x[i] = (w - weight) / w[i]
 weight = W
return x
```

Analysis

If the items are already sorted into decreasing order of v_i / w_i, then the while-loop takes a time in O(n); Therefore, the total time including the sort is in O(n log n).

If we keep the items in heap with largest v_i/w_i at the root. Then

11. creating the heap takes O(n) time
12. while-loop now takes O(log n) time (since heap property must be restored after the removal of root)

In a job-scheduling problem, we are given a list of n jobs. Every job i is associated with an integer deadline di³ 0 and a profit pi³ 0 for any job i, profit is earned if and only if the job is completed within its deadline. A feasible solution with maximum sum of profits is to be obtained now.

To find the optimal solution and feasibility of jobs we are required to find a subset J such that each job of this subset can be completed by its deadline. The value of a feasible solution J is the sum of profits of all the jobs in J.

Steps in finding the subset J are as follows:

a. □ p_i i □ J is the objective function chosen for optimization measure.

b. Using this measure, the next job to be included should be the one which increases p_i i J.
c. Begin with J = Æ and0 $p_i = 0$ iJ
d. Add a job to J which has the largest profit
e. Add another job to this J keeping in mind the following condition:
i. Search for job which has the next maximum profit.
ii. See if this job is union with J is feasible or not.
iii. If yes go to step (e) and continue else go to (iv)
iv. Search for the job with next maximum profit and go to step (b)
f. Terminate when addition of no more jobs is feasible.

Illustration
Consider 5 jobs with profits (p1,p2,p3,p4,p5) = (20,15,10,5,1)
and maximum delay allowed (d1,d2,d3,d4,d5) = (2,2,1,3,3).
Here maximum number of jobs that can be completed is
= Min(n,maxdelay(di))
= Min(5,3)
= 3.
Hence there is a possibility of doing 3 jobs.
There are 3 units of time
Time Slot
[0-1] [1-2] [2-3] Profit
Job
1 - yes - 20
2 yes - - 15
3 cannot accommodate —
4 - - yes 5
40
In the first unit of time job 2 is done and a profit of 15 is gained, in the second unit job 1 is done and a profit 20 is obtained finally in the 3rd unit since the third job is not available 4th job is done and 5 is obtained as the profit in the above job 3 and 5 could not be accommodated due to their deadlines.

VIVA QUESTION

A) **How come find returns the latest available time slot?**
B) How to Creates initial disjoint sets?

ASSIGNMENT NO – 15

15. Implement All-Pairs Shortest Paths Problem using Dynamic Programming (Floyd's algorithm)

Bellman-Ford and Dijkstra's algorithms provide a means to find the shortest path from a given source. However often we may wish to find the shortest paths between all pairs of vertices. One way to accomplish this would be to simply run Bellman-Ford or Dijkstra's algorithm for each vertex in the graph. Thus the run times for these strategies would be (particularly for dense graphs where $|E| \approx |V|^2$):

Bellman-Ford- $|V|\ O(VE) \approx O(V^4)$

Dijkstra- $|V|\ O(V^2+E) \approx O(V^3)$

$|V|\ O(V\lg V+E) \approx O(V^2\lg V+VE)$

In the case of dense graphs an often more efficient algorithm (with very low hidden constants) for finding all pairs shortest paths is the Floyd-Warshall algorithm.

FLOYD-WARSHALL(W)

1. n = W.rows
2. $D^{(0)} = W$
3. $\Pi^{(0)} = \pi^{(0)}_{ij}$ = NIL if i=j or $w_{ij} = \infty$
   = i if i≠j and $w_{ij} < \infty$
4. for k = 1 to n
5. let $D^{(k)} = (d^{(k)}_{ij})$ be a new nxn matrix
6. for i = 1 to n
7. for j = 1 to n
8. $d^{k}_{ij} = \min(d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj})$
9. if $d^{(k-1)}_{ij} \le d^{(k-1)}_{ik} + d^{(k-1)}_{kj}$
10. $\pi^{(k)}_{ij} = \pi^{(k-1)}_{ij}$
11. else
12. $\pi^{(k)}_{ij} = \pi^{(k-1)}_{kj}$
13. return $D^{(n)}$

VIVA QUESTION

A) Which of the following standard algorithms is not Dynamic Programming based.
(a) Bellman–Ford Algorithm for single source shortest path
(b) Floyd Warshall Algorithm for all pairs shortest paths
(c)0-1 Knapsack problem
(d) Prim's Minimum Spanning Tree

B) What is time complexity of fun()?

```
int fun(int n)
{
  int count = 0;
  for (int i = n; i > 0; i /= 2)
    for (int j = 0; j < i; j++)
      count += 1;
  return count;
}
```

(a)O(n^2)
(b)O(nLogn)
(c)O(n)
(d)O(nLognLogn)

ASSIGNMENT NO – 16

16. Study of NP-Complete theory.

INTRODUCTION

- A problem is said to be polynomial if there exists an algorithm that solves the problem in time $T(n)=O(n^c)$, where c is a constant.
- Examples of polynomial problems:
  - Sorting: $O(n \log n) = O(n^2)$
  - All-pairs shortest path: $O(n^3)$
  - Minimum spanning tree: $O(E \log E)= O(E^2)$
- A problem is said to be exponential if no polynomial-time algorithm can be developed for it and if we can find an algorithm that solves it in $O(n^{u(n)})$, where u(n) goes to infinity as n goes to infinity.
- The world of computation can be subdivided into three classes:
  - Polynomial problems (P)
  - Exponential problems (E)
  - Intractable (non-computable) problems (I)
- There is a very large and important class of problems that
  - we know how to solve exponentially,
  - we don't know how to solve polynomially, and
  - we don't know if they can be solved polynomially at all

This class is a gray area between the P-class and the E-class. It will be studied in this chapter.

Definition of NP

- Definition-1 of NP: A problem is said to be Non-deterministically Polynomial (NP) if we can find a non-determinsitic Turing machine that can solve the problem in a polynomial number of non-deterministic moves.
- For those who are not familiar with Turing machines, two alternative definitions of NP will be developed.
- Definition-2 of NP: A problem is said to be NP if
  1. its solution comes from a finite set of possibilities, and
  2. it takes polynomial time to verify the correctness of a candidate solution
- Remark: It is much easier and faster to "grade" a solution than to find a solution from scratch.
- We use NP to designate the class of all non-deterministically polynomial problems.
- Clearly, P is a subset of NP
- A very famous open question in Computer Science:

  P = NP ?

- To give the 3rd alternative definition of NP, we introduce an imaginary, non-implementable instruction, which we call "choose()".
- Behavior of "choose()":
  1. if a problem has a solution of N components, choose(i)magically returns the i-th component of the CORRECT solution in constant time
  2. if a problem has no solution, choose(i)returns mere "garbage", that is, it returns an uncertain value.
- An NP algorithm is an algorithm that has 2 stages:
  1. The first stage is a guessing stage that useschoose() to find a solution to the problem.
  2. The second stage checks the correctness of the solution produced by the first stage. The time of this stage is polynomial in the input size n.
- Template for an NP algorithm:

```
begin
  /* The following for-loop is the guessing stage*/
  for i=1 to N do
    X[i] := choose(i);
  endfor
```

```
  /* Next is the verification stage */
  Write code that does not use "choose" and that
  verifies if X[1:N] is a correct solution to the
  problem.
end
```

•Remark: For the algorithm above to be polynomial, the solution size N must be polynomial in n, and the verification stage must be polynomial in n.
•Definition3 of NP: A problem is said to be NP if there exists an NP algorithm for it.
•Example of an NP problem: The Hamiltonian Cycle (HC) problem
1.Input: A graph G
2.Question: Goes G have a Hamiltonian Cycle?
•Here is an NP algorithm for the HC problem:

```
begin
  /* The following for-loop is the guessing stage*/
  for i=1 to n do
    X[i] := choose(i);
  endfor

  /* Next is the verification stage */
  for i=1 to n do
      for j=i+1 to n do
        if X[i] = X[j] then
            return(no);
        endif
      endfor
  endfor
  for i=1 to n-1 do
      if (X[i],X[i+1]) is not an edge then
            return(no);
      endif
  endfor
  if (X[n],X[1]) is not an edge then
      return(no);
  endif

  return(yes);
end
```

•The solution size of HC is O(n), and the time of the verification stage is $O(n^2)$. Therefore, HC is NP.
•The K-clique problem is NP
1.Input: A graph G and an integer k
2.Question: Goes G have a k-clique?
•Here is an NP algorithm for the K-clique problem:

```
begin
  /* The following for-loop is the guessing stage*/
  for i=1 to k do
    X[i] := choose(i);
  endfor

  /* Next is the verification stage */
  for i=1 to k do
      for j=i+1 to k do
        if (X[i] = X[j] or (X[i],X[j]) is not an edge) then
            return(no);
```

```
        endif
      endfor
    endfor

    return(yes);
  end
```

- The solution size of the k-clique is O(k)=O(n), and the time of the verification stage is $O(n^2)$. Therefore, the k-clique problem is NP.

Focus on Yes-No Problems

- Definition: A yes-no problem consists of an instance (or input I) and a yes-no question Q.
- The yes-no version of the HC problem was described above, and so was the yes-no version of the k-clique problem.
- The following are additional examples of well-known yes-no problems.
- The subset-sum problem:
  - Instance: a real array a[1:n]
  - Question: Can the array be partitioned into two parts that add up to the same value?
- The satisfiability problem (SAT):
  - Instance: A Boolean Expression F
  - Question: Is there an assignment to the variables in F so that F evaluates to 1?
- The Treveling Salesman Problem

The original formulation:

  - Instance: A weighted graph G
  - Question: Find a minimum-weight Hamiltonian Cycle in G.

The yes-no formulation:

  - Instance: A weighted graph G and a real number d
  - Question: Does G have a Hamiltonian cycle of weight <= d?

Reductions and Transforms

- Notation: If P stands for a yes-no problem, then
  - I_P: denotes an instance of P
  - Q_P: denotes the question of P
  - Answer(Q_P,I_P): denotes the answer to the question Q_P given input I_P
- Let P and R be two yes-no problems
- Definition: A transform(that transforms a problem P to a problem R) is an algorithm T such that:
  - The algorithm T takes polynomial time
  - The input of T is I_P, and the output of T is I_R
  - Answer(Q_P,I_P)=Answer(Q_R,I_R)
- Definition: We say that problem problem P reduces to problem R if there exists a transform from P to R.

NP-Completeness

- Definition: A problem R is NP complete if
  1. R is NP
  2. Every NP problem P reduces to R
- An equivalent but casual definition: A problem R is NP-complete if R is the "most difficult" of all NP problems.
- Theorem: Let P and R be two problems. If P reduces to R and R is polynomial, then P is polynomial.
- Proof:
  1. Let T be the transform that transforms P to R. T is a polynomial time algorithm that transforms I_P to I_R such that

     $$\text{Answer}(Q_P,I_P) = \text{Answer}(Q_R,I_R)$$

  2. Let A_R be the polynomial time algorithm for problem R. Clearly, A takes as input I_R, and returns as output Answer(Q_R,I_R)

3.Design a new algorithm A_P as follows:

AlgorithmA_P(input: I_P)

begin

$I_R = T(I_P)$;

$x := A_R(I_R)$;

return x;

end

4.Note that this algorithm A_P returns the correct answer Answer(Q_P,I_P) because $x = A_R(I_R)$ = Answer(Q_R,I_R) = Answer(Q_P,I_P).

5.Note also that the algorithm A_P takes polynomial time because both T and A_R

Q.E.D.

- The intuition derived from the previous theorem is that if a problem P reduces to problem R, then R is at least as difficult as P.
- Theorem: A problem R is NP-complete if
  1. R is NP, and
  2. There exists an NP-complete problem R_0 that reduces to R
- Proof:
  1. Since R is NP, it remain to show that any arbitrary NP problem P reduces to R.
  2. Let P be an arbitrary NP problem.
  3. Since R_0 is NP-complete, it follows that P reduces to R_0
  4. And since R_0 reduces to R, it follows that P reduces to R (by transitivity of transforms).

- The previous theorem amounts to a strategy for proving new problems to be NP complete. Specifically, to problem a new problem R to be NP-complete, the following steps are sufficient:
  1. Prove R to be NP
  2. Find an already known NP-complete problem R_0, and come up with a transform that reduces $R_0$to R.
- For this strategy to become effective, we need at least one NP-complete problem. This is provided by Cook's Theorem below.
- Cook's Theorem: SAT is NP-complete.

. NP-Completeness of the k-Clique Problem

- The k-clique problem was already shown to be NP.
- It remain to prove that an NP-complete problem reduces to k-clique
- Theorem: SAT reduces to the k-clique problem
- Proof:
  - Let F be a Boolean expression.
  - F can be put into a conjunctive normal form: $F=F_1F_2...F_r$

    where every factor F_i is a sum of literals (a literal is a Bollean variable or its complement)
  - Let k=r and G=(V,E) defined as follows:

    $V=\{<x_i,F_j> \mid x_i$ is a variable in $F_j\}$

    $E=\{(<x_i,F_j>, <y_s,F_t>) \mid j \neq t$ and $x_i \neq y_s'\}$

    where y_s' is the complement of y_s
  - We prove first that if F is satisfiable, then there is a k-clique.
  - Assume F is satisfiable
  - This means that there is an assignment that makes F equal to 1
  - This implies that $F_1=1, F_2=1, ..., F_r=1$
  - Therefore, in every factor F_i there is (at least) one variable assigned 1. Call that variable z_i
  - As a result, $<z_1,F_1>, <z_2,F_2>, ..., <z_k,F_k>$ is a k-clique in G because they are k distinct nodes, and each pair ($<z_i,F_i>$, $<z_j,F_j>$) forms an edge since the endpoints come from different factors and $z_i \neq z_j'$ due to the fact that they are both assigned 1.

- We finally prove that if G has a k-clique, then F is satisfiable
- Assume G has a k-clique $<u_1,F_1>$, $<u_2,F_2>$, ... , $<u_k,F_k>$ which are pairwise adjacent
- These k nodes come the k different factors, one per factor, because no two nodes from the same factor can be adjacent
- Furthermore, no two u_i and u_j are complements because the two nodes $<u_i,F_i>$ and $<u_j,F_j>$ are adjacent, and adjacent nodes have non-complement first-components.
- As a result, we can consistently assign each u_i a value 1.
- This assignment makes each F_i equal to 1 because u_i is one of the additive literals in F_i
- Consequently, F is equal to 1.

- An illustration of the proof will be carried out in class on $F=(x_1+ x_2)(x_1' + x_3)(x_2+ x_3')$

VIVA QUESTION

A. Define the complexity of TSP.
B. Differentiate NP-complete and NP- Hard problem.