STES's

SINHGAD COLLEGE OF ENGINEERING

Vadgaon (Bk), Pune

Department of Computer Engineering

Sinhgad Institutes

LABORATORY MANUAL

2024-2025

LABORATORY PRACTICE-II

TE-COMPUTER ENGINEERING

SEMESTER-II

Subject Code: **310258**

TEACHING SCHEME	EXAMINATION SCHEME
Practical: 4 Hrs/Week	Practical: 25 Marks
	Term Work: 50 Marks

-: **Name of Faculty**:-

Prof. P. M. KAMDE

Prof. Runal Pawar

Sinhgad Institutes

Sinhgad College of Engineering

Department of Computer Engineering

List of Assignments

Sr. No.	Title of Assignment
	AI Assignments
1	Depth and Breadth First Search algorithm
2	A star Algorithm
3	Greedy search algorithm
4	Solution for a Constraint Satisfaction Problem
5	Development of elementary chatbot
6	Implement of Expert System
	IS Assignments
1	Perform Encryption and Decryption using method of transposition technique.
2	Implementation of DES (Data Encryption Standard) Algorithm
3	Implementation of AES (Advanced Encryption Standard) Algorithm
4	Implementation of RSA Algorithm
5	Implementation of Different-Hellman key Exchange (DH)
	CC Assignments
1	Case study on amazon EC2
2	Installation and configuration of Google App Engine
3	Creation an Application in Salesforce.com using Apex Programming Language.
4	Design and Develop custom Application using sales force Cloud.

Assignment No. 1

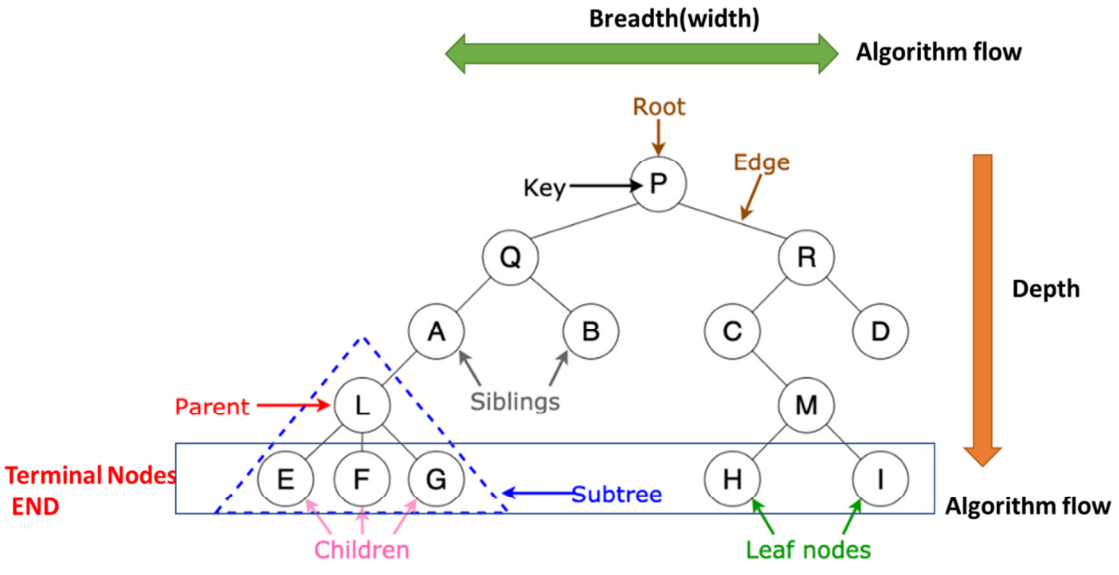
Title: Depth and Breadth First Search algorithm.

Problem Definition: Implement depth first search algorithm and Breadth First Search algorithm, use an undirected graph and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

Theory:

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. Breadth-first search is an instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion. This is achieved very simply by using a FIFO queue for the frontier.

Depth first Search or Depth first traversal is a recursive algorithm for searching all the vertices of a graph or tree data structure. Traversal means visiting all the nodes of a graph. This is achieved very simply by using a LIFO queue for the frontier.



Algorithm:

- Declare a queue and insert the starting vertex.
- Initialize a visited array and mark the starting vertex as visited.
- Follow the below process till the queue becomes empty:
- Remove the first vertex of the queue.
- Mark that vertex as visited.
- Insert all the unvisited neighbors of the vertex into the queue.

Code:

```
# Python3 Program to print BFS traversal
# from a given source vertex. BFS(int s)
# traverses vertices reachable from s.
from collections import defaultdict
# This class represents a directed graph
# using adjacency list representation


class Graph:

    # Constructor
    def __init__(self):
        # default dictionary to store graph
        self.graph = defaultdict(list)
    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)
    # Function to print a BFS of graph
    def BFS(self, s):
        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)
        # Create a queue for BFS
       queue = []
       queue.append(s)
        visited[s] = True
        while queue:
            s = queue.pop(0)
            print (s, end = " ")
            for i in self.graph[s]:
            if visited[i] == False:
            queue.append(i)
            visited[i] = True
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
```

```
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
print ("Following is Breadth First Traversal"
                " (starting from vertex 2)")
g.BFS(2)
```

Conclusion: Thus, we have learned depth first search (DFS) & breadth first search (BFS) algorithms.

Assignment No. 2

Title: A star Algorithm.

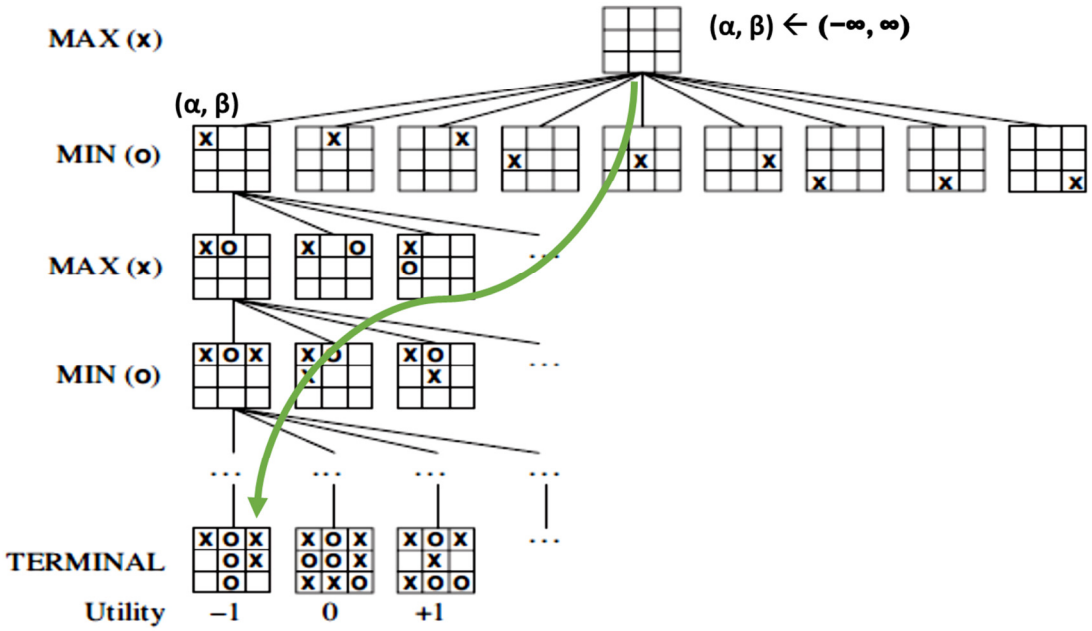
Problem Definition: Implement A star algorithm for tic tac toe game search problem

Concept: The most widely known form of best-first search is called A*("A-star search"). It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

$$f(n) = g(n) + h(n)$$

Since g(n) gives the path cost from the start node to node n, and h(n) is the estimated cost of the cheapest path from n to the goal, we have

f (n) = estimated cost of the cheapest solution through n.



Here:

f(s) = g(s,a) + h(s)
Evaluation function= Result(s,a) + Heuristic function
for MIN player h(α, β)= minimum(α, β)
Hence f(s) = Result(s,a) + minimum(α, β)

Algorithm

1. Initialize the open list
2. Initialize the closed list

   put the starting node on the open

   list (you can leave its f at zero)
3. while the open list is not empty

   a) find the node with the least f on the open list, call it "q"

   b) pop q off the open list

   c) generate q's 8 successors and set their parents to q

   d) for each successor

      i) if successor is the goal, stop search

      ii) else, compute both g and h for successor successor.g = q.g + distance between

      successor and q

      successor.h = distance from goal to successor

      successor.f = successor.g + successor.h

      iii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor

      iv) if a node with the same position as successor is in the CLOSED list which has

      a lower f than successor, skip this successor

      otherwise, add the node to the open list

      end (for loop)

   e) push q on the closed list

   end (while loop)

Code:

```
//  C++ Program to implement A* Search Algorithm
#include "math.h"
#include <array>
#include <chrono>
#include <cstring>
#include <iostream>
#include <queue>
#include <set>
#include <stack>
#include <tuple>
using namespace std;
```

```
typedef pair<int, int> Pair;
typedef tuple<double, int, int> Tuple;
struct cell {
	// Row and Column index of its parent
	Pair parent;
	// f = g + h
	double f, g, h;
	cell()
		: parent(-1, -1)
		, f(-1)
		, g(-1)
		, h(-1)
	{
	}
};
template <size_t ROW, size_t COL>
bool isValid(const array<array<int, COL>, ROW>& grid,const Pair& point)
{	if (ROW > 0 && COL > 0)
		return (point.first >= 0) && (point.first < ROW)
			&& (point.second >= 0)
			&& (point.second < COL);
	return false;
}
template <size_t ROW, size_t COL>
bool isUnBlocked(const array<array<int, COL>, ROW>& grid, const Pair& point)
{
	// Returns true if the cell is not blocked else false
	return isValid(grid, point)
		&& grid[point.first][point.second] == 1;
}
bool isDestination(const Pair& position, const Pair& dest)
{
	return position == dest;
}
double calculateHValue(const Pair& src, const Pair& dest)
{
	// h is estimated with the two points distance formula
	return sqrt(pow((src.first - dest.first), 2.0)
				+ pow((src.second - dest.second), 2.0));
}
template <size_t ROW, size_t COL>
void tracePath(
	const array<array<cell, COL>, ROW>& cellDetails,
	const Pair& dest)
{
	printf("\nThe Path is ");
```

```
        stack<Pair> Path;
        int row = dest.first;
        int col = dest.second;
        Pair next_node = cellDetails[row][col].parent;
        do {
                Path.push(next_node);
                next_node = cellDetails[row][col].parent;
                row = next_node.first;
                col = next_node.second;
        } while (cellDetails[row][col].parent != next_node);
        Path.emplace(row, col);
        while (!Path.empty()) {
                Pair p = Path.top();
                Path.pop();
                printf("-> (%d,%d) ", p.first, p.second);
        }
}
template <size_t ROW, size_t COL>
void aStarSearch(const array<array<int, COL>, ROW>& grid,
                                const Pair& src, const Pair& dest)
{
                if (!isValid(grid, src)) {
                printf("Source is invalid\n");
                return;
        }
                if (!isValid(grid, dest)) {
                printf("Destination is invalid\n");
                return;
        }       if (!isUnBlocked(grid, src)
                || !isUnBlocked(grid, dest)) {
                printf("Source or the destination is blocked\n");
                return;
        }
        if (isDestination(src, dest)) {
                printf("We are already at the destination\n");
                return;
        }
        bool closedList[ROW][COL];
        memset(closedList, false, sizeof(closedList));
        array<array<cell, COL>, ROW> cellDetails;
        int i, j;
        i = src.first, j = src.second;
        cellDetails[i][j].f = 0.0;
        cellDetails[i][j].g = 0.0;
        cellDetails[i][j].h = 0.0;
        cellDetails[i][j].parent = { i, j };
```

```
        std::priority_queue<Tuple, std::vector<Tuple>,
        std::greater<Tuple> >
                openList;
                openList.emplace(0.0, i, j);
                while (!openList.empty()) {
                const Tuple& p = openList.top();
                i = get<1>(p); // second element of tuple
                j = get<2>(p); // third element of tuple
                openList.pop();
                closedList[i][j] = true;
                for (int add_x = -1; add_x <= 1; add_x++) {
                for (int add_y = -1; add_y <= 1; add_y++) {
                                Pair neighbour(i + add_x, j + add_y);
                                        if (isValid(grid, neighbour)) {
                                        if (isDestination(neighbour,dest))
{
cellDetails[neighbour.first][neighbour.second].parent= { i, j };
printf("The destination cell is ""found\n");
tracePath(cellDetails, dest);
return;
}
else if (!closedList[neighbour.first][neighbour.second]&& isUnBlocked(grid,neighbour)) {
double gNew, hNew, fNew;
gNew = cellDetails[i][j].g + 1.0;
hNew = calculateHValue(neighbour,dest);
fNew = gNew + hNew;
if (cellDetails[neighbour.first][neighbour.second].f== -1|| cellDetails[neighbour.first]
[neighbour.second].f > fNew) {openList.emplace(fNew, neighbour.first,
neighbour.second);
cellDetails[neighbour.first][neighbour.second].g= gNew;
cellDetails[neighbour.first]
[neighbour.second].h= hNew;
cellDetails[neighbour.first][neighbour.second].f= fNew;
cellDetails[neighbour.first][neighbour.second].parent= { i, j };
                                                        }
                                        }
                                }
                        }
                }
        }
        printf("Failed to find the Destination Cell\n");
}
// Driver program to test above function
int main()
{
        /* Description of the Grid-
```

```
        1--> The cell is not blocked
        0--> The cell is blocked */
        array<array<int, 10>, 9> grid{
                { { { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 } },
                { { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1 } },
                { { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 } },
                { { 0, 0, 1, 0, 1, 0, 0, 0, 0, 1 } },
                { { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 } },
                { { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 } },
                { { 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 } },
                { { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 } },
                { { 1, 1, 1, 0, 0, 0, 1, 0, 0, 1 } } }
        };
        // Source is the left-most bottom-most corner
        Pair src(8, 0);
        // Destination is the left-most top-most corner
        Pair dest(0, 0);
        aStarSearch(grid, src, dest);
        return 0;
}
```

Conclusion: Thus, we have learned A star search algorithm for tic-tac-toe game.

Assignment No. 3

Title: Greedy search algorithm.

Problem Definition: Implement Greedy search algorithm for Kruskal's Minimal Spanning Tree Algorithm

Theory: Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function, that is, f(n) = h(n).

Minimum Spanning Tree

For a given connected tree(graph), a spanning tree of that tree is a subtree(subgraph) that connects all the node (Vertices) together without any cycle, while minimum spanning tree is having minimum path cost (Weight) from all possible spanning tree of that graph(tree).

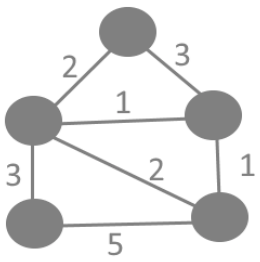
Obs: A spanning tree has (N – 1) paths (Branches/edges) where N is the total number of nodes(Vertices) in the given graph(tree).

Greedy Kruskal's Minimum Spanning Tree Algorithm steps

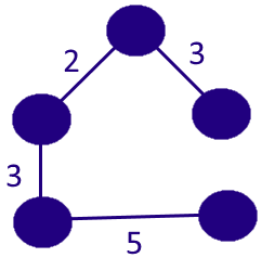
1. Sort all the paths (Branches/edges) with increasing order of their path cost (weight).
2. Pick the path with low path cost(weight). Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this path. Else, discard it.

Here greedy based **Heuristic value (H_{Cnp})** is minimum value path cost of all connected node pair path cost.

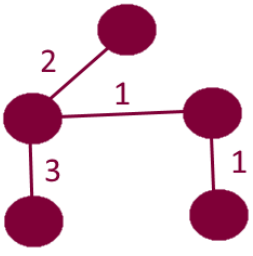
3. Repeat step-2 until there are (N-1) edges in the spanning tree.



Graph



Spanning Tree
Cost = 13



Minimum Spanning
Tree, Cost = 7

Code:

```
Algorithm Greedy (a, n)
{
  Solution : = 0;
 for i = 0 to n do
 {
    x: = select(a);
   if feasible(solution, x)
   {
      Solution: = union(solution , x)
   }
     return solution;
 } }
# Python program for Kruskal's algorithm to find
# Minimum Spanning Tree of a given connected,
# undirected and weighted graph
from collections import defaultdict
# Class to represent a graph
class Graph:
        def __init__(self, vertices):
                self.V = vertices
                self.graph = []

        def addEdge(self, u, v, w):
                self.graph.append([u, v, w])
                def find(self, parent, i):
                if parent[i] == i:
                        return i
                return self.find(parent, parent[i])
        def union(self, parent, rank, x, y):
                xroot = self.find(parent, x)
                yroot = self.find(parent, y)

                if rank[xroot] < rank[yroot]:
                        parent[xroot] = yroot
                elif rank[xroot] > rank[yroot]:
                parent[yroot] = xroot
                else:
                        parent[yroot] = xroot
                        rank[xroot] += 1

        def KruskalMST(self):
                result = [] # This will store the resultant MST
                # An index variable, used for sorted edges
                i = 0
                # An index variable, used for result[]
```

```
                e = 0
                self.graph = sorted(self.graph,
                key=lambda item: item[2])
                parent = []
                rank = []
                # Create V subsets with single elements
                for node in range(self.V):
                        parent.append(node)
                        rank.append(0)
                # Number of edges to be taken is equal to V-1
                while e < self.V - 1:
                        # Step 2: Pick the smallest edge and increment
                        # the index for next iteration
                        u, v, w = self.graph[i]
                        i = i + 1
                        x = self.find(parent, u)
                        y = self.find(parent, v)
                        if x != y:
                        e = e + 1
                        result.append([u, v, w])
                        self.union(parent, rank, x, y)
                        # Else discard the edge
                minimumCost = 0
                print("Edges in the constructed MST")
                for u, v, weight in result:
                        minimumCost += weight
                        print("%d -- %d == %d" % (u, v, weight))
                print("Minimum Spanning Tree", minimumCost)
# Driver's code
if __name__ == '__main__':
        g = Graph(4)
        g.addEdge(0, 1, 10)
        g.addEdge(0, 2, 6)
        g.addEdge(0, 3, 5)
        g.addEdge(1, 3, 15)
        g.addEdge(2, 3, 4)
        # Function call
        g.KruskalMST()
```

Conclusion: Thus, we have learned Greedy search algorithm for Kruskal's Minimal Spanning Tree Algorithm.

Assignment No. 4

Title: Solution for a Constraint Satisfaction Problem (CSP).

Problem Definition: Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem.

Concept: CSP search algorithms take advantage of the structure of states and use general-purpose heuristics rather than problem-specific heuristics to enable the solution of complex problems. Where each state is atomic, or indivisible—a black box with no internal structure.

There are three components: X, D & C

Where

X is a set of variables, $\{X_1, \ldots, X_n\}$.

D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

C is a set of constraints that specify allowable combinations of values.

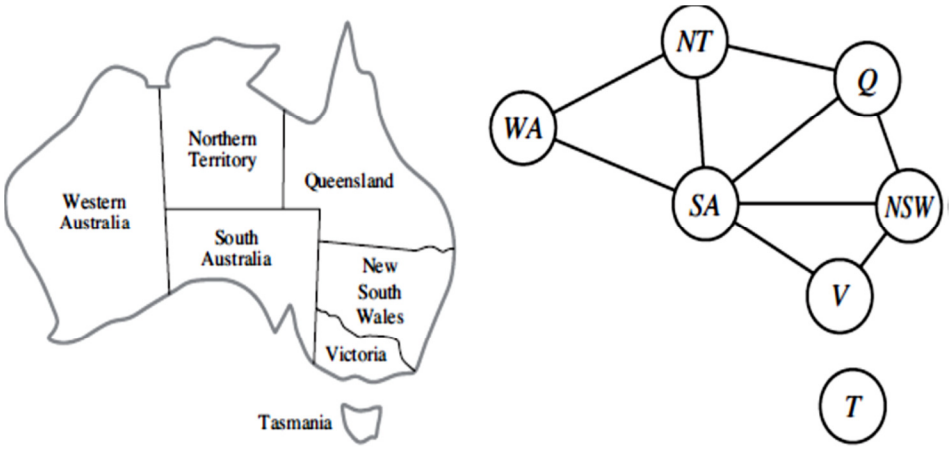
Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i

Here each constraint $\mathbf{C_i}$ consists of a pair **<scope, rel>**

where, scope is a **tuple of variables** that participate in the constraint and **rel** is a relation that defines the values that those variables can take

Example: If X_1 and X_2 both have the domain {A,B}, then the constraint saying the two variables must have **different values** can be written as $<(X_1,X_2), [(A,B), (B,A)]>$ or $<(X_1,X_2), X_1 \neq X_2>$

Example: Graph coloring problem is converted to tree data structure



N queen solution source code:

```
# Python program to solve N Queen
# Problem using backtracking

global N
N = 4

def printSolution(board):
	for i in range(N):
		for j in range(N):
			print (board[i][j],end=' ')
		print()


# A utility function to check if a queen can
# be placed on board[row][col]. Note that this
# function is called when "col" queens are
# already placed in columns from 0 to col -1.
# So we need to check only left side for
# attacking queens
def isSafe(board, row, col):

	# Check this row on left side
	for i in range(col):
		if board[row][i] == 1:
			return False

	# Check upper diagonal on left side
	for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	# Check lower diagonal on left side
	for i, j in zip(range(row, N, 1), range(col, -1, -1)):
		if board[i][j] == 1:
			return False

	return True

def solveNQUtil(board, col):
	# base case: If all queens are placed
	# then return true
	if col >= N:
		return True

	# Consider this column and try placing
```

```
        # this queen in all rows one by one
        for i in range(N):

                if isSafe(board, i, col):
                        # Place this queen in board[i][col]
                        board[i][col] = 1

                        # recur to place rest of the queens
                        if solveNQUtil(board, col + 1) == True:
                                return True

                        # If placing queen in board[i][col
                        # doesn't lead to a solution, then
                        # queen from board[i][col]
                        board[i][col] = 0

        # if the queen can not be placed in any row in
        # this column col then return false
        return False

# This function solves the N Queen problem using
# Backtracking. It mainly uses solveNQUtil() to
# solve the problem. It returns false if queens
# cannot be placed, otherwise return true and
# placement of queens in the form of 1s.
# note that there may be more than one
# solutions, this function prints one of the
# feasible solutions.
def solveNQ():
        board = [ [0, 0, 0, 0],
                        [0, 0, 0, 0],
                        [0, 0, 0, 0],
                        [0, 0, 0, 0]
                        ]

        if solveNQUtil(board, 0) == False:
                print ("Solution does not exist")
                return False

        printSolution(board)
        return True

# driver program to test above function
solveNQ()
```

Conclusion: Thus, we have learned a problem of constraint satisfaction problem.

Assignment No. 5

Title: Development of elementary chatbot.

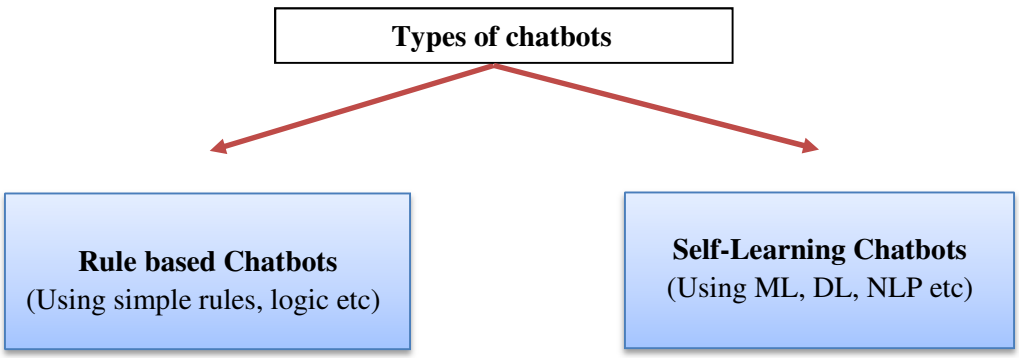
Problem Definition: Develop an elementary simple rule based chatbot for any suitable customer interaction application, using Python

Concept: A chatbot is an AI-based software designed to interact with humans in their natural languages. These chatbots are usually converse via auditory or textual methods, and they can effortlessly mimic human languages to communicate with human beings in a human-like manner. A chatbot is arguably one of the best applications of natural language processing.

Chatbot asks for basic information of customers like name, email address, and the query. If a query is simple like product fault, booking mistake, need some information then without any human connection it can solve it automatically and if some problem is high then it passes the details to the human head.

The Rule-based approach trains a chatbot to answer questions based on a set of pre-determined rules on which it was initially trained. These set rules can either be very simple or very complex. While rule-based chatbots can handle simple queries quite well, they usually fail to process more complicated queries/requests.

As the name suggests, self-learning bots are chatbots that can learn on their own. These leverage advanced technologies like Artificial Intelligence and Machine Learning to train themselves from instances and behaviours. Naturally, these chatbots are much smarter than rule-based bots. Self-learning bots can be further divided into two categories – Retrieval Based or Generative.



Chatbot source code:

```
# Import "chatbot" from
# chatterbot package.
from chatterbot import ChatBot
# Inorder to train our bot, we have
# to import a trainer package
# "ChatterBotCorpusTrainer"
from chatterbot.trainers import ChatterBotCorpusTrainer
# Give a name to the chatbot “corona bot”
# and assign a trainer component.
chatbot=ChatBot('corona bot')
# Create a new trainer for the chatbot
trainer = ChatterBotCorpusTrainer(chatbot)
# Now let us train our bot with multiple corpus
trainer.train("chatterbot.corpus.english.greetings",
"chatterbot.corpus.english.conversations" )
response = chatbot.get_response('What is your Number')
print(response)
response = chatbot.get_response('Who are you?')
print(response)
```

Conclusion: Thus, we have learned elementary implementation of chatbots.

IS Assignment No.1

1.1 Title:

Perform Encryption and Decryption using method of transposition technique.

1.2 Learning Objectives:

Learn how Perform Encryption and Decryption using method of transposition technique.

1.3 Problem Definition:

Perform Encryption and Decryption using method of transposition technique.

1.4 Outcomes:

After completion of this assignment students will be able to understand the Perform Encryption and Decryption using method of transposition technique.

1.5 Software Requirements:

Python 3

1.6 Hardware Requirements:

PC, 2GB RAM, 500 GB HDD

1.7 Theory Concepts:

The rail fence is the simplest example of a class of transposition ciphers, known as route ciphers. In general, the elements of the plaintext (usually single letters) are written in a prearranged order (route) into a geometric array (matrix)—typically a rectangle—agreed upon in advance by the transmitter and receiver and then read off by following another prescribed route through the matrix to produce the cipher. The key in a route cipher consists of keeping secret the geometric array, the starting point, and the routes. Clearly both the matrix and the routes can be much more complex than in this example; but even so, they provide little security. One form of transposition (permutation) that was widely used depends on an easily remembered key word for identifying the route in which the columns of a rectangular matrix are to be read. For example, using the key word AUTHOR and ordering the columns by the lexicographic order of the letters in the key word. In decrypting a route cipher, the receiver enters the cipher text symbols into the agreed-upon matrix according to the encryption route and then reads the plaintext according to

the original order of entry. A significant improvement in crypto security can be achieved by re encrypting the cipher obtained from one transposition with another transposition. Because the result (product) of two transpositions is also a transposition, the effect of multiple transpositions is to define a complex route in the matrix, which in itself would be difficult to describe by any simple mnemonic.

ALGORITHM DESCRIPTION:

In the rail fence cipher, the plaintext is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

1. Generate numerical key from the word key by the characters of the word in alphabetical order.
2. Encryption

i. The plain text is written in the matrix form, where the column of the matrix is number of characters in the word key and row of the matrix is to accommodate the characters of the plain text and the space left after the plain text characters in the last row is filled with any character. (eg. x or z)

ii. The cipher text is generated by reading the characters column by column in the order specified in the numerical key.

3. Decryption

i. The characters in the cipher text are filled in the matrix of same order used for encryption, but in the order specified in the key. The characters from cipher text equal to the number of rows in matrix are taken and filled in the matrix column based on the order specified in the key

ii. The plain text is generated from cipher text by reading the characters from the matrix row by row.

4. Stop

Sample Code:

```
# Python3 implementation of
# Columnar Transposition
import math

key = "HACK"

# Encryption
def encryptMessage(msg):
	cipher = ""

	# track key indices
	k_indx = 0

	msg_len = float(len(msg))
	msg_lst = list(msg)
	key_lst = sorted(list(key))

	# calculate column of the matrix
	col = len(key)

	# calculate maximum row of the matrix
	row = int(math.ceil(msg_len / col))

	# add the padding character '_' in empty
	# the empty cell of the matix
	fill_null = int((row * col) - msg_len)
	msg_lst.extend('_' * fill_null)

	# create Matrix and insert message and
	# padding characters row-wise
	matrix = [msg_lst[i: i + col]
			for i in range(0, len(msg_lst), col)]

	# read matrix column-wise using key
	for _ in range(col):
		curr_idx = key.index(key_lst[k_indx])
		cipher += ''.join([row[curr_idx]
						for row in matrix])
		k_indx += 1

	return cipher

# Decryption
```

```
def decryptMessage(cipher):
	msg = ""

	# track key indices
	k_indx = 0

	# track msg indices
	msg_indx = 0
	msg_len = float(len(cipher))
	msg_lst = list(cipher)

	# calculate column of the matrix
	col = len(key)

	# calculate maximum row of the matrix
	row = int(math.ceil(msg_len / col))

	# convert key into list and sort
	# alphabetically so we can access
	# each character by its alphabetical position.
	key_lst = sorted(list(key))

	# create an empty matrix to
	# store deciphered message
	dec_cipher = []
	for _ in range(row):
		dec_cipher += [[None] * col]

	# Arrange the matrix column wise according
	# to permutation order by adding into new matrix
	for _ in range(col):
		curr_idx = key.index(key_lst[k_indx])

		for j in range(row):
			dec_cipher[j][curr_idx] = msg_lst[msg_indx]
			msg_indx += 1
		k_indx += 1

	# convert decrypted msg matrix into a string
	try:
		msg = ''.join(sum(dec_cipher, []))
	except TypeError:
		raise TypeError("This program cannot",
					"handle repeating words.")

	null_count = msg.count('_')
```

```
        if null_count > 0:
                return msg[: -null_count]

        return msg

# Driver Code
msg = "Geeks for Geeks"

cipher = encryptMessage(msg)
print("Encrypted Message: {}".
                        format(cipher))

print("Decryped Message: {}".
        format(decryptMessage(cipher)))
```

Conclusion: Thus we learn that how to Perform Encryption and Decryption using method of transposition technique.

IS Assignment 2

2.1 Title: Implementation of DES (Data Encryption Standard) Algorithm

2.2 Learning Objectives:

Learn Data Encryption Standard Algorithm (DES)

2.3 Problem Definition:

Implementation of S-DES

2.4 Outcomes:

After completion of this assignment students will be able to understand the Data Encryption Standard.

2.5 Software Requirements:

Python 3

2.6 Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

2.7 Theory Concepts:

Data Encryption Standard (DES)

The Data Encryption Standard (DES) is a Symmetric-key block cipher issued by the national Institute of Standards & Technology (NIST).

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration –

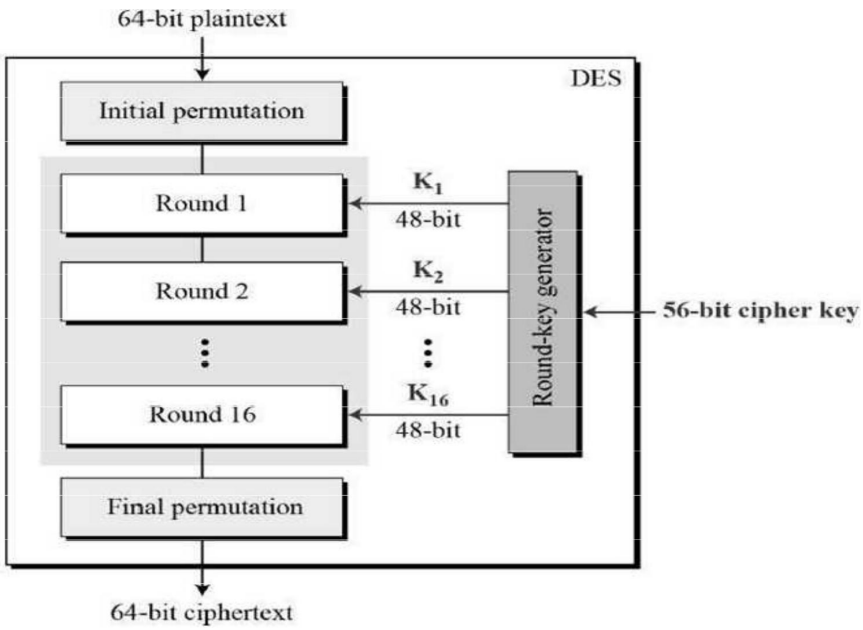


Figure 2.1: General Structure of DES

Since DES is based on the Feistel Cipher, all that is required to specify DES is –

- Round function
- Key schedule
- Any additional processing – Initial and final permutation

2.7.1 Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows

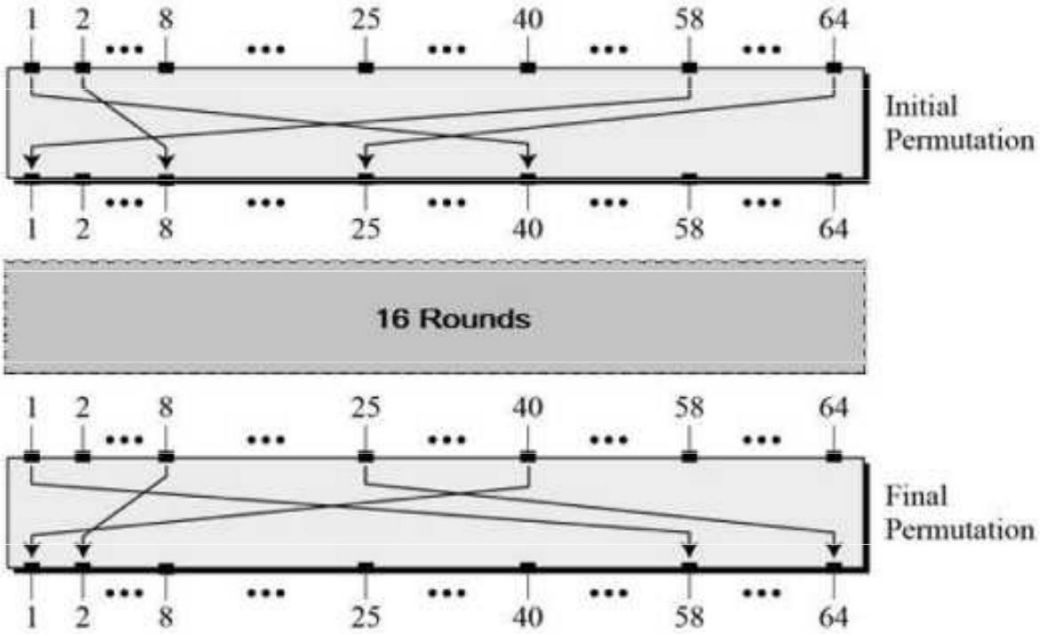
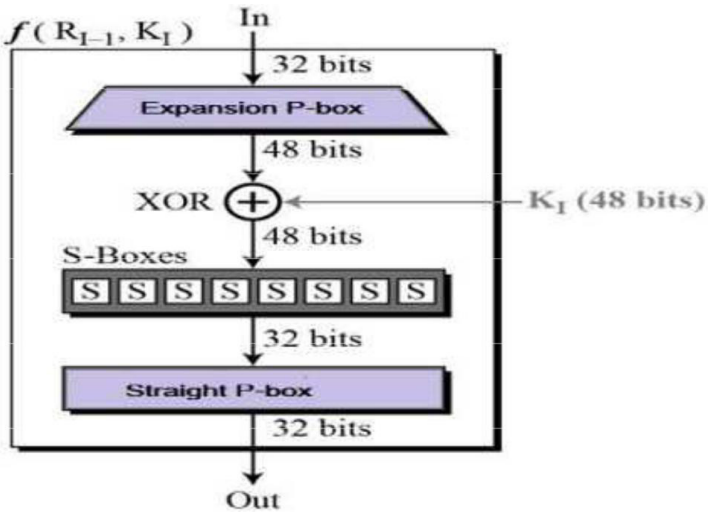


Figure 2.2 initial and final permutations

2.7.2 Round Function

The heart of this cipher is the DES function, *f*. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



igure 3.3 Round Functions

2.7.3 Expansion Permutation Box

Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration:

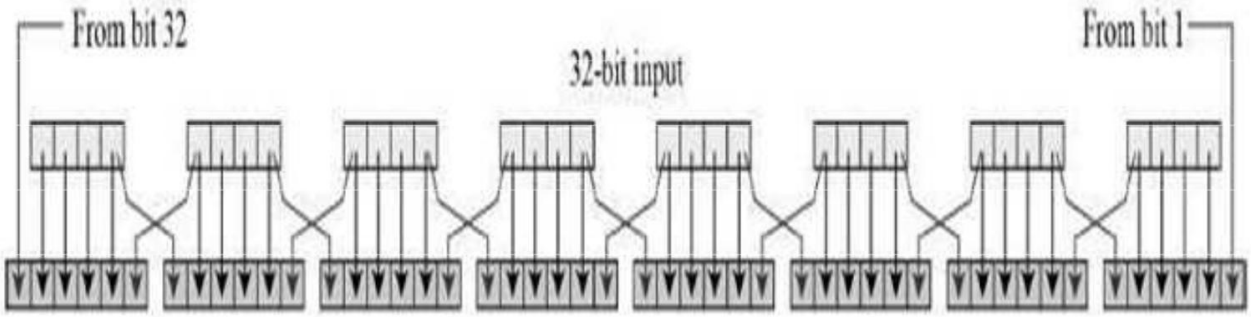


Figure 2.4 Permutation logic

The graphically depicted Permutation logic is generally described as table in DES specification illustrated as shown:

Table 2.1 Permutation logic

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

2.7.4 XOR(Whitener)

After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.

2.7.5 Substitution Boxes

The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration –

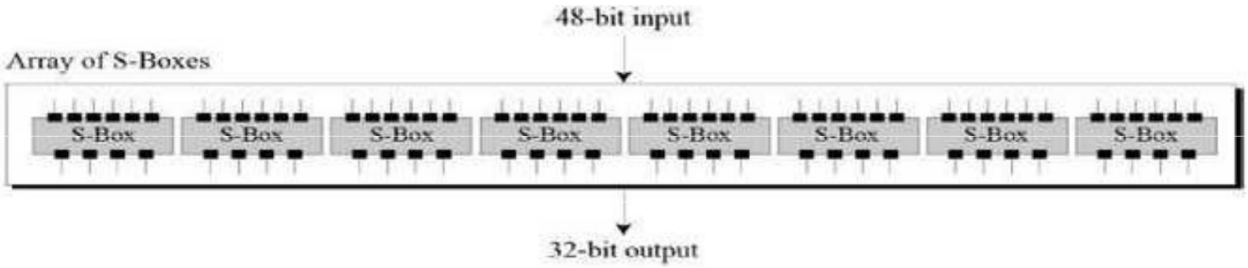


Figure 2.5 S-Boxes

The S-box rule is illustrated below –

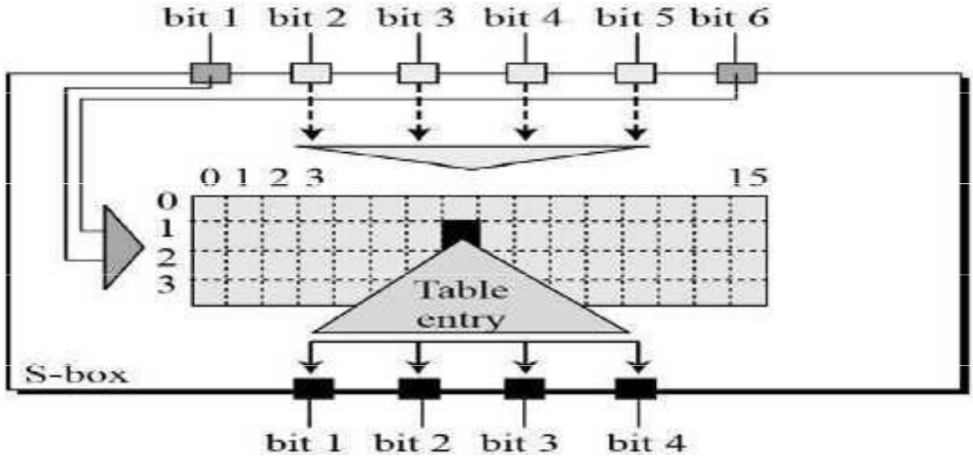


Figure 3.6 S-Box Rules

There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.

2.7.6 Straight Permutation – The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

Table 1.2 Straight Permutation

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

2.7.7 Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration –

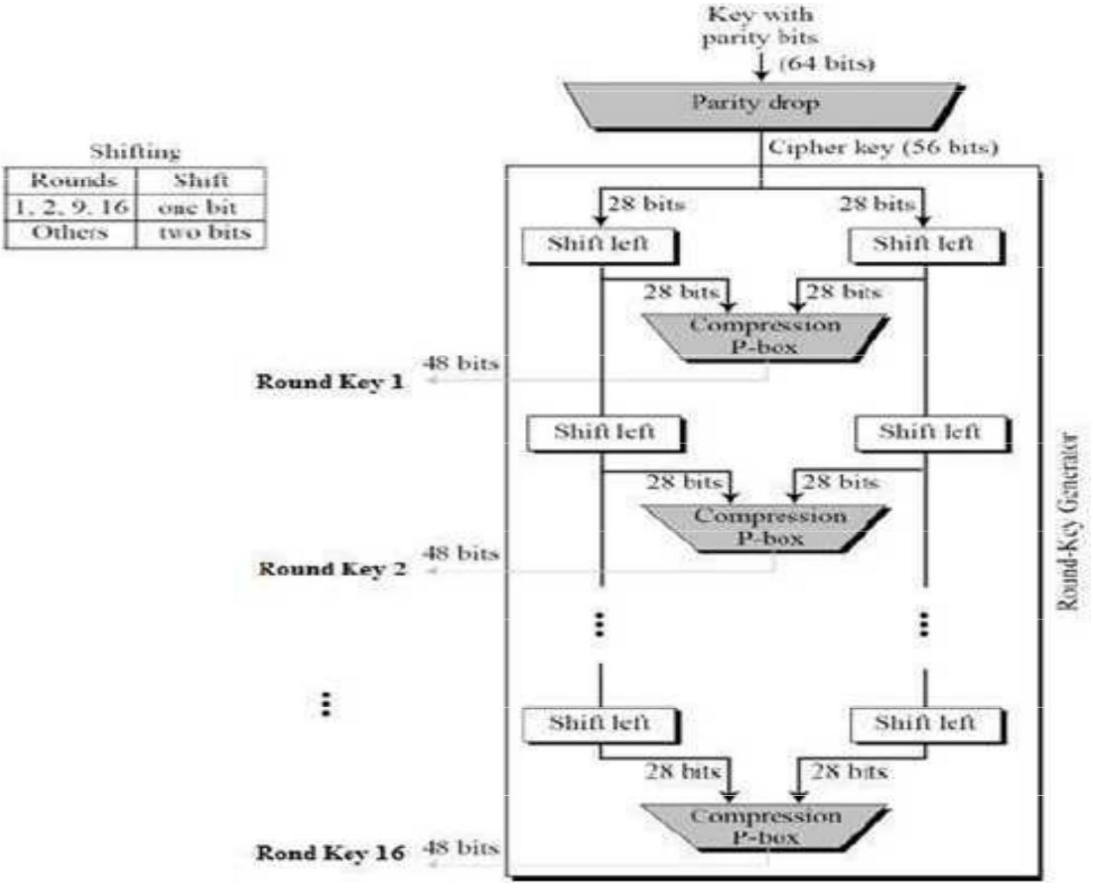


Figure 2.7 the process of key generation

The logic for Parity drops, shifting, and Compression P-box is given in the DES description.

2.7.8 DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect –** A small change in plaintext results in the very great change in the cipher text.
- **Completeness –** Each bit of cipher text depends on many bits of plaintext.

During the last few years, cryptanalysis has found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

Sample Code:-

```
# Python3 code for the above approach


# Hexadecimal to binary conversion



def hex2bin(s):
        mp = {'0': "0000",
                '1': "0001",
                '2': "0010",
                '3': "0011",
                '4': "0100",
                '5': "0101",
                '6': "0110",
                '7': "0111",
                '8': "1000",
                '9': "1001",
                'A': "1010",
                'B': "1011",
                'C': "1100",
                'D': "1101",
                'E': "1110",
                'F': "1111"}
        bin = ""
        for i in range(len(s)):
                bin = bin + mp[s[i]]
        return bin


# Binary to hexadecimal conversion
```

```
def bin2hex(s):
        mp = {"0000": '0',
                "0001": '1',
                "0010": '2',
                "0011": '3',
                "0100": '4',
                "0101": '5',
                "0110": '6',
                "0111": '7',
                "1000": '8',
                "1001": '9',
                "1010": 'A',
                "1011": 'B',
                "1100": 'C',
                "1101": 'D',
                "1110": 'E',
                "1111": 'F'}
        hex = ""
        for i in range(0, len(s), 4):
                ch = ""
                ch = ch + s[i]
                ch = ch + s[i + 1]
                ch = ch + s[i + 2]
                ch = ch + s[i + 3]
                hex = hex + mp[ch]

        return hex

# Binary to decimal conversion
```

```
def bin2dec(binary):

        binary1 = binary
        decimal, i, n = 0, 0, 0
        while(binary != 0):
                dec = binary % 10
                decimal = decimal + dec * pow(2, i)
                binary = binary//10
                i += 1
        return decimal

# Decimal to binary conversion


def dec2bin(num):
        res = bin(num).replace("0b", "")
        if(len(res) % 4 != 0):
                div = len(res) / 4
                div = int(div)
                counter = (4 * (div + 1)) - len(res)
                for i in range(0, counter):
                        res = '0' + res
        return res

# Permute function to rearrange the bits


def permute(k, arr, n):
        permutation = ""
        for i in range(0, n):
                permutation = permutation + k[arr[i] - 1]
```

```
        return permutation

# shifting the bits towards left by nth shifts


def shift_left(k, nth_shifts):
        s = ""
        for i in range(nth_shifts):
                for j in range(1, len(k)):
                        s = s + k[j]
                s = s + k[0]
                k = s
                s = ""
        return k

# calculating xow of two strings of binary number a and b


def xor(a, b):
        ans = ""
        for i in range(len(a)):
                if a[i] == b[i]:
                        ans = ans + "0"
                else:
                        ans = ans + "1"
        return ans


# Table of Position of 64 bits at initial level: Initial Permutation Table
initial_perm = [58, 50, 42, 34, 26, 18, 10, 2,
                                60, 52, 44, 36, 28, 20, 12, 4,
```

```
                    62, 54, 46, 38, 30, 22, 14, 6,
                    64, 56, 48, 40, 32, 24, 16, 8,
                    57, 49, 41, 33, 25, 17, 9, 1,
                    59, 51, 43, 35, 27, 19, 11, 3,
                    61, 53, 45, 37, 29, 21, 13, 5,
                    63, 55, 47, 39, 31, 23, 15, 7]

# Expansion D-box Table
exp_d = [32, 1, 2, 3, 4, 5, 4, 5,
            6, 7, 8, 9, 8, 9, 10, 11,
            12, 13, 12, 13, 14, 15, 16, 17,
            16, 17, 18, 19, 20, 21, 20, 21,
            22, 23, 24, 25, 24, 25, 26, 27,
            28, 29, 28, 29, 30, 31, 32, 1]

# Straight Permutation Table
per = [16, 7, 20, 21,
       29, 12, 28, 17,
       1, 15, 23, 26,
       5, 18, 31, 10,
       2, 8, 24, 14,
       32, 27, 3, 9,
       19, 13, 30, 6,
       22, 11, 4, 25]

# S-box Table
sbox = [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
            [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
            [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
            [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13]],
```

[[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
[3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
[0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
[13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9]],

[[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
[13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
[13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
[1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12]],

[[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
[13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
[10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
[3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14]],

[[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
[14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
[4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
[11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3]],

[[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
[10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
[9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
[4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13]],

[[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
[13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
[1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
[6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12]],

[[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],

```
                [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
                [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
                [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11]]]

# Final Permutation Table
final_perm = [40, 8, 48, 16, 56, 24, 64, 32,
                        39, 7, 47, 15, 55, 23, 63, 31,
                        38, 6, 46, 14, 54, 22, 62, 30,
                        37, 5, 45, 13, 53, 21, 61, 29,
                        36, 4, 44, 12, 52, 20, 60, 28,
                        35, 3, 43, 11, 51, 19, 59, 27,
                        34, 2, 42, 10, 50, 18, 58, 26,
                        33, 1, 41, 9, 49, 17, 57, 25]


def encrypt(pt, rkb, rk):
        pt = hex2bin(pt)

        # Initial Permutation
        pt = permute(pt, initial_perm, 64)
        print("After initial permutation", bin2hex(pt))

        # Splitting
        left = pt[0:32]
        right = pt[32:64]
        for i in range(0, 16):
                # Expansion D-box: Expanding the 32 bits data into 48 bits
                right_expanded = permute(right, exp_d, 48)

                # XOR RoundKey[i] and right_expanded
                xor_x = xor(right_expanded, rkb[i])
```

```
            # S-boxex: substituting the value from s-box table by calculating row and
column
            sbox_str = ""
            for j in range(0, 8):
                row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
                col = bin2dec(
                    int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] + xor_x[j * 6 + 3] +
xor_x[j * 6 + 4]))
                val = sbox[j][row][col]
                sbox_str = sbox_str + dec2bin(val)

            # Straight D-box: After substituting rearranging the bits
            sbox_str = permute(sbox_str, per, 32)

            # XOR left and sbox_str
            result = xor(left, sbox_str)
            left = result

            # Swapper
            if(i != 15):
                left, right = right, left
            print("Round ", i + 1, " ", bin2hex(left),
                " ", bin2hex(right), " ", rk[i])

        # Combination
        combine = left + right

        # Final permutation: final rearranging of bits to get cipher text
        cipher_text = permute(combine, final_perm, 64)
        return cipher_text
```

```
pt = "123456ABCD132536"
key = "AABB09182736CCDD"

# Key generation
# --hex to binary
key = hex2bin(key)

# --parity bit drop table
keyp = [57, 49, 41, 33, 25, 17, 9,
                1, 58, 50, 42, 34, 26, 18,
                10, 2, 59, 51, 43, 35, 27,
                19, 11, 3, 60, 52, 44, 36,
                63, 55, 47, 39, 31, 23, 15,
                7, 62, 54, 46, 38, 30, 22,
                14, 6, 61, 53, 45, 37, 29,
                21, 13, 5, 28, 20, 12, 4]

# getting 56 bit key from 64 bit using the parity bits
key = permute(key, keyp, 56)

# Number of bit shifts
shift_table = [1, 1, 2, 2,
                        2, 2, 2, 2,
                        1, 2, 2, 2,
                        2, 2, 2, 1]

# Key- Compression Table : Compression of key from 56 bits to 48 bits
key_comp = [14, 17, 11, 24, 1, 5,
                        3, 28, 15, 6, 21, 10,
```

```
                    23, 19, 12, 4, 26, 8,
                    16, 7, 27, 20, 13, 2,
                    41, 52, 31, 37, 47, 55,
                    30, 40, 51, 45, 33, 48,
                    44, 49, 39, 56, 34, 53,
                    46, 42, 50, 36, 29, 32]

# Splitting
left = key[0:28] # rkb for RoundKeys in binary
right = key[28:56] # rk for RoundKeys in hexadecimal

rkb = []
rk = []
for i in range(0, 16):
        # Shifting the bits by nth shifts by checking from shift table
        left = shift_left(left, shift_table[i])
        right = shift_left(right, shift_table[i])

        # Combination of left and right string
        combine_str = left + right

        # Compression of key from 56 to 48 bits
        round_key = permute(combine_str, key_comp, 48)

        rkb.append(round_key)
        rk.append(bin2hex(round_key))

print("Encryption")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ", cipher_text)
```

```
print("Decryption")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ", text)
```

Conclusion: Thus we learn that to how to Encrypt and Decrypt the message by using DES Algorithm.

IS Assignment 3

3.1 Title: Implementation of AES (Advanced Encryption Standard) Algorithm

3.2 Learning Objectives:

Learn How to Apply Advanced Encryption Standard Algorithm to encryption of given data.

3.3 Problem Definition:

Implementation of S-AES

3.4 Software Requirements:

Python 3

3.5 Hardware Requirement:

PIV, 2GB RAM, 500 GB HDD

3.6 Outcomes:

After completion of this assignment students will be able Implement code for **Advanced Encryption Standard Algorithm** for given data and find the encrypted data of the given data.

3.7 Theory Concepts:

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six times faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows –

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C ,Java and Python

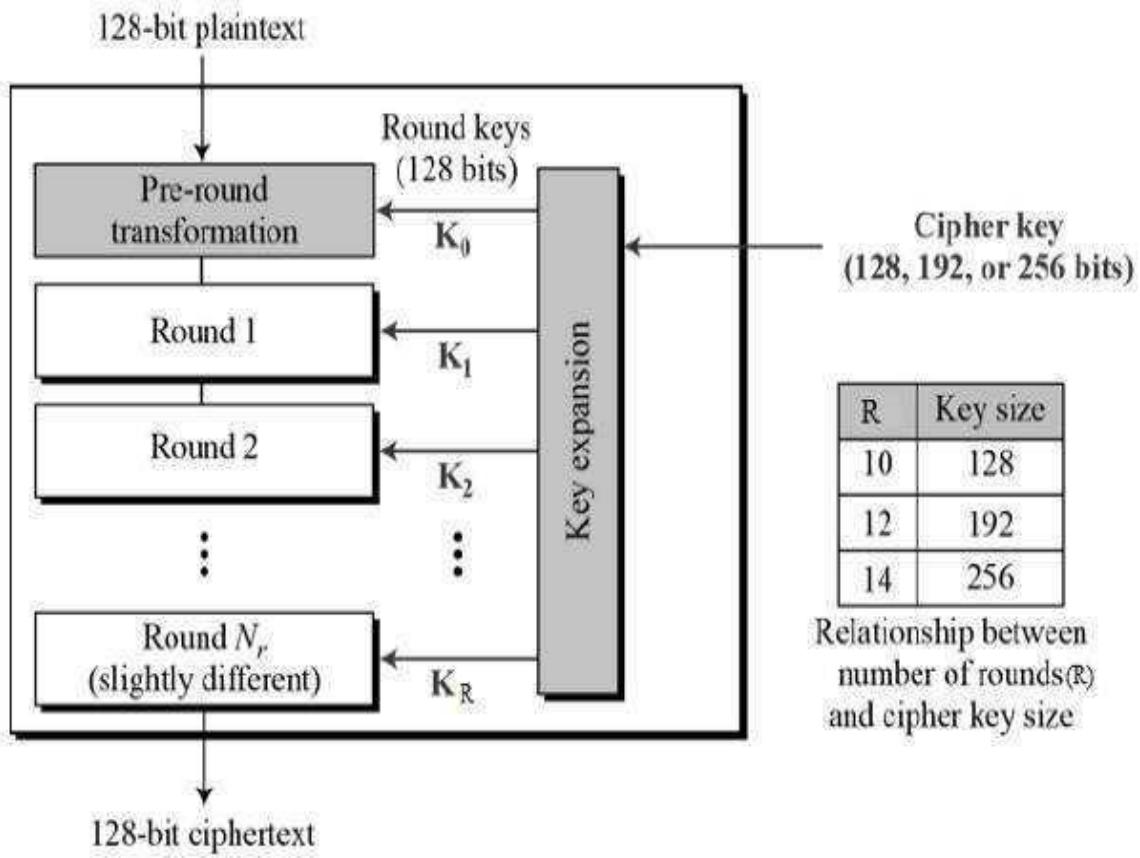
3.7.1 Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix –

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration –



R	Key size
10	128
12	192
14	256

Figure 3.1 AES structure

3.7.2 Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below –

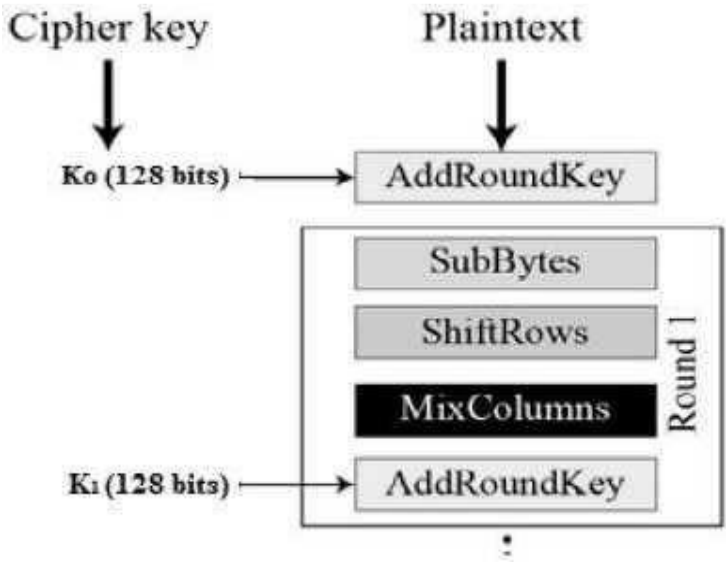


Figure 3.2 Encryption Process

Sample Code:

```
def encrypt(self, plain_text):
        plain_text = self.__pad(plain_text)
        iv = Random.new().read(self.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        encrypted_text = cipher.encrypt(plain_text.encode())
        return b64encode(iv + encrypted_text).decode("utf-8")

def decrypt(self, encrypted_text):
        encrypted_text = b64decode(encrypted_text)
        iv = encrypted_text[:self.block_size]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        plain_text = cipher.decrypt(encrypted_text[self.block_size:]).decode("utf-8")
        return self.__unpad(plain_text)
import hashlib
from Crypto import Random
from Crypto.Cipher import AES
from base64 import b64encode, b64decode

class AESCipher(object):
    def __init__(self, key):
        self.block_size = AES.block_size
        self.key = hashlib.sha256(key.encode()).digest()
```

```
def encrypt(self, plain_text):
    plain_text = self.__pad(plain_text)
    iv = Random.new().read(self.block_size)
    cipher = AES.new(self.key, AES.MODE_CBC, iv)
    encrypted_text = cipher.encrypt(plain_text.encode())
    return b64encode(iv + encrypted_text).decode("utf-8")

def decrypt(self, encrypted_text):
    encrypted_text = b64decode(encrypted_text)
    iv = encrypted_text[:self.block_size]
    cipher = AES.new(self.key, AES.MODE_CBC, iv)
    plain_text = cipher.decrypt(encrypted_text[self.block_size:]).decode("utf-8")
    return self.__unpad(plain_text)

def __pad(self, plain_text):
    number_of_bytes_to_pad = self.block_size - len(plain_text) % self.block_size
    ascii_string = chr(number_of_bytes_to_pad)
    padding_str = number_of_bytes_to_pad * ascii_string
    padded_plain_text = plain_text + padding_str
    return padded_plain_text


def __unpad(plain_text):
    last_character = plain_text[len(plain_text) - 1:]
    return plain_text[:-ord(last_character)]
```

Conclusion:

Thus we learn that to how to encrypt and Decrypt the message by using AES Algorithm.

IS ASSIGNMENT 4

4.1 Title: Implementation of RSA Algorithm

4.2 Learning Objectives:

Learn RSA Algorithm

4.3 Problem Definition:

Implementation of RSA Algorithm

4.4 Software Requirements:

Python

4.5 Hardware Requirement:

2GB RAM, 500 GB HDD

4.6 Outcomes:

After completion of this assignment students will be able to understand the How to encrypt and decrypt messages.

4.7 Theory Concepts:

4.7.1 RSA(Rivest, Shamir & Adleman)

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of the keys can be given to anyone. The other key must be kept private. The algorithm is based on the fact that finding the factors of a large composite number is difficult: when the integers are prime numbers, the problem is called prime factorization. It is also a key pair (public and private key) generator.

- RSA makes the public and prívate keys by multiplying two large prime numbers *p* and *q*

- It's easy to find & multiply large prime No. ($n=pq$)
- It is very difficult to factor the number n to find p and q
- Finding the private key from the public key would require a factoring operation
- The real challenge is the selection & generation of keys.

- RSA is complex and slow, but secure
- 100 times slower than DES on s/w & 1000 times on h/w

The Rivest-Shamir-Adleman (RSA) algorithm is one of the most popular and secures public-key encryption methods. The algorithm capitalizes on the fact that there is no efficient way to factor very large (100-200 digit) numbers.

Using an encryption key (e,n), the algorithm is as follows:

1. Represent the message as an integer between 0 and (n-1). Large messages can be broken up into a number of blocks. Each block would then be represented by an integer in the same range.
2. Encrypt the message by raising it to the eth power modulo n. The result is a ciphertext message C.
3. To decrypt ciphertext message C, raise it to another power d modulo n

The encryption key (e,n) is made public. The decryption key (d,n) is kept private by the user.

How to Determine Appropriate Values for *e*, *d*, and *n*

1. Choose two very large (100+ digit) prime numbers. Denote these numbers as p and q.
2. Set n equal to $p * q$.
3. Choose any large integer, d, such that GCD(d, ((p-1) * (q-1))) = 1
4. Find e such that $e * d = 1$ (**mod** ((p-1) * (q-1)))

Rivest, Shamir, and Adleman provide efficient algorithms for each required operation.

4.7.2. How secure is a communication using RSA?

Cryptographic methods cannot be proven secure. Instead, the only test is to see if someone can figure out how to decipher a message without having direct knowledge of the decryption key. The RSA method's security rests on the fact that it is extremely difficult to factor very large numbers. If 100 digit numbers are used for p and q, the resulting n will be approximately 200 digits. The fastest known factoring algorithm would take far too long for an attacker to ever break the code. Other methods for determining d without factoring n are equally as difficult.

Any cryptographic technique which can resist a concerted attack is regarded as secure. At this point in time, the RSA algorithm is considered secure.

4.7.3. How Does RSA Works?

RSA is an **asymmetric** system, which means that a key pair will be generated (we will see how soon) , a **public** key and a **private** key , obviously you keep your private key secure and pass around the public one.

The algorithm was published in the 70's by Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman, hence RSA, and it sort of implement's a trapdoor function such as Diffie's one.

RSA is rather slow so it's hardly used to encrypt data, more frequently it is used to encrypt and pass around **symmetric** keys which can actually deal with encryption at a **faster** speed.

4.7.4. RSA Security:

- It uses prime number theory which makes it difficult to find out the key by reverse engineering.
- Mathematical Research suggests that it would take more than 70 years to find P & Q if N is a 100 digit number.

4.8. Algorithm

The RSA algorithm holds the following features –

- RSA algorithm is a popular exponentiation in a finite field over integers including prime numbers.
- The integers used by this method are sufficiently large making it difficult to solve.

- There are two sets of keys in this algorithm: private key and public key.

You will have to go through the following steps to work on RSA algorithm –

Step 1: Generate the RSA modulus

The initial procedure begins with selection of two prime numbers namely p and q, and then calculating their product N, as shown –

N=p*q

Here, let N be the specified large number.

Step 2: Derived Number (e)

Consider number e as a derived number which should be greater than 1 and less than (p-1) and (q-1). The primary condition will be that there should be no common factor of (p-1) and (q-1) except 1

Step 3: Public key

The specified pair of numbers **n** and **e** forms the RSA public key and it is made public.

Step 4: Private Key

Private Key **d** is calculated from the numbers p, q and e. The mathematical relationship between the numbers is as follows –

ed = 1 mod (p-1) (q-1)

The above formula is the basic formula for Extended Euclidean Algorithm, which takes p and q as the input parameters.

4.8.1. Encryption Formula

Consider a sender who sends the plain text message to someone whose public key is **(n,e).** To encrypt the plain text message in the given scenario, use the following syntax –

C = Pe mod n

4.8.2. Decryption Formula

The decryption process is very straightforward and includes analytics for calculation in a systematic approach. Considering receiver **C** has the private key **d**, the result modulus will be calculated as –

Plaintext = Cd mod n

Example

1. P=7, Q=17
2. 119=7*17
3. (7-1)*(17-1)= 6*16 =96 factor 2 & 3, so E=5
4. (D*5) mod (7-1)*(17-1)=1, so D=77
5. CT=10^5 mod 119 =100000 mod 119 =40
6. Send 40
7. PT=40^{77} mod 119 = 10

Sample Code

```
# Python for RSA asymmetric cryptographic algorithm.
# For demonstration, values are
# relatively small compared to practical application
import math


def gcd(a, h):
        temp = 0
        while(1):
                temp = a % h
                if (temp == 0):
                        return h
                a = h
                h = temp


p = 3
q = 7
n = p*q
e = 2
phi = (p-1)*(q-1)
```

```
while (e < phi):

        # e must be co-prime to phi and
        # smaller than phi.
        if(gcd(e, phi) == 1):
                break
        else:
                e = e+1

# Private key (d stands for decrypt)
# choosing d such that it satisfies
# d*e = 1 + k * totient

k = 2
d = (1 + (k*phi))/e

# Message to be encrypted
msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n
c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n
m = pow(c, d)
m = math.fmod(m, n)
print("Original Message Sent = ", m)
```

Conclusion

Thus we learn that to how to Encrypt and Decrypt the message by using RSA Algorithm.

IS assignment 5

5.1 Title: Implementation of Different-Hellman key Exchange (DH)

5.2 Learning Objectives:

Learn Different-Hellman key Exchange (DH)

5.3 Problem Definition:

Implementation of Different-Hellman key Exchange (DH)

5.4 Software Requirements:

Python 3

5.5 Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

5.6 Outcomes:

After completion of this assignment students will be able to understand the Different-Hellman key Exchange

5.7 Theory Concepts:

5.7.1 Different-Hellman key Exchange (DH)

In the mid- 1970's, Whitefield Different, a student at the Stanford University met with Martin Hellman, his professor &the two began to think about it. After some research & complicated mathematical analysis, they came up with the idea of AKC. Many experts believe that this development is the first & perhaps the only truly revolutionary concept in the history of cryptography.

5.7.2 Silent Features of Different-Hellman key Exchange (DH)

1. Developed to address shortfalls of *key distribution* in symmetric key distribution.
2. A *key exchange algorithm*, not an encryption algorithm
3. Allows two users to share a *secret key* securely over a public network
4. Once the key has been shared Then both parties can use it to encrypt and decrypt messages using symmetric cryptography
5. Algorithm is based on "difficulty of calculating discrete logarithms in a finite field"

6. These keys are mathematically related to each other.
7. ''Using the public key of users, the session key is generated without transmitting the private key of the users.''

5.7.3 Different-Hellman Key Exchange/Agreement Algorithm with Example

1. Firstly, Alice and Bob agree on two large prime numbers, n and g. These two integers need not be kept secret. Alice and Bob can use an insecure channel to agree on them.

   Let n = 11, g = 7.

2. Alice chooses another large random number x, and calculates A such that:
   $A = g^x \bmod n$

   Let x = 3. Then, we have, $A = 7^3 \bmod 11 = 343 \bmod 11 = 2$.

3. Alice sends the number A to Bob.

   Alice sends 2 to Bob.

4. Bob independently chooses another large random integer y and calculates B such that:
   $B = g^y \bmod n$

   Let y = 6. Then, we have, $B = 7^6 \bmod 11 = 117649 \bmod 11 = 4$.

5. Bob sends the number B to Alice.

   Bob sends 4 to Alice.

6. A now computes the secret key K1 as follows:
   $K1 = B^x \bmod n$

   We have, $K1 = 4^3 \bmod 11 = 64 \bmod 11 = 9$.

7. B now computes the secret key K2 as follows:
   $K2 = A^y \bmod n$

   We have, $K2 = 2^6 \bmod 11 = 64 \bmod 11 = 9$.

5.7.4 Different-Hellman Key exchange

1. Public values:
   - large prime p, generator g (primitive root of p)
2. Alice has secret value x, Bob has secret y
3. Discrete logarithm problem: given x, g, and n, find A

4. A B: g^x (mod n)
5. B A: g^y (mod n)
6. Bob computes $(g^x)^y = g^{xy}$(mod n)
7. Alice computes $(g^y)^x = g^{xy}$ (mod n)
8. Symmetric key= g^{xy} (mod n)

5.7.5 Limitation: Vulnerable to "man in the middle" attacks*

5.7.5.1 Man-in-the-Middle Attack:

Alice	Tom	Bob
n = 11, g = 7	n = 11, g = 7	n = 11, g = 7

Figure 5.1 Man-in-the-Middle Attack Part-I

Alice	Tom	Bob
x = 3	x = 8, y = 6	y = 9

Figure 5.2 Man-in-the-Middle Attack Part-II

Alice	Tom	Bob
A = g^x mod n = 7^3 mod 11 = 343 mod 11 = 2	A = g^x mod n = 7^8 mod 11 = 5764801 mod 11 = 9 B = g^y mod n = 7^6 mod 11 = 117649 mod 11 = 4	B = g^y mod n = 7^9 mod 11 = 40353607 mod 11 = 8

Figure 5.3 Man-in-the-Middle Attack Part-III

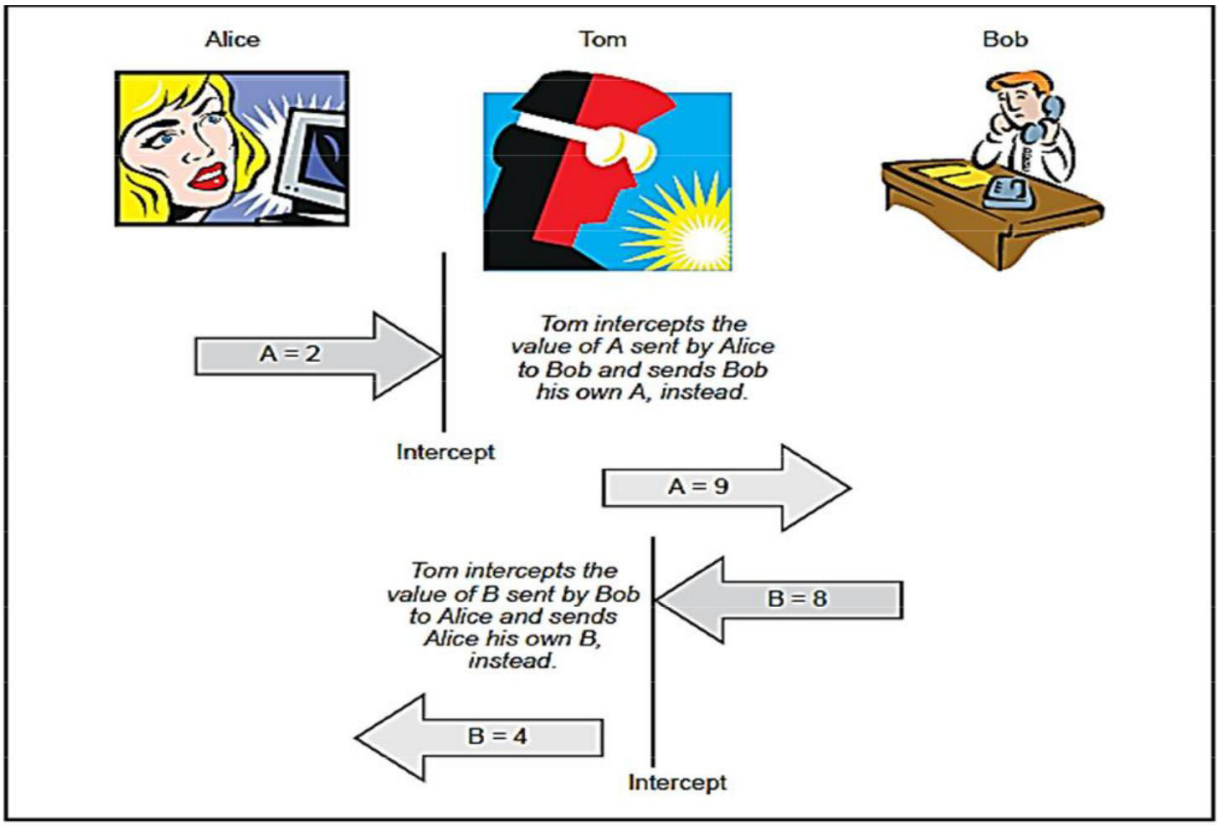


Figure 5.4 Man-in-the-Middle Attack Part-IV

Alice	Tom	Bob
A = 2, B = 4*	A = 2, B = 8	A = 9*, B = 8

(Note: * indicates that these are the values after Tom hijacked and changed them.)

Figure 5.5 Man-in-the-Middle Attack Part-V

Alice	Tom	Bob
$K1 = B^x \bmod n$	$K1 = B^x \bmod n$	$K2 = A^y \bmod n$
$= 4^3 \bmod 11$	$= 8^8 \bmod 11$	$= 9^9 \bmod 11$
$= 64 \bmod 11$	$= 16777216 \bmod 11$	$= 387420489 \bmod 11$
$= 9$	$= 5$	$= 5$
	$K2 = A^y \bmod n$	
	$= 2^6 \bmod 11$	
	$= 64 \bmod 11$	
	$= 9$	

Figure 5.6 Man-in-the-Middle Attack Part-VI

5.7.6 Preventing a Man-in-the-Middle Attack with Hashing

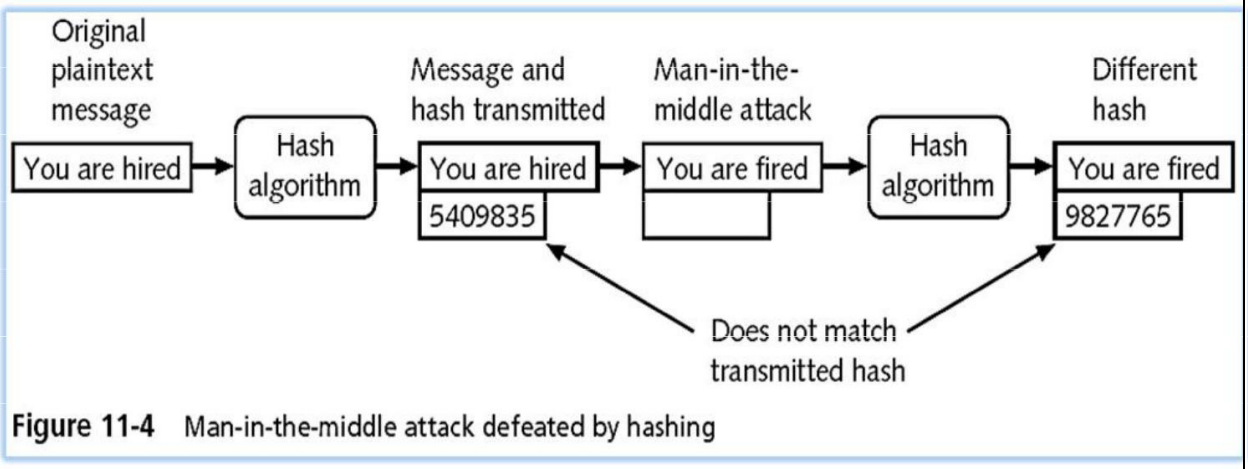


Figure 5.7 Man-in-the-Middle Attack Part-VII

Sample Code:

```
from random import randint
if __name__ == '__main__':

    # Both the persons will be agreed upon the
    # public keys G and P
    # A prime number P is taken
    P = 23

    # A primitive root for P, G is taken
    G = 9


    print('The Value of P is :%d'%(P))
    print('The Value of G is :%d'%(G))

    # Alice will choose the private key a
    a = 4
    print('The Private Key a for Alice is :%d'%(a))

    # gets the generated key
```

```
x = int(pow(G,a,P))

# Bob will choose the private key b
b = 3
print('The Private Key b for Bob is :%d'%(b))

# gets the generated key
y = int(pow(G,b,P))



# Secret key for Alice
ka = int(pow(y,a,P))

# Secret key for Bob
kb = int(pow(x,b,P))

print('Secret key for the Alice is : %d'%(ka))
print('Secret Key for the Bob is : %d'%(kb))
```

Conclusion: Thus we have studied and implement Different-Hellmen key exchange algorithm and how to prevent Man-in-the-Middle Attack

Assignment 1

Aim:

Case study on Amazon EC2 to learn about Amazon EC2 web services.

Objectives:

1. To learn Amazon Web Services.
2. To case study the Amazon EC2.

Software Requirements

Ubuntu 16.04,PHP, MySQL

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

Amazon Elastic Compute Cloud (EC2)

Elastic IP addresses allow you to allocate a static IP address and programmatically assign it to an instance. You can enable monitoring on an Amazon EC2 instance using Amazon CloudWatch2 in order to gain visibility into resource utilization, operational performance and overall demand patterns (including metrics such as CPU utilization, disk reads and writes, and network traffic). You can create Auto-Scaling Group using the Auto-Scaling feature to automatically scale your capacity on certain conditions based on metric that Amazon CloudWatch collects. You can also distribute incoming traffic by creating an elastic load balancer using the Elastic Load Balancing4 service. Amazon Elastic Block Storage (EBS) volumes provide network network-attached persistent storage to Amazon EC2 instances.

Point-in-time consistent snapshots of EBS volumes can be created and stored on Amazon Simple Storage Service (Amazon S3). Amazon S3 is highly durable and distributed data store. With a simple web services interface, you can store and retrieve large amounts of data as objects in buckets (containers) at any time, from anywhere on the web using standard HTTP verbs. Copies of objects can be distributed and cached at 14 edge locations around the world by creating a distribution using Amazon CloudFront service a web service for content delivery (static or streaming content). Amazon

SimpleDB is a web service that provides the core functionality of a database- real-time lookup and **Amazon Route53** is a highly scalable DNS service that allows you manage your DNS records by creating a HostedZone for every domain you would like to manage.

AWS Identity and Access Management (IAM) enable you to create multiple Users with unique security credentials and manage the permissions for each of these Users within your AWS Account. IAM is natively integrated into AWS Services. No service APIs have changed to support IAM, and exiting applications and tools built on top of the AWS service

APIs will continue to work when using IAM.

AWS also offers various payment and billing services that leverages Amazon's payment infrastructure. All AWS infrastructure services offer utility-style pricing that require no long term commitments or contracts. For example, you pay by the hour for Amazon EC2 instance usage and pay by the gigabyte for storage and data transfer in the case of Amazon S3. More information about each of these services and their pay-as-you-go pricing is available on the AWS Website.

Note that using the AWS cloud doesn't require sacrificing the flexibility and control you've grown accustomed to:

You are free to use the programming model, language, or operating system (Windows, OpenSolaris or any flavor of Linux) of your choice.

You are free to pick and choose the AWS products that best satisfy your requirements—you can use any of the services individually or in any combination.

Because AWS provides resizable (storage, bandwidth and computing) resources, you are free to consume as much or as little and only pay for what you consume.

You are free to use the system management tools you've used in the past and extend your datacenter into the cloud.

Conclusion

Thus studied study of Amazon web services: Amazon EC2.

Assignment 2

Aim: Installation and configure Google App Engine.

Objectives:

1. To learn basics of Google App Engine.
2. To install and configure Google App Engine.

Software Requirements:

Ubuntu 16.04

Python

MySQL

Hardware Requirements:

Pentium IV system with latest configuration

Theory:

Google App Engine is Google's platform as a service offering that allows developers and businesses to build and run applications using Google's advanced infrastructure. These applications are required to be written in one of a few supported languages, namely: Java, Python, PHP and Go. It also requires the use of Google query language and that the database used is Google Big Table. Applications must abide by these standards, so applications either must be developed with GAE in mind or else modified to meet the requirements. GAE is a platform, so it provides all of the required elements to run and host Web applications, be it on mobile or Web. Without this all-in feature, developers would have to source their own servers, database software and the APIs that would make all of them work properly together, not to mention the entire configuration that must be done. GAE takes this burden off the developers so they can concentrate on the app front end and functionality, driving better user experience.

Advantages of GAE include:

1. Readily available servers with no configuration requirement
2. Power scaling function all the way down to "free" when resource usage is minimal

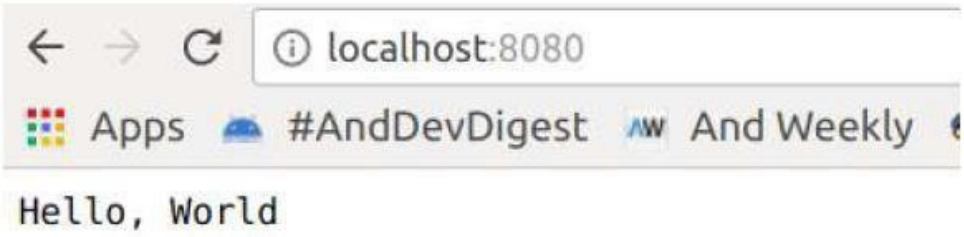
Automated cloud computing tools

1. Make sure you have python installed in your ubuntu system. run the command "*python –V"* and most probably you will get "Python 2.7.6" or above.

2. Crul https://sdk.cloud.google.com and use bash to run the commands by typing this command curl https://sdk.cloud.google.com | bash
3. Whenever you get to choose directories just hit enter, “YEAH IT WILL BE FINE”.
4. Follow the instructions in the installation process.
5. Then run gcloud init
6. Follow the installation instructions as they are very straight forward.
7. Choose the account you want to use for google app engine.
8. Choose the project with numeric choice (don’t use textual, you might make mistake). If you do not already have a google app engine project create a app engine project by following this link. https://console.cloud.google.com/start
9. Enable google api by pressing Y in the command line prompt.

Now as we have finished installing appengine, now it’s time to create and upload an app. In this case we will be taking example of a “HELLO WORLD” app in python.

1. As we already have made sure that we have python installed in our system, It will be easier for us to clone existing code and deploy it rather than creating our own so we will use pythondocs-sample. Run the command “git clone https://github.com/GoogleCloudPlatform/python-docs-samples”.
2. cd to hello world sample by typing the command “ cd python- docssamples/appengine/standard/hello_world”.
3. Then run the command “dev_appserver.py app.yml”. It will run and give you the url of default and admin. If you go to the link of default you see the text hello world like this.



This is how you run the python app in your local server. But what we have to do is hosting the app in google app engine. To do so now let’s follow the following instructions.

1. Run the command *Ctrl + C*.
2. Being in the same working directory hello-world runt the command *gcloud app deploy*
3. Select the project you want to deploy the app , press Y and enter to continue. after that you will get the console output “Deployed service[default] to [Your web url for appengine] ” 4. If you copy and paste the url, you will see the hello world in the browser too.

Web output

Now you have successfully uploaded your web app into app engine.

Conclusion

Hence, we learnt to install and configure Google App Engine

Assignment 3

Title:

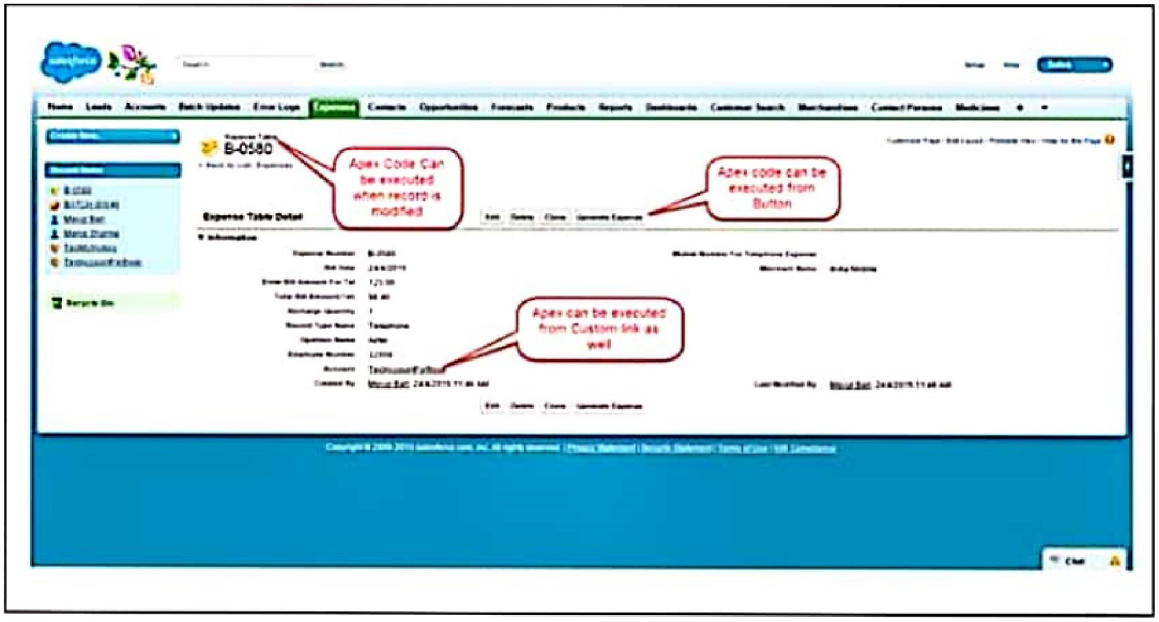
Creating an Application in SalesForce.con using Apex programming Language

Theory:

What is Apex?

Apex is a proprietary language developed by the Salesforce.com. As per the official definition, Apex is a strongly typed, object-oriented programming language that allows developers to execute the flow and transaction control statements on the Force.com platform server in conjunction with calls to the Force.com API.

It has a Java-like syntax and acts like database stored procedures. It enables the developers to add business logic to most system events, including button clicks, related record updates, and Visual force pages. Apex code can be initiated by Web service requests and from triggers on objects. Apex is included in Performance Edition, Unlimited Edition, Enterprise Edition, and Developer Edition.



Features ofApex as a Language

Let us now discuss the features of Apex as a Language —

- Integrated

  Apex has built in support for DML operations like INSERT, UPDATE, DELETE and also DML Exception handling. It has support for inline SOQL and SOSL query handling which returns the set of Object records. We will study the Object, SOQL, SOSL in detail in future chapters.

- Java like syntax and easy to use

  Apex is easy to use as it uses the syntax like Java. For example, variable declaration, loop syntax and conditional statements.

- Strongly Integrated With Data

  Apex is data focused and designed to execute multiple queries and DML statements together. It issues multiple transaction statements on Database.

- Strongly Typed

  Apex is a strongly typed language. It uses direct reference to schema objects like Object and any invalid reference quickly fails if it is deleted or if is of wrong data type.

- Multitenant Environment

  Apex runs in a multitenant environment. Consequently, the Apex runtime engine IS designed to guard closely against runaway code, preventing it from monopolizing shared resources. Any code that violates limits fails with easy-to-understand error messages.

- Upgrades Automatically

  Apex is upgraded as part of Salesforce releases. We don't have to upgrade it manually.

- Easy Testing

  Apex provides built-in support for unit test creation and execution, including test results that indicate how much code is covered, and which parts of your code can be more efficient.

When Should Developer Choœe Apex?

Apex should be used when we are not able to implement the complex business functionality using the pre-built and existing out of the box functionalities. Below are the cases where we need to use apex over Salesforce configuration.

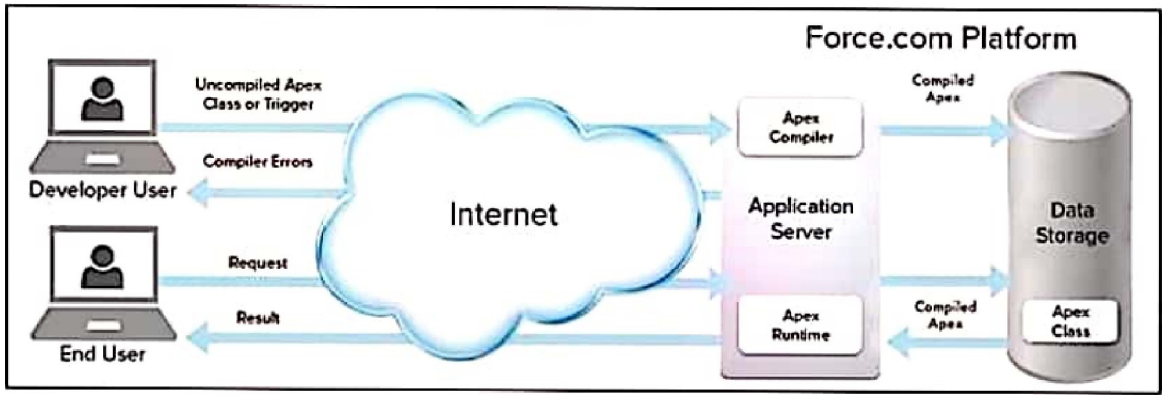
Apex Applications

We can use Apex when we want to _

1. Create Web services with integrating other svstems.
2. Create email services for email blast or email setup.
3. Perform complex validation over multiple objects at the same time and also custom validation implementation.
4. Create complex business processes that are not supported by existing workflow functionality or flows.
5. Create custom transactional logic (logic that occurs over the entire transaction, not just with a single record or object) like using the Database methods for updating the records.
6. Perform some logic when a record is modified or modify the related object's record when there is some event which has caused the trigger to fire.

Working Structure of Apex

As shown in the diagram below (Reference: Salesforce Developer Documentation), Apex runs entirely on demand Force.com Platform



now ofActions

There are two sequence of actions when the developer saves the code and when an end user performs some action which invokes the Apex code as shown below — Developer Action

When a developer writes and saves Apex code to the platform, the platform application server first compiles the code into a set of instructions that can be understood by the Apex runtime interpreter, and then saves those instructions as metadata.

End User Action

When an end-user triggers the execution of Apex, by clicking a button or accessing a Visualforce page, the platform application server retrieves the compiled instructions from the metadata and sends them through the runtime intemreter before returning the result. The enduser observes no differences in execution time as compared to the standard application platform request.

Since Apex is the propnetary language of Salesforce.com, it does not support some features which a general programming language does. Following are a few features which Apex does not support —

- It cannot show the elements in User Interface.

1. You cannot change the standard SFDC provided functionality and also it is not possible to prevent the standard functionality execution.

- You cannot change the standard SFDC provided functionality and also it is not possible to prevent the standard functionality execution.

2. Creating multiple threads is also not possible as we can do it in other languages.

Undustanding the Apex Syntax

Apex code typically contains many things that we might be familiar with from other programming languages.

Variable Declaration

As strongly typed language, you must declare every variable with data type in Apex. As seen In the code below (screenshot below), IstAcc is declared with data type as List of Accounts.

SOQL Query

This will be used to fetch the data from Salesforce database-

Loop Statement

This loop statement is used for iterating over a list or iterating over a piece of code for a specified number of times. In the code shown in the screenshot below, iteration will be same as the number of records we have.

Flow Control Statement

The If statement is used for flow control in this code. Based on certain condition, it is decided whether to go for execution or to stop the execution of the particular piece of code. For example, in the code shown below, it is checking whether the list is empty or it contains records.

DML Statement

Performs the records insert, update, upsert, delete operation on the records in database. For example, the code given below helps in updating Accounts with new field value.

Apex Code Development Tools

In all the editions, we can use any of the foiowing three tools to develop the code —

1. Force.com Developer Console
2. Force.com IDE
   - Code Editor in the Salesforcc User Interface

Conclusion: Hence we created application in SalesForce.com using Apex programming Language

Assignment 4

Title:

Design and develop custom Application (Mini Project) using Salesforce Cloud.

Theory:

Introduction

Salesforce.com Inc. is an American cloud-based software company headquartered in San Francisco, California. Though the bulk of its revenue comes from a customer relationship management (CRM) product, Salesforce also sells a complementary suite of enterprise applications focused on customer service, marketing automation, analytics and application development.

Salesforce is the primary enterprise offering within the Salesforce platform. It provides an interface for case management and task management, and a system for automatically routing and escalating important events. The Salesforce customer portal provides customers the ability to track their own cases, includes a social networking plug-in that enables the user to the conversation about their company on social networking websites, provides analytical tools and other services including email alert, Google search, and access to customers' entitlement and contracts.

Lightning Platform

Lightning Platform (also known as Force.com) is a platform as a service (PaaS) that allows developers to create add-on applications that integrate into the main Salesforce.com application. These third-party applications are hosted on Salesforce.com[t]s infrastructure.Force.com applications are built using declarative tools, backed by Lightning and Apex (a proprietary Java-like programming language for Force.com) md Lightning and Visual force (a framework that includes an XML syntax typically used to generate HTML). The Force.com platform typically receives three complete releases a year. As the platform is provided as a service to its developers, every single development instance also receives all these updates.

Community Cloud

Community Cloud provides Salesforce customers the ability to create online web properties for external collaboration, customer service, channel sales, and other custom portals in

their instance of Salesforce. Tightly integrated to Sales Cloud, Service Cloud, and App Cloud, Community Cloud can be quickly customized to provide a wide variety of web properties

Sales force Sales Cloud

Salesforce Sales Cloud is a customer relationship management (CRM) platform designed to support sales, marketing and customer support in both business-to-business (B2B) and business-to-customer (B2C) contexts. Sales Cloud is a fully customizable product that brings all the customer information together in an integrated platform that incorporates marketing, lead generation, sales, customer service and business analytics and provides access to thousands of applications through the AppExchange. The platform is provided as Software as a Service (SaaS) for browser-based access; a mobile app is also available. A real. time social feed for collaboration allows users to share information or ask questions of the user community-Salesforce.com offers five versions of Sales Cloud on a per-user, per month basis, from lowest to highest: Group, Professional, Enterprise, Unlimited and Performance. The company offers three levels of support contracts: Standard Success Plan, Premier Success Plan and Premier+ Success Plan.

Create Custom Apps for Salesforce Classic

Create custom apps to give your Salesforce Classic users' access to everything they need all in one place.

If you're new to custom apps, we recommend using Lightning Platform quick start to create an app. With this tool, you can generate a basic working app in just one step.

If you've already created the objects, tabs, and fields you need for your app, follow these steps. With this option, you create an app label and logo, add items to the app, and assign the app to profiles.

l . From Setup, enter Apps in the Quick Find box, then select Apps.

1. Click New.
2. If the Salesforce console is available, select whether you want to define a custom app or a Salesforce console.
3. Give the app a name and description.
4. An app name can have a maximum of 40 characters, Including spaces.

5. Optionally, brand your app by giving it a custom logo.

Select which items to include in the app.

1. Optionally, set the default landing tab for your new app using the Default Landing Tab drop-down menu below the list of selected tabs. This determines the first tab a user sees when logging into this app.
2. Choose which profiles the app will be visible to.
3. Check the Default to set the app as that profile's default app, meaning that new users with the profile see this app the first time they log in. Profiles with limits are excluded from this list.
4. Click Save

What is the difference between custom application and console application in sales force?

A custom application is a collection of tabs, objects etc that function together to solve a particular problem.

A console application uses a specific Salesforce UI the console. Console applications are intended to enhance productivity by allowing everything to be done from a single, tabbed, screen.

Conclusion:

Hence, we studied application for Sales force in cloud.