

# DETERMINISTIC FINITE AUTOMATA (DFA)

Aim

= = =

To write a C program to simulate a Deterministic Finite Automata.

Algorithm

= = =

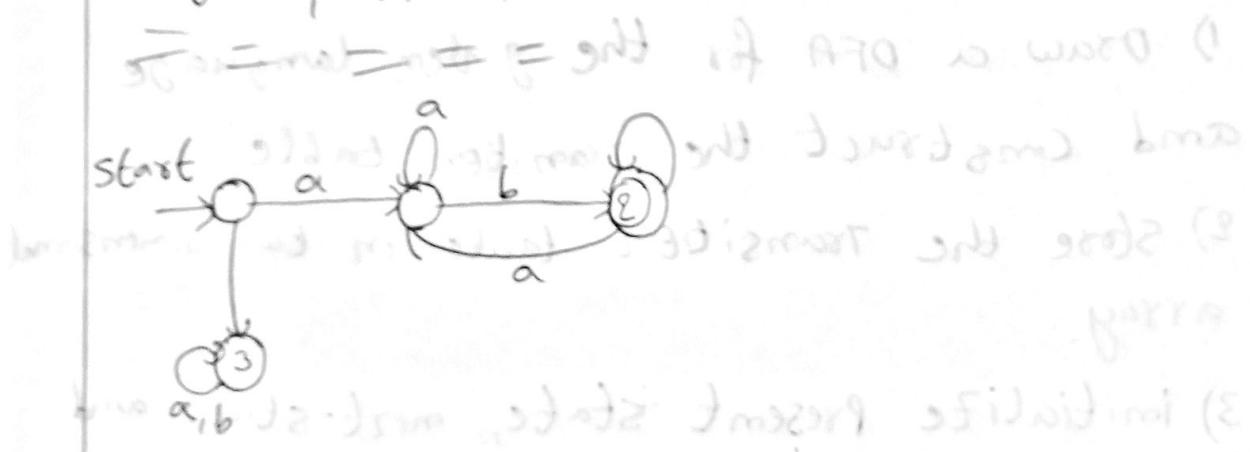
- 1) Draw a DFA for the given language and construct the Transition Table
- 2) Store the Transition table in two-dimensional array.
- 3) Initialize present state, next-state and Final state
- 4) Get the input string from the user.
- 5) Find the length of the string
- 6) Read the input string character by character
- 7) Repeat step 8 for every character
- 8) Refer the Transition table for the entry corresponding to the present state and the current input symbol and update the next state
- 9) When we reach the end of the input, if the

final state is reached, the input is accepted. otherwise the inputs is not accepted.

### Example

simulate a DFA for the language representing strings over  $\Sigma = \{a, b\}$  that start with a and end with b

## Design of the OFA



Transition table of DFA state transition

state/input	a	b	c
q0 → q1	1	3	1
q1 → q2	1	2	1
q2 → q3	3	1	1
q3 → q1	3	3	1

Program

====

```
#include <stdio.h>
#include <string.h>
#define max 20
int main()
{
    int trans_table[4][2] = {{1,3}, {1,2}, {1,2}, {2,3}};
    int final_state = 2, i;
    int present_state = 0;
    int next_state = 0;
    int invalid = 0;
    char input_string[max];
    printf("Enter a string");
    scanf("%s", input_string);
    for (int l = strlen(input_string));
        for (i = 0; i < l; i++)
    {
        if (input_string[i] == 'a')
            next_state = trans_table[present_state][0];
        else if (input_string[i] == 'b')
            next_state = trans_table[present_state][1];
        else
            invalid = 1;
        present_state = next_state;
    }
}
```

3.

```
if (invalid == 1)
{
    printf ("Invalid input");
}
else if (present-state == final-state)
printf ("Accept(m)");
else
printf ("don't Accept(m)");
}
```

output  
=====

Enter a string: abaaab

Accept

Enter a string: abbbbaaaba

Accept

Enter a string: aaabb

Invalid Input.

~~decide whether a string belongs to a grammar~~

CHECKING WHETHER A STRING BELONGS  
TO A GRAMMER

Aim

= = =

To write a C program to check whether  
a string belongs to the grammar

$S \rightarrow 0A1$

$A \rightarrow 0A / 1A / \epsilon$

Language defined by the grammar

= = = = =  
set of all strings over  $\Sigma = \{0, 1\}$  that starts  
with 0 and ends with 1

Algorithm

= = =

- 1) Get the input string from the user.
- 2) Find the length of the string.
- 3) Check whether all the symbols in the input are either 0 or 1. If so, print "string is valid" and go to step 4. Otherwise print "string not valid" and exit the program.

If the first symbol is '0' and the last symbol is '1', print "string accepted". Otherwise print "string not accepted".

Program

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s[100];
    int i, flag;
    int l;
    printf("Enter a string to check:");
    scanf("%s", s);
    l = strlen(s);
    flag = 1;
    for(i=0; i<l; i++)
    {
        if(s[i] != '0' & s[i] != '1')
            flag = 0;
    }
    if(flag == 1)
        printf("String is valid");
    else
        printf("String is not valid");
}
```

```
if (S[0] == '0' && S[1-1] == '1')  
    printf("string is accepted (%s)", S);  
else  
    printf("string not accepted (%s)", S);
```

y

output

====

Enter a string to check: 0|01011101

string is Accepted.

011101010110

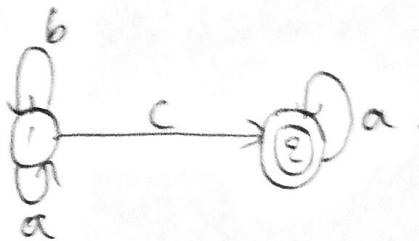
string is not Accepted.

abbbbabab

string is not valid.

Design DFA using simulator to accept the string

DFA  
==

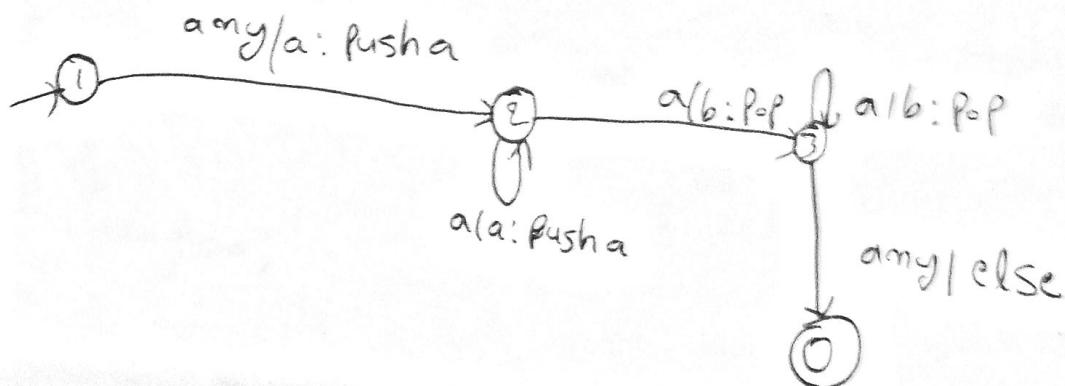


output  
==

$\rightarrow bcaaaa$   
 $\rightarrow bc$   
 $\rightarrow a$

Design PDA using simulator to accept the input string aabb.

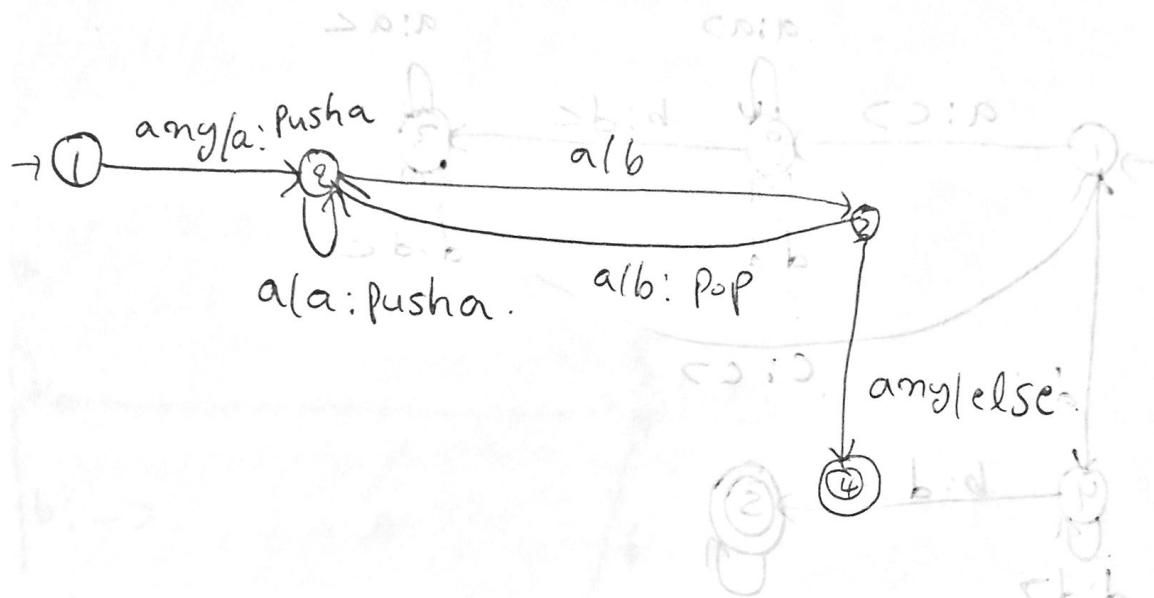
PDA  
==



Design PDA using simulator to accept the string  $a^n b^m$

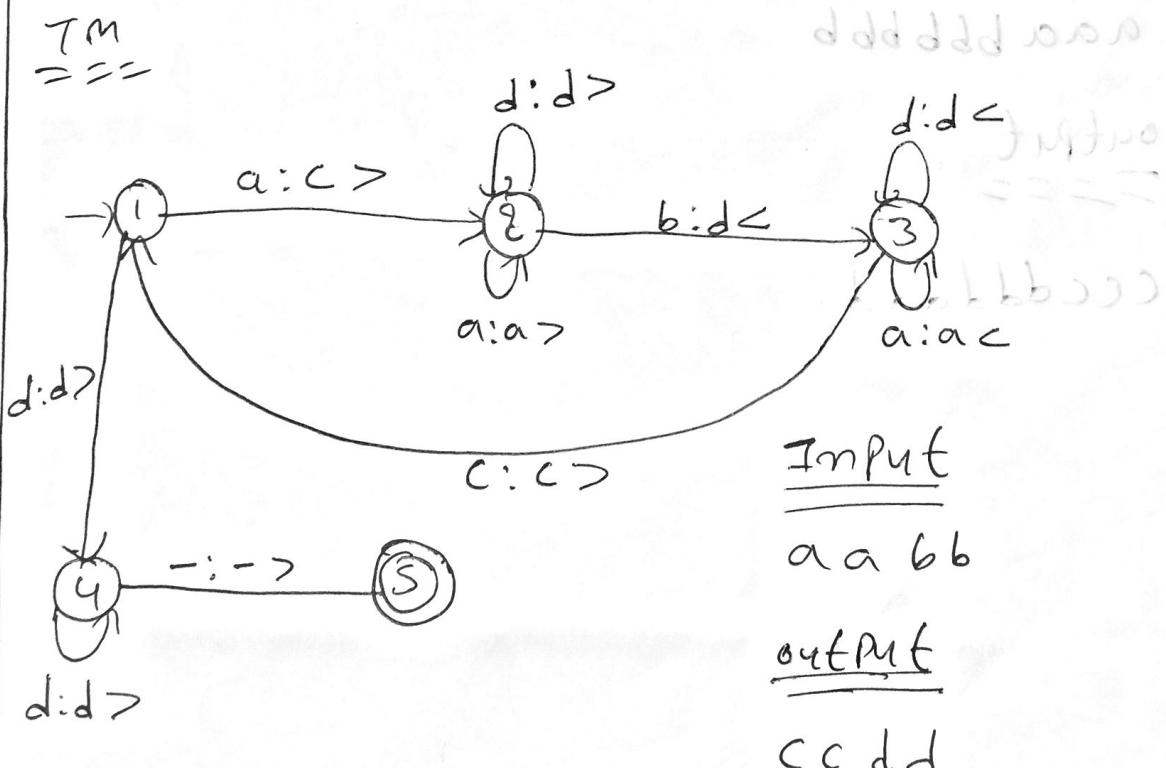
PDA

====



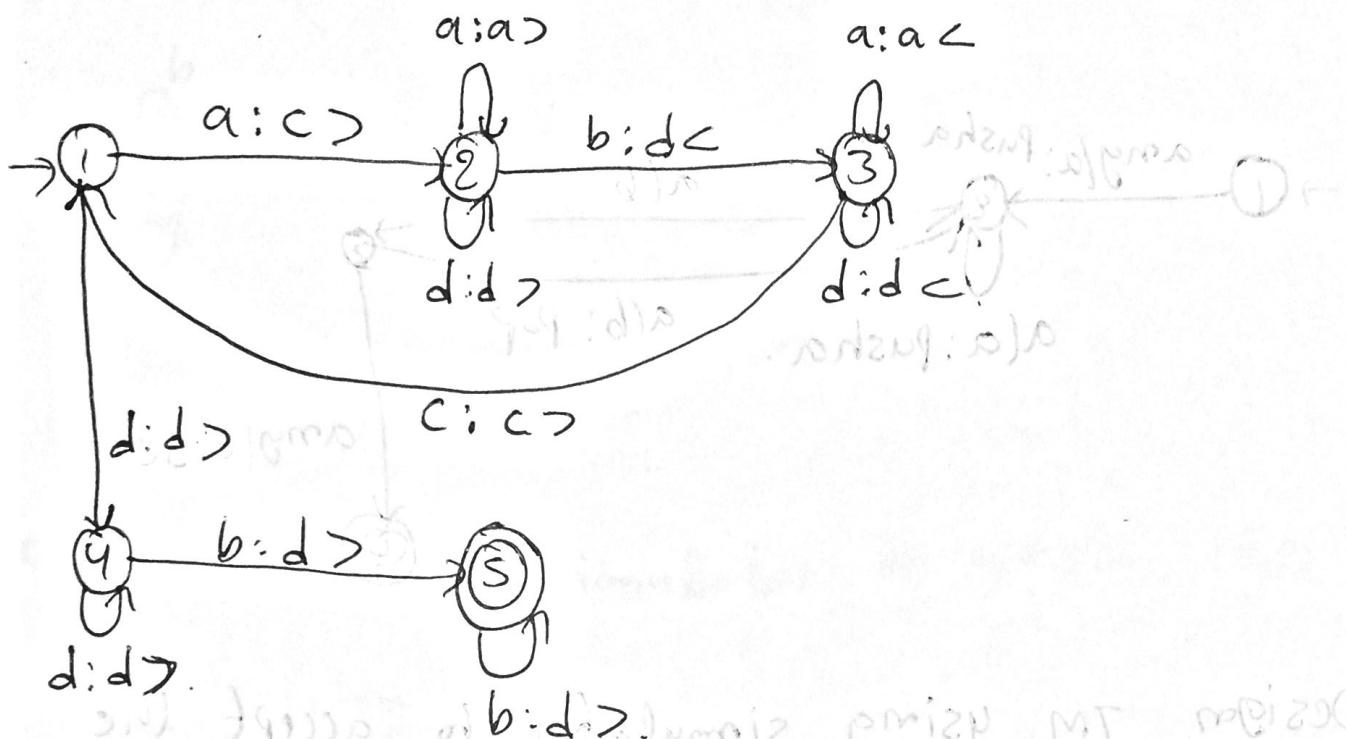
Design TM using simulator to accept the Input string  $a^n b^m$ .

TM



Design TM of using simulator to accept  
the string  $A^n B^{2n}$ .

TM  
====



Input  
=====

aaa bbb bbb

Output  
=====

ccc ddd ddd

Input

dd pp

<b:b

<d:d

MJ

EE

sb:b

<c:c

(2)

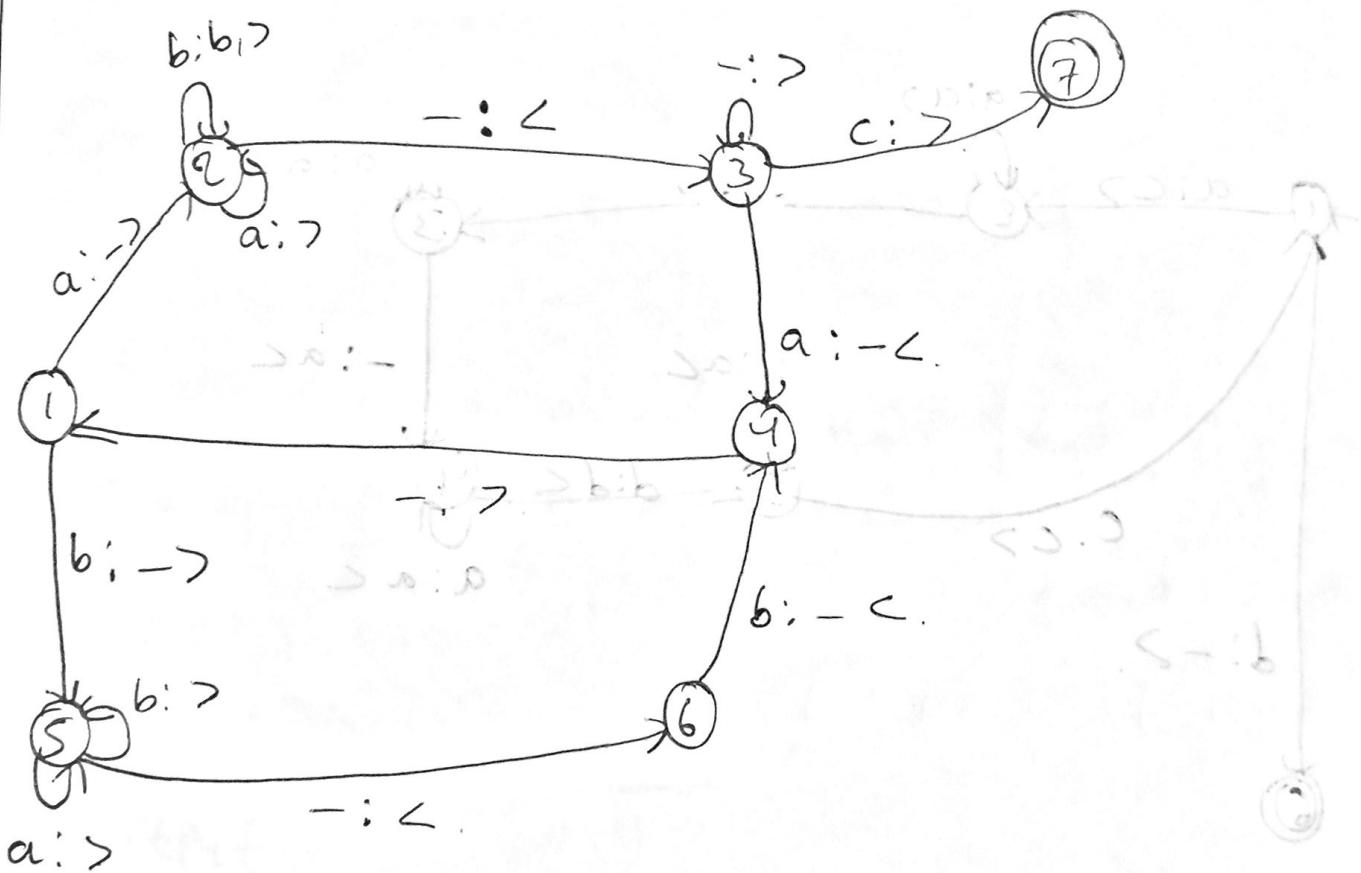
S-:-

(p)

Design TM using simulator to accept the  
Input string palindrome.

TM

====



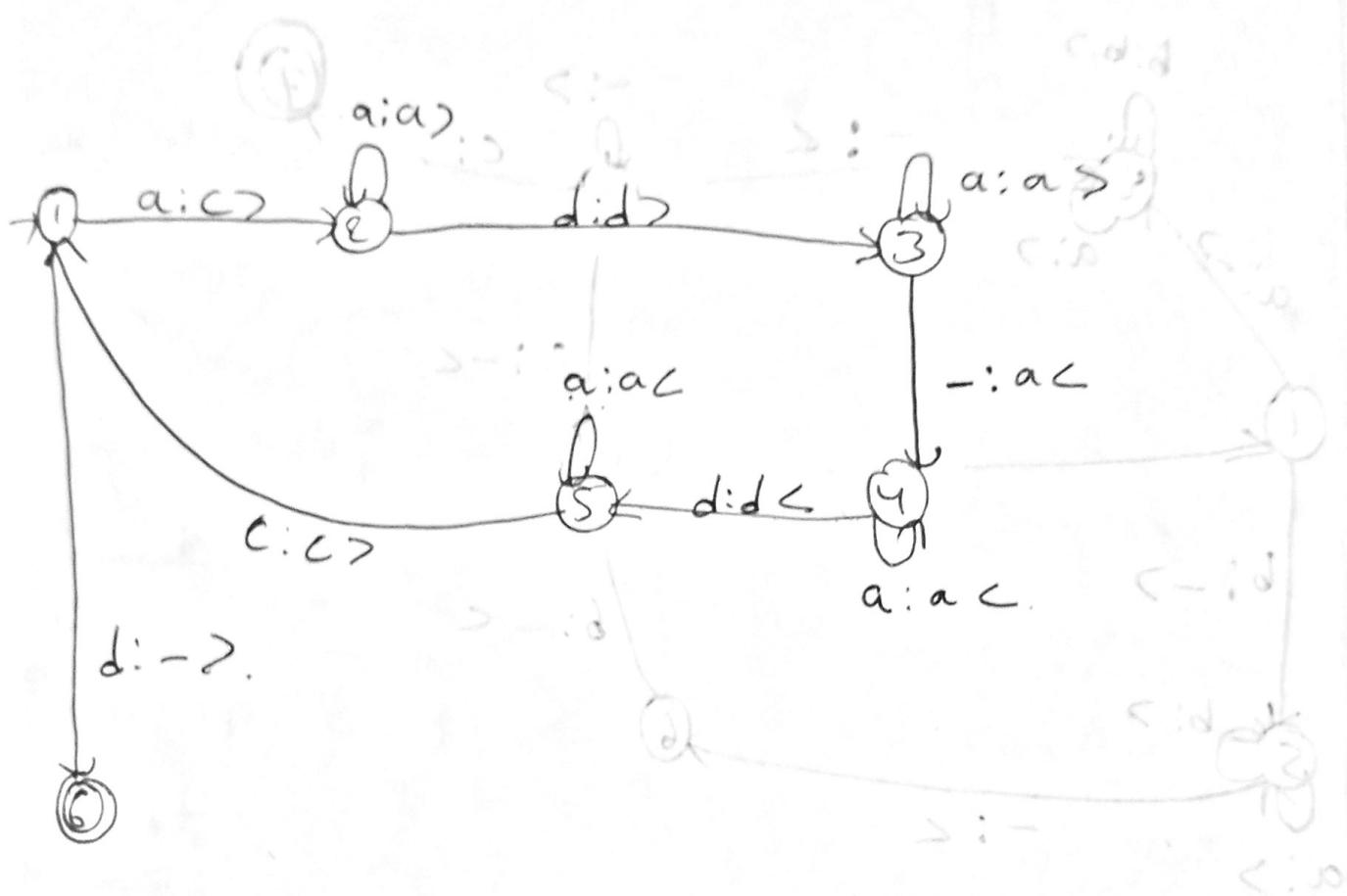
Input

====

ababa c.

Design TM using simulator to perform addition of 'aa' and 'aaa'

TM  
==

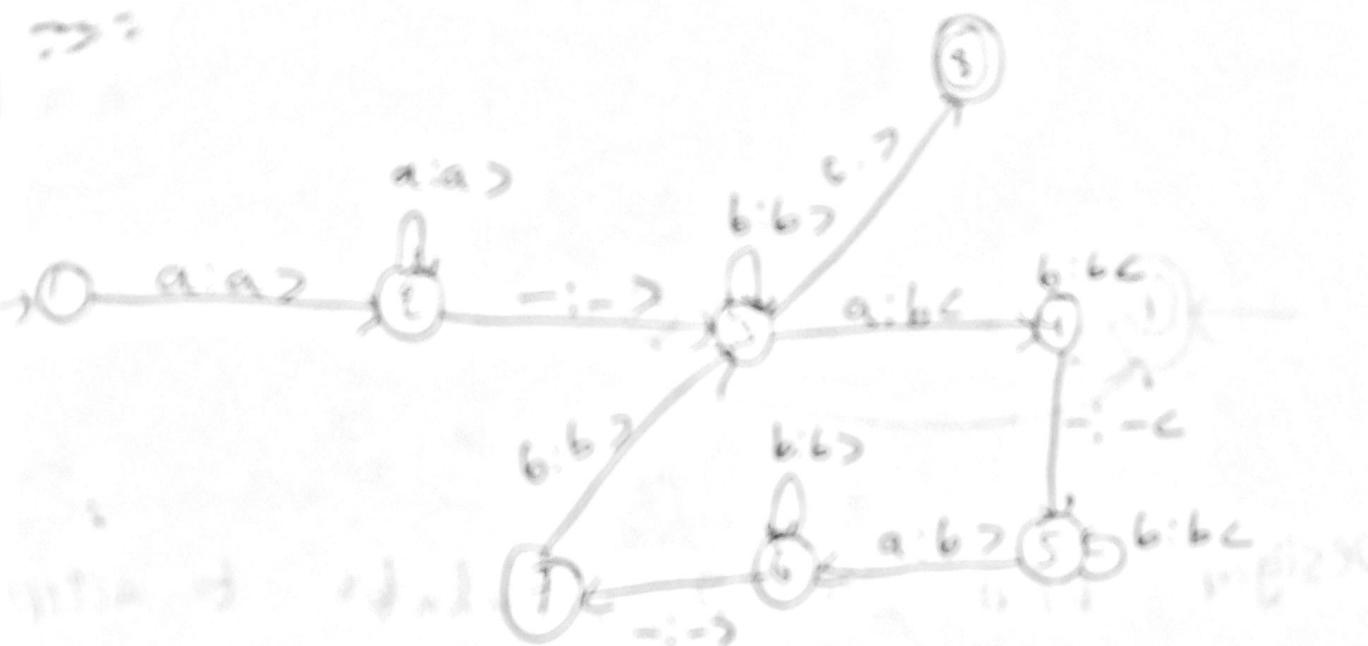


Design TM using Simulab to perform subtraction of aaa - aa.

TM

$\Rightarrow \Sigma = \{a, b\}$

$\Sigma^* = \{a, b\}^*$



Input  
 $\Sigma^* = \{a, b\}^*$

aaa - aa.

Output

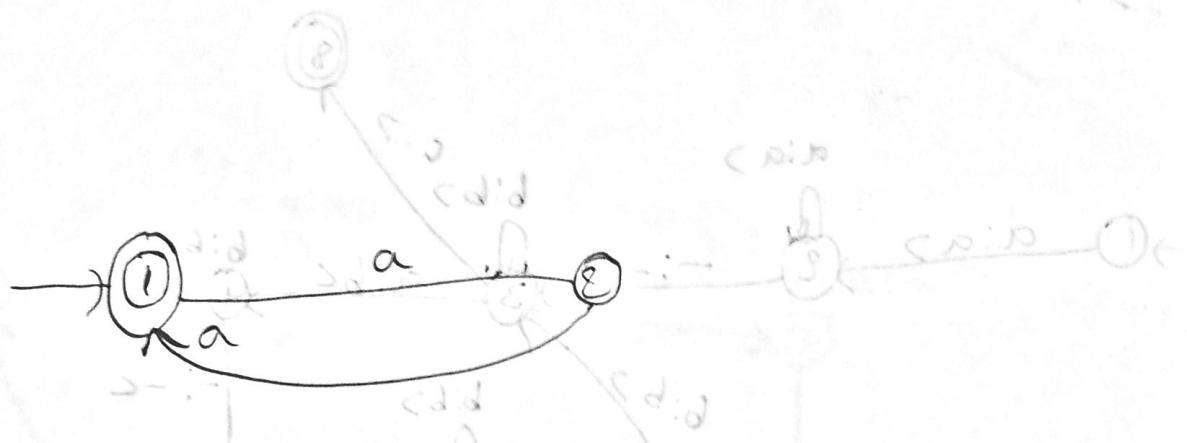
$\Sigma^* = \{a, b\}^*$

abb - bb.

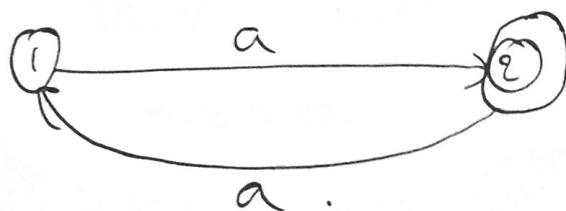


Design DFA using simulator to accept  
even number a's

DFA  
====

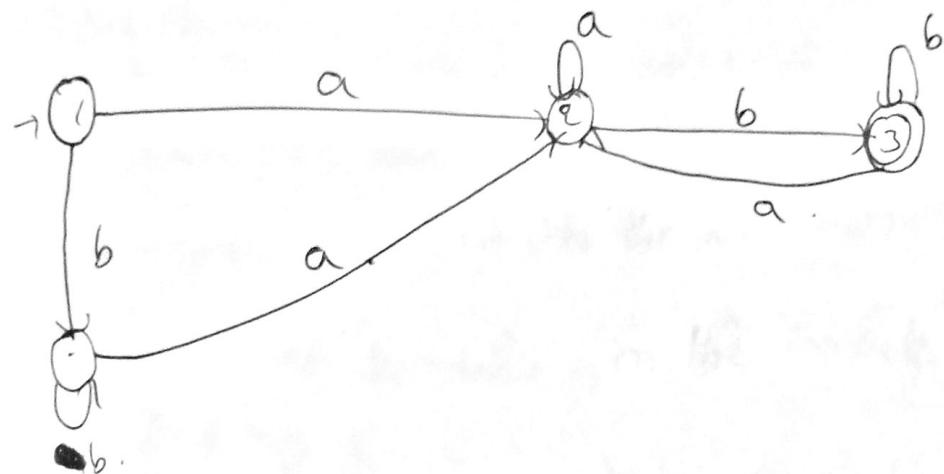


Design DFA using simulator to accept  
odd number a's

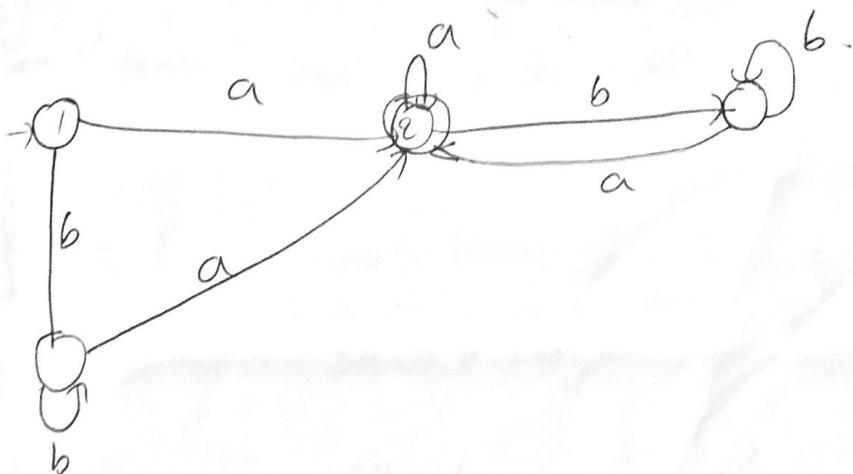


design DFA using simulator to accept  
 the strings which end with ab over set {a,b}  
 $w = aabab.$

DFA  
 ===

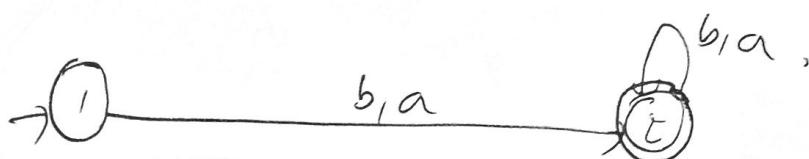


design DFA using simulator to accept the  
 string having 'ab' as substring over the set {a,b}

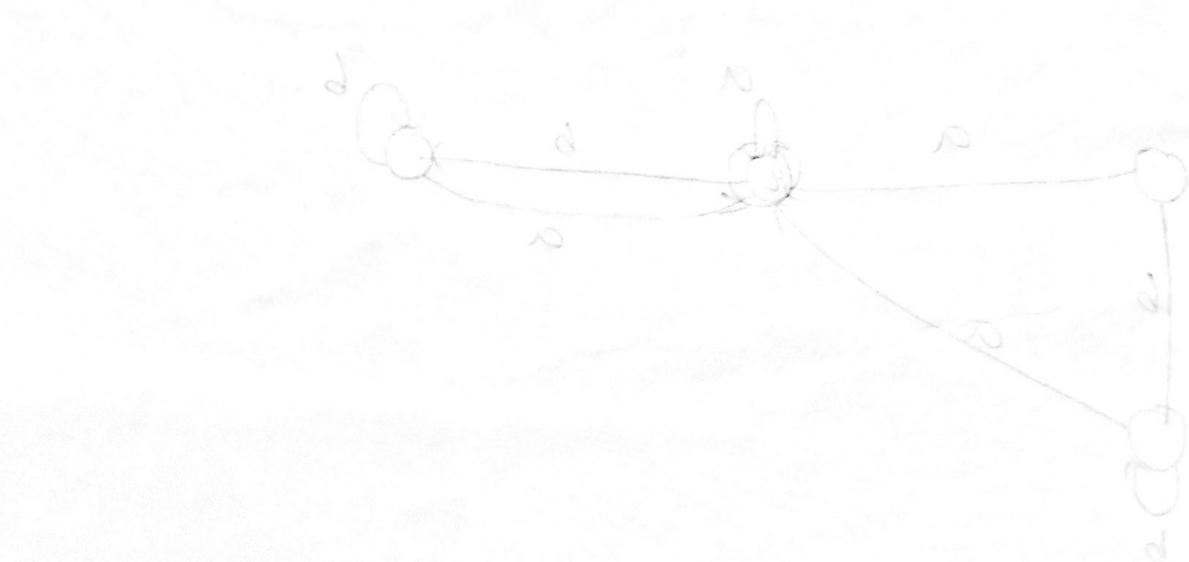


Design DFA using simulation to accept the string start with a or b moves the set

{a, b}



After this, a random error happened so  
it's not in the middle (a lot of private circles)



## FINDING $\epsilon$ -CLOSURE FOR NFA WITH $\epsilon$ -MOVES

~~=  $\epsilon$ -closure = union of all states~~

Aims  
~~= = =~~ To find the  $\epsilon$ -closure of a state  $s$  is

To write a C program to find  $\epsilon$ -closure of a non-deterministic finite automata with  $\epsilon$ -moves.

~~Algorithm~~  
~~= = =~~ state need modification  $\rightarrow$  no change in

i) initializes all the values ~~from left of i~~

ii) number of states in the NFA ~~mid way forward~~

iii) number of symbols in the input alphabet including  $\epsilon$

iv) input symbols

v) number of final states and their names.

2) declare a 3-dimensional matrix to store the transitions and initialize all the entries with -1

3) get the transitions from every state for every input symbol from the user and store it in the matrix.

4) Initialize a two-dimensional Matrix closure with 1 in all the entries.

5)  $\epsilon$ -closure of a state  $i$  is defined as the set of all states that can be reached from state  $i$  using only  $\epsilon$ -transitions.

6) For every state  $i$ , find  $\epsilon$ -closure as follow:

if there is an  $\epsilon$ -transition from state  $i$  to state  $j$ ,

add  $j$  to the matrix  $\epsilon$ -closure( $i, j$ ) (call it  $\text{find-}\epsilon\text{-closure}(i)$ )

Recursive function  $\text{find-}\epsilon\text{-closure}(j)$ : and add all the other states that are reachable from  $i$  in  $\text{find-}\epsilon\text{-closure}(i)$  using  $\epsilon$

7) For every state, print the  $\epsilon$ -closure value.

Program

```
#include<stdio.h> // Initialization -> a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, z1, z2, z3, z4, z5, z6, z7, z8, z9, z10, z11, z12, z13, z14, z15, z16, z17, z18, z19, z20, z21, z22, z23, z24, z25, z26, z27, z28, z29, z30, z31, z32, z33, z34, z35, z36, z37, z38, z39, z40, z41, z42, z43, z44, z45, z46, z47, z48, z49, z50, z51, z52, z53, z54, z55, z56, z57, z58, z59, z60, z61, z62, z63, z64, z65, z66, z67, z68, z69, z70, z71, z72, z73, z74, z75, z76, z77, z78, z79, z80, z81, z82, z83, z84, z85, z86, z87, z88, z89, z90, z91, z92, z93, z94, z95, z96, z97, z98, z99, z100, z101, z102, z103, z104, z105, z106, z107, z108, z109, z110, z111, z112, z113, z114, z115, z116, z117, z118, z119, z120, z121, z122, z123, z124, z125, z126, z127, z128, z129, z130, z131, z132, z133, z134, z135, z136, z137, z138, z139, z140, z141, z142, z143, z144, z145, z146, z147, z148, z149, z150, z151, z152, z153, z154, z155, z156, z157, z158, z159, z160, z161, z162, z163, z164, z165, z166, z167, z168, z169, z170, z171, z172, z173, z174, z175, z176, z177, z178, z179, z180, z181, z182, z183, z184, z185, z186, z187, z188, z189, z190, z191, z192, z193, z194, z195, z196, z197, z198, z199, z200, z201, z202, z203, z204, z205, z206, z207, z208, z209, z210, z211, z212, z213, z214, z215, z216, z217, z218, z219, z220, z221, z222, z223, z224, z225, z226, z227, z228, z229, z230, z231, z232, z233, z234, z235, z236, z237, z238, z239, z240, z241, z242, z243, z244, z245, z246, z247, z248, z249, z250, z251, z252, z253, z254, z255, z256, z257, z258, z259, z260, z261, z262, z263, z264, z265, z266, z267, z268, z269, z270, z271, z272, z273, z274, z275, z276, z277, z278, z279, z280, z281, z282, z283, z284, z285, z286, z287, z288, z289, z290, z291, z292, z293, z294, z295, z296, z297, z298, z299, z299, z300, z301, z302, z303, z304, z305, z306, z307, z308, z309, z310, z311, z312, z313, z314, z315, z316, z317, z318, z319, z320, z321, z322, z323, z324, z325, z326, z327, z328, z329, z330, z331, z332, z333, z334, z335, z336, z337, z338, z339, z340, z341, z342, z343, z344, z345, z346, z347, z348, z349, z350, z351, z352, z353, z354, z355, z356, z357, z358, z359, z360, z361, z362, z363, z364, z365, z366, z367, z368, z369, z370, z371, z372, z373, z374, z375, z376, z377, z378, z379, z380, z381, z382, z383, z384, z385, z386, z387, z388, z389, z390, z391, z392, z393, z394, z395, z396, z397, z398, z399, z400, z401, z402, z403, z404, z405, z406, z407, z408, z409, z410, z411, z412, z413, z414, z415, z416, z417, z418, z419, z420, z421, z422, z423, z424, z425, z426, z427, z428, z429, z430, z431, z432, z433, z434, z435, z436, z437, z438, z439, z440, z441, z442, z443, z444, z445, z446, z447, z448, z449, z450, z451, z452, z453, z454, z455, z456, z457, z458, z459, z460, z461, z462, z463, z464, z465, z466, z467, z468, z469, z470, z471, z472, z473, z474, z475, z476, z477, z478, z479, z480, z481, z482, z483, z484, z485, z486, z487, z488, z489, z490, z491, z492, z493, z494, z495, z496, z497, z498, z499, z500, z501, z502, z503, z504, z505, z506, z507, z508, z509, z510, z511, z512, z513, z514, z515, z516, z517, z518, z519, z520, z521, z522, z523, z524, z525, z526, z527, z528, z529, z530, z531, z532, z533, z534, z535, z536, z537, z538, z539, z540, z541, z542, z543, z544, z545, z546, z547, z548, z549, z550, z551, z552, z553, z554, z555, z556, z557, z558, z559, z559, z560, z561, z562, z563, z564, z565, z566, z567, z568, z569, z569, z570, z571, z572, z573, z574, z575, z576, z577, z578, z579, z579, z580, z581, z582, z583, z584, z585, z586, z587, z588, z589, z589, z590, z591, z592, z593, z594, z595, z596, z597, z598, z599, z599, z600, z601, z602, z603, z604, z605, z606, z607, z608, z609, z609, z610, z611, z612, z613, z614, z615, z616, z617, z618, z619, z619, z620, z621, z622, z623, z624, z625, z626, z627, z628, z629, z629, z630, z631, z632, z633, z634, z635, z636, z637, z638, z639, z639, z640, z641, z642, z643, z644, z645, z646, z647, z648, z649, z649, z650, z651, z652, z653, z654, z655, z656, z657, z658, z659, z659, z660, z661, z662, z663, z664, z665, z666, z667, z668, z669, z669, z670, z671, z672, z673, z674, z675, z676, z677, z678, z679, z679, z680, z681, z682, z683, z684, z685, z686, z687, z688, z689, z689, z690, z691, z692, z693, z694, z695, z696, z697, z698, z699, z699, z700, z701, z702, z703, z704, z705, z706, z707, z708, z709, z709, z710, z711, z712, z713, z714, z715, z716, z717, z718, z719, z719, z720, z721, z722, z723, z724, z725, z726, z727, z728, z729, z729, z730, z731, z732, z733, z734, z735, z736, z737, z738, z739, z739, z740, z741, z742, z743, z744, z745, z746, z747, z748, z749, z749, z750, z751, z752, z753, z754, z755, z756, z757, z758, z759, z759, z760, z761, z762, z763, z764, z765, z766, z767, z768, z769, z769, z770, z771, z772, z773, z774, z775, z776, z777, z778, z779, z779, z780, z781, z782, z783, z784, z785, z786, z787, z788, z789, z789, z790, z791, z792, z793, z794, z795, z796, z797, z798, z799, z799, z800, z801, z802, z803, z804, z805, z806, z807, z808, z809, z809, z810, z811, z812, z813, z814, z815, z816, z817, z818, z819, z819, z820, z821, z822, z823, z824, z825, z826, z827, z828, z829, z829, z830, z831, z832, z833, z834, z835, z836, z837, z838, z839, z839, z840, z841, z842, z843, z844, z845, z846, z847, z848, z849, z849, z850, z851, z852, z853, z854, z855, z856, z857, z858, z859, z859, z860, z861, z862, z863, z864, z865, z866, z867, z868, z869, z869, z870, z871, z872, z873, z874, z875, z876, z877, z878, z879, z879, z880, z881, z882, z883, z884, z885, z886, z887, z888, z889, z889, z890, z891, z892, z893, z894, z895, z896, z897, z898, z899, z899, z900, z901, z902, z903, z904, z905, z906, z907, z908, z909, z909, z910, z911, z912, z913, z914, z915, z916, z917, z918, z919, z919, z920, z921, z922, z923, z924, z925, z926, z927, z928, z929, z929, z930, z931, z932, z933, z934, z935, z936, z937, z938, z939, z939, z940, z941, z942, z943, z944, z945, z946, z947, z948, z949, z949, z950, z951, z952, z953, z954, z955, z956, z957, z958, z959, z959, z960, z961, z962, z963, z964, z965, z966, z967, z968, z969, z969, z970, z971, z972, z973, z974, z975, z976, z977, z978, z979, z979, z980, z981, z982, z983, z984, z985, z986, z987, z988, z989, z989, z990, z991, z992, z993, z994, z995, z996, z997, z998, z999, z999, z1000, z1001, z1002, z1003, z1004, z1005, z1006, z1007, z1008, z1009, z1009, z1010, z1011, z1012, z1013, z1014, z1015, z1016, z1017, z1018, z1019, z1019, z1020, z1021, z1022, z1023, z1024, z1025, z1026, z1027, z1028, z1029, z1029, z1030, z1031, z1032, z1033, z1034, z1035, z1036, z1037, z1038, z1039, z1039, z1040, z1041, z1042, z1043, z1044, z1045, z1046, z1047, z1048, z1049, z1049, z1050, z1051, z1052, z1053, z1054, z1055, z1056, z1057, z1058, z1059, z1059, z1060, z1061, z1062, z1063, z1064, z1065, z1066, z1067, z1068, z1069, z1069, z1070, z1071, z1072, z1073, z1074, z1075, z1076, z1077, z1078, z1079, z1079, z1080, z1081, z1082, z1083, z1084, z1085, z1086, z1087, z1088, z1089, z1089, z1090, z1091, z1092, z1093, z1094, z1095, z1096, z1097, z1098, z1099, z1099, z1100, z1101, z1102, z1103, z1104, z1105, z1106, z1107, z1108, z1109, z1109, z1110, z1111, z1112, z1113, z1114, z1115, z1116, z1117, z1118, z1119, z1119, z1120, z1121, z1122, z1123, z1124, z1125, z1126, z1127, z1128, z1129, z1129, z1130, z1131, z1132, z1133, z1134, z1135, z1136, z1137, z1138, z1139, z1139, z1140, z1141, z1142, z1143, z1144, z1145, z1146, z1147, z1148, z1149, z1149, z1150, z1151, z1152, z1153, z1154, z1155, z1156, z1157, z1158, z1159, z1159, z1160, z1161, z1162, z1163, z1164, z1165, z1166, z1167, z1168, z1169, z1169, z1170, z1171, z1172, z1173, z1174, z1175, z1176, z1177, z1178, z1179, z1179, z1180, z1181, z1182, z1183, z1184, z1185, z1186, z1187, z1188, z1189, z1189, z1190, z1191, z1192, z1193, z1194, z1195, z1196, z1197, z1198, z1199, z1199, z1200, z1201, z1202, z1203, z1204, z1205, z1206, z1207, z1208, z1209, z1209, z1210, z1211, z1212, z1213, z1214, z1215, z1216, z1217, z1218, z1219, z1219, z1220, z1221, z1222, z1223, z1224, z1225, z1226, z1227, z1228, z1229, z1229, z1230, z1231, z1232, z1233, z1234, z1235, z1236, z1237, z1238, z1239, z1239, z1240, z1241, z1242, z1243, z1244, z1245, z1246, z1247, z1248, z1249, z1249, z1250, z1251, z1252, z1253, z1254, z1255, z1256, z1257, z1258, z1259, z1259, z1260, z1261, z1262, z1263, z1264, z1265, z1266, z1267, z1268, z1269, z1269, z1270, z1271, z1272, z1273, z1274, z1275, z1276, z1277, z1278, z1279, z1279, z1280, z1281, z1282, z1283, z1284, z1285, z1286, z1287, z1288, z1289, z1289, z1290, z1291, z1292, z1293, z1294, z1295, z1296, z1297, z1298, z1299, z1299, z1300, z1301, z1302, z1303, z1304, z1305, z1306, z1307, z1308, z1309, z1309, z1310, z1311, z1312, z1313, z1314, z1315, z1316, z1317, z1318, z1319, z1319, z1320, z1321, z1322, z1323, z1324, z1325, z1326, z1327, z1328, z1329, z1329, z1330, z1331, z1332, z1333, z1334, z1335, z1336, z1337, z1338, z1339, z1339, z1340, z1341, z1342, z1343, z1344, z1345, z1346, z1347, z1348, z1349, z1349, z1350, z1351, z1352, z1353, z1354, z1355, z1356, z1357, z1358, z1359, z1359, z1360, z1361, z1362, z1363, z1364, z1365, z1366, z1367, z1368, z1369, z1369, z1370, z1371, z1372, z1373, z1374, z1375, z1376, z1377, z1378, z1379, z1379, z1380, z1381, z1382, z1383, z1384, z1385, z1386, z1387, z1388, z1389, z1389, z1390, z1391, z1392, z1393, z1394, z1395, z1396, z1397, z1398, z1399, z1399, z1400, z1401, z1402, z1403, z1404, z1405, z1406, z1407, z1408, z1409, z1409, z1410, z1411, z1412, z1413, z1414, z1415, z1416, z1417, z1418, z1419, z1419, z1420, z1421, z1422, z1423, z1424, z1425, z1426, z1427, z1428, z1429, z1429, z1430, z1431, z1432, z1433, z1434, z1435, z1436, z1437, z1438, z1439, z1439, z1440, z1441, z1442, z1443, z1444, z1445, z1446, z1447, z1448, z1449, z1449, z1450, z1451, z1452, z1453, z1454, z1455, z1456, z1457, z1458, z1459, z1459, z1460, z1461, z1462, z1463, z1464, z1465, z1466, z1467, z1468, z1469, z1469, z1470, z1471, z1472, z1473, z1474, z1475, z1476, z1477, z1478, z1479, z1479, z1480, z1481, z1482, z1483, z1484, z1485, z1486, z1487, z1488, z1489, z1489, z1490, z1491, z1492, z1493, z1494, z1495, z1496, z1497, z1498, z1499, z1499, z1500, z1501, z1502, z1503, z1504, z1505, z1506, z1507, z1508, z1509, z1509, z1510, z1511, z1512, z1513, z1514, z1515, z1516, z1517, z1518, z1519, z1519, z1520, z1521, z1522, z1523, z1524, z1525, z1526, z1527, z1528, z1529, z1529, z1530, z1531, z1532, z1533, z1534, z1535, z1536, z1537, z1538, z1539, z1539, z1540, z1541, z1542, z1543, z1544, z1545, z1546, z1547, z1548, z1549, z1549, z1550, z1551, z1552, z1553, z1554, z1555, z1556, z1557, z1558, z1559, z1559, z1560, z1561, z1562, z1563, z1564, z1565, z1566, z1567, z1568, z1569, z1569, z1570, z1571, z1572, z1573, z1574, z1575, z1576, z1577, z1578, z1579, z1579, z1580, z1581, z1582, z1583, z1584, z1585, z1586, z1587, z1588, z1589, z1589, z1590, z1591, z1592, z1593, z1594, z1595, z1596, z1597, z1598, z1599, z1599, z1600, z1601, z1602, z1603, z1604, z1605, z1606, z1607, z1608, z1609, z1609, z1610, z1611, z1612, z1613, z1614, z1615, z1616, z1617, z1618, z1619, z1619, z1620, z1621, z1622, z1623, z1624, z1625, z1626, z1627, z1628, z1629, z1629, z1630, z1631, z1632, z1633, z1634, z1635, z1636, z1637, z1638, z1639, z1639, z1640, z1641, z1642, z1643, z1644, z1645, z1646, z1647, z1648, z1649, z1649, z1650, z1651, z1652, z1653, z1654, z1655, z1656, z1657, z1658, z1659, z1659, z1660, z1661, z1662, z1663, z1664, z1665, z1666, z1667, z1668, z1669, z1669, z1670, z1671, z1672, z1673, z1674, z1675, z1676, z1677, z1678, z1679, z1679, z1680, z1681, z1682, z1683, z1684, z1685, z1686, z1687, z1688, z1689, z1689, z1690, z1691, z1692, z1693, z1694, z1695, z1696, z1697, z1698, z1699, z1699, z1700, z1701, z1702, z1703, z1704, z1705, z1706, z1707, z1708, z1709, z1709, z1710, z1711, z1712, z1713, z1714, z1715, z1716, z1717, z1718, z1719, z1719, z1720, z1721, z1722, z1723, z1724, z1725, z1726, z1727, z1728, z1729, z1729, z1730, z1731, z1732, z1733, z1734, z1735, z1736, z1737, z1738, z1739, z1739, z1740, z1741, z1742, z1743, z1744, z1745, z1746, z1747, z1748, z1749, z1749, z1750, z1751, z1752, z1753, z1754, z1755, z1756, z1757, z1758, z1759, z1759, z1760, z1761, z1762, z1763, z1764, z1765, z1766, z1767, z1768, z1769, z1769, z1770, z1771, z1772, z1773, z1774, z1775, z1776, z1777, z1778, z1779, z1779, z1780, z1781, z1782, z1783, z1784, z1785, z1786, z1787, z1788, z1789, z1789, z1790, z1791, z1792, z1793, z1794, z1795, z1796, z1797, z1798, z1799, z1799, z1800, z1801, z1802, z1803, z1804, z1805, z1806, z1807, z1808, z1809, z1809, z1810, z1811, z1812, z1813, z1814, z1815, z1816, z1817, z1818, z1819, z1819, z1820, z1821, z1822, z1823, z1824, z1825, z1826, z1827, z1828, z1
```

```

int i, j, k, m, num-states, num-symbols; // ...
for (i=0; i<num-states; i++)
{
    for (j=0; j<num-symbols; j++)
    {
        f1(k=0; k<3; k++)
        {
            if ((j>3 || (k>2) & (j==3)) & (trans-table[i][j][k] == -1))
                cout << "Error in state " << i << endl;
        }
    }
}
num-states = 3; // i.e., (1-2)(000) mod 3 = 0
num-symbols = 4; // i.e., {0, 1, 2, 3}
symbol(1) = 'e';
m = 1;
trans-table[0][0][0] = 1; // ...
f1(i=0; i<m; i++)
{
    for (j=0; j<num-symbols; j++)
    {
        if (trans-table[i][j][0] == -1)
            cout << "Error in state " << i << endl;
    }
}
e-close[0][0] = -1;
}
}
for (i=0; i<num-states; i++)
{
    if (trans-table[i][0][0] == -1)
        continue;
    else;
}

```

```

{
state = i;
Pfx = 1;
find e-closure (i);
}
for (i=0; i<num-states; i++)
{
printf ("e-closure(%d) = %s", i);
for (j=0; j<num-states; i++)
{
if (e-closure[i][j] != -1)
{
printf ("%d, ", e-closure[i][j]);
}
printf ("%s\n");
}
void find e-closure (int x)
{
int i, j, y[10], num_trans;
i=0;
while (trans_table[x][0][i] != -1)
{
y[i] = trans_table[x][0][i];
i=i+1;
}
}

```

```
num_trans = i;  
for (j=0; j < num_trans; j++)  
{  
    e-closure[state][ptr] = s[j];  
    ptr++;  
    find_e-closure(y[i]);  
}
```

output

== =

e-closure(0) = { ; , }

e-closure(1) = { , }

e-closure(2) = { }.