EXPERIMENT NO. 2

AIM :- To write a C program to minimize the number of states of a DFA.

ASSUMPTIONS:-

1. Number of inputs =2: a, b
2. Maximum number of states =10

ALGORITHM:-

1. Ask the user to input the number of states
2. Ask the user to provide the number of final states
3. Get set of final states
4. Store set of final states in an array
5. Get transition function from the user
6. Store the transition function and some suitable data structure (2 dimensional array or Link list)
7. Divide the final states and non final states in to 2 different equivalence classes.
8. For each equivalence class do the following
   1. Check for each pair of states(Qi and Qj) in an equivalence class whethe Tr\_f(Qi,a) and Tr\_f(Qj,a) are in the same equivalence class for all input symbols a in Ʃ.
   2. If yes put Qi and Qj in same equivalence class for next iteration
   3. Else Put Qi and Qj in separate Equivalence classes for next iteration
9. If the equivalence classes created are not same as in previous iteration got step 8
10. The states in same equivalence classes are equivalent. Hence, retain only one of them.
11. Display the modified transition function of minimized automata

CODE :-

#include <stdio.h>

#include <string.h>

#define STATES 99

#define SYMBOLS 20

int N\_symbols; /\* number of input symbols \*/

int N\_DFA\_states; /\* number of DFA states \*/

char \*DFA\_finals; /\* final-state string \*/

int DFAtab[STATES][SYMBOLS];

char StateName[STATES][STATES+1]; /\* state-name table \*/

int N\_optDFA\_states; /\* number of optimized DFA states \*/

int OptDFA[STATES][SYMBOLS];

char NEW\_finals[STATES+1];

/\*

Print state-transition table.

State names: 'A', 'B', 'C', ...

\*/

void print\_dfa\_table(

int tab[][SYMBOLS], /\* DFA table \*/

int nstates, /\* number of states \*/

int nsymbols, /\* number of input symbols \*/

char \*finals)

{

int i, j;

puts("\nDFA: STATE TRANSITION TABLE");

/\* input symbols: '0', '1', ... \*/

printf(" | ");

for (i = 0; i < nsymbols; i++) printf(" %c ", '0'+i);

printf("\n-----+--");

for (i = 0; i < nsymbols; i++) printf("-----");

printf("\n");

for (i = 0; i < nstates; i++) {

printf(" %c | ", 'A'+i); /\* state \*/

for (j = 0; j < nsymbols; j++)

printf(" %c ", tab[i][j]); /\* next state \*/

printf("\n");

}

printf("Final states = %s\n", finals);

}

/\*

Initialize NFA table.

\*/

void load\_DFA\_table()

{

DFAtab[0][0] = 'B'; DFAtab[0][1] = 'C';

DFAtab[1][0] = 'E'; DFAtab[1][1] = 'F';

DFAtab[2][0] = 'A'; DFAtab[2][1] = 'A';

DFAtab[3][0] = 'F'; DFAtab[3][1] = 'E';

DFAtab[4][0] = 'D'; DFAtab[4][1] = 'F';

DFAtab[5][0] = 'D'; DFAtab[5][1] = 'E';

DFA\_finals = "EF";

N\_DFA\_states = 6;

N\_symbols = 2;

}

/\*

Get next-state string for current-state string.

\*/

void get\_next\_state(char \*nextstates, char \*cur\_states,

int dfa[STATES][SYMBOLS], int symbol)

{

int i, ch;

for (i = 0; i < strlen(cur\_states); i++)

\*nextstates++ = dfa[cur\_states[i]-'A'][symbol];

\*nextstates = '\0';

}

/\*

Get index of the equivalence states for state 'ch'.

Equiv. class id's are '0', '1', '2', ...

\*/

char equiv\_class\_ndx(char ch, char stnt[][STATES+1], int n)

{

int i;

for (i = 0; i < n; i++)

if (strchr(stnt[i], ch)) return i+'0';

return -1; /\* next state is NOT defined \*/

}

/\*

Check if all the next states belongs to same equivalence class.

Return value:

If next state is NOT unique, return 0.

If next state is unique, return next state --> 'A/B/C/...'

's' is a '0/1' string: state-id's

\*/

char is\_one\_nextstate(char \*s)

{

char equiv\_class; /\* first equiv. class \*/

while (\*s == '@') s++;

equiv\_class = \*s++; /\* index of equiv. class \*/

while (\*s) {

if (\*s != '@' && \*s != equiv\_class) return 0;

s++;

}

return equiv\_class; /\* next state: char type \*/

}

int state\_index(char \*state, char stnt[][STATES+1], int n, int \*pn,

int cur) /\* 'cur' is added only for 'printf()' \*/

{

int i;

char state\_flags[STATES+1]; /\* next state info. \*/

if (!\*state) return -1; /\* no next state \*/

for (i = 0; i < strlen(state); i++)

state\_flags[i] = equiv\_class\_ndx(state[i], stnt, n);

state\_flags[i] = '\0';

printf(" %d:[%s]\t--> [%s] (%s)\n",

cur, stnt[cur], state, state\_flags);

if (i=is\_one\_nextstate(state\_flags))

return i-'0'; /\* deterministic next states \*/

else {

strcpy(stnt[\*pn], state\_flags); /\* state-division info \*/

return (\*pn)++;

}

}

/\*

Divide DFA states into finals and non-finals.

\*/

int init\_equiv\_class(char statename[][STATES+1], int n, char \*finals)

{

int i, j;

if (strlen(finals) == n) { /\* all states are final states \*/

strcpy(statename[0], finals);

return 1;

}

strcpy(statename[1], finals); /\* final state group \*/

for (i=j=0; i < n; i++) {

if (i == \*finals-'A') {

finals++;

} else statename[0][j++] = i+'A';

}

statename[0][j] = '\0';

return 2;

}

/\*

Get optimized DFA 'newdfa' for equiv. class 'stnt'.

\*/

int get\_optimized\_DFA(char stnt[][STATES+1], int n,

int dfa[][SYMBOLS], int n\_sym, int newdfa[][SYMBOLS])

{

int n2=n; /\* 'n' + <num. of state-division info> \*/

int i, j;

char nextstate[STATES+1];

for (i = 0; i < n; i++) { /\* for each pseudo-DFA state \*/

for (j = 0; j < n\_sym; j++) { /\* for each input symbol \*/

get\_next\_state(nextstate, stnt[i], dfa, j);

newdfa[i][j] = state\_index(nextstate, stnt, n, &n2, i)+'A';

}

}

return n2;

}

/\*

char 'ch' is appended at the end of 's'.

\*/

void chr\_append(char \*s, char ch)

{

int n=strlen(s);

\*(s+n) = ch;

\*(s+n+1) = '\0';

}

void sort(char stnt[][STATES+1], int n)

{

int i, j;

char temp[STATES+1];

for (i = 0; i < n-1; i++)

for (j = i+1; j < n; j++)

if (stnt[i][0] > stnt[j][0]) {

strcpy(temp, stnt[i]);

strcpy(stnt[i], stnt[j]);

strcpy(stnt[j], temp);

}

}

/\*

Divide first equivalent class into subclasses.

stnt[i1] : equiv. class to be segmented

stnt[i2] : equiv. vector for next state of stnt[i1]

Algorithm:

- stnt[i1] is splitted into 2 or more classes 's1/s2/...'

- old equiv. classes are NOT changed, except stnt[i1]

- stnt[i1]=s1, stnt[n]=s2, stnt[n+1]=s3, ...

Return value: number of NEW equiv. classses in 'stnt'.

\*/

int split\_equiv\_class(char stnt[][STATES+1],

int i1, /\* index of 'i1'-th equiv. class \*/

int i2, /\* index of equiv. vector for 'i1'-th class \*/

int n, /\* number of entries in 'stnt' \*/

int n\_dfa) /\* number of source DFA entries \*/

{

char \*old=stnt[i1], \*vec=stnt[i2];

int i, n2, flag=0;

char newstates[STATES][STATES+1]; /\* max. 'n' subclasses \*/

for (i=0; i < STATES; i++) newstates[i][0] = '\0';

for (i=0; vec[i]; i++)

chr\_append(newstates[vec[i]-'0'], old[i]);

for (i=0, n2=n; i < n\_dfa; i++) {

if (newstates[i][0]) {

if (!flag) { /\* stnt[i1] = s1 \*/

strcpy(stnt[i1], newstates[i]);

flag = 1; /\* overwrite parent class \*/

} else /\* newstate is appended in 'stnt' \*/

strcpy(stnt[n2++], newstates[i]);

}

}

sort(stnt, n2); /\* sort equiv. classes \*/

return n2; /\* number of NEW states(equiv. classes) \*/

}

/\*

Equiv. classes are segmented and get NEW equiv. classes.

\*/

int set\_new\_equiv\_class(char stnt[][STATES+1], int n,

int newdfa[][SYMBOLS], int n\_sym, int n\_dfa)

{

int i, j, k;

for (i = 0; i < n; i++) {

for (j = 0; j < n\_sym; j++) {

k = newdfa[i][j]-'A'; /\* index of equiv. vector \*/

if (k >= n) /\* equiv. class 'i' should be segmented \*/

return split\_equiv\_class(stnt, i, k, n, n\_dfa);

}

}

return n;

}

void print\_equiv\_classes(char stnt[][STATES+1], int n)

{

int i;

printf("\nEQUIV. CLASS CANDIDATE ==>");

for (i = 0; i < n; i++)

printf(" %d:[%s]", i, stnt[i]);

printf("\n");

}

/\*

State-minimization of DFA: 'dfa' --> 'newdfa'

Return value: number of DFA states.

\*/

int optimize\_DFA(

int dfa[][SYMBOLS], /\* DFA state-transition table \*/

int n\_dfa, /\* number of DFA states \*/

int n\_sym, /\* number of input symbols \*/

char \*finals, /\* final states of DFA \*/

char stnt[][STATES+1], /\* state name table \*/

int newdfa[][SYMBOLS]) /\* reduced DFA table \*/

{

char nextstate[STATES+1];

int n; /\* number of new DFA states \*/

int n2; /\* 'n' + <num. of state-dividing info> \*/

n = init\_equiv\_class(stnt, n\_dfa, finals);

while (1) {

print\_equiv\_classes(stnt, n);

n2 = get\_optimized\_DFA(stnt, n, dfa, n\_sym, newdfa);

if (n != n2)

n = set\_new\_equiv\_class(stnt, n, newdfa, n\_sym, n\_dfa);

else break; /\* equiv. class segmentation ended!!! \*/

}

return n; /\* number of DFA states \*/

}

/\*

Check if 't' is a subset of 's'.

\*/

int is\_subset(char \*s, char \*t)

{

int i;

for (i = 0; \*t; i++)

if (!strchr(s, \*t++)) return 0;

return 1;

}

/\*

New finals states of reduced DFA.

\*/

void get\_NEW\_finals(

char \*newfinals, /\* new DFA finals \*/

char \*oldfinals, /\* source DFA finals \*/

char stnt[][STATES+1], /\* state name table \*/

int n) /\* number of states in 'stnt' \*/

{

int i;

for (i = 0; i < n; i++)

if (is\_subset(oldfinals, stnt[i])) \*newfinals++ = i+'A';

\*newfinals++ = '\0';

}

void main()

{

load\_DFA\_table();

print\_dfa\_table(DFAtab, N\_DFA\_states, N\_symbols, DFA\_finals);

N\_optDFA\_states = optimize\_DFA(DFAtab, N\_DFA\_states,

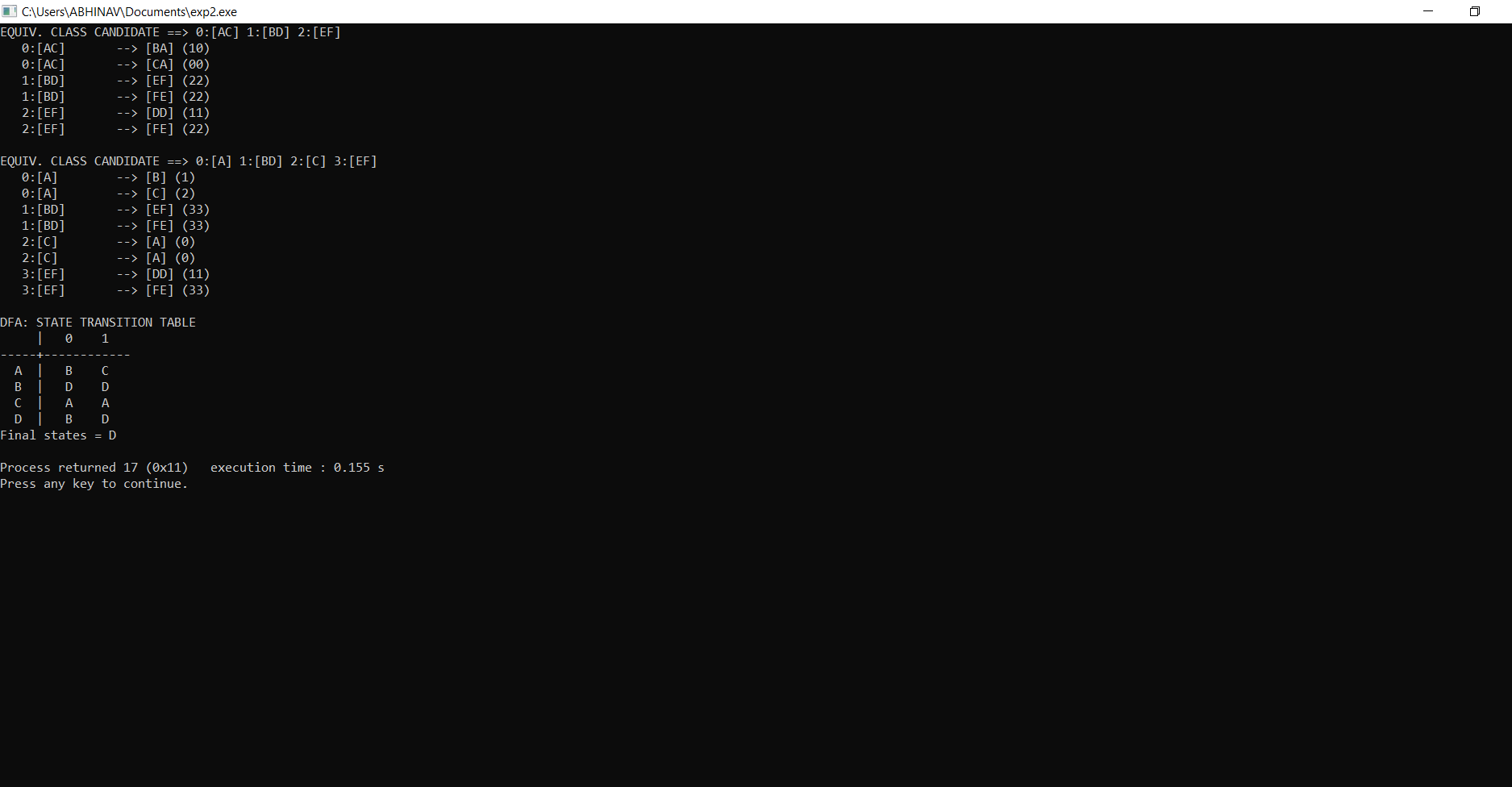
N\_symbols, DFA\_finals, StateName, OptDFA);

get\_NEW\_finals(NEW\_finals, DFA\_finals, StateName, N\_optDFA\_states);

print\_dfa\_table(OptDFA, N\_optDFA\_states, N\_symbols, NEW\_finals);

}

OUTPUT :-



RESULT:-

The C Program to minimize the number of states of a DFA

has been successfully Executed.