Name: Pradyumn R Pai
Roll No: 50
Class: CS7A

PROGRAM CODE

nfa_ds.c:

```
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

struct TransitionNode {
    int target_state;
    char input;
    struct TransitionNode* next;
};

struct State {
    int id;
    struct TransitionNode* transitionListHead;
    bool finalState;
};

struct NFA {
    int stateNum;
    char * inputAlphabet;
    struct State* stateList;
};

struct NFA* init_NFA(int n, char* inputAlphabet){
    struct NFA* out = malloc(sizeof(struct NFA));
    if (!out){
        return NULL; //failed allocation
    }
```

```
    out->stateNum = n;
    out->inputAlphabet = inputAlphabet;
    out->stateList = malloc(sizeof(struct State)*n);
    if (!out->stateList){
        free(out);
        return NULL;
    }

    for (int i=0;i<n;++i){
        out->stateList[i].id = i;
        out->stateList[i].transitionListHead = NULL;
        out->stateList[i].finalState = false;
    }

    return out;
}

void addTransitionNFA(struct NFA* n, int s, int t, char c){
    struct TransitionNode** head = &(n->stateList[s].transitionListHead);
    while (*head){
        if ((*head)->input==c && (*head)->target_state==t){
            return ; //avoid duplicates
        }
        head = &((*head)->next);
    }
    *head = malloc(sizeof(struct TransitionNode));
    if (!*head){
        return; // allocation failed
    }
    (*head)->target_state = t;
```

```
    (*head)->input = c;
    (*head)->next = NULL;
}

void freeStateNFA(struct State s){
    struct TransitionNode* head = s.transitionListHead;
    while (head){
        struct TransitionNode* next = head->next;
        free(head);
        head = next;
    }
}

void freeNFA(struct NFA* n){
    if (!n) return;
    for (int i=0;i<(n->stateNum);++i){
        freeStateNFA(n->stateList[i]);
    }

    free(n->inputAlphabet);
    free(n->stateList);
    free(n);
}

void printNFA(struct NFA* nfa){
    printf("The transition table is as follows:\n");
    int n = nfa->stateNum;
    int m = strlen(nfa->inputAlphabet);
    printf("\t");
    for (int i=0;i<m;++i){
        printf("%c\t",nfa->inputAlphabet[i]);
```

```
    }
    printf("epsilon\n");
    for (int i=0;i<n;++i){
        if (i==0){
            printf("->");
        }
        if (nfa->stateList[i].finalState){
            printf("*");
        }
        printf("q%d\t",i);
        struct State s = nfa->stateList[i];
        for (int j=0;j<=m;++j){
            char c = nfa->inputAlphabet[j];
            if (j==m){
                c = 'e';
            }
            for (struct TransitionNode *current=s.transitionListHead;current;current=current->next){
                if (current->input==c){
                    printf("q%d",current->target_state);
                }
            }
            printf("\t");
        }
        printf("\n");
    }
}



struct NFA* readNFA() {
    // read input
```

```
int n, m, t, f;
scanf("%d%d%d%d", &n, &f, &m, &t);
if (f<0 || f>n){
    printf("Invalid number of final states\n");
    return NULL;
}

int finalStates[f];
for (int i=0;i<f;++i){
    scanf("%d",finalStates+i);
    if (finalStates[i]<0 || finalStates[i]>=n){
        printf("Invalid final state %d\n",finalStates[i]);
        return NULL;
    }
}



char* inputChars = malloc(sizeof(char)*(m+1));
if (!inputChars) {
    printf("Failed to allocate memory for input characters\n");
    return NULL;
}

scanf("%s\n", inputChars);
if (strlen(inputChars) != m) {
    free(inputChars);
    printf("Input characters length mismatch\n");
    return NULL;
}

struct NFA *nfa = init_NFA(n,inputChars);
```

```
if (!nfa) {
    free(inputChars);
    printf("Failed to initialize NFA\n");
    return NULL;
}

for (int i=0;i<f;++i){
    nfa->stateList[finalStates[i]].finalState = true;
}

for (int i = 0; i < t; ++i) {
    int a, b;
    char c;
    scanf("q%d q%d %c\n", &a, &b, &c);
    if (a < 0 || a >= n || b < 0 || b >= n) {
        printf("Invalid transition from %d to %d\n", a, b);
        freeNFA(nfa);
        return NULL;
    }
    bool validChar = false;
    for (int j = 0; j < m; ++j) {
        if (inputChars[j] == c) {
            validChar = true;
            break;
        }
    }
    if (!validChar && c != 'e') { // 'e' for epsilon transition
        printf("Invalid input character '%c' for transition from %d to %d\n", c, a, b);
        freeNFA(nfa);
        return NULL;
    }
```

```
        addTransitionNFA(nfa, a, b, c);
    }
    return nfa;
}
```

dfa_ds.c:

```
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include "dsu.c"

struct DFA {
    int stateNum;
    bool* finalState;
    char * inputAlphabet;
    int ** transitionTable;
};

void freeDFA(struct DFA* dfa){
    if (!dfa) return;
    if (dfa->finalState){
        free(dfa->finalState);
    }
    if (dfa->transitionTable){
        int n = dfa->stateNum;
        for (int i=0;i<n;++i){
            if (dfa->transitionTable[i]){
                free(dfa->transitionTable[i]);
            }
        }
        free(dfa->transitionTable);
    }
```

```
    free(dfa);
}

struct DFA* init_DFA(int n, char* inputAlphabet){
    struct DFA* out = malloc(sizeof(struct DFA));
    if (!out){
        return NULL; //failed allocation
    }

    out->stateNum = n;
    out->inputAlphabet = inputAlphabet;
    int m = strlen(inputAlphabet);

    out->transitionTable = malloc(sizeof(int*)*n);
    if (!out->transitionTable){
        freeDFA(out);
        return NULL;
    }

    out->finalState = malloc(sizeof(bool)*n);
    if (!out->finalState){
        freeDFA(out);
    }
    for (int i=0;i<n;++i){
        out->finalState[i] = false;
    }

    for (int i=0;i<n;++i){
        out->transitionTable[i] = malloc(sizeof(int)*m);
        if (!out->transitionTable[i]){
            freeDFA(out);
```

```
            return NULL;
        }
        for (int j=0;j<m;++j){
            out->transitionTable[i][j] = i;
        }
    }

    return out;
}

int inputIndexDFA(struct DFA* dfa, char c){
    int m = strlen(dfa->inputAlphabet);
    for (int i=0;i<m;++i){
        if (c==dfa->inputAlphabet[i]){
            return i;
        }
    }
    return -1;
}

void addTransitionDFA(struct DFA* dfa, int s, int t, char c){
    int i = inputIndexDFA(dfa,c);
    if (i!=-1){
        dfa->transitionTable[s][i] = t;
    }
}



void printDFA(struct DFA* dfa){
    printf("The transition table is as follows:\n");
    int n = dfa->stateNum;
```

```
    int m = strlen(dfa->inputAlphabet);
    printf("\t");
    for (int i=0;i<m;++i){
        printf("%c\t",dfa->inputAlphabet[i]);
    }
    printf("\n");
    for (int i=0;i<n;++i){
        if (i==0){
            printf("->");
        }
        if (dfa->finalState[i]){
            printf("*");
        }
        printf("q%d\t",i);
        for (int j=0;j<m;++j){
            printf("q%d\t",dfa->transitionTable[i][j]);
        }
        printf("\n");
    }
}

struct DFA* readDFA() {
    // read input
    int n, f, m;
    scanf("%d%d%d", &n, &f, &m);
    if (f<0 || f>n){
        printf("Invalid number of final states\n");
        return NULL;
    }

    int finalStates[f];
```

```
for (int i=0;i<f;++i){
    scanf("%d",finalStates+i);
    if (finalStates[i]<0 || finalStates[i]>=n){
        printf("Invalid final state %d\n",finalStates[i]);
        return NULL;
    }
}


char* inputChars = malloc(sizeof(char)*(m+1));
if (!inputChars) {
    printf("Failed to allocate memory for input characters\n");
    return NULL;
}

scanf("%s\n", inputChars);
if (strlen(inputChars) != m) {
    free(inputChars);
    printf("Input characters length mismatch\n");
    return NULL;
}

struct DFA *dfa = init_DFA(n,inputChars);
if (!dfa) {
    free(inputChars);
    printf("Failed to initialize DFA\n");
    return NULL;
}

for (int i=0;i<f;++i){
    dfa->finalState[finalStates[i]] = true;
```

```
    }

    for (int i=0;i<n;++i){
        for (int j=0;j<m;++j) {
            scanf("%d",dfa->transitionTable[i]+j);
        }
    }

    return dfa;
}
```

dfa_conversion.c:

```
#include <stdio.h>
#include "enfa_functions.c"
#include "dfa_ds.c"

struct StateMappingNode {
    int nfa_value, dfa_value;
    int* transitions;
    struct StateMappingNode* next;
};

void freeStateMappingList(struct StateMappingNode* head){
    if (!head) return;
    freeStateMappingList(head->next);
    free(head->transitions);
    free(head);
}

void printStateMapping(struct StateMappingNode* head){
    for (struct StateMappingNode* current = head;current;current = current->next){
        printf("%d: ", current->dfa_value);
```

```
        for (int i=0,bm=current->nfa_value;bm>0;bm>>=1,++i){
            if (bm&1){
                printf("q%d",i);
            }
        }
        printf("\n");
    }
}

int stateCount = 0;

int transition(struct NFA* nfa, int state, int input){
    int out = 0;
    int copy = state;
    for (int bitCounter=0;state>0;++bitCounter,state >>= 1){
        if ((state&1)==0) continue;
        for (struct TransitionNode* current = nfa-
>stateList[bitCounter].transitionListHead;current;current = current->next){
            if (current->input!=nfa->inputAlphabet[input]) continue;
            out = out | (1<<(current->target_state));
        }
    }
    return out;
}

void recursiveConvert(struct NFA* nfa,struct StateMappingNode** head,int state){
    int m = strlen(nfa->inputAlphabet);

    struct StateMappingNode** indirect = head;

    while (*indirect){
```

```
        if ((*indirect)->nfa_value==state) return;
        indirect = &((*indirect)->next);
    }
    *indirect = malloc(sizeof(struct StateMappingNode));
    (*indirect)->dfa_value = stateCount++;
    (*indirect)->nfa_value = state;
    (*indirect)->transitions = malloc(sizeof(int)*m);
    (*indirect)->next = NULL;

    for (int i=0;i<m;++i){
        int t = transition(nfa,state,i);
        recursiveConvert(nfa,head,t);
        (*indirect)->transitions[i] = t;
    }
}

int stateMapping(struct StateMappingNode* head,int nfa_state){
    while(head){
        if (head->nfa_value==nfa_state){
            return head->dfa_value;
        }
        head = head->next;
    }
    return -1;
}

struct DFA* dfa_conversion(struct NFA* enfa){
    struct NFA* nfa = epsilon_removal(enfa);

    int n = nfa->stateNum;
    int m = strlen(nfa->inputAlphabet);
```

```
if (n>=32){
    printf("NFA with 32 or more states cannot be converted\n");
}

struct StateMappingNode* head = NULL;

recursiveConvert(nfa,&head,1);

char* inputAlphabet = malloc(strlen(nfa->inputAlphabet)*sizeof(char));
strcpy(inputAlphabet,nfa->inputAlphabet);
struct DFA* dfa = init_DFA(stateCount,inputAlphabet);
printf("Mapping:\n");
if (dfa){
    struct StateMappingNode* current = head;
    while (current){
        int s = current->dfa_value;
        for (int i=0;i<n;++i){
            if (nfa->stateList[i].finalState && ((current->nfa_value) & (1 << i))) {
                dfa->finalState[i] = true;
            }
        }
        for (int i=0;i<m;++i){
            int t = stateMapping(head,current->transitions[i]);
            dfa->transitionTable[s][i] = t;
        }
        current = current->next;
    }
}

freeStateMappingList(head);
freeNFA(nfa);
```

```
    return dfa;
}

int main(){
    struct NFA* nfa = readNFA();
    if (!nfa) {
        printf("NFA creation failed\n");
    }
    printf("For the NFA:\n");
    printNFA(nfa);

    struct DFA* dfa = dfa_conversion(nfa);
    if (!dfa){
        printf("DFA conversion failed\n");
    }

    printf("\n\nFor the DFA:\n");
    printDFA(dfa);

    freeDFA(dfa);
    freeNFA(nfa);
}
```

OUTPUT:

input.txt:

```
5 1 2 7
2
01
q0 q1 1
q1 q0 1
q0 q2 e
```

q2 q3 0

q3 q2 0

q2 q4 1

q4 q2 0

output:

For the NFA:

The transition table is as follows:

	0	1	epsilon
->q0		q1	q2
q1		q0	
*q2	q3	q4	
q3	q2		
q4	q2		

Mapping:

For the DFA:

The transition table is as follows:

	0	1
->*q0	q1	q5
q1	q2	q4
*q2	q1	q3
q3	q2	q4
q4	q4	q4
q5	q2	q6
q6	q1	q5