Experiment – 4

AIM- Write a program to convert given NFA to DFA.

Description -

NFA

An NFA can have zero, one or more than one move from a given state on a given input symbol. An NFA can also have NULL moves (moves without input symbol). On the other hand, DFA has one and only one move from a given state on a given input symbol.

Conversion of NFA to DFA

Suppose there is an NFA N < Q, ∑, q0, δ, F > which recognizes a language L. Then the DFA D < Q’, ∑, q0, δ’, F’ > can be constructed for language L as:

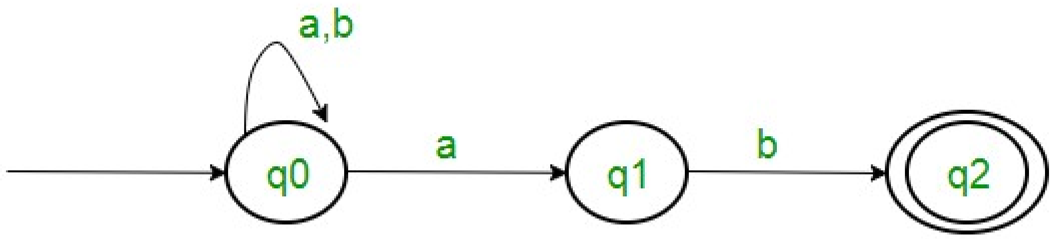
Step 1: Initially Q’ = ϕ.

Step 2: Add q0 to Q’.

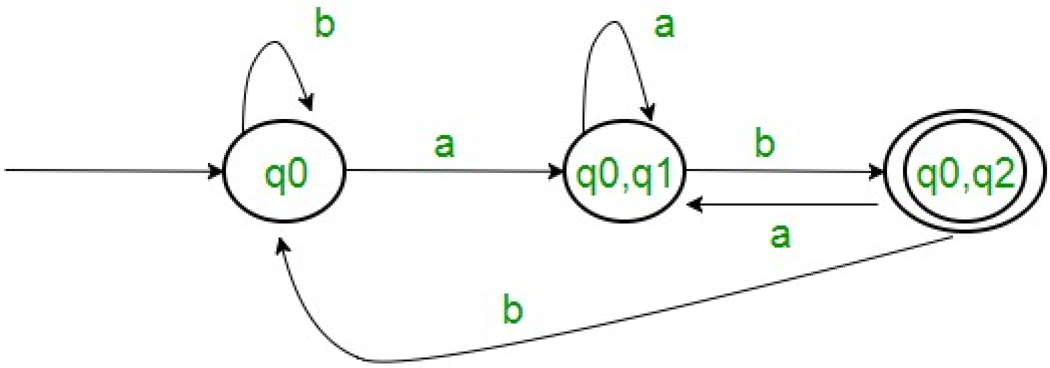
Step 3: For each state in Q’, find the possible set of states for each input symbol using transition function of NFA. If this set of states is not in Q’, add it to Q’.

Step 4: Final state of DFA will be all states with contain F (final states of NFA)

Consider the following NFA



The final DFA for above NFA



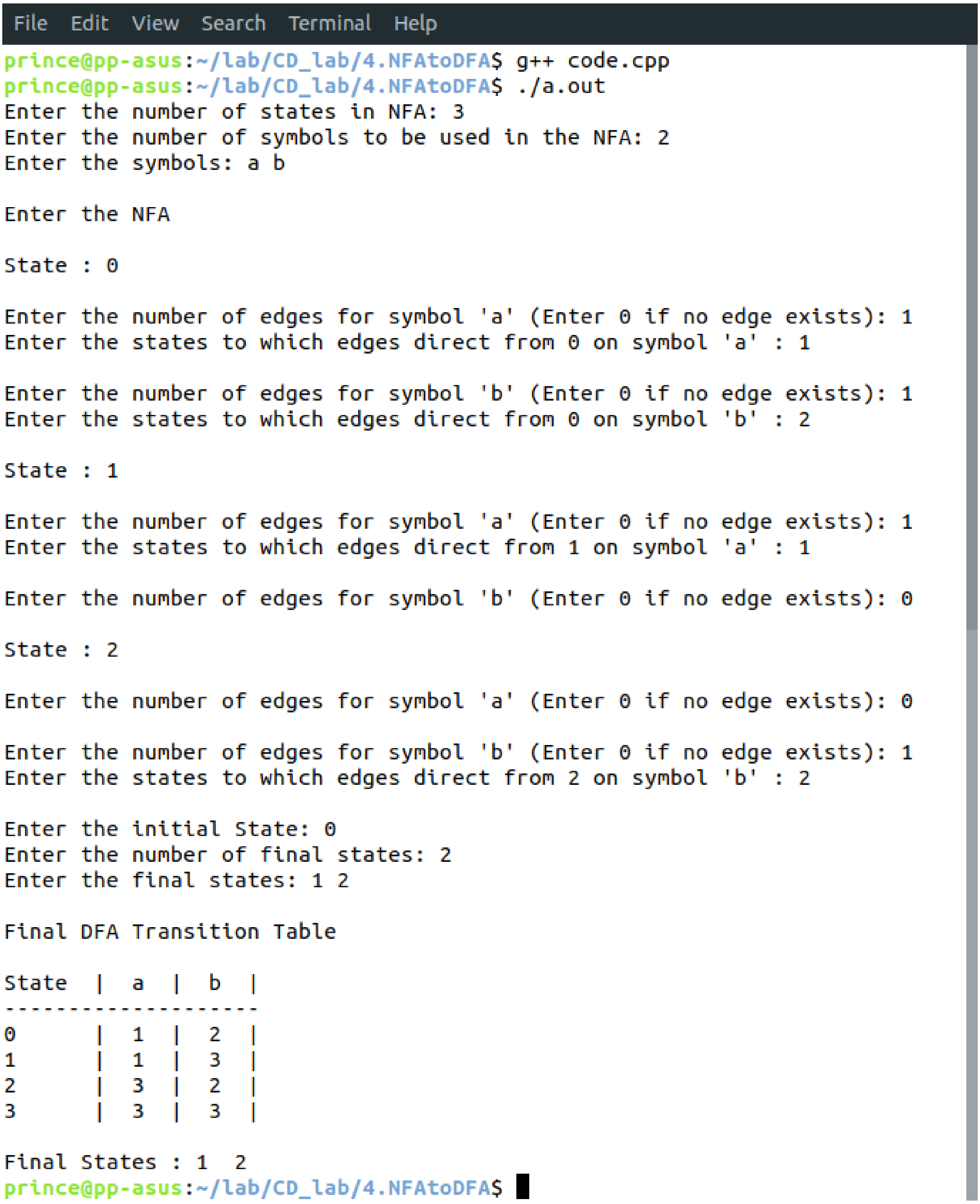
Code -

```
#include <bits/stdc++.h>

using namespace std;

void pushNullState(
    vector<
        pair<
            vector<int>,
            unordered_map<char, vector<int>>
        >
    > &NFA,
    int n,
    vector<char> &symbols
    ){
    unordered_map<char, vector<int>> edges;
    for (int i = 0; i < symbols.size(); ++i){
        vector<int> d;
        d.push_back(n);
        edges[symbols[i]] = d;
    }
    vector<int> source(1, n);
    NFA.push_back(make_pair(source, edges));
    return;
}

bool alreadyExists(
    vector<vector<int>> &temp,
    vector<int> &state){
    for (int i = 0; i < temp.size(); ++i){
        if (temp[i] == state){
            return true;
        }
    }
    return false;
}

void removeDuplicates(vector<int> &t){
    unordered_set<int> s;
    for (int i = 0; i < t.size(); ++i){
        s.insert(t[i]);
    }
    unordered_set<int>::iterator it = s.begin();
    t.clear();

    while (it != s.end()){
        t.push_back(*it);
        ++it;
    }
    sort(t.begin(), t.end());
    return;
}

vector<int> getDestinationStates(
    vector<
        pair<
            vector<int>,
            unordered_map<char, vector<int>>
        >
    > NFA,
    vector<int> state,
    char symbol
    ){
    vector<int> ans;
    for (int i = 0; i < NFA.size(); ++i){
        int temp = NFA[i].first[0];
        if (find(state.begin(), state.end(), temp) != state.end()){
            vector<int> tobeAdded = NFA[i].second[symbol];
            for (int j = 0; j < tobeAdded.size(); ++j){
                ans.push_back(tobeAdded[j]);
            }
        }
    }
    removeDuplicates(ans);
    return ans;
}

void PrintDFA(
    vector<
        pair<
            vector<int>,
            unordered_map<char, vector<int>>
        >
    > &dfa,
    vector<char> &symbols,
    vector<int> &final_states
    ){
    cout << endl;
    cout << "State";
    cout << "  |  ";
    for (int i = 0; i < symbols.size(); ++i){
        cout << symbols[i] << "  |  ";
    }
    cout << endl;

    for (int i = 0; i < 10 * symbols.size(); ++i){
        cout << "-";
    }
    cout << endl;

    for (int i = 0; i < dfa.size(); ++i){
        cout << dfa[i].first[0] << "      |  ";
        for (int j = 0; j < symbols.size(); ++j){
            cout << dfa[i].second[symbols[j]][0] << "  |  ";
        }
        cout << endl;
    }

    cout << endl;
    cout << "Final States : ";
    for (int i = 0; i < final_states.size(); ++i){
        cout << final_states[i] << "  ";
    }
    cout << endl;
    return;
}

void processEdges(
    unordered_map<
        char,
        vector<int>
    > &edges,
    vector<int> &state,
    vector<int> &new_state
    ){
```

```
    unordered_map<char, vector<int>>::iterator it = edges.begin();

    while (it != edges.end()){
        if (it->second == state){
            it->second = new_state;
        }
        ++it;
    }
    return;
}

vector<int> processDFA(
    vector<
        pair<
            vector<int>,
            unordered_map<char, vector<int>>
        >
    > &dfa,
    vector<int> final_states
    ){
    int stateNumber = 0;
    vector<int> final;
    for (int i = 0; i < dfa.size(); ++i){
        vector<int> state = dfa[i].first;
        vector<int> new_state(1, stateNumber++);

        for (int j = 0; j < dfa.size(); ++j){
            processEdges(dfa[j].second, state, new_state);
        }
        int ns = new_state[0];
        if (
            find(
                final_states.begin(),
                final_states.end(),
                ns
            ) != final_states.end()){
            final.push_back(ns);
        }
    }
    return final;
}
```

```
int main(){
    vector<
        pair<
            vector<int>,
            unordered_map<char, vector<int>>
        >
    > NFA;

    int N;
    cout << "Enter the number of states in NFA: ";
    cin >> N;

    cout << "Enter the number of symbols to be used in the NFA: ";
    int n_symbols;
    cin >> n_symbols;

    vector<char> symbols(n_symbols);
    cout << "Enter the symbols: ";
    for (int i = 0; i < n_symbols; ++i){
        cin >> symbols[i];
    }

    cout << "\nEnter the NFA " << endl;
    for (int i = 0; i < N; i++){
        cout << "\nState : " << i << endl;
        unordered_map<char, vector<int>> edges;
        vector<int> source(1, i);

         for (int j = 0; j < symbols.size(); ++j){
            cout<< "\nEnter the number of edges for symbol '"
                << symbols[j]
                << "' (Enter 0 if no edge exists): ";

            int n_edges;
            cin >> n_edges;
            vector<int> destination(n_edges);

             if (n_edges == 0){
                destination.push_back(N);
                edges[symbols[j]] = destination;
                continue;
            }
            cout<< "Enter the states to which edges direct from "
                << i
                << " on symbol '"
                << symbols[j]
                << "' : ";
            for (int k = 0; k < n_edges; ++k){
                cin >> destination[k];
            }
            edges[symbols[j]] = destination;
            edges[symbols[j]] = destination;
        }
        NFA.push_back(make_pair(source, edges));
    }
    cout << endl;

    pushNullState(NFA, N, symbols);

    int initial_state;
    cout << "Enter the initial State: ";
    cin >> initial_state;

    int n_final;
    cout << "Enter the number of final states: ";
    cin >> n_final;

    vector<int> final_states(n_final);
    cout << "Enter the final states: ";
    for (int i = 0; i < n_final; ++i){
        cin >> final_states[i];
    }

    vector<int> initial_states(1, initial_state);
    queue<vector<int>> Q;
    Q.push(initial_states);

    vector<
        pair<
            vector<int>,
            unordered_map<char, vector<int>>
        >
    > dfa;
    vector<vector<int>> temp;
    while (!Q.empty()){
        vector<int> state = Q.front();
        Q.pop();

        unordered_map<char, vector<int>> edges;
        for (int i = 0; i < symbols.size(); ++i){
            vector<int> destination =
            getDestinationStates(   NFA,
                                    state,
                                    symbols[i]
            );
            edges[symbols[i]] = destination;

            if (!alreadyExists(temp, destination)){
                Q.push(destination);
                temp.push_back(destination);
            }
        }
        dfa.push_back(make_pair(state, edges));
    }
    vector<int> dfaFinalStates = processDFA(dfa, final_states);
    cout << endl;

    cout << "Final DFA Transition Table" << endl;
    PrintDFA(dfa, symbols, dfaFinalStates);

    return 0;
}
```

Output –



```
prince@pp-asus:~/lab/CD_lab/4.NFAtoDFA$ g++ code.cpp
prince@pp-asus:~/lab/CD_lab/4.NFAtoDFA$ ./a.out
Enter the number of states in NFA: 3
Enter the number of symbols to be used in the NFA: 2
Enter the symbols: a b

Enter the NFA

State : 0

Enter the number of edges for symbol 'a' (Enter 0 if no edge exists): 1
Enter the states to which edges direct from 0 on symbol 'a' : 1

Enter the number of edges for symbol 'b' (Enter 0 if no edge exists): 1
Enter the states to which edges direct from 0 on symbol 'b' : 2

State : 1

Enter the number of edges for symbol 'a' (Enter 0 if no edge exists): 1
Enter the states to which edges direct from 1 on symbol 'a' : 1

Enter the number of edges for symbol 'b' (Enter 0 if no edge exists): 0

State : 2

Enter the number of edges for symbol 'a' (Enter 0 if no edge exists): 0

Enter the number of edges for symbol 'b' (Enter 0 if no edge exists): 1
Enter the states to which edges direct from 2 on symbol 'b' : 2

Enter the initial State: 0
Enter the number of final states: 2
Enter the final states: 1 2

Final DFA Transition Table

State  |  a  |  b  |
--------------------
0      |  1  |  2  |
1      |  1  |  3  |
2      |  3  |  2  |
3      |  3  |  3  |

Final States : 1  2
prince@pp-asus:~/lab/CD_lab/4.NFAtoDFA$ 
```

Learnings – We came to know about NFA and DFA and we learnt how to convert an NFA to a DFA with the help of a transition table and made a program to do so.