College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

September 16, 2020

Exp 7

1 DFA Minimisation

1.1 Aim

Write program to minimize any given DFA.

1.2 Theory

DFA

- In a DFA, for a particular input character, the machine goes to one state only.
- A transition function is defined on every state for every input symbol.
- Also in DFA null (or ϵ) move is not allowed,
- i.e., DFA cannot change state without any input character.
- DFA consists of 5 tuples Q, Σ , q, F, δ.
  - Q : set of all states.
  - Σ : set of input symbols. (Symbols which machine takes as input)
  - q : Initial state. (Starting state of a machine)
  - F : set of final state.
  - δ : Transition Function, defined as δ : Q X $\Sigma \rightarrow$ Q.

Minimization of DFA

Suppose there is a DFA D $< Q, \Sigma, q0, \delta, F >$which recognizes a language L.
Then the minimized DFA D $< Q^{'}, \Sigma, q0, \delta^{'}, F^{'} >$ can be constructed for language L as:

1. We will divide Q (set of states) into two sets. One set will contain all final states and other set will contain non-final states. This partition is called P0.
2. Initialize k = 1
3. Find Pk by partitioning the different sets of Pk-1. In each set of Pk-1, we will take all possible pair of states. If two states of a set are distinguishable, we will split the sets into different sets in Pk.
4. Stop when Pk = Pk-1 (No change in partition)
5. All states of one set are merged into one. No. of states in minimized DFA will be equal to no. of sets in Pk.

1.3 Algorithm

Algorithm 1: Algorithm to minimize states in a DFA

```
function MINIMIZEDFA ( dfa )
    Initialize an empty object of type dfa with variable name minDfa
    Initialize minDfa . num_alphabets = dfa . num_alphabets
    Initialize a matrix m of size a . num_states x a . num_states and
    set every cell in the matrix to 0
    Initialize a flag variable f to 1
    For all state pairs (x , y ) , set m [ x ][ y ] = 1 if x is a final
    state and y is a non - final state or vice - versa ( Choose either
    upper or lower triangle of the matrix ) .
    While f ! = 0
        Set f to 0
        For all states i from 0 to dfa . num_states
        For all states j from i + 1 to dfa . num_states
            If for any symbol u in Z , m [ i ][ j ] = 0 and m [ dfa .
            transitiontable [ i ][ u ]][ dfa . transitiontable [ j ][ u ]] = 1 ,
            Then Set m[i][j] = 1 and f = 1
        Represent those pair of states (a , b ) which has m [ a ][ b ] = 0 by
        a single state a in the minimized DFA minDfa .
    Return minDfa as the minimised DFA .
end function
```

1.4 Code

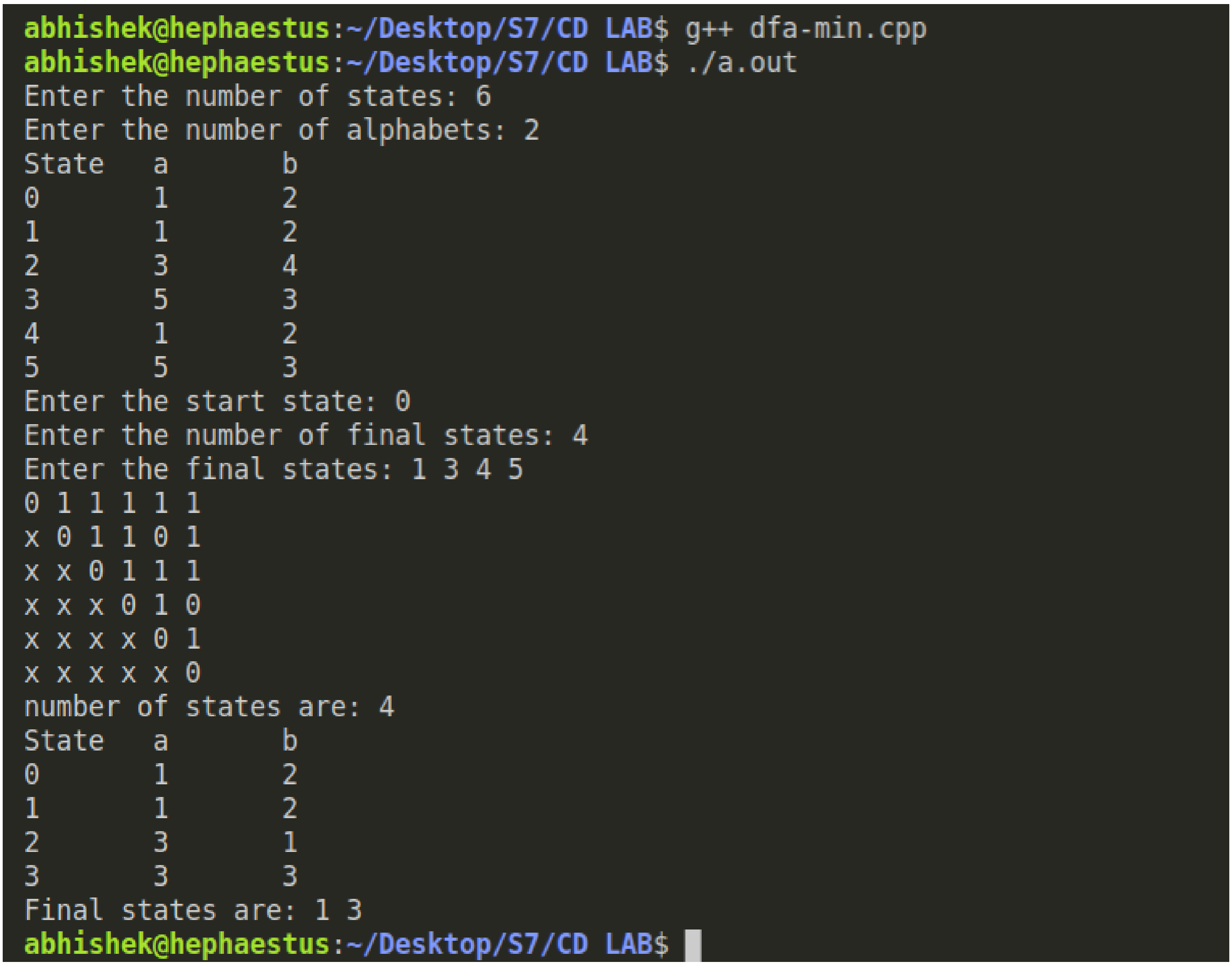
```cpp
#include <bits/stdc++.h>
using namespace std;
void print_header(int m)
{
    cout << "State";
    for (int j = 0; j < m; ++j)
    {
        char ch = char(int(’a’) + j);
        cout << "\t" << ch;
    }
    cout << endl;
}

vector<vector<int>> input_dfa(int *row, int *column)
{
    int n, m;
    cout << "Enter the number of states: ";
    cin >> n;
    cout << "Enter the number of alphabets: ";
    cin >> m;
    vector<vector<int>> dfa(n, vector<int>(m, 0));
    print_header(m);
    for (int i = 0; i < n; ++i)
    {
        cout << i << "\t";
        for (int j = 0; j < m; ++j)
        {
            cin >> dfa[i][j];
        }
    }
    *row = n;
    *column = m;

    return dfa;
}
int main()
{
    int n, m;
    vector<vector<int>> dfa = input_dfa(&n, &m);
    vector<vector<int>> matrix(n, vector<int>(m, 0));
    int s, last, temp;
    cout << "Enter the start state: ";
    cin >> s;
    set<int> f;
    cout << "Enter the number of final states: ";
    cin >> last;
    cout << "Enter the final states: ";
    for (int i = 0; i < last; ++i)
    {
        cin >> temp;
        f.insert(temp);
    }
    //differentiate final and non finale
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            if (f.find(i) != f.end() && f.find(j) == f.end())
            {
                matrix[i][j] = 1;
            }
            if (f.find(i) == f.end() && f.find(j) != f.end())
            {
                matrix[i][j] = 1;
            }
        }
    }
    int flag = 1;
    while (flag)
    {
        flag = 0;
        for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < n; ++j)
            {
                for (int k = 0; k < m; ++k)
                {
                    if (matrix[i][j] != 1)
                        if (matrix[dfa[i][k]][dfa[j][k]] == 1)
                        {
                            matrix[i][j] = 1;
                            flag = 1;
                            break;
                        }
                }
            }
        }
    }
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            if (i > j)
            {
                cout << "x"
                     << " ";
            }
            else
            {
                cout << matrix[i][j] << " ";
            }
        }
        cout << endl;
    }
    int num_state = 0;
    vector<int> visited(n, 0);
    vector<set<int>> minimised;
    unordered_map<int, int> mapping;
    for (int i = 0; i < n; ++i)
    {
        set<int> new_state;
        if (visited[i] != 1)
        {
            new_state.insert(i);
            mapping[i] = num_state;
            for (int j = i + 1; j < n; ++j)
            {
                if (matrix[i][j] == 0)
                {
                    new_state.insert(j);
                    mapping[j] = num_state;
                    visited[j] = 1;
                }
            }
            minimised.push_back(new_state);
            num_state++;
        }
    }
    cout << "number of states are: " << num_state << endl;

    print_header(m);
    for (auto x : minimised)
    {
        for (auto y : x)
        {
            cout << mapping[y] << "\t";
            for (int j = 0; j < m; ++j)
            {
                cout << mapping[dfa[y][j]] << "\t";
            }
            break;
        }
        cout << endl;
    }
    cout << "Final states are: ";
    for (auto x : minimised)
    {
        for (auto y : x)
        {
            if (f.find(y) != f.end())
            {
                cout << mapping[y] << " ";
                break;
            }
        }
    }
    cout << endl;
    return 0;
}
```

1.5 Output



```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ dfa-min.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 6
Enter the number of alphabets: 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Enter the start state: 0
Enter the number of final states: 4
Enter the final states: 1 3 4 5
0 1 1 1 1 1
x 0 1 1 0 1
x x 0 1 1 1
x x x 0 1 0
x x x x 0 1
x x x x x 0
number of states are: 4
State   a       b
0       1       2
1       1       2
2       3       1
3       3       3
Final states are: 1 3
abhishek@hephaestus:~/Desktop/S7/CD LAB$
```

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ dfa-min.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 6
Enter the number of alphabets: 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Enter the start state: 0
Enter the number of final states: 4
Enter the final states: 1 3 4 5
0 1 1 1 1 1
x 0 1 1 0 1
x x 0 1 1 1
x x x 0 1 0
x x x x 0 1
x x x x x 0
number of states are: 4
State   a       b
0       1       2
1       1       2
2       3       1
```

```
3       3       3
Final states are: 1 3
```

1.6 Result

Implemented the program to minimise a DFA in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.