**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY**

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

# LAB - ASSIGNMENT- 1

## COMPILER CONSTRUCTION

## UCS802

**SUBMITTED BY-**

**GOURISH SINGLA**

**101803698**

**COE-15**

**SUBMITTED TO-**

**Dr. RAJ KUMAR TEKCHANDANI**

**Code:**

```
#include<bits/stdc++.h>
using namespace std;

class node
{
        public:
                int my_val;
                int visited=0;
                char read1,read2;
                node* next1=NULL;
                node* next2=NULL;
};
int alpha=0;
pair<node* , node*> standalone(char ch)
{
        node* head=new node;
        head->my_val=alpha++;
        node* tail=new node;
        tail->my_val=alpha++;

        head->next1=tail;
        head->read1=ch;
        pair<node* , node*> PAIR1;
   PAIR1.first=head;
   PAIR1.second =tail;
        return PAIR1;
}
```

```
pair<node* , node*> addition(pair<node* , node*> pr1,pair<node* , node*> pr2)
{
        node* head=new node;
        head->my_val=alpha++;
        node* tail=new node;
        tail->my_val=alpha++;

        head->next1=pr1.first;
        head->read1='z';

        head->next2=pr2.first;
        head->read2='z';

        pr1.second->next1=tail;
        pr1.second->read1='z';

        (pr2.second)->next1=tail;
        (pr2.second)->read1='z';

        pair<node* , node*> PAIR1;
  PAIR1.first=head;
  PAIR1.second =tail;
        return PAIR1;
}
```

```
pair<node* , node*> closure(pair<node* , node*> pr)
{
            node* head=new node;
            head->my_val=alpha++;
            node* tail=new node;
            tail->my_val=alpha++;

            head->next1=pr.first;
            head->read1='z';

            head->next2=tail;
            head->read2='z';

            (pr.second)->next1=tail;
            (pr.second)->read1='z';

            (pr.second)->next2=pr.first;
            (pr.second)->read2='z';

            pair<node* , node*> PAIR1;
    PAIR1.first=head;
    PAIR1.second=tail;
            return PAIR1;
}
```

```
pair<node* , node*> concatenate(pair<node* , node*> oldnode, pair<node* , node*> newnode)
{
        (oldnode.second)->next1=newnode.first;
        (oldnode.second)->read1='z';
        pair<node* , node*> PAIR1;
   PAIR1.first=oldnode.first;
   PAIR1.second=newnode.second;
        return PAIR1;
}

pair<node* , node*> nfa(string s)
{
        vector<char> st;
        vector< pair<node* , node*> > loners;

        for(int i=0;i<s.length();i++)
        {
                if(s[i]=='(')
                {
                        st.push_back(s[i]);
                }
                else if(s[i]==')')
                {
                        while(st.back()!='(')
                        if(st.back()=='|')
                        {
                                st.pop_back();
                                pair<node* , node*> pr1=loners.back();
```

```
            loners.pop_back();
            pair<node* , node*> pr2=loners.back();
            loners.pop_back();
            pair<node* , node*> pr=addition(pr2,pr1);
            loners.push_back(pr);
        }
        st.pop_back();
    }
    else if(s[i]=='*')
    {
        pair<node* , node*> pr1=loners.back();
        loners.pop_back();
        pair<node* , node*> pr=closure(pr1);
        loners.push_back(pr);
    }
    else if(s[i]=='|')
    {
        st.push_back(s[i]);
    }
    else
    {
        pair<node* , node*> pr=standalone(s[i]);
        loners.push_back(pr);
    }
}

while(loners.size()>1)
{
        pair<node* , node*> pr1=loners.back();
        loners.pop_back();
        pair<node* , node*> pr2=loners.back();
        loners.pop_back();
        pair<node* , node*> pr=concatenate(pr2,pr1);
        loners.push_back(pr);
    }
    return loners.back();
}

class nfa_nodes
{
    public:
        int start;
        char read;
        int end;
};

vector<nfa_nodes> get_nodes(pair<node* , node*> pr)
{
    vector<nfa_nodes> all_nfa_nodes;
    vector<node* > st;

    node* head=pr.first;
    node* tail=pr.second;

    st.push_back(head);
```

```
while(st.size()>0)
{
    node* temp=st.back();
    if(temp->visited==0)
    {
        st.pop_back();
        if(temp->next1)
        {
            nfa_nodes a;
            a.start=temp->my_val;
            a.read=temp->read1;
            a.end=temp->next1->my_val;
            all_nfa_nodes.push_back(a);
            st.push_back(temp->next1);
        }
        if(temp->next2)
        {
            nfa_nodes a;
            a.start=temp->my_val;
            a.read=temp->read2;
            a.end=temp->next2->my_val;
            all_nfa_nodes.push_back(a);
            st.push_back(temp->next2);
        }
        temp->visited=1;
    }
```

```
        else
        {
            st.pop_back();
        }

    }
    return all_nfa_nodes;
}

void print_nfa_nodes(vector<nfa_nodes> nodes)
{
    for(int i=0;i<nodes.size();i++)
    {
        nfa_nodes a=nodes[i];
        cout<<a.start<<" - "<<a.read<<" -> "<<a.end<<endl;
    }
    cout<<endl<<"Start: "<<nodes[0].start<<endl;
    cout<<"Final: "<<alpha-1<<endl;
}

vector<int> closure(vector<int> nodes,char ch,vector<nfa_nodes> nfa)
{
    vector<int> res,st;

    for(int j=0;j<nodes.size();j++)
    {
        st.push_back(nodes[j]);
        res.push_back(nodes[j]);
        while(st.size()>0)
        {
            int x=st.back();
            st.pop_back();
            for(int i=0;i<nfa.size();i++)
            {
                if(nfa[i].start==x and nfa[i].read==ch)
                {
                    st.push_back(nfa[i].end);
                    res.push_back(nfa[i].end);
                }
            }
        }
    }
    return res;
}

class dfa_nodes
{
    public:
        vector<int> start;
        char read;
        vector<int> next;
};
```

```
vector<int> after_reading_dfa(vector<nfa_nodes> nfa,char ch,vector<int> cl)
{
        vector<int> res;
        for(int i=0;i<cl.size();i++)
        {
                for(int j=0;j<nfa.size();j++)
                {
                        if(cl[i]==nfa[j].start and nfa[j].read==ch)
                        {
                                res.push_back(nfa[j].end);
                        }
                }
        }
        return res;
}
bool is_present(dfa_nodes a,vector<dfa_nodes> dfa)
{
        for(int i=0;i<dfa.size();i++)
        if(a.start==dfa[i].start and a.read==dfa[i].read and a.next==dfa[i].next)
        return true;
        return false;
}
vector<dfa_nodes> nfa_to_dfa(vector<nfa_nodes> nfa)
{
        vector<dfa_nodes> dfa;
        vector<int> start;
        start.push_back(nfa[0].start);
        vector<int> cl=closure(start,'z',nfa);
```

```
vector< vector<int> > st;
st.push_back(cl);
while(st.size()>0)
{
        vector<int> x=st.back();
        st.pop_back();
        vector<int> after_a=after_reading_dfa(nfa,'a',x);
        vector<int> cl_a=closure(after_a,'z',nfa);
        sort(x.begin(),x.end());
        sort(cl_a.begin(),cl_a.end());
        dfa_nodes a;
        a.start=x;
        a.read='a';
        a.next=cl_a;
        if(is_present(a,dfa)==false)
        {
                dfa.push_back(a);
                st.push_back(cl_a);
        }

        vector<int> after_b=after_reading_dfa(nfa,'b',x);
        vector<int> cl_b=closure(after_b,'z',nfa);
        sort(x.begin(),x.end());
        sort(cl_b.begin(),cl_b.end());
        dfa_nodes b;
        b.start=x;
        b.read='b';
        b.next=cl_b;
```

```
                if(is_present(b,dfa)==false)
                {
                        dfa.push_back(b);
                        st.push_back(cl_b);
                }
        }
        return dfa;
}

vector<int> first,final;

void print_dfa_nodes(vector<dfa_nodes> nodes)
{
        //cout<<nodes.size();
        vector<int> final_states;
        for(int i=0;i<nodes.size();i++)
        {
                dfa_nodes a=nodes[i];
                cout<<"[ ";
                for(int j=0;j<a.start.size();j++)
                {
                        cout<<(a.start)[j]<<" ";
                }
                cout<<"] - "<<a.read<<" -> ";
                cout<<"[ ";
                for(int j=0;j<a.next.size();j++)
                {
                        if((a.next)[j]==alpha-1)
```

}

```
            final_states.push_back(i);

            cout<<(a.next)[j]<<" ";

        }

        cout<<"]"<<endl;

    }

    first=nodes[0].start;

    cout<<endl<<"Start: ";

    cout<<"[ ";

    for(int j=0;j<first.size();j++)

    {

        cout<<first[j]<<" ";

    }

    cout<<"] "<<endl;


    cout<<"Final: ";

    cout<<"[ ";

    for(int i=0;i<final_states.size();i++)

    {

        for(int j=0;j<nodes[final_states[i]].next.size();j++)

        {

            final.push_back((nodes[final_states[i]].next)[j]);

            cout<<(nodes[final_states[i]].next)[j]<<" ";

        }

        cout<<"]"<<endl;

    }

    sort(final.begin(),final.end());

}
```

```
class transitions
{
    public:
        vector<int> state;
        vector<int> read_a;
        vector<int> read_b;
};

vector<transitions> transition_table;

vector<int> after_reading(vector<int> state,vector<dfa_nodes> dfa,char ch)
{
    for(int i=0;i<dfa.size();i++)
    {
        if(dfa[i].start==state and dfa[i].read==ch)
        {
            return dfa[i].next;
        }
    }

}
```

```
int get_index(vector<int> state,vector<vector<vector<int> > > equivalence)
{
    int f=0;
    for(int i=0;i<equivalence.size();i++)
    {
        for(int j=0;j<equivalence[i].size();j++)
        {
            if(equivalence[i][j]==state)
            {
                f=1;
                break;
            }
        }
        if(f==1)
        return i;
    }
}

bool check_equivalence(vector<vector<vector<int> > > equivalence,vector<int> equi,vector<int> next_equi,vector<transitions> transition_table)
{
    int a_index_0,b_index_0,a_index_1,b_index_1;
    for(int i=0;i<transition_table.size();i++)
    {
        if(transition_table[i].state==equi)
        {
            a_index_0=get_index(transition_table[i].read_a,equivalence);
            b_index_0=get_index(transition_table[i].read_b,equivalence);
```

```
		}
		if(transition_table[i].state==next_equi)
		{
			a_index_1=get_index(transition_table[i].read_a,equivalence);
			b_index_1=get_index(transition_table[i].read_b,equivalence);
		}
	}
	if((a_index_0==a_index_1) and (b_index_0==b_index_1))
	{
		return true;
	}
	return false;
}

vector<vector<vector<int> > > minimized_dfa(vector<dfa_nodes> dfa)
{
	vector< vector<int> > distinct_nodes;
	for(int i=0;i<dfa.size();i++)
	{
		if(find(distinct_nodes.begin(),distinct_nodes.end(),dfa[i].start)==distinct_nodes.end())
		{
			distinct_nodes.push_back(dfa[i].start);
		}
		if(find(distinct_nodes.begin(),distinct_nodes.end(),dfa[i].next)==distinct_nodes.end())
		{
			distinct_nodes.push_back(dfa[i].next);
		}
	}
```

```
vector<vector<vector<int> > > equivalence;
equivalence.push_back({});
equivalence.push_back({});
for(int i=0;i<distinct_nodes.size();i++)
{
        if(distinct_nodes[i]==final)
        {
                equivalence[1].push_back(distinct_nodes[i]);
        }
        else
        {
                equivalence[0].push_back(distinct_nodes[i]);
        }
}

for(int i=0;i<distinct_nodes.size();i++)
{
        vector<int> readA=after_reading(distinct_nodes[i],dfa,'a');
        vector<int> readB=after_reading(distinct_nodes[i],dfa,'b');
        transitions t;
        t.state=distinct_nodes[i];
        t.read_a=readA;
        t.read_b=readB;
        transition_table.push_back(t);
}

cout<<"Transtion Table: "<<endl;
for(int i=0;i<transition_table.size();i++)
```

```
{
        cout<<"State: "<<"[";
        for(int j=0;j<transition_table[i].state.size();j++)
        cout<<transition_table[i].state[j]<<" ";
        cout<<"]                        "<<"Read_a: "<<"[";
        for(int j=0;j<transition_table[i].read_a.size();j++)
        cout<<transition_table[i].read_a[j]<<" ";
        cout<<"]                        "<<"Read_b: "<<"[";
        for(int j=0;j<transition_table[i].read_b.size();j++)
        cout<<transition_table[i].read_b[j]<<" ";
        cout<<"]";
        cout<<endl;
}
cout<<endl;

vector<vector<vector<int> > > next_equivalence;
int al=0;
while(1)
{
        vector<vector<vector<int> > > alpha;
        for(int i=0;i<equivalence.size();i++)
        {
                for(int j=0;j<equivalence[i].size();j++)
                {
                        if(alpha.size()==0)
                        {
                                vector<vector<int> > temp;
                                temp.push_back(equivalence[i][j]);
                alpha.push_back(temp);
                continue;
            }
            bool t=false;
            for(int k=0;k<alpha.size();k++)
            {
t=check_equivalence(equivalence,equivalence[i][j],alpha[k][0],transition_table);
                if(t==true)
                {
                    alpha[k].push_back(equivalence[i][j]);
                    break;
                }
            }
            if(t==false)
            {
                vector<vector<int> > t;
                t.push_back(equivalence[i][j]);
                alpha.push_back(t);
            }
        }
        for(int n=0;n<alpha.size();n++)
        {
            next_equivalence.push_back(alpha[n]);
        }
        alpha.clear();
    }
```

```
                if(equivalence==next_equivalence)
                break;
                else
                {
                        equivalence=next_equivalence;
                        next_equivalence.clear();
                }
        }
        return next_equivalence;
}

class minimized_dfa_nodes
{
        public:
                vector<vector<int> > start;
                char read;
                vector<vector<int> > end;
};

vector<minimized_dfa_nodes> get_minimized_dfa(vector<vector<vector<int> > >
minimized_equivalence)
{
        vector<minimized_dfa_nodes> res;
        for(int i=0;i<minimized_equivalence.size();i++)
        {
                vector<vector<int> > s,t;
                for(int j=0;j<minimized_equivalence[i].size();j++)
                {
```

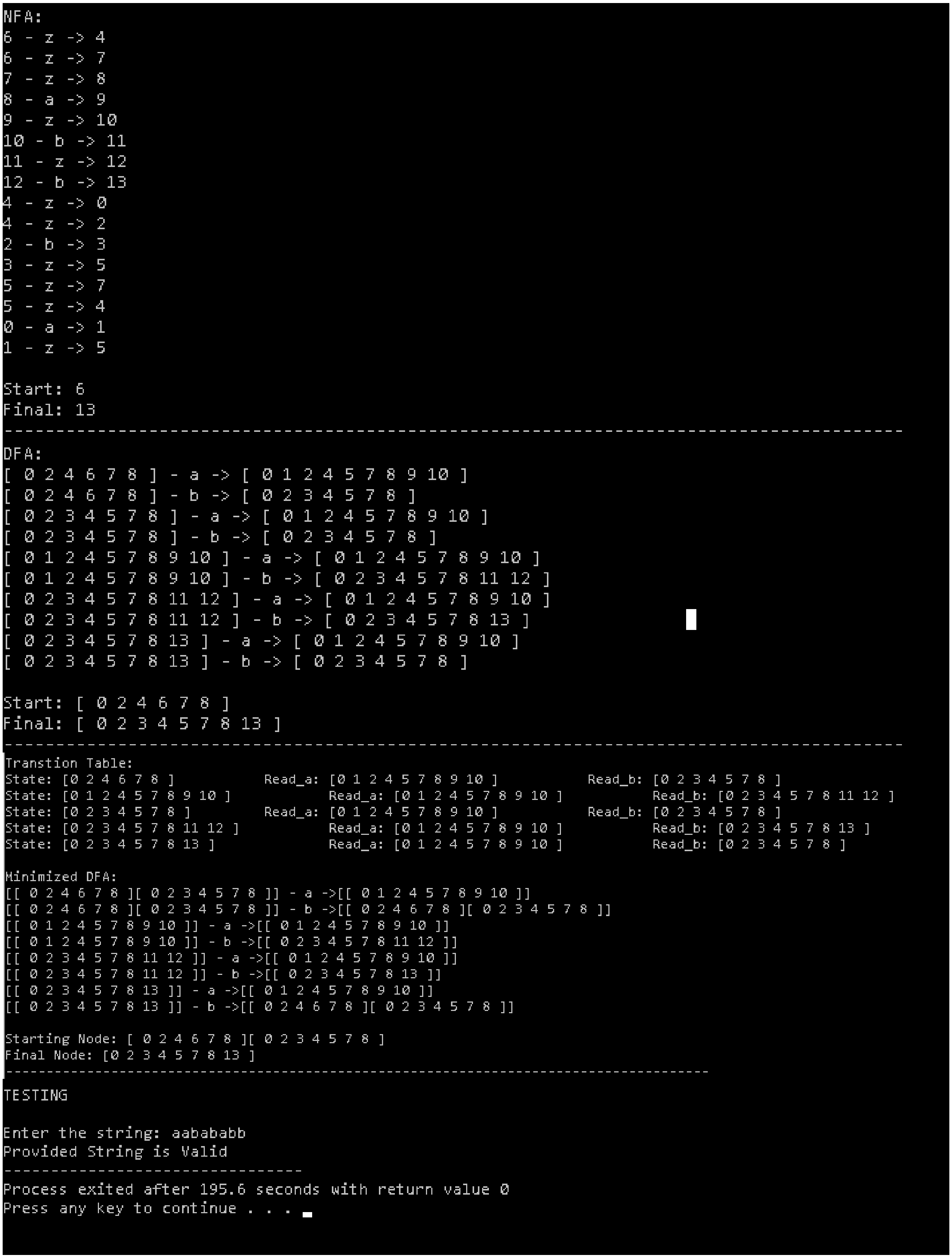
```
        s.push_back(minimized_equivalence[i][j]);
    }
    char r='a';
    for(int j=0;j<transition_table.size();j++)
    {
        if(transition_table[j].state==minimized_equivalence[i][0])
        {
            int f;
            for(int n=0;n<minimized_equivalence.size();n++)
            {
                for(int m=0;m<minimized_equivalence[n].size();m++)
                {
if(minimized_equivalence[n][m]==transition_table[j].read_a)
                    {
                        f=n;
                        break;
                    }
                }
                if(f==n)
                break;
            }
            for(int n=0;n<minimized_equivalence[f].size();n++)
            {
                t.push_back(minimized_equivalence[f][n]);
            }
            break;
        }
    }
    minimized_dfa_nodes ap;
    ap.start=s;
    ap.end=t;
    ap.read=r;
    res.push_back(ap);

    t.clear();
    r='b';
    for(int j=0;j<transition_table.size();j++)
    {
        if(transition_table[j].state==minimized_equivalence[i][0])
        {
            int f;
            for(int n=0;n<minimized_equivalence.size();n++)
            {
                for(int m=0;m<minimized_equivalence[n].size();m++)
                {
if(transition_table[j].read_b==minimized_equivalence[n][m])
                    {
                        f=n;
                        break;
                    }
                }
            }
            for(int n=0;n<minimized_equivalence[f].size();n++)
            {
                    t.push_back(minimized_equivalence[f][n]);
                }
                break;
            }
        }
        ap.end=t;
        ap.read=r;
        res.push_back(ap);
    }
    return res;
}

vector<vector<int> > start_node,final_node;

void print_minimized_dfa(vector<minimized_dfa_nodes> dfa)
{
    for(int i=0;i<dfa.size();i++)
    {
        cout<<"[";
        for(int j=0;j<dfa[i].start.size();j++)
        {
            cout<<"[ ";
            for(int k=0;k<dfa[i].start[j].size();k++)
            {
                cout<<dfa[i].start[j][k]<<" ";
            }
            cout<<"]";
        }
    cout<<"] - "<<dfa[i].read<<" ->";
    cout<<"[";
    for(int j=0;j<dfa[i].end.size();j++)
    {
        cout<<"[ ";
        for(int k=0;k<dfa[i].end[j].size();k++)
        {
            cout<<dfa[i].end[j][k]<<" ";
        }
        cout<<"]";
    }
    cout<<"]"<<endl;
}

for(int i=0;i<dfa.size();i++)
{
    for(int j=0;j<dfa[i].start.size();j++)
    {
        if(dfa[i].start[j]==first)
        {
            start_node=dfa[i].start;
            break;
        }
    }
}
cout<<endl;
cout<<"Starting Node: ";
for(int i=0;i<start_node.size();i++)
```

```
{
    cout<<"[ ";
    for(int j=0;j<start_node[i].size();j++)
    {
        cout<<start_node[i][j]<<" ";
    }
    cout<<"]";
}
cout<<endl;
for(int i=0;i<dfa.size();i++)
{
    for(int j=0;j<dfa[i].start.size();j++)
    {
        if(dfa[i].start[j]==final)
        {
            final_node=dfa[i].start;
            break;
        }
    }
}
cout<<"Final Node: ";
for(int i=0;i<final_node.size();i++)
{
    cout<<"[";
    for(int j=0;j<final_node[i].size();j++)
    {
        cout<<final_node[i][j]<<" ";
    }
```

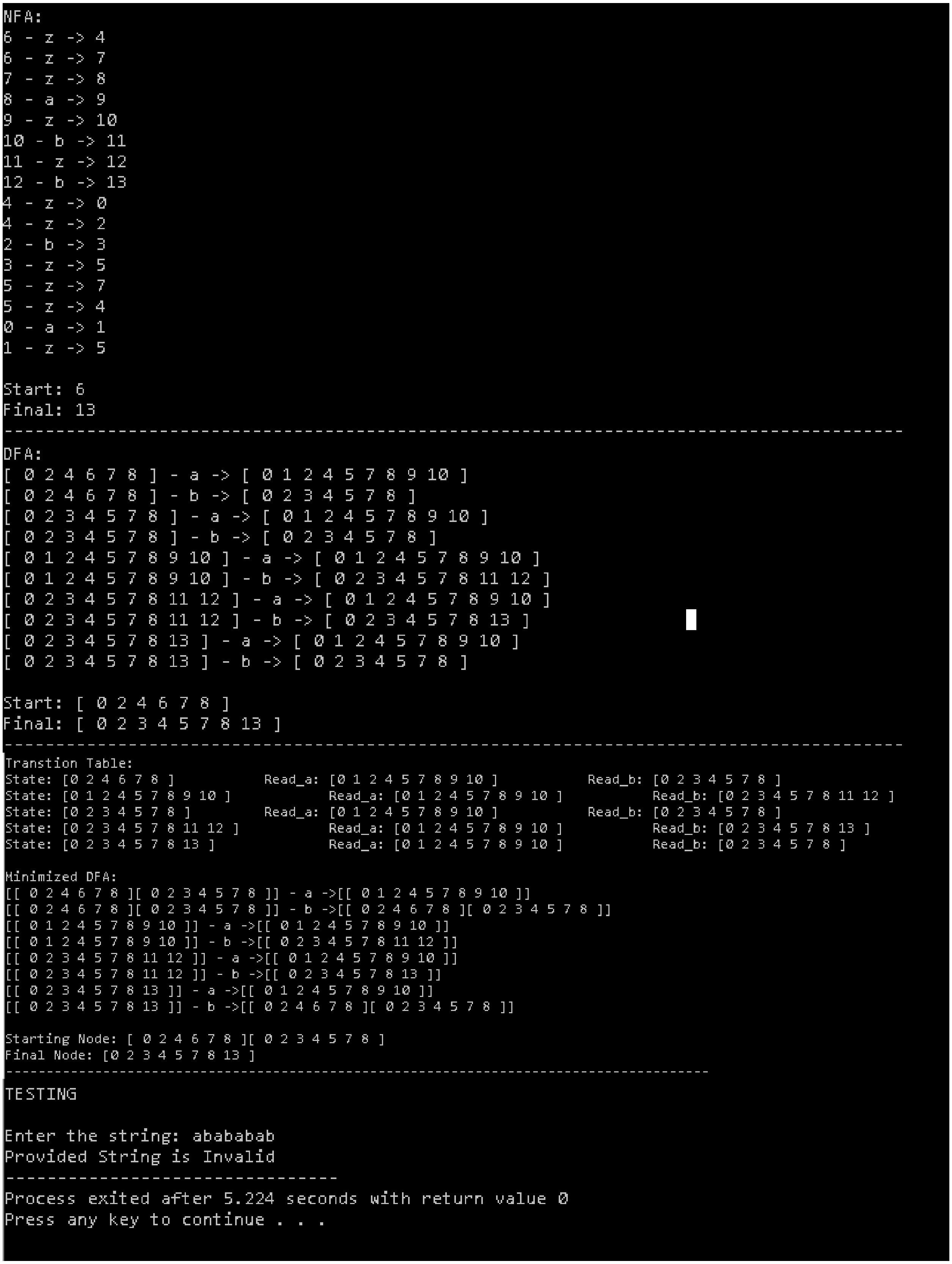
```
        cout<<"]";
    }
    cout<<endl;
}


string valid_string(vector<minimized_dfa_nodes> minimized_dfa_nodes,string s)
{
    for(int i=0;i<s.length();i++)
    {
        int found=0;
        for(int j=0;j<minimized_dfa_nodes.size();j++)
        {
            if(start_node==minimized_dfa_nodes[j].start and minimized_dfa_nodes[j].read==s[i])
            {
                found=1;
                start_node=minimized_dfa_nodes[j].end;
                break;
            }
        }
        if(found==0)
        return "Invalid";
    }
    if(start_node!=final_node)
    return "Invalid";


    return "Valid";
}
```

```
int main()
{
	string s="(a|b)*abb";
	pair<node* , node*> my_nfa=nfa(s);
	vector<nfa_nodes> all_nfa_nodes=get_nodes(my_nfa);
	cout<<"NFA: "<<endl;
	print_nfa_nodes(all_nfa_nodes);

	cout<<"--------------------------------------------------------------------------------"<<endl;
	cout<<"DFA: "<<endl;
	vector<dfa_nodes> all_dfa_nodes=nfa_to_dfa(all_nfa_nodes);
	print_dfa_nodes(all_dfa_nodes);

	cout<<"--------------------------------------------------------------------------------"<<endl;
	vector<vector<vector<int> > > minimized_equivalence=minimized_dfa(all_dfa_nodes);
	vector<minimized_dfa_nodes> minimized_dfa_nodes=get_minimized_dfa(minimized_equivalence);
	cout<<"Minimized DFA: "<<endl;
	print_minimized_dfa(minimized_dfa_nodes);
	cout<<"--------------------------------------------------------------------------------"<<endl;
	cout<<"TESTING"<<endl;
	string str;
	cout<<endl;
	cout<<"Enter the string: ";
	cin>>str;
	cout<<"Provided String is "<<valid_string(minimized_dfa_nodes,str);
	return 0;
}
```

**OUTPUT:**

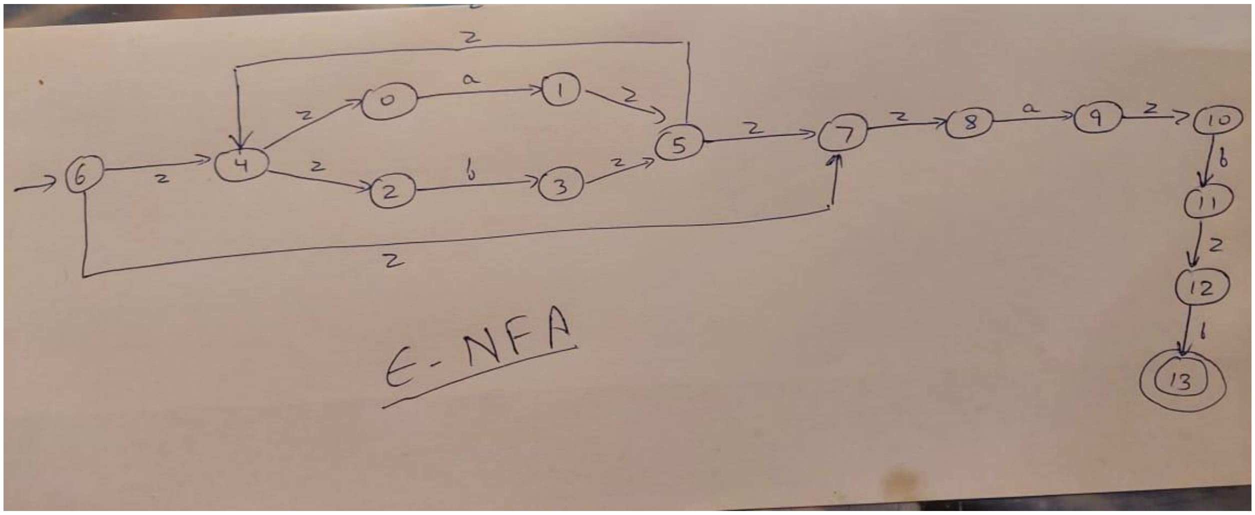


```
NFA:
6 - z -> 4
6 - z -> 7
7 - z -> 8
8 - a -> 9
9 - z -> 10
10 - b -> 11
11 - z -> 12
12 - b -> 13
4 - z -> 0
4 - z -> 2
2 - b -> 3
3 - z -> 5
5 - z -> 7
5 - z -> 4
0 - a -> 1
1 - z -> 5

Start: 6
Final: 13
------------------------------------------------------------------------------------------
DFA:
[ 0 2 4 6 7 8 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 4 6 7 8 ] - b -> [ 0 2 3 4 5 7 8 ]
[ 0 2 3 4 5 7 8 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 3 4 5 7 8 ] - b -> [ 0 2 3 4 5 7 8 ]
[ 0 1 2 4 5 7 8 9 10 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 1 2 4 5 7 8 9 10 ] - b -> [ 0 2 3 4 5 7 8 11 12 ]
[ 0 2 3 4 5 7 8 11 12 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 3 4 5 7 8 11 12 ] - b -> [ 0 2 3 4 5 7 8 13 ]
[ 0 2 3 4 5 7 8 13 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 3 4 5 7 8 13 ] - b -> [ 0 2 3 4 5 7 8 ]

Start: [ 0 2 4 6 7 8 ]
Final: [ 0 2 3 4 5 7 8 13 ]
------------------------------------------------------------------------------------------
Transtion Table:
State: [0 2 4 6 7 8 ]           Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 ]
State: [0 1 2 4 5 7 8 9 10 ]            Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 11 12 ]
State: [0 2 3 4 5 7 8 ]         Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 ]
State: [0 2 3 4 5 7 8 11 12 ]           Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 13 ]
State: [0 2 3 4 5 7 8 13 ]              Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 ]

Minimized DFA:
[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]] - b ->[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]]
[[ 0 1 2 4 5 7 8 9 10 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 1 2 4 5 7 8 9 10 ]] - b ->[[ 0 2 3 4 5 7 8 11 12 ]]
[[ 0 2 3 4 5 7 8 11 12 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 2 3 4 5 7 8 11 12 ]] - b ->[[ 0 2 3 4 5 7 8 13 ]]
[[ 0 2 3 4 5 7 8 13 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 2 3 4 5 7 8 13 ]] - b ->[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]]

Starting Node: [ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]
Final Node: [0 2 3 4 5 7 8 13 ]
------------------------------------------------------------------------------------
TESTING

Enter the string: aababab
Provided String is Valid
--------------------------------
Process exited after 195.6 seconds with return value 0
Press any key to continue . . .
```

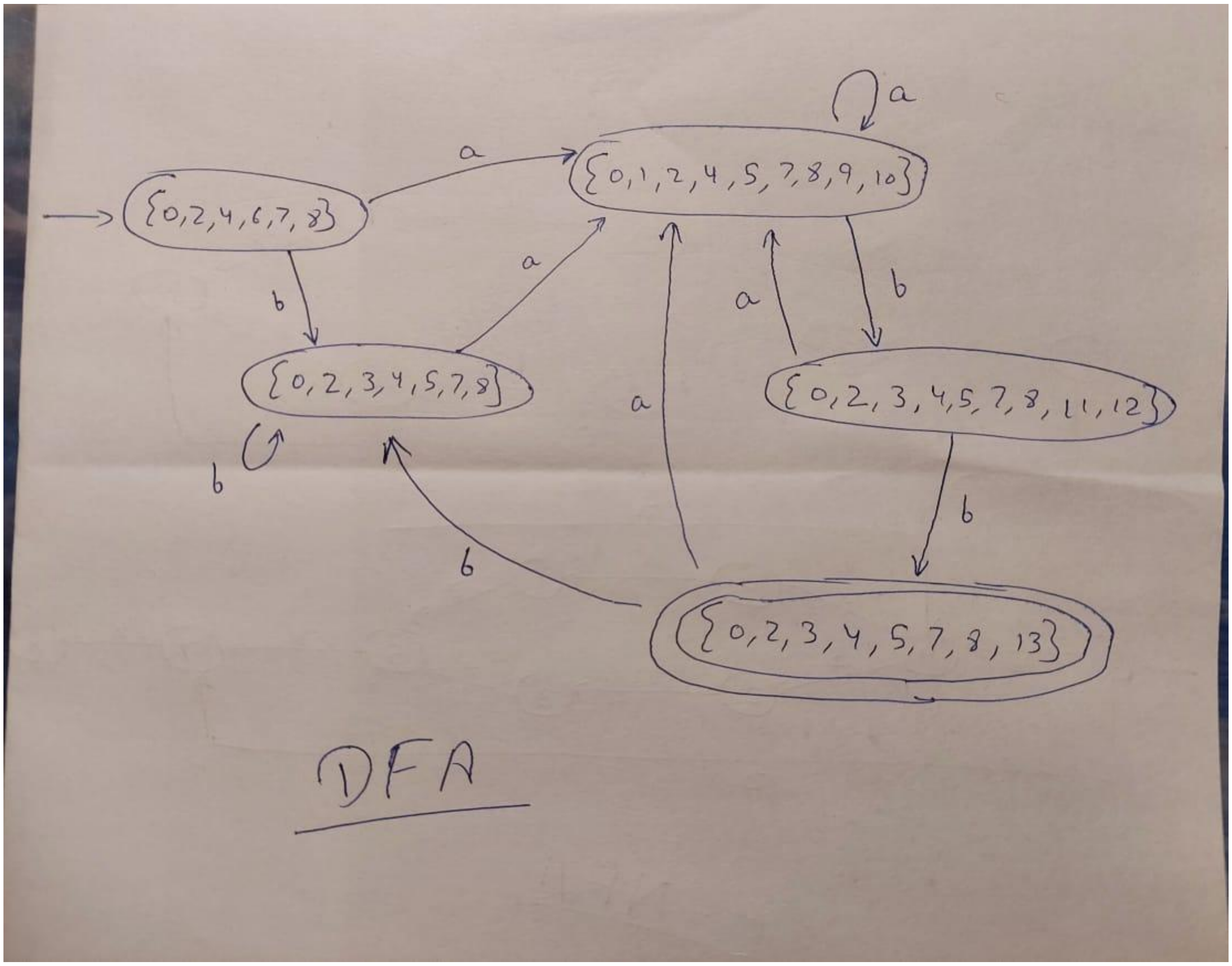


```
NFA:
6 - z -> 4
6 - z -> 7
7 - z -> 8
8 - a -> 9
9 - z -> 10
10 - b -> 11
11 - z -> 12
12 - b -> 13
4 - z -> 0
4 - z -> 2
2 - b -> 3
3 - z -> 5
5 - z -> 7
5 - z -> 4
0 - a -> 1
1 - z -> 5

Start: 6
Final: 13
----------------------------------------------------------------------------------------
DFA:
[ 0 2 4 6 7 8 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 4 6 7 8 ] - b -> [ 0 2 3 4 5 7 8 ]
[ 0 2 3 4 5 7 8 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 3 4 5 7 8 ] - b -> [ 0 2 3 4 5 7 8 ]
[ 0 1 2 4 5 7 8 9 10 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 1 2 4 5 7 8 9 10 ] - b -> [ 0 2 3 4 5 7 8 11 12 ]
[ 0 2 3 4 5 7 8 11 12 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 3 4 5 7 8 11 12 ] - b -> [ 0 2 3 4 5 7 8 13 ]
[ 0 2 3 4 5 7 8 13 ] - a -> [ 0 1 2 4 5 7 8 9 10 ]
[ 0 2 3 4 5 7 8 13 ] - b -> [ 0 2 3 4 5 7 8 ]

Start: [ 0 2 4 6 7 8 ]
Final: [ 0 2 3 4 5 7 8 13 ]
----------------------------------------------------------------------------------------
Transtion Table:
State: [0 2 4 6 7 8 ]           Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 ]
State: [0 1 2 4 5 7 8 9 10 ]            Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 11 12 ]
State: [0 2 3 4 5 7 8 ]         Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 ]
State: [0 2 3 4 5 7 8 11 12 ]           Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 13 ]
State: [0 2 3 4 5 7 8 13 ]              Read_a: [0 1 2 4 5 7 8 9 10 ]           Read_b: [0 2 3 4 5 7 8 ]

Minimized DFA:
[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]] - b ->[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]]
[[ 0 1 2 4 5 7 8 9 10 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 1 2 4 5 7 8 9 10 ]] - b ->[[ 0 2 3 4 5 7 8 11 12 ]]
[[ 0 2 3 4 5 7 8 11 12 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 2 3 4 5 7 8 11 12 ]] - b ->[[ 0 2 3 4 5 7 8 13 ]]
[[ 0 2 3 4 5 7 8 13 ]] - a ->[[ 0 1 2 4 5 7 8 9 10 ]]
[[ 0 2 3 4 5 7 8 13 ]] - b ->[[ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]]

Starting Node: [ 0 2 4 6 7 8 ][ 0 2 3 4 5 7 8 ]
Final Node: [0 2 3 4 5 7 8 13 ]
----------------------------------------------------------------------------------------
TESTING

Enter the string: abababab
Provided String is Invalid
--------------------------------
Process exited after 5.224 seconds with return value 0
Press any key to continue . . .
```

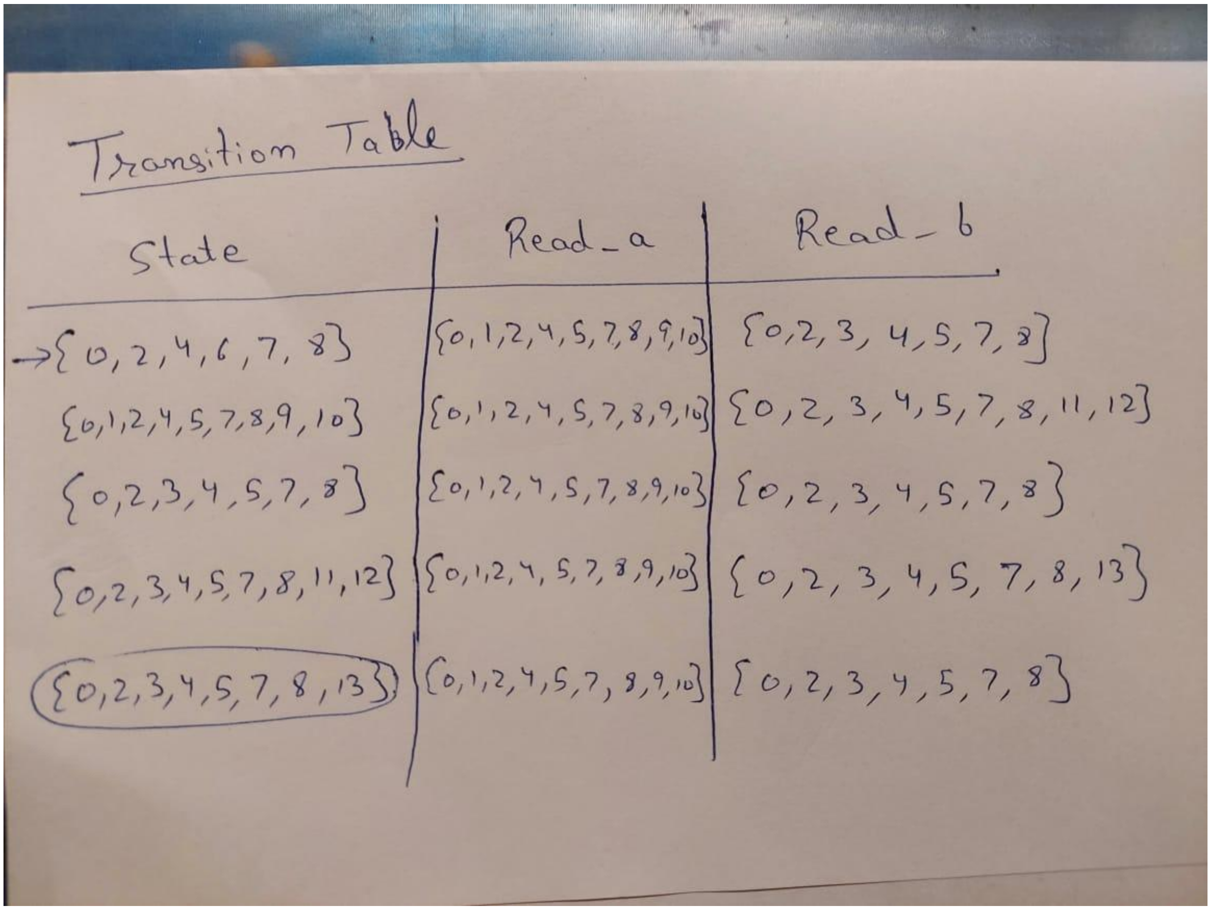
**NFA:**


z
0
a
1
z
z
z
6
z
4
z
2
b
3
z
5
z
7
z
8
a
9
z
10
b
11
z
12
b
13
z
ε-NFA

**DFA:**


{0,2,4,6,7,8}
a
{0,1,2,4,5,7,8,9,10}
a
b
a
b
a
{0,2,3,4,5,7,8}
b
a
{0,2,3,4,5,7,8,11,12}
b
b
{0,2,3,4,5,7,8,13}
DFA

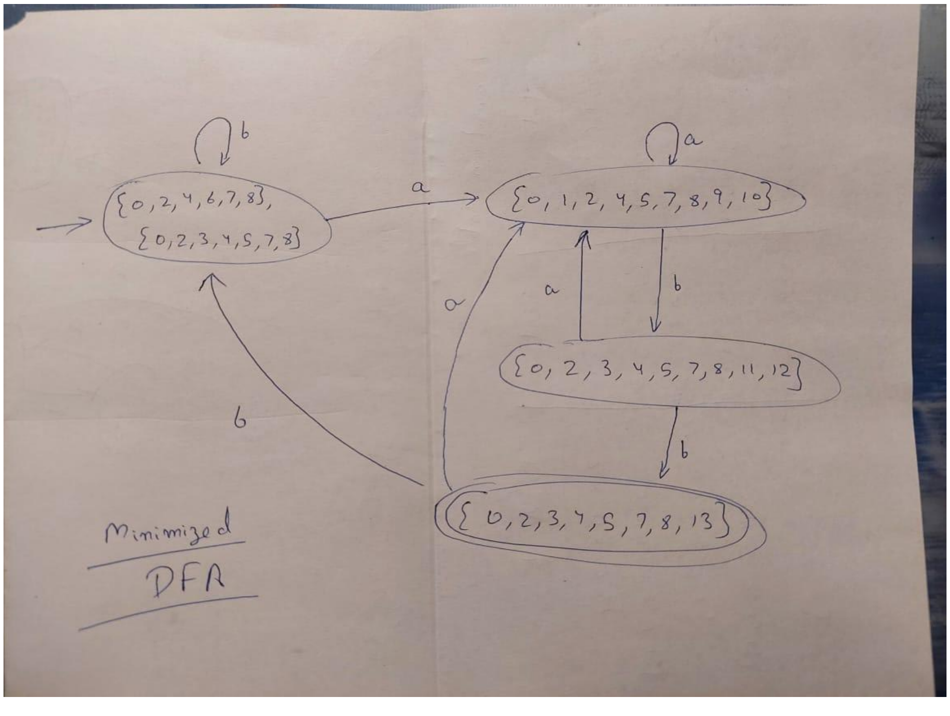
**Transition Table:**



Transition Table

| State | Read_a | Read_b |
|---|---|---|
| →{0,2,4,6,7,8} | {0,1,2,4,5,7,8,9,10} | {0,2,3,4,5,7,8} |
| {0,1,2,4,5,7,8,9,10} | {0,1,2,4,5,7,8,9,10} | {0,2,3,4,5,7,8,11,12} |
| {0,2,3,4,5,7,8} | {0,1,2,4,5,7,8,9,10} | {0,2,3,4,5,7,8} |
| {0,2,3,4,5,7,8,11,12} | {0,1,2,4,5,7,8,9,10} | {0,2,3,4,5,7,8,13} |
| {0,2,3,4,5,7,8,13} | {0,1,2,4,5,7,8,9,10} | {0,2,3,4,5,7,8} |

**Minimized DFA:**


b
{0,2,4,6,7,8},
{0,2,3,4,5,7,8}
a
a
{0,1,2,4,5,7,8,9,10}
a
b
a
{0,2,3,4,5,7,8,11,12}
b
b
{0,2,3,4,5,7,8,13}
Minimized
DFA