T.C.

SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ PR. (İÖ)

Dersin Adı: Biçimsel Diller ve Soyut Makineler

Öğrencinin Adı - Soyadı: Manar - AL SAYED ALI

Öğrenci NO - Grubu: G221210558 - 2.A

Ödevin Konusu: DFA Durum Sadeleştirme

Git-hub Linki: https://github.com/manar-alsayedali/DFA-Minimization/tree/main

1.Sorunun Tanımı

Soru: Bir DFA makinesinde durum sayısını indirgemek için bir algoritma geliştiriniz ve istediğiniz bir programlama dilinde kodlayınız. (Ulaşılamayan durumların kalıdırlması ve denk durumların birleştirilmesi).

DFA minimizasyonu, temelde iki ana adımdan oluşur:

1. Ulaşılamayan durumların kaldırılması: Bu adımda, başlangıç durumuna ulaşamayan veya bir kabul durumuna geçişi olmayan durumlar silinir.

2. Denk durumların birleştirilmesi: Aynı geçişlere sahip olan, yani eşdeğer davranan durumlar birleştirilir. Bu, dilin kabulüne etki etmeyen gereksiz durumların ortadan kaldırılması anlamına gelir.

2. Algoritmanın Çalışma Prensibi

Program, kullanıcıdan bir DFA'nın tüm bileşenlerini alır:

(Durumlar, Geçişler, Başlangıç Durumu, Final Durumları)

Bu bilgileri aldıktan sonra, minimizasyon algoritması aşağıdaki şekilde çalışır:

1. Ulaşılamayan Durumların Kaldırılması: Başlangıç durumundan başlayarak, hangi durumlara ulaşılabildiğini kontrol ederiz. Ulaşılamayan durumlar silinir.

2. Denk Durumların Birleştirilmesi: Durumlar, geçiş fonksiyonları ve kabuliyet durumlarına göre gruplanır. Eşdeğer durumda olanlar birleştirilir.

3. Minimize Edilmiş DFA'nın Çıktısı: Minimize edilmiş DFA, her birleştirilmiş grup için yeni bir durum üreterek oluşturulur. Bu durumlar birleştirilir ve geçiş fonksiyonları güncellenir.

3. Programın İşleyişi

- **Kullanıcı Girdisi**

Kullanıcıdan alınan bilgiler sırasıyla şunlardır:

- **DFA Durum Sayısı:** Toplam durum sayısı (Ör, 10 durum).
- **Sembollerin Sayısı:** Geçişlere ait semboller.
- **DFA Durumları:** Durumların isimleri (Ör, 0 1 2 3 ..).
- **Geçis Sembolleri:** DFA'nın üzerinde çalışacağı semboller seti (Ör, a ve b).
- **Başlangıç Durumu:** DFA'nın başlangıç durumu (Ör, 0).
- **Kabul Durumları:** DFA'nın kabul ettiği durumlar (Ör, 3 4 8 9).
- **Geçiş Tablosu:** DFA'nın her durumu ve sembolü için geçişlerin tanımlandığı tablo.

- **Minimizasyon İşlemi**

1. Ulaşılamayan Durumların Kaldırılması:

- Başlangıç durumundan başlayarak, hangi durumlara geçiş yapılabiliyor olduğunu kontrol ederiz.
- Ulaşılamayan durumlar, geçiş fonksiyonundan ve kabul durumları listesinden çıkarılır.

2. Denk Durumların Birleştirilmesi:

- Durumlar, geçiş sembollerine göre gruplandırılır.
- Aynı geçişlere sahip olan durumlar birleştirilir.

- **Çıktı**

Program, minimize edilmiş DFA'nın durumlarını, geçişlerini ve kabul durumlarını şu şekilde sunar:

- **Minimize Edilmiş DFA Durumları**

Durumlar, orijinal DFA'daki benzer özelliklere sahip durumların birleştirilmesiyle oluşturulur. Yeni durumlar, birleşmiş durumları temsil eder.

- **Başlangıç Durumu**

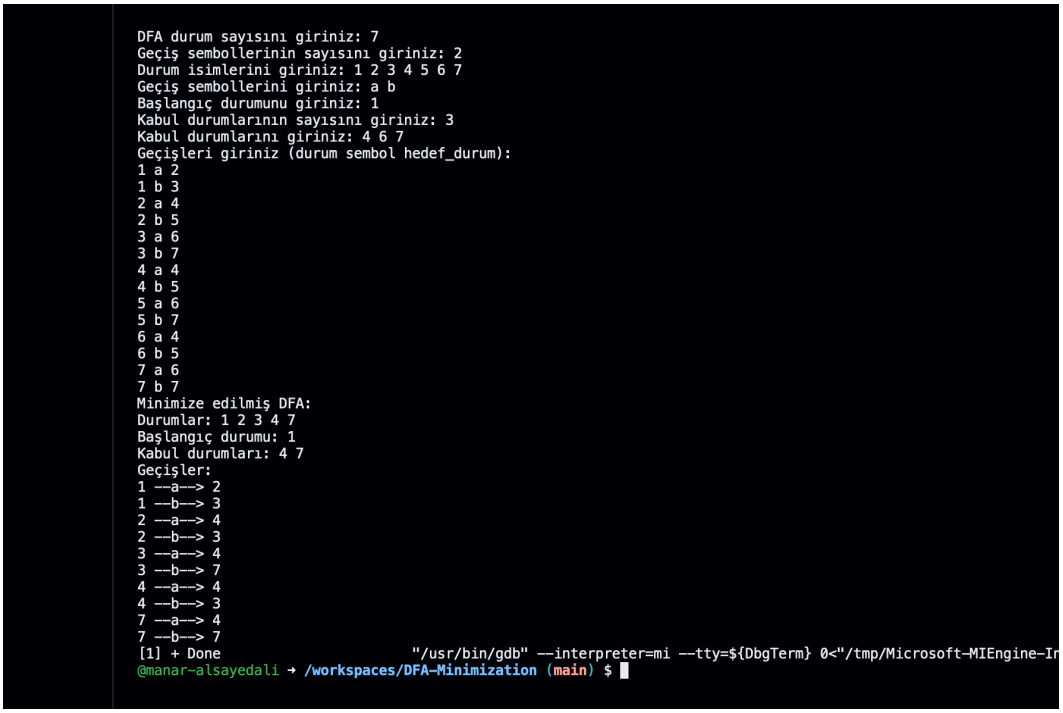
Minimize edilen DFA'nın başlangıç durumu, orijinal DFA'nın başlangıç durumlarından türetilir.

- **Kabul Durumları**

Kabul durumları, orijinal DFA'nın kabul durumlarının birleşiminden oluşur.

- **Geçişler**

Geçişler, her sembol için hangi durumdan hangi yeni duruma geçileceğini belirtir.



Örnek Çıktı

Kaynak Kod:

```
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <queue>
#include <algorithm>
using namespace std;

// DFA yapılarını temsil eder
struct DFA {
    map<string, map<char, string>> transitions;
    set<string> states;
    set<char> symbols;
    set<string> finalStates;
    string startState;
};
// Kullanıcıdan DFA alır
DFA inputDFA() {
    DFA dfa;
    int stateCount, symbolCount, finalStateCount;
    cout << "DFA durum sayısını giriniz: ";
    cin >> stateCount;
    cout << "Geçiş sembollerinin sayısını giriniz: ";
    cin >> symbolCount;
    cout << "Durum isimlerini giriniz: ";
    for (int i = 0; i < stateCount; i++) {
```

```
    string state;
    cin >> state;
    dfa.states.insert(state);
}
cout << "Geçiş sembollerini giriniz: ";
for (int i = 0; i < symbolCount; i++) {
    char symbol;
    cin >> symbol;
    dfa.symbols.insert(symbol);
}
cout << "Başlangıç durumunu giriniz: ";
cin >> dfa.startState;
cout << "Kabul durumlarının sayısını giriniz: ";
cin >> finalStateCount;
cout << "Kabul durumlarını giriniz: ";
for (int i = 0; i < finalStateCount; i++) {
    string finalState;
    cin >> finalState;
    dfa.finalStates.insert(finalState);
}
cout << "Geçişleri giriniz (durum sembol hedef_durum):" << endl;
for (int i = 0; i < stateCount * symbolCount; i++) {
    string state, target;
    char symbol;
    cin >> state >> symbol >> target;
    dfa.transitions[state][symbol] = target;
}
```

```
    return dfa;
}
// DFA'nın minimize edilmesi
DFA minimizeDFA(const DFA& dfa) {
    set<string> nonFinalStates;
    for (const auto& state : dfa.states) {
        if (dfa.finalStates.find(state) == dfa.finalStates.end()) {
            nonFinalStates.insert(state);
        }
    }
    vector<set<string>> partitions = {dfa.finalStates, nonFinalStates};
    bool stable = false;
    while (!stable) {
        stable = true;
        vector<set<string>> newPartitions;
        for (const auto& group : partitions) {
            map<vector<int>, set<string>> splitGroups;
            for (const auto& state : group) {
                vector<int> transitionClasses;
                for (const auto& symbol : dfa.symbols) {
                    string target = dfa.transitions.at(state).at(symbol);
                    int partitionIndex = -1;
                    for (int i = 0; i < partitions.size(); i++) {
                        if (partitions[i].find(target) != partitions[i].end()) {
                            partitionIndex = i;
                            break;
                        }
```

```
            }
            transitionClasses.push_back(partitionIndex);
        }
        splitGroups[transitionClasses].insert(state);
    }
    if (splitGroups.size() > 1) {
        stable = false;
    }
    for (const auto& [_, newGroup] : splitGroups) {
        newPartitions.push_back(newGroup);
    }
  }
  partitions = newPartitions;
}
// Minimize edilmiş DFA oluşturma
DFA minimizedDFA;
map<string, string> stateMapping;
for (const auto& group : partitions) {
  string newState = *group.begin();
  for (const auto& state : group) {
    stateMapping[state] = newState;
  }
  minimizedDFA.states.insert(newState);
}
for (const auto& state : dfa.states) {
  if (dfa.finalStates.find(state) != dfa.finalStates.end()) {
    minimizedDFA.finalStates.insert(stateMapping[state]);
```

```
        }
    }
    minimizedDFA.startState = stateMapping[dfa.startState];
    minimizedDFA.symbols = dfa.symbols;
    for (const auto& [state, transitions] : dfa.transitions) {
        for (const auto& [symbol, target] : transitions) {
            string newSource = stateMapping[state];
            string newTarget = stateMapping[target];
            minimizedDFA.transitions[newSource][symbol] = newTarget;
        }
    }
    return minimizedDFA;
}
// DFA'yı yazdırma
void printDFA(const DFA& dfa) {
    cout << "Minimize edilmiş DFA:" << endl;
    cout << "Durumlar: ";
    for (const auto& state : dfa.states) {
        cout << state << " ";
    }
    cout << endl;
    cout << "Başlangıç durumu: " << dfa.startState << endl;
    cout << "Kabul durumları: ";
    for (const auto& state : dfa.finalStates) {
        cout << state << " ";
    }
    cout << endl;
```

```
    cout << "Geçişler:" << endl;
    for (const auto& [state, transitions] : dfa.transitions) {
        for (const auto& [symbol, target] : transitions) {
            cout << state << " --" << symbol << "--> " << target << endl;
        }
    }
}
int main() {
    DFA dfa = inputDFA();
    DFA minimizedDFA = minimizeDFA(dfa);
    printDFA(minimizedDFA);
    return 0;
}
```