Київський національний університет
імені Т.Шевченка

Звіт

до лабораторної роботи №2
з дисципліни:
«Системне Програмування»

Студента третього курсу
Групи МІ-32
Факультету комп'ютерних наук
та кібернетики
Федорича Андрія

Київ-2023

Мета

Метою лабораторної роботи є розробка та реалізація представлення скінченного автомата в пам'яті ЕОМ та виконати наступну задачу (варіант – один з 9, обрати згідно порядкового номеру у списку групи (циклічно); 10-19 варіант (дещо ускладнений) обирається за тим же принципом для додаткових балів).

Теоретичні відомості

Скінчений автомат без виходів: $\boldsymbol{A} = <\mathbf{A}, \mathbf{S}, \mathbf{s_0}, \mathbf{F}, \mathbf{f}>$, де

$\mathbf{A}$ = {a, b, c, …} – вхідний алфавіт,

$\mathbf{S}$ = {0, 1, 2, …} – множина станів,

$\mathbf{s_0} \in \mathbf{S}$ – початковий стан,

$\mathbf{F} \subseteq \mathbf{S}$ – множина фінальних (заключних) станів,

$\mathbf{f}$: $\mathbf{S} \times \mathbf{A} \to \mathbf{S}$ – функція переходів (автомат, знаходячись у певному стані та читаючи черговий символ зі вхідного слова переходить в інший стан згідно цієї функції доти, поки не закінчилось слово та поки існують відповідні переходи).

Функція переходів може бути не всюди визначеною та може бути багатозначною (у випадку недетермінованого автомату).

Слово $\mathbf{e}$ – слово нульової довжини (зарезервоване, $\boldsymbol{\varepsilon}$). Таким чином, для будь-якого символу або слова $\boldsymbol{w}$ справедливо: $\boldsymbol{we}=\boldsymbol{ew}=\boldsymbol{w}$. Якщо автомат допускає $\mathbf{e}$-переходи, то використовується функція переходів вигляду $\mathbf{S} \times (\mathbf{A} \cup \{\mathbf{e}\}) \to \mathbf{S}$ замість звичайної для задання такого автомату.

Автомат $\boldsymbol{A}$ сприймає (допускає, розпізнає) слово $\boldsymbol{w}$, якщо, читаючи по одному символу з цього слова (за кожен такт роботи – зліва направо) і виконуючи переходи відповідно до функції переходів $\mathbf{f}$, починаючи з стану $\mathbf{s_0}$, він потрапляє через $|\boldsymbol{w}|$ кроків у стан $f \in \mathbf{F}$.

$|\mathbf{w}|$ = довжина слова $\mathbf{w}$,

||Set|| = потужність множини Set, для скінчених – кількість її (унікальних) елементів.

—

Автомат ***A*** на вході програми (та на виході, де потрібно) подається у вигляді текстового файлу наступної структури:

||**A**||

||**S**||

$\mathbf{s_0}$

||**F**|| $fs_1 \in \mathbf{F}$ … $fs_{||\mathbf{F}||} \in \mathbf{F}$ // перелічені через проміжок кількість та всі стани з множини **F**

s a *s'* // всі такі трійки, що (*s*, a, *s'*) ∈ **f**, через проміжок, по одній на рядок – до кінця файлу.

Основні принципи виконання роботи

Варіант: 11**
Реалізувати алгоритм мінімізації детермінованого скінченного автомата.

Мова виконання: C

Джерело:
Код програми, тестові та результуючі файли можна знайти за посиланням:*https://github.com/StopFuture/KNU_Bachelor_Assignments/tree/main/SystemProgramming/Lab2*

Реалізація:

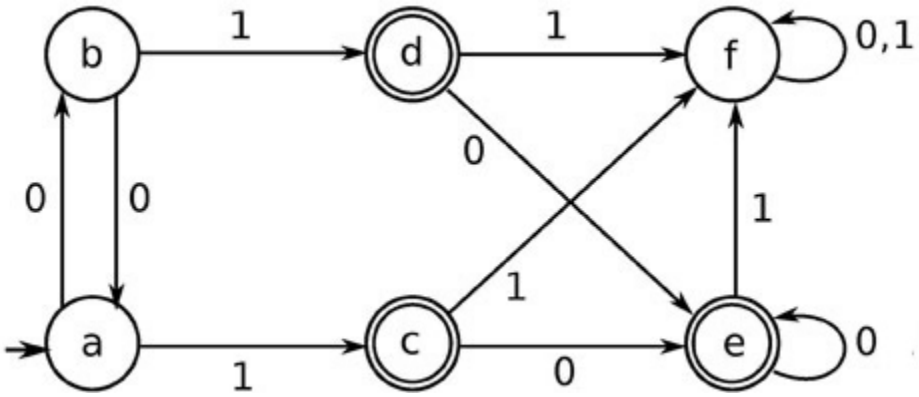
- Як основу роботи алгоритму вибрано: **Minimization of DFA Using Equivalence Theorem.**
- Всі приклади, що будуть наведені в подальшому взято із:
  - https://en.wikipedia.org/wiki/DFA_minimization
  - https://www.gatevidyalay.com/minimization-of-dfa-minimize-dfa-example/

- Створено два **.c** файли та один **.h** файл-заголовок.
- В файлах **dfa.c та dfa.h** реалізовано структуру даних **DFA (Deterministic finite automaton),** що виконує поставлену задачу.
- Автомат має декілька методів :
    - **initialise_dfa** – для ініціалізації структури даних під час виконання зчитуються дані із вхідного файлу**.**
    - **initialise_transition_map** – створює 2D масив в якому зберігаються всі можливі переходи між станами в атоматі.
    - **determine_reachable_states** – знаходить список станів, які є досяжними.
    - **dfs –** звичайний рекурсивний алгоритм пошуку в ширину.
    - **partition_dfa –** основна функція, що виконує мінімізацію нашого автомата.
    - **print_dfa_details –** запис результатів в консоль та файл.
    -
- В основній частині програми наступна логіка:
    - Ініціалізуємо файл **input_<number>** з якого зчитуватимемо слова, при чому перевіряємо коректність його відкриття (поки користувач не введе правильний файл).
    - Ініціалізуємо та мінімізуємо даний автомат.
    - Створюємо відповідний файл **output_<number>** в який записуємо результат, якщо файл вже існував перезаписуємо його та дублюємо вивід в консоль**.**

Демонстрація роботи

Приклад 1

Вхідний автомат:



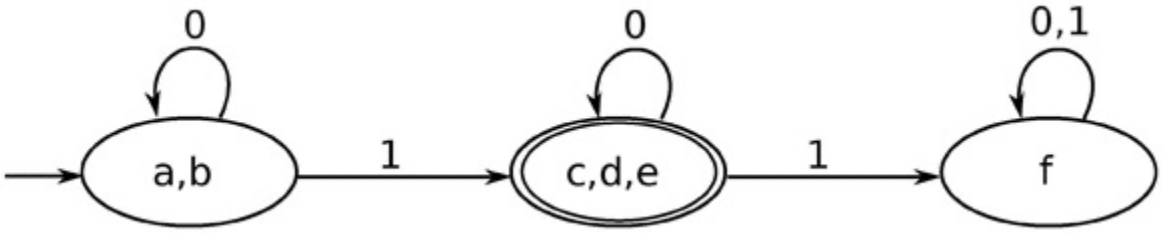
В нашому представлені:

```
2
6
0
3 2 3 4
0 a 1
0 b 2
1 a 0
1 b 3
2 b 5
2 a 4
3 a 4
3 b 5
4 a 4
4 b 5
5 a 5
5 b 5
```

Тобто алфавіт переходить із 0,1 → a, b

Та стани із a - e → 0 - 5

Результат:

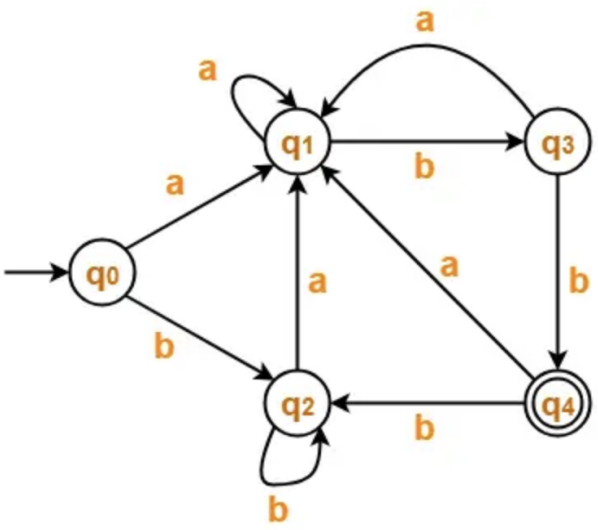


Еквівалетно результату програми:

```
/Users/stopfuture/CLionProjects/SystemProgramming/Lab2/
Enter a file name: input_1.txt
DFA initialized successfully.
Minimised DFA:
Alphabet Size: 2
Number of States: 3
Start State: 2
Number of Final States: 1
Final States:0
0 a 0
0 b 1
1 a 1
1 b 1
2 a 2
2 b 0
```

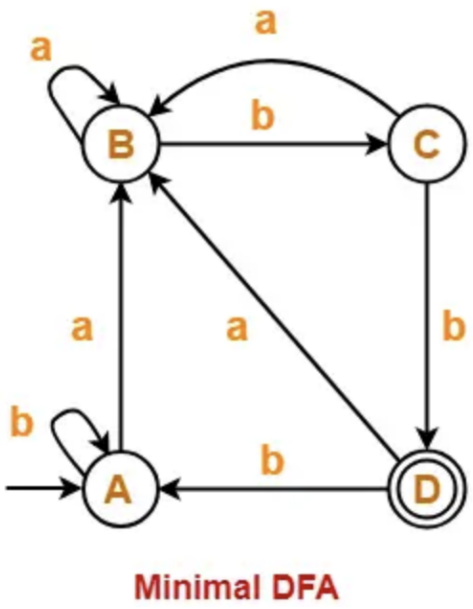
Приклад 2

```
2
5
0
1 4
0 a 1
0 b 2
1 a 1
1 b 3
2 a 1
2 b 2
3 a 1
3 b 4
4 a 1
4 b 2
```



Результат:

```
/Users/stopfuture/CLionProjects/SystemProgramming/Lab2/
Enter a file name: input_2.txt
DFA initialized successfully.
Minimised DFA:
Alphabet Size: 2
Number of States: 4
Start State: 2
Number of Final States: 1
Final States:0
0 a 3
0 b 2
1 a 3
1 b 0
2 a 3
2 b 2
3 a 3
3 b 1
```



Minimal DFA

Висновок

У даній лабораторній роботі була успішно виконана основна умова завдання, а саме - програма незалежно від вхідного автомату, виконує його мінімізацію та зберігає весь результат в структурі даних DFA, інформацію з якої потім послідовно записує в результуючий файл.