Лабораторна робота 2
З дисципліни “Системне програмування”
Студента групи МІ-31
Бідзілі Святослав Олексійовича

Постановка задачі:

Скінчений автомат без виходів: $\boldsymbol{A} = <\mathbf{A}, \mathbf{S}, \mathbf{s}_0, \mathbf{F}, \mathbf{f}>$, де

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

$\mathbf{S} = \{0, 1, 2, \ldots\}$ – 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 **e** – слово нульової довжини (зарезервоване, **ε**). Таким чином, для будь-якого символу або слова $\boldsymbol{w}$ справедливо: $\boldsymbol{w}\mathbf{e}=\mathbf{e}\boldsymbol{w}=\boldsymbol{w}$. Якщо автомат допускає **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 **f**, починаючи з стану $\mathbf{s}_0$, він потрапляє через $|\boldsymbol{w}|$ кроків у стан $f \in \mathbf{F}$.

$|\mathbf{w}|$ = довжина слова **w**,

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

—

Автомат $\boldsymbol{A}$ на вході програми (та на виході, де потрібно) подається у вигляді текстового файлу наступної структури:

$||\mathbf{A}||$

$||\mathbf{S}||$

$\mathbf{s}_0$

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

s a s' // всі такі трійки, що $(s, a, s') \in \mathbf{f}$, через проміжок, по одній на рядок – до кінця файлу.

—

Розробити та реалізувати представлення скінченого 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).

1. Встановити, які літери вхідного алфавіту не сприймає скінчений детермінований автомат.

Виконання

Весь код для лабораторної роботи розміщений за посиланням

https://github.com/anakib1/KNU_SP_2023_lab2/

Розберемо структуру коду на функціями. Наступний блок коду містить ввід та вивід даних, який вводить числа/рядки до пробілу/нового рядка. Аналогічно вводу через Scanner в Java.

```
bool isDigitChar(char c) {
    return ('0' <= c && c <= '9');
}
bool isSpace(char c) {
    return (c == ' ' || c == '\n' || c == '\t');
}

int getInt() {
    int ret = 0; int sgn = 1;
    char c = fgetc(in);
    while (!isDigitChar(c)) {
        c = fgetc(in);
        if (c == EOF) return NaN;
    }
    do {
        ret = ret * 10 + (c - '0');
        c = fgetc(in);
    } while (isDigitChar(c));

    return ret;
}
char* getStr() {
    char c = fgetc(in);
    while(isSpace(c)) c = fgetc(in);
    char* buffer = malloc(1000); // make constant
    int i = 0;
    do {
        buffer[i++] = c;
        c = fgetc(in);
    } while (! isSpace(c));
    return buffer;
}
void putChar(char c) {
    fputc(c, out);
}
```

Наступна частина коду відповідає за створення та зберігання автомату. Через особливості та відносну давність мови С, буде зручно зберігати автомат як масив ребер, а також для кожної вершини зберігати перше ребро, яке з неї виходить. Кожне ребро також буде мати індекс наступного ребра для її вершини. Так ми зможемо в двох одновимірних масивах зберігати весь автомат і швидко та легко ітеруватися по ньому. Додатково ми зберігаємо масиви isFinal (для термінальних вершин) та поачтковий стан. Функція edgeConstruct по факту замінює конструктор з більш сучасних мов. Звернемо увагу, що ця функция додає ребро в прямий автомат, та обернене в обернений. Це важливо для корректної роботи алгоритма.

```
struct Edge
{
    int stateFromId, stateToId, symbolId;
    int nxtEdgeIndex;
} automaton[mxn], automaton2[mxn]; int automatonSize = 1;
int s0;
int g[mxn], g2[mxn];
int isFinal[mxn];

struct Edge edgeConstruct(int from, int to, int how) {
    struct Edge ret;
    ret.stateFromId = from;
    ret.stateToId = to;
    ret.symbolId = how;
    return ret;
}
```

В наступній частині коду виконується просто ввід вивід інформації, та заповнення автомату. Завдяки своєму вводжу виводу, код виглядає більш-менш сучасно. Також ми можемо так легко перевірити наявність останнього ребра. Ми зберігаємо переходи не як символи, а як номер символу в алфавіті.

```
int main() {
    in = fopen("input.txt", "r");
    out = fopen("output.txt", "w");
    if (in == NULL) {
        printf("Error, no such file: input.txt");
        return 0;
    }

    int alphabetSize = getInt();
    int stateSize = getInt();
    if (alphabetSize > 26) {
        printf("Error, too large alphabet.");
        return 0;
    }
    if (stateSize > mxn) {
        printf("Error, too large state size.");
        return 0;
    }
    s0 = getInt();
    int finalStatesSize = getInt();
    for (int i = 0; i < finalStatesSize; i ++ ) {
        int stateId = getInt();
        isFinal[stateId] = 1;
    }
    while (true) {
        int s1 = getInt();
        if (s1 == NaN) {
            break;
        }
        char a = getStr()[0];
        int s2 = getInt();

        addEdge(s1, (int)(a - 'a'), s2);
    }
```

Для виконання основної частини завдання – ми будемо користуватися наступним алгоритмом.

Для початку, знайдемо всі вершини, які досяжні з початкового стану.

Також знайдемо всі вершини, які досяжні в оберненому автоматі з будь-якої з термінальних вершин.

Нехай множина перших вершин – S1, других – S2. Тоді, всі символи, які можуть бути досяжні в словах з автомата мають мати початок і кінець в перетині множин S1, S2.

```
int dfs(int v) {
    used1[v] = 1;
    int i = g[v];
    while (i != 0) {

        int to = automaton[i].stateToId;
        if (!used1[to]) dfs(to);
        i = automaton[i].nxtEdgeIndex;
    }
}
```

Пошук множини S1 виконує функція dfs1 – вона записує в used1 маску вершин, які досяжні з v

```
int dfs2(int v) {
    used2[v] = 1;
    int i = g2[v];
    while (i != 0) {

        int to = automaton2[i].stateToId;
        if (!used2[to]) dfs2(to);
        i = automaton2[i].nxtEdgeIndex;
    }
}
```

Аналогічно функція df2 і used2.

Викличемо функції з стартової, та усіх терміналів відповідно.

```
dfs(s0);
for (int i = 1; i <= stateSize; i ++ ) {
    if (!used2[i] && isFinal[i]) dfs2(i);
}
```

Пройдемося по всіх ребрах і знайдемо, допустимі символи. Виведемо всі НЕдопустимі.

```
for (int i = 0; i < automatonSize; i ++ ) {
    int from = automaton[i].stateFromId;
    int to = automaton[i].stateToId;
    if (used1[from] && used1[from] && used2[to] && used2[to]) {
        canUseLetter[automaton[i].symbolId] = true;
    }
}

for (int i = 0; i < alphabetSize; i ++) {
    if (!canUseLetter[i]) fprintf(out, "%c\n", ('a' + i));
}
```

Приклади

```
20
5
1
3 3 4 5
1 a 2
1 b 1
1 c 3
1 d 4
1 e 2
1 f 2
1 g 5
```

```
1  a
2  e
3  f
4  h
5  i
6  j
7  k
8  l
9  m
10 n
11 o
12 p
13 q
14 r
15 s
16 t
```

В цьому прикладі наявні шляхи до **непродуктивних** вершин. Літера a веде лише в непродуктивну 2 , аналогічно літера e. Інших літер взагалі немає в ребрах.

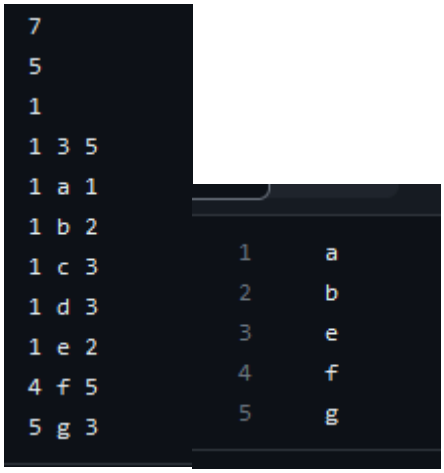
Code Blame 9 lines (9

```
1  5
2  3
3  1
4  1 3
5  1 a 1
6  1 b 2
7  1 c 3
8  1 d 3
9  1 e 2
```

Code Blame 2

```
1  b
2  e
```

Літери б,е ведуть в непродуктивний нетермінал 2.



В цьому прикладі маємо по перше **недосяжний термінал** 5. Одже, програма коректно знаходить, що букви f, g недосяжні через відсутність шляху від початкової вершини до станів 4,5. Досяжні лише літери c, d оскільки ведуть перехід в термінал 3.

Висновок

В лабораторній роботі ми змогли створити представлення детермінованого автомату в пам’яті ЕОМ та виконати просте тестування на приймання слова автоматом.