ПОСТРОЕНИЕ МИНИМАЛЬНЫХ КОНЕЧНЫХ АВТОМАТОВ ПО РЕГУЛЯРНЫМ ВЫРАЖЕНИЯМ

Регулярное выражение – один из точных способов написания регулярной грамматики.

Правила написания регулярных выражений для типовых конструкций таковы: фигурные скобки обозначают итерацию, то, что в них заключено, может повторяться от нуля до бесчисленного числа раз. Круглые или квадратные скобки – используют для объединения альтернатив, которые разделяются символом дизъюнкции. Знак конъюнкции иногда опускают.

Пусть имеется алфавит, составленный из символов (букв, литер) $V_T = \{x_1, x_2, \ldots, x_n\}$.

1. Множество всевозможных цепочек, составленных из букв $x_i \in V_T$:

$$L = \{x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n\}.$$

2. Множество цепочек, составленных из литер $x_i \in V_T$ и оканчивающихся литерой x_1:

$$L = \{x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n\} \wedge x_1.$$

3. Множество цепочек, составленных из литер $x_i \in V_T$ начинающихся цепочкой l_1 и оканчивающихся l_2**.**

$$L = l_1 \wedge \{x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n\} \wedge l_2.$$

4. Множество однолитерных цепочек (однобуквенных слов) совпадает с алфавитом V_T:

$$L = x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n.$$

5. Множество двулитерных цепочек (двухбуквенных слов):

$$L = (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n) \wedge (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n).$$

6. Множество m – буквенных слов

$$L = (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n) \underbrace{\wedge \ldots \wedge}_{m-2} (x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n)$$

Алгебру регулярных выражений поясним на примерах.

Пусть $L_1 = (x_1, x_1x_2)$, $L_2 = (x_3)$, $L_3 = (x_2, x_1)$. Тогда операции выполняются следующим образом.

Дизъюнкция:

$L = L_1 \mid L_2 \mid L_3 = L_1 \vee L_2 \vee L_3 = (x_1, x_1x_2, x_3, x_2)$.

Конъюнкция:

$L = L_1L_2 = L_1 \wedge L_2 = (x_1x_3, x_1x_2x_3)$.

Рекурсия:

$L = \{x_1x_2\} = (\varepsilon, x_1x_2, x_1x_2x_1x_2, x_1x_2x_1x_2 \ldots x_1x_2)$.

Регулярное выражение однозначно определяет конечный автомат.

Рассмотрим фрагменты регулярных выражений, соответствующие элементарным операциям в совокупности с продукциями автоматной грамматики, ими определяемыми.

Дизъюнкция с конъюнкцией $L = xy \vee x \vee y \vee z$

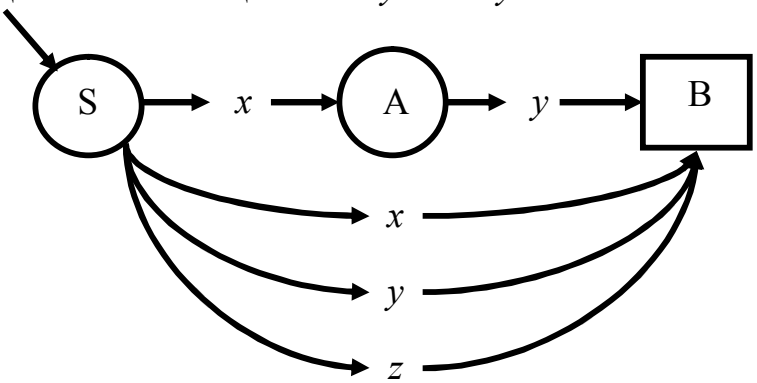


Рисунок 1 – Граф КА, принимающего цепочки *xy*, *x*, *y*, и *z*

Конъюнкция с рекурсией $L = x\ \{\ yx\ \}$

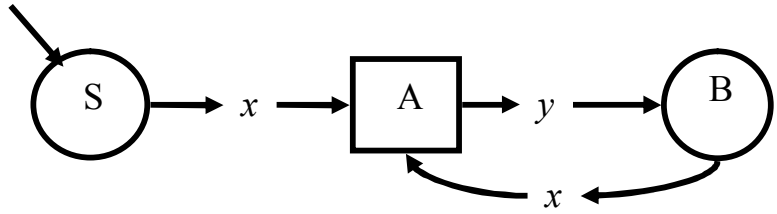


Рисунок 2 – Граф КА, принимающего цепочки *x*, *yx*, *yxyx*, *yx…yx*

Алгоритм построения ДКА по регулярному выражению

Алгоритм разработан профессором кафедры кибернетики и вычислительной техники Е. А. Бутаковым для построения и минимизации КА по регулярному выражению и основывается на системе индексации мест регулярного выражения.

Место регулярного выражения – это промежуток между двумя его компонентами (двумя буквами, буквой и знаком, скобкой и буквой, буквой и скобкой), а также начало и конец выражения.

Различают следующие типы мест

- Начало выражения – ***начальное*** место.
- Конец выражения – ***конечное*** место.
- ***Основное*** место – место, слева от которого находится литера, а также начальное место.
- ***Предосновное*** место – место, справа от которого находится литера.

По существу, конечный автомат переходит из предосновного места в основное место по символу (литере) находящемуся между этими местами.

Алгоритм состоит из этапов

1. Разметка мест.

2. Минимизация числа внутренних состояний автомата по разметке.

3. Построение таблицы переходов.

4. Построение ДКА из НДКА, если это необходимо.

5. Минимизация числа состояний путём объединения повторяющихся столбцов при условии равенства выходных сигналов. Конечные автоматы бывают двух видов: Мура и Мили. Они названы по именам их разработчиков. Для автомата Мили сигнал на выходе КА зависит от предыдущего входного сигнала, в отличие от автомата Мура, где такая зависимость отсутствует.

Правила разметки мест.

1. Первоначально осуществляется сквозная нумерация всех основных мест (короткие вертикальные линии на чертеже).

2. Выставляются индексы всех предосновных мест, не являющихся основными (одному месту может соответствовать несколько индексов). Правила

а) Индекс перед любыми скобками распространяется на все начальные места дизъюнктивных членов, записанных в этих скобках.

б) Индекс конечного места любого дизъюнктивного члена, заключённого в любые скобки, распространяется на место, непосердственно следующее за этими скобками.

в) Индекс места перед итерационными скобками распространяется на место, непосредственно следующее за этими скобками.

г) Индекс места за итерационными скобками распространяется в начальные места всех дизъюнктивных членов в этих скобках.

д) Индекс конечного места любого дизъюнктивного члена, заключённого в итерационные скобки, распространяется на начальные места всех дизъюнктивных членов, заключённых в эти скобки.

е) Индексы места, справа и слева от которого стоят буквы никуда не распространяется.

ж) Индекс конечного места распространяется на те же места, на которые и индекс начального места.

Совокупность индексов места ***является множеством***, следовательно, ***не содержит повторяющихся*** элементов.

Правило минимизации.

Если ***несколько предосновных*** мест ***отмечено одинаковой совокупностью индексов*** и ***справа*** от этих мест расположены ***одинаковые литеры***, то ***основные места***, расположенные ***справа*** от этих литер, ***можно отметить одинаковыми индексами***.

Пример 1. Определить минимальный ДКА, который принимает строки соответствующие регулярному выражению

$$S = \{b \vee ab \vee aab\}\ aaa\ \{a\}\ b$$

Пусть работа конечного автомата сопровождается следующими выходными сигналами:

x – рабочее состояние автомата;

y – встретилось сочетание “aaa”;

z – конец слова.

I-я итерация.

Разметка мест. Расстановка индексов мест проиллюстрирована на рисунке 3, на котором арабские цифры со скобкой отмечают этапы разметки, а буквы кириллицы со скобкой – правила, применяемые при выставлении того или иного индекса.

1) Разметка основных мест.

2) Основные места, которые совпадают с предосновными, поэтому индексы предосновных мест совпадают с индексами основных мест.

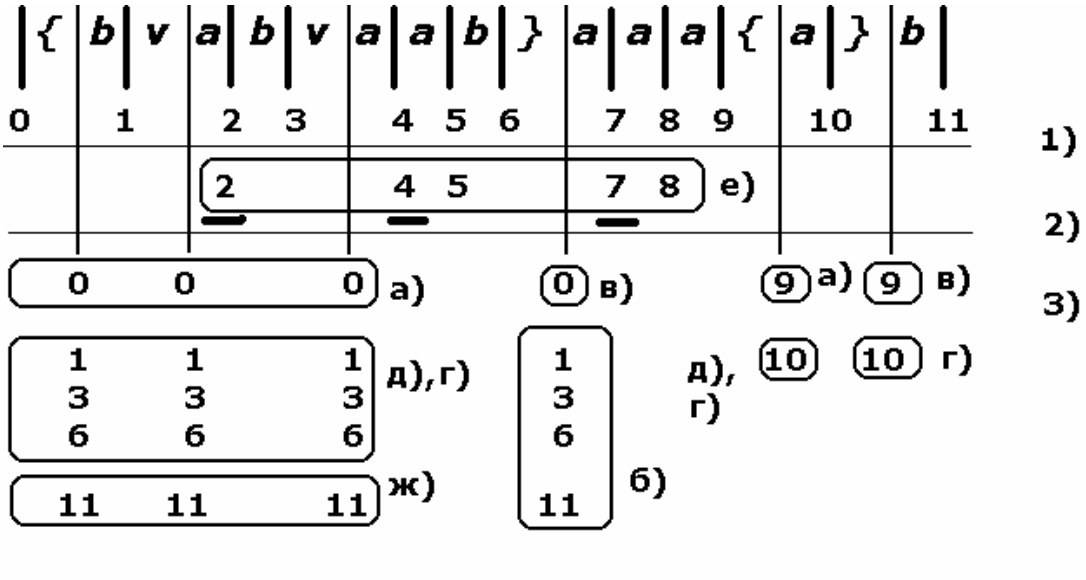


Рисунок 3 – Иллюстрация правил разметки мест

3) Выставление индексов предосновных мест.

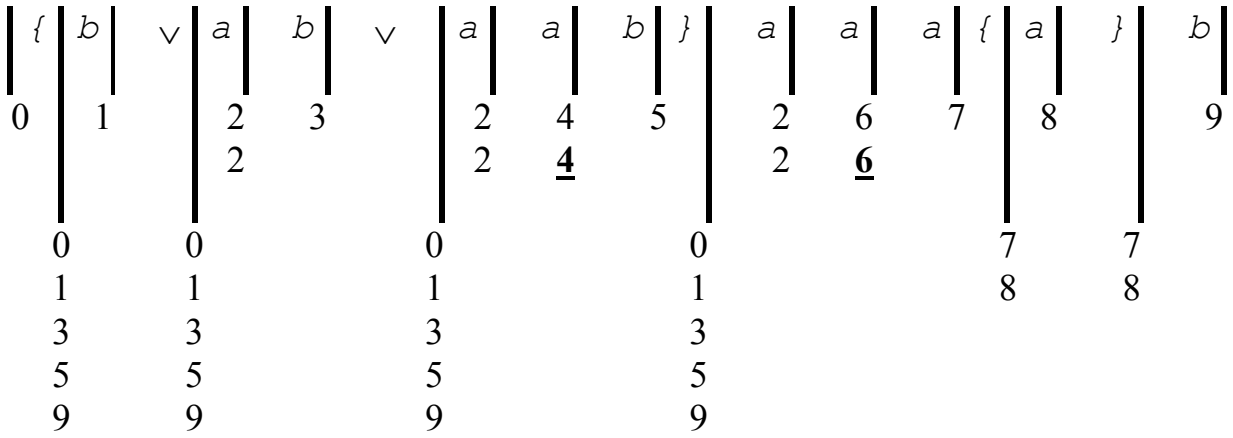
Позициями а) ... ж) обозначены правила, которыми руководствовались при разметке соответствующих мест.

Минимизация.

Под правило минимизации подпадают позиции выражения 2, 4 и 7

II-я итерация.

Разметка осуществляется с учётом эквивалентных состояний

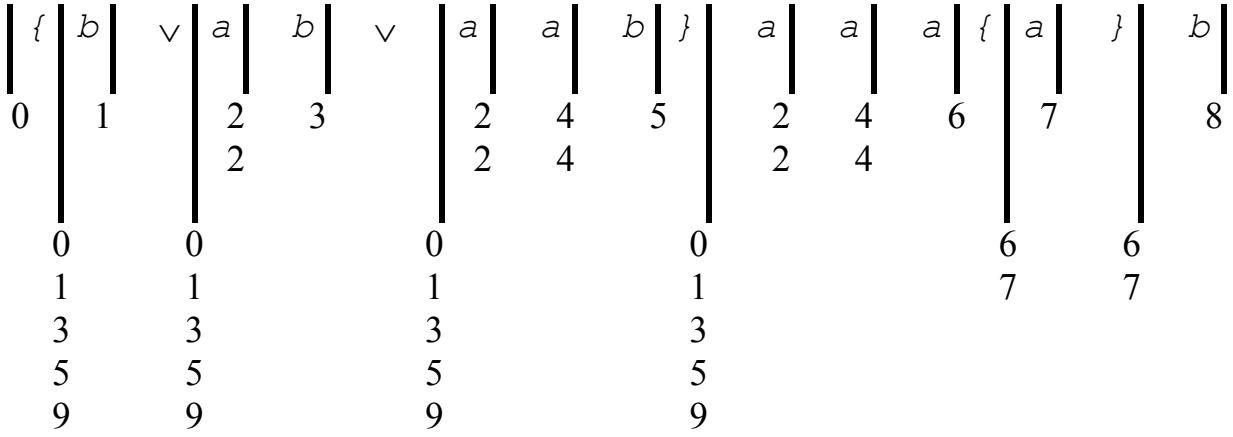


Минимизация.

Судя по разметке, эквивалентны позиции 4 и 6.

III-я итерация.

Перестоим систему разметки с учётом сокращений. Получим:



Видно, что дальнейшую минимизацию по регулярному выражению провести не удаётся.

Построим таблицу переходов КА.

Состояния автомата эквивалентны индексам основных мест регулярного выражения, а между местами располагается символ входного алфавита КА, по которому осуществляется переход.

Выходной сигнал	*x*	*x*	*x*	*x*	*x*	*x*	*y*	*x*	*z*
Состояния	0	1	2	3	4	5	6	7	8
a	2	2	4	2	6	2	7	7	2
b	1	1	3	1	5	1	8	8	1
	*	*		*		*			

Из таблицы видно (*), что состояния (0, 1, 3, 5) неразличимы ни по входному сигналу, ни по выходному. Состояние 8 является конечным

согласно пункту ж) правил разметки предосновных мест, и соответствующие образы функции переходов определены так, чтобы из заключительного состояния КА мы автоматически могли перейти на обработку очередной цепочки литер.

При программной реализации КА иногда бывает целесообразным после завершения анализа входной строки принудительно возвращать его в начальное состояние, не прибегая к функции перехода.

Обозначим: A = {0, 1, 3, 5}, B = 2, C = 4, D = 6, E = 7, F = 8.

Перестроим предыдущую таблицу переходов с учётом обозначений. Получим такую таблицу переходов:

Выходной сигнал	*x*	*X*	*x*	*y*	*x*	*z*
Состояния	A	B	C	D	E	F
a	B	C	D	E	E	B
b	A	A	A	F	F	A

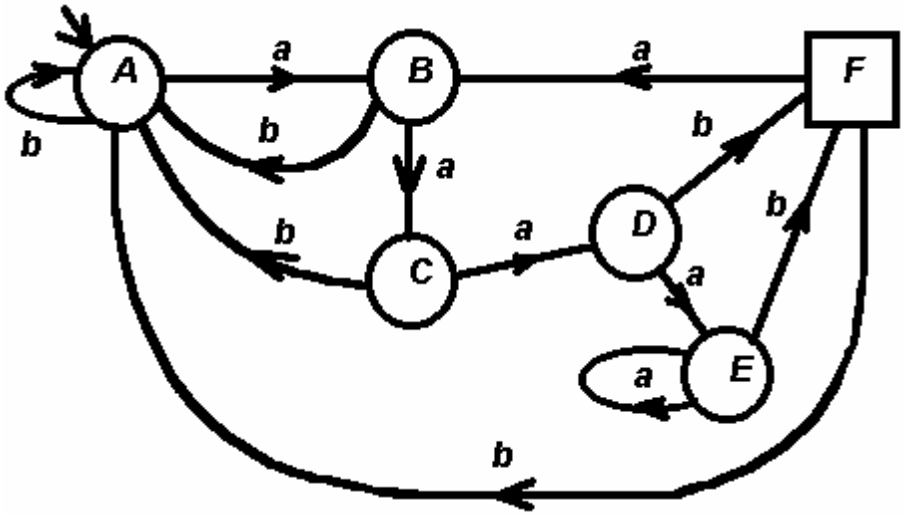


Рисунок 4 – Граф МКА, описываемый выражением *{b ∨ ab ∨ aab} aaa {a} b*

Пример 2

Примеры выполнения заданий.

Построить минимальный КА, который будет принимать любое английское слово, начинающееся на ***un*** и заканчивающееся на ***d***.

Построим регулярное выражение:

un{u∨n∨σ∨d}d{d}ρ,

где ***u***, ***n***, ***d***, - искомые символы, σ - любой символ, ρ - пробел. Это выражение подвергнем минимизации. Расставим индексы мест регулярного выражения.

Сперва выставим индексы основных мест в порядке возрастания (короткие линии на рисунке), затем индексы предосновных мест (длинные линии на рисунке). При выставлении индексов предосновных мест

руководствуемся изложенными выше правилами Расстановка индексов представлена на рисунке 5.

Рисунок 5 – Первоначальная разметка регулярного выражения

Под правило мимнимизации подпадают позиции с номерами 6 и 7. Переопределяем индексы, и выполняем разметку. Новая разметка представлена на рисунке 6.

Рисунок 6 - Разметка регулярного выражения после минимизации

Дальнейшую минимизацию по регулярному выражению произвести не удаётся.

Анализ разметки выражения показывает, что автомат является *недерминированным*, ибо образ функции перехода δ(6, ***d***)={6,7} определён неоднозначно. Выполним построение ДКА по НДКА. Алгоритм построения состоит в следующем.

1) Если НКА является частичным, то доопределим функцию перехода, вводя фиктивное состояние. 2) Построим таблицу переходов и, с её помощью, рассмотрим функцию переходов. Если значение функции не содержится в множестве состояний автомата, то дополним это множество данной функцией. 3) Для вновь введенного состояния строим функцию переходов; анализируем функцию переходов в состояниях, определяющих данное состояние. Определяем выходной сигнал, как дизъюнкцию выходных

сигналов, входящих в данное обобщенное состояние состояний. 4) Повторяем п.п. 2 - 3 до тех пор, пока множество состояний автомата не будет изменено.

Введём обобщённое состояние {6,7} для переходов 6 → 6 и 6 → 7, проанализируем его по входной строке. Иллюстрация применения алгоритма построения детерминированного конечного автомата для данного конкретного случая, приводится на рисунке 7.

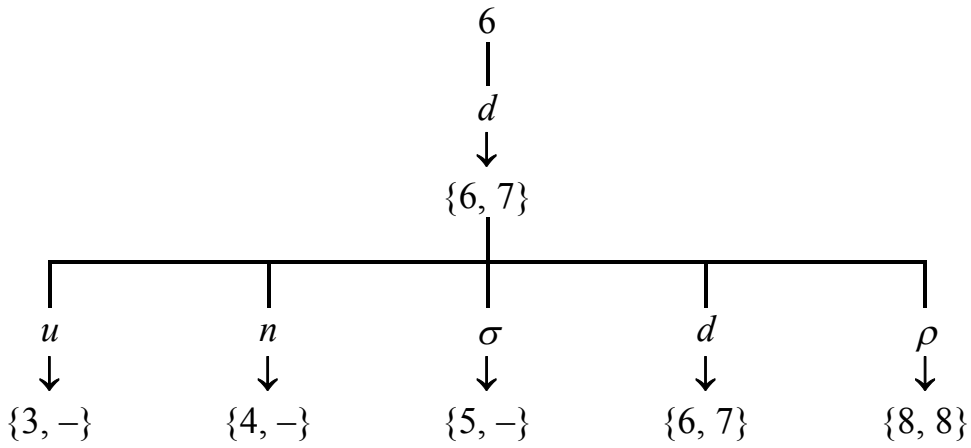


Рисунок 7 – Приведение НКА к ДКА

Обозначим обобщённое состояние {6, 7} ≡ 7, введём состояния: 9 с выходным сигналом *Error*, в котором автомат будет дожидаться пробела (*ρ*), означающего окончание слова, 10 с сигналом *Bed*, соответствующего слову, не удовлетворяющему условию задачи, сигнал *Good* для состояния 8 («правильное» слово), сигнал *x* для всех рабочих состояний. Дополним все неопределенные образы функции переходов переходами в состояние 9, а частично определенные состояния {3, –}, {4, –}, {5, –} дополним до {3, 3}, {4, 4}, {5, 5}. Построим таблицу переходов и выходов конечного автомата.

	Сигнал	*x*	*x*	*x*	*x*	*x*	*x*	*x*	*x*	*Good*	*Error*	*Bed*
	Состояние	0	1	2	3	4	5	6	7	8	9	10
Вход	***u***	1	9	3	3	3	3	3	3	—	9	—
	n	9	2	4	4	4	4	4	4	—	9	—
	d	9	9	6	6	6	6	7	7	—	9	—
	ρ	10	10	10	10	10	10	8	8	—	10	—
	σ	9	9	5	5	5	5	5	5	—	9	—

Из таблицы видно, группы столбцов 2 – 5 и 6,7 имеют одинаковое содержимое и выходные сигналы. Следовательно, *они не различимы по входной строке и по выходному сигналу,* и автомат можно минимизировать по таблице. Обозначим: 0→0 (S), 1→1(A), {2 – 5}→2 (B), {6,7}→3 (C), 8→6

(F), 9→4 (D), 10→5 (E). Преобразуем функцию переходов и выходов КА. Функция переходов МДКА приводится в таблице, а соответствующий ему граф показан на рисунке 8.

	Сигнал	*x*	*x*	*x*	*x*	*Error*	*Bed*	*Good*
	Состояние	0	1	2	3	4	5	6
		S	A	B	C	D	E	F
Вход	***u***	1	4	2	2	4	-	-
	n	4	2	2	2	4	-	-
	d	4	4	3	3	4	-	-
	ρ	5	5	5	6	5	-	-
	σ	4	4	4	2	4	-	-

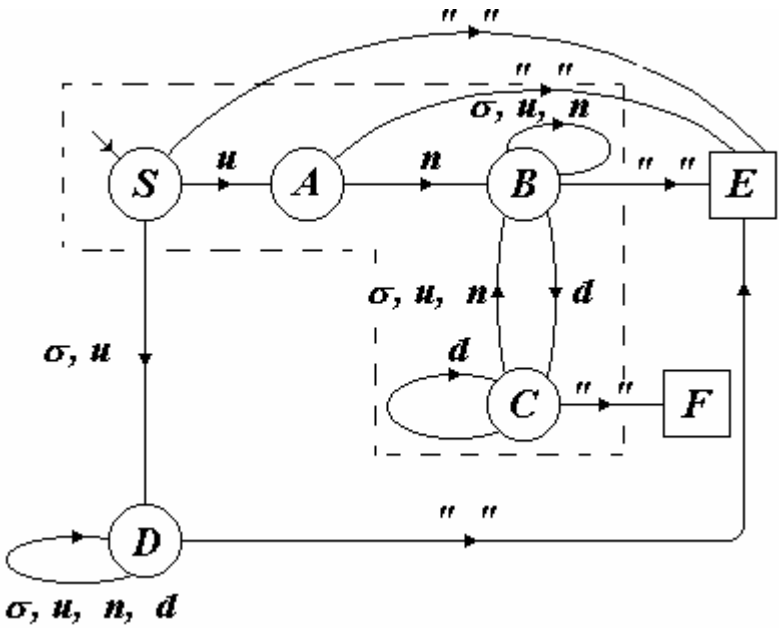


Рисунок 8 – Граф минимального конечного автомата

Вариант программы, реализующей конечный автомат, подсчитывающий числа «правильных» слов в произвольном текстовом файле, на языке программирования **С**и выглядит следующим образом.

```
main()
{
char c;                  /* текущая литера */
int i, n;
file * text;
int mp[5][6] =   {{1, 4, 2, 2, 4}, // управляющая
                                    // таблица
                 {4, 2, 2, 2, 4},
                 {4, 4, 3, 3, 4},
```

```
                    {5, 5, 5, 6, 5},
                    {4, 4, 2, 2, 4}};
text = fopen('xx.txt', 'r'); // исходный текст в файле
i=0; n=0;         /* i – номер состояния КА, n – число
правильных слов */
while ((c = getc(text))! = EOF)
     {
     switch(c)
          {
          case 'u': i = mp[0][i]; break;
          case 'n': i = mp[1][i]; break;
          case 'd': i = mp[2][i]; break;
          case '  ': i = mp[3][i]; break;
          default:  i = mp[4][i];break;
          }
     if (i ==6) {i = 0; n++;}
     if (i ==5) {i = 0;}
     }
printf("количество символов =%d", n);
fclose(text);
}
```