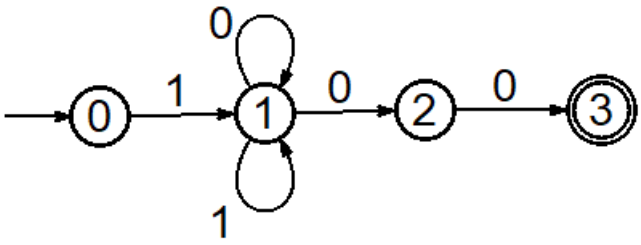
Регулярни (нормални) изрази

Операцуя	Регулярен израз	Вярно	Невярно
Конкатенация	aabaab	aabaab	Всичко друго
ИЛИ	aa | baab	aa baab	Всичко друго
Повторение Затваряне на Клийни (Kleene)	ab*a	aa aba abba	ε ab ababa
Групиране	a(a|b)aab	aaaab abaab	Всичко друго
Глобални символи	a..a	abba abaa	aa aaaaa

Инструменти в Java за работа с регулярни изрази

В Java е включена силно развита библиотека **java.util.regex.***, облекчаваща решаването на правата задача и нейни варианти при работа с регулярни изрази. За решаване на обратната задача все пак ще трябва да разчитаме на взаимното съответствие между нормални изрази и крайни автомати (и на разработената от нас библиотека за работа с автомати). Ще покажем как изглежда най-простото приложение на библиотеката **regex** с конкретен пример.

Нека е даден регулярният израз R=1(0|1)*00. Както можем да се досетим, той описва двоичния запис на всички естествени числа, кратни на 4. Лесно можем да създадем краен автомат, съответен на R:



Java дава възможност да не минаваме през автомат за решаване на правата задача:

```
import java.util.regex.*; //Включваме библиотеката regex
public class Test{
 public static void main(String[] args) {
 //Създаваме обект от тип Pattern (шаблон), който описва регулярния израз.
 //(Обърнете внимание на синтаксиса на методите, с които боравим!)
 Pattern p = Pattern.compile("1(0|1)*00");
 //Създаваме обект от тип Matcher: описва низа, за който ще решаваме правата задача.
 Matcher m=p.matcher("100110100");
 //Методът matches на този обект решава задачата - връща boolean, което очаквано
 // е true, ако низът се поражда от описания шаблон и false, ако не се поражда.
 if (m.matches()) System.out.println("Yes");
 else System.out.println("No");
 }
}
```

Горните етапи могат да се обединят и в едно изречение:

```
 System.out.println((Pattern.matches("1(0|1)*00", "100110100"))?"Yes":"No");
```

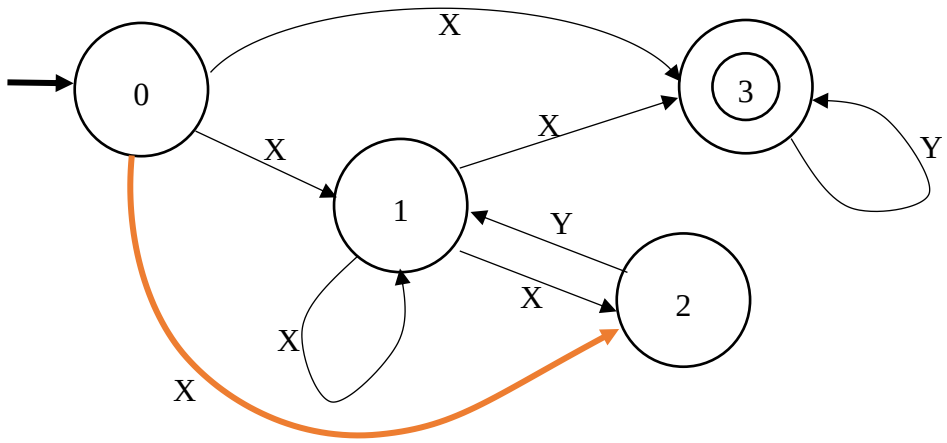
Разликата е само в това, че обикновено на практика по-често имаме един шаблон и множество низове за проверка. В „късото“ изречение шаблонният низ ще се компилира всеки път (създаваните обекти в него са анонимни и нямаме многократен достъп до тях), което прави „късия“ подход по-неефективен.
Така Java ни дава възможност за евентуално откриване на пропуски при превръщане на нормален израз в краен автомат. При тестване чрез обратната задача можем да използваме директно регулярния израз, който сме преобразували, за откриване на неправилно породени думи (за съжаление, в откриването на евентуално пропуснати думи **regex** не може да помогне).

ЗАДАЧИ

Създайте и тествайте краен (детерминиран) автомат, еквивалентен на регулярния израз E.

1) `E=(X|XY)*XY*`
2) `E=(01|10)*1(01)*`
3) `E=(BA|A)*A(B|A)`
4) `E=A(A|B)*AB*`
5) `E=X(XY)*(X|XY)`
6) `E=1(1|00)*01*`

E = (X|XY)*XY*

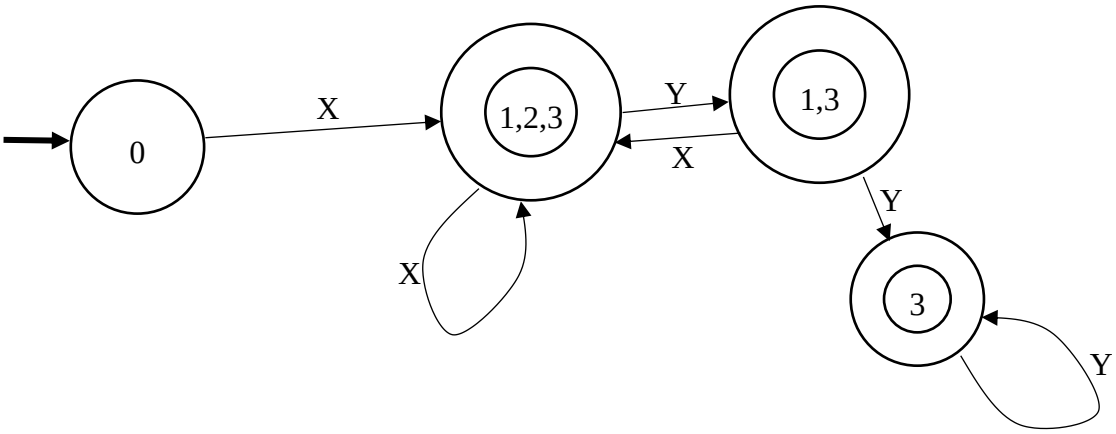


Файл zad1.txt

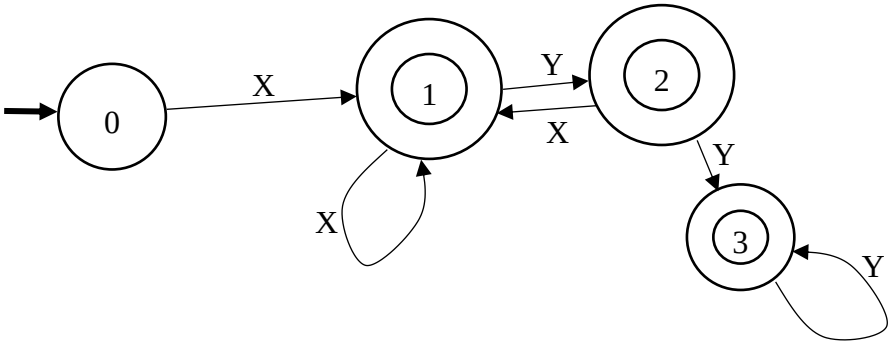
```
3 1
3
8
0 1 X
0 2 X
0 3 X
1 1 X
1 2 X
1 3 X
2 1 Y
3 3 Y
```

Детерминиране

1) Акумулиране



2) Преномериране



Файл zad1d.txt

```
4 3
1 2 3
6
0 1 X
1 2 Y
1 1 X
2 1 X
2 3 Y
3 3 Y
```

```
import autom.*;
import java.util.regex.*;
class Proc implements Callback{
 private String rEx="";
 private Pattern pat;
 private long count;
 public long getCount(){
  return count;
 }
 public void setREx(String s){
  rEx=s;
  pat=Pattern.compile(rEx);
  count=0;
 }
@Override
 public void proceed(String s){
  count++;
  Matcher m=pat.matcher(s);
  System.out.println(s+": "+m.matches());
 }
}
public class Zad1 {
 public static void main(String[] args) {
  Automaton a;
  try{
   //a=new Automaton("zad1d.txt");
   a=new Automaton("zad1d.txt");
  } catch(Exception e){
   System.out.println("No file");
   return;
  }
  Proc p=new Proc();
  p.setREx("(X|XY)*XY*");
  a.makeWords(11, p);
  System.out.println("Count="+p.getCount());
 }
}
```