МГТУ им. Баумана

Рубежный контроль №2

По курсу: "Анализ алгоритмов"

Регулярные выражения

Работу выполнил: Мокеев Даниил, ИУ7-54

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение

Цель работы: изучение возможности регулярных выражений. Реализация поиска по тексу при использовании регулярных выражений и конечного автомата. В ходе рубежного контроля предстоит:

- Изучить регулярные выражения;
- реализовать поиск по тексту всех вхождений цикла for с помощью регулярных выражений и конечного автомата.

1 Аналитическая часть

Регулярные выражения (англ. regular expressions) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters). Для поиска используется строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

1.1 Использование регулярных выражений

Регулярные выражения используются некоторыми текстовыми редакторами и утилитами для поиска и подстановки текста. Например, при помощи регулярных выражений можно задать шаблоны, позволяющие:

- найти все последовательности символов «кот» в любом контексте, как то: «кот», «котлета», «терракотовый»;
- найти отдельно стоящее слово «кот» и заменить его на «кошка»;
- найти слово «кот», которому предшествует слово «персидский» или «чеширский»;
- убрать из текста все предложения, в которых упоминается слово кот или кошка.

Регулярные выражения позволяют задавать и гораздо более сложные шаблоны поиска или замены.

Результатом работы с регулярным выражением может быть:

- проверка наличия искомого образца в заданном тексте;
- определение подстроки текста, которая сопоставляется образцу;
- определение групп символов, соответствующих отдельным частям образца.

Если регулярное выражение используется для замены текста, то результатом работы будет новая текстовая строка, представляющая из себя исходный текст, из которого удалены найденные подстроки (сопоставленные образцу), а вместо них подставлены строки замены (возможно, модифицированные запомненными при разборе группами символов из исходного текста). Частным случаем модификации текста является удаление всех вхождений найденного образца — для чего строка замены указывается пустой.

1.2 Регулярные выражения в теории формальных языков

Регулярные выражения состоят из констант и операторов, которые определяют множества строк и множества операций на них соответственно. Определены следующие константы:

- (пустое множество) $\emptyset$.
- (пустая строка) ε обозначает строку, не содержащую ни одного символа; эквивалентно "".
- (символьный литерал) "a где a — символ используемого алфавита.
- (множество) из символов, либо из других множеств. и следующие операции:
- (сцепление, конкатенация) RS обозначает множество $\alpha\beta|\alpha \in R \& \beta \in S$. Например, {"boy "girl"}{"friend "cott"} = {"boyfriend "girlfriend "boycott "girlcott"}.
- (дизъюнкция, чередование) $R|S$ обозначает объединение R и S.

  Например, {"ab "c"}|{"ab "d "ef"} = {"ab "c "d "ef"}.
- (замыкание Клини, звезда Клини) R* обозначает минимальное надмножество множества R, которое содержит ε и замкнуто относительно конкатенации. Это есть множество всех строк, полученных конкатенацией нуля или более строк из R.

  Например, {"Run "Forrest"}* = {ε, "Run "Forrest "RunRun "RunForrest "ForrestRun "ForrestForrest "RunRunRun "RunRunForrest "RunForrestRun }.
- Регулярные выражения, входящие в современные языки программирования (в частности, PCRE), имеют больше возможностей, чем то, что называется регулярными выражениями в теории формальных языков; в частности, в них есть нумерованные обратные ссылки. Это позволяет им разбирать строки, описываемые не только регулярными грамматиками, но и более сложными, в частности, контекстно-свободными грамматиками.

1.3 Вывод

Были рассмотренны регулярныные выражения и их обоснование в теории формальных языков.

2 | Конструкторская часть

2.1 Требования к программе

Требования к вводу:

- На вход подается код программы на языке golang в формате .txt;
- количество вхождений цикла for в строке не больше 1.

Требования к программе:

- Программа выводит все строки, в которых встречается вхождения условия цикла for;
- программа должна выводить строки с циклом for даже если его условие некорректно.

2.2 Матрица переходов

В этом разделе будет рассмотрена матрица переходов конечного автомата (Рис. 2.1)

	*	f	o	r	{	/n
0	0	1	0	0	0	0
1	0	0	2	0	0	0
2	0	0	0	3	0	0
3	3	3	3	3	4	0
4	0	0	0	0	0	found
	0 = beggining					
	4 = final					

Рис. 2.1: Таблица переходов состояний

2.3 Вывод

В данном разделе была рассмотрена матрица переходов реализованного автомата и требования к работе программы.

3 | Технологическая часть

3.1 Выбор ЯП

Я выбрал в качестве языка программирования Golang, потому как он достаточно удобен.

3.2 Листинг кода алгоритмов

В данном разделе будет представлен листинги кода поиска с помощью регулярных выражений (3.1), с помощью конечного автомата(3.2)

Листинг 3.1: Использование регулярных выражений

```
func get_fors(name_file string) []string{
  answer:=make([]string, 0)
  for_pattern:="(for).*\\{+"
  get_for, _ := regexp.Compile(for_pattern)
  file, err := os.Open(name_file)
  if err != nil{
    fmt.Println(err)
    os.Exit(1)
  }
  defer file.Close()
  reader := bufio.NewReader(file)
  for {
    str, err := reader.ReadString('\n')
    if err == io.EOF{
    break}
    if (get_for.MatchString(str)){
      answer = append(answer, get_for.FindString(str))
    }
  }
  return answer
}
```

Листинг 3.2: Использование конечного автомата

```
type avt struct {
  state int
  trans map[key]int
}

type key struct{
  symb byte
  state int
```

```
}

func new_avt() *avt{
  avt := new(avt)
  avt.state  = 0
  avt.trans = make(map[key]int)
  avt.trans[key{symb: byte('f'), state: 0}] = 1
  avt.trans[key{symb: byte('f'), state: 1}] = 0
  avt.trans[key{symb: byte('f'), state: 2}] = 0
  avt.trans[key{symb: byte('f'), state: 3}] = 3
  avt.trans[key{symb: byte('f'), state: 4}] = 0

  avt.trans[key{symb: byte('o'), state: 0}] = 0
  avt.trans[key{symb: byte('o'), state: 1}] = 2
  avt.trans[key{symb: byte('o'), state: 2}] = 0
  avt.trans[key{symb: byte('o'), state: 3}] = 3
  avt.trans[key{symb: byte('o'), state: 4}] = 0

  avt.trans[key{symb: byte('r'), state: 0}] = 0
  avt.trans[key{symb: byte('r'), state: 1}] = 0
  avt.trans[key{symb: byte('r'), state: 2}] = 3
  avt.trans[key{symb: byte('r'), state: 3}] = 3
  avt.trans[key{symb: byte('r'), state: 4}] = 0

  avt.trans[key{symb: byte('{'), state: 0}] = 0
  avt.trans[key{symb: byte('{'), state: 1}] = 0
  avt.trans[key{symb: byte('{'), state: 2}] = 0
  avt.trans[key{symb: byte('{'), state: 3}] = 4
  avt.trans[key{symb: byte('{'), state: 4}] = 0

  return avt
}
func (avt *avt) change_state(char byte){
  key := key{char, avt.state}
  if st, ok := avt.trans[key]; ok{
    avt.state = st
  }else{
    if avt.state == 3{
      if char ==  byte('\n'){
        avt.state = 0
      }else{
        avt.state = 3
      }
    }else{
      avt.state = 0
  }
  }
}
func get_fors(name_file string) []string{
  answer:=make([]string, 0)
  file, err := os.Open(name_file)
  if err != nil{
    fmt.Println(err)
```

```
62      os.Exit(1)
63    }
64    defer file.Close()
65    reader := bufio.NewReader(file)
66    for {
67      str, err := reader.ReadString('\n')
68      str = strings.TrimSuffix(str, "\r\n")
69      avt := new_avt()
70      if err == io.EOF{
71      break}
72      for _, j := range str{
73        avt.change_state(byte(j))
74      }
75      if avt.state == 4{
76        answer = append(answer, str)
77      }
78    }
79    return answer
80  }
```

3.3 Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы.

Заключение

В ходе лабораторной работы были изучены возможности регулярных выражений. Был реализован алгоритм поиска вхождений подстроки при помощи регулярных выражений.

Литература

[1] Фридл, Дж. Регулярные выражения = Mastering Regular Expressions. — СПб.: «Питер», 2001. — 352 с. — (Библиотека программиста).

[2] Смит, Билл. Методы и алгоритмы вычислений на строках (regexp) = Computing Patterns in Strings. — М.: «Вильямс», 2006. — 496 с.

[3] Форта, Бен. Освой самостоятельно регулярные выражения. 10 минут на урок = Sams Teach Yourself Regular Expressions in 10 Minutes. — М.: «Вильямс», 2005. — 184 с.