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

2 | Конструкторская часть

Требования к вводу: На вход подаются две матрицы
Требования к программе:

- корректное умножение двух матриц;
- при матрицах неправильных размеров программа не должна аварийно завершаться.

2.1 Вывод

В данном разделе была рассмотрена схема алгоритма Винограда и способ ее распараллеливания.

3 | Технологическая часть

3.1 Выбор ЯП

Я выбрал в качестве языка программирования Golang, потому как он достаточно удобен, быстр и успешно использует концепции мультипоточного программирования.

Время работы алгоритмов было замерено с помощью функции Now() из библиотеки time.

3.2 Описание структуры ПО

В данном разделе будет представленна функциональная схема программы (Рис. 3.1.)

3.3 Сведения о модулях программы

3.4 Листинг кода алгоритмов

В данном разделе будет представлен листинги кода алгоритма Винограда (3.1), распараллеленого алгоритма Винограда (3.2)

Листинг 3.1: Алгоритм Винограда

```
1 func winograd(mtr1, mtr2 *mtr) *mtr{
2   if mtr2.rows != mtr1.cols{
3     panic("Wrong size of matrix")
4   }
5   row1 := mtr1.rows; col1 := mtr1.cols
6   row2:= mtr2.rows; col2 := mtr2.cols
```

```
7
8   row_factor := make([]int, row1, row1)
9   col_factor := make([]int, col2, col2)
10
11  for i:=0; i<row1;i++{
12    for j:=0;j<col1 / 2;j++{
13      row_factor[i] += mtr1.buff[i][2*j] * mtr1.buff[i][2*j
          +1]
14  }}
15
16  for i:=0; i<col2;i++{
17    for j:=0;j<row2 / 2;j++{
18      col_factor[i] += mtr2.buff[2*j][i]*mtr2.buff[2*j+1][i
          ]
19  }}
20
21    answer := new_mtr(row1, col2)
22
23  for i:=0;i<row1;i++{
24    for j:=0;j<col2;j++{
25      answer.buff[i][j]+= - row_factor[i] - col_factor[j]
26      for k:=0;k<col1 / 2;k++{
27        answer.buff[i][j] += ((mtr1.buff[i][2 * k] + mtr2.
            buff[2 * k + 1][j]) * (mtr1.buff[i][2 * k + 1] +
             mtr2.buff[2 * k][j]))
28  }}}
29
30  if (row2 % 2) != 0{
31    for i:=0;i<row1;i++{
32      for j:=0;j<col2;j++{
33        answer.buff[i][j] += mtr1.buff[i][col1 - 1] * mtr2.
            buff[col1 - 1][j]
34  }}}
35  return answer
36 }
```

Листинг 3.2: Распараллеленый Алгоритм Винограда

```
1 func winograd_parallel(mtr1, mtr2 *mtr, threads int) *mtr{
2   if mtr2.rows != mtr1.cols{
3     panic("Wrong size of matrix")
```

```
4   }
5   row1 := mtr1.rows; col1 := mtr1.cols
6   row2:= mtr2.rows; col2 := mtr2.cols
7
8   row_factor := make([]int, row1, row1)
9   col_factor := make([]int, col2, col2)
10
11  for i:=0; i<row1;i++{
12    for j:=0;j<col1 / 2;j++{
13      row_factor[i] += mtr1.buff[i][2*j] * mtr1.buff[i][2*j
          +1]
14  }}
15  for i:=0; i<col2;i++{
16    for j:=0;j<row2 / 2;j++{
17      col_factor[i] += mtr2.buff[2*j][i]*mtr2.buff[2*j+1][i
          ]
18  }}
19  answer := new_mtr(row1, col2)
20  in := make(chan int); quit := make(chan bool)
21  mult := func(){
22    for{
23      select {
24        case i := <- in:
25          for j:=0;j<col2;j++{
26          answer.buff[i][j]+= - row_factor[i] - col_factor[
              j]
27          for k:=0;k<col1 / 2;k++{
28          answer.buff[i][j] += ((mtr1.buff[i][2 * k] + mtr2
              .buff[2 * k + 1][j]) * (mtr1.buff[i][2 * k +
              1] + mtr2.buff[2 * k][j]))
29          }}
30      case <- quit:
31        return
32      }
33    }
34  }
35  for i:=0;i<threads;i++{
36    go mult()
37  }
38  for i:=0;i<mtr1.rows;i++{
```

```
39      in <- i
40    }
41    for i:= 0 ; i<threads; i++{
42      quit<-true
43    }
44    return answer
45  }
```

3.5 Вывод

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

4 | Исследовательская часть

Был проведен замер времени работы алгоритмов с использованием разного колличества потоков. Исследования были проведены на процессоре Intel Core i5-6200U

4.1 Примеры работы

В данном разделе приведен пример работы программы (Рис. 4.1)

4.2 Постановка эксперемента

Проведем сравнение для каждого из алгоритмов. Для замера времени будем использовать функцию time.Now()

Сравним результаты для обычного Винограда и Винограда с распараллеленным главным циклом:

4.2.1 Вывод эксперементальной части

Экперемент показывает, что использование одного потока эквивалентно обычной версии алгоритма. Использование двух потоков значительно ускоряет работу алгоритма. Данный эффект наблюдается и при увеличении числа потоков, однако при использовании более четырех потоков дальнейшего ускорения не происходит.

Заключение

В ходе лабораторной работы были изучены возможности параллельных вычислений, реализован алгоритм Винограда умножения матриц с помощью параллельных вычислений.

Было произведено сравнение работы обычного алгоритма Винограда и параллельной реализации при увеличении количества потоков. Выяснилось, что увеличение потоков до 4х сокращает время работы на 70% по сравнению с однопоточной реализацией. Однако дальнейшее увеличение количества потоков не дает значительного выигрыша во времени (разница менее 1%).

Литература

[1] Фридл, Дж. Регулярные выражения = Mastering Regular Expressions. — СПб.: «Питер», 2001. — 352 с. — (Библиотека программиста). — ISBN 5-318-00056-8.

[2] Смит, Билл. Методы и алгоритмы вычислений на строках (regexp) = Computing Patterns in Strings. — М.: «Вильямс», 2006. — 496 с. — ISBN 0-201-39839-7.

[3] Форта, Бен. Освой самостоятельно регулярные выражения. 10 минут на урок = Sams Teach Yourself Regular Expressions in 10 Minutes. — М.: «Вильямс», 2005. — 184 с. — ISBN 5-8459-0713-6.