Eine Funktion zur Extraktion von Tabellen aus Internetseiten

Peter Wolf,
Pfad: /home/pwolf/R/div

13. September 2013, 17. Januar 2014

Inhalt

1 Einleitung

Ausgangspunkt zum Entwurf einer Funktion zur Extraktion von Tabellen aus Internetseiten war der Hinweis von D. Trenkler, dass auf der Internetseite `www.transfermarkt.de` sehr schöne Daten für die Statistik-Ausbildung zu finden sind. Beispielsweise können wir einen Blick auf die folgenden Seiten werfen:

1 ⟨*einige Seiten des Servers* `www.transfermarkt.de` 1⟩ ≡ ⊂ 12

```
urls <- c("http://www.transfermarkt.de/de/default/top-teams/basics.html",
          "http://www.transfermarkt.de/de/default/marktwerte/basics.html",
          "http://www.transfermarkt.de/de/spieler/goldenerschuh/basics.html",
          "http://www.transfermarkt.de/de/statistiken/transferrekorde/transfers.html",
```

```
          "http://www.transfermarkt.de/de/default/rekorde/erfolge.html",
          "http://www.transfermarkt.de/de/geruechtekueche/uebersicht/geruechte.html",
          "http://www.transfermarkt.de/de/geruechtekueche/mostwanted/geruechte.html",
          "http://www.transfermarkt.de/de/1-bundesliga/geruechtequellen/wettbewerb_L1.html",
          "http://www.transfermarkt.de/de/default/basics/marktwert-aenderung.html",
          "http://www.transfermarkt.de/de/default/marktwerte/basics.html",
          "http://www.airdisaster.com/cgi-bin/view_year.cgi?year=2008", "flugzeugunfaelle.html"
)
```

Auf Basis der ersten Seite lässt zum Beispiel das folgende Objekt extrahieren:

```
http://www.transfermarkt.de/de/default/top-teams/basics.html read
body extracted
contents of page saved in aa.txt
only one table found
  # -         Verein       Wettbewerb Kader Durchschnitts-Alter Marktwert
1 1     FC Barcelona Primera Division    25                26.8 587300000
2 2      Real Madrid Primera Division    23                25.8 571500000
3 3   Bayern M\"unchen     1.Bundesliga    26                  25.7 483650000
4 4          Chelsea   Premier League    27                27.3 475750000
5 5         Man City   Premier League    24                27.3 469750000
  Durchschnitts-MW
1         23492000
2         24847826
3         18601924
4         17620370
5         19572916
```

Um ein solches Ergebnis zu erhalten, kann die Funktion `get.html.tables()` – wie folgt – aktiviert werden. Daneben sind als Kommentartexte Zugriffe auf die weiteren Beispiele zu sehen.

Hinweis: Falls ein ungeeignetes Encoding gewählt wird, werden für entsprechende Zeilen NA-Einträge erzeugt und dadurch eventuell die Tag-Strukturen zerstört.

2 ⟨*erste urls auswerten* 2⟩ ≡

```
⟨define get.html.tables() 4⟩
 result <- get.html.tables(urls[12], encoding="latin1",first=1,depth=4,collapse.sep="|",
                           body.to.file="aa.txt", conversion.to.numbers=FALSE)
    result <- t(sapply(strsplit(unlist(result),"[|]")[-1], function(x) x[1:6]))
    result <- sub("^ *","",result); result <- sub(" *$","",result)[-1,]
    # a <- as.Date(result[,1],format="%m.%d.%Y"); sum(diff(a)[-1])
# result <- get.html.tables(urls[1], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[2], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[3], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[4], pattern="40.000.000",encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[5], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[6], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[7], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[8], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[9], encoding="utf8",first=1,body.to.file="aa.txt")
# result <- get.html.tables(urls[10], encoding="utf8",first=1,body.to.file="aa.txt")
if(is.data.frame(result) || is.matrix(result)) print(result[1:5,]) else {
  for(j in seq(result)) {
    cat("--------- Matrix",j,"----------")
    if(is.data.frame(result[[j]]) || is.matrix(result[[j]])) print(result[[j]][1:5,])
  }
}
#result
```

2 Die Syntax der Funktion `get.html.tables()`

Für den Job der Tabellenextraktion müssen vier Dinge geleistet werden:

- Fokussierung: Entfernung der Dinge, die vor oder hinter der Tabelle stehen
- Reduktion: Entfernung aller Dinge, die für die Tabelle irrelevant sind
- Translation: Übersetzung der Tabelle in eine R-Datenstruktur
- Interpretation: Aufbereitung der Tabellenspalten, so dass sie gut verwendbar sind.

Die Extraktion von Tabellen wird dadurch erschwert, dass manche Seiten mehrere Tabellen enthalten. Weiterhin gibt es Tabellen, in deren Feldern sich wiederum komplette Tabellen befinden. Während der erste Fall zu einer iterativen Abarbeitung führt, ist der zweite pragmatisch durch Aufhebung der inneren Tabellenstruktur lösbar. Die Funktion `get.html.tables()` soll die Extraktion umsetzen und mehrfache Tabellen berücksichtigen.

Syntax: Die Argumente dienen dazu, die gewünschte Tabelle zu spezifizieren:

`file` Der Funktion `get.html.tables()` muss natürlich die Stelle übergeben werden, an der sie die fragliche Seite findet.

`pattern` Wenn die Seite mehrere Tabellen enthält, kann mit Hilfe eines regulären Musters, das einen Zelleninhalt beschreibt, eine Tabelle zur Extraktion definiert werden.

`table.number` Falls eine Seite mehrere Tabellen enthält, kann über das zweite Argument die gewünschte Tabelle gewählt werden. Dieses Argument kommt nur zum Einsatz, wenn nicht vom `pattern`-Argument Gebrauch gemacht wird.

`depth` Manchmal ist die gewünschte Tabelle in einer Zelle einer anderen Tabelle untergebracht. Dann lässt sich mit dem Parameter `depth` die Schachtelungstiefe der gesuchten Tabelle angeben. Dieses Argument kommt nur zum Einsatz, wenn nicht vom `pattern`-Argument Gebrauch gemacht wird.

`collapse.sep` Wenn eine hoffentlich unbrauchbare Tabellen-Unterstruktur aufgrund der Tiefenwahl entfernt werden muss, werden Elemente einer aufzulösenden Tabelle in eine Zeichenkette überführt. Hierzu kann mit dem Seperator `collapse.sep` ein Trennzeichen definiert werden, das sich von `/` unterscheidet.

`first.lines.to.header` Oft steht in der ersten Zeile einer Tabelle der Header; durch Setzung des dritten Argument auf `TRUE` kann diese Zeile zum Header gemacht werden. Müssen mehrere Zeilen für die Header-Information verwendet werden, ist die Zeilenanzahl dem Parameter zuzuweisen.

`conversion.to.numbers` Mit `conversion.to.numbers=FALSE` kann eine Zahleninterpretation von Spalten unterdrückt werden. Andernfalls werden Spalten mit mehr als 50% Zahleneinträgen in Zahlen überführt.

`encoding` Encoding der Inputinformation.

`body.to.file` Falls man nicht weiß, welche Parameter richtig sind, kann man den extrahierten Body in eine Datei schreiben und dann mit dieser verschiedene Zugriffe ohne Internet-Einsatz ausprobieren.

`show.structure` Manchmal möchte man die Tag-Struktur sehen, die sich auf (geschachtelte) Tabellen bezieht. Hierzu kann man das Flag `show.structure` auf `TRUE` setzen.

3 ⟨* 3⟩ ≡
```
args(get.html.tables)
```

```
function (file, pattern = "", table.number = NA, depth = 1, first.lines.to.header = FALSE,
    conversion.to.numbers = TRUE, encoding = "latin1", body.to.file = "", show.structure=FALSE)
NULL
```

3 Bemerkungen zur Vorgehensweise

Die Seite wird als Textdatei eingelesen, der Body extrahiert und von unwichtigen Dingen bereinigt. Danach werden dann alle Table-Umgebungen gesucht und verarbeitet.

Eine Tabelle: Im einfachsten Fall besitzt die Seite nur eine Tabelle. Dann wird ihr Inhalt mit der loaklen Hilfsfunktion `extract.table()` extrahiert und ausgegeben.

Meherere Tabellen Falls mehrere Tabellen-Umgebungen ermittelt worden sind, werden zwei Fälle unterschieden:

a) Es liegt eine Schachtung von Tabellen vor.

b) Es stehen mehrere Tabellen sequenziell hinter einander.

Der Fall geschachtelter Tabellen wird aufgelöst, indem alle Tabellen-Strukturen mit größerer oder geringerer Tiefe als die Tiefe der gewünschten Tabelle(n) entfernt werden. Dann liegt Fall b) vor und es werden – wenn erwünscht – mehrere Ergebnistabellen erzeugt, die letztlich als Elemente einer Liste ausgegeben werden.

Grobstruktur der Funktion `get.html.tables()`: Nach der Definition einiger Hilfsfunktionen wird die html-Seite eingelesen, der Body extrahiert und alle `table`-Umgebungen werden von der Hilfsfunktion von `find.tag.pairs()` ermittelt. `find.tag.pairs()` stellt die Positionen von Tabellen-Tags wie auch Schachtelungstiefen fest und trägt diese Infos in das Verwaltungsobjekt `idx.tables` ein. Für alle Tabellen-Tags mit einer Tiefe verschieden von `depth` wird die Tabellenstruktur entfernt. Hierzu kommt die Funktion `paste.table()` zum

Einsatz. Nach dieser Vorbehandlung werden ggf. in einer Schleife alle verbleibenden Tabellen per `extract.table()` extrahiert und als Liste ausgegeben. Falls beim Aufruf jedoch eine spezielle Tabelle spezifiziert wurde, wird nur diese ausgegeben.

4 ⟨*define* `get.html.tables()` 4⟩ ≡ ⊂ 2, 5, 12, 20, 23, 26, 27, 28, 29, 30

```
get.html.tables <- function(file, pattern="", table.number=NA, depth=1, collapse.sep="/",
      first.lines.to.header=FALSE, conversion.to.numbers=TRUE,
      encoding="latin1", body.to.file="", show.structure=FALSE){
  ⟨define some functions 13⟩
  ⟨read html-page 8⟩
  ⟨extract body 9⟩
  ⟨find tables 10⟩
  if(1 == nrow(idx.tables)){
    ⟨extact table in case of one table 6⟩
  } else {
    ⟨handle more than one tables 7⟩
  }
  if(length(result)==1) result[[1]] else result
}
# res <- get.html.tables(urls[12], encoding="Latin1",first=1,depth=5,body.to.file="aa.txt")
```

Noch ein paar Testbeispiele: Spannend sind Testbeispiele. Deshalb folgen einige sogleich.

5 ⟨*test* get.html.tables() 5⟩ ≡ ⊂ 25

```
⟨define get.html.tables() 4⟩
# h<-get.html.tables("http://www.transfermarkt.de/de/default/top-teams/basics.html",first=1,conversion.to.numbers=TRUE,encoding="utf8")
# h<-get.html.tables("http://www.transfermarkt.de/de/default/marktwerte/basics.html",first=1)
# h<-get.html.tables("http://www.transfermarkt.de/de/spieler/goldenerschuh/basics.html",first=1)
# h<-get.html.tables("http://www.transfermarkt.de/de/statistiken/transferrekorde/transfers.html",table.number=2,first=1)
# h<-get.html.tables("http://www.focus.de/finanzen/karriere/die-einkommens-tabelle-wo-in-deutschland-die-loehne-am-hoechsten-sind_aid_738960.html",table.number=3,first=2,encodin
# h<-get.html.tables("http://www.sport1.de/dynamic/datencenter/sport/team-tabelle/fussball/arminia-bielefeld",table.number=1,first=1,encoding="utf8")
# h<-get.html.tables("http://www.meteoblue.com/de_DE/wetter/vorhersage/woche/bielefeld_de_11720",first=2)
# h<-get.html.tables("http://www.lanuv.nrw.de/luft/immissionen/aktluftqual/eu_luft_akt.htm",pattern="AABU",first=2); plot(h[,5:6])
 h<-get.html.tables("http://www.lanuv.nrw.de/luft/immissionen/aktluftqual/eu_pm10_akt.htm",pattern="AABU",first=2);plot(h[,3:4])
if(!is.data.frame(h) && !2 == length(dim(h))) {
  for(i in 1:length(h)){ cat("--- Matrix",i,"------"); ii <- min(5,nrow(h[[i]])); print(h[[i]][1:ii,]) }
} else {
  ii <- min(5,nrow(h)); print(h[1:ii,])
}
```

4 Verfeinerungen

Fall einer einzigen Tabelle: Wir müssen im einfachsten Fall nur den Bereich der einzigen Table-Umgebung auswerten.

6 ⟨*extact table in case of one table* 6⟩ ≡ ⊂ 4

```
cat("only one table found")
txt <- txt[ idx.tables[1]:idx.tables[2] ]
result <- extract.table(txt,conversion.to.numbers)
```

Mehrere Tabellen: Im Fall mehrerer Table-Umgebungen werden zunächst die genesteten Tabellen aufgelöst und in einfache Tabellenfelder umgewandelt. Dann werden die Grenzen der verbleibenden Tabellen bestimmt. Wenn bspw. über das Argument `pattern` nur eine Ergebnistabelle definiert wird, wird diese

mittels `extract.table()` extrahiert. Andernfalls wird die Extraktionsfunktion mehrmals aufgerufen und die Ergebnisse werden in einer Liste gesammelt.

Eine Setzung des Arguments `pattern` erfordert eine Untersuchung des Quelltextes. Dabei wird ermittelt, welche Nummer und welche Schachtelungstiefe die angepeilte Tabelle besitzt. Wird das Muster mehrfach gefunden, wird nur die erste Fundstelle ausgewertet.

7 ⟨*handle more than one tables* 7⟩ ≡ ⊂ 4

```
cat("more than one table or nested tables found")
# search table dependent on pattern
if(pattern != ""){
  index <- grep(pattern, txt) # ; cat("index",index); print(idx.tables)
  if(0 < length(index)){
    depth <- max(idx.tables[ idx.tables[,1] < index[1] &
                             idx.tables[,2] > index[1],, drop=FALSE ][,3])
    cat("depth of table with pattern",pattern,":",depth)
    h <- idx.tables[ idx.tables[,3] == depth,, drop=FALSE ]
    table.number <- which( h [,1] < index[1] & h[,2] > index[1])[1]
    cat("table.number of table with pattern",pattern,":",table.number)
  }
cat("DEPTH:",depth)
}
# collapse inner tables
inner.table <- FALSE  #; print(idx.tables)
for( id in seq(idx.tables[,1]) ){
  if(idx.tables[id,3] == depth) next else inner.table <- TRUE
  if(idx.tables[id,3] > depth) {
# print(txt[ idx.tables[id,1]:idx.tables[id,2] ])
    ptable <- paste.table(txt[ idx.tables[id,1]:idx.tables[id,2] ],sep=collapse.sep)
    ptable <- c(ptable, rep(" ",diff(idx.tables[id,1:2])))
    txt[ idx.tables[id,1]:idx.tables[id,2] ] <- ptable
  }
  if(idx.tables[id,3] < depth) {
    txt[ c(idx.tables[id,1],idx.tables[id,2]) ] <- "<not interesting table>"
  }
}
if(inner.table) cat("nested table structure removed")
# find table(s)
idx.tables <- find.tag.pairs("table",txt); n.tab <- nrow(idx.tables); result <- NULL
# cat("idx.tables.new:") ; print(idx.tables)
if(!any(is.na(table.number))){
  for(tn in seq(along=table.number)){
    table.txt <- txt[idx.tables[table.number[tn],1]:idx.tables[table.number[tn],2]]
    result <- c(result,list(extract.table(table.txt, conversion.to.numbers)))
    cat("table", table.number[tn], "extracted")
  }
  if(length(result)==1) result <- result[[1]]
} else {
  for(i in 1:n.tab){
    table.txt <- txt[idx.tables[i,1]:idx.tables[i,2]]
    result <- c(result, list(extract.table(table.txt, conversion.to.numbers)))
  }
  if(length(result)==1) result <- result[[1]]
  cat(length(result), "table(s) extracted")
}
```

Einlesen: Die Datei wird als Text mit Hilfe von `readline()` eingelesen.

8 ⟨*read html-page* 8⟩ ≡ ⊂ 4

```
txt <- scan(file, what="", sep="\n") #; txt <- scan("tank.txt","",sep="\n")
txt <- iconv(txt,encoding,"")
cat(file,"read")
# cat(txt,sep="\n",file="nice.html")
```

Extraktion des Body: Zunächst trennen wir Tags und Inhalte. Dazu überführen wir die einzelnen Zeilen in eine einzige Zeichenkette. Dann markieren wir die Stellen, öffnender und schließender Tags und splitten die Zeichenkette an den Tags-Stellen auf, so dass anschließend alle Tags in isolierten Zeilen stehen. Im nächsten Schritt überführen wir alle Großbuchstaben der Tags in kleine. Tabulatorenzeichen werden in Leerzeichen überführt und einige Html-Zeichen ersetzt. Leerzeilen werden entfernt, wie auch führende und endende Leerzeichen.

Bei TD- und TH-Tags kann angegeben werden, wie viele Spalten das nächste Feld umfassen soll. Dieser Fall wird dadurch abgehandelt, dass entsprechend der Setzung von `colspan` leere Felder ergänzt werden.

Mit Hilfe der Funktion `find.tag.txt()` werden die `body`-Tags gesucht, um alle Zeilen außerhalb des Körpers zu entfernen. `remove.not.needed.tags()` entfernt aus dem Text alle nicht relevanten Tags.

Falls `body.to.file` mit einem Namen belegt ist, wird die entkernte Html-Seite gespeichert.

9 ⟨*extract body* 9⟩ ≡ ⊂ 4

```
txt <- paste(txt,collapse="")
# find tags and split txt
txt <- gsub("[<]([^/])","TAGAUF<\\1",txt); txt <- gsub("</","TAGZU</",txt)
txt <- unlist(strsplit(txt,"TAGAUF"))[-1]; txt <- unlist(strsplit(txt,"TAGZU"))
txt <- sub(">",">TAGEND",txt)
txt <- unlist(strsplit(txt,"TAGEND"))
idx <- substring(txt,1,1) == "<"; txt[idx] <- tolower(txt[idx])
# remove blanks
txt <- gsub("\t"," ",txt); txt <- grep("^ *$",txt,invert=TRUE,value=TRUE)
txt <- sub("^ *","",txt); txt <- sub(" *$","",txt)
txt <- sub("^[<]th","<td",txt); txt <- sub("^[<][/]th","</td",txt)
# extract body
idx <- find.tag.txt("body"); txt <- txt[idx[1]:idx[2]]
# convert some symbols
txt <- gsub("&oslash;","Durchschnitts",txt)
txt <- gsub("[&]nbsp;"," ",txt);  txt <- gsub("&euro;"," ",txt)
txt <- gsub("[&]auml;","ae",txt); txt <- gsub("[&]ouml;","oe",txt);
txt <- gsub("[&]uuml;","ue",txt); txt <- gsub("[&]Auml;","Ae",txt);
txt <- gsub("[&]Ouml;","Oe",txt); txt <- gsub("[&]Uuml;","Ue",txt);
txt <- gsub("[&]szlig;","ss",txt)
# handle colspan
idx <- grep("colspan",txt)
if(0 < length(idx)){
  # get colspan lines
  txt.colspan <- txt[idx]
  # find number of cols
  #anz <- sub('.*colspan *[=] *["]{0,1}([0-9]+).*','\\1",txt.colspan)
  anz <- sub(".*colspan *[=][ ']*[\"]{0,1}([0-9]+).*","\\1",txt.colspan)
  anz <- as.numeric(anz) #; print(anz)
  # remove colspan argument
  txt.colspan <- sub("([<][/]{0,1}td) .*","\\1>",txt.colspan)
  # add tags
  anz <- anz[!is.na(anz)]; if(0 == length(anz)) anz <- 1
```

```
  h <- rep("<td>NEWTAG NEWTAG</td>NEWTAG",max(anz))
  h <- sapply(anz, function(x) paste(h[1:x][-1],collapse=""))
  txt.colspan <- paste(h, txt.colspan,sep="")
  # print(txt.colspan); print(unlist(strsplit(txt.colspan,"NEWTAG")))
  # update txt
  txt[idx] <- txt.colspan
  txt <- unlist(strsplit(txt,"NEWTAG"))
}
# remove unimportant tags
txt <- remove.not.needed.tags(txt)
cat("body extracted")
if("" != body.to.file){
  spaces <- "                                                    "
  indent <- c(0,cumsum( substring(txt,1,6) == "<table"))
  indent <- indent[-length(indent)] - cumsum(substring(txt,1,7) == "</table")
  txtout <- paste(sep="","<!--depth: ",indent,"-->",
                  substring(spaces,1, 2 + 2*indent),iconv(txt,"",encoding))
  cat("<body>",txtout,"</body>",file=body.to.file,sep="\n")
  cat("contents of page saved in", body.to.file)
}
```

Tabellensuche: Die letzte wichtige Vorarbeit besteht darin potentielle Tabellen zu suchen. Dieses übernimmt die Funktion `find.tag.pairs()`.

10 ⟨*find tables* 10⟩ ≡ ⊂ 4

```
idx.tables <- find.tag.pairs("table",txt)
if(show.structure){
  ⟨show structure 11⟩
}
```

11 ⟨*show structure* 11⟩ ≡ ⊂ 10

```
depth.vec <- rep(0,length(txt))
ind.tab <- idx.tables[ order(idx.tables[,3]), ]
for(i in 1:dim(idx.tables)[1]) depth.vec[ind.tab[i,1]:ind.tab[i,2]] <- ind.tab[i,3]
# print(depth.vec)
out <- paste( substring("                                ",1,2*depth.vec+1),sep="",
              substring( gsub(" ","",txt),1,10) )
h <- rep(1,length(txt)); h[grep(" *[<]", invert=TRUE, out)] <- 0
# for(i in 2:length(txt)) if( h[i-1] == 1 ) h[i] <- 2
h[-1] <- ifelse( h[-length(h)] == 1 & h[-1] == 0, h[-length(h)]*2, h[-1])
out <- paste(depth.vec[h>0],sep=":",out[h>0])
print(noquote(cbind(out)))
```

5 Bereitstellung der Objekte beim Start

Damit nach dem Laden des rev-Dokumentes es gleich losgehen kann, werden `get.html.tables()` und das Objekt mit den Beispiel-Urls in einem Start-Chunk definiert.

12 ⟨*start* 12⟩ ≡

```
⟨define get.html.tables() 4⟩
⟨einige Seiten des Servers www.transfermarkt.de 1⟩
```

6 Definition wichtiger Funktionen

Tag-Ermittlung: `find.tag.txt` findet im globalen Objekt `txt` alle Tags. Falls die Tags ohne spitze Klammern der Funktion übergeben wurden, werden diese ergänzt und in ein Anfangssuchmuster verwandelt. Dann werden die Beginn- und End-Tags ermittelt und als zwei elementiger Vektor oder als zwei spaltige Matrix ausgegeben.

13 ⟨*define some functions* 13⟩ ≡ ⊂ 4, 21

```
find.tag.txt <- function(tag.pattern){
  if(substring(tag.pattern,1,1) != "<"){
    tag.pattern <- paste(sep="","^<",tag.pattern)
  } else {
    tag.pattern <- paste(sep="","^", tag.pattern)
  }
  idx.begin <- grep(tag.pattern,txt)
  if( 0 == length(idx.begin) ) idx.begin <- NA
  tag.pattern <- paste(sep="", "^</",substring(tag.pattern,3))
  idx.end <- grep(tag.pattern,txt)
  if( 0 == length(idx.end) ) idx.end <- length(txt)
  if( 1 == length(idx.begin)) c(idx.begin,idx.end) else cbind(idx.begin,idx.end)
}
```

Ermittlung von Klammerpaaren: `find.tag.pairs` findet die Tag-Paare. Erst werden ggf. die spitzen Klammern ergänzt und ein Anfangssuchmuster gebildet. Die Anfänge der Tag-Klammern werden gesucht, dann die End-Tags. Die Klammer-Logik wird auf dem Vektor `idx` per -1 und $+1$ vermerkt. Ein Zählprozess entlang des Vektors erlaubt Tiefen zu ermitteln und Klammer-Paare zu finden. Ausgegeben werden Klammer-Anfänge, -Enden und die Tiefen als dreispaltige Matrix.

14 ⟨*define some functions* 13⟩+ ≡ ⊂ 4, 21

```
find.tag.pairs <- function(tag.pattern,txt){
  # Tags bilden Klammern, bei Verschachtelung ist es interessant, Klammerpaare zu ermitteln
  if(substring(tag.pattern,1,1) != "<"){
    tag.pattern <- paste(sep="","^<",tag.pattern)
  } else {
    tag.pattern <- paste(sep="","^", tag.pattern)
  }
  idx.begin <- grep(tag.pattern,txt); if( 0 == length(idx.begin) ) return(NA)
  tag.pattern <- paste(sep="", "^</",substring(tag.pattern,3))
  idx.end <- grep(tag.pattern,txt)
  idx <- rep(0,length(txt))
  idx[idx.begin] <- 1; idx[idx.end] <- -1
  stack <- NULL; pairs <- NULL
  for(i in 1:length(txt)){
    if(idx[i] == 1) stack <- c(i,stack)
    if(idx[i] == -1) {
      pairs <- rbind(pairs, c(stack[1],i,length(stack)))
      stack <- stack[-1]
    }
  }
  pairs
```

```
} # find.tag.pairs("div",txt)
```

Entfernung aller Tags: `remove.all.tags` entfernt alle Tags aus dem Input-Objekt.

15 ⟨*define some functions* 13⟩+ ≡ ⊂ 4, 21
```
remove.all.tags <- function(x) x[ "<" != substring(x,1,1) ]
```

Entfernung unwichtiger Tags: `remove.not.needed.tags` entfernt alle Tags, die nicht in der Tag-Liste `tag.needed` aufgeführt sind.

16 ⟨*define some functions* 13⟩+ ≡ ⊂ 4, 21
```
remove.not.needed.tags <- function(txt,
    tags.needed=c( "table", "tbody", "th", "thead", "tr", "td",
                   "/table","/tbody","/th","/thead","/tr","/td")){
  if(substring(tags.needed[1],1,1) != "<") tags.needed <- paste(sep="","<",tags.needed)
  tags.needed <- substring(paste(sep="",tags.needed,"............."), 1, 10)
  index <- "<" == substring(txt,1,1)
  # find tag keyword and add some points
  txt.tag <- sub("^([<][/]{0,1}[a-z]+).*","\\1",txt[index])
  txt.tag <- substring(paste(sep="",txt.tag,"............."), 1, 10)
  txt[index] <- txt.tag
  # find matches
  idx <- !is.na(match(txt,tags.needed)) | substring(txt,1,1) != "<"
  # remove not needed tags
  txt <- txt[idx]
  # remove added points
  txt <- sub("([<][^.]+).*[>]{0,1}.*","\\1>",txt) #??
} #;substring(remove.not.needed.tags(txt),1,20)
```

Verschmelzung innerer Tabellen: `paste.table` entfernt alle Tags des Arguments und fügt alles per `paste()` zusammen.

17 ⟨*define some functions* 13⟩+ ≡ ⊂ 4, 21
```
paste.table <- function(txt,sep="/"){
  txt <- remove.all.tags(txt)
  txt <- paste(txt,collapse=sep)
}
```

Extraktion einzelner Tabellen: `extract.table` versucht zunächst einen `thead`-Header in einer Table-Umgebung zu finden und legt ggf. den Kopf auf `head` ab. Dann werden Zeilen mit Hilfe von `tr`-Tags gesucht. In den Zeilen werden alle `td`-Tags entfernt und die Einträge in Vektor-Elemente überführt. Die Vektoren werden als Spalten eines data frame abgelegt. Zum Schluss werden überflüssige Leerzeichen und Punkte in Zahlen entfernt und ggf. Zahlen in Zahlen überführt.

18 ⟨*define some functions* 13⟩+ ≡ ⊂ 4, 21
```
extract.table <- function(txt,conversion.to.numbers=TRUE){
  # extract header defined by thead
  head <- NULL
  idx <- find.tag.pairs("thead",txt)
```

```
if( all(!is.na(idx)) && 0 < length(idx) ){
  cat("head of table found",idx)
  head <- txt[ idx[1]:idx[2] ]; txt <- txt[ -(idx[1]:idx[2]) ]
  head <- remove.all.tags(head)
  # specials
  head <- sub("^ *","",head); head <- sub(" *$","",head)
}
# find lines of table:
# idx <- find.tag.pairs("tr",txt); # doesn't work in cause of dirty html syntax
# if(any(h <- is.na(idx[,1]))){ idx[ h, 1] <- idx[ which(h)-1, 2] + 1 } # open bracket missing
txt <- c("<table","<tr>",txt[-c(1,length(txt))],"<tr>") # add tr-tags at start and end
txt <- sub("</tr>","<tr>",txt)                          # convert end tag to begin tag
idx <- which("<tr>" == txt); idx <- cbind(idx[-length(idx)], idx[-1]-1)
zeilen <- NULL
# idx[i,] defines range of line i of table; loop along the lines of the table
for(i in 1:nrow(idx)){
  zeile <- txt[ idx[i,1]:idx[i,2] ] # get line
  zeile <- remove.not.needed.tags(zeile,c("td","th")) # remove unimportant tags
  zeile <- paste(zeile,collapse=" ") # pack elements of line of table into one element
  # field separation may be done by <th> or <td>
  if(0 < length(grep("[<]td[>]",zeile))) zeile <- unlist(strsplit(zeile,"<td>"))[-1]
  if(0 < length(grep("[<]th[>]",zeile))) zeile <- unlist(strsplit(zeile,"<th>"))[-1]
  # skip empty lines
  if(0 == length(zeile)) next
  # lines with the different length: make them fit
  if( 0 < length(zeilen) && # not in the first iteration
      (n.zn<-ncol(zeilen)) != (n.z<-length(zeile))) {
    if(n.z < n.zn) zeile  <- c(    zeile,  rep(" ",n.zn-n.z))
    if(n.z > n.zn) zeilen <- cbind(zeilen, matrix(" ", nrow(zeilen), n.z - n.zn))
  }
  # add extracted line
  zeilen <- rbind(zeilen, zeile)
}
# remove lines without any text
idx <- apply(zeilen, 1, function(x){ length(x) != length(grep("^ *$",x)) })
zeilen <- zeilen[idx,,drop=FALSE]
if(is.matrix(zeilen)) rownames(zeilen) <- seq(nrow(zeilen))
# return if no transformation to numbers is desired
if(FALSE == conversion.to.numbers){return(zeilen)}
# transform numbers: remove dots from numbers in text style
#zeilen   <- gsub("([0-9]{1})[.]([0-9]{1})","\\1\\2",zeilen)
zeilen <- gsub("[.]([0-9]{3})","\\1",zeilen)
zeilen <- sub(" *$","",zeilen); zeilen <- sub("^ *","",zeilen)
# construct result object
result <- as.data.frame(zeilen); if(is.matrix(result)) rownames(result) <- NULL
if(ncol(result) == length(head) ) colnames(result) <- as.vector(head)
if(is.matrix(result)) rownames(result) <- as.character(seq(result[,1]))
# try to convert numbers to numeric
if(first.lines.to.header)
  result <- make.first.lines.to.header(result,first.lines.to.header)
n.result <- nrow(result)
for(j in 1:ncol(result)){ # print(j)
  res <- as.character(result[,j])
  if( (n.result/2) <= length(grep("^[-]{0,1}[0-9.,]*$",res))){
    h <- sub("[,]",".",res); h <- gsub("[^0-9.-]","",h)
    h[grep("^[-]{0,1}[0-9.]+$",invert=TRUE,h)] <- NA; h[nchar(h)==0] <- NA
    try(if(!all(h == "")) h <- ifelse( h=="", NA, as.numeric(h)))
    res <- h
  }
  result[,j] <- res
} # cat("END of extract"); print(result)
# convert first line(s) to header line
```

```
  result
}
```

Reparatur der ersten Zeile: `make.first.lines.to.header()`: Oft wird eine Html-Tabelle ohne explizitem Kopf erstellt. Dann muss nach der Extraktion noch eine Reparatur erfolgen. Hierfür kann die Funktion `make.first.lines.to.header()` verwendet werden.

19 ⟨*define some functions* 13⟩+ ≡ ⊂ 4, 21

```
make.first.lines.to.header <- function(tab,n.rows=1){
  if(0 == length(dim(tab)) ||  nrow(tab) <= n.rows ) return(tab)
  if(is.data.frame(tab)) for(j in 1:ncol(tab)) tab[,j] <- as.character(tab[,j])
  colnam <- tab[1:n.rows,]; tab <- tab[-(1:n.rows),,drop=FALSE]
  #  cat("colnam"); print(rbind(colnam))
  colnam <- apply(rbind(colnam), 2, paste, collapse="/")
  colnam <- gsub("[/] */","/",colnam); colnam <- gsub("^/","",colnam);
  colnam <- gsub("[/] */","",colnam)
  colnam[colnam==""] <- "-"; colnames(tab) <- as.vector(colnam)
  if(is.matrix(tab) || is.data.frame(tab)) rownames(tab) <- 1:nrow(tab)
  tab
}
```

20 ⟨*test* `make.first.lines.to.header()` 20⟩ ≡

```
⟨define get.html.tables() 4⟩
h <- get.html.tables(
    "http://www.sport1.de/dynamic/datencenter/sport/team-tabelle/fussball/arminia-bielefeld", table.number=1,f
print(h)
```

21 ⟨* 3⟩+ ≡

```
testmfltoh <- function(h){
  ⟨define some functions 13⟩
  tab <- make.first.lines.to.header(h)
  umlaut.repl <- function(txt){
    txt <- gsub("Ãd","ae",txt); txt <- gsub("Ãü","oe",txt);
    txt <- gsub("Ãij","ue",txt); txt <- gsub("ÃÖ","oe",txt);
    txt <- gsub("ÃÜ","Oe",txt); txt <- gsub("ÃU","oe",txt);
    txt <- gsub("Ã§","ss",txt)
  }
  for(j in 1:ncol(tab)) tab[,j] <- umlaut.repl(tab[,j])
  tab
}
testmfltoh(h)
```

22 ⟨* 3⟩+ ≡

```
rm("make.first.lines.to.header")
```

```
http://www.sport1.de/dynamic/datencenter/sport/team-tabelle/fussball/arminia-bielefeld read
body extracted
more than one table or nested tables found
table 1 extracted
Thu Sep 12 19:31:39 2013
    #                   Team Sp. S U N  Tore +/-  P
2   1  SpVgg Greuther Fuerth   6 4 2 0   9:3   6 14
3   2     1. FC Union Berlin   6 3 2 1  11:8   3 11
4   3            1. FC Koeln   6 2 4 0   9:4   5 10
```

```
5   4   1. FC Kaiserslautern   6 3 1 2  9:10  -1 10
6   5          Karlsruher SC   6 2 3 1   6:4   2  9
7   6      TSV 1860 Muenchen   6 3 0 3   5:6  -1  9
8   7         Erzgebirge Aue   6 3 0 3   7:9  -2  9
9   8        Energie Cottbus   6 2 2 2  14:8   6  8
10  9          FSV Frankfurt   6 2 2 2   7:5   2  8
11 10             VfL Bochum   6 2 2 2   9:8   1  8
12 11      Arminia Bielefeld   6 2 2 2 11:11   0  8
13 12           FC St. Pauli   6 2 2 2   7:7   0  8
14 13              VfR Aalen   6 2 2 2   7:8  -1  8
15 14    Fortuna Duesseldorf   6 2 1 3   7:9  -2  7
16 15          SV Sandhausen   6 1 3 2   5:6  -1  6
17 16        SC Paderborn 07   6 1 2 3  7:13  -6  5
18 17       FC Ingolstadt 04   6 1 1 4  6:11  -5  4
19 18         Dynamo Dresden   6 0 3 3  5:11  -6  3
```

7 R-Datei-Erstellung

23 ⟨* 3⟩+ ≡

```
⟨define get.html.tables() 4⟩
dump("get.html.tables",file="get-html-tables.R")
```

8 Definition von `html-tags`

24 ⟨*start* 12⟩+ ≡

```
html.tags <- paste(c(
   "<!--> <!DOCTYPE> <a> <abbr> <acronym> <address> <applet> <area> <article> <aside> <audio>",
   "<b> <base> <basefont> <bdi> <bdo> <big> <blockquote> <body> <br> <button> <canvas>",
   "<caption> <center> <cite> <code> <col> <colgroup> <command> <datalist> <dd> <del>",
   "<details> <dfn> <dialog> <dir> <div> <dl> <dt> <em> <embed> <fieldset> <figcaption>",
   "<figure> <font> <footer> <form> <frame> <frameset> <head> <header> <h1> <h2> <h3> <h4>",
   "<h5> <h6> <hr> <html> <i> <iframe> <img> <input> <ins> <kbd> <keygen> <label> <legend>",
   "<li> <link> <map> <mark> <menu> <meta> <meter> <nav> <noframes> <noscript> <object> <ol>",
   "<optgroup> <option> <output> <p> <param> <pre> <progress> <q> <rp> <rt> <ruby> <s> <samp>",
   "<script> <section> <select> <small> <source> <span> <strike> <strong> <style> <sub>",
   "<summary> <sup> <table> <tbody> <td> <textarea> <tfoot> <th> <thead> <time> <title> <tr>",
   "<track> <tt> <u> <ul> <var> <video> <wbr>"))
html.tags <- unlist(strsplit(html.tags, " "))
html.tags <- substring(html.tags,2,nchar(html.tags)-1)
```

9 Anhang

Object Index

Code Chunk Index