HTML TABLE EXTRACTOR

OVERVIEW

The goal is to design a tool to extract data from tables in HTML files.

If the tables were well-defined, then the task is much simpler. An example input file:

```
 <table class="details" border="0" cellpadding="5" cellspacing="2" width="95%">
   <tr valign="top">
     <th>Cash on Hand</th>
     <th>Deposits</th>
     <th>Earnings per share</th>
     <th>Total shares</th>
   </tr>
   <tr valign="top" class="Failure">
     <td>103</td>
     <td>24</td>
     <td>3.5</td>
     <td>71</td>
  </tr>
</table>
```

The above table will be very easy to parse and extract data from the table. However in the SEC files, the tables are more complicated, the style used to display ie, the formatting is mixed with the content (data), thereby making it very difficult to write a generic tool. Also tables are used as formatting aids and thus every table need not necessarily contain tabular data as in numbers. For example, tables are used to format the 'Signatories' at the end of the document. Presumably this data is not of interest to the system under consideration.

Example of available tools that will work for simple extraction:

A python library for extracting data from html table

https://pypi.python.org/pypi/html-table-extractor/1.3.0

PROBLEM STATEMENT:

It is desirable to write a universal tool that will handle all kinds of tables; however that would be very ambitious. The aim of the tool being discussed here is to fit into the overall architecture of analyzing financial data from SEC and other such sites. But it would be tedious to write specific code for every table of interest in all of these files.

So the scope could be redefined to writing a tool to extract tables of interest from financial documents without too much specificity. This will be a tradeoff between time and effort and achieve the goal in a short amount of time.

By reducing the scope, a systems analyst could assist in providing hints to the tools as allowed by its configuration files. I will take this approach and explain it further in the document.

CHALLENGES:

The challenges could be broken into

I. SYNTACTIC CHALLENGES

Questions here are:

- how to parse tags
- how to handle ill-formed documents
- are all the tags closed or not?
- Should we read the whole document into memory and then parse it further based on requirements?
- or could it be parsed like a sax parser and the tags and data identified as one reads through the document?

The examples given in this document are in python, but it could equally apply to java as well.

APPROACHES TO PARSE:

1. Read the whole document into memory by using a parser and a wrapper around it.

Example 1: Use lxml parser ot html.parse in python and use BeautifulSoup (a MIT licensed tool) to provide ease of access to the tags and the data within those tags.

Example 2: Use BeautifulSoup and Pandas to read a table into a DataFrame in Pandas.

These approaches work when the Table is well defined as shown in the simple example above.

In the present architecture, I presume that the system will be used to parse thousands of documents on any given day - there are company reports, mutual funds reports etc and this TableExtractor is a small part of the whole system. Also the degree of control provided by these tools is at a higher level than what is required to extract data in complex documents like the ones we have to deal with.

2. Write a parser from scratch either using a parser generator tool (like yacc or PLY) or use regular expressions on the raw file to parse HTML tags.

This offers higher degree of flexibility but is more time consuming and may not be required for our purposes.

3. [Recommended] Use a basic parser to parse HTML tags and the extracted data to build the semantic connections.

In this approach, we use a basic parser and then use either regular expressions or other methods to extract the data of interest to us. This would also be much faster as it is more or less a one pass approach with minor peeps into the previous data stored to build the semantic connections.

The recommendation is to use a basic HTML parser that will produce callbacks as it encounters each tag of interest. An example of this is the python3 HTMLParser class in html.parser. We could subclass the HTML parser class and override the startTag, endTag and data methods to implement the behaviour that we need.

The idea is to ignore all formatting and style elements and look only at the data to build connections.

Style elements and formatting like alignment, font sizes, indent spacing could vary widely across documents and also across time in the same document type - so these would be very unreliable markers in extracting semantic connections.

The tags of interest to us in the SEC documents are Table related tags like 'table','tbody','th','tr','td' for table data.

The 'font' and 'br' tag are used often to specify the text - so we should include them as well. The 'div' tag is not needed.

The tag 'type' is needed to identify the type of the SEC document (10K, 10Q etc).

Example snippet:

```
start tag: td
start tag: font
data: July 1,
start tag: br
data: 2017
end tag: font
end tag: td
```

Specifically the 'br' tag is used to format multi column names. In this example, the br tag is nested between font tags. And the font tags are nested between td tags. The column text extracted should be 'July 1, 2017' as a concatenation of 'July 1' and '2017' and will be used as the column value.

II. SEMANTIC CHALLENGES

Semantically associating related fields and values is the main challenge in these documents. SEC has only specified a guideline of what items need to be in the report be it 10Q, the quarterly report or 10K, the annual report. We will assume that this is the approach of SEC for other documents types too. The intention is to allow a human reader to find the mandatory information that these companies have to provide, but the format is not necessarily friendly for machine readability. So there is a wide variation in formatting, the order of presentation etc.

Here we will list the main semantic information to be linked to the parsed document.

We will group the information under Table information and Non-Table information. Non-Table information does not appear in a table but is very essential to place and identify the Table information that will be processed.

1. NON-TABLE INFORMATION

All the Non-Table information could be extracted from the first page of the SEC document. Though there is no strict pagination followed in the parsing of the document, it could be heuristically assumed that anything that occurs before the occurrence of 'Table of Contents' (case-insensitive compare) could be considered to be the Header information of the document.

Accordingly, the following could be extracted from the Header.

a. The originating document type if there is a type

Look for the Tag 'type' and get the data value from it.

Example:

Tag - 'type'
value - 10-Q

b. The company name / ID associated with the table

Look for the text "Exact name of Registrant" and get the previous data before this tag.

Example snippet:
start tag: font
data: Apple Inc.
end tag: font
start tag: font
data: (Exact name of Registrant as specified in its charter)
end tag: font

This should produce 'Apple Inc.'

For ID number, search for 'Employer Identification' in the data collected. This refers to the I.R.S. Employer identification No. Go back in the data collected till you find a number (the number may have a '-' in it). This is the ID of the company. This data may or may not be part of a table.

This should produce ‘94-2404110’.

Example snippet:

start tag: td
start tag: font
data: 94-2404110
end tag: font
end tag: td
end tag: tr
start tag: tr
start tag: td
start tag: font
data: (State or other jurisdiction
end tag: font
start tag: font
data: of incorporation or organization)
end tag: font

```
end tag: td
start tag: td
start tag: font
data:
end tag: font
end tag: td
start tag: td
start tag: font
data: (I.R.S. Employer Identification No.)
end tag: font
end tag: td
end tag: tr
```

c. The date of the document from the index

Look for the text 'quarterly period ended' or 'fiscal year ended' or 'year ended' or 'transition period from' and then get the following data from the tags.

Example snippet:

```
start tag: font
data: QUARTERLY REPORT PURSUANT TO SECTION 13 OR 15(d) OF THE SECURITIES EXCHANGE
ACT OF 1934
end tag: font
end tag: div
start tag: font
data: For the quarterly period ended
end tag: font
start tag: font
data: July 1, 2017
end tag: font
```

If it is period ended or year ended then only endDate will be available, else startDate and EndDate will be available.

So either one could calculate the startDate based on whether it is quarterly or yearly and fill it in the startDate or leave it blank. This will depend on what this data will be used for later in the system.

This should produce July 1, 2017 and the year could be extracted from that if necessary.

2. TABLE INFORMATION

Table information is present inside 'table' tags. Within this tag the other tags may not appear as expected. The tag 'th' may not be used to specify column headers. The number of 'td' tags may not be the same under each row identified by the 'tr' tag. So a general heuristic has to be used to place them together.

The parser will group all information under the Table tags as a NamedRecord (explained later). The semantic routine will operate on these NamedRecord datatypes and align the required information.

The datavalue for a column should be determined as follows:

```
If the program is within a 'tr' tag
  if it is within a 'td' tag
     datavalue could be here
     if a 'font' tag
         datavalue could be here
     optional 'font' end tag
  'td' end tag
```

a. The name of the table from any headers above the table or outside table

The name of the table is usually found above the table.

When a 'table' tag is found, go backwards in the identified tags till you find data with 'Note' or 'Item' or any item from the [NamedRecord tags] specified in the config file - if nothing is specified, then 'Note' or 'Item' will do.

THe [NamedRecord tags] should contain tables of interest.

Examples are:

-Comprehensive Income

-ShareRepurchase Program

-Shareholders Equity

-Balance Sheets

-Statements of Income

When I say look, it means do a case insensitive string match or a 're' match with key words or a string similarity score of number of words that match between the NamedRecord tag and the data in the document.

This will be the name of the Table. In the example snippet below, it should produce Comprehensive Income from the data 'Note 8 – Comprehensive Income'.

b. The column header including any information for multiple headers

As illustrated above with this example snippet, this will produce 'July 1, 2017'

Example snippet:

```
start tag: td
start tag: font
data: July 1,
start tag: br
data: 2017
end tag: font
end tag: td
```

c. The date associated with the number from the header eg 2001

This has already been found from the "header information'.

d. Any ancillary information such as if it is a pro forma number or not

Look for words like 'Unaudited', 'Pro forma' in text data etc to check whether it is pro forma. This means that this table is pro forma.

e. The units for the number e.g. dollars, shares

Look for words like 'millions' or 'shares' or 'thousands'. The closest proximity to the 'millions' or 'thousands' to the word shares/securities will be the unit for that. If no proximate word is found, then default Amount (identified when a '$' is encountered in the Table to millions and thousands if '$' not encountered.

If '$' is encountered, then that column is 'money' field.

Decide on [UnitDescriptor] as one of ['money','number','weight','volume']. This will be an additional filed maintained in the [NamedRecord].

Remove all ',' from a number. If it has a decimal point, keep that and do not multiply with the million or thousand as the case may be. Decimal points only occur in EarnigsPerShare figures.

Also look for words like '(gains)/losses' in the column text. If there is a '(' and ')' surrounding a word like 'gain' or 'loss' then the corresponding number (surrounded by brackets or not) should be multiplied by '+" or '-'.

If 'gains' is surrounded by brackets then it is a positive number, else a negative number.

Example snippet:

```
start tag: font
data: Note 8 – Comprehensive Income
end tag: font
start tag: font
data: Comprehensive income consists of two components, net income and OCI. OCI refers to
revenue, expenses, and gains and losses that under GAAP are recorded as an element of
shareholders’ equity but are excluded from net income. The Company’s OCI consists of foreign
currency translation adjustments from those subsidiaries not using the U.S. dollar as their functional
currency, net deferred gains and losses on certain derivative instruments accounted for as cash flow
hedges and unrealized gains and losses on marketable securities classified as available-for-sale.
end tag: font
start tag: br
start tag: font
data: Apple Inc. | Q3 2017 Form 10-Q |
end tag: font
start tag: font
data: 15
```

```
end tag: font
start tag: br
start tag: font
data: The following table shows the pre-tax amounts reclassified from AOCI into the Condensed
Consolidated Statements of Operations, and the associated financial statement line item, for
end tag: font
start tag: font
data: the three- and nine-month periods ended July 1, 2017 and June 25, 2016
end tag: font
start tag: font
data:  (in millions):
end tag: font
start tag: table
```

f. The row name including sub-row names aligned to the tree

This relates to the text in the column that is a line descriptor/line item. This is a tricky problem and is addressed below:

Reference: Note 8 - Comprehensive Income in Apple 10Q.

-Extract a row from the table (series of 'td' tags under the 'tr' tag)

Note: If a row is completely empty, remove that row.

-Line item in the row is 1st or second value in the row (most likely 1st).

-if this row does not contain any other items, designate this as a [SUBHEADING].

Note: A SUBHEADING is one that will be propagated to the rows below it so the new rowname will get appended to this one.

Example

'Unrealized (gains)/losses on derivative instruments' does not have any item in that row - so SUBHEADING

Next line 'Foreign exchange contracts' will be appended to nake the new value

'<Unrealized (gains)/losses on derivative instruments><Foreign exchange contracts>'

Separator could be a dot - but the datavalue could itself have a dot, so I suggest <> as it is easy to read. Of course, only one symbol < or > could be used but using the pair <> makes it easy to read. However if it is easier for other program down the chain to use '.' as a separator to possibly subscribe to these values then one could always use '.' as a separator.

-This SUBHEADING will be propagated to other line items or row till we encounter another SUBHEADING

OR

a '---------------------' in a line item

OR

after a TOTAL line.

A TOTAL line is identified by the word 'Total' or 'Balance' or

A TOTAL line contains no lime descriptions but only numbers.

- keep building up the sub-row names and use the latest SUBHEADING name till the end of the table.

STRUCTURE OF SEC FINANCIAL DOCUMENTS:

Form 10Q and 10K

These are guidelines for the companies on what information should be filed with the SEC. The only fixed information to be provided is in the cover page (I call it the HEADER) that relates to company name, dates involved, ID number among other information.

The rest of the document consists of optional sections (I call these the 'NamedRecords') as applicable to the company.

I assume that other documents like SEC Form 13 relating to Fund Managers may have similar recommendations. In general, documents appearing in HTML format are likely to be ill-structured as they are likely to be recommendations whether they are SEC documents or documents from other institutions.

Resources checked

1. The Annual Report Algorithm: Retrieval of Financial Statements and Extraction of Textual Information

Link: Hering, Jörg, The Annual Report Algorithm: Retrieval of Financial Statements and Extraction of Textual Information (November 16, 2016). Available at SSRN: https://ssrn.com/abstract=2870309 or http://dx.doi.org/10.2139/ssrn.2870309

Comment:

Relating to Form 10K (and hence 10Q as the structures are very similar), this could be referenced for ideas to extract textual information and may provide insights into extracting Tables of interest for the system in this contest.

2. Competitive Analysis of Business Filings Using Ontologies and Linguistic Analysis

Published in:

Conference: Joint SIGDSS & TUN Business Intelligence Congress 3: Driving Innovation through Big Data Analytics 2012

Comment:

Talks about the XBRL (busuness markup language) and is more of a framework than any specific detail on extracting Tables of interest.

3. Automatic extraction and analysis of financial data from the EDGAR database

Link: https://www.sajim.co.za/index.php/SAJIM/article/viewFile/127/124

Comment:

Published in 2001 in the South African Journal of Information Management - the structure of SEC filings then had very less markups and tags and this is likely to be outdated.

4. XBRL and SEC

Link: https://www.sec.gov/structureddata/osd-inline-xbrl.html

Link: https://www.sec.gov/page/osd-homepage

I also suppose that the system designers have looked at the XBRL intitatives of SEC and have decided that it is more useful to extract data from HTML tables. As of now, it is voluntary and not mandatory to publish data in XBRL format.

The following link

https://www.tagnifi.com/dont-drink-the-pond-water/

seems to indicate that the HTML data is more reliable than XBRL data.

5. Companies working in automatic extraction of financial data

Link: http://starmine.ai/

There seem to be companies working in this area.

PROGRAM:

Points to consider for the program:

-The idea is to compare equivalent reports from different companies in the system, be it Balance Sheets, Income Statements or hares or products sold. So a title (or a NamedRecord) to identify a Table will be useful.

-An ability to Extract Table by DocType (ex: 10K) or a helper program by Doc Type will be useful.

-An ability to specify sections of interest (NamedRecord) is useful. The system may not be interested in comparing signing authorities - so need not spend effort in extracting that table both at run time and at design time.

-Needs to have a work area (internal data structure) where all tags and text associated with tags is stored.

-Remove all formatting in the internal Data structure.

-An ability to specify an OutputProducer program. An OutputProducer Program will use the internal Data structure and output the tbales of interest in the required JSON format for further consumption by other downstream programs.

The internal Data structure will be of the form:

```
REC
[CompanyName
 CompanyID
 DateFrom, DateTo
 DocType
 [SectionName
   [Section Records]
 ]
 [SectionName
   [SectionRecords]
 ]
 .....
]
```

- Applications are likely to subscribe to Tables identified by CompanyName/CompanyID, DocType, DateFrom and DateTo, SectionName (IncomeStatement, BalanceSheet).

So the OutputProducer may have to include these items in its JSON output structure.

Ex:

```
1. [CompanyName,
 DocType,
 DateFrom,DateTo,
 [balanceSheet
  List/Set of [BalalnceSheet rows]
 ]
]
 2.  [CompanyName,
 DocType,
 DateFrom,DateTo,
 [IncomeStatement
  List/Set of [IncomeStatement rows]
 ]
]
```

- Each company operates in different geographies, deals with different products and so have different line items. So NLP is limited to identifying weights, number, money for the units. It is difficult to compare directly across companies programmatically. Apple may call number of IPad sales as product sales while Alphabet(Google) when it refers to its product sales may mean entirely different things. So one could only compare EarningsPerShare, TotalSales etc on JSON tables like BalanceSheet of both or IncomeStatements of both. Having a JSON line item of its own like {"productsales.july17": 2000} may not make sense without the context of the company name, TableName etc.

Sample Program:

A sample program exttable.py written in python3 is listed in the directory. This is used to study the tags in the SEC document.

PROGRAM pesudocode:

Config File: htmlprop.properties (present in this directory)

This config file defines specific functions/data (hints to the system) that are input to the extractHTML Table function. The properties are described in the config file.

Function extractHTMLTable:

```
Description: This reads the files from its storage area into a list and passes it to the extHTML function
  Each extHTML runs in its own thread, receives a list of HTML lines and outputs a JSON structure.
Input: Reads the HTML files of interest
      Reads the config file(s) of interest
      Config files could be one per DocType (10Q,10K), same for multiple types (to be defined separately)
      A new type of file (may not be from SEC) may need completely different hints for extraction and output.
Logic:
  Read Htmlfiles from the directory or from its storage
  Read ConfigFile(s) and build a ConfigFile Lookup Table
  Instantiate the functions/data in the config file and provide a configFileFunctionGetter function to the extHTML.
  The configFileFunctionGetter function will return the function/data of interest to the caller- extHTML
  This will include both the TablesOfInterest functions, OutputSection functions and other data.
  Setup and manage the distribution of HTML files to extHTML
  (This could be in Spark or any other method to setup space for the return JSONs and other communication with  further systems like Kafka or any other system.
  Call the extHTML with an HTML file and configFileFunctionGetter (function pointer)
  Gather results and end
```

Function extHTML(List htmlLines,configFile table,configFileFunctionGetter):

```
Input: Defined above
Logic:
  Read the HEADER information from the file.
  A HEADER is defined as the beginning section of the HTML page till it reaches he Table of Contents.
  Build the internal data structure with the information extracted from the HEADER.
  Read the rest of the HTML file and gather sections (NamedRecords)
  These NamedRecords may be specified in the config file table.
  If NamedRecords specified,
    gather them by calling the specified handler through the ConfigFileFunctionGetter
    If no NamedRecordhandler specified,
      use DefaultHandler (Default logic discussed in the syntactic and semantic challenges)
    ELSE - no NamedRecordsSpecified
      use DefaultHandler - extractTable (use heuristics to decide on TableNames and tables as
discussed in the sections above).
  At this point the internal DataStructures should be filled up.
  Call the OutputSection handlers specified in the config file through the configFileFunctionGetter
  If not specified,
    use Default output function
  return output
```

Function extractTable(HTMLlines):

```
Input: lines of HTML
Logic:
  Start with the Table tag.
  Read previous lines to identify table name (Section name/NamedRecord)
  Read the next lines.
  Ignore empty rows
  if 'th' tags are specified then table headers are in this row.
  ELSE
  Identify/Guess Table Headers
    (Usually the first line in a Table OR
     the line before the row with a number OR
     if the line before the row with the number is a SUBHEADING, then the row before that).
  Read the tr tags and collect data from the td tags.
  The end of the table tag closes the table.
```

Data Structures.

Maintain internal structures - These could be in JSON fpormat or native format of the programming language.

For example I could maintain these records as lists in python and convert them to JSON through the Output function.

1. NamedRecord
Attributes:
Name: 'Comprehensive Income'
DateFrom:
DateTo: July 1,2017
set of [TableRecords]

2. TableRecords
Attributes:
Set of [Rows]
3. Rows

Set of [Colname, colvalue, unitDesc].

Example Output:

For the example given in the forum.
JSON output format for BalanceSheet

```
{
    "CompanyName": "Apple",
    "SectionName":"BalanceSheet",
    "DateTo": July 1, 2017,
    "DocType": "10Q",
    "Rows": [{"Assets":"Current Assets.Cash and cash equivalents",
             "units": "money",
            "July 1, 2017": 18571,
             "units": "money",
           "September 24, 2016": 20484},
             {"Assets": "Current Assets.Short-term marketable securities",
           "units": "money",
           "July 1, 2017": 58188,
           "units": "money",
           "September 24,2016": 46671},
          ...
         ]
}
```