Detect HTML links

The problem is categorized in the regex domain, so we'll start with a regular expression solution. Regexes are quite powerful, but they get very complicated very quickly. Instead of trying to completely solve the problem with a single regex, we use two. The first finds the <a> elements, and the second gets the text inside the element.

```
import re

def detect_links(html):
    links = re.findall(r'<\s*a [^>]*href="([^"]*)"[^>]*>(.*?)</a', html)
    return ((href.strip(), re.sub('<[^>]*>', '', title.strip())) for href,

if __name__ == '__main__':
    import sys
    for link, text in detect_links(sys.stdin.read()):
        print('{},{}'.format(link, text))
```

The first regex matches:

`<\s*`
: the open angle bracket followed by any amount of white space.

`a [^>]*`
: an "a", a space, and any number of characters that aren't the close angle bracket.

`href="([^"]*)"`
: the string "href=" followed by any amount of text in double quotes. The paratheses indicate that this text is a capture group.

`[^>]*>`
: any text that isn't a close angle bracket, followed by a close angle bracket.

`(.*?)`
: any amount of text, in a capture group. The use of `.*?` indicates this is a lazy match (more below).

`</a`
: the close tag.

The `*` operator is greedy, matching as much text as possible. If we had used it in the second capture group, it would have matched all the text from the end of the first <a> tag to

the beginning of the last tag. We want it to match only up to the first tag, so we use the lazy version, *?, which matches as little as possible.

When used with capture groups, the findall() function returns a list of tuples containing the matches to those groups. Thus, we get a list of (href, title) tuples. The title may contain other tags, so the second regex strips out everything inside html tags.

“HTML and regex go together like love, marriage, and ritual infanticide.”

That nugget of the wisdom is offered in this Stack Overflow answer. Read it for the horrific details, but the basic summary is that HTML is not a regular language, and therefore cannot be completely parsed by a regex. Some well-behaved subsets can be, but any time you are faced with arbitrary HTML content, you want a more powerful tool.

Luckily, such tools are readily available in XML parsers. Beautiful Soup is particularly forgiving of the malformed HTML out there in the wild:

```
from bs4 import BeautifulSoup
from xml.sax.saxutils import escape

def detect_links(html):
    soup = BeautifulSoup(html, "html")
    links = soup.find_all('a')
    return ((escape(link.get('href')), link.text.strip()) for link in links

if __name__ == '__main__':
    import sys
    for link, text in detect_links(sys.stdin.read()):
        print('{},{}'.format(link, text))
```

Not only will this handle various edge cases better, but it’s signficantly easier to read.

(It is worth noting that we need the call to escape() because Beautiful Soup does the right thing and unescapes HTML entities, converting < to <, etc. This is always what you want to do in real life, but was apparently judged too difficult to include in this regex challenge. Thus, we undo the right thing and re-escape those characters that need it.)