1. 请找到一个英文的 tokenization 工具，分析其代码中如何处理这些问题？

由于英文中每一个单词都使用空格分开，因此英文 tokenization 比中文分词简单很多，而且分词产生歧义的情况比起中文来说也少了很多，但是在实际使用过程中仍然存在歧义情况。以下以 nltk 为例说明在英文 tokenization 过程中如何解决歧义问题。

首先分析 nltk 在正常情况下如何进行 tokenization：主要使用的就是空格作为分隔符，将一些标点如.等作为句子的分隔符进行分割，同时可以将一些标点符号作为句子中一部分的分割符，如“”等标点符号。在分割过程中实际使用的方式是正则表达式的方式来分割，使用正则表达式的方式还可以做到将分词中可能和单词在一起的标点符号全部去掉，这一部分的部分源代码展示如下：

```
STARTING_QUOTES = [
    (re.compile("([«“‘„]|[`]+)", re.U), r" \1 "),
    (re.compile(r"^\""), r"``"),
    (re.compile(r"(``)"), r" \1 "),
    (re.compile(r"([ \(\[{<])(\"|\'{2})"), r"\1 `` "),
    (re.compile(r"(?i)(\')(?!re|ve|ll|m|t|s|d|n)(\w)\b", re.U), r"\1 \2"),
]
```

英文 tokenization 中的歧义问题主要集中在对于一些缩写词的时候，例如如果句子中出现“i.e.”的时候可能出现分词错误，错误示例如下所示：

```
1. from nltk.tokenize import sent_tokenize
2. document = ''
3. sentences = sent_tokenize(document)
4. print(sent_tokenize('fight among communists and anarchists '
5.                     '(i.e. at a series of events named May Days).'))
```

tokenization 结果为：['fight among communists and anarchists (i.e.', 'at a series of events named May Days).'] 很显然“i.e.”被作为分句标志分开了，但是这很显然是不符合认知的。对代码做如下修改，tokenization 结果就是正确的：

```
1. from nltk.tokenize.punkt import PunktSentenceTokenizer, PunktParameters
2. punkt_param = PunktParameters()
3. abbreviation = ['i.e']
4. punkt_param.abbrev_types = set(abbreviation)
5. tokenizer = PunktSentenceTokenizer(punkt_param)
6. tokenizer.tokenize('fight among communists and anarchists (i.e. at a series
   of events named May Days).')
```

而从源代码来看，使用方法如下：直接向词典中加入这些特有的词，当遇到这些特有词，比如一些缩写的时候不将其视为句尾而是将其视为一个单词，这些特有的缩写词有一部分是 nltk 内置的，如果在实际使用过程中向这个特有词的词典中加入一些在 tokenization 中需要用到的特定的词，当遇到的时候，就不作为句尾处理，详细的处理源代码如下：

```
1. for typ in types:
2.             # Check some basic conditions, to rule out words that are
3.             # clearly not abbrev_types.
```

```
            if not _re_non_punct.search(typ) or typ == "##number##":
                continue

            if typ.endswith("."):
                if typ in self._params.abbrev_types:
                    continue
                typ = typ[:-1]
                is_add = True
            else:
                if typ not in self._params.abbrev_types:
                    continue
                is_add = False
```

2. 从最长匹配到最大频率分词，体现了什么工程实践中的普遍规律？

体现了工程实践中从实际的结果出发，选用较为简单而且效果较好的方式来实现某个功能的普遍规律。对于最长匹配来说，最初使用的时候可能并没有在数学上严格证明其合理性，最大频率分词同理，都是基于经验提出的一些理论，而这些理论能够在大多数测试结果中表现出较好的结果，因此在工程实践中决定采用这些实现方式。

1. 站在工程技术高度，分词/tokenization 于 NLP 的意义是什么

首先，对于 NLP 任务来说，在大部分时候如果能切断上下文耦合，降低词序的影响，那么能够使用的模型就会多很多。例如很多算法都假设特征之间都是相互独立的，例如 SVM 等，因此在实际操作的时候如果不分词，使用单字作为输入，容易导致很多单子之间是有比较强的联系的，那么“特征之间相互独立”的假设比较难成立，分类结果可能略差。即使是目前看来不需要分词的 CNN 等算法，实际也是使用卷积操作将文本的部分进行一定的组合，其实和分词目的类似。

其次，单针对中文来说，一个字在不同的词内可能有不同的意思，分词之后就减少了识别不同词内单字语义不同的计算代价，分词缓解了“一字多义”的问题。

再次，同样是中文，存在词缀问题，一个单字可能没有具体意义，只在词中才有含义，例如“者”在现代汉语中其实意义不大，但是如果是“作者”等词中，“者”就带有某一类人的含义。

2. 可否证明最长匹配分词的合理性？

如果将分词问题视为一个最优路径上的问题，最长匹配分词事实上就是每一步都采用贪心策略。我们希望我们分出来的每一个词都是正确的，也可以理解为在分词不发生错误的前提下，一个词尽量能多储存一些信息。例如“哈尔滨工业大学”比起“哈尔滨/工业大学”这种分法来说，前一种分法单个词携带了更多的信息。而对于中文来说，单个字能表示的信息量大部分是相同的，也就是说使用最大匹配分词的时候就是在每一步寻找携带最大信息量的分词方式，因此最大匹配分词的合理性转化为贪心算法的合理性，而对于贪心算法来说，虽然无法保证达到全局最优点，但是基本可以到达局部最优，这就已经达到我们分词的要求了。也就是我们分词方法的合理性并不要求达到全局最优（正确率 100%），只需要性能满足要求即可。