Lecture 1 超基础复习笔记 - 中文分词

零、先理解问题（非常重要！）

中文和英文的区别

英文句子：

```
I love Python
```

单词之间有**空格**，电脑很容易知道有3个词。

中文句子：

```
我爱编程
```

没有空格！电脑不知道这是1个词还是3个词。

我们的任务

把 `我爱编程` 变成 `我 爱 编程`（加空格分开）

一、算法的基本思路（用人话说）

想象你在读一本中文书，但是所有字都连在一起：

```
今天天气很好我们去公园
```

你会怎么断句?

人类的方法: 1. 从头开始看 2. 试着找"最长的词" 3. 找到了就继续往后看

例子:

```
今天天气很好我们去公园
↓
"今天天" 不是词 ❌
"今天" 是词 ✅  → 输出"今天"
↓
继续从"天"开始
"天气很好我" 不是词 ❌
"天气很好" 不是词 ❌
"天气很" 不是词 ❌
"天气" 是词 ✅  → 输出"天气"
↓
继续...
```

这就是**贪婪匹配算法**!

二、算法详细步骤(图解)

例子:分词 "中文句子"

假设词典里有: - 中文(2个字) - 句子(2个字) - 中(1个字) - 文(1个字) - 句(1个字) - 子(1个字)

最大词长 = 5(大部分中文词不超过5个字)

第1步:从开头"中"开始

```
位置:  0  1  2  3
句子:  中 文 句 子
       ↑
    从这里开始
```

尝试匹配:

> 尝试5个字：中文句子？ → 只剩4个字，最多试4个
> 尝试4个字：中文句子 → 词典里没有 ❌
> 尝试3个字：中文句 → 词典里没有 ❌
> 尝试2个字：中文 → 词典里有！✅

结果：输出"中文"，位置跳到索引2

第2步：从"句"开始

```
位置：  0  1  2  3
句子：  中 文 句 子
           ↑
        从这里开始
```

尝试匹配：

> 剩余2个字，最多试2个
> 尝试2个字：句子 → 词典里有！✅

结果：输出"句子"，位置跳到索引4，结束！

最终输出： `中文 句子`

三、代码超详细讲解（一行一行解释）

先看整体结构

```
def segment1(sentence, wordset):  # 函数名：segment1，输入是句子和词典
    words = []                     # 创建一个空列表，用来存放分好的词
    sentlen = len(sentence)        # 句子有多少个字？例如"中文句子"是4
    current = 0                    # 当前处理到第几个字？从0开始

    while current < sentlen:       # 只要还没处理完整个句子
        # ... 在这里做分词

    return words                   # 返回分好的词列表
```

第一部分：计算能尝试的最大长度

```
maxlen = min(sentlen - current, MAXWORDLEN)
```

详细解释：

`MAXWORDLEN = 5` （这是常量，在文件开头定义的）

假设： - 句子总长 `sentlen = 10` - 当前位置 `current = 2` - 剩余字符 = 10 - 2 = 8个字

那么 `maxlen = min(8, 5) = 5`

为什么要用 `min`？

情况1：中间位置

```
句子：一二三四五六七八九十
位置：0 1 2 3 4 5 6 7 8 9
          ↑ current=2
剩余：8个字
→ min(8, 5) = 5，可以试5个字长度
```

情况2：快到结尾

```
句子：一二三四五六七八九十
位置：0 1 2 3 4 5 6 7 8 9
                    ↑ current=7
剩余：3个字
→ min(3, 5) = 3，只能试3个字长度
```

如果不用min会怎样？

```
candidate = sentence[7:7+5]  # 想要取5个字
# 但只剩3个字！会越界出错
```

第二部分：从长到短尝试匹配

```
for i in range(maxlen, 0, -1):
```

`range(maxlen, 0, -1)` 是什么意思？

假设 `maxlen = 5`：

```
range(5, 0, -1)  →  生成：5, 4, 3, 2, 1
```

- 第1个参数（5）：从哪里开始
- 第2个参数（0）：到哪里结束（不包括0）
- 第3个参数（-1）：每次减1（倒着数）

为什么要倒着数？

因为我们要**先试最长的词**！

```
先试5个字：中文句子由
再试4个字：中文句子
再试3个字：中文句
再试2个字：中文 ✅ 找到了！
（不用再试1个字了）
```

第三部分：提取候选词

```
candidate = sentence[current:current+i]
```

字符串切片详解：

`sentence[起始位置:结束位置]`

例子：

```
sentence = "中文句子"
# 索引:     0 1 2 3

sentence[0:2]  →  "中文"   # 从索引0到2（不包括2）
sentence[1:3]  →  "文句"   # 从索引1到3（不包括3）
sentence[2:4]  →  "句子"   # 从索引2到4（不包括4）
```

在循环中：

```
current = 0, i = 2
candidate = sentence[0:0+2] = sentence[0:2] = "中文"

current = 2, i = 2
candidate = sentence[2:2+2] = sentence[2:4] = "句子"
```

第四部分：核心判断（最重要！）

```
if i == 1 or candidate in wordset:
    words.append(candidate)
    current += i
    break
```

逐句解释：

`if i == 1 or candidate in wordset:`

这是**两个条件**，满足**任意一个**就执行：

条件1： `i == 1` - 意思：已经试到只剩1个字了 - 为什么要这个条件？

想象这个情况：

```
句子：我爱♥编程  # ♥是个奇怪符号，不在词典里
词典：我，爱，编程

处理到♥时：
试5个字：不在词典
试4个字：不在词典
试3个字：不在词典
试2个字：不在词典
试1个字：还是不在词典！但我们必须处理它！
```

如果没有 `i == 1` 这个条件： - 程序会一直卡在♥这里，永远循环下去！

有了 `i == 1` 这个条件： - 即使不在词典，单个字也强制接受，继续往下走

条件2： `candidate in wordset` - 意思：候选词在词典里找到了 - `in` 是Python的关键字，检查某个东西是否在集合/列表里

```
wordset = {"中文", "句子", "我"}
"中文" in wordset  →  True  ✅
"中文句" in wordset  →  False  ❌
```

`words.append(candidate)`

把找到的词**添加**到结果列表里：

```
words = []                  # 一开始是空的
words.append("中文")        # → words = ["中文"]
words.append("句子")        # → words = ["中文", "句子"]
```

`current += i`

把位置向前移动 `i` 个字符：

```
current = 0
i = 2  # 找到了2个字的词"中文"
current += i  # 等同于 current = current + i = 0 + 2 = 2
```

图解：

```
位置:  0  1  2  3
句子:  中 文 句 子
       ↑
   current=0，找到"中文"（2个字）

移动后：
位置:  0  1  2  3
句子:  中 文 句 子
             ↑
         current=2
```

`break`

跳出当前的for循环，不再尝试更短的长度。

```
for i in range(5, 0, -1):  # 5, 4, 3, 2, 1
    if 找到了:
        break  # 立即退出for循环，不再试4, 3, 2, 1
```

为什么要break?

因为我们已经找到词了，不需要再试更短的了！

试5个字：中文句子由 ❌
试4个字：中文句子 ❌
试3个字：中文句 ❌
试2个字：中文 ✅ 找到了！break！
（不需要再试1个字"中"了）

四、完整执行过程示例

分词句子："我爱你"

词典： `{"我", "爱", "你", "我爱"}`

执行流程：

轮次1：

```
current = 0
sentence = "我爱你"
sentlen = 3

maxlen = min(3 - 0, 5) = 3  # 剩3个字，最多试3个

# for i in range(3, 0, -1):  → 3, 2, 1

i = 3: candidate = sentence[0:3] = "我爱你"
       "我爱你" in wordset? → False ❌

i = 2: candidate = sentence[0:2] = "我爱"
       "我爱" in wordset? → True ✅
       words.append("我爱") → words = ["我爱"]
       current += 2 → current = 2
       break  # 退出for循环
```

轮次2：

```
current = 2
剩余：1个字

maxlen = min(3 - 2, 5) = 1

# for i in range(1, 0, -1):  → 1

i = 1: candidate = sentence[2:3] = "你"
       i == 1 → True ✅   # 即使不检查词典，也接受
       words.append("你") → words = ["我爱", "你"]
       current += 1 → current = 3
       break
```

轮次3：

```
current = 3
sentlen = 3

while current < sentlen:  # 3 < 3? → False
# 循环结束

return words  # 返回 ["我爱", "你"]
```

最终输出： `["我爱", "你"]` → 打印成 `我爱 你`

五、为什么用 `set` 而不是 `list`？

情景对比

词典有17,000个词

用 list：

```
wordset = ["中文", "句子", "我", "爱", ..., 17000个词]

"中文" in wordset
# 电脑要做什么?
# 从第1个词开始，一个一个检查:
# 是"中文"吗? → 是！找到了
# 最坏情况：检查17000次
```

用 set：

```
wordset = {"中文", "句子", "我", "爱", ..., 17000个词}

"中文" in wordset
# 电脑要做什么?
# 用数学魔法（哈希表），直接计算位置
# 只需要检查1次!
```

速度对比

假设要分词1000个句子： - **用 list**：可能需要1小时 - **用 set**：只需要1秒

考试重点： set 查找速度是 O(1)，list 是 O(n)

六、UTF-8 编码（简单了解）

为什么需要编码?

电脑只认识 0 和 1（二进制）。

英文字母 'A'： - 电脑存储：01000001（1个字节）

中文字 '中'： - 电脑存储：11100100 10111000 10101101（3个字节！）

代码中的体现

```
with open(filename, encoding="utf8") as f:
```

`encoding="utf8"` 告诉Python： - 这个文件里的汉字用了3个字节存储 - 请正确翻译成汉字给我

如果不写 `encoding="utf8"`： - 可能看到乱码： `ä¸æ–‡`

七、segment2 的区别（选读）

segment1 和 segment2 做的事情**完全一样**，只是写法不同。

类比

segment1：用书签

```
current = 0  # 书签在第0页
读取 sentence[current:current+2]
current = 2  # 移动书签到第2页
```

segment2：撕纸

```
candidate = sentence[:2]  # 看前2个字
sentence = sentence[2:]   # 把前2个字撕掉
```

结果一样，方式不同而已！

八、纸笔考试怎么答?

可能的题目

题目：用贪婪匹配算法分词 "今天很好"

词典：今天, 天, 很, 好, 很好

答题步骤：

步骤1：
位置=0，句子="今天很好"，剩余4个字
试4个字："今天很好" → 不在词典 ❌
试3个字："今天很" → 不在词典 ❌
试2个字："今天" → 在词典 ✅
输出："今天"，位置移到2

步骤2：
位置=2，句子="今天很好"，剩余2个字
试2个字："很好" → 在词典 ✅
输出："很好"，位置移到4

步骤3：
位置=4，等于句子长度，结束

最终结果：今天 很好

必背要点

1. **算法从左到右处理**
2. **每次先试最长的词（5→4→3→2→1）**
3. **找到词或只剩1个字就接受**
4. **用 set 存词典，查找快**
5. **i == 1 是兜底，防止死循环**

九、记忆口诀

从左到右不回头，　　（从头开始处理）
先长后短慢慢找，　　（5→4→3→2→1）
词典查到就前进，　　（找到了就移动位置）
单字保底不会卡。　　（i==1 防止死循环）

十、常见错误理解（避坑）

❌ 错误1：以为要找"所有可能的分词"

```
"中文句子" 可能分成：
- 中文 句子
- 中 文句 子
- 中 文 句 子

贪婪算法只找第一种（最长匹配）！
```

❌ 错误2：以为 `i == 1` 可以去掉

```
if candidate in wordset:  # 只有这一个条件
```

问题： 遇到不在词典的字会死循环！

❌ 错误3：不理解为什么用 `break`

没有break：

```
找到"中文"（2个字），还会继续试1个字的"中"
结果：输出了"中文"，又输出"中"，错误！
```

十一、手工模拟练习题

练习1

句子： "我爱编程" **词典：** 我, 爱, 编程, 编, 程 **问：** 分词结果是什么？

▶ 点击查看答案

练习2

句子： "中国人" **词典：** 中国, 国人, 中, 人 **问：** 分词结果是什么?

▶ **点击查看答案**

最后叮嘱： - 考试时画图！把位置、候选词都标出来 - 多练几遍手工模拟，理解每一步 - 记住"为什么"比记住代码更重要

祝你考试顺利！🎉