ARTS: Algorithm, Review, Tip and Share

向世界分享你的作品：算法题解、文章点评、技术点和思想

作者：Lewis Tian

时间：March 18, 2021

版本：w19

关于作者：一个爱摸鱼吸猫放鸽子的懒狗

温柔正确的人总是难以生存，因为这世界既不温柔，也不正确。——比企谷八幡

目录

第一章 你的头脑是二值逻辑，还是三值逻辑？

readme | next

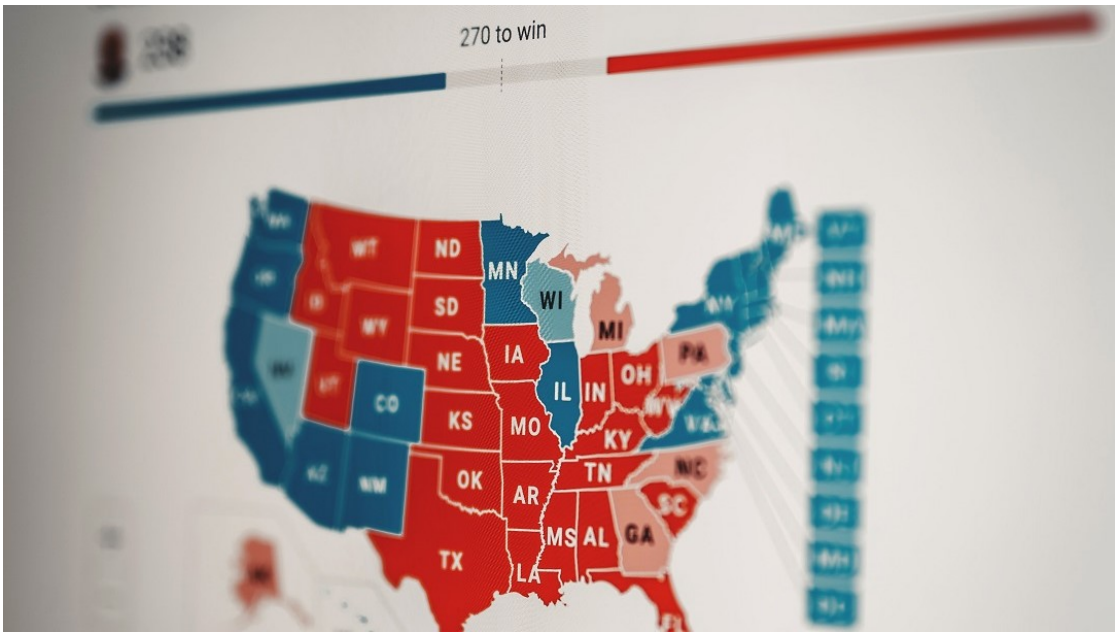

图 1.1: *Photo by Clay Banks on Unsplash*

总字数：*1753* 个（汉字：*1087*，英文单词：*178*，数字：*62*，中文标点：*124*，英文标点：*302*），阅读时长约：*3* 分 *30* 秒。

1.1 algorithm top

1.1.1 1356. 根据数字二进制下 1 的数目排序

20/11/6 的每日一题，将数组中的元素按照其二进制表示中数字 1 的数目升序排序。思路很简单，直接将数组按照 1 的个数分组，然后排序即可。

- code/leetcode_1356.cpp

```
class Solution {
  public:
  vector<int> sortByBits(vector<int>& arr) {
    vector<int> ret;
    map<int, vector<int>> mp;
    for (auto& num : arr) {
      int cnt = 0;
      int tmp = num;
      while (tmp != 0) {
        if (tmp & 1)
          cnt++;
```

```
      tmp >>= 1;
    }
    mp[cnt].push_back(num);
  }
  for (auto& iter : mp) {
    sort(iter.second.begin(), iter.second.end());
    for (auto& it : iter.second)
      ret.push_back(it);
  }
  return ret;
 }
};
```

1.2 review top

1.2.1 Python 的 eval() 函数很危险（英文）

Python 中的 eval() 函数用来执行一个字符串表达式，并返回表达式的值。它的语法如下：

```
eval(expression[, globals[, locals]])
```

- expression – 表达式。
- globals – 变量作用域，全局命名空间，如果被提供，则必须是一个字典对象。
- locals – 变量作用域，局部命名空间，如果被提供，可以是任何映射对象。

它并不会进行检查，所以即使你传入一个 `os.system('rm -rf /')` 这类命令它也会执行，所以这个函数是很危险的。

文中介绍了使用 eval 的" 攻防"，先是给出一段代码无法运行，似乎是保证了 eval 函数的安全，但是紧接着立马给出破解的方法，使得 eval 能跑出结果，挺有趣的。

本来是边看边跑文中给出的代码，然而文中的下面这段代码现在无法运行，文中介绍说需要提供 12 个参数，但是放到 ipython 中运行提示说参数不够，需要 14 个，不太懂这段代码的展开，以后有机会再看吧。

```
s = """
(lambda fc=(
    lambda n: [
        c for c in
            ().__class__.__bases__[0].__subclasses__()
            if c.__name__ == n
        ][0]
    ):
    fc("function")(
        fc("code")(
```

```
            0,0,0,0,"KABOOM",(),(),(),"","",0,""
        ),{}
    )()
)()
"""
eval(s, {'__builtins__':{}})
```

能够直接解析字符串运行的函数似乎都挺危险的，之前了解到 js 中也有这样的问题。算是个双刃剑吧，看使用者如何使用，用在恰当的地方就是好的 feature，被利用来干坏事就是 bug 了。

1.3 tip top

1.3.1 如何在 Markdown 文档显示 diff 效果（英文）

挺有趣的一个小技巧，之前也有想过怎么显示 diff 效果，后面就忘记去研究了。

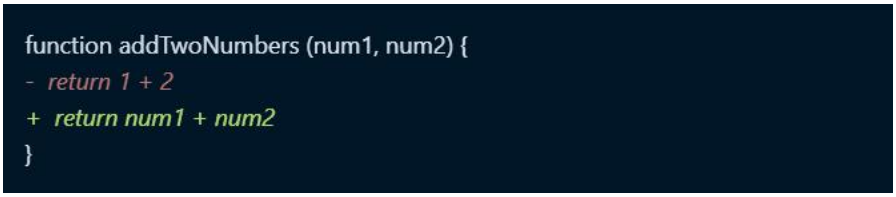

图 1.2: 20201106114217.jpg

其实也简单，本身就是 md 支持的，用 ‘diff‘ 标记语言类型，然后用 ‘+/-‘ 表示修改的代码行即可：

```
function addTwoNumbers (num1, num2) {
-  return 1 + 2
+  return num1 + num2
}
```

1.3.2 华氏度与摄氏度的简单估算（英文）

华氏度与摄氏度的转换，有一个简单的估算方法。有三个华氏度，颠倒个位数和十位数，等于对应的摄氏度。

因此，记住这三个数字（40、61、82），就可以简单估算：

摄氏度范围	华氏度范围	描述
< 4°C	< 40°F	Cold
4°C ~16°C	40°F ~61°F	Cool
16°C ~28°C	61°F ~82°F	Warm
> 28°C	> 82°F	Hot

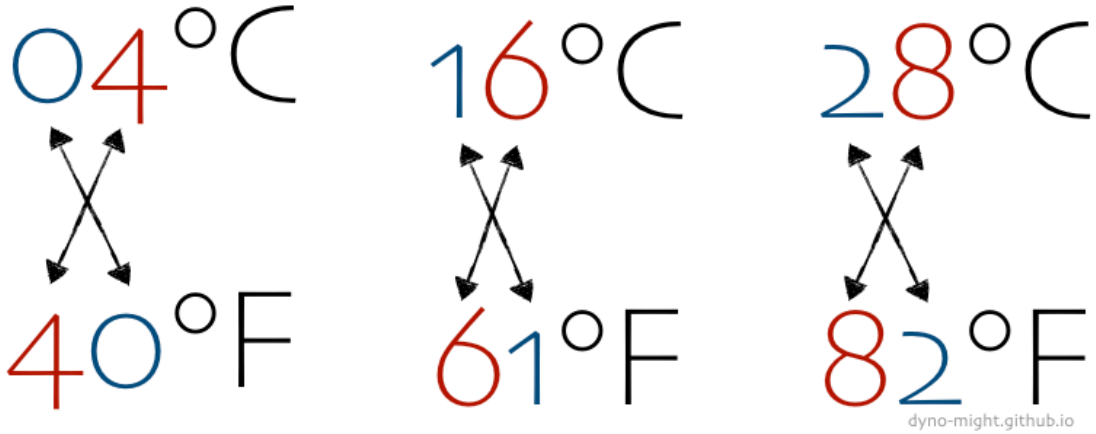

图 1.3: transpose.png

1.4 share top

1.4.1 你的头脑是二值逻辑，还是三值逻辑？

这个话题来自阮一峰的 第 131 期 科技爱好者周刊，他文中写道：

| 我现在的看法是，这可以区分一个人的世界观是否成熟深刻。有些年轻朋友就是二值逻辑的头脑，一看到不赞成、不理解、不喜欢的言论，就认定对方是错误的，完全否定，这其实是思想不成熟的表现。世界太复杂，很难用两分法来判断，三值逻辑会让你的心态好很多，而且有利于个人的进步：正确和错误之间，存在一个广阔的中间地带，任何一种言论都可能有正确的成分，要学会从中间地带去看待事物，吸收对自己有用的部分，摒弃无用的部分。

我比较赞成阮一峰老师的观点，为人处事不是做数学题，在你做完某道数学题后，结果就已经确定了，要么对要么错，但人不是，对于一个陌生人，你能用一个简单标准划分他是好人还是坏人吗？在对与错中间应该还有一个区域，就拿我们学过的牛顿力学一样，在某种条件下它是对的，在另外一种条件下，它又变成错的了。我们得发散地去看待事物，不要非黑即白，应该看到它中间的灰色地带。

两值世界观的人是没长大的小屁孩吧（指特定人群），轻易就能被人煽动，因为他们的思维是非黑即白的，一旦接受白变黑或者黑变白的设定，他们会很有认同感，然后就被牵着鼻子走。令人厌恶的网络喷子大概也是这类人吧。

readme | next

第 二 章 忘记业余项目，专注于工作

readme | previous | next

图 2.1: *Photo by Vasilios Muselimis on Unsplash*

总字数：*2225* 个（汉字：*1175*，英文单词：*428*，数字：*34*，中文标点：*149*，英文标点：*439*），阅读时长约：*4* 分 *27* 秒。

2.1 algorithm top

2.1.1 句子逆序

将一个英文语句以单词为单位逆序排放。题目很简单，在 TheAlgorithms/Python 中有对应的实现，具体见 strings/reverse_words.py 和 strings/reverse_letters.py，这俩有细微区别。

- code/nowcoder_48b3cb4e3c694d9da5526e6255bb73c3.py

```
def reverse_words():
    return "␣".join(input().split()[::-1])

print(reverse_words())
```

2.1.2 根据快速排序的思路，找出数组中第 K 大的数

来自牛客，想吐槽给的函数参数，既然第一个参数是 vector 还要传数组大小干嘛？

- code/nowcoder_e016ad9b7f0b45048c58a9f27ba618bf.cpp

```
class Finder {
  public:
  int findKth(vector<int> a, int n, int K) {
    return findKth(a, 0, n - 1, n - K + 1);
  }
  int findKth(vector<int>&a, int l, int r, int K) {
    if (l == r) return a[l];
    int i = l - 1, j = r + 1;

    int x = a[l + r >> 1];
    while (i < j) {
      do i++; while (a[i] < x);
      do j--; while (x < a[j]);
      if (i < j) swap(a[i], a[j]);
    }
    int s = j - l + 1;
    if (K <= s) return findKth(a, l, j, K);
    else return findKth(a, j + 1, r, K - s);
  }
};
```

2.2 review top

2.2.1 Why I Love Golang（英文）

作者介绍了他喜欢 go 的原因，读完你发现原因其实就是 go 的特性，比如 GOPATH 的设计；模块化；包的资源管理（go get github.com/xxx/yyy）；强制语言格式特性（大括号禁止换行）；编译安装的路径（GOPATH/pkg 和 GOPATH/bin）；跨平台编译；并发等等。

最近在开始学习 go，上面的几个特性也慢慢都接触到了，GOPATH 的设计我觉得挺妙的，导入包的时候不用去思考到底它在哪；一些格式特性也很爽，比如不用写分号（习惯 Python 之后再写 C/C++ 老是忘），大括号强制在行尾这正是我想要的，看到大括号换行的代码我强迫症真的忍不了！

另外感觉它跟 C 还挺像的，所以基础语法学起来挺快的。但是学语言都会陷入一种困境，你发现看完它的语法之后，似乎感觉已经学会了，但是好像什么都做不了。所以还是需要学以致用，像之前学 Python，写了很多爬虫之后对它的语法特性有了更深刻的了解，同时也写起来更熟练。目前学 go 可能没有很多时间去搞一些小项目，然而仅仅只是看不做的话很快就会忘记了，但这也是没办法的事。

2.3 tip top

2.3.1 std::pair 作为 std::unordered_map 的 key（中文）

unordered_map 是 C++11 中新加入的容器，底层是用 hash 实现的，对于键值就需要有 hash 函数计算出对应的 hash 值了。

对于 int 和 string 这种基础类型，C++ 提供了计算他们的 hash 值的函数。但是对于 `std::pair` 或者 `std::vector` 这种就没有，编译器会报错："The C++ Standard doesn't provide a hash for this type."。

下面是 unordered_map 的定义：

```
template < class Key,                                    // unordered_map::key_type
           class T,                                      // unordered_map::
               mapped_type
           class Hash = hash<Key>,                       // unordered_map::hasher
           class Pred = equal_to<Key>,                   // unordered_map::key_equal
           class Alloc = allocator< pair<const Key, T> > // unordered_map::
               allocator_type
           >
class unordered_map;
```

第一个模板参数是键，第二个模板参数是值，第三个模板参数是 hash 函数，第四个模板参数是相等的比较函数，最后一个是分配器。

我们跳到 hash 这个结构的定义部分，如下代码，对于基础类型，都给出了对应的模板特化，也可以看到上面编译错误的报错位置。

```
// TEMPLATE STRUCT hash
template<class _Kty>
struct hash: public _Bitwise_hash<_Kty> {
  static constexpr bool _Value = __is_enum(_Kty);
  static_assert(_Value,
               "The␣C++␣Standard␣doesn't␣provide␣a␣hash␣for␣this␣type.");
};
template<>
struct hash<bool>: public _Bitwise_hash<bool> {
  // hash functor for bool
};
template<>
struct hash<char>: public _Bitwise_hash<char> {
  // hash functor for char
};
```

所以如果对于自定义类型，或者 hash 类没有提供模板特化的数据类型，那就需要自己定义了，最简单的方法就是：

```
struct pair_hash {
  template<class T1, class T2>
```

```
  std::size_t operator() (const std::pair<T1, T2>& p) const {
    auto h1 = std::hash<T1> {}(p.first);
    auto h2 = std::hash<T2> {}(p.second);
    return h1 ^ h2;
  }
};

int main() {
  //unordered_map<pair<int, int>, int> error_mmp;    // error
  unordered_map<pair<int, int>, int, pair_hash> ok_mmp; // ok
}
```

2.3.2 GitHub Actions 入门教程（中文）

阮一峰老师教你如何使用 GitHub Actions。我看 TheAlgorithms/Python 那个项目就使用了 GitHub Actions，之前一直想学老是忘记了。

2.3.3 如何只对某些提交执行 GitHub Actions（英文）

```
jobs:
  format:
    runs-on: ubuntu-latest
    if: "contains(github.event.head_commit.message,␣'[build]')"
```

如果有上述配置，任何包含"[build]" commit 信息的提交都将触发这些作业，其他所有内容将被跳过。

```
jobs:
  format:
    runs-on: ubuntu-latest
    if: "!␣contains(github.event.head_commit.message,␣'wip')"
```

如果有上述配置，任何包含"wip" commit 信息的提交都跳过这些作业。

2.4 share top

2.4.1 忘记业余项目，专注于工作（英文）

我比较认同作者的观点，增加一些无用的项目对于提升自身并没有多大用处，就拿我来说，即便我现在学习了 Vue，并且仿做了几个热门应用（GitHub 上有很多这种），由于平时并不会使用到 Vue，很快我就会忘记这项技术，还不如做好自身的工作；

但是在日常工作会用到的项目或者技术却不一样，记得在本科时粗浅了解过很多语言，Java（大创要用）、ActionScript（看 B 站一个游戏视频入坑想自己做一个）、HTML+CSS+JavaScript

（当时写博客的冲动）、Bash Shell（使用 Linux）、Python（写爬虫玩）和 C++（自己的工作）等等，现在回头来看，前面提到的一些语言的学习都可以说是浪费时间，当时花了很多时间和精力去学习它们，现在还是都忘了，如果当初花多点时间去干点别的，比如 BY 的选择上，或许现在会很不一样。

不过有一说一，Python 我觉得是当初学习语言最对的一个，现在基本每天都会使用，无论是对于日常一些琐碎事情的自动化，还是对于自己的手头工作的黏合，都有很大帮助。

谢谢你当初选择和学习了 Python，谢谢你一路都坚持了下来。

readme | previous | next

第 三 章 自由并不简单

readme | previous | next

图 3.1: *Photo by unknown on* 中国政府网

总字数：*2869* 个（汉字：*1508*，英文单词：*531*，数字：*47*，中文标点：*195*，英文标点：*588*），阅读时长约：*5* 分 *44* 秒。

3.1 algorithm top

3.1.1 922. 按奇偶排序数组 II

20/11/12 的每日一题，对数组进行排序，以便当 A[i] 为奇数时，i 也是奇数；当 A[i] 为偶数时，i 也是偶数。

题目不难，首先建一个跟原数组同样大的数组，用两个变量标记当前奇偶的下标，遍历原数组，依次将数据填入返回数组中。本来是用 C++ 实现的，然后又用 Go 实现了一遍，区别并不大，只不过有些语法不太熟，需要查一下才知道怎么用。

- code/leetcode_0922.go

```
func sortArrayByParityII(A []int) []int {
  ret := make([]int, len(A))
  even_idx, odd_idx := 0, 1
  for _, x := range(A) {
    if x % 2 == 0 {
      ret[even_idx] = x
      even_idx += 2
    } else {
```

```
      ret[odd_idx] = x
      odd_idx += 2
    }
  }
  return ret
}
```

3.1.2 402. 移掉 K 位数字

是一道 medium 的题，不知道咋优化，看了下题解拨云见日，茅塞顿开（老吕布了）。

思路：从左往右遍历原字符串，我们保留每个遍历到的字符（X），但是我们可以选择保留还是舍弃前一个字符（Y），关键看 X 和 Y 的大小，如果 Y > X，显然，舍弃 Y 得到的数会更小，否则保留 Y。如果遍历过程中就删除了 K 个字符，那么遍历结束；如果遍历结束还没删够 K 个，那么删除最后的若干字符。

- code/leetcode_0402.go

```
func removeKdigits(num string, k int) string {
  stack := []byte{}
  for i := range num {
    digit := num[i]
    for k > 0 && len(stack) > 0 && digit < stack[len(stack)-1] {
      stack = stack[:len(stack)-1]
      k--
    }
    stack = append(stack, digit)
  }
  stack = stack[:len(stack)-k]
  ans := strings.TrimLeft(string(stack), "0")
  if ans == "" {
    ans = "0"
  }
  return ans
}
```

3.2 review top

3.2.1 亚马逊银河数据湖（英文）

这篇文章介绍了亚马逊为了解决他们关于大数据的问题而整出的一个名叫银河数据湖（Galaxy data lake）的东西。

看完之后我感觉它是一个大型集中式非关系型数据库，提供了导入、访问、管理等功能。像亚马逊这种大公司都存在类似的问题，在全球分布着若干数据中心，每个地方

的数据以不同的格式存储，以不同的方式管理，若想访问多个数据中心的数据得获得相应的凭证（授权），所以五花八门无法统一管理，还存在安全隐患。因此亚马逊提出数据湖的想法，建立一个集中式数据库，将所有不同格式的数据存到数据湖中，对外提供统一的访问和管理接口，既方便管理也更安全，同时大量的数据也为机器学习和人工智能提供了海量的训练集，这就为亚马逊将来的成本、服务都能提供很好的决策。

在建立数据湖过程中也利用了一些现有的组件，它是构建在 Amazon S3 之上的，使用了 AWS Glue、AWS DMS、Amazon DynamoDB、Amazon ES 以及 Amazon Athena 和 Amazon SageMaker 等组件，他们分别提供了 ETL、迁移、存储、查询等功能。

数据库真是个有趣的东西，可能以后工作了跟它打交道的次数会更多，那时体会可能更深刻。但是不得不说它确实是 CS 最重要的课程之一，然而在学校（华科，没错我说的就是你）的学习体验贼差。

3.3 tip top

3.3.1 Python 异常处理之 try/except/else/finally（中文）

之前都是无脑 `try...except...`，今天来看看其他的用法。

1. try - except

```
try:
  print("Before error")
  a = b
  print(a)
  print("After error")
except:
  print("Error")

print("Continue!")

# Before error
# Error
# Continue!
```

所以流程如下：

- 先执行 try block，直到发现了错误，不再执行异常之后的代码
- 执行 except block
- 向下继续

对于多个 except，则会依次匹配错误，如果没有匹配到程序会报错，所以可以最后加一个通用匹配。像下面如果没有最后一个 except 程序会报错：

```
try:
  a = b
  print(a)
```

```
except SyntaxError:
  print("<<<<␣SyntaxError")
except SystemExit:
  print("<<<<␣NameError")
except:
  print("I␣don't␣know,␣but␣error.")

print("He,␣try/except␣is␣so␣difficult!")

# I don't know, but error.
# He, try/except is so difficult!
```

2. try - except - else

```
try:
  a = b
  print(a)
except SyntaxError:
  print("<<<<␣SyntaxError")
except SystemExit:
  print("<<<<␣NameError")
except:
  print("I␣don't␣know,␣but␣error.")
else:
  print("That's␣good,␣no␣error.")

print("He,␣try/except␣is␣so␣difficult!")

# I don't know, but error.
# He, try/except is so difficult!
```

可见，有异常时，else block 时不执行的。

```
try:
  b = 1
  a = b
  print(a)
except SyntaxError:
  print("<<<<␣SyntaxError")
except SystemExit:
  print("<<<<␣NameError")
except:
  print("I␣don't␣know,␣but␣error.")
else:
  print("That's␣good,␣no␣error.")
```

```
print("He, try/except is so difficult!")

# 1
# That's good, no error.
# He, try/except is so difficult!
```

可见，无异常时 else block 执行。

3. try - finally

   无论 try 语句是否有异常，最后都要执行的代码。

```
try:
  b = 1
  a = b
  print(a)
except SyntaxError:
  print("<<<< SyntaxError")
except SystemExit:
  print("<<<< NameError")
except:
  print("I don't know, but error.")
else:
  print("That's good, no error.")
finally:
  print("I will be here!")

print("He, try/except is so difficult!")

# 1
# That's good, no error.
# I will be here!
# He, try/except is so difficult!
```

```
try:
  a = b
  print(a)
except SyntaxError:
  print("<<<< SyntaxError")
except SystemExit:
  print("<<<< NameError")
except:
  print("I don't know, but error.")
else:
  print("That's good, no error.")
```

```
finally:
  print("I␣will␣be␣here!")

print("He,␣try/except␣is␣so␣difficult!")

# I don't know, but error.
# I will be here!
# He, try/except is so difficult!
```

3.3.2 Python `yield from`（StackOverflow）

很棒的一个回答，详细地介绍了 `yield from` 的作用，如果只是简单使用，可以把他当作是对于迭代器的展开（`yield from iterator == for v in iterator: yield v`），但是它不仅仅如此，它在调用者与子生成器之间构建了一个透明的双向联系（*establishes a transparent bidirectional connection between the caller and the sub-generator*），只不过这个关键词字面意思并没有表现出来罢了。

另外使用 `yield from` 可以从生成器中读取/向生成器发送数据（双向），而且它自动完成了异常向子生成器的传递。

```
class SpamException(Exception):
    pass

def writer():
    while True:
        try:
            w = (yield)
        except SpamException:
            print('***')
        else:
            print('>>␣', w)

def writer_wrapper(coro):
    yield from coro

w = writer()
wrap = writer_wrapper(w)
wrap.send(None) # "prime" the coroutine
for i in [0, 1, 2, 'spam', 4]:
    if i == 'spam':
        wrap.throw(SpamException)
    else:
        wrap.send(i)
```

```
# >> 0
# >> 1
# >> 2
# ***
# >> 4
```

建议去看原回答，写得很详细。

3.3.3 Python async/await 入门指南（中文）

看懂了前半部分，后面就看不懂了 @-@，异步编程还是太复杂了，可能写一些简单的东西会很简单，一旦涉及更深的东西就很复杂了。

3.4 share top

3.4.1 自由并不简单（英文）

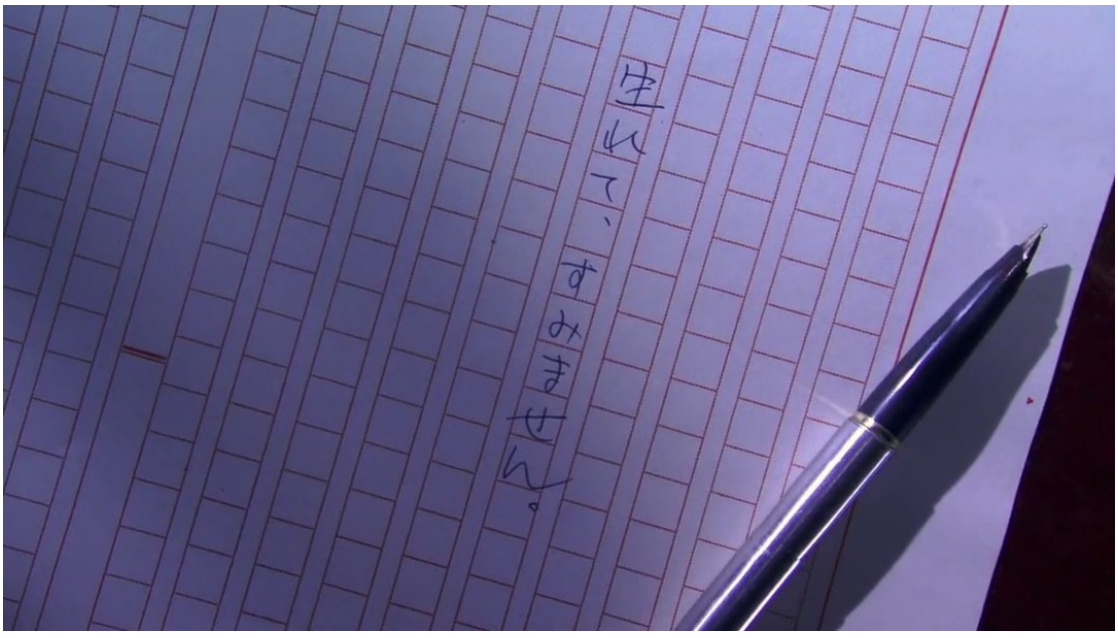

图 3.2: 《被嫌弃的松子的一生》电影截图

作者观点很有趣，他认为自由并不是一维的，并不能简单地对其进行评分说你的自由是增加了还是减少了。因为人是群居动物，身处一个群体中就会受到其他人的影响，比如，邻居凌晨 4 点播放音乐的自由肯定会影响你睡觉的自由，这种情况下，你觉得自由是增加了还是减少了？自由是一个多维的，增加一个人/群体的自由往往会减少其他人的自由。

很显然，自由是不简单的，每个人都能清醒地意识到。尽管我的格言是【唯自由与美食不可负】，但也只是美好的愿景罢了，从出生就背负了各种责任怎么可能自由。小时候，你常听到的或许是【好好学习，少跟别人出去玩，你看那个 X，又考了 yy 分，你再

看看你...】；再大一点，你会听到【你看那个 X，他妈说他要去 xx 高中，以后要去 yy 大学，你看看你，不好好努力，总搞些与学业无关的东西，以后只能去普通 zz 高中，读个水 vv 大学...】；再后来啊，或许会被家里人所有人催着找男/女朋友，催着结婚生子，如此而已，没什么好说的。

生活中往往受到各种限制，无论是来自家庭还是旁人还是社会，或许某一天你财富自由了，那个时候你就能追求更广阔的自由了吧。

生而为人，我很抱歉。

readme | previous | next