问题解答

动态规划H4作业详解

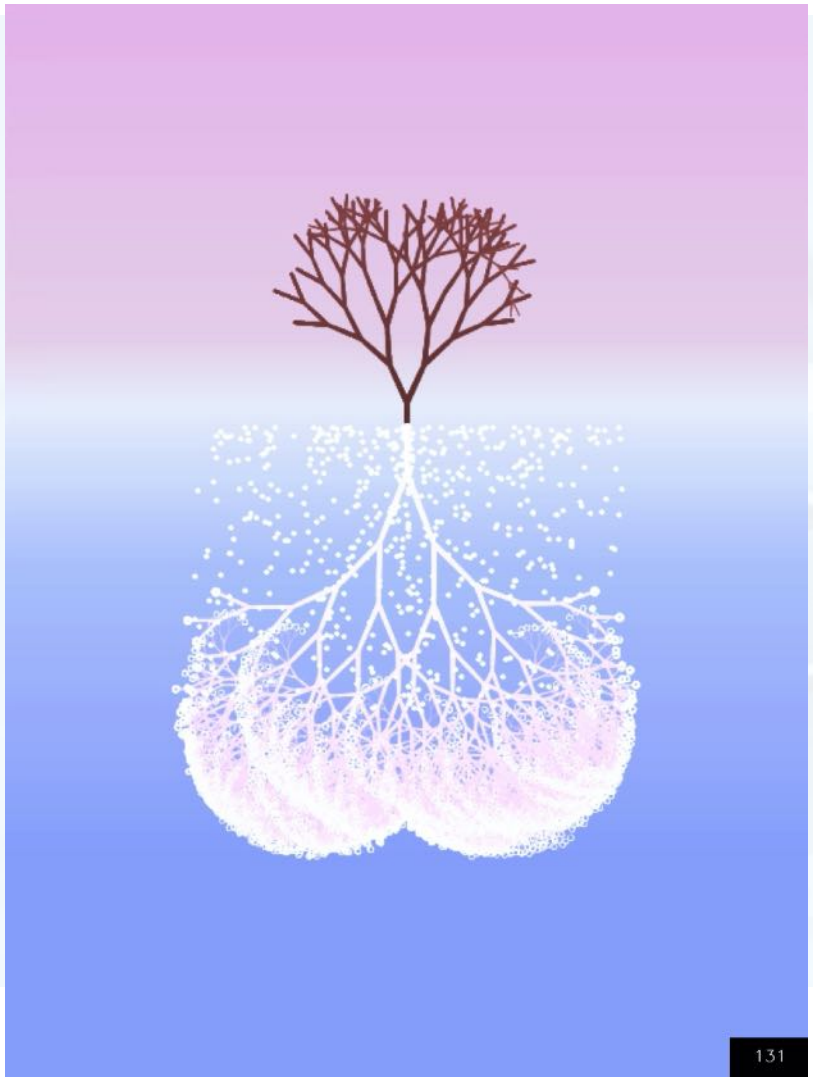
背包问题和伪多项式时间复杂度

课堂练习

数据结构与算法（Python）-11/0331

陈斌 gischen@pku.edu.cn 北京大学地球与空间科学学院

目录



问题解答？

- 【H4】博物馆大盗问题如果要列出所有情况应该如何处理？
  （动态规划回溯查找的思路，能否覆盖所有情况？）
- 【H4】回溯程序的原理及写法
  （因为可以每生成一个方格就记录下对应的操作所以没写回溯）
- 【H4】最小编辑距离问题？
- 【H4】大佬说的H4十几行写完是怎么做到的？
- 【Python】如何形成良好的代码风格？
- 【Python】集合是怎么实现的？为什么是iterable?

问题解答？

- 【OJ】 OJ第二题这类需要考虑两边的情况的题怎么用动态规划

分发糖果是一个贪心策略可解的问题

从左到右，从右到左，两次扫描即可

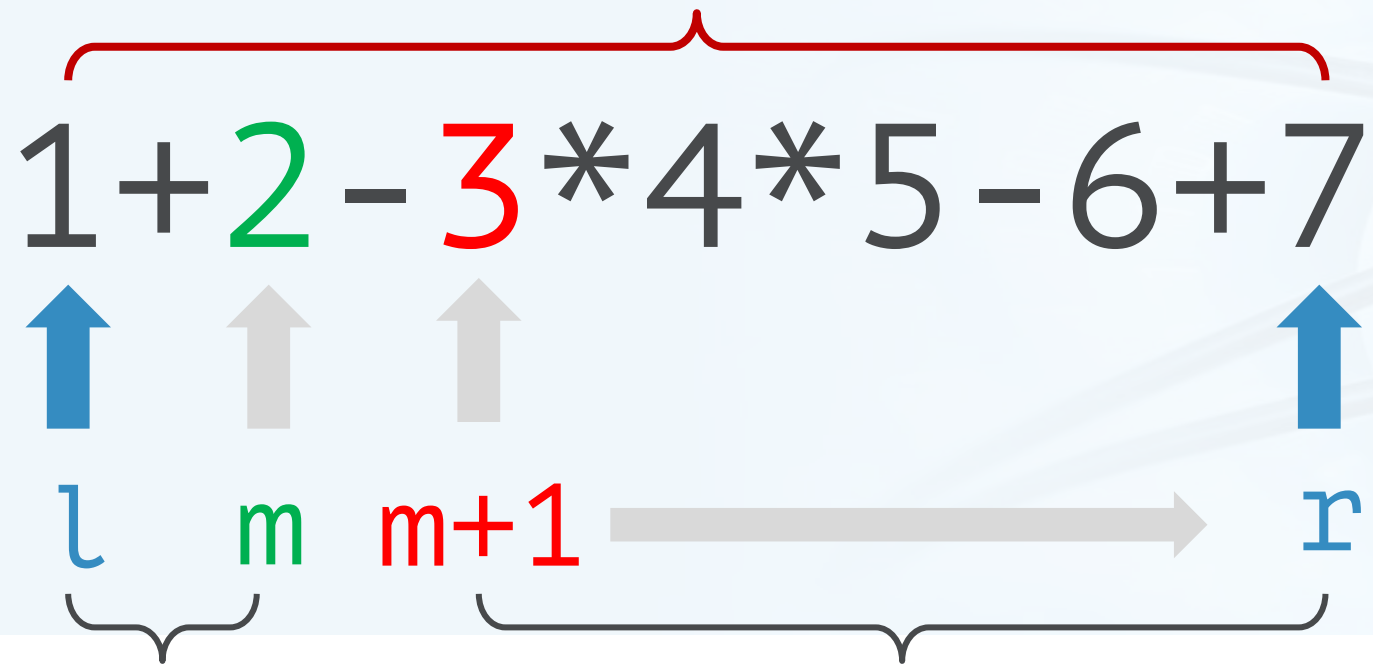
每趟扫描的规则：至少1颗糖；如果下一人评分变高加1颗；不高就不加

取每人两趟的最大值累加，就是最少糖果数量

https://leetcode-cn.com/problems/candy/solution/

问题解答

- OJ第三题：表达式按不同顺序求值
  http://39.100.145.216/p/view/16467069-14da-47f0-86ee-edb41af89c68/



OJ第三题：表达式按不同顺序求值


```
26 ▼    def helper(l, r):
27          res = set()
28 ▼        if l == r:
29              res.add(nums[l])
30 ▼        else:
31 ▼            for m in range(l, r):
32                  fun = op_lst[ops[m]]
33 ▼                for n1 in helper(l, m):
34 ▼                    for n2 in helper(m + 1, r):
35                          res.add(fun(n1, n2))
36          return list(res)
```

递归结束条件：没有算符

变动m

缩小规模，递归调用

问题解答？

- 背包问题为什么是np问题？能否详细讲一下背包问题？
  伪多项式时间算法
- 动态规划如何优化到比O(n^2)更好的结果？
- 贪心算法要求的“局部最优等同于总体最优”，和动态规划要求的“问题最优解包括规模更小相同问题的最优解”，二者有什么不同？
- 希望以后编程作业描述清楚，最好另附文件进行描述，不要和代码放在一起，否则题目描述过于简略。
  比如这次作业，“编辑距离”在给出的参考资料是操作次数，而在题目中没有描述清楚

问题解答

- **麻烦老师讲一下八皇后问题**

  https://www.bilibili.com/video/BV1xV411f7gK

- **函数值缓存为什么最好用字典？**

  稀疏数组？

- **如何计算递归的时间复杂度和空间复杂度**

  可以观察在函数的一次执行中

  同一个层次会被递归调用几次

```python
def fibonacci(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

```

```python
def queen(A, cur=0):
    if cur == len(A):
        print(A)
        return 0
    for col in range(len(A)):
        A[cur], flag = col, True
        for row in range(cur):
            if A[row] == col or abs(col
                flag = False
                break
        if flag:
            queen(A, cur+1)
queen([None]*8)
```

问题解答

- **为什么列表排序不能用动态规划？**
  周四开放的查找与排序，里面不少排序算法都接近动态规划的策略
- **记忆化递归是否也属于动态规划？动态规划完整定义是如何定义的？**
- **动态规划用递归的方式需要解所有的子问题，不管子问题在最终的求解中有没用被用到；而记忆化递归只需依次求解用到的子问题，是否记忆化递归的效率高于动态规划？动态规划的优势在哪里？**

问题解答

H4模板代码treasurelist在函数内外都有定义是不是不太好

Line 9 : Redefining name 'treasureList' from outer scope (line 50)

It looks like the local variable is hiding a global variable with the same name.

Most likely there is nothing wrong with this. I just wanted to remind you that you can't access the global variable like this. If you knew it then please ignore the warning.

问题解答

- Python没有强制数据类型，容易把想要存储的数据类型误存为其他类型而没有发现，请问有什么办法防止吗？（如lst[i][j].append(x)写成lst[i][j]=x）

  可以使用assert函数，参数是一个逻辑表达式，额外添加检验

```
22      # 逐个填写二维表格
23      for i in range(1, len(tr)):
24          for w in range(1, maxWeight + 1):
25
26              assert(isinstance(m[(i,w)], list))
27
28              if tr[i]['w'] > w:  # 装不下第i个宝物
29                  m[(i, w)][0]= m[(i-1, w)][0]
30                  m[(i, w)][1]= None # 不装宝物
```

问题解答

H系列一共多少次？

查找与排序

树及其算法

图及其算法

？

【H4-1】作业详解：博物馆大盗

- 视频412详细解释了算法过程
- 最大价值表格m(i,W)

W →

i ↓

m	0	1	2	3	4	5	...
0	0	0	0	0	0	0	
1	0	0	3	3	3	3	
2	0	0	3	4	4	7	
3	0	0	3	4	8	8	
4	0	0	3	4	8	8	
5	0	0	3	4	8	8	

m(5,5)=m(4,5)=max(m(3,5), m(3,0)+8)

$$m(i,W)=\begin{cases}0 & \text{if } i=0\\ 0 & \text{if } W=0\\ m(i-1,W) & \text{if } w_i>W\\ \max\{m(i-1,W),v_i+m(i-1,W-w_i)\} & \text{otherwise}\end{cases}$$

* 装不下第i个
* 装或者不装/价值大者

【H4-1】作业详解：博物馆大盗

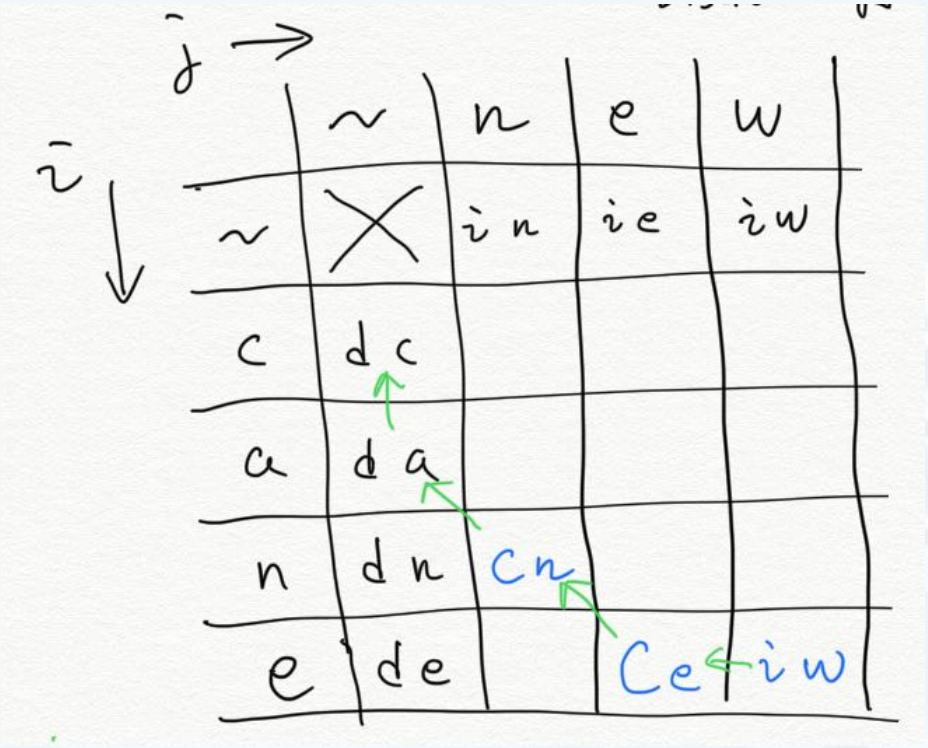
- 主要是输出选择的宝物
- 在m(i,W)中记录：[最大价值, 加了哪个宝物得到的最大价值]
- 小技巧：让宝物编码从1开始

```
 9  def dpMuseumThief(treasureList, maxWeight):
10      maxValue = 0
11      chosenList = []
12
13      # 请在此编写你的代码（可删除pass语句）
14      tr = [None] + treasureList #让宝物编码从1开始
15
16      # 初始化二维表格m[(i, w)]，均为0,None
17      # 表示前i个宝物中，最大重量w的组合，所得到的最大价值
18      # 以及加了哪个宝物得到的这个最大价值
19      # 当i什么都不取，或w上限为0，价值均为0
20      m = {(i, w):[0, None] for i in range(len(tr))
21                            for w in range(maxWeight + 1)}
```

```
22      # 逐个填写二维表格
23      for i in range(1, len(tr)):
24          for w in range(1, maxWeight + 1):
25              if tr[i]['w'] > w:  # 装不下第i个宝物
26                  m[i, w][0]= m[i-1, w][0]
27                  m[i, w][1]= None # 不装宝物
28              else:
29                  # 不装第i个宝物，装第i个宝物，两种情况下最大价值
30                  if m[i-1, w][0]> m[i-1, w-tr[i]['w']][0] + tr[i]['v']:
31                      m[i, w][0]= m[i-1, w][0]
32                      m[i, w][1]= None # 不装宝物
33                  else:
34                      m[i, w][0]= m[i-1, w-tr[i]['w']][0] + tr[i]['v']
35                      m[i, w][1]= tr[i]
36
37      maxValue = m[i, w][0]
38      while w > 0:
39          if m[i, w][1] is not None: # 如果装了宝物，就输出
40              chosenList.insert(0, m[i, w][1])
41              w = w - m[i, w][1]['w']
42          i = i - 1
43
44      # 代码结束
45
46      return maxValue, chosenList
```

【H4-2】作业详解：单词最小编辑距离

- 实际上，这个问题跟博物馆大盗算法基本一样
- 构造最小距离表格m(i, j)



$$m(i,j)=\begin{cases}\text{insert } *j & \text{当 } i==0\\ \text{delete } *i & \text{当 } j==0\\ \min\begin{cases}\text{copy } o[i] + m(i-1,j-1) & \text{"有条件"}\ o[i]==t[j]\\ \text{delete } o[i] + m(i-1,j)\\ \text{insert } t[j] + m(i,j-1)\end{cases}\end{cases}$$

【H4-2】作业详解：单词最小编辑距离

- 在m(i,j)中记录：[最小距离, 使用了哪个操作得到的最小距离]
- 小技巧：让original、target字符编码从1开始

```
10  def dpWordEdit(original, target, oplist):
11      score = 0
12      operations = []
13
14      # 请在此编写你的代码（可删除pass语句）
15
16      # 初始化二维表格m[(i, j)]，均为0,None
17      # 表示源单词前i个字符子串，变为目标单词前j个字符子串，的最小编辑距离
18      # 以及最后进行了哪个操作得到的这个最小编辑距离
19      m = {(i, j):[0, None] for i in range(len(original)+1)
20                            for j in range(len(target)+1)}
21
22      # 让字符的序号从1开始
23      original, target = "-" + original, "-" + target
24
25      # 设定首行m[0,j]，全insert
26      for j in range(1, len(target)):
27          m[0, j] = [oplist["insert"] * j, "insert " + target[j]]
28      # 设定首列m[i,0]，全delete
29      for i in range(1, len(original)):
30          m[i, 0] = [oplist["delete"] * i, "delete " + original[i]]
```

```
    # 逐个填写二维表
    for i in range(1, len(original)):
        for j in range(1, len(target)):
            ops = [] # 可能的操作和对应的分数
            # copy操作，有条件
            if original[i] == target[j]:
                ops.append([m[i-1, j-1][0] + oplist["copy"], "copy " + original[i]])
            # delete操作
            ops.append([m[i-1,j][0] + oplist["delete"], "delete " + original[i]])
            # insert操作
            ops.append([m[i,j-1][0] + oplist["insert"], "insert " + target[j]])

            # 取分数最小的
            m[i,j] = min(ops, key=lambda x:x[0])

    score = m[i,j][0]
    while i> 0 or j> 0:
        op = m[i,j][1]
        operations.insert(0, op) # 倒序输出操作序列
        if "copy" in op:
            i, j = i - 1, j - 1
        elif "delete" in op:
            i = i - 1
        elif "insert" in op:
            j = j - 1

    # 代码结束

    return score, operations
```

关于输出所有的组合

所有的最大价值组合、所有的最小距离操作组合

```
if tr[i]['w'] > w:  # 装不下第i个宝物
    m[(i, w)][0]= m[(i-1, w)][0]
    m[(i, w)][1]= None # 不装宝物
else:
    # 不装第i个宝物，装第i个宝物，两种情况下最大价值
    if m[(i-1, w)][0]> m[(i-1, w-tr[i]['w'])][0] + tr[i]['v']:
        m[(i, w)][0]= m[(i-1, w)][0]
        m[(i, w)][1]= None # 不装宝物
    else:
        m[(i, w)][0]=
        m[(i, w)][1]=
```

```
ops = [] # 可能的操作和对应的分数
# copy操作，有条件
if original[i] == target[j]:
    ops.append([m[(i-1, j-1)][0] + oplist["copy"], "
# delete操作
ops.append([m[(i-1,j)][0] + oplist["delete"], "delet
# insert操作
ops.append([m[(i,j-1)][0] + oplist["insert"], "inser

# 取分数最小的
m[(i,j)] = min(ops, key=lambda x:x[0])
```

大佬的十几行完成H4怎么回事？

- 实际上，刚才的单词最小编辑距离详解，去掉所有的空行和注释，也仅仅25行
- 如果再把什么三元运算符、推导式都用上，压缩到20行以内并不困难
- 再对表格的表示进行压缩转换，还可以再短
- 一行狂魔……

关于如何形成良好的代码风格？

可读性

变量、函数命名；必要的注释（关键点、函数参数返回值等）

代码格式（PEP8自动格式化）

可维护性

尽量不用全局变量；

不依赖于某些特定缺省值；

不硬编码某些特定位置、特定值等；

可扩展性

一些常量用符号代替，并集中放在文件头部

不要`import *`

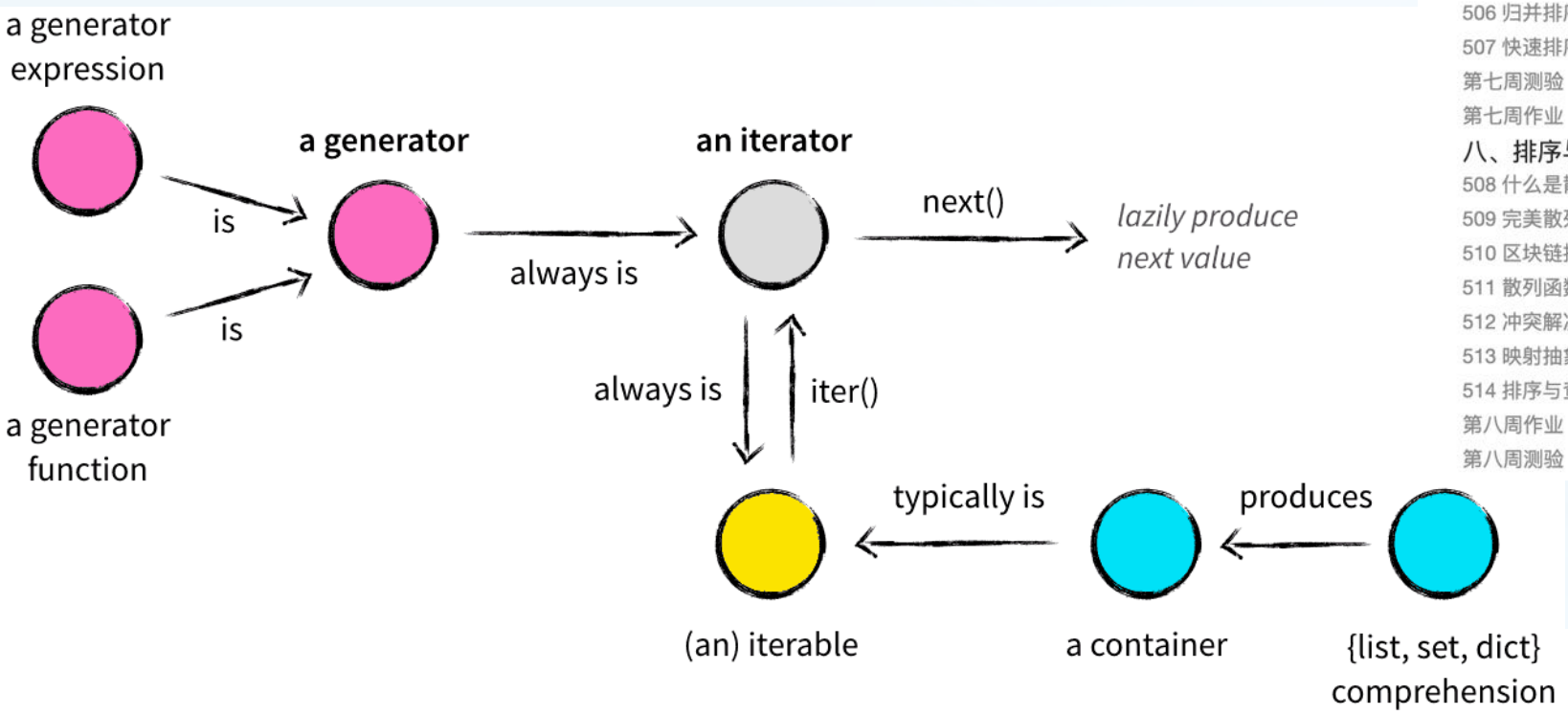
用类的接口方法而不是类变量

【Python】集合是怎么实现的？为什么是可迭代对象

- 所谓的可迭代对象，是可以对所包含数据项**逐个枚举**的对象
- 集合是“简版”字典，只有key没有value的字典

  都是通过散列表（Hashtable）来实现，见下下周的内容

七、排序与查找（上）
501 顺序查找算法及分析 9m41s
502 二分查找算法及分析 12m20s
503 冒泡和选择排序算法及分析 12m14s
504 插入排序算法及分析 7m06s
505 谢尔排序算法及分析 6m15s
506 归并排序算法及分析 9m13s
507 快速排序算法及分析 12m30s
第七周测验
第七周作业
八、排序与查找（下）
508 什么是散列 7m21s
509 完美散列函数 15m02s
510 区块链技术 17m20s
511 散列函数设计 8m47s
512 冲突解决方案 11m59s
513 映射抽象数据类型及Python实现 14m58s
514 排序与查找小结 9m45s
第八周作业
第八周测验



背包问题和伪多项式时间复杂度

- 看起来0-1背包问题的复杂度O（nW）是个多项式时间复杂度
- 但是，在图灵机等价的计算模型下
- 背包问题的规模取决于最大重量W的二进制位数logW，
  当logW增加1，计算时间就是原来的2倍，这是指数增长（2^{logW}）
- 背包问题目前尚未找到真正的多项式时间复杂度算法

背包问题和伪多项式时间复杂度

- 另一个具有伪多项式时间复杂度算法的NP问题是**判断素数**
- 其输入规模为数N的二进制位数logN

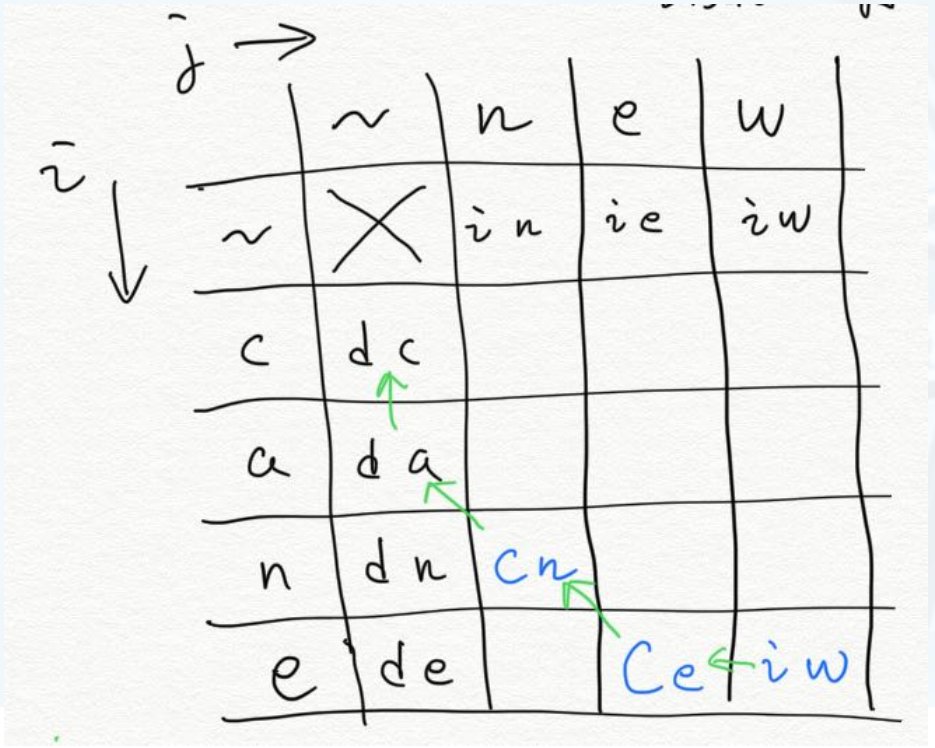
  当logN增加1，计算时间就是原来的2倍
- 那么运行时间检测是怎么回事？

  实际的计算机整数是固定长度，已经预留了至少32bits二进制

贪心策略和动态规划的区别

- 贪心算法要求的“局部最优等同于总体最优”，
- 和动态规划要求的“问题最优解包括规模更小相同问题的最优解”，
- 二者有什么不同？
- 包括：仅依赖于规模小的最优解

尽量大步 → 尽量大步 → 到达最优



课堂练习

- 请用递归方法解决单词最小编辑距离问题。
- 代码模版同【H4-2】
- 可以自由编辑，用附带的用例检测即可。
- 请代码截图和运行结果截图做成PDF上传。