最小基因变化：

题目：

一条基因序列由一个带有8个字符的字符串表示，其中每个字符都属于`"A"`，`"C"`，`"G"`，`"T"`中的任意一个。

假设我们要调查一个基因序列的变化。**一次**基因变化意味着这个基因序列中的**一个**字符发生了变化。

例如，基因序列由`"AACCGGTT"`变化至`"AACCGGTA"`即发生了一次基因变化。

与此同时，每一次基因变化的结果，都需要是一个合法的基因串，即该结果属于一个基因库。

现在给定3个参数 — start, end, bank，分别代表起始基因序列，目标基因序列及基因库，请找出能够使起始基因序列变化为目标基因序列所需的最少变化次数。如果无法实现目标变化，请返回 -1。

注意：

1. 起始基因序列默认是合法的，但是它并不一定会出现在基因库中。
2. 所有的目标基因序列必须是合法的。
3. 假定起始基因序列与目标基因序列是不一样的。

示例 1：

```
start: "AACCGGTT"
end:   "AACCGGTA"
bank: ["AACCGGTA"]

返回值: 1
```

示例 2：

```
start: "AACCGGTT"
end:   "AAACGGTA"
bank: ["AACCGGTA", "AACCGCTA", "AAACGGTA"]

返回值: 2
```

示例 3：

```
start: "AAAAACCC"
end:   "AACCCCCC"
bank: ["AAAACCCC", "AAACCCCC", "AACCCCCC"]

返回值: 3
```

思路：

因为 字符的可能就是 ACGT，所以从0开始 ，依次替换start中的字符为ACGT：

tmp = word[:i] + p + word[i+1:]

判断这个这个字符是否在bank中出现，出现就移除，避免重复计算，一直循环下去，如果找到end终结，否则返回 -1

```
def minMutation(self, start, end, bank):
        """
```

```
        :type start: str
        :type end: str
        :type bank: List[str]
        :rtype: int
        """
        possible = ["A", "C", "G", "T"]
        queue = [(start, 0)]
        while queue:
            # BFS
            (word, step) = queue.pop(0)
            if word == end:
                return step
            for i in range(len(word)):
                for p in possible:
                    # 从第0个位置开始匹配新的字符串
                    tmp = word[:i] + p + word[i+1:]
                    # 在bank里就处理，（set中in操作复杂度为 O(1)
                    if tmp in bank:
                        # 从bank中移除，避免重复计数
                        bank.remove(tmp)
                        # 加入队列，步数+1
                        queue.append((tmp, step+1))
        return -1
```

单词接龙：

给定两个单词（*beginWord* 和 *endWord*）和一个字典，找到
从 *beginWord* 到 *endWord* 的最短转换序列的长度。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换过程中的中间单词必须是字典中的单词。

说明:

- 如果不存在这样的转换序列，返回 0。
- 所有单词具有相同的长度。
- 所有单词只由小写字母组成。
- 字典中不存在重复的单词。
- 你可以假设 *beginWord* 和 *endWord* 是非空的，且二者不相同。

示例 1:

```
输入:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

输出: 5

解释: 一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog",
     返回它的长度 5。
```

示例 2:

```
输入:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

输出: 0

解释: endWord "cog" 不在字典中，所以无法进行转换。
```

思路：

```
def __init__(self):
        self.length = 0
        self.all_combo_dict = default(list)

    def visitWordNode(self, queue, visited, others_visited):
        current_word, level = queue.pop(0)
        for i in range(self.length):
            intermediate_word = current_word[:i] + "*" + current_word[i+1

             for word in self.all_combo_dict[intermediate_word]:
```

```
                if word in others_visited:
                    return level + others_visited[word]
                if word not in visited:
                    visited[word] = level+1
                    queue.append((word, level+1))
        return None

    def ladderLength(self, beginword, endWord, wordList):

        if endWord not in wordList or not endWord or not beginword or no
            return 0
        self.length = len(beginword)
        for word in wordList:
            for i in range(self.length):
                self.all_combo_dict[word[:i] + "*" + word[i+1 :]].append

        queue_begin = [(beginword, 1)]
        queue_end = [(endWord, 1)]

        visited_begin = {beginword, 1}
        visited_end = {endWord, 1}
        ans = None

        while queue_begin and queue_end:
            ans = self.visitWordNode(queue_begin, visited_begin, visited
            if ans:
                return ans
            ans = self.visitWordNode(queue_end, visited_end,visited_begi
            if ans:
                return ans

        return 0

```

思路：

BFS

构建一个字典，wordList中只改变其中一个字符，all_combo_dict = {"*ot":["hit","dot","lot"],"h*t":["hot"],....}

从头开始遍历字典，如果中value值与endword相同，就输出经过了几轮变换，

```
def ladderLength(self, beginWord, endWord, wordList):
        """
        :type beginWord: str
```

```
        :type endWord: str
        :type wordList: List[str]
        :rtype: int
        """
        if endWord not in wordList or not endWord or not beginWord or not
            return 0

        L = len(beginWord)
        # 这一步构建一个字典,
        # all_combo_dict = {"*ot":["hit","dot","lot"],"h*t":["hot"],....
        # 设置一个默认value为list的字典
        all_combo_dict = defaultdict(list)
        for word in wordList:
            for i in range(L):
                all_combo_dict[word[:i] + "*" + word[i+1:]].append(word)

        queue = [(beginWord, 1)]
        visited = {beginWord:True}
        while queue:
            current_word,level = queue.pop(0)
            for i in range(L):
                intermediate_word = current_word[:i] + "*" + current_wor

                for word in all_combo_dict[intermediate_word]:
                    if word == endWord:
                        return level + 1
                    if word not in visited:
                        visited[word] = True
                        queue.append((word, level+1))
                all_combo_dict[intermediate_word] = []
        return 0
```

单词接龙II

给定两个单词（*beginWord* 和 *endWord*）和一个字典 *wordList*，找出所有从 *beginWord* 到 *endWord* 的最短转换序列。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换后得到的单词必须是字典中的单词。

说明:

- 如果不存在这样的转换序列，返回一个空列表。
- 所有单词具有相同的长度。
- 所有单词只由小写字母组成。
- 字典中不存在重复的单词。
- 你可以假设 *beginWord* 和 *endWord* 是非空的，且二者不相同。

示例 1:

```
输入:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

输出:
[
  ["hit","hot","dot","dog","cog"],
  ["hit","hot","lot","log","cog"]
]
```

示例 2:

```
输入:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

输出: []

解释: endWord "cog" 不在字典中，所以不存在符合要求的转换序列。
```

思路：

其实总体思路和上面的一样，只不过，这次输出的转换的顺序

```
def findLadders(self, beginWord, endWord, wordList):
        """
        :type beginWord: str
        :type endWord: str
        :type wordList: List[str]
        :rtype: List[List[str]]
        """
        if endWord not in wordList or not endWord or not beginWord or not
            return 0

         L = len(beginWord)
         # 这一步显示构建一个字典,
         # {"*ot":["hit","dot","lot"],"h*t":["hot"],....}
         # 设置一个默认value为list的字典
         all_combo_dict = defaultdict(list)
```

```
        for word in wordList:
            for i in range(L):
                all_combo_dict[word[:i] + "*" + word[i+1:]].append(word)

        queue = [(beginWord, 1)]
        visited = {beginWord:True}
        res = []
        while queue:
            current_word,level = queue.pop(0)
            for i in range(L):
                intermediate_word = current_word[:i] + "*" + current_wor

                for word in all_combo_dict[intermediate_word]:
                    if word == endWord:
                        return res.append(word)
                    if word not in visited:
                        visited[word] = True
                        queue.append((word, level+1))
                        res.append(word)
                all_combo_dict[intermediate_word] = []
        return 0
```

这三道题，都是从wordList找到匹配字符，不通的是最小基因变化中，转换的字母只能从ACGT中选择，但是单词变化却不是