【哈希表】-单词接龙

视频直播讲解：2024/09/07 真题讲解（2024E卷）

题目描述与示例

题目描述

单词接龙的规则是：

可用于接龙的单词首字母必须要前一个单词的尾字母相同

当存在多个首字母相同的单词时，取长度最长的单词，如果长度也相等，则取字典序最小的单词；已经参与接龙的单词不能重复使用

现给定一组全部由小写字母组成单词数组，并指定其中的一个单词作为起始单词，进行单词接龙，

请输出最长的单词串，单词串是单词拼接而成，中间没有空格

输入描述

输入的第一行为一个非负整数，表示起始单词在数组中的索引 `K` ， `0 <= K < N`

输入的第二行为一个非负整数，表示单词的个数 `N` ；

接下来的 `N` 行，分别表示单词数组中的单词

备注：

单词个数 `N` 的取值范围为 `[1,20]` ；

单个单词的长度的取值范围为 `[1,30]`

输出描述

输出一个字符串，表示最终拼接的单词串

示例一

输入

```
```

```
0
6
word
dd
da
dc
dword
d
```

输出

```
worddwordda
```

说明

先确定起始单词`word`，再接以`d`开头的且长度最长的单词`dword`，剩余以`d`开头且长度最长的有`dd、da、dc`，则取字典序最小的`da`，所以最后输出`worddwordda`。

示例二

输入

```
4
6
word
dd
da
dc
dword
d
```

输出

```
dwordda
```

说明

先确定起始单词`dword`，剩余以`d`开头且长度最长的有`dd、da.`

`dc`，则取字典序最小的 `da`，所以最后输出 `dwordda`。

解题思路

题意理解

首先必须避免一个题目理解上的误区。

假设已知当前的已经接龙好的字符串为 `ans`，其最后一个字符为 `ans[-1]`。

那么下一个接龙的单词，是首字母为 `ans[-1]` 的**长度尽可能大、字典序尽可能小的那个单词**。

换句话说，对于已经接龙好的字符串为 `ans`，其下一个接龙的单词是**唯一确定的**。

> 此处可能存在另一种理解，就是接龙过程中选择单词的策略是不确定的，目的是使得最终接龙完毕的字符串最长。如果是这种理解，则示例二的答案会是 `dwordddda`，需要使用动态规划来完成，难度陡升。

同一首字母的单词列表排序

由于每一次接龙之后，下一个接龙的单词的首字母就是当前字符串的最后一个字母 `ans[-1]`。

假设以字母 `ans[-1]` 为开头的单词已经储存为一个列表 `lst`。

我们需要找到 `lst` 中，**长度尽可能大、字典序尽可能小的那个单词**。

可以使用 `lambda` 匿名函数对 `lst` 进行排序， `lst` 末尾的元素就是要延长的单词。

```
lst.sort(key = lambda x: (-len(x), x), reverse = True)
nxt_word = lst.pop()
ans += nxt_word
```

> 上述排序写法可能不容易理解，举个例子就好理解了。

> 对于以 `"a"` 为开头的若干单词 `lst = ["a", "ab", "ac", "abd", "abc"]`，我们期望其获得如下排序：
>
> `lst = ["a", "ab", "ac", "abc", "abd"]`
>
> `lst.sort(key = lambda x: (-len(x), x))` 的含义是，先按照长度从大到小排序，长度相等的时候再按照字典序从小到大排序。这样排序后，具有最高优先级的单词就排在 `lst` 的开头了。
>
> 为了使得具有最高优先级的单词排在 `lst` 的末尾，方便使用 `pop()` 操作取出，我们在排序的过程中再设置参数 `reverse = True`，即 `lst.sort(key = lambda x: (-len(x), x), reverse = True)`，这样就能达到排序要求。

以首字母分类单词构建哈希表

很显然，所有具有同一个首字母的单词都可以分为同一组，放在同一个列表中进行排序。

所以我们可以构建哈希表 `dic`， `key` 为首字母 `first_ch`， `value` 为以字母 `first_ch` 为首字母开头的若干单词构成的列表。对应的代码如下

```
from collections import defaultdict

# 输入起始索引
startIdx = int(input())
# 输入单词个数
n = int(input())

# 构建一个哈希表，用于按照单词首字母储存单词列表
# key为某一个首字母first_ch
# value为以字母first_ch为首字母的单词列表
dic = defaultdict(list)
# 初始化答案变量ans
ans = ""
# 循环n次，输入每一个单词并储存在哈希表dic中
for i in range(n):
    # 输入单词word
    word = input()
    # 如果是起始单词，则将word储存在ans中
    if i == startIdx:
        ans += word
    else:
        # 获得单词word的首字母
```

```
        first_ch = word[0]
        # 把word储存在哈希表dic中，
        dic[first_ch].append(word)
```

除此之外，`dic` 中的每一个列表都要再按照前一小点中的排序方案进行排序，即

```
# 需要对dic中，value储存的每一个单词列表进行排序
# 先按照单词长度从小到大排序，再按照字典序逆序排序
for first_ch in dic:
    dic[first_ch].sort(key = lambda x: (-len(x), x), reverse = True)
```

譬如对于示例二，构建出来的 `dic` 如下

```
dic = {
'w': ['word'],
'd': ['d', 'dd', 'dc', 'da', 'dword']
}
```

单词接龙模拟过程

在上述预处理过程完成之后，剩下就是按照题意进行单词接龙的模拟过程了。

显然这里的循环过程不应该使用 `for` 循环（因为不知道接龙次数），应该使用 `while` 循环。

当前字符串 `ans` 的最后一个字母 `ans[-1]` 是下一个单词的首字母，因此我们 `while` 持续进行的条件是，首字母 `ans[-1]` 在 `dic` 中还能找到对应的单词，即 `len(dic[ans[-1]]) > 0`。

在循环中，由于前面预处理已经将每一个单词列表都按照要求排序完毕，我们需要弹出列表 `dic[ans[-1]]` 中的最后一个单词，作为接下来延长的单词 `following_word = dic[ans[-1]].pop()`。

对 `ans` 进行延长就是直接 `ans += following_word`。

故代码为

```
# 进行while循环
# 退出循环的条件为，ans的末尾字母ans[-1]，在dic中对应的单词列表长度为0
# 即无法找到进一步单词接龙的单词
while len(dic[ans[-1]]) > 0:
    # 弹出dic[ans[-1]]的最后一个单词，作为接下来延长的单词
    following_word = dic[ans[-1]].pop()
    # 对ans进行延长
    ans += following_word
```

代码

Python

```
# 题目：2024E-单词接龙
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：哈希表/排序
# 代码看不懂的地方，请直接在群上提问


from collections import defaultdict

# 输入起始索引
startIdx = int(input())
# 输入单词个数
n = int(input())

# 构建一个哈希表，用于按照单词首字母储存单词列表
# key为某一个首字母first_ch
# value为以字母first_ch为首字母的单词列表
dic = defaultdict(list)
# 初始化答案变量ans
ans = ""
# 循环n次，输入每一个单词并储存在哈希表dic中
for i in range(n):
    # 输入单词word
    word = input()
```

```python
    # 如果是起始单词，则将word储存在ans中
    if i == startIdx:
        ans += word
    else:
        # 获得单词word的首字母
        first_ch = word[0]
        # 把word储存在哈希表dic中，
        dic[first_ch].append(word)


# 需要对dic中，value储存的每一个单词列表进行排序
# 先按照单词长度从小到大排序，再按照字典序逆序排序
# 譬如以'd'为首字母的单词列表应该排序为
# 'd' : ['d', 'dd', 'dc', 'da', 'dword']
for ch in dic:
    dic[ch].sort(key = lambda x: (-len(x), x), reverse = True)


# 进行while循环
# 退出循环的条件为，ans的末尾字母ans[-1]，在dic中对应的单词列表长度为0
# 即无法找到进一步单词接龙的单词
while len(dic[ans[-1]]) > 0:
    # 弹出dic[ans[-1]]的最后一个单词，作为接下来延长的单词
    following_word = dic[ans[-1]].pop()
    # 对ans进行延长
    ans += following_word

print(ans)
```

Java

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int startIdx = scanner.nextInt();
        int n = scanner.nextInt();

        // 读取换行符
        scanner.nextLine();

        HashMap<Character, List<String>> dic = new HashMap<>();
        StringBuilder ans = new StringBuilder();

```

```
        for (int i = 0; i < n; i++) {
            String word = scanner.nextLine();
            if (i == startIdx) {
                ans.append(word);
            } else {
                char firstCh = word.charAt(0);
                dic.putIfAbsent(firstCh, new ArrayList<>());
                dic.get(firstCh).add(word);
            }
        }

        for (List<String> words : dic.values()) {
            words.sort((a, b) -> {
                if (a.length() != b.length()) {
                    return Integer.compare(a.length(), b.length());
                } else {
                    return b.compareTo(a);
                }
            });
        }

        while (dic.get(ans.charAt(ans.length() - 1)) != null &&
!dic.get(ans.charAt(ans.length() - 1)).isEmpty()) {
            String followingWord = dic.get(ans.charAt(ans.length() -
1)).remove(dic.get(ans.charAt(ans.length() - 1)).size() - 1);
            ans.append(followingWord);
        }

        System.out.println(ans.toString());
    }
}

```

C++

```
#include <iostream>
#include <unordered_map>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    int startIdx;
    cin >> startIdx;
```

```cpp
    int n;
    cin >> n;
    cin.ignore();

    unordered_map<char, vector<string>> dic;
    string ans = "";

    for (int i = 0; i < n; i++) {
        string word;
        getline(cin, word);
        if (i == startIdx) {
            ans += word;
        } else {
            char firstCh = word[0];
            dic[firstCh].push_back(word);
        }
    }

    for (auto& entry : dic) {
        sort(entry.second.begin(), entry.second.end(), [](const string& a, const string& b) {
            if (a.length() != b.length()) {
                return a.length() < b.length();
            } else {
                return a > b;
            }
        });
    }

    while (!dic[ans.back()].empty()) {
        string followingWord = dic[ans.back()].back();
        dic[ans.back()].pop_back();
        ans += followingWord;
    }

    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(NlogM + N)`。排序所花费的时间复杂度为`O(N/M * MlogM)` = `O(NlogM)`，接龙过程的时间复杂度为`O(N)`。

空间复杂度：`O(N)`。哈希表所需要的额外空间。

`N`为单词个数，`M`是以某个字母为首字母的单词个数。