【回溯】-字符串拼接

题目描述与示例

题目描述

给定 `M (0<M<=30)` 个字符 `(a-z)`，从中取出任意字符(每个字符只能用一次)拼接成长度为 `N (0<N<=5)` 的字符串，要求相同的字符不能相邻，计算出给定的字符列表能拼接出多少种满足条件的字符串，输入非法或者无法拼接出满足条件的字符串则返回 `0` 。

输入描述

给定的字符列表和结果字符串长度，中间使用空格 `(" ")` 拼接

输出描述

满足条件的字符串个数

示例一

输入

```
1 abc 1
```

输出

```
1 3
```

说明

给定的字符为 `abc` ，结果字符串长度为 `1` ，可以拼接成 `a,b,c` ，共 `3` 种

示例二

输入

```
1 aabc 3
```

输出

```
1 8
```

说明

给定的字符为 `aabc` ，结果字符串长度为 `3` ，可以拼接成
`abc,acb,bac,bca,cba,cab,aba,aca` ，共 `8` 种

解题思路

本题数据量不大，虽然只是要求枚举数量而不是枚举具体的字符串，但仍然可以通过回溯来解决。

注意到在本题中，原输入字符串 `s` 中的字符顺序并不重要，只有出现次数是关键信息。

所以很容易考虑应该使用哈希表计数器来统计 `s` 中各个字符出现的次数。即

```
1 cnt = Counter(s)
```

回溯的重点在于回溯函数的编写。函数传参需要包含三个参数：

- `n`：目标字符串的长度
- `path`：当前所选择的路径字符串的情况
- `cnt`：原字符串 `s` 中的字符剩余情况

考虑递归终止条件，显然当 `len(path) == n` 时，我们找到了一个长度为 `n` 的拼接字符串，更新 `ans` 并退出递归，即

```
global ans
if len(path) == n:
    ans += 1
    return
```

考虑横向遍历过程，我们需要考虑 `cnt` 中的所有 `key` 以及这些 `key` 的 `value`。当

- `value == 0` 时，说明 `key` 这个字符已经在之前被选择过了，无法再选
- `(path and key == path[-1])` 时，说明此时 `key` 和当前 `path` 的最后一个字符一致，此时是不能够选择 `key` 延长在 `path` 的后面的。

对于上述两种情况，都可以直接跳过该 `(key, value)` 对。

当不满足上述情况时，则可以进行状态更新和回溯，以及回溯结束后的回滚操作。即

```
for k, v in cnt.items():
    if v == 0 or (path and k == path[-1]):
        continue
    cnt[k] -= 1
    dfs(cnt, path + k, n)
    cnt[k] += 1

```

综上整体的回溯函数为

```
def dfs(cnt, path, n):
    global ans
```

```
    if len(path) == n:
        ans += 1
        return
    for k, v in cnt.items():
        if v == 0 or (path and k == path[-1]):
            continue
        cnt[k] -= 1
        dfs(cnt, path + k, n)
        cnt[k] += 1
```

这就完成了本题的核心递归函数。

至于递归入口，则传入 `path = ""` 即可。

另外题目说明当出现非法输入时需要返回 `0` ，因此还需要判断 `s` 中的每一个字符是否均为小写字符。

> PS：感兴趣的同学可以思考一下如何使用dp完成这个问题

代码

Python

```python
# 题目：【回溯】2024E-字符串拼接
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：回溯
# 代码看不懂的地方，请直接在群上提问


from collections import Counter


# 回溯的函数
# path是列表或者字符串均可，这里path选择字符串
def dfs(cnt, path, n):
    global ans
    # path长度已经为n，找到了一个答案
    if len(path) == n:
```

```
        ans += 1
        return
    # 横向遍历，考虑字符k以及其剩余个数v
    for k, v in cnt.items():
        # 如果剩余个数为0，或者k和path前一个字符一样，则直接跳过k
        if v == 0 or (path and k == path[-1]):
            continue
        # 状态更新：选择k，所以k的的剩余个数-1
        cnt[k] -= 1
        # 回溯：path的末尾要加上k这个字符来作为新的path
        dfs(cnt, path + k, n)
        # 回滚：把选择的这个k还回去，所以k的剩余个数+1
        cnt[k] += 1

# 有可能存在输入非法的情况，所以使用try-except语句
try:
    # 输入字符串s，答案字符的长度n
    s, n = input().split()
    n = int(n)
    # 使用哈希表统计s种每一个字符出现的次数
    # 显然，s中字符的出现顺序不重要
    cnt = Counter(s)
    # 如果s中的字符出现非小写字母的情况（非法输入），输出0
    # 只有全为小写字母，才可以进行回溯过程
    if all("a" <= k <= "z" for k in cnt):
        ans = 0
        dfs(cnt, "", n)
        print(ans)
    else:
        print(0)
except:
    print(0)
```

Java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {
    static int ans = 0;

    // 回溯的函数
    static void dfs(Map<Character, Integer> cnt, StringBuilder path, int n) {
        // path长度已经为n，找到了一个答案
```

```
        if (path.length() == n) {
            ans++;
            return;
        }
        // 横向遍历，考虑字符k以及其剩余个数v
        for (char k : cnt.keySet()) {
            int v = cnt.get(k);
            // 如果剩余个数为0，或者k和path前一个字符一样，则直接跳过k
            if (v == 0 || (path.length() > 0 && k == path.charAt(path.length() - 1))) {
                continue;
            }
            // 状态更新：选择k，所以k的的剩余个数-1
            cnt.put(k, v - 1);
            // 回溯：path的末尾要加上k这个字符来作为新的path
            dfs(cnt, new StringBuilder(path).append(k), n);
            // 回滚：把选择的这个k还回去，所以k的剩余个数+1
            cnt.put(k, v);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        String[] parts = input.split(" ");
        String s = parts[0];
        int n = Integer.parseInt(parts[1]);
        Map<Character, Integer> cnt = new HashMap<>();
        for (char ch : s.toCharArray()) {
            cnt.put(ch, cnt.getOrDefault(ch, 0) + 1);
        }
        // 如果s中的字符出现非小写字母的情况（非法输入），输出0
        // 只有全为小写字母，才可以进行回溯过程
        boolean valid = true;
        for (char ch : cnt.keySet()) {
            if (ch < 'a' || ch > 'z') {
                valid = false;
                break;
            }
        }
        if (valid) {
            ans = 0;
            dfs(cnt, new StringBuilder(), n);
            System.out.println(ans);
        } else {
            System.out.println(0);
        }
```

```
    }
}

```

C++

```cpp
#include <iostream>
#include <string>
#include <unordered_map>

using namespace std;

int ans = 0;

// 回溯的函数
void dfs(unordered_map<char, int>& cnt, string path, int n) {
    // path长度已经为n，找到了一个答案
    if (path.length() == n) {
        ans++;
        return;
    }
    // 横向遍历，考虑字符k以及其剩余个数v
    for (auto& kv : cnt) {
        char k = kv.first;
        int v = kv.second;
        // 如果剩余个数为0，或者k和path前一个字符一样，则直接跳过k
        if (v == 0 || (!path.empty() && k == path.back())) {
            continue;
        }
        // 状态更新：选择k，所以k的的剩余个数-1
        cnt[k]--;
        // 回溯：path的末尾要加上k这个字符来作为新的path
        dfs(cnt, path + k, n);
        // 回滚：把选择的这个k还回去，所以k的剩余个数+1
        cnt[k]++;
    }
}

int main() {
    string input;
    getline(cin, input);
    string s;
    int n;
    size_t pos = input.find(' ');
    s = input.substr(0, pos);
```

```
    n = stoi(input.substr(pos + 1));
    unordered_map<char, int> cnt;
    for (char ch : s) {
        cnt[ch]++;
    }
    // 如果s中的字符出现非小写字母的情况（非法输入），输出0
    // 只有全为小写字母，才可以进行回溯过程
    bool valid = true;
    for (auto& kv : cnt) {
        char ch = kv.first;
        if (ch < 'a' || ch > 'z') {
            valid = false;
            break;
        }
    }
    if (valid) {
        ans = 0;
        dfs(cnt, "", n);
        cout << ans << endl;
    } else {
        cout << 0 << endl;
    }
    return 0;
}

```

时空复杂度

时间复杂度：`O(NT^N)`。`T`为字符集的大小，最多为`26`。`N`为目标字符串的长度。状态树的高度为`N`，叶子节点的数量最多为`T^N`。

空间复杂度：`O(N)`。为状态树的高度。