24程设I-周11-课后2

题目描述

字符串全排列

给定一个字符串 `str`，要求你输出该字符串的所有不同的排列。

输入要求：

- 输入一个字符串 `str`，长度不超过 5，且只包含英文字母（大小写均可）。注：可能会出现某些字符重复如 `JWXww`。

输出要求：

- 输出该字符串的所有不同排列，按照字典序升序排列，且不出现重复的字符串。注：要求如cab在cba之前。

参考代码

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// 用于存储结果的动态数组
char **result;
int resultSize;

// 交换两个字符
void swap(char *a, char *b) {
    char temp = *a;
    *a = *b;
    *b = temp;
}

// 深度优先搜索函数
void dfs(char *s, int index) {
    if (index == strlen(s)) {
        result[resultSize] = strdup(s);
        resultSize++;
        return;
    }
    char used[128] = {0}; // 标记当前层已经使用过的字符，避免重复
    for (int i = index; i < strlen(s); i++) {
        if (used[(int)s[i]]) continue; // 跳过重复字符
        used[(int)s[i]] = 1;           // 标记字符已使用
        swap(&s[index], &s[i]);        // 交换字符
        dfs(s, index + 1);             // 递归处理下一个位置
        swap(&s[index], &s[i]);        // 回溯
    }
}

// 比较函数，用于 qsort
int compare(const void *a, const void *b) {
    return strcmp(*(const char **)a, *(const char **)b);
}
```

```c
// 主函数
char **permutation(char *s, int *returnSize) {
    int n = strlen(s);
    result = (char **)malloc(sizeof(char *) * 120); // 5! 最大排列数
    resultSize = 0;

    dfs(s, 0);

    // 对结果数组进行排序
    qsort(result, resultSize, sizeof(char *), compare);

    *returnSize = resultSize;
    return result;
}

int main() {
    char s[5];
    scanf("%s", &s);
    int returnSize;
    char **permutations = permutation(s, &returnSize);

    for (int i = 0; i < returnSize; i++) {
        printf("%s\n", permutations[i]);
        free(permutations[i]);
    }
    free(permutations);
    return 0;
}
```

代码解析

1. 全局变量

```c
char **result;      // 存储所有排列结果的动态数组
int resultSize;     // 当前存储结果的个数
```

- `result` 用于存储生成的所有排列，使用动态内存分配（`malloc`），以适应不同输入字符串长度。
- `resultSize` 记录当前生成排列的数量。

2. 字符交换函数

```c
void swap(char *a, char *b) {
    char temp = *a;
    *a = *b;
    *b = temp;
}
```

- 用于交换两个字符的位置。
- 在排列算法中，每次尝试将当前索引位置的字符换成其他字符，从而生成新排列。

3. 深度优先搜索函数

```
void dfs(char *s, int index) {
    if (index == strlen(s)) {
        result[resultSize] = strdup(s);  // 复制当前排列
        resultSize++;                    // 增加结果数量
        return;
    }

    char used[128] = {0};  // 用于标记当前层中已经使用过的字符
    for (int i = index; i < strlen(s); i++) {
        if (used[(int)s[i]]) continue;  // 如果字符已经在当前层用过，则跳过
        used[(int)s[i]] = 1;            // 标记字符已使用
        swap(&s[index], &s[i]);         // 将字符固定在当前位置
        dfs(s, index + 1);              // 递归处理剩余字符
        swap(&s[index], &s[i]);         // 回溯，恢复原字符串
    }
}
```

深度优先搜索工作流程

1. 递归基础条件:
    - 当 `index == strlen(s)` 时，说明已经生成了一个完整的排列，将其保存到 `result` 中。
2. 去重逻辑:
    - `used` 数组用于标记当前递归层中已经用过的字符，避免重复排列。例如，当有重复字符 `aa` 时，第二个 `a` 在同一层不能再被处理。
    - `used` 数组的大小为 128 是因为在 ASCII 编码表中，字符的值范围是 `0` 到 `127`，总共有 128 个可能的字符。
3. 回溯逻辑:
    - 使用 `swap` 交换字符，将当前排列恢复到上一状态，以便处理其他分支。
4. 复杂度:
    - 每个字符固定后有 `n!` 种排列，且 `used` 数组检查的复杂度为 O(1)，整体效率较高。

4. 排序函数

```
int compare(const void *a, const void *b) {
    return strcmp(*(const char **)a, *(const char **)b);
}
```

- `compare` 是一个比较函数，用于 `qsort` 按字典顺序排列字符串。
- `strcmp`

  比较两个字符串的字典序：
    - 返回负值：第一个字符串小于第二个字符串。
    - 返回正值：第一个字符串大于第二个字符串。
    - 返回 0：两者相等。

5. 主排列函数

```c
char **permutation(char *s, int *returnSize) {
    int n = strlen(s);
    result = (char **)malloc(sizeof(char *) * 120); // 5! 最大排列数
    resultSize = 0;

    dfs(s, 0); // 调用深度优先搜索生成所有排列

    qsort(result, resultSize, sizeof(char *), compare); // 排序结果

    *returnSize = resultSize; // 返回排列个数
    return result;
}
```

关键点

1. 动态分配内存:
   - 分配 `120` (8!) 空间，因为字符串长度最大为 5，最多有 5! 个排列。
   - 这里事实上并不需要动态分配内存，但掌握该种思想是极为重要的。
2. 调用递归函数:
   - `dfs(s, 0)` 开始递归，固定每一层的字符。
3. 排序结果:
   - 使用 `qsort` 将所有排列按字典序排序。
4. 返回结果:
   - `returnSize` 返回生成的排列数量。

6. 主程序

```c
int main() {
    char s[5];
    scanf("%s", &s);
    int returnSize;
    char **permutations = permutation(s, &returnSize);

    for (int i = 0; i < returnSize; i++) {
        printf("%s\n", permutations[i]); // 打印每个排列
        free(permutations[i]);           // 释放每个排列的内存
    }
    free(permutations); // 释放存储排列的动态数组
    return 0;
}
```

关键点

1. 输入字符串:
   - 示例使用 `abc`，可以修改为其他字符串测试。
2. 结果打印:
   - 遍历排列结果数组，将每个字符串输出。
3. 内存释放:

- 使用 `free` 释放动态分配的内存，避免内存泄漏。