14.最长公共前缀

题目大意：
找不同字符串里的公共部分并输出

分析

把第一个当匹配串和第二个匹配找公共的部分，然后把公共的部分放到第三个、第四个、以此类推最后剩下的就是公共前缀

代码

```
public:
    // 主函数：寻找字符串数组的最长公共前缀
    string longestCommonPrefix(vector<string>& strs) {
        // 边界条件：如果数组为空，返回空字符串
        if (!strs.size()) {
            return "";
        }

        // 初始化前缀为第一个字符串
        string prefix = strs[0];
        int count = strs.size();

        // 遍历数组中的每个字符串
        for (int i = 1; i < count; ++i) {
            // 不断更新前缀，使其成为当前前缀和下一个字符串的公共前缀
            prefix = longestCommonPrefix(prefix, strs[i]);

            // 如果前缀为空，说明没有公共前缀，直接跳出循环
            if (!prefix.size()) {
                break;
            }
        }

        // 返回最终的最长公共前缀
        return prefix;
```

```
    }

    // 辅助函数：寻找两个字符串的最长公共前缀
    string longestCommonPrefix(const string& str1, const string&
str2) {
        // 取两个字符串中较短的长度
        int length = min(str1.size(), str2.size());
        int index = 0;

        // 逐个字符比较，直到找到不同的字符或到达字符串末尾
        while (index < length && str1[index] == str2[index]) {
            ++index;
        }

        // 返回从开始到不同字符位置的子字符串
        return str1.substr(0, index);
    }
```

方法二

分析

除了可以横向扫描字符串
还可以把所有字符串对其然后纵向比对，如果一列的字符都相同更新公共字符串；如果有不同的，就停止遍历，返回之前遍历的结果

代码

```
class Solution{
public:
    string longestCommonPrefix(vector<string>& strs){
        //先检查字符串数组是否为空
        if(!strs.size()){
            return "";
        }
        int length = strs[0].size();
        int count = strs.size();
```

```
        //遍历
        for(int i = 0;i < length;++i){
            char c = strs[0][i];
            for(int j = 1;j < count;++j){
                //先加j
                if(i == strs[j].size()||strs[j][i]!=c){
                    return strs[0].substr(0,i);
                }
            }
        }
        return strs[0];
    }
};
```

方法三

分析

通过分治的方法，两个字符一组，分解成两两比对公共字符串，最后再将结果进行比较，找出公共的字符串

代码

```
class Solution{
public:
    string longestCommonPrefix(vector<string>& strs){
        if(!strs.size()){
            return "";
        }
        else{
               return longestCommonPrefix(strs,0,strs.size()-1);
        }
    }
    string longestCommonPrefix(const vector<string>& strs,int start,int end){
        if(start == end){
            return strs[start];
        }
        else{
            int mid = (start+end)/2;
```

```
            string lcpleft =
longestCommonPrefix(strs,start,mid);
            string lcpright =
longestCommonPrefix(strs,mid+1,end);
            return commonPrefix(lcpleft,lcpright);
        }
    }
    string commonPrefix(const string& lcpleft,const string&
lcpright){
        int minlength = min(lcpleft.size(),lcpright.size());
        for(int i=0;i<minlength;++i){
            if(lcpleft[i]!=lcpright[i]){
                return lcpleft.substr(0,i);
            }
        }
        return lcpleft.substr(0,minlength);
    }
};
```

方法五

分析

二分查找，以第一串字符为例，分一半各找公共前缀，最后合并。

最长公共前缀的长度不会超过字符串数组中的最短字符串的长度。用 minLength 表示字符串数组中的最短字符串的长度，则可以在 [0,minLength] 的范围内通过二分查找得到最长公共前缀的长度。每次取查找范围的中间值 mid，判断每个字符串的长度为 mid 的前缀是否相同，如果相同则最长公共前缀的长度一定大于或等于 mid，如果不相同则最长公共前缀的长度一定小于 mid，通过上述方式将查找范围缩小一半，直到得到最长公共前缀的长度。

（感觉有点隔路）

代码

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (!strs.size()) {
```

```
            return "";
        }
        int minLength = min_element(strs.begin(), strs.end(), []
(const string& s, const string& t) {return s.size() <
t.size();})->size();
        int low = 0, high = minLength;
        while (low < high) {
            int mid = (high - low + 1) / 2 + low;
            if (isCommonPrefix(strs, mid)) {
                low = mid;
            }
            else {
                high = mid - 1;
            }
        }
        return strs[0].substr(0, low);
    }

    bool isCommonPrefix(const vector<string>& strs, int length)
{
        string str0 = strs[0].substr(0, length);
        int count = strs.size();
        for (int i = 1; i < count; ++i) {
            string str = strs[i];
            for (int j = 0; j < length; ++j) {
                if (str0[j] != str[j]) {
                    return false;
                }
            }
        }
        return true;
    }
};
```