14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

**Tags: String**

class Solution {

public:

string longestCommonPrefix(vector<string>& strs) {

int i, j;

int minLen;

int strLen;

int size = strs.size();

char c;

bool flag;

if (size == 0)

return "";

minLen = strs[0].size();

for (i = 1;i < size;i++) {

strLen = strs[i].size();

if (strLen < minLen)

minLen = strLen;

}

for (j = 0;j < minLen;j++) {

flag = false;

c = strs[0][j];

for (i = 1;i < size;i++) {

if (strs[i][j] != c) {

flag = true;

break;

}

}

if (flag)

break;

}

return strs[0].substr(0, j);

}

};

**解题思路：**

1. 找到最短的字符串长度minLen
2. i从0到minLen-1，对于个字符串的第i个位置，当该字符与第一个字符串的第i个字符不相等时，结束程序，最长公共前缀的最大长度恰好为i
3. 空间复杂度： O(1)， 时间复杂度O(n)

20. Valid Parentheses

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

**Tags: Stack String**

class Solution {

public:

bool isValid(string s) {

stack<char> stack;

unordered\_map<char, char> map({ {'(',')'},{ '[',']' },{ '{','}' } });

int strlen = s.size();

if (strlen == 1)

return false;

int i = 0;

while (i < strlen) {

if (!stack.empty() && map[stack.top()] == s[i])

stack.pop();

else

stack.push(s[i]);

i++;

}

if (!stack.empty())

return false;

return true;

}

};

**解题思路：**

1. 当遍历完整个字符串时程序结束
2. 首先，初始化空栈；
3. 判断条件：当栈为空时，字符直接入栈；当栈不为空时，栈顶字符与新来的字符进行匹配，如果能匹配上，如‘（’和‘）’，则栈顶字符出栈，继续判断下一个；如果不匹配，则新来的字符直接入栈。
4. 最终判断字符串是否合法的判别条件：栈是否为空。如果栈为空，则说明合法；否则，不合法。
5. 时间复杂度：O(n)，空间复杂度：O(n)

22. Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

"((()))", "(()())", "(())()", "()(())", "()()()"

**Tags: Backtracking String**

class Solution {

public:

vector<string> generateParenthesis(int n) {

vector<string> vec;

generate(vec, n, n, "");

return vec;

}

void generate(vector<string>& vec, int L, int R, string str) {

if (L == 0 && R == 0) {

vec.push\_back(str);

return;

}

if (L > 0) {

generate(vec, L - 1, R, str + "(");

}

if (L < R) {

generate(vec, L, R - 1, str + ")");

}

}

};

**解题思路：**

1. 对于合法的字符串，都满足一个规则：左括号的个数大于等于右括号的个数。
2. 所以，我们用L表示剩余的左括号的个数，R表示剩余的右括号的个数。
3. 当L>0时，我们可以直接打印左括号，而且不会违背规则；而能否打印右括号，则需要进行判断，如果L>=R，那么我们是不能打印右括号的，因为这意味着右括号的数量多于或者等于左括号的数量，如果再打印右括号会违背1中所述规则，因此只能在L<R的时候打印右括号。
4. 因此就可以生成如代码所述的递归规则，并以L=R=0作为结束条件。
5. 复杂度（生成节点的个数）：1/(n+1)\*C(2n,n) （Catalan数）