字符串算法

1、KMP算法

字符串匹配算法

```

class Solution {

public:

void getNext(int\* next, const string& s) {

int j = 0;

next[0] = 0;

for (int i = 1; i < s.size(); i++) {

while (j > 0 && s[i] != s[j]) {

j = next[j - 1];

}

if (s[i] == s[j]) {

j++;

}

next[i] = j;

}

}

bool kmp(string haystack, string needle) {

if (needle.size() == 0) return 0; //排除特殊情况

int next[needle.size()];

getNext(next, needle); //1.获取next数组

int j = 0;

for (int i = 1; i < haystack.size(); i++) {

while (j > 0 && haystack[i] != needle[j]) { // 不匹配

j = next[j - 1]; // j寻找之前匹配的位置

}

if (haystack[i] == needle[j]) { // 匹配，j和i同时向后移动

j++;

}

if (j == needle.size()) { //子串匹配完成

if (i == haystack.size() - 1) { //但是遍历完了主串

return false;

}

return true;

}

}

return false;

}

bool repeatedSubstringPattern(string s) {

return kmp(s + s, s);

}

};

```

2、马拉车算法

回文串：求最大回文子串

```

class Solution {

public:

int expand(const string& s, int left, int right) {

while (left >= 0 && right < s.size() && s[left] == s[right]) {

--left;

++right;

}

return (right - left - 2) / 2;

}

string longestPalindrome(string s) {

int start = 0, end = -1;

string t = "#";

for (char c: s) {

t += c;

t += '#';

}

t += '#';

s = t;

vector<int> arm\_len;

int right = -1, j = -1;

for (int i = 0; i < s.size(); ++i) {

int cur\_arm\_len;

if (right >= i) {

int i\_sym = j \* 2 - i;

int min\_arm\_len = min(arm\_len[i\_sym], right - i);

cur\_arm\_len = expand(s, i - min\_arm\_len, i + min\_arm\_len);

} else {

cur\_arm\_len = expand(s, i, i);

}

arm\_len.push\_back(cur\_arm\_len);

if (i + cur\_arm\_len > right) {

j = i;

right = i + cur\_arm\_len;

}

if (cur\_arm\_len \* 2 + 1 > end - start) {

start = i - cur\_arm\_len;

end = i + cur\_arm\_len;

}

}

string ans;

for (int i = start; i <= end; ++i) {

if (s[i] != '#') {

ans += s[i];

}

}

return ans;

}

};

```