中山大学计算机学院

算法设计与分析

本科生实验报告

（2023学年秋季学期）

课程名称：Algorithm design and analysis

教学班级	专业(方向)	学号	姓名
计科二班	计算机科学与技术	21307185	张礼贤

实验题目与要求

- **给定字符串，找出其中最长的重复出现并且非重叠的子字符串**

  例子:

  输入:str ="geeksforgeeks'

  输出:geeks

  输入:str="aabaabaaba'

  输出:aaba

算法设计

1.直接遍历

算法原理

首先拿到题目，最容易想到的方法就是直接进行遍历操作，先设定 str1 的开始位置 i ，结束位置 j ，再设定 str2 的开始位置 j + 1 与结束位置k + j - i + 1，注意边界的问题

代码实现

```cpp
#include<iostream>
#include<string.h>
#include<vector>

using namespace std;
string dpLRS(string s)
{
    int length = s.size();
    int Max = 0;
    string res = "";
    for(int i=0;i<length;i++)
    {
        for(int j=i;j<length;j++)
        {
            int sub_size = j - i + 1;
            string str1 = s.substr(i,sub_size);
            for(int k=j+1;k<length;k++)
            {
                string str2 = s.substr(k,sub_size);
                if(str1 == str2 && str1.size() > Max)
                {
                    res = str1;
                    Max = str1.size();
                }
            }
        }
    }
    return res;

}

int main()
{
    cout<<"Please input the string:\n";
    string s;
    cin>>s;
    cout << "The longest repeating substring is: " <<
dpLRS(s) << endl; //输出答案
    return 0;
}
```

实验结果与复杂度分析

- 实验结果展示

```
Please input the string:
geeksforgeeks
The longest repeating substring is: geeks
PS D:\桌面\算法分析与设计> cd "d:\桌面\算法分析与设计\"
Please input the string:
aabaabaaba
The longest repeating substring is: aaba
PS D:\桌面\算法分析与设计>
```

  根据上面的代码，所显示的结果如上，正确地通过了上面的测试

- 复杂度分析

  由于是暴力算法，总共三重循环，总的时间复杂度为 $O(n^3)$。空间复杂度为 $O(1)$

2.动态规划

算法原理

由于暴力算法的时间复杂度过高，因此我们考虑使用动态规划来优化时间复杂度。 考虑案例：str = abcdabcdab

- 首先我们构建二维dp数组如下：

		a	b	c	d	a	b	c	d	a	b
	i/j	0	1	2	3	4	5	6	7	8	9
a	0	0	0	0	0	0	0	0	0	0	0
b	1	0	0	0	0	0	0	0	0	0	0
c	2	0	0	0	0	0	0	0	0	0	0
d	3	0	0	0	0	0	0	0	0	0	0

		a	b	c	d	a	b	c	d	a	b
a	4	0	0	0	0	0	0	0	0	0	0
b	5	0	0	0	0	0	0	0	0	0	0
c	6	0	0	0	0	0	0	0	0	0	0
d	7	0	0	0	0	0	0	0	0	0	0
a	8	0	0	0	0	0	0	0	0	0	0
b	9	0	0	0	0	0	0	0	0	0	0

- 遍历过程中，i 从 0 开始遍历， j 从 i + 1 处开始遍历，避免与自己进行匹配，当遍历到s[i] = s[j]时，一共有两种情况要讨论：

  1. 当 i = 0 的时候，由于只有一个字符，因此直接将 dp[i][j] + 1，也就是为 1
  2. 当 i > 0 的时候，由于前面的 dp[i-1][j-1] 已经记忆了前面的匹配情况，因此利用前面的情况进行更新。由于dp[i][j]即为重复字符串的长度，我们可以知道，第一个字符串的终点为s[i]，而第二个字符串的起点即为dp[j - dp[i][j] + 1]，因此根据第二个字符串的起点要大于i的条件，可得当dp[i][j] > j-i 时，不符合条件，此时将dp[i][j]更新为1，因为只有当前位置的相等字符满足。

  $$dp[i][j]=\begin{cases}1, & 当\ i=0\\ dp[i-1][j-1]+1, & 当\ i>0\end{cases}$$

  之后再对$\mathrm{dp}[\mathrm{i}][\mathrm{j}]$ 进行约束性检查：$dp[i][j]=1, if \quad dp[i][j]>j-i$

- 因此，根据上面的递推公式，我们可以得到上面的初始化dp矩阵的最终结果为

		a	**b**	**c**	**d**	**a**	**b**	**c**	**d**	**a**	**b**
	i/j	0	1	2	3	4	5	6	7	8	9
a	0	0	0	0	0	**1**	0	0	0	**1**	0
b	1	0	**1**	0	0	0	**2**	0	0	0	**1**
c	2	0	0	**2**	0	0	0	**3**	0	0	0
d	3	0	0	0	**3**	0	0	0	**4**	0	0
a	4	0	0	0	0	**4**	0	0	0	**1**	0
b	5	0	**1**	0	0	0	**1**	0	0	0	**2**
c	6	0	0	**2**	0	0	0	**2**	0	0	0
d	7	0	0	0	**3**	0	0	0	**3**	0	0
a	8	0	0	0	0	**4**	0	0	0	**4**	0
b	9	0	0	0	0	0	**1**	0	0	0	**1**

- 对于答案的获取，直接利用临时字符串保存即可，将其初始化为空字符串，如果遍历得到的子字符串大于临时字符串长度，则进行更新，最后的结果即为答案

代码实现

```
string dpLRS(string s)
{
    string ans = "";
    int Max = 0;
    int length = s.size();
    vector<vector<int>> dp(length, vector<int>(length));
    for(int i=0; i<length; i++)
    {
        for(int j=i+1; j<length; j++)  //从i+1开始遍历
        {
            if(s[i] == s[j])    //若两字符相等
            {
                dp[i][j] = i == 0 ? 1 : dp[i-1][j-1] + 1;
//状态转移方程
                if(dp[i][j] > j-i)  dp[i][j] = 1;
                if(dp[i][j] > Max) //更新答案
                {
                    Max = dp[i][j];
                    ans = s.substr(i-Max+1, Max);
                }
            }
        }
    }
    return ans;
}

int main()
{
    string s,ans;
    cout << "Please input the string:\n";
    cin >> s;
    ans = dpLRS(s);
    cout << "The longest repeating substring is: " << ans
<< endl; //输出答案
    return 0;
}
```

实验结果与复杂度分析

- 实验结果展示：

```
Please input the string:
geeksforgeeks
The longest repeating substring is: geeks
PS D:\桌面\算法分析与设计> cd "d:\桌面\算法分析与设计\"
Please input the string:
aabaabaaba
The longest repeating substring is: aaba
PS D:\桌面\算法分析与设计> cd "d:\桌面\算法分析与设计\"
Please input the string:
abcdabcdab
The longest repeating substring is: abcd
PS D:\桌面\算法分析与设计>
```

  可以看到，三个测试样例都通过了，说明实验结果良好，代码功能得到实现

- 复杂度分析： 对于上面的动态规划，时间复杂度为遍历二维数组的复杂度，即为 $O(n^2)$,空间复杂度为 $O(n^2)$