中山大学计算机学院算法设计与分析本科生实验报告

课程名称：算法设计与分析

教学班级	专业（方向）	学号	姓名
2班	计算机科学与技术	21307174	刘俊杰

一、实验内容

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

二、算法思想

1、蛮力算法

可以直接使用蛮力算法，直接对字符串进行遍历第一个子字符串的起始位置i、第二个子字符串的起始位置j和长度l查找的方法:

假设输入的字符串为str

1. 首先遍历确定选择的第一个子字符串str1的起始位置i。
2. 再在[i+1, len(str)]中遍历第二个子字符串(即可能与第一个子字符串重复的子字符串)的起始位置j。
3. 接着遍历子字符串的长度，在[1, len(str-i-1)]中遍历l。
4. 若j+l-1 >= len(str)则说明越界，或者 i+l-1>=j则说明重叠，就不考虑此次的第二个子字符串。
5. 若两个子字符串相等，则与记录的最长的重复出现并且非重叠的子字符串的长度比较，若大于，则更新最长长度max_len和最长的重复出现并且非重叠的子字符串ans。

6. 三次循环结束后的ans便是str中最长的重复出现并且非重叠的子字符串。

蛮力算法源代码

```
def findRepeatedSubstr(str):#找出str中最长的重复出现并且非重叠的子字符串
    max_len,ans = 0, ""
    for i in range(len(str)):
        for j in range(i+1,len(str)):
            for l in range(1,len(str)-i-1):
                if j+l-1 >= len(str):#越界
                    continue
                if i+l-1>=j:#重叠
                    continue
                if str[i:i+l] == str[j:j+l]:
                    if max_len < l:
                        max_len = l
                        ans = str[i:l]
    return ans


def main():
    str = input("请输入字符串:")
    ans = findRepeatedSubstr(str)
    #输出结果
    print("最长且无重叠重复子字符串:",ans)

if __name__ == "__main__":
    main()
```

蛮力算法的复杂度

时间复杂度为O(n^3)(不考虑子字符串的比较操作)

空间复杂度也为O(1)

2、动态规划

实现思想

可以使用动态规划来解决此问题,用 **dp[i][j]** 的二维数组来记录以str中第i个字符结尾的第一个子字符串与str中第j个字符结尾的第二个子字符串最长重复的长度。

dp[i][j] 的推导有两种情况:

1. 若str[i-1]==str[j-1],则说明第i个字符与第j个字符相等，此时以str中第i个字符结尾的第一个子字符串与str中第j个字符结尾的第二个子字符串最长重复的长度等于以str中第i-1个字符结尾的第一个子字符串与str中第j-1个字符结尾的第二个子字符串最长重复的长度加1。
2. 若不相等，则第i个字符与第j个字符不相等，以str中第i个字符结尾的第一个子字符串与str中第j个字符结尾的第二个子字符串最长重复的长度等于0.

于是推导出递推公式为:

$$dp[i][j] = \begin{cases} dp[i-1][j-1]+1, str[i-1] == str[j-1] \\ 0, otherwise \end{cases}$$

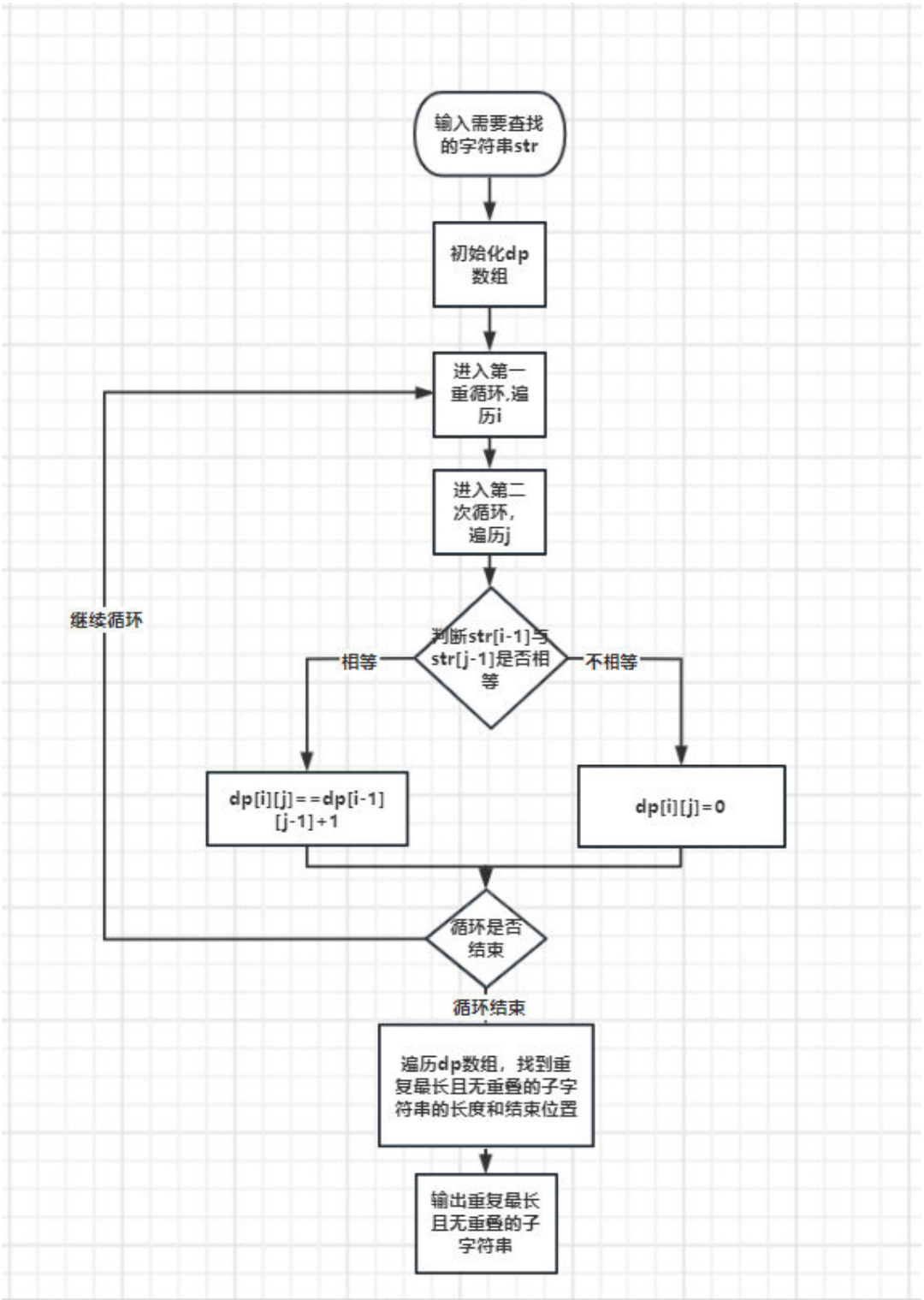
同时因为没有第0个字符，所以就没有重复的子字符串

于是边界条件为:

$$dp[0][j] = 0, j \in [0, len(str)]$$
$$dp[i][0] = 0, i \in [0, len(str)]$$

同时题目中要求不能重叠，于是设置j从i+1开始遍历推导，并且在最后遍历**dp**数组寻找最大长度时若**j-dp[i][j]+1 <= i <= j**，则说明出现了重叠，不考虑此个**dp[i][j]**。

流程图


输入需要查找的字符串str
初始化dp数组
进入第一重循环,遍历i
进入第二次循环，遍历j
判断str[i-1]与str[j-1]是否相等
相等
不相等
dp[i][j]==dp[i-1][j-1]+1
dp[i][j]=0
循环是否结束
继续循环
循环结束
遍历dp数组，找到重复最长且无重叠的子字符串的长度和结束位置
输出重复最长且无重叠的子字符串

三、源代码

```
def findRepeatedSubstr(str):#找出str中最长的重复出现并且非重叠的子字符串
    # 初始化
    dp = [[0] * (len(str)+1) for i in range(len(str)+1)]
    # 动态规划
    for i in range(1,len(str)+1):
        for j in range(i+1,len(str)+1):#j从i+1开始遍历
            if str[i-1] == str[j-1] :
                dp[i][j] = dp[i-1][j-1]+1
            else:
                dp[i][j] = 0

    #从dp数组中寻找str中最长的重复出现并且非重叠的子字符串的长度与结束位置
    max_len , end = 0, 0
    for i in range(len(str)+1):
        for j in range(len(str)+1):
            # 有重叠部分
            if j-dp[i][j]+1 <= i <= j :
                continue
            # 无重叠
            if dp[i][j] > max_len:
                max_len = dp[i][j]
                end = i-dp[i][j]
    return max_len,end

def main():
    str = input("请输入字符串:")
    max_len,end = findRepeatedSubstr(str)
    #输出结果
    print("最长且无重叠重复子字符串:",str[end:end+max_len])

if __name__ == "__main__":
    main()
```

四、输出截图

例子1:

例子2:

五、实验分析

动态规划

动态规划的时间主要在**dp**数组的两层循环中，所以时间复杂度为O(n^2)。

空间则耗费在**dp**数组，故空间复杂度也为O(n^2)。