Introduction To The Theory Of Computation Michael Sipser
计算理论导论

LAB B	实验 B—-CFG 是 P 成员
Name	屈德林
Student No.	201808010522
Class	计算机科学与技术 1805
Department	CSEE
Email	qdl.cs@qq.com
Date	2021 年 5 月 15 日

湖南大學

目录

1 Problem description

上下文无关文法 CFG G 是否派生某个串 W。采用动态规划（Dynamic Programming）设计一个一个多项式时间的验证算法。
实验方法：编写一个算法/程序，对于给定的输入，可以在多项式时间内判定 ACFG。
实验结果: 交一个程序验证。

1.1 Input

输入第一行为一个正整数 n，接下来 n 行为一个满足乔姆斯基范式的文法描述。然后一个正整数 m，接下来 m 行为 m 个小写字母组成的字符串 (长度小于 100) 表示 m 个待测试的串。

1.2 Output

对于每一个测试串返回”yes” 或者”no”，表示该串是否能被文法派生出来。

1.3 Sample Input

4
S->AB
A->AB|a
B->BC|b
C->CA|CC|c
3
ab
ac
bc

1.4 Sample Output

yes
no
no

2 Lab Environment 环境

- 操作系统：Arch Linux
- 程序运行环境：gcc (GCC) 10.2.0

- 报告编写环境：TeX Live 2020
- 开发工具：VSCode

3 Lab Steps 步骤

3.1 分析问题

1. 思路 1：拿到这道题首先想到的是用 STL 的结合 set 容器，可以快速判断一个字符串是否与其他的字符在同一个集合，遍历这个字符穿，判断是否存在于同一个集合，从而可以得出这个字符串是否能够被派生。Pass 掉了，因为题目要求用动态规划。

2. 思路 2：动态规划法，对于一个字符串 str，我们可以用 dp[j][k] 来表示这个字符串从第 j 个字母到第 k 个字母被派生，dp[j][k] 中存储一个字符串，如果 str[j:k] 可以被派生, 则这个字符串的所有字母存储在 dp[j][k] 中。对于一个字符串 str[j:k], 在 j…k 中取到一个分割点 r，如果 str[j:r] 和 str[r:k] 可以被派生，那么必须满足一个 S->BC 类型的乔姆斯基范式，其中 B 必须在 dp[j][r] 中找到，C 必须在 dp[r][k] 中找到。

3.2 算法思想

$$dp[j][k]+=\begin{cases} S,B\in dp[r][k],j\leq r<k,S->BC \\ s,j=k,s->a \end{cases}$$

需要注意的是，问题中给了 m 个测试字符串，所以我们要增加一个维度来描述动态规划过程，因而 dp 数组变化为 dp[i][j][k], 其中 i 代表第 i 个测试字符串，j 代表待测试字符串的起始位置，k 代表其结束位置。

1. 首先将输入的状态转移矩阵保存在 S 数组中，其中其中 S[i][j] 表示第 i 行第 j 列，意义为状态 i 经过字母 j 到达状态 S[i][j];

2. 对每一个输入的串 W, 从 after（after 表示每次转换后的状态，初始为起始状态）开始，按照每一个字符，得到相应的后继状态，保存在 after 中。

3. 最后判断 accept[after] 的值，即串在 DFA 上运行之后最终状态是否可接受。

3.2.1 算法伪代码表述

经过上述分析，算法伪代码可以表述为：

```
int main()
{
    //找出所有A->b型规则
    loop:
    if(cfg[i][k]>='a'&&cfg[i][k]<='z')
        sa[num]+=cfg[i][0];//S->AB|a,记录S
        sa[num]+=cfg[i][k];//记录S->a
        num++;

    //找出所有A->BC型规则
    loop:
    if(cfg[i][k]<'a')
        SAB[num2]+=cfg[i][0];//记录S
        SAB[num2]+=cfg[i][k];//记录AB
        SAB[num2]+=cfg[i][k+1];

    //考察每一个长为1的子串
    loop:
    if(str[i][j]==sa[k][1]) //考察CFG文法的每一个字符
        dp[i][j][j]=sa[k][0];//记录dp[i][j][j]=sa[k][0]起始位置

    //考察l长度的子串
    loop:
    if(dp[i][p][k].find(SAB[q][1])!=-1&&dp[i][k+1][j].find(SAB[q][2])!=-1)
        dp[i][p][j]+=SAB[q][0];

    //检查起始变元是否在dp[0][n]中
    for(int i=0;i<m;i++)
        if(dp[i][0][lstr[i]-1].find(cfg[0][0])!=-1) printf("yes\n");
        else printf("no\n");
}
```

3.3 算法复杂度分析

- 时间复杂度：由于在计算时，我们用了一个 5 重循环，并且在循环内部用了 find 函数搜索，我们假设所以复杂度为 $O(mlen(i)(len(i)-i)jnum2length)$, 我们假设

len(i)=C1,length=C2，num2 的最大值是 n，综上时间复杂度为 $T(n) = O(C_1^2 C_2 mnl)$, 为一个多项式时间。

- 空间复杂度：主要来自于辅助数组的存储，dp 需要空间 mC_12C_2, 所以空间复杂度为 $S(n) = O(m * C_1^2 * C_2)$

4 Lab Results 结果

4.1 实验结果

HUNAN UNIVERSITY ACM/ICPC Judge Online

Realtime judge status

Solution	User	Problem	Language	Judge Result	Memory	Time Used	Code Length	Submit Time
705132	jsll201808010522	12596	GNU C++	Accepted	5604KB	375ms	2377B	2021-05-15 20:34:20.0
705131	jsll201808010522	12596	GNU C++	Wrong Answer	5280KB	125ms	2399B	2021-05-15 20:33:53.0
697550	jsll201808010522	13120	GNU C++	Accepted	3072KB	0ms	3865B	2021-03-20 22:04:18.0
697525	jsll201808010522	13120	GNU C++	Accepted	1068KB	0ms	2258B	2021-03-19 10:08:00.0
697436	jsll201808010522	12595	GNU C++	Accepted	5056KB	15ms	1035B	2021-03-05 14:10:58.0
697394	jsll201808010522	12595	GNU C++	Accepted	1056KB	0ms	1882B	2021-03-02 08:35:25.0

<<Previous Page NextPage>>

User Id: jsll2018080105 Problem Id: Language: All Result: ALL Course Id: 244 Rank?: No Query(Q)

Total 6 solution(s) return to contest/course

图 1: http://acm.hnu.cn/online 提交结果

在 http://acm.hnu.cn/online 提交代码，AC 通过. SolutionID 705132 ，User jsll201808010522，Memory 5604KB Time Used 375ms ， 实验正确。

5 Lab Experience 心得

5.1 实验心得

对于这道 CFG 是 P 成员题，有两种解法

1. 思路 1：拿到这道题首先想到的是用 STL 的结合 set 容器，可以快速判断一个字符串是否与其他的字符在同一个集合，遍历这个字符穿，判断是否存在于同一个集合，从而可以得出这个字符串是否能够被派生。Pass 掉了，因为题目要求用动态规划。

2. 思路 2：动态规划法，对于一个字符串 str，我们可以用 dp[j][k] 来表示这个字符串从第 j 个字母到第 k 个字母被派生，dp[j][k] 中存储一个字符串，如果 str[j:k] 可以被派生, 则这个字符串的所有字母存储在 dp[j][k] 中。对于一个字符串 str[j:k], 在 j …k 中取到一个分割点 r，如果 str[j:r] 和 str[r:k] 可以被派生，那么必须满足一个 S->BC 类型的乔姆斯基范式，其中 B 必须在 dp[j][r] 中找到，C 必须在 dp[r][k] 中找到。

为了训练 dp, 最终我选择了第二种解法，需要注意的是，问题中给了 m 个测试字符串，所以我们要增加一个维度来描述动态规划过程，因而 dp 数组变化为 dp[i][j][k], 其中 i 代表第 i 个测试字符串，j 代表待测试字符串的起始位置，k 代表其结束位置。

A 附录 1：Solution

```
//12596
#include <iostream>
#include <string>
using namespace std;
string dp[100][100][100];//表格
string cfg[100];//cfg规则
string str[100];//待测字符串
string sa[100];//sa型规则
string SAB[100];//SAB形规则
int main()
{
    //输入
    int n,m;
    scanf("%d",&n);
    for(int i=0;i<n;i++) cin>>cfg[i];
    scanf("%d",&m);
    for(int i=0;i<m;i++) cin>>str[i];
    //记录长度
    int lstr[m];
    for(int i=0;i<m;i++) lstr[i]=str[i].length();
    //找出所有A->b型规则
    int num=0;
    for(int i=0;i<n;i++)
    {
        for(int k=3;k<cfg[i].length();k++)
        {
            if(cfg[i][k]>='a'&&cfg[i][k]<='z')
            {
                sa[num]+=cfg[i][0];//S->AB|a,记录S
                sa[num]+=cfg[i][k];//记录S->a
                num++;
```

```
        }
    }
}
//找出所有A->BC型规则
int num2=0;
for(int i=0;i<n;i++)
{
    for(int k=3;k<cfg[i].length();k++)
    {
        if(cfg[i][k]<'a')
        {
            SAB[num2]+=cfg[i][0];//记录S
            SAB[num2]+=cfg[i][k];//记录AB
            SAB[num2]+=cfg[i][k+1];
            k++;
            num2++;
        }
    }
}
//考察每一个长为1的子串
for(int i=0;i<m;i++)//m个字符串中的i
{
    for(int j=0;j<lstr[i];j++)//第i个字符串的第j个字母
    {
        for(int k=0;k<num;k++)//A->a类型字符串的第k个字符串
        {
            if(str[i][j]==sa[k][1]) //考察CFG文法的每一个字符
                dp[i][j][j]=sa[k][0];//记录dp[i][j][j]=sa[k][0]起始位置
        }
    }
}
//考察l长度的子串
for(int i=0;i<m;i++)
{
    for(int l=2;l<=lstr[i];l++)  //l是子串的长度
    {
        for(int p=0;p<lstr[i]-l+1;p++) //p是子串的起始位置
```

```
            {
                int j=p+l-1;              //j是子串的结束位置
                for(int k=p;k<j;k++)  //k是分裂的位置 k<=j-1 ->k<j
                    for(int q=0;q<num2;q++)
                        if(dp[i][p][k].find(SAB[q][1])!=-1
                        &&dp[i][k+1][j].find(SAB[q][2])!=-1)
                            dp[i][p][j]+=SAB[q][0];
            }
        }
    }
    //检查起始变元是否在dp[0][n]中
    for(int i=0;i<m;i++)
    {
        int flag=0;
        if(dp[i][0][lstr[i]-1].find(cfg[0][0])!=-1) printf("yes\n");
        else printf("no\n");
    }
    return 0;
}
```