科大讯飞
iFLYTEK

智能化技术训练营
二期题目及相关技术培训

2014.05.16

写在前面的话

一期颁奖典礼——最佳风格奖

```
//+-------------------------------------------------------------------+
//+ 资源编译组件 实现文件
//+ 对导出的DLL进行功能解耦合
//+
//+ 作者：杜婷
//+ 创建时间：2014-5-13
//+ 修改说明：
//+      1,用简单的单价模式封装，没有考虑到线程安全
//+      2,用C++11 的智能指针进行封装，进行垃圾回收处理
//+-------------------------------------------------------------------+

#pragma once

#ifndef ___RESBUILD_IMPL_H___
#define ___RESBUILD_IMPL_H___

#include "../ComUtils/StrUtil.hpp"
using namespace StrUtil;

#include "stdafx.h"
#include <memory>
#include <string>
#include <sstream>
#include <iostream>
#include <fstream>
#include <map>
using namespace std;

namespace Util {
    namespace ResBuild {
//+-------------------------------------------------------------------+
//+ ResBuilder 类声明部分
//+-------------------------------------------------------------------+
#pragma region Declaration
class ResBuilder {
```

```
        private:
            Proc_Initialize pInitialize;
            Proc_ProcessBuildRes pProcessBuildRes;
            Proc_UnInitialize pUnInitialize;

            // dll 句柄
            HINSTANCE _hdll;

            ResBuildHelper();
            virtual ~ResBuildHelper();
            friend class std::auto_ptr<ResBuildHelper>;
            static std::auto_ptr<ResBuildHelper> _instance;
        public:
            static ResBuildHelper* Inst() {
                if (0 == _instance.get()){
                    _instance.reset(new ResBuildHelper);
                }
                return _instance.get();
            }
        }; // end of ResBuildHelper
#pragma endregion
    }
```

```
//+-------------------------------------------------------------------+
//+ 压缩歌手列表
//+-------------------------------------------------------------------+
//+ singerListPath        -[in] 歌单文件路径
//+ resPath               -[in] 需要保存的压缩后的文件路径
//+ return                - 状态码,0 表示成功,其他表示错误码
//+-------------------------------------------------------------------+
int ResBuilder::ProcessBuildResImpl(string singerListPath, string resPath)
```

科大讯飞

一期颁奖典礼——最佳翻译奖

```
printf("[%d].wav端点检测序列结果: \n",q);
for (long j = 0; j <= i - 1; j++){
  switch (status)  {        //status 0 为静音状态, 1 为可能
  case 0:
    if (e[j] > amp1) {     //进入语音状态
      if ((j - count) < 0)    //x1记录语音起点帧序列;
        x1 = 0;
      else x1 = j - count;
      status = 2;
      silence = 0;    //silence  无声长度
      count = count + 1;
    }
    else if (e[j]>amp2 || zcr[j] > zcr1) { //可能语音状态条
      status = 1;
      count = count + 1;
    }
    else {              //静音状态
      status = 0;
      count = 0;
    }
    break;
  case 1:
    if (e[j] > amp1) {     //进入语音状态
      if ((j - count) < 0)    //x1记录语音起点;
        x1 = 0;
      else x1 = j - count;
      status = 2;
      silence = 0;    //silence  无声长度
      count = count + 1;
    }
    else if (e[j]>amp2 || zcr[j] > zcr1) { //可能语音状态条
      status = 1;
      count = count + 1;
    }
    else {              //静音状态
      status = 0;
      count = 0;
```

```
%开始端点检测
x1 = 0;
x2 = 0;
for n=1:length(zcr)
   goto = 0;
   switch status
   case {0,1}                    % 0 = 静音, 1 = 可能开始
      if amp(n) > amp1           % 确信进入语音段
         x1 = max(n-count-1,1);
         status  = 2;
         silence = 0;
         count   = count + 1;
      elseif amp(n) > amp2 | ... % 可能处于语音段
             zcr(n) > zcr2
         status = 1;
         count  = count + 1;
      else                       % 静音状态
         status  = 0;
         count   = 0;
      end
   case 2,                       % 2 = 语音段
      if amp(n) > amp2 | ...     % 保持在语音段
         zcr(n) > zcr2
         count = count + 1;
      else                       % 语音将结束
         silence = silence+1;
         if silence < maxsilence % 静音还不够长, 尚未结束
            count  = count + 1;
         elseif count < minlen   % 语音长度太短, 认为是噪声
            status  = 0;
            silence = 0;
            count   = 0;
         else                    % 语音结束
            status  = 3;
```

科大讯飞

一期颁奖典礼——最老实奖

、程序思路说明

招供如下：由于我并不熟悉数据压缩算法，所以就考虑现有的压缩软件。Zip 和 7z 是开源的，测试了一下，7z 压缩比很高，但是比 zip 慢一些，所以我用了 zip。用的是 zlib 的库。

10 万条歌单数据，考虑一次载入多次查询，没有运行时增删改查，所以直接开静态数组了，qsort 排序做初始化。然后根据 txt 的 ANSI 编码，第一个字节做索引，索引开数组 256 项。这样平均每项索引下的数据条目是 10 万/256 = 400，平均 400 条，然后按第二个字节折半查找（目的是找到第一第二个字节匹配的条目），最后从第三个字节开始逐条匹配。

解释一下结构体中的冗余项，保留了一个 int 类型，记录串长度。本来想实现一下稍微模糊一点的匹配，比如“丁丁”和“丁丁（中国好声音）”，在匹配的时候如果一直扫描到带匹配串结尾都匹配，也算成功匹配。但是形如这组“丁当”和“小丁当”的相似匹配，暂时没有想到更好的方法。所以就没附加功能了。

最后，无论是哪位大神看到我这篇粗浅的说明，都深表感谢：

zlib.h	2013/4/29 8:23	C/C++ Header	86 KB
zlibwapi.dll	2014/5/14 21:23	应用程序扩展	166 KB
zlibwapi.lib	2014/5/14 21:23	Object File Library	28 KB
程序说明文档.doc	2014/5/14 23:10	Microsoft Office...	16 KB

科大讯飞

一期颁奖典礼——最佳钻研奖

基本思路：首先将文件读到内存中，保存到 char *sour，将*sour 进行筛选（以'\t', '\r\n' 为筛选条打包成两个部分，第一个部分，文字部分 数字部分。

第一部分：将所有名字部分的*name 缩，然后再压缩文件里，先写入 huffma {char，code（01 字符串）}，在写入*name 的信息，紧缩排列。

因为要综合压缩率以及解压速度，本次选择了 LZSS 算法；

本算法对于 ASCII 编码的文本压缩比较高，但是对于本次给出的 ANSI 编码的文本压缩比较低，压缩率大概为%70~~%80（文本越大，压缩比越高，压缩率越低）。但是本算法的解压速度较快，可以稍微弥补压缩比低的缺点。

Build 部分使用的是 Huffman 编码方式进行压缩的，同时出于对解析压缩文件的速度和进一步的查找速度的考虑，在正式存储编码并写入文件之前，程序先对数据排了个序。

Select 部分就是简单的解压缩和二分查找。由于在压缩之前有对文件进行过处理，所以这里只要单纯的解压缩就可以了。另外我实验过使用 unordered_map 自带的 hash 特性进行查找，发现速度与自己写的二分差不太多，出于性能稳定的考虑最后使用了二分查找。

至于第一部分与第二部分在反解析时，可以在文件最头，先存入*name 部分压缩后的 size。

科大讯飞

一期颁奖典礼——最佳钻研奖

```
/*
*
* LZWAlgorithm.h   LZW算法头文件
* Copyright (C) 王汉超(hanchaow@mail.ustc.edu.cn) 2014-05-13
* 参考资料:
*   http://blog.chinaunix.n
*
*/
#ifndef _LZW_ALGORITHM_H_
#define _LZW_ALGORITHM_H_

#include <stdio.h>
#include <stdlib.h>

#define VALUE
#define FUNCTION
#define INIT_BITS 9
#define MAX_BITS  14
#define HASHING_SHIFT MAX_B

#define TABLE_SIZE 18041
#define CLEAR_TABLE 256
#define TERMINATOR  257
#define FIRST_CODE  258
#define CHECK_TIME  100

#define MAXVAL(n) (( 1 <<(
```

```
/*
 * 压缩程序的核心代码
 * 参数1 input 要解压的文件
 * 参数2 output 解压后的文件
 */
void LZWAlgorithm::compress(FILE *input, FILE *output)
{
   unsigned int next_code=FIRST_CODE;   //下一个字符
   unsigned int character;   //当前编码
   unsigned int string_code; //原始字符
   unsigned int index;       //索引
   int i,
   ratio_new,          //新的压缩比率
   ratio_old=100;      //上一次的压缩比率 刚开始100%

   for (i=0;i<TABLE_SIZE;i++)   //初始化编码表
      code_value[i]=-1;
   string_code=getc(input);     //读取文件，获取第一个字符

   while((character=getc(input)) != (unsigned)EOF) {   //读取文件，依次获取每个字符
      ++bytes_in;
      index=find_match(string_code,character);   //查找编码表中是否还有该字符，参数是 （前缀码pre，当前字符c）
      if (code_value[index] != -1)   //找到，继续读取下个字符
         string_code=code_value[index];
      else {   //没找到，写入压缩文件并判断编码表是否已满
         if (next_code <= max_code ) {
            code_value[index]=next_code++;
            prefix_code[index]=string_code;
            append_character[index]=character;
         }
         output_code(output,string_code);   // 写入压缩字符
         string_code=character;
         if (next_code > max_code) {        // 如果编码表已经满了
```

科大讯飞

一期思路汇总——端点检测

方法	采用思路
时域与频域混合解析	将人声分为清音与浊音，清音采用能量阈值，浊音采用频域阈值检查。
四门限（能量，过零率）	利用四门限的阈值划分
能量检查	单纯使用能量大小来区分，或辅以低通滤波

一期思路汇总——打包与检索

打包方法	采用思路
LZW算法	提取原始文本文件数据中的不同字符，基于这些字符创建一个编译表，然后用编译表中的字符的索引来替代原始文本文件数据中的相应字符，减少原始数据大小。
Zlib开源库	计算信息熵，构建模型，编码
哈夫曼树	依据字符出现概率来构造异字头的平均长度最短的码字
LZ4算法	哈希表

检索方法
字符串匹配
二分查找
构造二叉树
哈希表

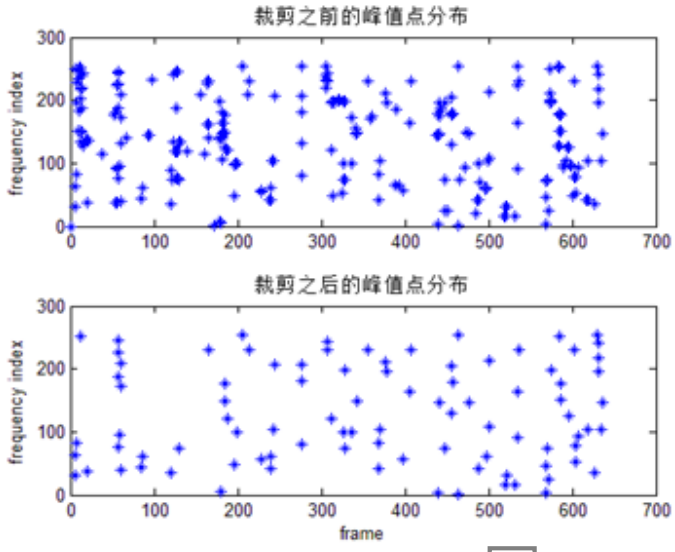
二期题目

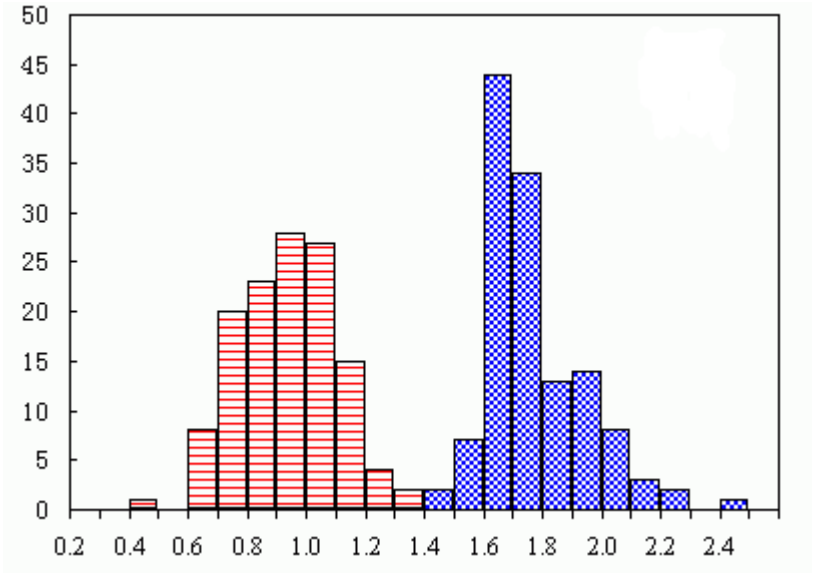
- 原声检索
- 语义理解

二期题目——原声检索


AT&T 2:26 PM 39%
Listening
AT&T 2:27 PM 38%
Done
Tag
THE 40 MPG* FIAT*500
LEARN MORE
返回
摇一摇
摇一摇搜歌已开启
中国联通 12:42 38%
正在识别...

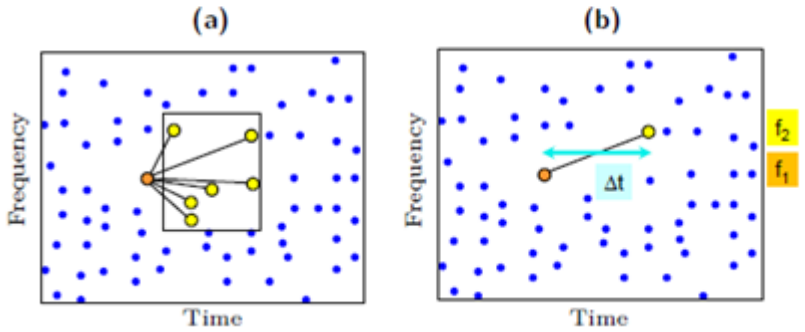
原声检索——解题思路





$(f_1, f_2, \Delta t)$: (time_offset, song_id)

- 所有时间偏移上 hash 键命中的最大个数作为音频相似度得分



Hash:

Consists of two frequency values and a time difference:

$(f_1, f_2, \Delta t)$

原声检索——题目

1. 构建音频库。
2. 输入音频片段，进行检索

- 例：

输入：音频的路径

输出：{[海阔天空]}

头文件-打包

```
//+-------------------------------------------------------------------------+
//+ 初始化
//+ 对需要的数据结构进行初始化操作
//+-------------------------------------------------------------------------+
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int Initialize();
typedef int (*Proc_Initialize)();

//+-------------------------------------------------------------------------+
//+ 数据打包
//+ 对给出的歌单进行打包，输出资源文件
//+-------------------------------------------------------------------------+
//+ szResDir       - 歌曲文件夹输入路径
//+ szResPath      - 资源输出路径
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int ProcessBuildRes(const char *szResDir, const char *szResPath);
typedef int (*Proc_ProcessBuildRes)(const char *szResDir, const char *szResPath);

//+-------------------------------------------------------------------------+
//+ 逆初始化
//+ 对需要的数据结构进行析构
//+-------------------------------------------------------------------------+
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int unInitialize();
typedef int (*Proc_unInitialize)();
```

头文件-检索

```
//+-------------------------------------------------------------------------+
//+ 初始化
//+ 对需要的数据结构进行初始化操作
//+-------------------------------------------------------------------------+
//+ szResPath      - 资源输入路径
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int Initialize(const char *szResPath);
typedef int (*Proc_Initialize)(const char *szResPath);

//+-------------------------------------------------------------------------+
//+ 资源检索
//+ 加载编译好的资源包，对输入的音频进行检索，并输出对应的歌曲名
//+-------------------------------------------------------------------------+
//+ szAudioPath    - 待检索音频路径
//+ szResultString     - 输出结果，格式：{[忘情水]}
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int ProcessSelelct(const char *szAudioPath, std::string &szResultString);
typedef int (*Proc_ProcessSelelct)(const char *szAudioPath, std::string &szResultString);

//+-------------------------------------------------------------------------+
//+ 逆初始化
//+ 对需要的数据结构进行析构
//+-------------------------------------------------------------------------+
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int unInitialize();
typedef int (*Proc_unInitialize)();
```

原声检索——要求

- 使用c/c++
- 使用单线程
- 衡量指标
  - 音频检索速度
  - 音频库的大小
  - 音频检测的准确性

二期题目——语义理解

- 我想听刘德华的忘情水
- 我就是想听刘德华的忘情水
- 有没有刘德华的忘情水
- 刘德华唱的忘情水
- 刘德华的歌曲忘情水

- **Singer:**刘德华
- **Song :**忘情水

语义理解——硬匹配

◆维护一个拥有大量例句的数据集，如下表

◆识别就是查找与用户短信相匹配的例句，返回例句对应的结果

◆项目中，硬匹配应要求改为严格匹配，即查找与短信完全相同的例句

◆对数字串提供通配识别，要求例句中含有通配数字串<num>的例句，识别结果中添加返回num字段

1	开通我的100m本地流量包	biz:数据流量套餐	opera:开通
2	定制100m本地流量包	biz:数据流量套餐	opera:开通
3	我要订购100m本地流量包	biz:数据流量套餐	opera:开通
4	订制100m本地流量包	biz:数据流量套餐	opera:开通
5	开100m本地流量包	biz:数据流量套餐	opera:开通
6	订购100m本地流量包	biz:数据流量套餐	opera:开通
7	定购100m本地流量包	biz:数据流量套餐	opera:开通
8	办理100m本地流量包	biz:数据流量套餐	opera:开通
9	办100m本地流量包	biz:数据流量套餐	opera:开通

文法网络的产生

观察发现：数据集中的问题存在大量相同的句式，有相同或相似的前缀、后缀以及识别结果。对于识别结果相同的句式，可合并其相同的前后缀

查询刘德华的歌	**biz:**歌曲推荐	**opera:**帮助
查询刘德华的歌曲	**biz:**歌曲推荐	**opera:**帮助
查询刘若英的歌	**biz:**歌曲推荐	**opera:**帮助
查询张学友的歌	**biz:**歌曲推荐	**opera:**帮助
查询张学友的歌曲	**biz:**歌曲推荐	**opera:**帮助

数据集

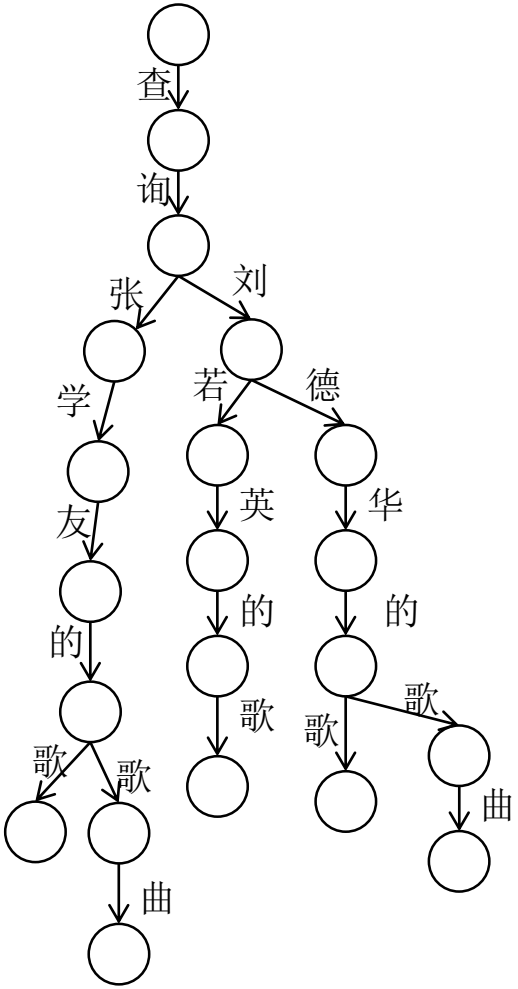
$query = 查询 ;
$song = 歌 | 歌曲;
$singer =刘德华 | 刘若英 | 张学友 ;
$r1 = **$query $singer** 的 **$song** ;
$main = **$r1** ;

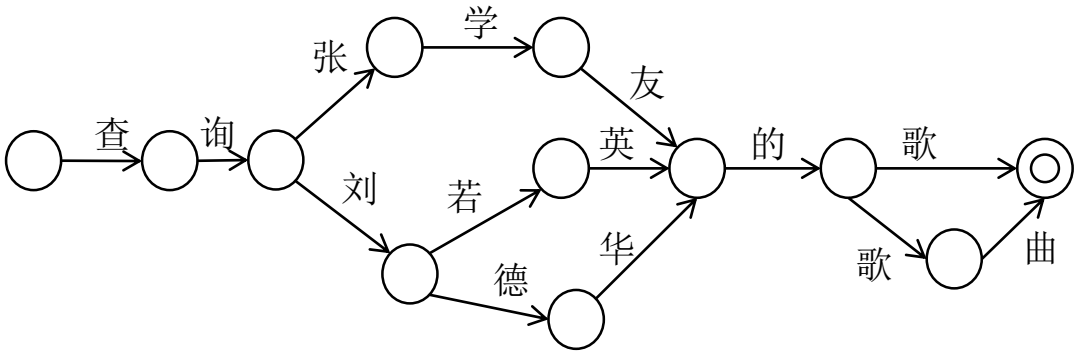
文法

科大讯飞

文法网络的产生

前缀搜索树 → 文法网络





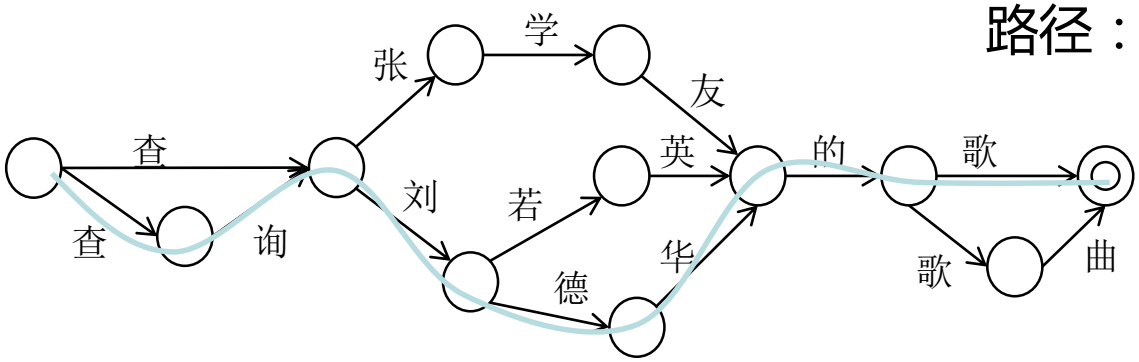
搜索树转化为文法网络
减少节点数量，降低内存占用，提高识别效率

简单文法网络实例

为了便于说明，前文文法稍作修改如下

```
$query = 查 | 查询;
$song = 歌 | 歌曲;
$singer =刘德华 | 刘若英 | 张学友 ;
$r1 = $query $singer 的 $song ;
$main = $r1;
```

编译文法生成文法网络，用于识别

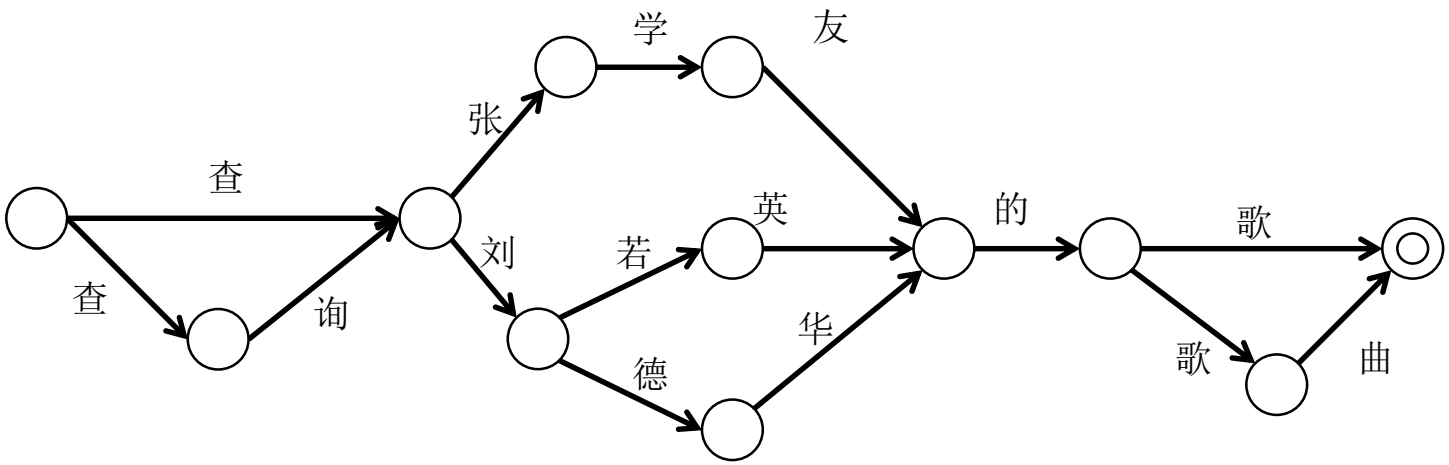


边：识别字符
点：识别状态
路径：由边点组成的有序序列

文法网络识别：在网络中找出一条有效路径，其起点为网络首节点，终点为网络终止节点，路径上的边的有效字符的有序组合与输入短信相同

简单文法网络识别过程及原理介绍

语句“查询刘德华的歌”的识别过程



识别成功

文法网络的意义

◆占用内存较少，识别效率较高

文法网络合并了相同的前后缀，极大减少了重复节点，降低了内存的占用，提高识别效率

◆文法的扩充和修改更加灵活方便

在原有文法的基础上，添加部分变量和句式，就可以涵盖大量的识别内容。一些实时信息的更新和添加，只要修改文法中某些变量即可。比如添加歌手及新歌，应用层不必关心这些修改。

科大讯飞

语义理解——题目

请设计程序，判断用户所需求的歌手及歌曲。

- 例：

输入：我想听刘德华的忘情水

输出：{[刘德华，忘情水]}

头文件

```
//+-------------------------------------------------------------------------+
//+ 初始化
//+ 对需要的数据结构进行初始化操作
//+-------------------------------------------------------------------------+
//+ szResPath      - 资源输入路径
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int Initialize(const char *szResPath);
typedef int (*Proc_Initialize)(const char *szResPath);

//+-------------------------------------------------------------------------+
//+ 语义理解
//+ 对输入的语义进行理解，输出结果
//+-------------------------------------------------------------------------+
//+ szMessage      - 输入的问句（我想听刘德华的忘情水）
//+ szResultString     - 输出结果，格式：{[刘德华，忘情水]}
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int ProcessUnderstand(const char *szMessage, std::string &szResultString);
typedef int (*Proc_ProcessUnderstand)(const char *szMessage, std::string &szResultString);


//+-------------------------------------------------------------------------+
//+ 逆初始化
//+ 对需要的数据结构进行析构
//+-------------------------------------------------------------------------+
//+ return         - 返回码
//+-------------------------------------------------------------------------+
extern "C" __EXPORT int unInitialize();
typedef int (*Proc_unInitialize)();
```

原声检索——要求

- 使用c/c++
- 使用单线程
- 衡量指标
  - 语义理解速度
  - 支持说法的多样性

二人合作——自由分组

衡量标准

- 效果分 = 数据集效果得分
- 效果满分80分

- 依据代码的编程风格，稳定性等一些指标，给出附加分。
- 附加满分20分

- 继续加油！