**阅读下面这段代码，说出它的目的，每个函数的作用，最终的输出结果以及设计思路：**

public class PinyinNet {

private ArrayList<String> mPyMap = ["a", "ai", "an", "ang", "ao",

...

"gu", "gua", "guai", "guan", "guang", "gui", "gun", "guo", ...

"o", "ou", ...

"xi", "xia", "xian", ...

];// 共412个合法的音节串，按字母序排列

public ArrayList<Integer> findValidSyllable(String str){

ArrayList<Integer> result = new ArrayList<Integer>();

if(str.length() == 0)

return result;

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

String syllable = mPyMap.get(i);

if(syllable.charAt(0) > str.charAt(0))

return result;

if(str.startsWith(syllable))

result.add(syllable.length() - 1);

}

return result;

}

public ArrayList<ArrayList<Integer> > preProcess(String inputStr){

ArrayList<ArrayList<Integer> > seg = new ArrayList<ArrayList<Integer> >();

for(int i=0; i<inputStr.length(); i++){

seg.add(this.findValidSyllable(inputStr.substring(i)));

}

return seg;

}

private void printPyNet(String inputStr, ArrayList<Integer> pyNet, ArrayList<ArrayList<Integer> > seg){

String str = new String();

for(Integer item : pyNet){

int nStart = (item.intValue() >> 16) & 0x0000ffff;

int nIdx = item.shortValue();

int nEnd = nStart + seg.get(nStart).get(nIdx) + 1;

str = str.concat(inputStr.substring(nStart, nEnd));

str = str.concat("'");

}

System.out.println(str);

}

public void MainProc(String inputStr){

ArrayList<ArrayList<Integer> > seg = this.preProcess(inputStr);

int nStart = 0;

int nIdx = 0;

ArrayList<Integer> pyNet = new ArrayList<Integer>();

while(true){

if(nStart<seg.size() && seg.get(nStart).size() !=0 && nIdx<seg.get(nStart).size()){

pyNet.add(((nStart << 16)&0xffff0000) |(nIdx & 0x0000ffff));

int nStep = seg.get(nStart).get(nIdx) + 1;

nStart += nStep;

nIdx = 0;

if(nStart == inputStr.length()){

this.printPyNet(inputStr, pyNet, seg);

}

}else{

if(nStart == 0)

return;

Integer item = pyNet.remove(pyNet.size() - 1);

nStart = (item.intValue() >> 16) & 0x0000ffff;

nIdx = item.shortValue() + 1;

}

}

}

public static void main(String [] args){

PinyinNet pinyinNet = new PinyinNet();

pinyinNet.MainProc("xianguo");

}

}

**阅读下面这段代码，说出它的目的，每个函数的作用，最终的输出结果以及设计思路：**

#include <stdio.h>

#include <string.h>

class PyNetMaker

{

public:

void MainProc(const char \*inputStr);

protected:

short\*\* preProcess(const char\* inputStr);

void printPyNet(const char\* inputStr, int\* route, int top, short\*\* arcArray);

const char\* mPyMap[] = {"a", "ai", "an", "ang", ...

"gu", "gua", "guai", ..., "guo", ...

"o", "ou", ...

"xi", "xia", "xiang", ...

};// 共412个合法的音节串，按字母序排列

};

short\*\* PyNetMaker::preProcess(const char\* inputStr)

{

if(inputStr == NULL || strlen(inputStr) == 0)

return NULL;

short\*\* arcArray = (short\*\*)malloc(sizeof(short\*) \* strlen(inputStr));

for(int i=0; i<strlen(inputStr); i++){

const char\* tmpStr = inputStr + i;

static const int nMaxArc = 16;

short arcLen[nMaxArc] = {0};

int arcCount = 0;

for(int j=0; j<(sizeof(mPyMap) / sizeof(char\*)); j++){

char\* syllable = mPyMap[j];

if(syllable[0] > tmpStr[0])

break;

if(strstr(tmpStr, syllable) == tmpStr){

arcLen[arcCount++] = strlen(syllable) - 1;

}

}

arcArray[i] = (short\*)malloc(sizeof(short) \* arcCount + 1);

arcArray[i][0] = arcCount;

memcpy(arcArray[i] + 1, arcLen, sizeof(short) \* arcCount);

}

return arcArray;

}

void PyNetMaker::printPyNet(const char\* inputStr, int\* route, int top, short\*\* arcArray)

{

char result[256] = {0};

for(int i=0; i<top; i++){

int value = route[i];

short nStart = (value >> 16) & 0x0000ffff;

short nIdx = value & 0x0000ffff;

strncat(result, inputStr + nStart, arcArray[nStart][nIdx + 1] + 1);

strcat(result, "'");

}

printf("%s\n", result);

}

void PyNetMaker::MainProc(const char\* inputStr)

{

if(inputStr == NULL || strlen(inputStr) == 0)

return;

short\*\* arcArray = preProcess(inputStr);

int nStart = 0;

int nIdx = 0;

int\* route = (int\*)malloc(sizeof(int) \* strlen(inputStr));

int top = 0;

while(1){

if(nStart<strlen(inputStr) && arcArray[nStart][0]!=0 && nIdx<arcArray[nStart][0]){

route[top++] = (((nStart << 16) & 0xffff0000) | (nIdx & 0x0000ffff));

nStart += arcArray[nStart][nIdx + 1] + 1;

nIdx = 0;

if(nStart == strlen(inputStr))

printPyNet(inputStr, route, top, arcArray);

}else{

if(nStart == 0)

return;

int value = route[--top];

nStart = (value >> 16) & 0x0000ffff;

nIdx = (value & 0x0000ffff) + 1;

}

}

free(route);

for(int i=0; i<strlen(inputStr); i++)

free(arcArray[i]);

free(arcArray);

}

int main(int argc, char \*argv[])

{

PyNetMaker pyNetMaker();

pyNetMaker.MainProc("xianguo");

return 0;

}

**你处理过的最大的难题，让你印象深刻的，选一个深入地讲解。（难点在哪？分析过程，怎么解决？）**

**你所做过的最好的设计，让你印象深刻的，选一个深入地讲解。（解决什么问题？关键点在哪？你的设计方案优点在哪？有什么不足？）**

**谷歌为什么引入Direct Boot模式？**

磁盘加密的目的是保护用户数据。Android 6.x及以下版本采用加密方式是fde，全部磁盘数据都加密，解锁以后才能访问。带来的问题是：手机启动未解锁之前，App不能访问磁盘数据，如果长时间不解锁，某些功能就不能用。所以，谷歌在7.0上引入快速启动模式，但是不得不改变加密方式(由fde升级成fbe)。

**手机进入Direct Boot模式，必须使用adb命令或者恢复出厂设置吗？**

答案是：不一定。

第一种手机：出厂时预装Android 7.x的，自动使用快速启动模式，无需任何操作。

第二种手机：从旧版本升级到Android 7.x的，必须手动操作才能进入快速启动模式。

第三种手机：版本＜＝Android 6.x的手机，不支持快速启动模式。

**对于第二种手机，为什么必须手动操作才能进入Direct Boot模式？根本原因是什么？**

答案是Android的磁盘加密方式。

Android 2.3.x~6.x，加密方式是fde(full-disk encryption)，也就是全磁盘加密。2.3.x~4.x的系统，用户手动操作才能加密，5.x~6.x的系统默认是加密的。要取消加密，必须恢复出厂设置。手机启动后必须解锁，APP才能访问磁盘数据。

Android 7.x，加密方式是fbe(file-based encryption)，支持对单个文件进行加密。磁盘分成两个区：凭据存储区，解锁后才能访问；设备存储区，解锁前和解锁后都能访问。

从旧版本升级到7.x的手机，加密方式不能自动升级，必须手动操作。所以，这种手机默认不支持快速启动模式。有三种方式使这种手机进入快速启动模式：

1，设置中的“convert to file encryption”；

2，adb-wipe命令；

3，adb-emulation。

前两种方式都会恢复出厂设置，第三种方式只是模拟快速启动模式。

**关于如何知道App处于Direct Boot状态，官方提供了两种方式。**  
一、调用接口UserManager.isUserUnlocked() ，我验证了解锁前和解锁后，这个接口是OK的。还没验证再次锁屏的情况。  
二、监听系统广播，进入快速启动状态的时候，App会收到LOCKED\_BOOT\_COMPLETED，离开快速启动状态时，App会收到ACTION\_USER\_UNLOCKED和ACTION\_BOOT\_COMPLETED. 这种方法需要再验证一下。结论如下：

1、 接口UserManagerCompat.isUserUnlocked()可以用来判断是否处于快速启动状态，不表示解锁状态。

2、 无法收到LOCKED\_BOOT\_COMPLETED.

3、 可以收到ACTION\_USER\_UNLOCKED，表示离开快速启动状态，不表示解锁。开机首次解锁时可以收到这个消息，再次解锁不会收到这个消息。监听这个广播，不能在manifest注册，必须动态注册，建议在App启动时注册。

4、 可以收到ACTION\_BOOT\_COMPLETED，这个消息比ACTION\_USER\_UNLOCKED晚，表示系统已经启动。

在小米手机上能收到 LOCKED\_BOOT\_COMPLETED, 华为P10收不到

**如何定位分析Android Native Crash？**

**怎么查看Native崩溃日志？里面包含了哪些信息？你的查看顺序是什么？需要用到什么工具，分别什么作用？**

崩溃日志中记录的是模块和地址，首先应当把偏移地址转换为行数，可以使用ndk-stack或者addr2line，需要传入带符号的so，和偏移地址，它输出对应的行数；还可以使用objdump将so中的机器指令转成汇编，并找到发生崩溃的偏移地址的汇编语句，当没有带符号的so时，这种方式比较有效。