《软件开发综合实验》

Comprehensive Experiment on Software Development

詹文翰

zhanwenhan@163. com

课程背景

1. 本课程属于计算机专业的**实践类必修**课程，面向高年级学生开设，旨在培养学生对所学知识的**综合**运用能力。
2. 课程将围绕一个**模拟实战**的软件项目，让学生以**团队**为单位，完成从需求分析、系统设计、程序编码到集成测试的**整个软件生命周期**。
3. 通过对软件工程的全流程实践，提升学生的系统建模与分析能力、程序设计与实现能力、团队协作及领导能力，使学生具备足够的面向市场的**工程能力**和**职业素养**。

前置课程

必备：

《程序设计》
《数据结构与算法》
《软件工程》
《软件开发环境》

推荐：

《基于操作系统编程》
《UML统一建模语言》
《软件配置管理》
《操作系统》
《计算机网络》

实验内容及要求

1. 设计并实现一款**数据备份软件**，以项目组形式推进，每组最多**三人**。

2. 基于软件工程方法学进行项目推进，经历从需求分析、系统设计、编码实现、软件测试的整个**软件生命周期**。

3. 实验最终成果包括一款基本可用的**软件**及其对应**文档**。

4. 软件应包括指明的完整功能，重点考察其**正确性、易用性、健壮性**。

5. 软件文档应包括：**需求分析说明书、系统设计文档、软件测试报告**，重点考察其**规范性、一致性、可读性**。

6. 采用现代化**软件开发工具**辅助项目开发，包括但不限于：项目管理工具，UML建模工具，集成开发环境，版本控制工具，软件测试工具。

实验难度分级和评分标准

基本要求

各小组“独立”实现一款数据备份软件（对应**基础分总分40分**）：

数据备份：将目录树中的文件保存到指定位置

数据还原：将目录树中的文件恢复到指定位置

扩展要求

各项目组根据自身情况自行选择扩展要求（对应**扩展分总分**）。

文件类型支持（10分）：支持特定文件系统的特殊文件（管道/软链接/硬链接等）

元数据支持（10分）：支持特定文件系统的文件元数据（属主/时间/权限等）

自定义备份（10分）：允许用户筛选需要备份的文件（路径/类型/名字/时间/定时）

压缩解压（10分）：通过文件压缩节省备份文件的存储空间

打包解包（10分）：将所有备份文件拼接为一个大文件保存

加密备份（10分）：由用户指定密码，将所有备份文件均加密保存

实时备份（10分）：自动感知用户文件变化，进行自动备份

图形界面（10分）：实现友好易用的GUI界面

网络备份（30分）：将数据备份软件从单机模式扩展为网盘模式（10分），还涉及到的功能包括：用户管理（5分）、元数据管理（5分）、传输加密（5分）、增量备份（5分）等。

其它功能：视功能难度讨论加分。

实验难度分级和评分标准

开发环境

操作系统选择：Linux/Windows/MacOS

开发语言选择：C/C++/C#/Java/Go；用户界面可以采用脚本语言编写；后台逻辑若也选择脚本语言，则小组基础分记10分。

库的使用：对所有扩展功能，如使用第三方库/程序/代码“直接”实现，对应功能扩展分总分记为原来的50%。

实验评分标准

项目难度分 = 项目基础分+项目扩展分　　*（不超过120分）*

项目完成分 = 需求分析说明书（10分）+
系统设计文档（20分）+
软件测试报告（20分）+
变更管理（5分）+
源码质量（15分）+
项目答辩（10分）+
项目演示（20分）

小组得分 = 项目难度分 - 100 + 项目完成分

组长得分 = 小组得分　　*（不超过100分）*
组员得分 = 小组得分-5　　*（不超过100分）*

提交资料

以小组为单位：

1. 项目报告文档
2. 项目答辩PPT
3. 源代码+可执行程序（包含程序构建脚本，推荐dockerfile）
4. 项目演示视频（2分钟以内）

提交方式

1-2：提交到实验平台（单独的pdf文档，打包提交）；
3-4：打包提交到老师邮箱。

截止日期

最后一次实验课（第八次课）。

课程安排

1. 项目概述与技术基础
2. 项目管理与需求分析
3. 系统设计
4. 软件配置与项目实现
5. 需求变更与敏捷开发
6. 软件测试
7. 知识拓展与项目完善
8. 项目答辩及演示

自由分组时间

填写在线文档确认分组

班级通知讨论群

第一节 技术基础

詹文翰

zhanwenhan@163.com

技术基础

1

项目概述与技术基础

文件类型支持

元数据支持

自定义备份

压缩解压

打包解包

加密解密

实时备份

界面友好

网络备份

技术基础


文件类型支持
元数据支持
自定义备份
压缩解压
打包解包
加密解密
实时备份
界面友好
网络备份
Unix文件系统
编码解码
打包解包
对称加密和散列算法
Unix文件系统事件感知
Unix网络编程
并发编程
GUI编程
数据库编程

技术基础

Unix文件系统

编码解码

打包解包

对称加密和散列算法

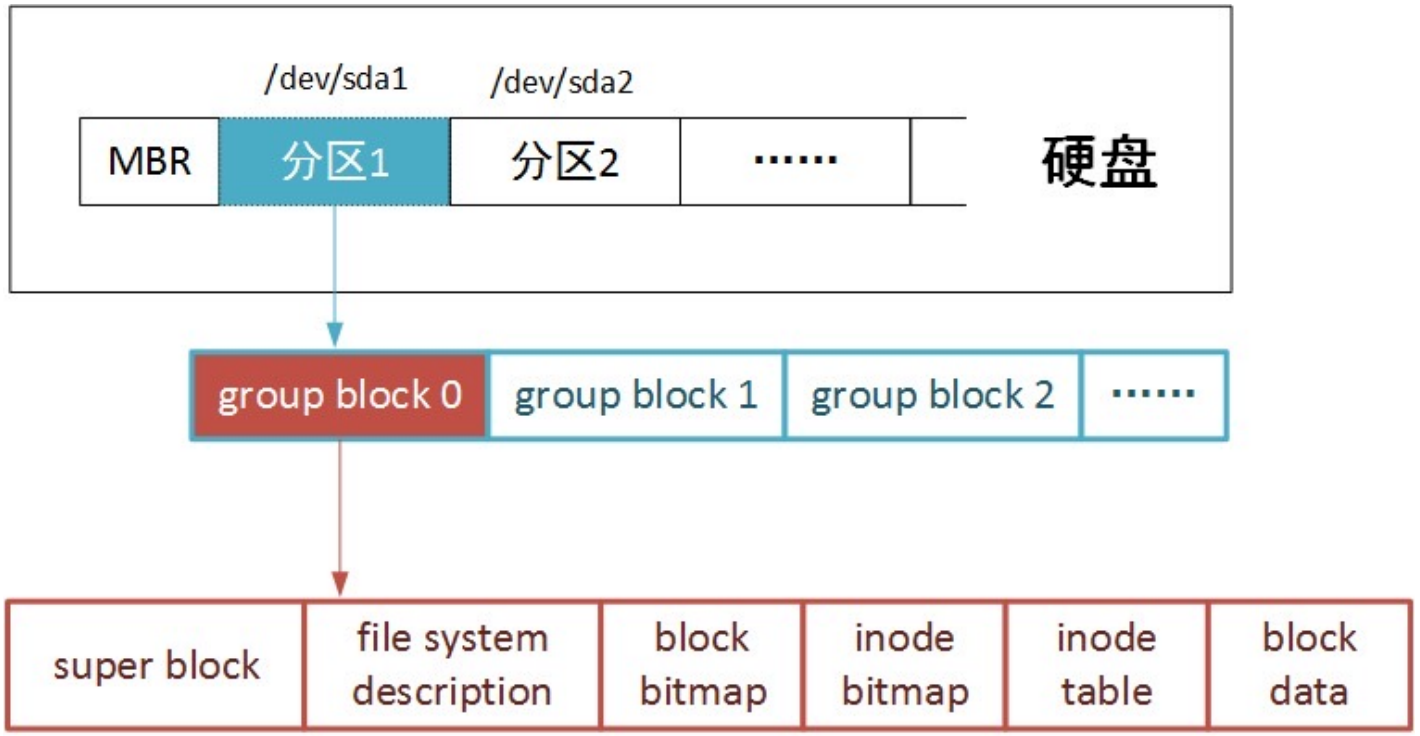
Unix文件系统事件感知

Unix网络编程

并发编程

Unix文件系统

经典的ext2文件系统结构图

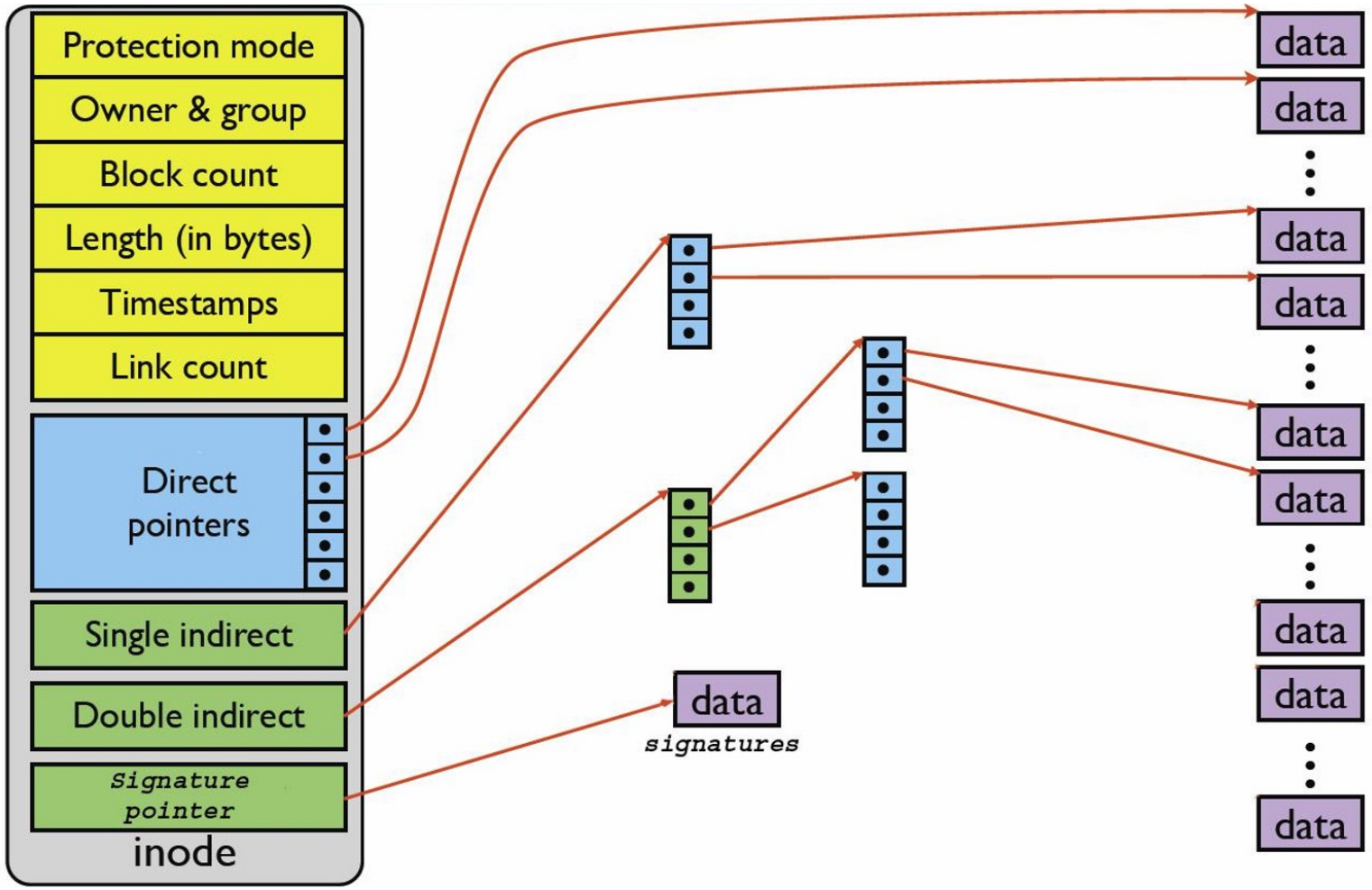


Unix文件系统

ext2文件系统中的普通文件

1

项目概述与技术基础

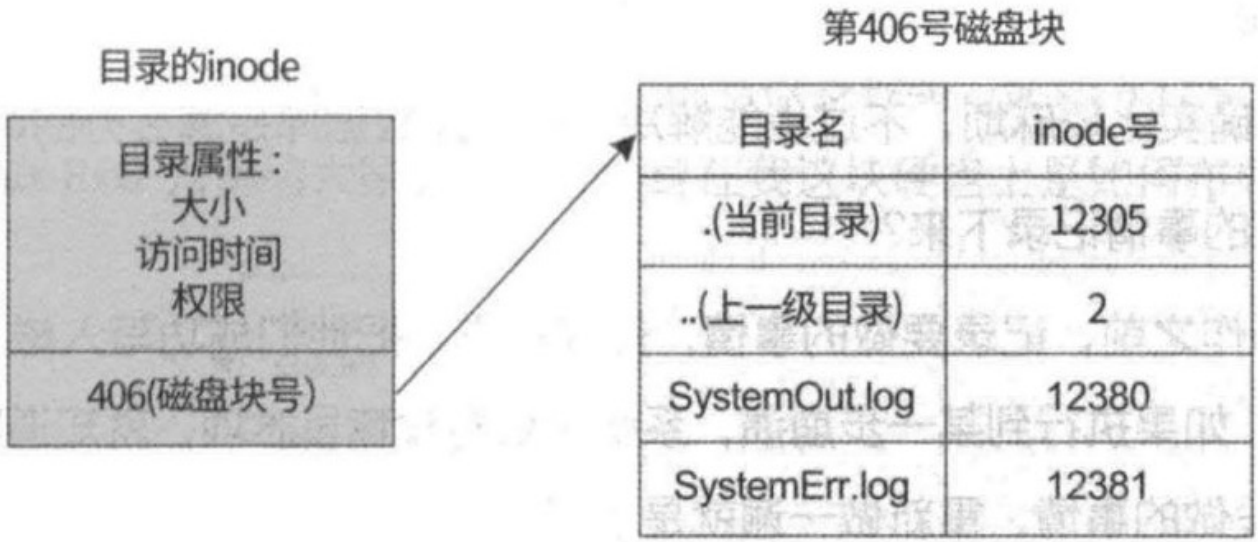


Unix文件系统

ext2文件系统中的目录

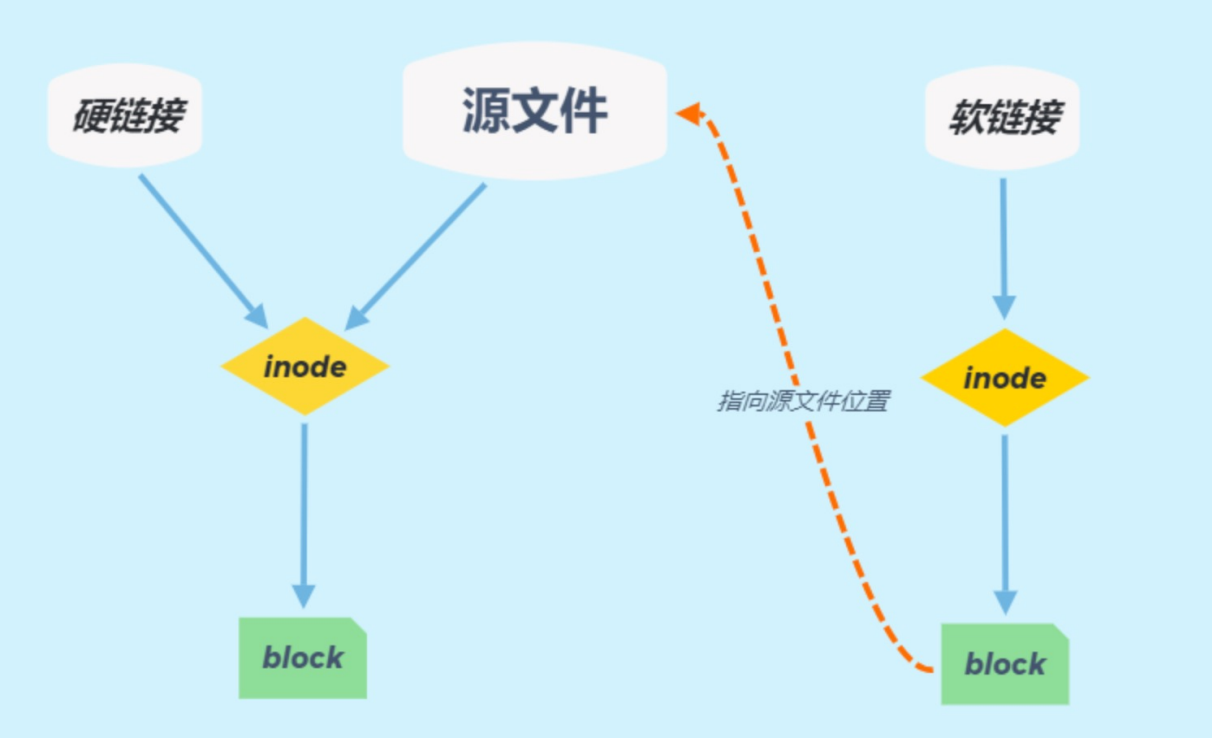
1

项目概述与技术基础



Unix文件系统

文件链接



Unix文件系统

Unix系统中的文件类型：

- 普通文件
- 目录文件
- 块设备文件
- 字符设备文件
- 套接字文件
- 管道文件
- 链接文件

Unix文件系统

Unix系统中的文件类型：

- 普通文件
- 目录文件
- 块设备文件
- 字符设备文件
- 套接字文件
- 管道文件
- 链接文件

Unix文件系统

1

项目概述与技术基础

如何保证备份软件的正确性：

1、文件类型
2、对链接文件的处理（软/硬）
3、文件属性（属主/权限/时间）

Unix文件系统

项目涉及到的系统API（部分）：

open
close
read
write
stat
chown
chmod
utimes
opendir
closedir
readdir
rewinddir
mkdir
mkfifo
unlink
link
symlink
……

Unix文件系统

项目涉及到的系统API（部分）：

open
close
read
write
stat
chown
chmod
utimes
opendir
closedir
readdir
rewinddir
mkdir
mkfifo
unlink
link
symlink
……

man手册

技术基础

Unix文件系统

编码解码

打包解包

对称加密和散列算法

Unix文件系统事件感知

Unix网络编程

并发编程

编码解码

在电文传输中，需要将电文中出现的每个字符进行二进制**编码**。在设计编码时需要遵守两个原则：

（1）发送方传输的二进制编码，到接收方解码后必须具有**唯一性**，即解码结果与发送方发送的电文完全一样；

（2）发送的二进制编码尽可能地**短**。

等长编码

每个字符的**编码长度相同**（编码长度就是每个编码所含的二进制位数）。这种编码的特点是解码简单且具有唯一性，但编码长度并不一定是最短的。

不等长编码

每个字符的**编码长度不同**。

压缩（无损）：若对出现频度较高的字符分配相对较短的编码，同时，对出现频度较低的字符分配相对较长的编码，则编码之后的数据的二进制位数可以变小。

压缩编码分类

基于统计的方法：Huffman编码、Shannon-Fano编码

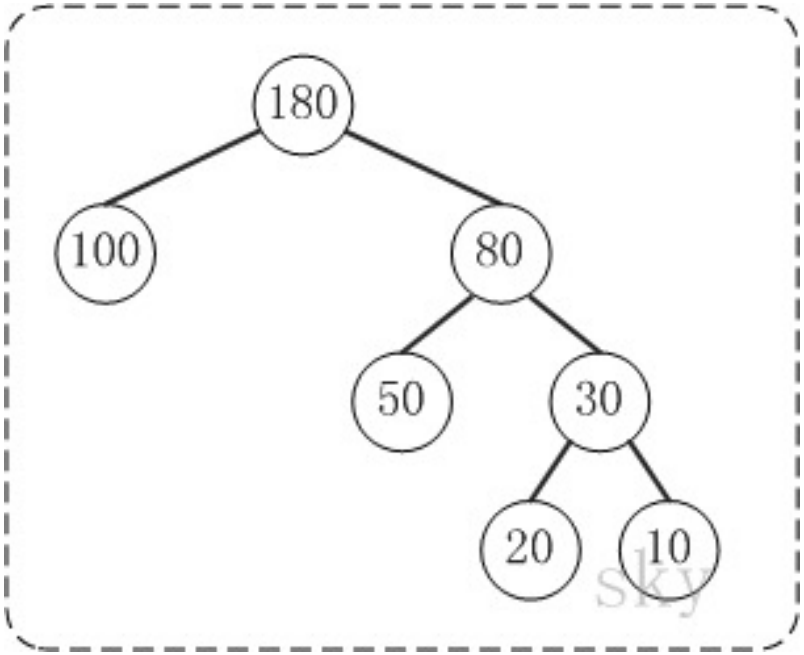
基于字典的方法：LZ77算法、 LZ78算法

混合方法：DEFLATE算法

以Huffman编码和LZ77算法为例。

哈夫曼树

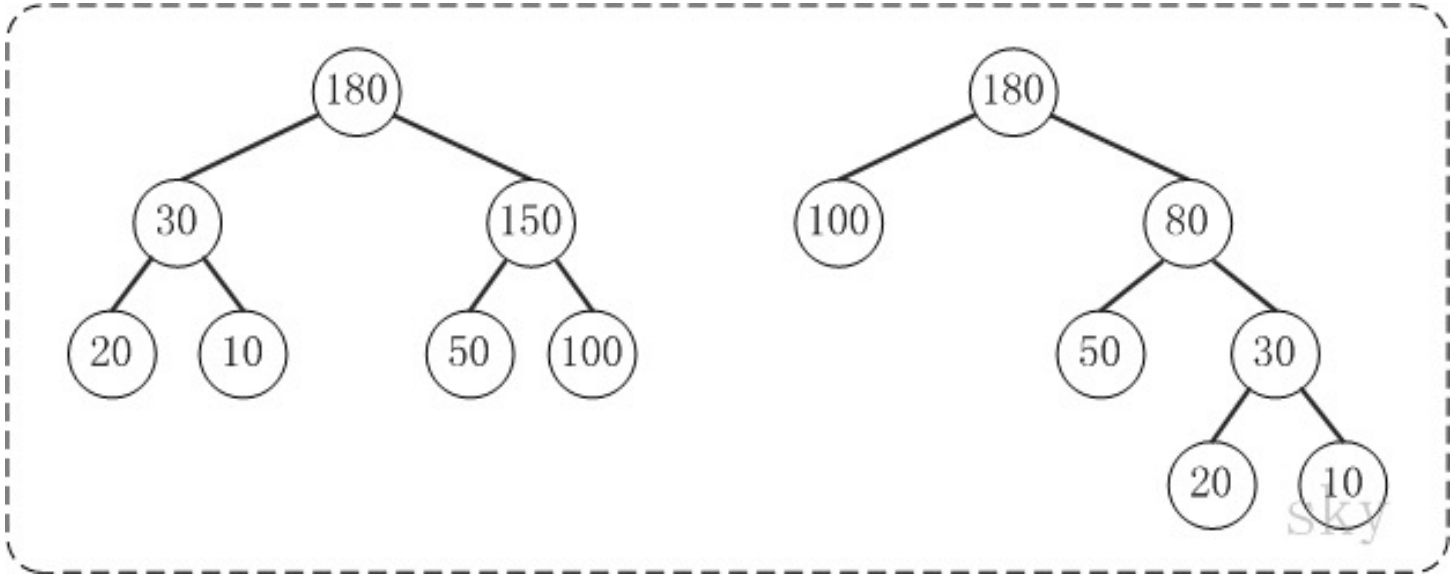
定义：给定n个权值作为n个**叶子结点**，构造一棵二叉树，若树的**带权路径长度（WPL）**达到最小，则这棵树被称为哈夫曼树。



例子：示例中，
WPL
= 1*100 + 2*50 + 3*20 + 3*10
= 100 + 100 + 60 + 30
= 290

例子

比较下面两棵树：



左边的树WPL= 2*10 + 2*20 + 2*50 + 2*100 = 360
右边的树WPL= 1*100 + 2*50 + 3*20 + 3*10 = 290

问题：基于带有权值的叶子结点w_1、w_2、…，w_n，怎样构建哈夫曼树？

构造哈夫曼树

算法思路：

（1）将w_1、w_2、…，w_n看成是有n 棵树的森林(每棵树仅有一个结点)；

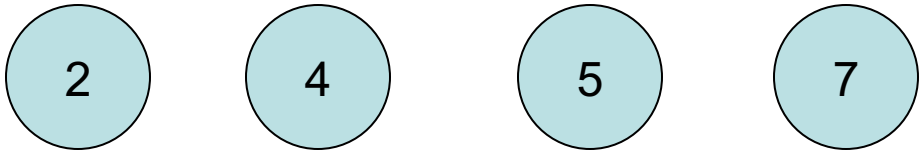
（2）在森林中选出根结点的**权值最小的两棵树进行合并**，作为一棵新树的左、 右子树，且新树的根结点权值为其左、右子树根结点权值之和；

（3）从森林中删除选取的两棵树，并将新树加入森林；

（4）重复(02)、(03)步，**直到森林中只剩一棵树为止**，该树即为所求得的哈夫曼树。

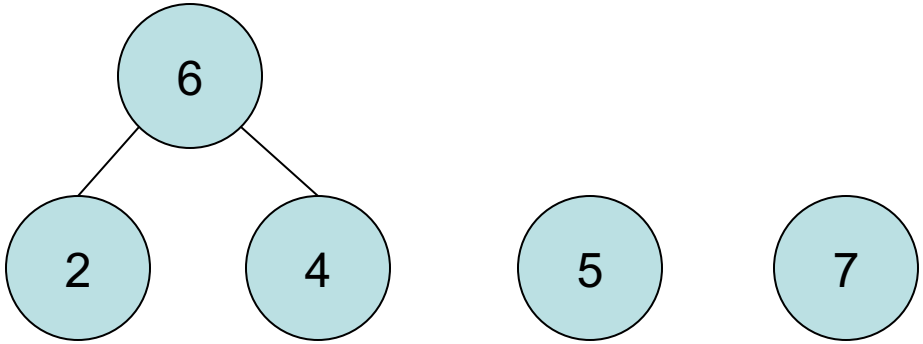
构造哈夫曼树示例

当权值为{7，5，2，4}时，构造哈夫曼树。



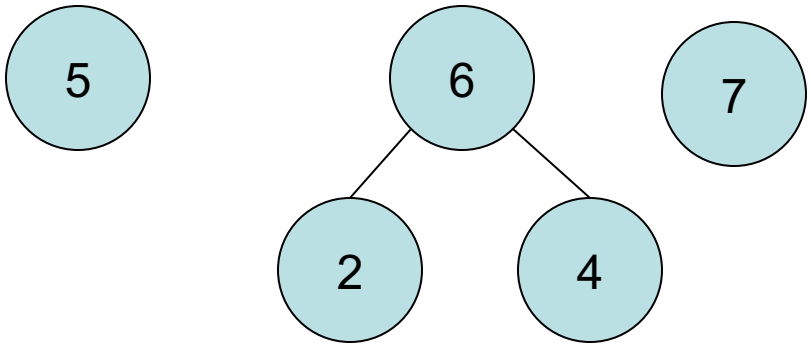
构造哈夫曼树示例

当权值为{7，5，2，4}时，构造哈夫曼树。



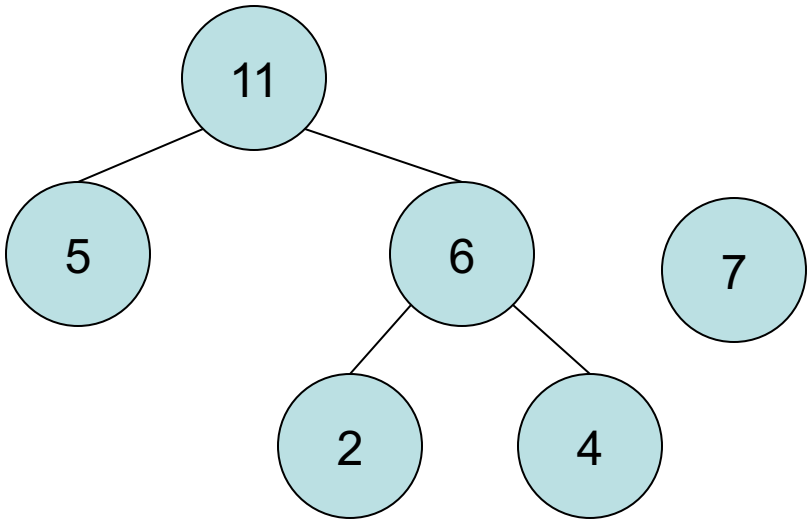
构造哈夫曼树示例

当权值为{7，5，2，4}时，构造哈夫曼树。



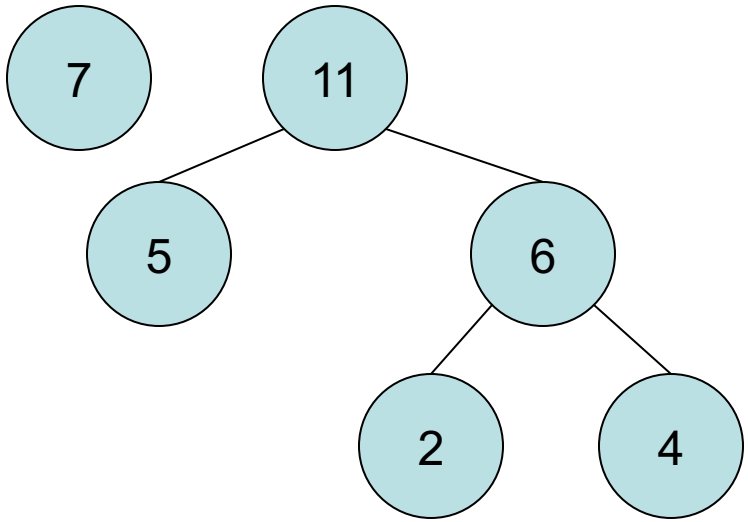
构造哈夫曼树示例

当权值为{7，5，2，4}时，构造哈夫曼树。



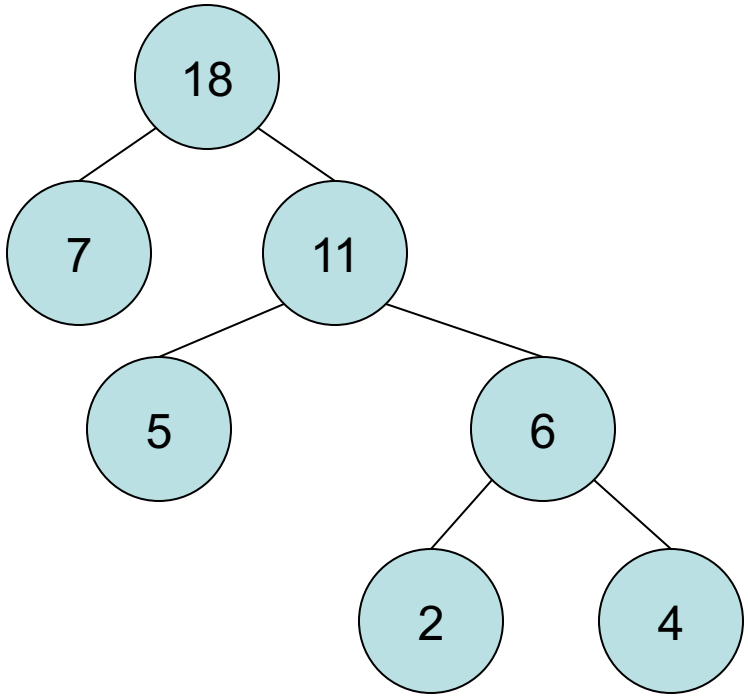
构造哈夫曼树示例

当权值为{7，5，2，4}时，构造哈夫曼树。



构造哈夫曼树示例

当权值为{7，5，2，4}时，构造哈夫曼树。



哈夫曼编码

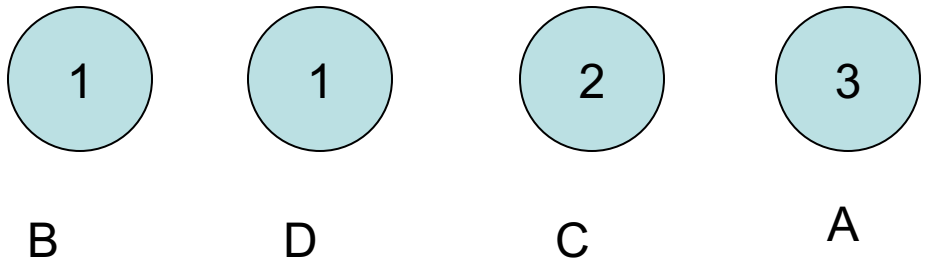
算法思路：

（1）利用字符集中每个字符（叶子结点）的使用频率作为权值构造哈夫曼树。

（2）从根结点开始，为其每个子结点赋予不同的**但顺序相同的**编码（如：左分支赋0，右分支赋1）。将从根结点到该叶子结点的路径编码作为该字符的编码。

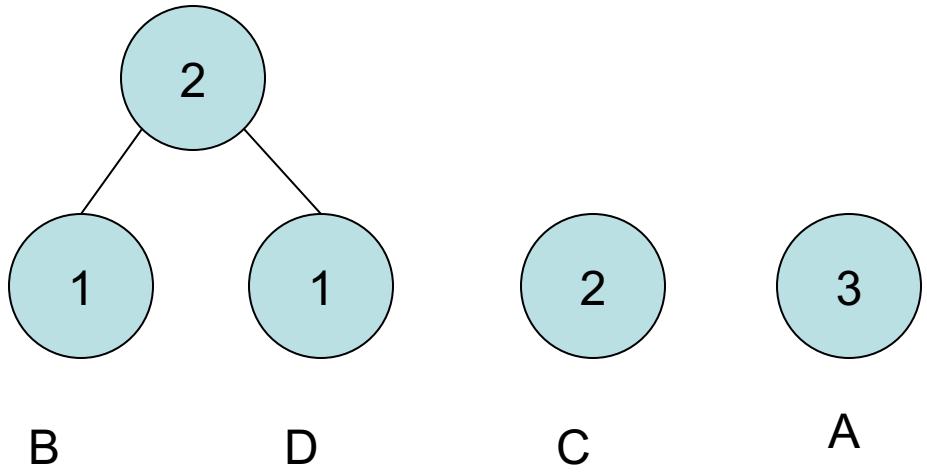
哈夫曼编码举例：ABACCDA

对于字符串“ABACCDA”，共有7个字符，4种字符。其中A、B、C、D出现的次数分别为3、1、2、1。根据权值{3，1，2，1} 构造哈夫曼树



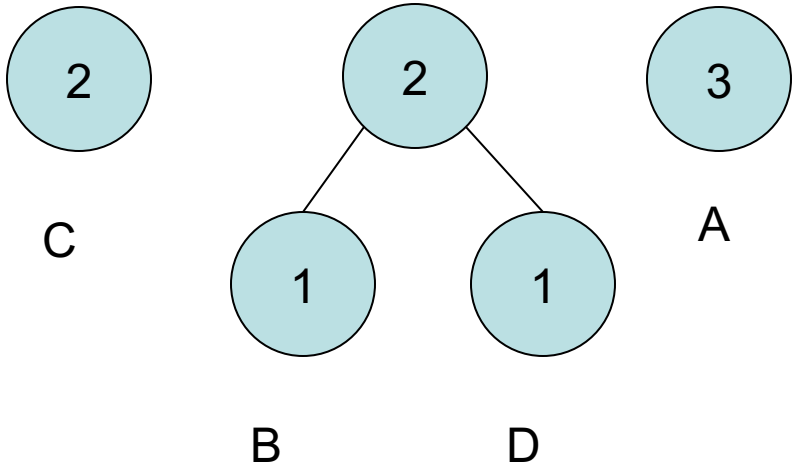
哈夫曼编码举例：ABACCDA

对于字符串“ABACCDA”，共有7个字符，4种字符。其中A、B、C、D出现的次数分别为3、1、2、1。根据权值{3，1，2，1} 构造哈夫曼树



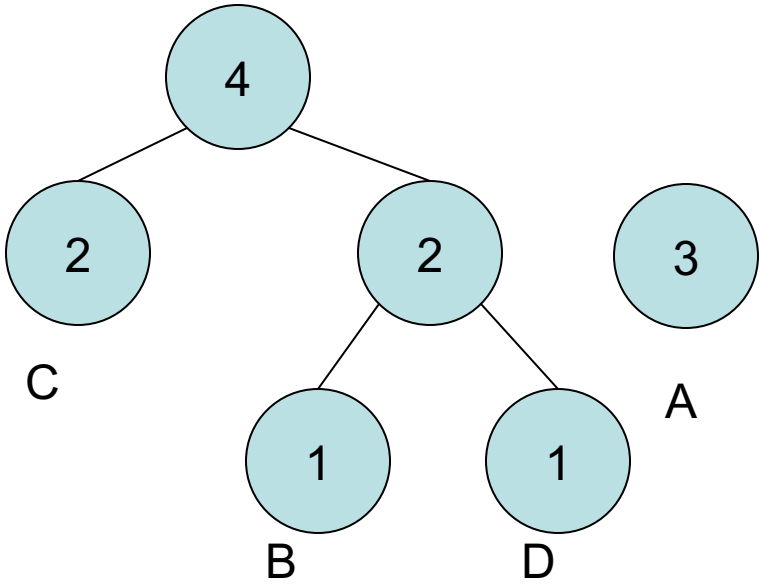
哈夫曼编码举例：ABACCDA

对于字符串“ABACCDA”，共有7个字符，4种字符。其中A、B、C、D出现的次数分别为3、1、2、1。根据权值{3，1，2，1} 构造哈夫曼树



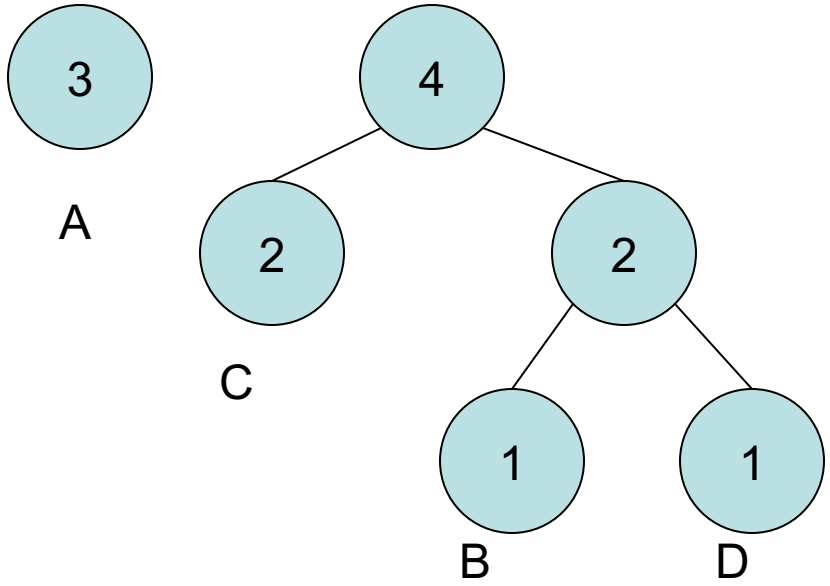
哈夫曼编码举例：ABACCDA

对于字符串“ABACCDA”，共有7个字符，4种字符。其中A、B、C、D出现的次数分别为3、1、2、1。根据权值{3，1，2，1} 构造哈夫曼树



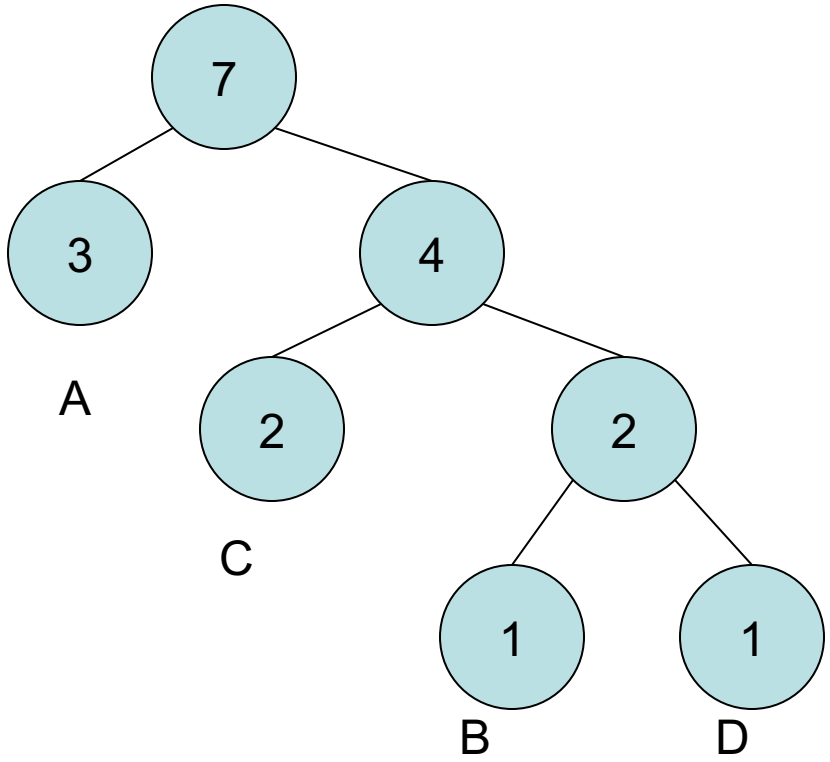
哈夫曼编码举例：ABACCDA

对于字符串“ABACCDA”，共有7个字符，4种字符。其中A、B、C、D出现的次数分别为3、1、2、1。根据权值{3，1，2，1} 构造哈夫曼树



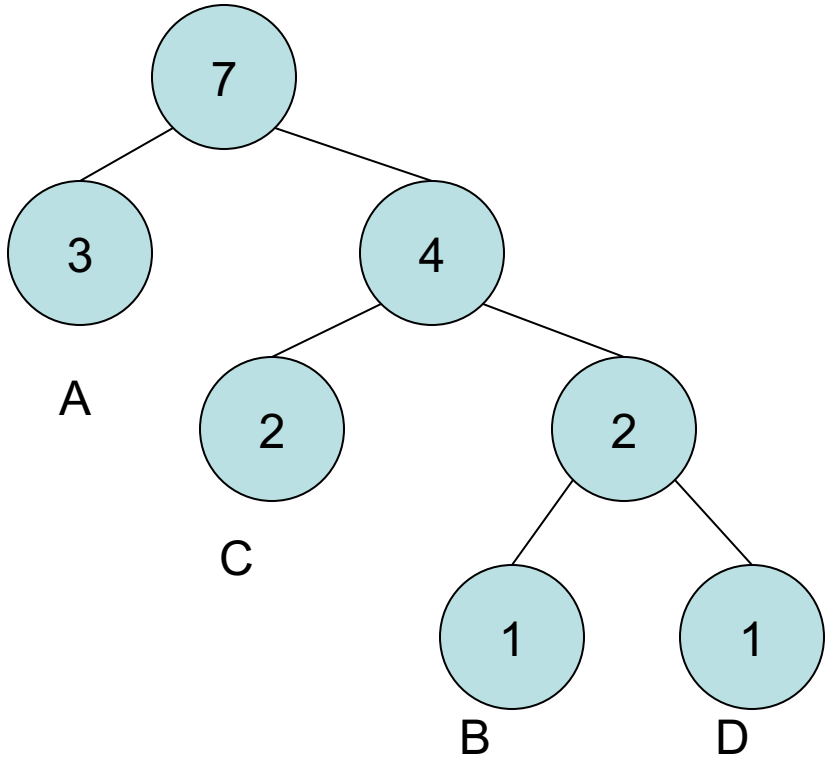
哈夫曼编码举例：ABACCDA

对于字符串“ABACCDA”，共有7个字符，4种字符。其中A、B、C、D出现的次数分别为3、1、2、1。根据权值{3，1，2，1} 构造哈夫曼树



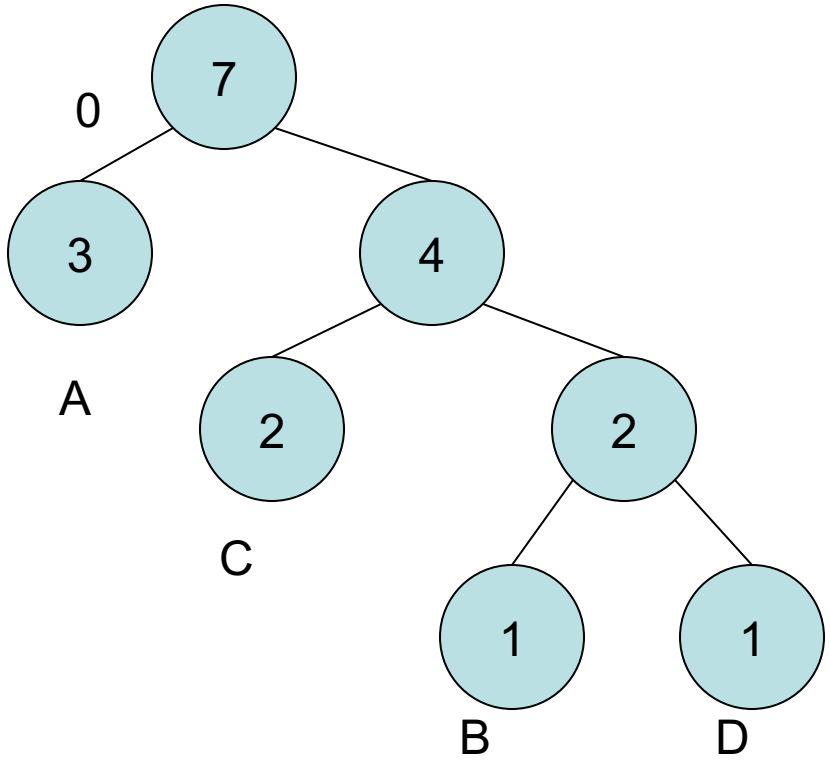
哈夫曼编码举例：ABACCDA

对哈夫曼树进行编码，每一个结点的左分支上面标0，右分支上面标1。



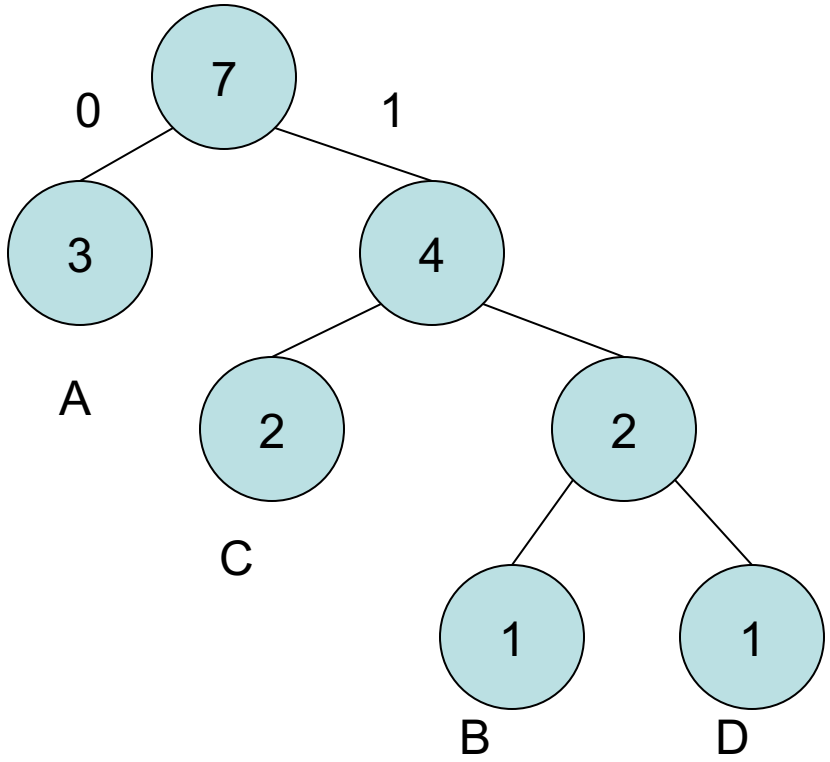
哈夫曼编码举例：ABACCDA

对哈夫曼树进行编码，每一个结点的左分支上面标0，右分支上面标1。



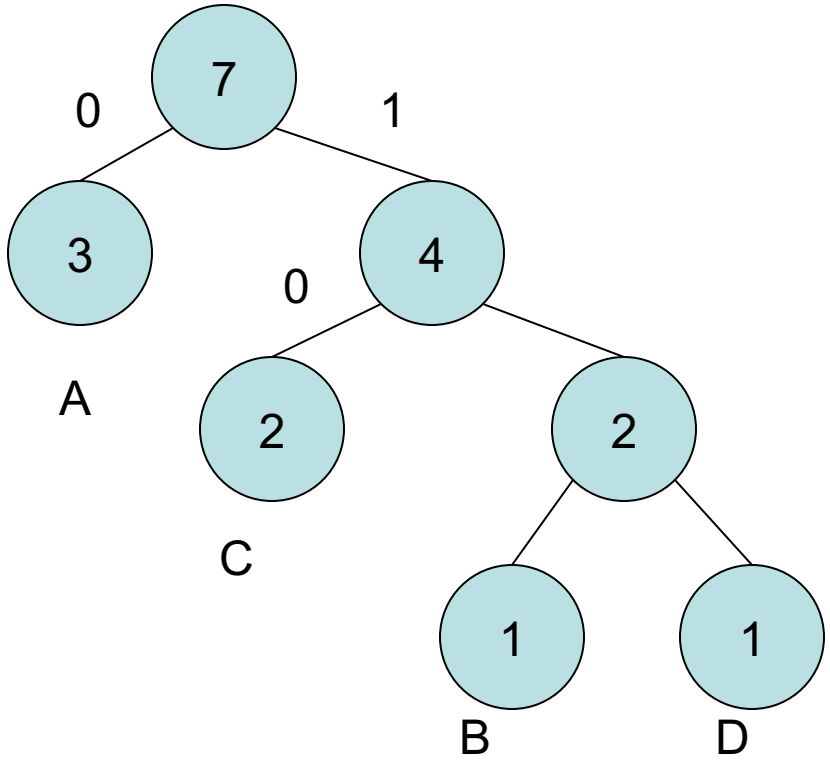
哈夫曼编码举例：ABACCDA

对哈夫曼树进行编码，每一个结点的左分支上面标0，右分支上面标1。



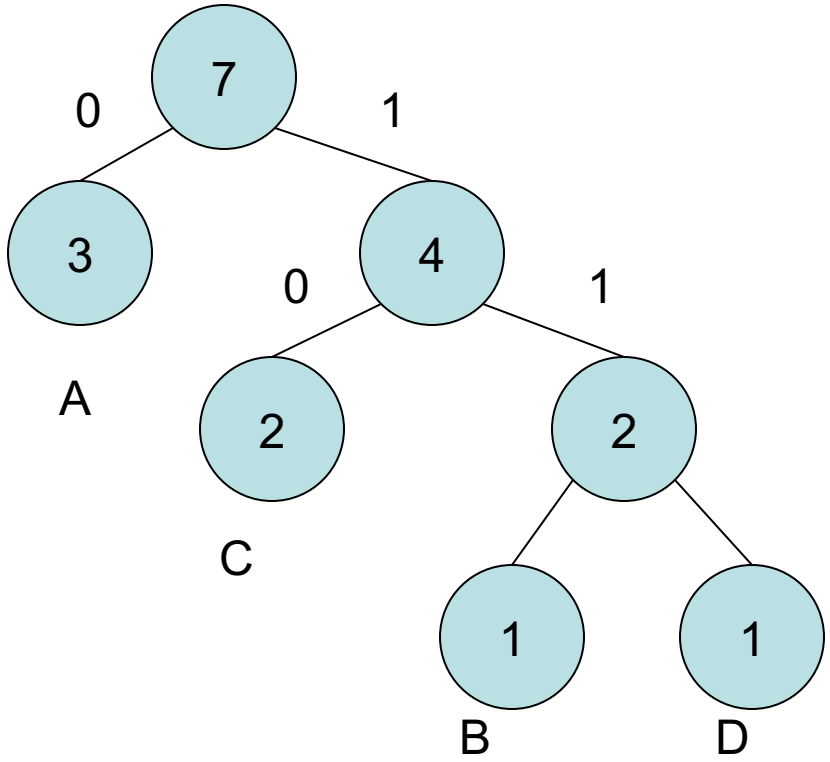
哈夫曼编码举例：ABACCDA

对哈夫曼树进行编码，每一个结点的左分支上面标0，右分支上面标1。



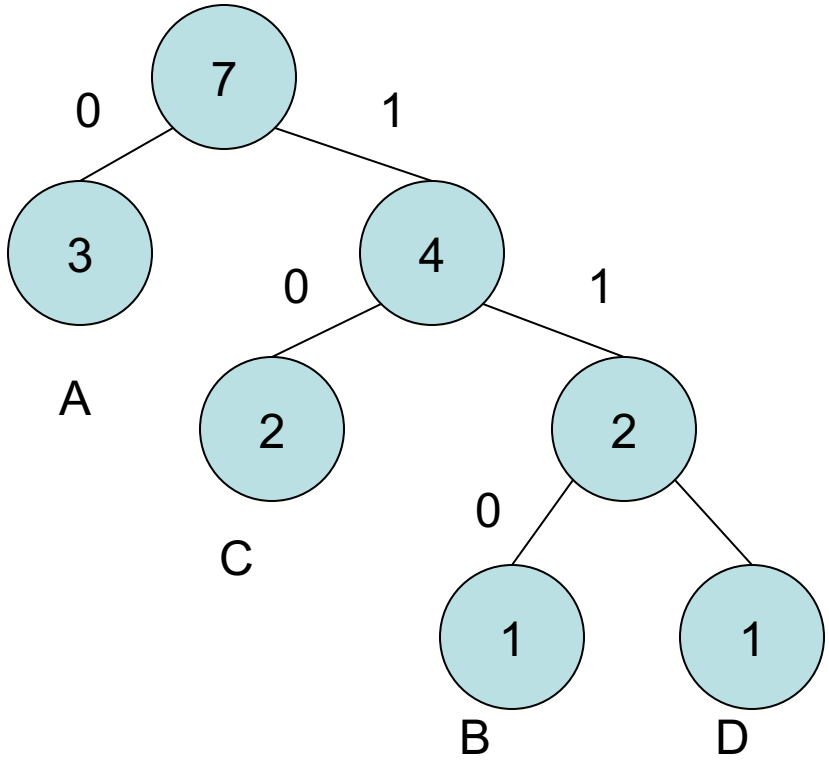
哈夫曼编码举例：ABACCDA

对哈夫曼树进行编码，每一个结点的左分支上面标0，右分支上面标1。



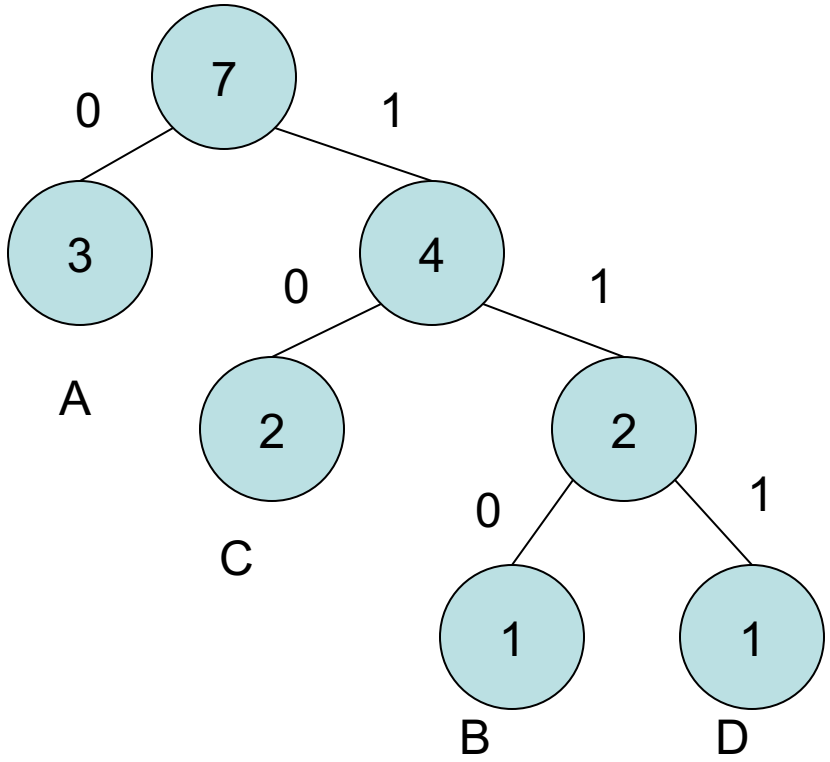
哈夫曼编码举例：ABACCDA

对哈夫曼树进行编码，每一个结点的左分支上面标0，右分支上面标1。



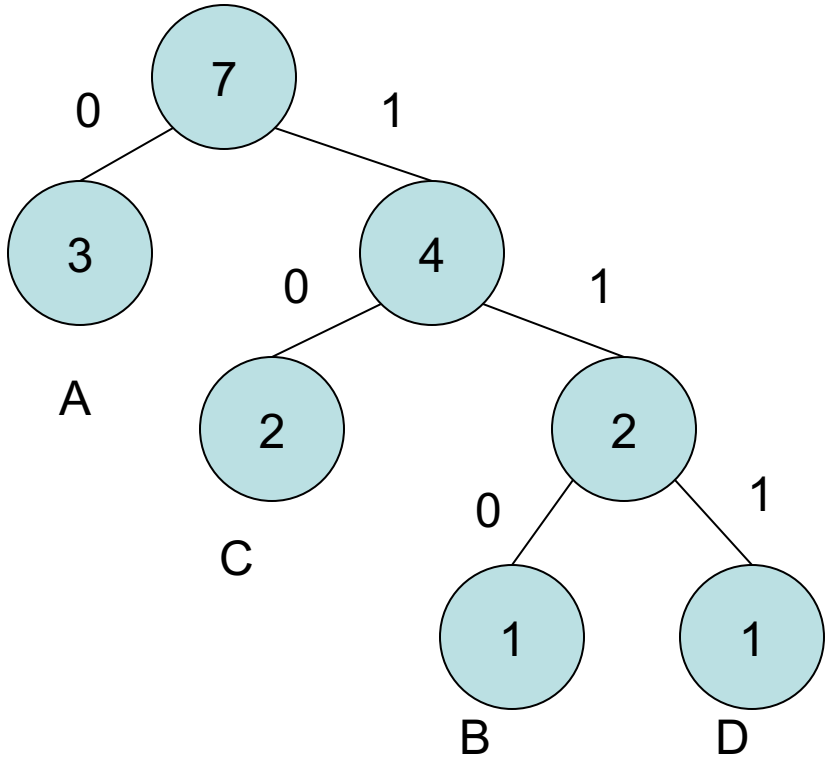
哈夫曼编码举例：ABACCDA

对哈夫曼树进行编码，每一个结点的左分支上面标0，右分支上面标1。



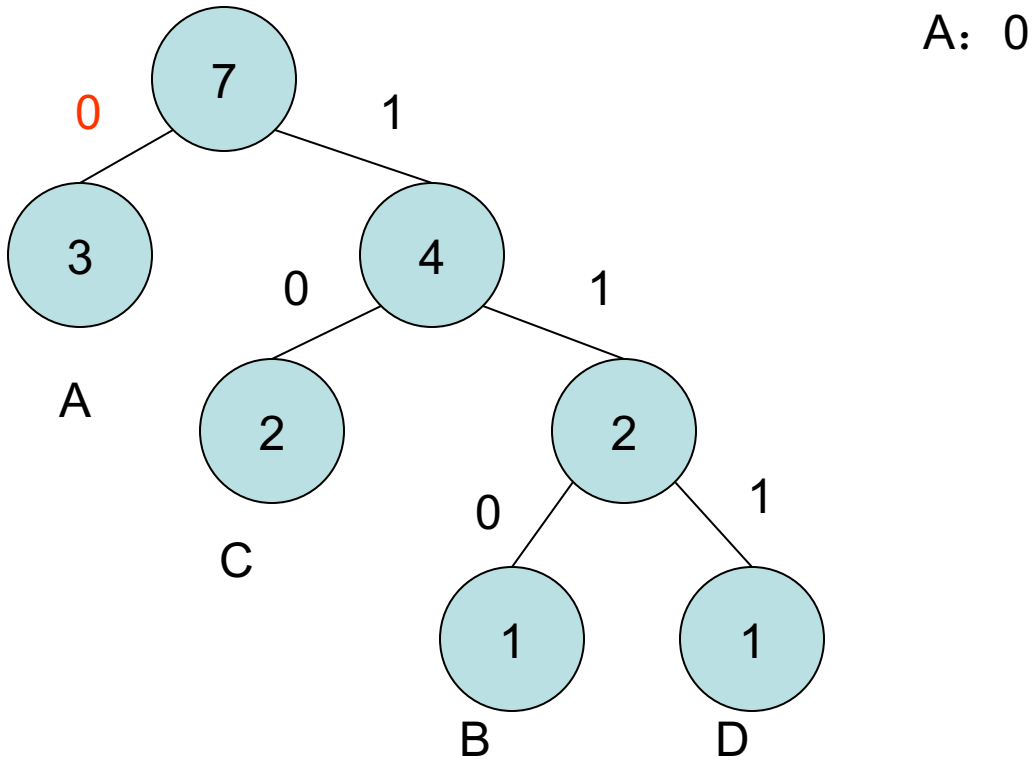
哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



哈夫曼编码举例：ABACCDA

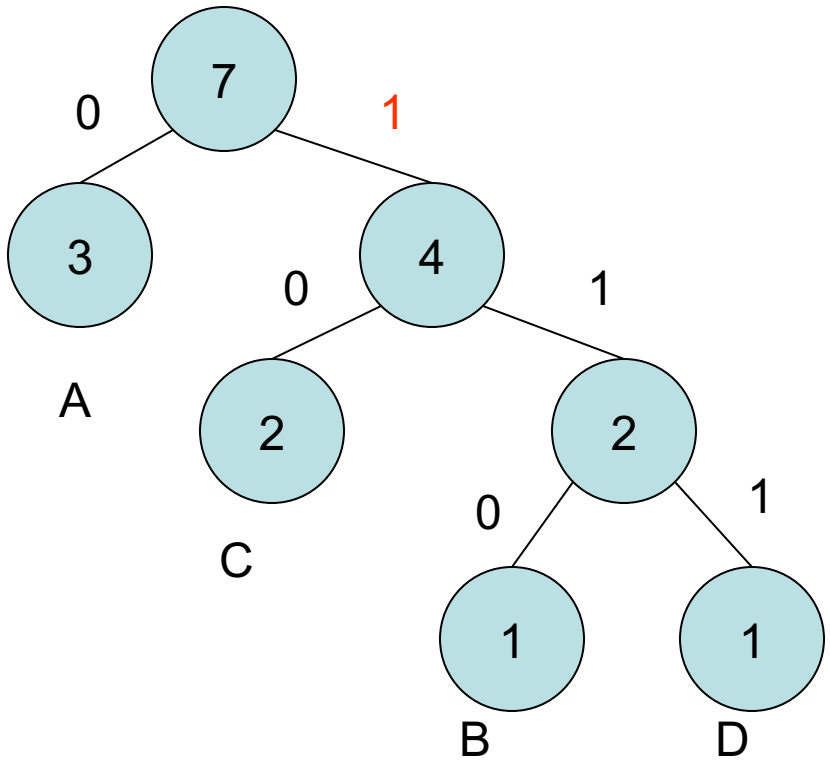
从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



A：0

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。

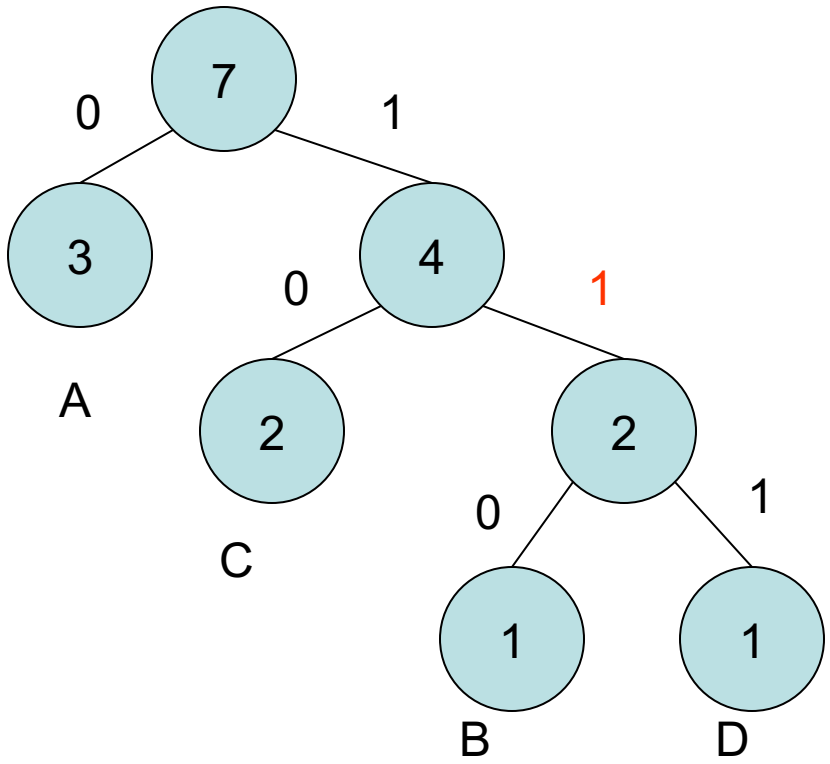


A：0

B：1

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。

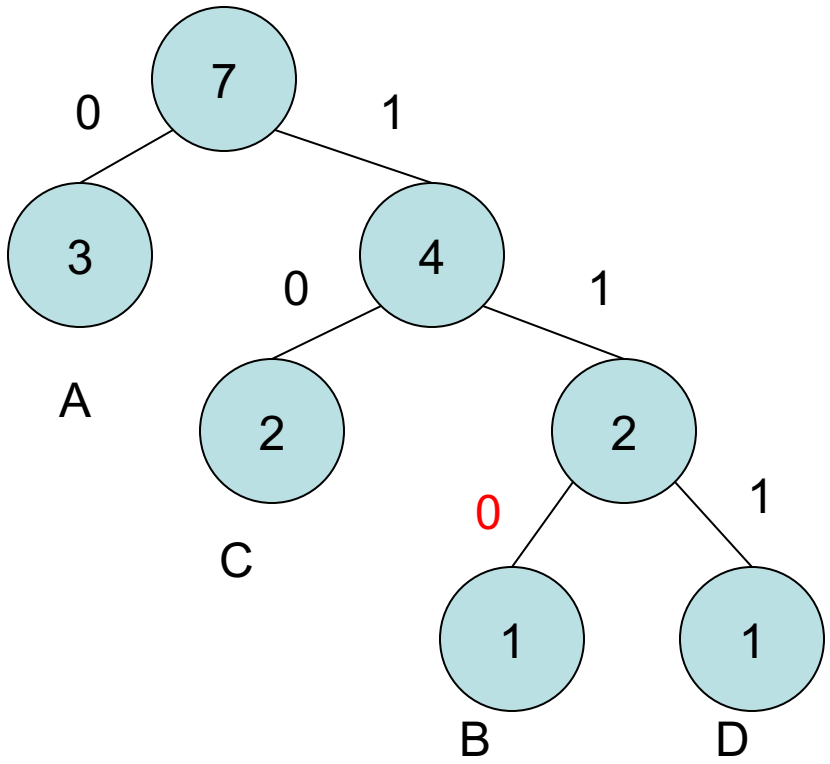


A：0

B：11

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。

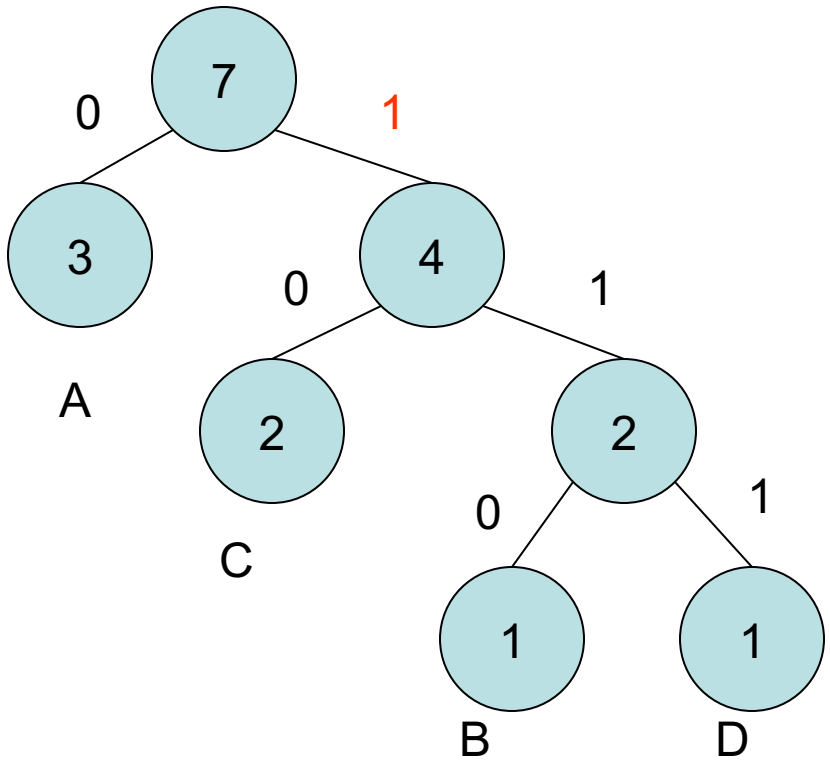


A：0

B：110

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



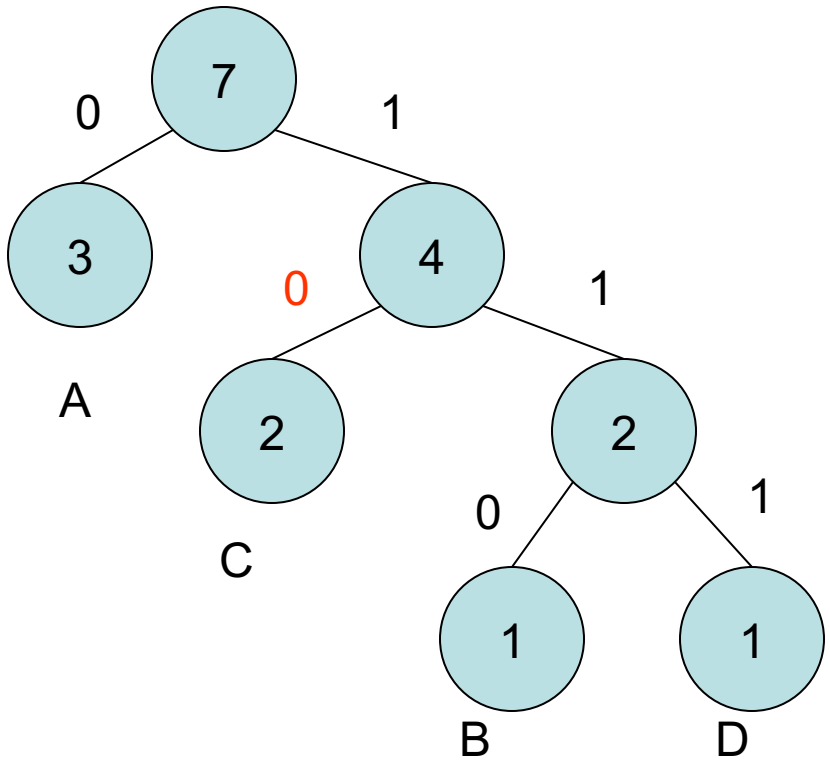
A：0

B：110

C：1

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



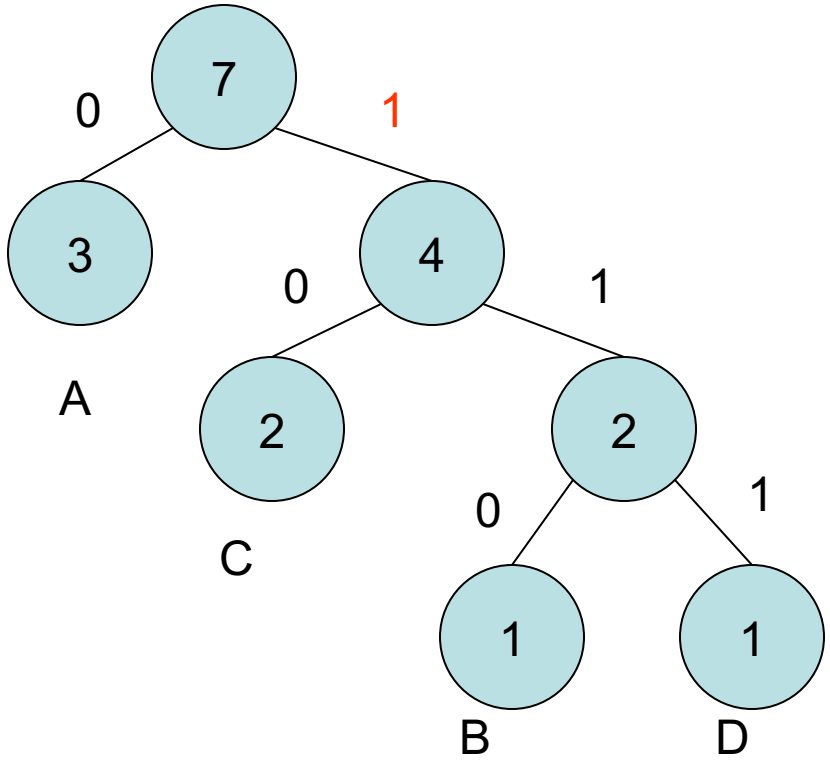
A：0

B：110

C：10

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



A：0

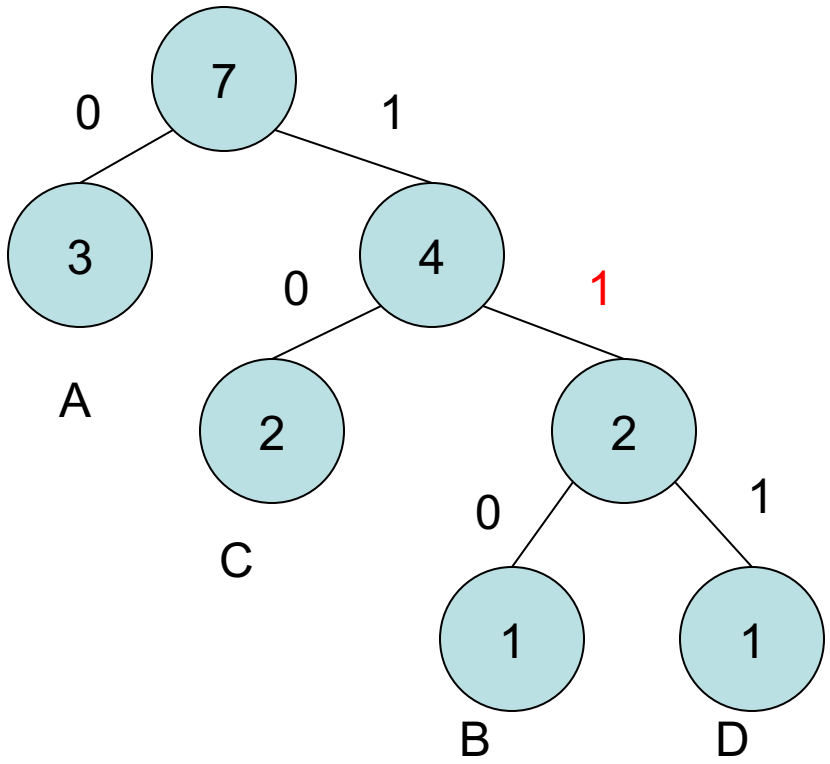
B：110

C：10

D：1

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



A：0

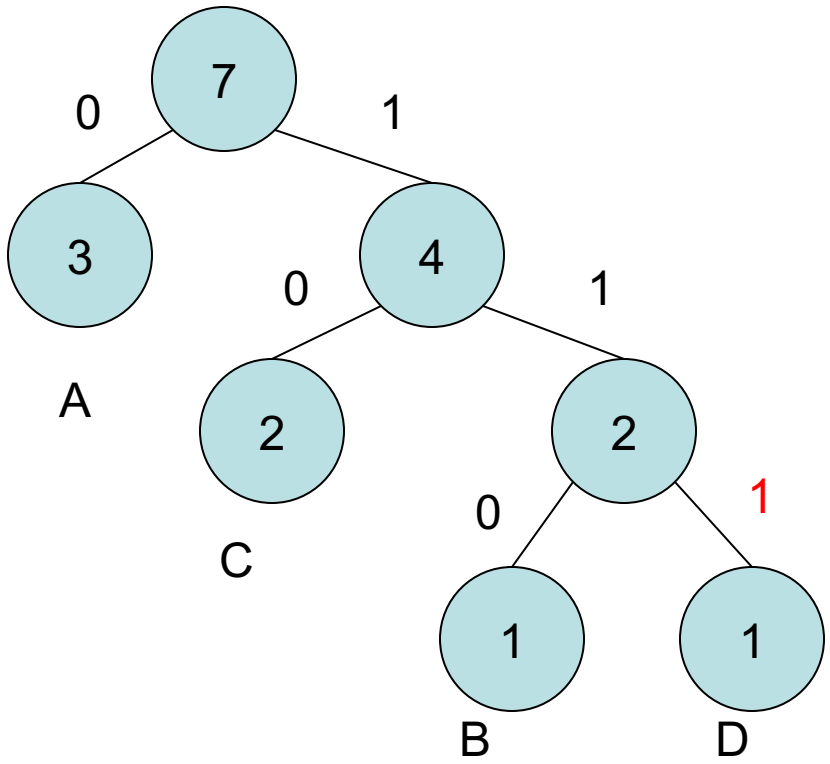
B：110

C：10

D：11

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



A：0

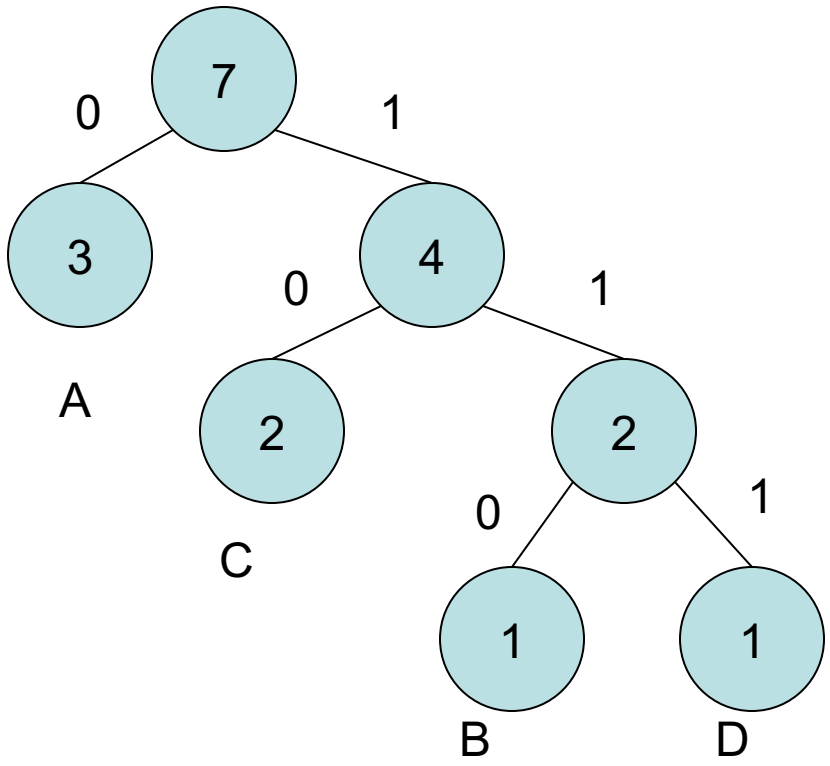
B：110

C：10

D：111

哈夫曼编码举例：ABACCDA

从根结点到各个叶子结点，所经分支上面的0、1序列，就是该叶子结点所代表字符的编码。



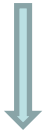
A：0

B：110

C：10

D：111

ABACCDA



0110010101110

使用哈夫曼编码进行文件压缩

1

项目概述与技术基础

目标

思路

基于单词编码进行文件压缩

使用哈夫曼编码进行文件压缩

1

项目概述与技术基础

目标

基于单词编码进行文件压缩

思路

统计词频

使用哈夫曼编码进行文件压缩

1

项目概述与技术基础

目标

基于单词编码进行文件压缩

思路

统计词频

⇩

根据字母出现次数？
根据汉字出现次数？

使用哈夫曼编码进行文件压缩

目标

基于单词编码进行文件压缩

思路

统计词频

⇩

~~根据字母出现次数？~~
~~根据汉字出现次数？~~

⇩

文件在内存都是二进制存储的
根据Byte出现次数

使用哈夫曼编码进行文件压缩

目标

基于单词编码进行文件压缩

思路

统计词频

⇩

~~根据字母出现次数？~~
~~根据汉字出现次数？~~

⇩

文件在内存都是二进制存储的
根据Byte出现次数

对文件以二进制方式读入，然后对每8个bit，即1个Byte（刚好对应256个ACSII字符）进行统计，这样就可以统计出文件对应的256个字符的权值。

使用哈夫曼编码进行文件压缩

目标

基于单词编码进行文件压缩

思路

统计词频

~~根据字母出现次数？~~
~~根据汉字出现次数？~~

文件在内存都是二进制存储的
根据Byte出现次数

根据Byte词频构建哈夫曼树

对文件以二进制方式读入，然后对每8个bit，即1个Byte（刚好对应256个ACSII字符）进行统计，这样就可以统计出文件对应的256个字符的权值。

使用哈夫曼编码进行文件压缩

目标

基于单词编码进行文件压缩

⇧

获得每个单词的编码

⇧

根据Byte词频构建哈夫曼树

思路

统计词频

⇩

~~根据字母出现次数？~~
~~根据汉字出现次数？~~

⇩

文件在内存都是二进制存储的
根据Byte出现次数

⇦

对文件以二进制方式读入，然后对每8个bit，即1个Byte（刚好对应256个ACSII字符）进行统计，这样就可以统计出文件对应的256个字符的权值。

使用哈夫曼编码进行文件压缩

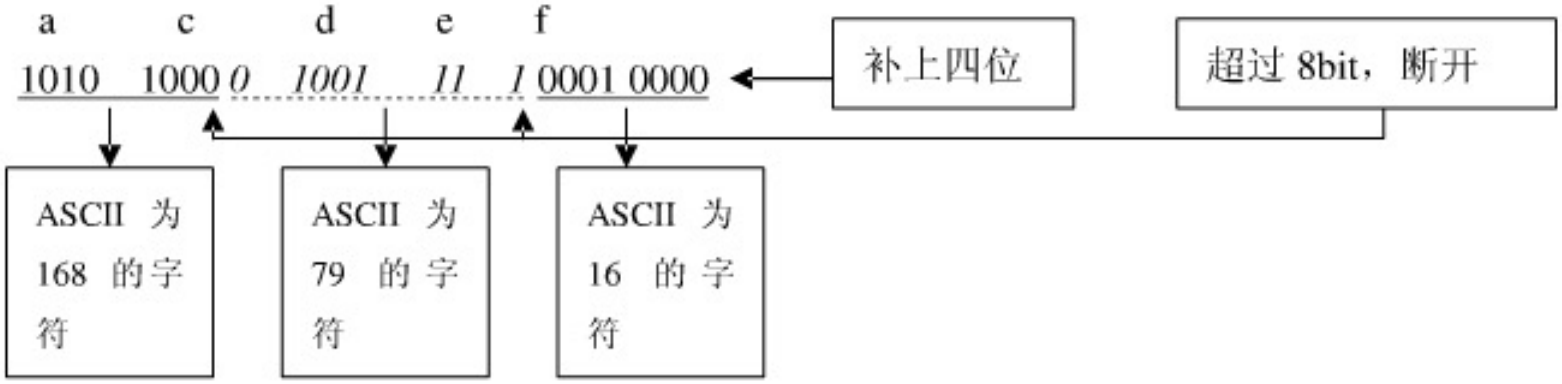
注意：不要用char数据类型来保存每一位编码。这样得到的编码实际上是由字符'0'和字符'1'组成的字符串，不但不能使文件得到压缩，反而会使文件增大。

解决方案：位操作！

举例如下：

已知：a: 1010　b: 00　c: 10000　d: 1001　e: 11　f: 10001　g: 01　h: 1001

如果其中的一段是:: acdef 则处理如下：



注意：需要记录真实编码位数或者补位的位数。否则，最后一个Byte无法正确解码。

使用哈夫曼编码进行文件压缩

1

项目概述与技术基础

需要存储的用于解码的信息：

1、编码之后的数据

2、编码后数据的真实长度

3、解码用数据

解码用数据存储方法

方法一：编码对照表

A：0

B：110

C：10

D：111

编码对照表存储方式：

字符+编码位长度+编码（补零）

解码用数据

方法一：编码对照表

A：0

B：110

C：10

D：111

编码对照表存储方式：

字符+编码位长度+编码（补零）

方法二：词频表

A：3

B：1

C：2

D：1

直接记录原始文件中各字符出现的频率，解码时读取该频率信息，重新生成一棵同样的哈夫曼树。

LZ77算法

基于统计的压缩编码算法（如Huffman编码）需要事先得到数据的出现频率。但有时（如流数据压缩），这种先验知识很难预先获得。因此，需要更为通用的压缩编码算法。

LZ77：基于滑动窗口的动态字典编码。

原理：利用数据的**重复结构信息**来进行数据压缩。

LZ77核心概念

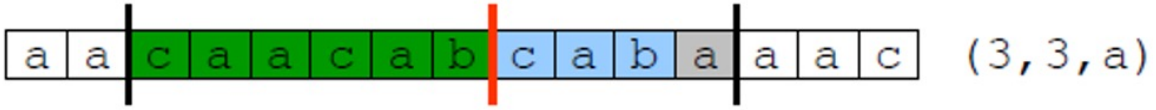
滑动窗口：任一时刻，编解码过程仅关注的数据段。

光标：当前需要编码的字符位置，也是字典区域与前向缓冲之间的边界

字典区域：随滑动窗口滑动，用作编码时的参考依据。

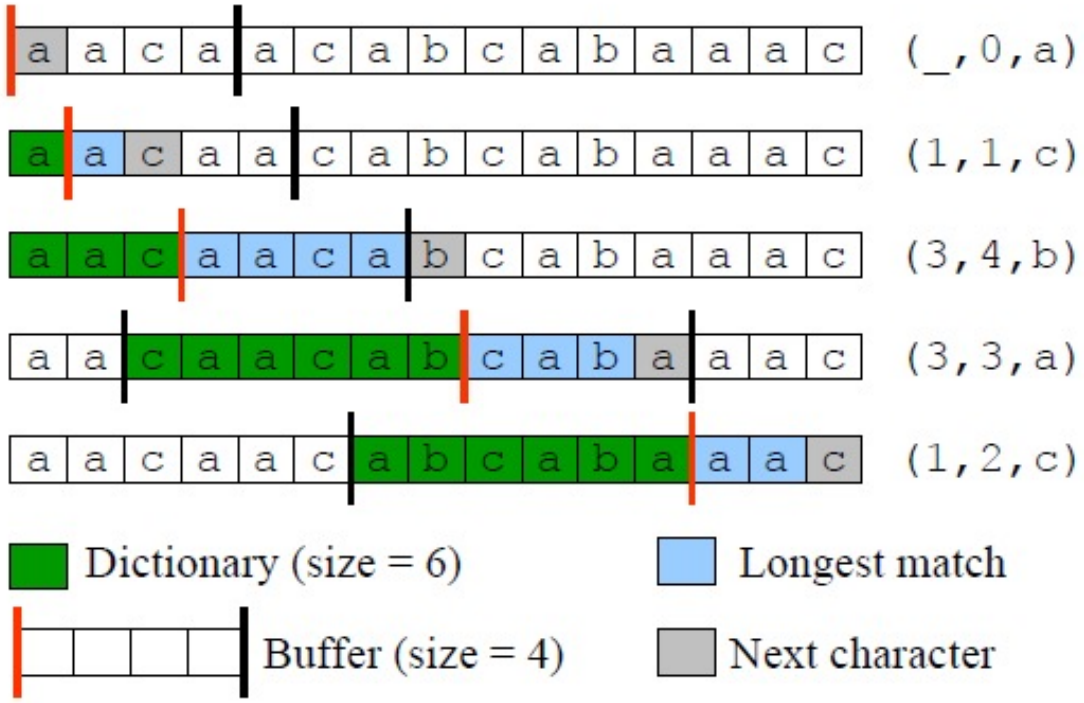
前向缓冲：随滑动窗口滑动，采用最大最长匹配算法来匹配字典（包括前向缓冲本身）中出现过的重复数据。

编码格式：在搜索到匹配数据之后（也可以没匹配到），将匹配结果进行编码。



LZ77算法举例

编码以下数据：aacaacabcabaaac



思考：

- 1、哈夫曼编码和LZ77算法各有什么优缺点？
- 2、在本项目中，应如何选择压缩算法？

技术基础

1

项目概述与技术基础

Unix文件系统

编码解码

打包解包

对称加密和散列算法

Unix文件系统事件感知

Unix网络编程

并发编程

文件打包

打包：将多个文件/目录整合为一个大文件的过程。

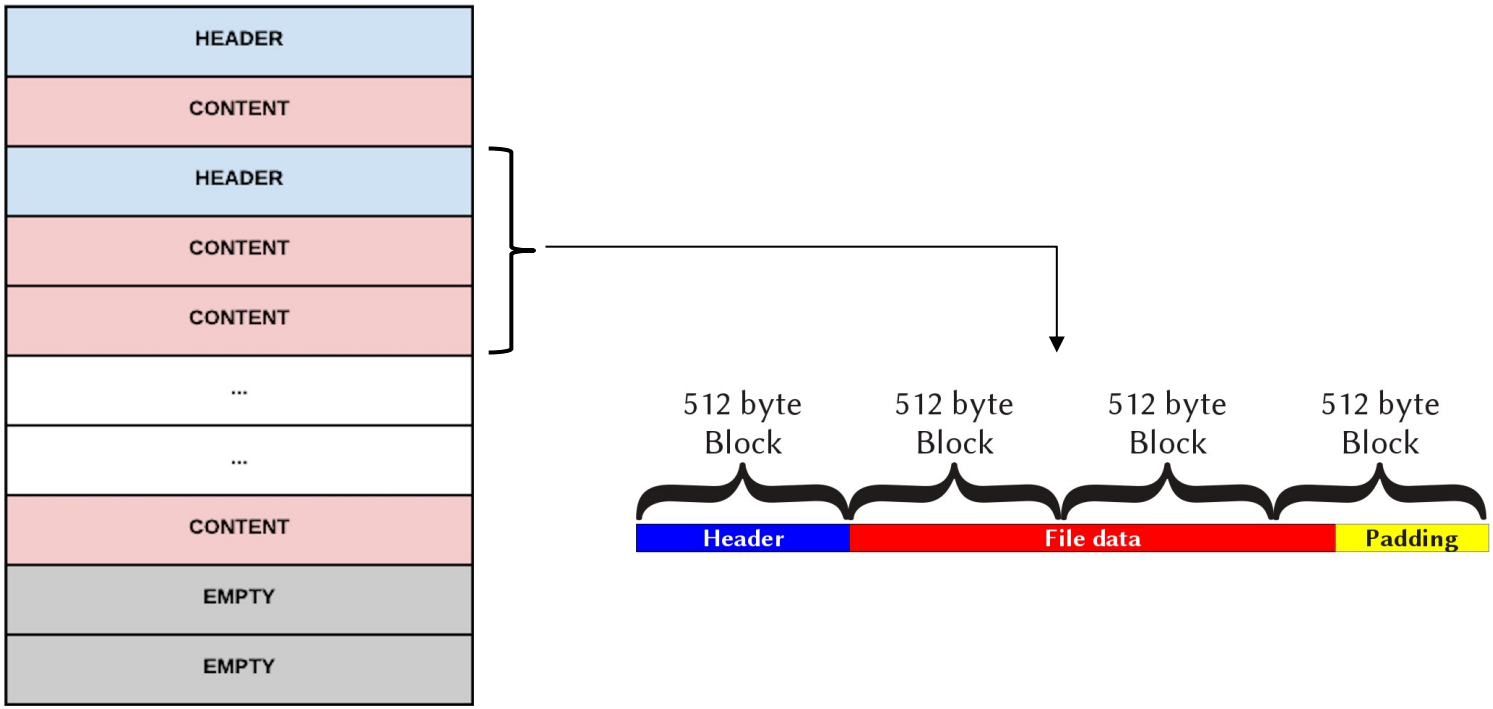
解包：将多个文件/目录从一个大文件中按原样恢复的过程。

大部分压缩软件均支持同时对文件打包。最经典的文件打包程序：Tar。

Tar打包原理

Tar是Unix和类Unix系统上的归档打包工具，可以将多个文件合并为一个文件，打包后的文件名亦为“tar”。目前，tar文件格式已经成为POSIX标准。

Tar归档文件由一系列文件对象通过线性排列的方式组合而成。每个文件对象都包含一个或者若干个512字节的记录块，并以一个头记录开头。文件数据按原样写入，但其长度舍入为512字节的倍数。



头记录数据结构


```c
union Record
{
    union
    {
        // Pre-POSIX.1-1988 format
        struct
        {
            char name[100];      // file name
            char mode[8];        // permissions
            char uid[8];         // user id (octal)
            char gid[8];         // group id (octal)
            char size[12];       // size (octal)
            char mtime[12];      // modification time (octal)
            char check[8];       // sum of unsigned characters in block (octal)
            char link;           // link indicator
            char link_name[100]; // name of linked file
        };

        // UStar format (POSIX IEEE P1003.1)
        struct
        {
            char old[156];      // first 156 octets of Pre-POSIX.1-1988 format
            char type;                  // file type
            char also_link_name[100]; // name of linked file
            char ustar[8];              // ustar\000
            char owner[32];             // user name (string)
            char group[32];             // group name (string)
            char major[8];              // device major number
            char minor[8];              // device minor number
            char prefix[155];
        };
    };

    char block[512]; // raw memory (padded to 1 block)
};
```

```c
#define REGULAR      0
#define NORMAL      '0'
#define HARDLINK    '1'
#define SYMLINK     '2'
#define CHAR        '3'
#define BLOCK       '4'
#define DIRECTORY   '5'
#define FIFO        '6'
```

打包算法流程

打包算法（输入：被打包的文件或目录）：

```
if 该元素为目录
        构造该目录的头记录，并写入打包目标文件
        打开该目录，递归执行该打包算法

if 该元素为普通文件
        构造该文件的头记录，并写入打包目标文件
        以512B为单位，复制文件到打包目标文件
        如最后一次读取时得到的数据不够512B，则补0
返回
```

在**打包算法**返回之后，需要在打包目标文件末尾追加两个全0记录，表示文件结束。

思考：

- 1、现有tar打包算法存在什么问题？
- 2、解包算法流程是什么样的？
- 3、怎样支持其他文件类型（硬链接、软链接、管道文件等）？

技术基础

Unix文件系统

编码解码

打包解包

对称加密和散列算法

Unix文件系统事件感知

Unix网络编程

并发编程

对称加密和散列算法

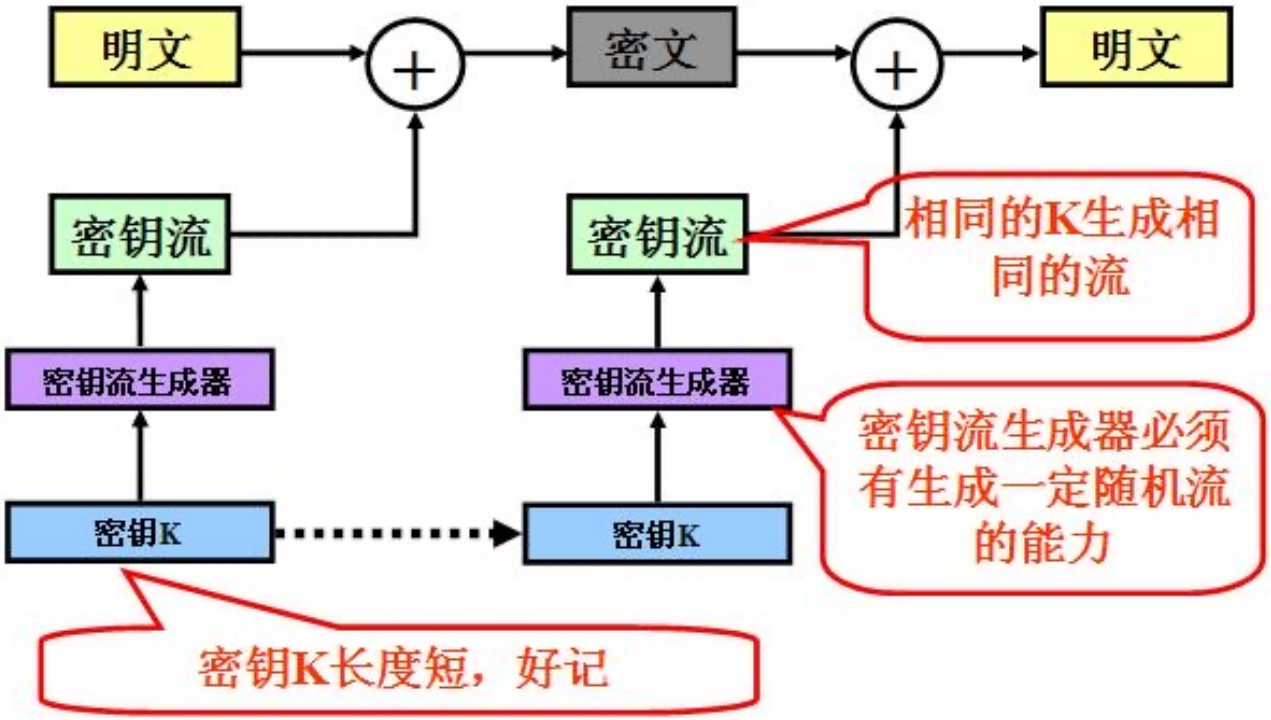
对于数据备份而言，主要考虑对称加密和散列算法。

散列算法也称为摘要算法，如MD5、SHA1等。

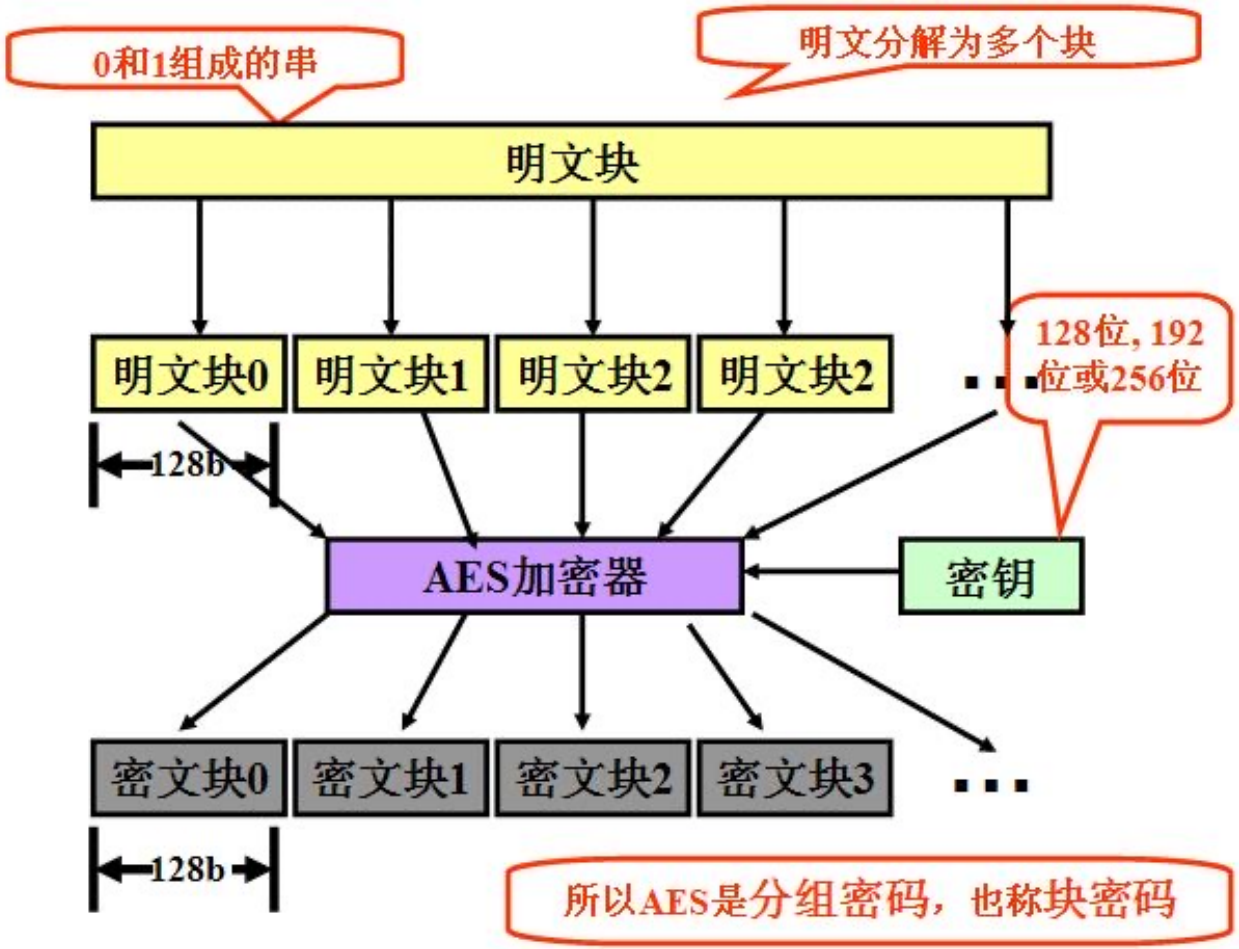
简单对称加密方式可考虑流密码。

更优的对称加密方式为分组加密，如DES、AES等。

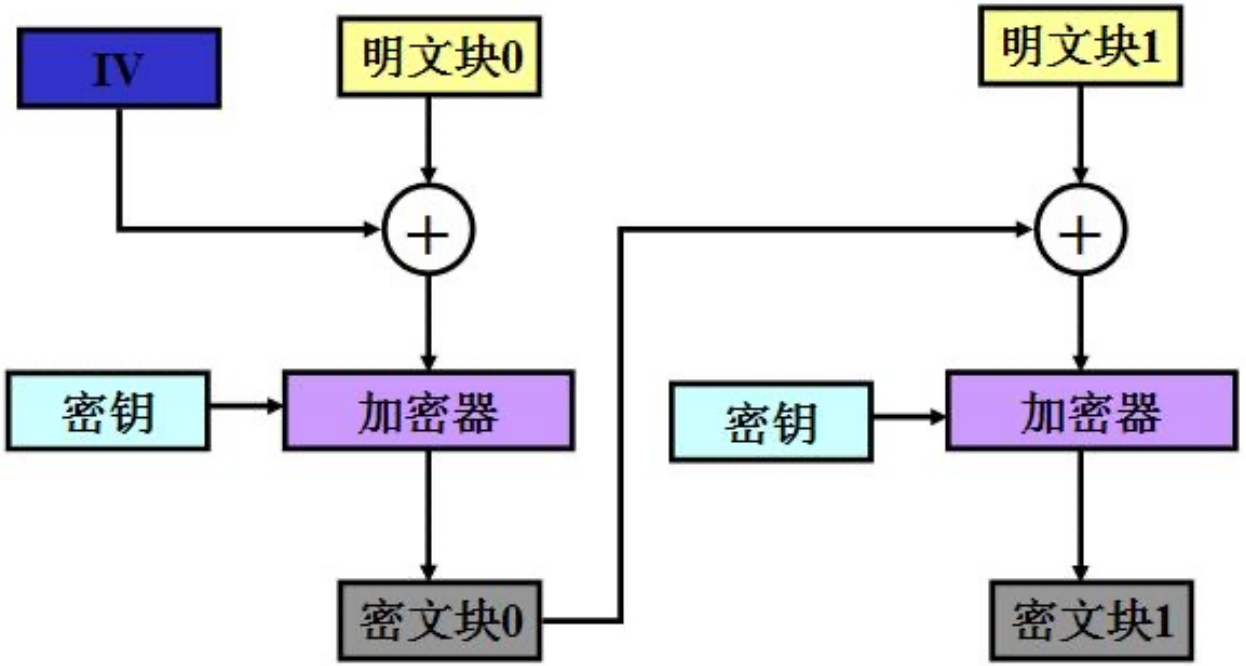
流密码


明文
密文
明文
密钥流
密钥流
相同的K生成相同的流
密钥流生成器
密钥流生成器
密钥流生成器必须有生成一定随机流的能力
密钥K
密钥K
密钥K长度短，好记

AES加密


0和1组成的串
明文分解为多个块
明文块
明文块0
明文块1
明文块2
明文块2
128位, 192位或256位
128b
AES加密器
密钥
密文块0
密文块1
密文块2
密文块3
128b
所以AES是分组密码，也称块密码

AES加密–CBC模式



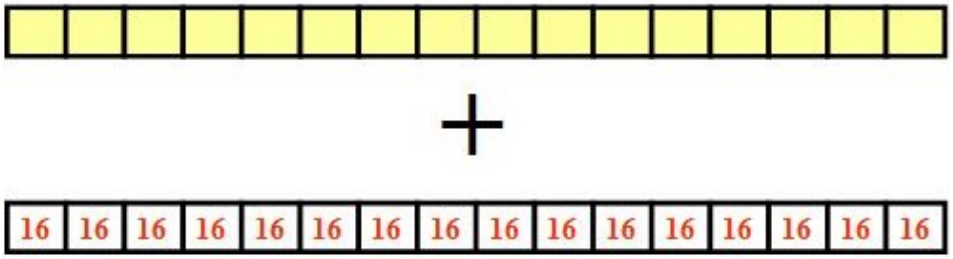
这种模式称为CBC模式，又密码分组链接

分组加密中的块补齐问题

一个AES块为16个字节（128位），需要考虑数据补齐问题。
主要分两种情况：

1、若数据的最后一块不够16个字节，则将最后一个数据块补齐为16个字节，填补的每个字符的内容为**填补字节的个数。**

2、若数据的最后一块刚好16个字节，则新增一块，该块中的每个字符的内容均为**块大小**，即16。



加解密常用库——OpenSSL

OpenSSL中的MD5、AES相关函数：

MD5_Init
MD5_Update
MD5_Final
AES_set_encrypt_key
AES_set_decrypt_key
AES_ecb_encrypt
AES_cbc_encrypt
AES_cfb128_encrypt
AES_ofb128_encrypt

技术基础

Unix文件系统

编码解码

打包解包

对称加密和散列算法

Unix文件系统事件感知

Unix网络编程

并发编程

Unix文件系统事件感知

现代操作系统均提供了对自身文件系统状态变化的通知机制，如
Windows中的FindFirstChangeNotification系列API
Linux中的inotify系列API

通过对文件系统中的某些文件和文件夹的监控，从而感知文件系统的各种变化。该机制特别适合本项目的实时备份需求。

以下对Linux中的inotify系列API进行介绍。

Inotify系列API

inotify是Linux内核中的一个子系统，提供了一种监控文件系统（基于inode的）事件的机制，可以监控文件系统的变化如文件修改、新增、删除等，并可以将相应的事件通知给应用程序，可以同时监控多个文件和目录。

inotify使用文件描述符作为接口，符合Linux中“一切皆文件”的设计思想，可以与Linux中的IO复用机制（如select、poll、epoll）一起使用。

其能够监控的事件包括：

```
IN_ACCESS：文件被访问
IN_MODIFY：文件被修改
IN_ATTRIB，文件属性被修改
IN_CLOSE_WRITE，以可写方式打开的文件被关闭
IN_CLOSE_NOWRITE，以不可写方式打开的文件被关闭
IN_OPEN，文件被打开
IN_MOVED_FROM，文件被移出监控的目录
IN_MOVED_TO，文件被移入监控着的目录
IN_CREATE，在监控的目录中新建文件或子目录
IN_DELETE，文件或目录被删除
IN_DELETE_SELF，自删除，即一个可执行文件在执行时删除自己
IN_MOVE_SELF，自移动，即一个可执行文件在执行时移动自己
```

Inotify系列API

涉及到的系统API（部分）：

inotify_init
inotify_init1
inotify_add_watch
inotify_rm_watch
read
close

注意：对于目录监控，Linux内核并不支持递归监控其中的文件和子目录，需要手动逐层添加监控。

技术基础

1

项目概述与技术基础

Unix文件系统

编码解码

打包解包

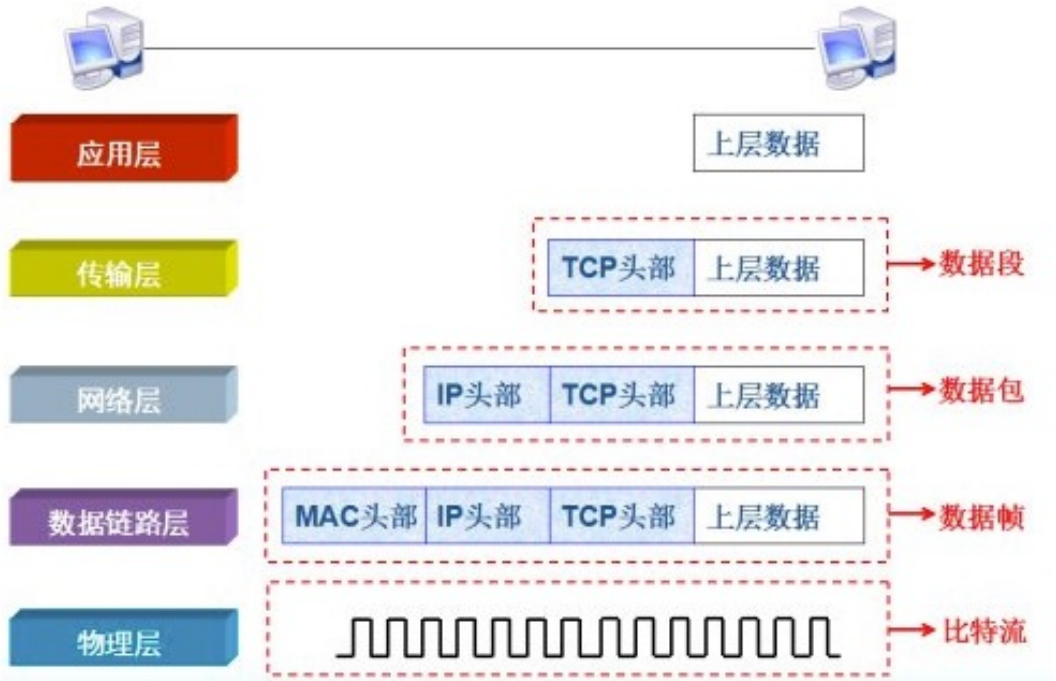
对称加密和散列算法

Unix文件系统事件感知

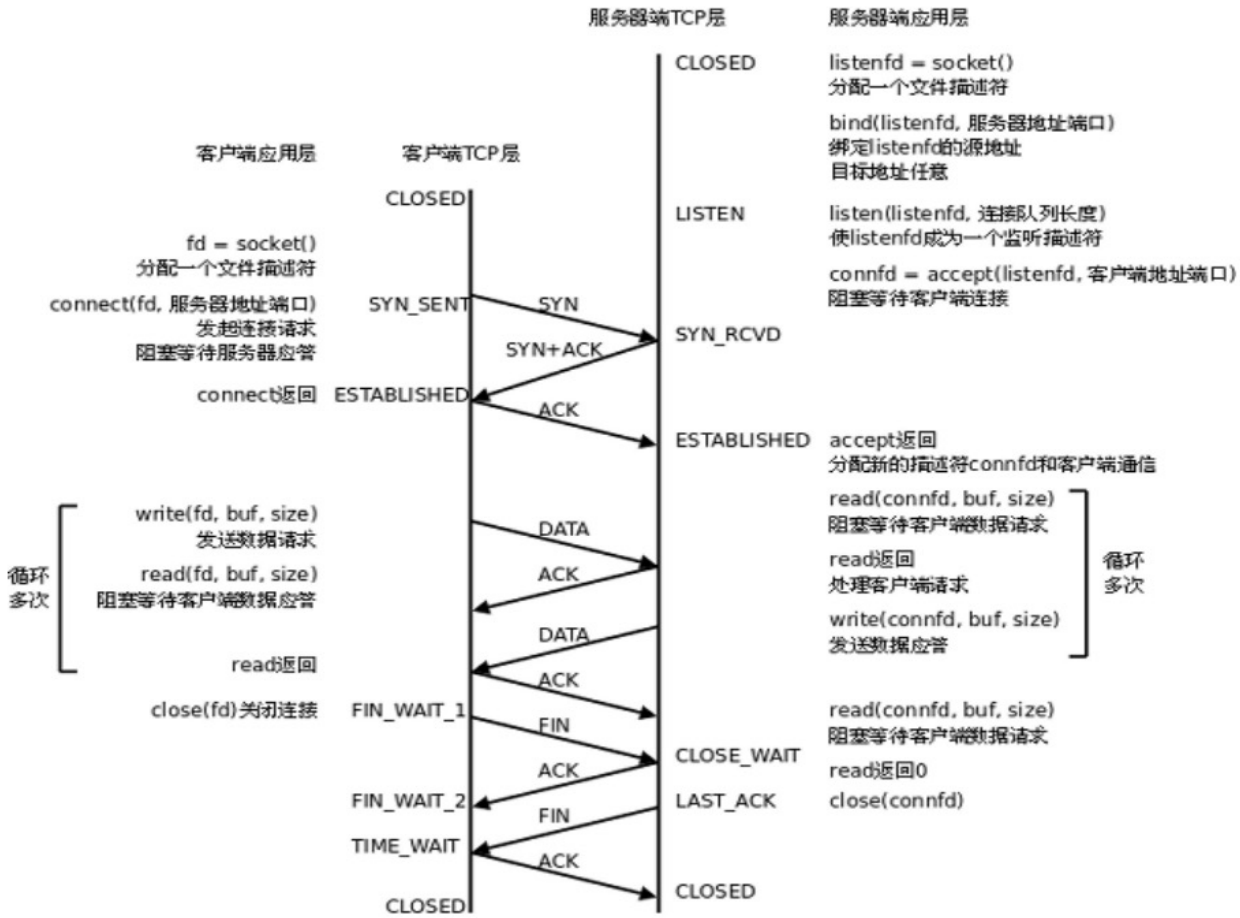
Unix网络编程

并发编程

计算机网络的层次结构


应用层
传输层
网络层
数据链路层
物理层
上层数据
TCP头部
上层数据
数据段
IP头部
TCP头部
上层数据
数据包
MAC头部
IP头部
TCP头部
上层数据
数据帧
比特流

客户端/服务器之间的交互



Unix套接字API

涉及到的系统API（部分）：

- socket
- bind
- listen
- connect
- accept
- read
- write
- close
- fcntl

网络编程模型

迭代模型
多进程模型
多线程模型
I/O复用模型（事件驱动）
信号驱动模型

技术基础

1

项目概述与技术基础

Unix文件系统

编码解码

打包解包

对称加密和散列算法

Unix文件系统事件感知

Unix网络编程

并发编程

Unix中的并发编程

并发编程的意义：提高性能、业务拆分、同时处理多个事件
现代操作系统通常都提供了以下三种并发编程机制：

进程：为每一个执行流单独分配一个进程执行。
线程：为每一个执行流单独分配一个线程执行。
I/O复用：借助I/O复用机制，集中监控多个待处理事件，并进行异步处理。

三种并发编程机制各有优缺点：
进程：编程逻辑最为简单，开销最大，不利于资源共享
线程：编程逻辑比进程机制复杂，开销稍小（不适用与Linux），资源共享方便，但容易出错。
I/O复用：编程逻辑最为复杂，开销最少，资源共享方便。

多进程并发编程API

涉及到的系统API：

fork
wait族
exec族
exit
signal

多进程编程需要注意的问题：信号、文件描述符共享

多线程并发编程API

涉及到的系统API：

pthread_create
pthread_exit
pthread_join
pthread_detach
pthread_self
pthread_cancel
pthread_mutex_init
pthread_mutex_destroy
pthread_mutex_lock
pthread_mutex_trylock
pthread_mutex_unlock
pthread_cond_init
pthread_cond_destroy
pthread_cond_wait
pthread_cond_signal
pthread_cond_broadcast

多线程编程需要注意的问题：互斥、同步、死锁

其它知识点

GUI编程（C++ -> QT、python -> PyQt ……）
数据库编程（连接、SQL）
软件开发环境的搭建
动/静态链接库的使用和开发
程序构建方法（makefile、autoconfig、cmake）
程序部署方法（docker）

实验环节

1、以小组为单位，安装各自的软件开发环境，包括：

- 操作系统（推荐虚拟环境）
- 开发语言（编译器）
- 依赖库
- 构建工具

注意软件开发环境组内统一（包括版本统一），推荐基础设施即代码。

2、思考自身项目可能选择的功能点，并对相关功能点进行实验性代码编写，确保项目本身的技术可行。相关技术基础包括：

- Unix文件系统
- 编码解码
- 打包解包
- 对称加密和散列算法
- Unix文件系统事件感知
- Unix网络编程
- 并发编程
- GUI编程
- 数据库