Huffman编码实现压缩和解压缩软件

chensihang

Part0 前言

相信同学们在平常的生活中经常使用压缩软件对文件进行压缩，以更小但完备的姿态去面对令人焦灼的网速。在本项大作业中，我们将带领大家完成一个压缩软件的编写，当然其远远达不到一个商业软件的标准，但是我们更希望同学们能够感受软件编写的内核，体会模块化编写的重要性。

让我们从一点点前置知识开始吧！

Part1 前置知识

图的定义

什么是图？

我们从考虑图1和图2开始，他们描绘的是一幅道路图和一幅电路图。

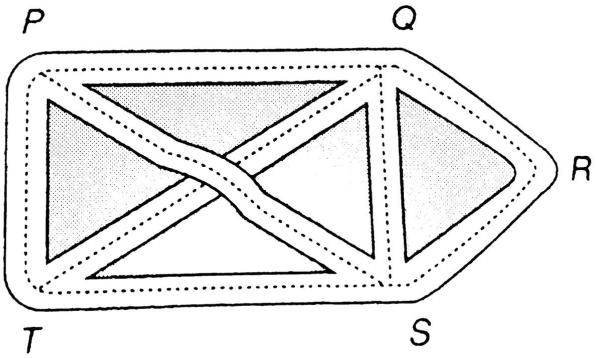


图 1

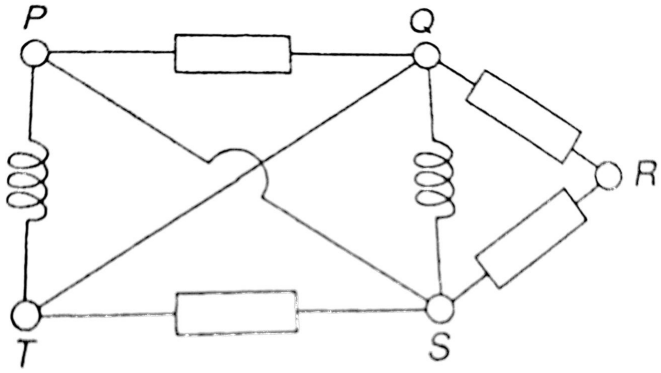


图2

这两种情况可以通过点和线用一种几何化的方式来表述，就像在图3中做的那样。点P,Q,R,S,T,被称作**顶点（vertex）**，这些线被称作**边（edge）**，以及这整一个形状被成为**图（graph）**。注意！PS和QT的交点不是所谓的顶点，因为这个点并不代表道路图中的交叉路口或者两条电线的交点。顶点的**度（degree）**是指以该点为端点的边的数量。在图1中它代表了某个交叉路口的路的数量。比如说，顶点P的度是3，顶点Q的度是4。

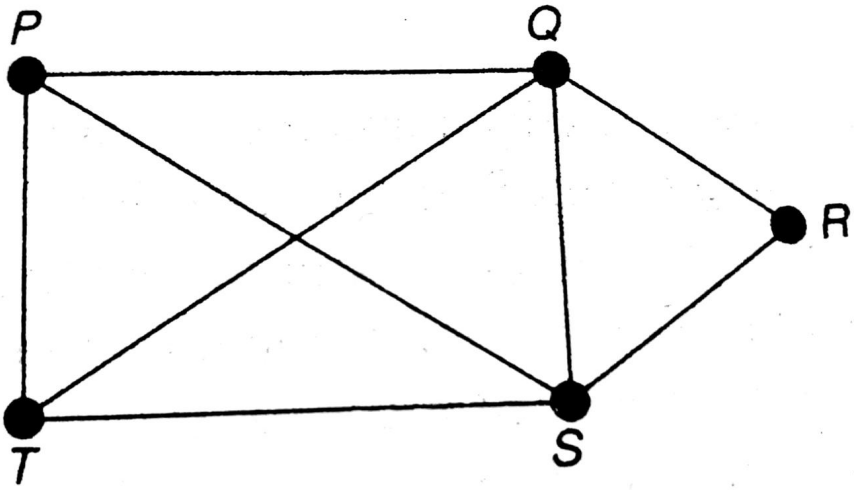


图3

图三中的图也可以代表其他情况。比如说，如果顶点P,Q,R,S,T代表足球队，那么边就可以代表两队之间的比赛。如此，在图3中队P与队Q，S，T比赛，但是没有和队R比赛。在这种情况下。一个顶点的度表示了与这个队伍比赛过的队伍的数量。

另一种描绘这种情况的方法是用图4中的图。通过重新把PS这条线画到长方形PQST外面，我们就可以移去边PS和QT的交点。然后得到的图仍然可以告诉我们这里是否有一条连接两个路口的直接的路，电路如何被电线连接起来，以及某队如何与其他队伍比赛。我们所损失的只是一些物质上的属性，比如道路的长度与电线的曲直情况。

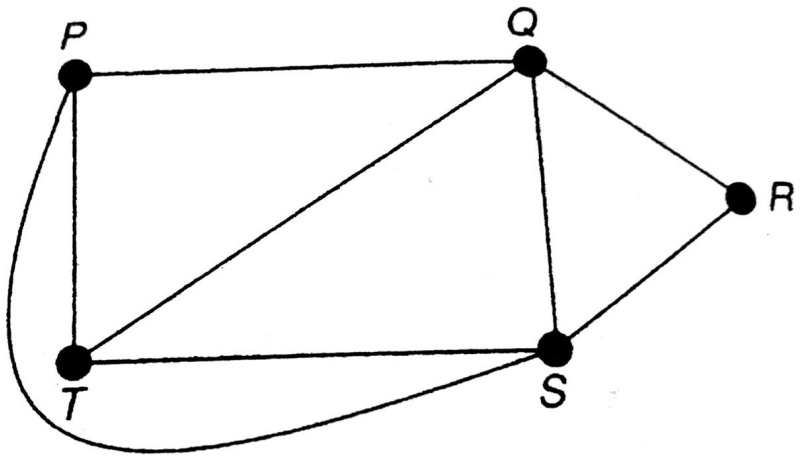


图4

更一般地，两个图等价与否只取决于在一幅图中连接两个顶点的边是否与另一幅图中连接两个顶点的边一一对应。在图5中给出了另一幅与图3图4相同的图。在这里，空间与距离的概念不再被考虑了，但是我们仍然能一眼就能看出两个点是否被路或者电线而连接。

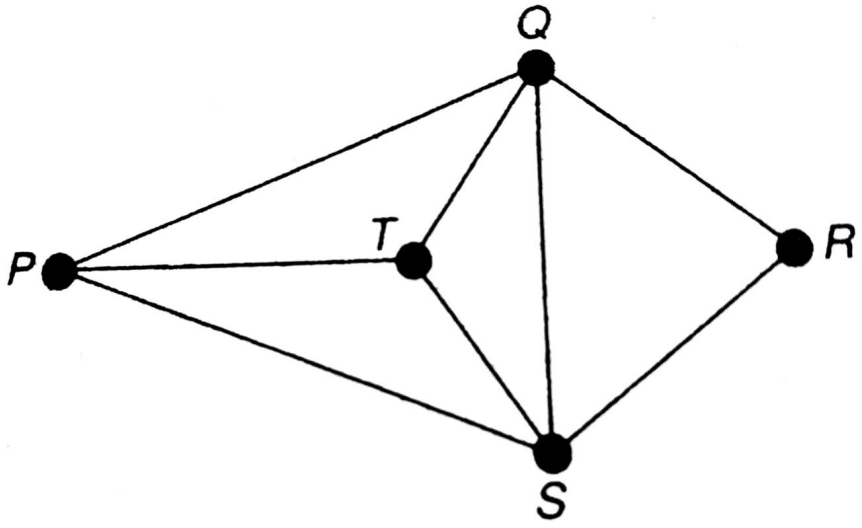


图5

在这个图中至多只有一条边连接了一对点。假设现在在图5中的连接Q，S的路和S，T的路要承载太多的车流量了。那么我们可以考虑建设更多连接这些点的道路来缓解交通压力，结果图就像图六一样，这些连接Q和S，S和T的边被成为**重边（平行边/multiple edge）**。在此之上，如果我们需要建设一个在P的停车场，那么我们可以通过画一条连接P与它自身的边来表示这种情况。叫做**自边（圈/loop）**（如图7）。图可能包含一些自边或者重边，像图5这种没有自边和重边的图叫做**简单图（simple graph）**。

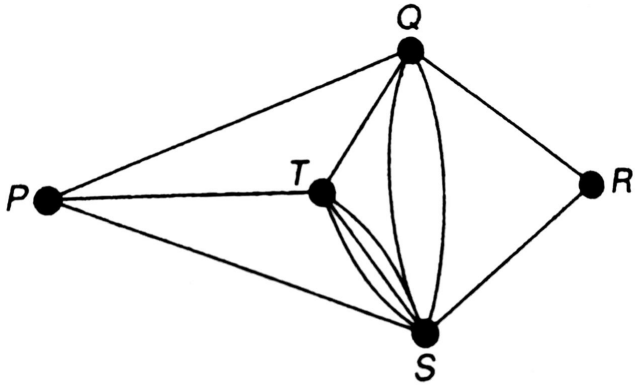


图6

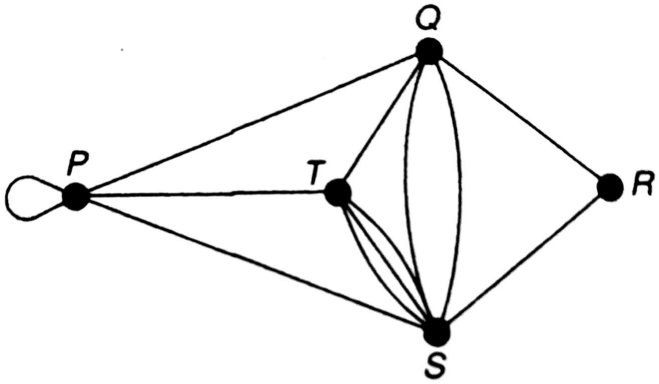


图7

关于**有向图（directed graph/digraph）**的研究来源于单行道路的研究。一个有向图的例子在图8中给出。在这里，单行线的方向用箭头来表示，这样有方向的边被成为**有向边（弧/arc）**。

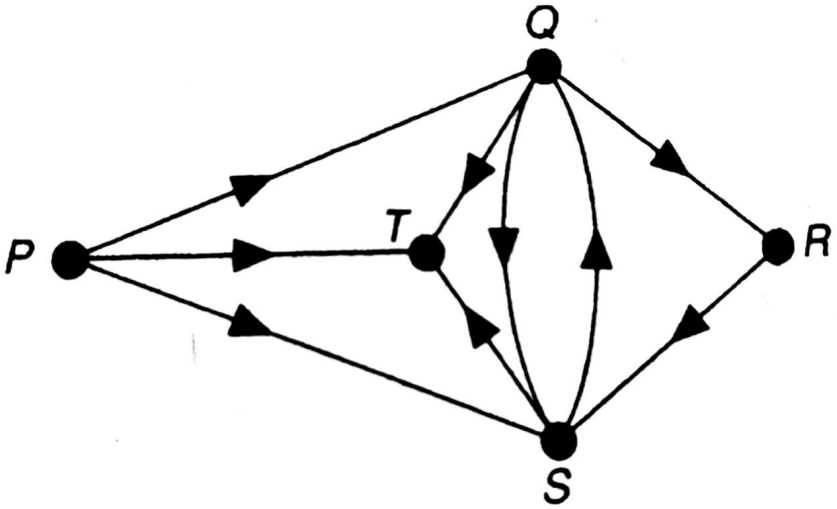
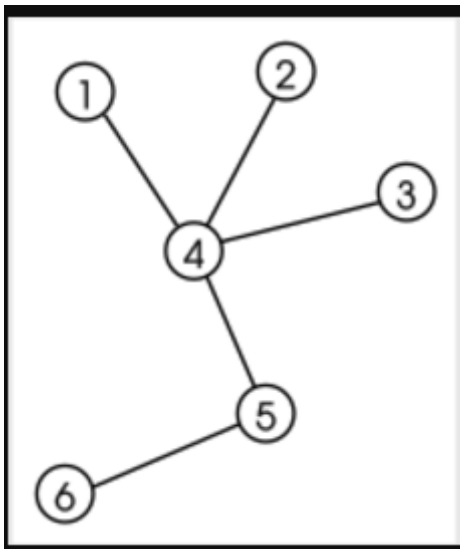


图8

参考链接（图 (数学) - 维基百科，自由的百科全书）

树的定义



包括6个顶点，5条边的树

在图论中，**树**（英语：tree）是一种无向图（英语：undirected graph），其中任意两个顶点间存在唯一一条路径。或者说，只要没有环的连通图就是树。

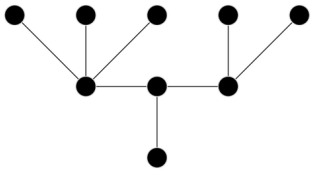
一棵树中任两个顶点之间都有且只有一条路径（指没有重复边的路径）。一棵有 n 个顶点的树有 $n-1$ 条边，其为具有 n 个点连通图的所需的最少边数。所以如果去掉树中的一条边，树就会不连通。

如果在一棵树中加入任意的一条边，就会得到有且只有一个环的图。这是因为这条边连接的两个点（或是一个点）中有且只有一条路径，这条路径和新加的边连在一起就是一个环。

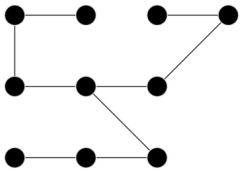
如果要在树中加入一个点，就要加入一条这个点和原有的点相连的边。这条边不会给这棵树增加一个环或者多余的路径。所以每次这样加入一个点，就可以构成一棵树。

显然树中也没有自环和重复边。

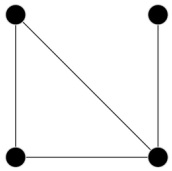
示例：



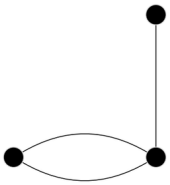
A tree



A tree



Not a tree



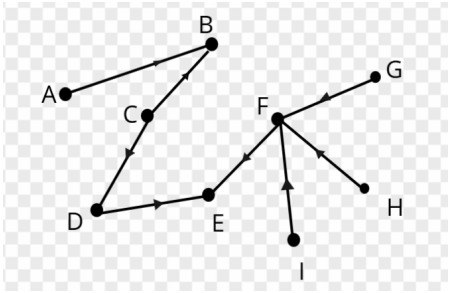
Not a tree

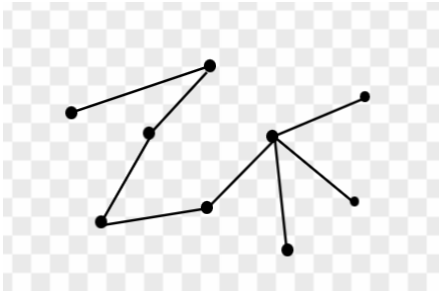
Not a tree

有根树

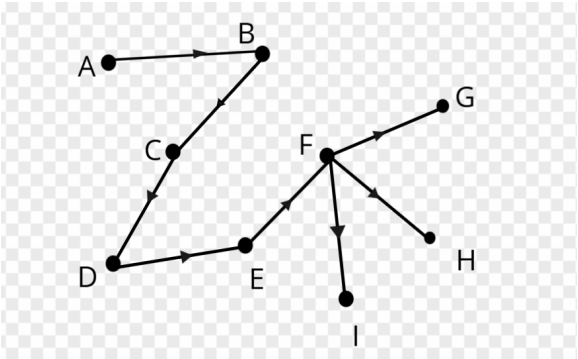
一个有向图，如果略去所有边的方向之后得到的图是一棵树，那么这个有向图是一个有向树。

例如：





当有向树中的一个节点入度为0，其余节点入度均为1的时候，这棵有向树是一棵有根树。



有根树中的节点可以根据到根的距离分**层**。一颗有根树的层数叫做这棵树的**高度**。节点最多的那一层的节点数叫做这棵树的**宽度**。对于有根树，每条边都有一个特殊的方向：指向根节点的方向，或者说上一层的方向（或者相反的，指向叶节点的方向，下一层的方向）。

可以看到上面这棵有根树的根节点是A，第一层是B，第二层是C，第三层是D，第四层是E，第五层是F，第六层是G、H、I。

一条边的两个端点中，靠近根的那个节点叫做另一个节点的**父节点**（也叫**父亲、双亲、双亲节点**），相反的，距离根比较远的那个节点叫做另一个节点的**子节点**（也可以叫**孩子，儿子，子女**等）。父亲方向的所有节点都叫做这个节点的**祖先**，儿子方向的所有节点都叫做这个节点的**子孙**。没有子节点的子节点叫做**叶节点**（或者**叶子节点**）。

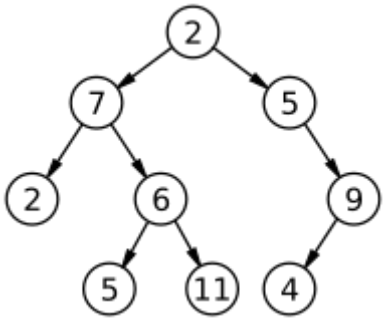
A的儿子是B；B的儿子是C，父亲是A；C的儿子是D，父亲是B；.....F的儿子是G、H、I，父亲是E；A是所有除B以外节点的祖先，G、H、I是叶子节点

由于到根的路径只有一条，根节点以外的节点的父节点永远只有一个，祖先就是这个点到根的路径上的所有节点（包括根，不包括这个节点本身）。另外，以一个节点为根的树是指包括这个节点和其所有子孙，并以这个节点为根的树。由于一般不需要这以外的子树，每一个节点也可以对应到一个以其为根的树，一个节点的子树通常也是指以这个节点的子节点为根的树。

参考链接（树 (图论) - 维基百科，自由的百科全书）

二叉树

一个连通的无环图，并且每一个顶点的度不大于3。有根二叉树还要满足根节点的度不大于2。有了根节点之后，每个顶点定义了唯一的父节点，和最多2个子节点。



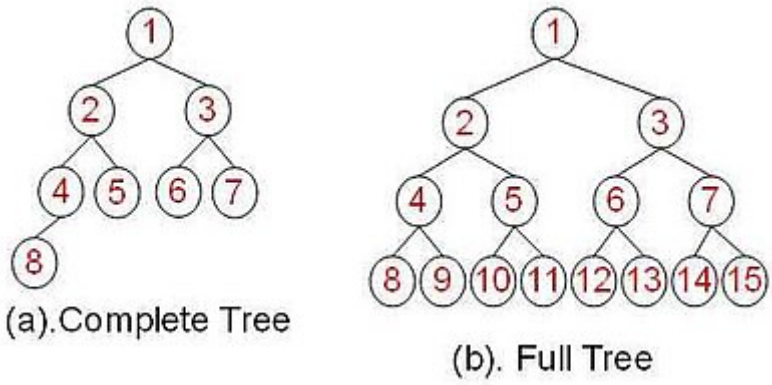
一棵有9个节点且深度为3的二叉树，其根节点的值为2

满二叉树(complete tree)

在一颗二叉树中，除了叶节点外，每个节点都有 2 个子节点。

完美二叉树

一棵深度为k，且有2^k-1个节点的二叉树，称为完美二叉树（full Binary Tree）。这种树的特点是其中所有内部节点都有两个子节点，并且所有叶子都处于同一级别。



参考链接（二叉树 - 维基百科，自由的百科全书）

重点：可以利用有根树给叶子编码，（注意，并不局限于二叉树，n叉树也可以，只是编码的集合有n个元素，二叉树只需要0、1即可），例如上图a，给叶子8编码，不妨让往左走是0，右走是1，则8叶子编码为0（从根节点往左走到2），0（从2节点左走到4），0（从节点4左走到8），因此编码为000，相应地，节点5编码为01，节点6编码为10，节点7为11.

数据压缩

背景介绍(可以不看，直接看例子)

压缩编码是一种用于有效地表示信息的方法，它通过对数据进行压缩，可以减少存储空间和传输开销。在信息论、计算机科学和电信领域，压缩编码具有广泛的应用。本文将从理论知识、核心概念、算法原理、实例代码以及未来发展等多个方面进行全面的介绍。

信息论基础

信息论是研究信息的数学性质和信息处理系统性能的科学。信息论的基本概念之一是“熵”(Entropy)，用于衡量信息的不确定性。熵是一个度量信息量的标准，它反映了信息中的随机性。

1. 熵的定义

   $H(X) = -\sum_{x \in X} P(x) \log P(x)$

   其中，X 是一个有限的随机变量，$P(x)$ 是取值 x 的概率。

2. 信息量
   信息量(Information)是一种度量信息的量度，它反映了信息的有用性。信息量的定义如下：

   $I(X;Y) = H(X) - H(X|Y)$ 其中，$I(X;Y)$ 是随机变量 X 和 Y 之间的条件熵，$H(X|Y)$ 是 X 给定 Y 的熵。

压缩编码的需求

在信息传输过程中，信息通常会经历编码、传输和解码的过程。为了减少信息传输的开销，我们需要对信息进行压缩，以减少存储空间和传输开销。压缩编码的目标是使得编码后的信息尽可能小，同时保证信息的完整性和可靠性。

压缩编码的性能指标

压缩编码的性能主要由以下几个指标来衡量：

压缩率(Compression Ratio)：压缩率是指原始信息的大小与编码后信息的大小之间的比值。压缩率越小，表示编码效果越好。
解码时间复杂度(Decoding Time Complexity)：解码时间复杂度是指解码过程中所需的计算资源。解码时间复杂度越低，表示编码效果越好。
编码时间复杂度(Encoding Time Complexity)：编码时间复杂度是指编码过程中所需的计算资源。编码时间复杂度越低，表示编码效果越好。
编码后信息的可读性(Readability of Encoded Information)：编码后信息的可读性是指编码后的信息是否容易被人阅读和理解。可读性越高，表示编码效果越好。

压缩编码的应用场景

压缩编码在各种应用场景中都有广泛的应用，如：

数据存储：压缩编码可以减少数据存储空间，提高存储设备的利用率。
数据传输：压缩编码可以减少数据传输开销，提高数据传输速度。
信号处理：压缩编码可以用于处理信号，如音频和视频压缩。
文本处理：压缩编码可以用于处理文本，如文本压缩和文本检索。

压缩例子

Letter	Relative frequency in the English language
e	11.162%
t	9.356%
a	8.497%
r	7.587%
i	7.546%
o	7.507%
n	6.749%
s	6.327%
h	6.094%
d	4.253%
l	4.025%
u	2.758%
w	2.560%
m	2.406%
f	2.228%
c	2.202%
g	2.015%
y	1.994%
p	1.929%
b	1.492%
k	1.292%
v	0.978%
j	0.153%
x	0.150%
q	0.095%
z	0.077%

给定一个文件内的字符统计情况如上图所示，如果我们想要压缩这个文件（注意文件的本质其实仍然是0和1），即使用更少的0和1来表示这个文件，那么由公式可知，我们应该让总编码长度越小越好。而总编码长度有：

$$L = \sum_{x \in X} P(x)\ell(x)$$

其中，$\ell(x)$ 是符号 x 的编码长度。

所以我们很自然地应该让$P(x)$越大的符号编码越短，因此我们不妨这样编码：

LETTER	CODE
e	0
t	1
a	00
r	01
i	10
o	11
n	000

LETTER	CODE
s	001
h	010
d	011
l	100
u	101
w	110
m	111
f	0000
c	0001
g	0010
y	0011
p	0100
b	0101
k	0110
v	0111
j	1000
x	1001
q	1010
z	1011

这样的编码看起来似乎完全没有问题，并且完美地符合了我们对压缩的要求，但是真的没有问题吗？我们不妨试一下。

假设现在有一句话："i got it"，将其编码后会变成什么呢？以01的形式转换出来：10 0010 11 1 10 1（这是为了方便对照所以加了空格），我们现在来解压缩：1000 1011 110 1，我们的解压缩结果是：jzwt。为什么解压缩出来是错的呢？或者说，我们似乎有非唯一的方式来进行解压，每一种解压方式都会得到不同的结果，这是为什么？

相信聪明的你一定已经看出来了！这是因为i的编码10，是j编码1011的前缀！所以我们必须要规避这种情况！

前缀码

前置码（英语：Prefix code），又译**前缀码**、**前缀编码**，是一种编码系统。这种编码系统通常是可变长度编码，在其中的每个码字，都具备“前置性质”（prefix property），也就是说，在编码中的每个码字，都不能被其他码字当成前置部位。举例而言，编码字 `{9, 55}` 具备了前置性质，但编码字 `{9, 5, 59, 55}` 就不具备，因为其中的“5”是“59”及“55”的前置字。这也被称为**无首码的代码**（prefix-free codes，PFC，**无前缀码**）。虽然哈夫曼编码只是派生的前缀码中众多算法之一，但前缀码也被称为广义上的“哈夫曼编码”。对于任何唯一可解编码，都有一个具有相同码字长度的前缀码。克拉夫特不等式表征了在唯一可解编码中可能出现的码字长度集。

参考链接（前置码 - 维基百科，自由的百科全书）

Huffman 编码（哈夫曼）

终于来到前置知识的正题啦！走到这一步，恭喜你已经完成了80%的前置知识学习进度！

哈夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的路径长度（若根结点为0层，叶结点到根结点的路径长度为叶结点的层数）。树的路径长度是从树根到每一结点的路径长度之和，记为$WPL=（W_1*L_1+W_2*L_2+W_3*L_3+...+W_n*L_n）$，$N$个权值$W_i（i=1,2,...n）$构成一棵有$N$个叶结点的二叉树，相应的叶结点的路径长度为$L_i（i=1,2,...n）$（这里的路径长度也就是该叶子的编码长度，由前面我们对二叉树叶子进行编码的实例可以看出）。可以证明霍夫曼树的WPL是最小的（不妨先武断地下这个结论，后续的学习会证明），即总编码长度最短！（最优性证明参考链接信息与编码系列（二）最优码——Huffman码 - 知乎）

$huffman$编码是通过构建一棵二叉树来进行编码的，具体的$huffman$算法流程如下所示：

算法问题定义：

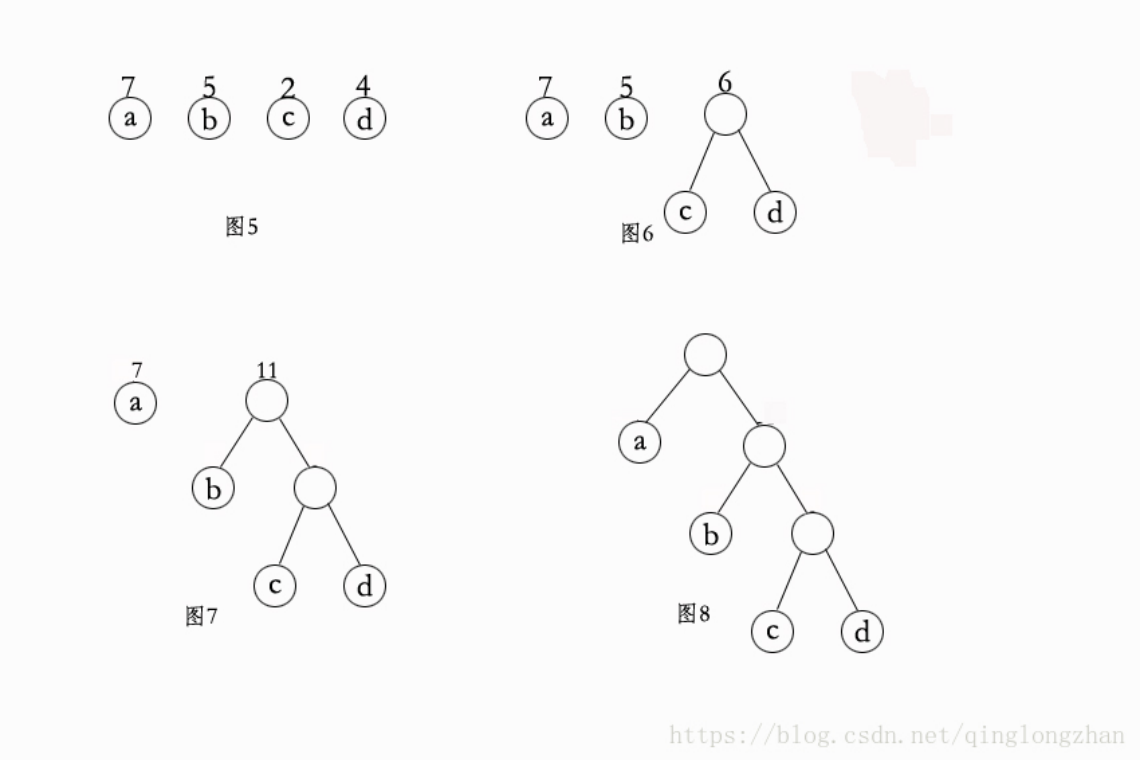
- 输入　符号集合$S=s_1,s_2,\cdots,s_n$，其S集合的大小为n。

  权重集合$W=w_1,w_2,\cdots,w_n$，其W集合不为负数且$w_i=weight(s_i),1\leq i\leq n$
- 输出　一组编码$C(S,W)=c_1,c_2,\cdots,c_n$，其C集合是一组二进制编码且$c_i$为$s_i$相对应的编码，$1\leq i\leq n$。
- 目标　设$L(C)=\sum_{i=1}^{n}w_i\times length(c_i)$为C的加权的路径长，对所有编码$T(S,W)$，则$L(C)\leq L(T)$

算法过程：

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. 在森林中选出两个**根结点**的权值最小的树合并，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(2)、(3)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。
5. 可以根据左0右1，或者左1右0的方式给出字母的$huffman$编码

示例：

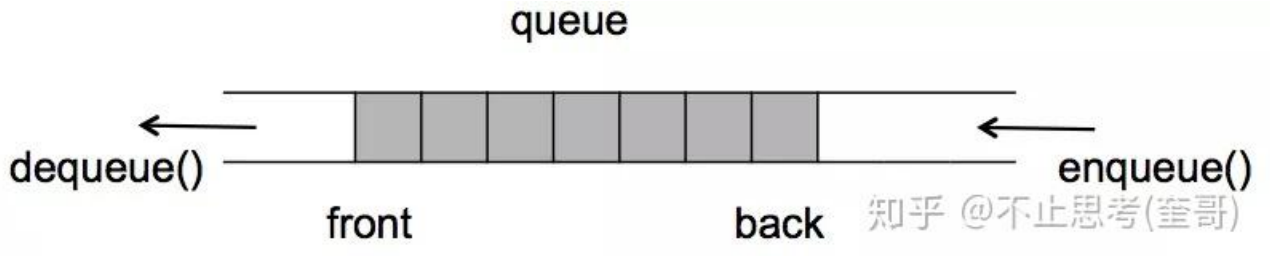


最终的编码是a: 0, b: 10, c: 110, d: 111.

队列

队列是一种常见的数据结构，是人为设计的，因为设计的时候给其加了一些数据使用的限制，这种结构也就具备了一些特征。

队列中的数据呈线性排列。队列中添加和删除数据的操作分别是在两端进行的。

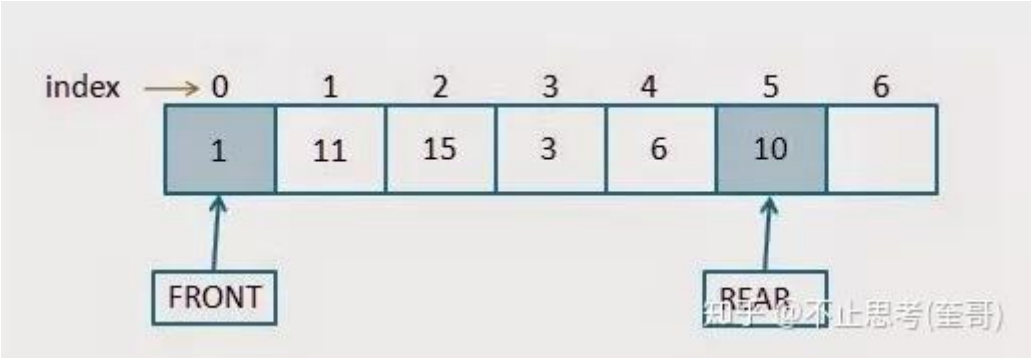


队列这种数据结构非常容易理解，就像我们平时去超市买东西，在收银台结账的时候需要排队，先去排队的就先结账出去，排在后面的就后结账，有其他人再要过来结账，必须排在队尾不能在队中间插队。

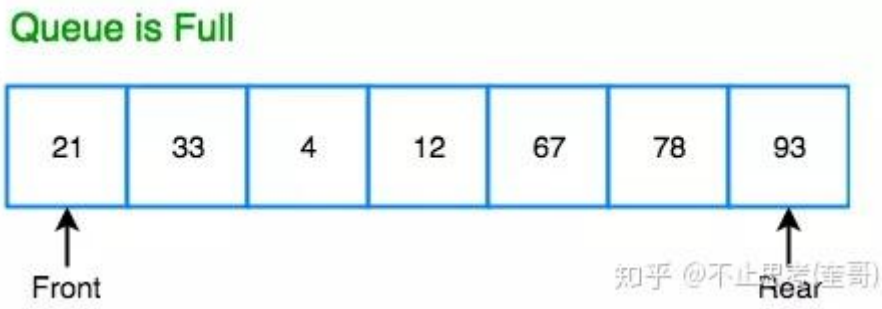
我们在实现$Huffman$压缩的过程中可能会使用队列。

用数组实现的队列，叫做 **顺序队列**：

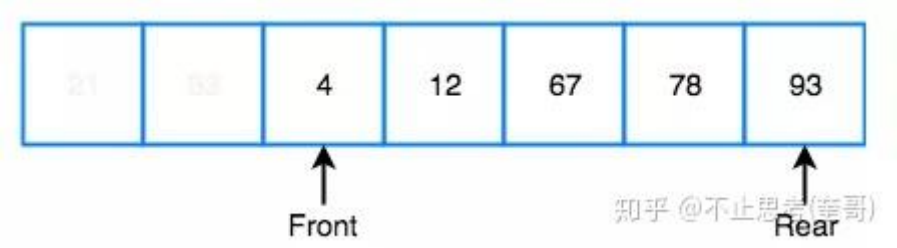
用数组实现的思路是这样的：初始化一个长度为n的数组，创建2个变量指针front和rear，front用来标识队头的下标，而rear用来标识队尾的下标。因为队列总是从对头取元素，从队尾插入数据。因此我们在操作这个队列的时候通过移动front和rear这两个指针的指向即可。初始化的时候front和rear都指向第0个位置。
当有元素需要入队的时候，首先判断一下队列是否已经满了，通过rear与n的大小比较可以进行判断，如果相等则说明队列已满（队尾没有空间了），不能再插入了。如果不相等则允许插入，将新元素赋值到数组中rear指向的位置，然后rear指针递增加一（即向后移动了一位），不停的往队列中插入元素，rear不停的移动，如图：



当队列装满的时候，则是如下情况：



当需要做出队操作时，首先要判断队列是否为空，如果front指针和rear指针指向同一个位置（即front==rear）则说明队列是空的，无法做出队操作。如果队列不为空，则可以进行出队操作，将front指针所指向的元素出队，然后front指针递增加一（即向后移动了一位），加入上图的队列出队了2个元素：



所以对于数组实现的队列而言，需要用2个指针来控制（front和rear），并且无论是做入队操作还是出队操作，front或rear都是往后移动，并不会往前移动。入队的时候是rear往后移动，出队的时候是front往后移动。出队和入队的时间复杂度都是O(1)的。

参考链接（算法一看就懂之「 队列 」 - 知乎）

过渡篇

上面的大家都可以不必看，只要能看懂最后的huffman例题就好，如果看不懂例题导致后面代码越写越乱没有思路可是很浪费时间的一件事情！

Part2 实验设计

这个部分会对实验思路进行讲解，是完成实验的重中之重。

文件是个啥样？对于文本文件，例如txt、c、cpp我们可以直接对文件内的字符进行统计，然后对其进行$huffman$编码，再进行压缩。但是对于非文本文件呢？例如图片，音频。聪明的你一定想到了，我们可以使用固定位数的二进制数来表示文件（比如8bit），即便它们并不是文本文件。我们可以每次只读取8bit的数据，然后把读出来的数据当做一个字符，以此统计该字符在文件中出现的频率。

我们应该如何只读取8bit的数据呢？请自己查找方法并完成代码框架中的TODO.（也可以不读8bit，可以自己设计，如果不使用8bit，请修改$init.h$文件中的Nchar宏定义）

上面的方法可以让我们得到频率，字符种类一共是256种，因此一个定长数组可以存下这些频数。所以接下来我们需要搞清楚如何构建一棵$huffman$树，并得到$huffman$编码。

资料压缩

实现霍夫曼编码的方式主要是创建一个二叉树和其节点。这些树的节点可以存储在数组里，数组的大小为符号（symbols）数的大小**n**，而节点分别是终端节点（叶节点）与非终端节点（内部节点）。

一开始，所有的节点都是终端节点，节点内有三个字段：

1. 符号（Symbol）
2. 权重（Weight、Probabilities、Frequency）
3. 指向父节点的链接（Link to its parent node）

而非终端节点内有四个字段：

1. 权重（Weight、Probabilities、Frequency）
2. 指向两个子节点的 链接（Links to two child node）
3. 指向父节点的链接（Link to its parent node）

我们统一一下节点所需要的字段：权重（被使用过后记为INT_MAX以确保后续不会选择该节点），字符（如果非叶子则为0），指向两个孩子的节点的指针（叶子节点值为NULL），指向父亲节点的指针（根节点为NULL），标志本身在数组中的位置（idx)。一共6个字段。

一般地，我们用'0'与'1'分别代表指向左子节点与右子节点，最后为完成的二叉树共有**n**个终端节点与**n-1**个非终端节点（思考一下为什么，可能会作为检查时的问题），去除了不必要的符号并产生最佳的编码长度。

实现霍夫曼树的方式有很多种，我们这里只需要使用两个队列来实现$huffman$树的构建，具体请看代码框架，助教已经实现了队列的操作：初始化，出队，入队。大家只需要调用即可。

过程中，每个终端节点都包含着一个权重（Weight、Probabilities、Frequency），两两终端节点结合会产生一个新节点，**新节点的权重是由两个权重最小的终端节点权重之总和**，并持续进行此过程直到只剩下一个节点为止。

然后就是利用编码压缩，大家可以按照框架里函数的处理方式：用一个byte存储长度，32个byte存储编码，也可以修改为自己的存储方式，例如使用一个字符串来直接进行存储。（请思考为什么要用32byte存储编码）

资料解压缩

简单来说，霍夫曼码树的解压缩就是将得到的前置码（Prefix Huffman code）变换回符号，通常借由树的追踪（Traversal），将接收到的位串（Bits stream）一步一步还原。但是要追踪树之前，必须要先重建霍夫曼树；某些情况下，如果每个符号的权重可以被事先预测，那么霍夫曼树就可以预先重建，并且存储并重复使用，否则，发送端必须预先发送霍夫曼树的相关信息给接收端。

我们这里不需要实现跨计算机的压缩和解压缩软件，因此你可以把自己的编码信息写入文件保存，等到需要解压缩的时候，可以通过直接读取文件获得编码信息，对压缩文件进行解压缩。

（在本实验中我们采用了直接在一次主函数中进行压缩和解压缩的方式，因此直接把$huffman$树传入解压缩函数进行文件解压缩即可。）

实验思路

我们建议先对一个txt文件进行测试，例如abbcccddddeeeee这样简单的文本，你可以通过调试来观察值的变化，掌控编码的构建和压缩的正确性。

最终实现的效果：我们期望对一张图片进行压缩解压缩。

具体实现思路详见代码框架！

Part3 项目架构说明

```
D:.
|  --counter.h //对文件中的字符计数
|  --decode.h  //解压缩
|  --encode.h //huffman编码
|  --init.h //初始定义结构
|  --main.c //主函数所在文件
|  --queue.h //队列的实现
|  --readme.md //实验文档
|
└─File //测试文件夹
```

Part4 评分细则

各位同学不必遵循助教给出的代码框架，如果觉得助教写的框架太烂可以自己独立写一份代码，但是相应地，为了避免大家去找资源，进行CV，助教会对独立写的同学进行着重抽查。如果是自己写的，相信很容易就能通过助教的检验，如果是copy且回答不上来问题的，那么将会受到扣分惩罚！

项目总分30分，做出选做的同学额外得到5分实验分

1. $main.c$中的$TODO$：3分
2. $counter.h$中的$TODO$：3分
3. $encode.h$中的$TODO$：18分，每4.5分
4. $decode.h$中的$TODO$：6分
5. 选做：实现一个能够跨计算机的压缩解压缩软件：5分。即把huffman树存储到压缩后的文件中，在另一台电脑上运行你的decode仍然能够正常解压缩。

（注意：大家可以在$vscode$中下载$TODOHighlight$插件高亮$TODO$）

最后再次声明，大家可以随意更改助教给出的框架，不必遵循任意的约束，希望大家享受这次大作业的编程！