POJ3253

题目描述（POJ3253）：约翰想修牧场周围的篱笆，需要 N 块（1≤N≤20000）木板，每块木板都具有整数长度 L_i（1≤L_i≤50000）米。他购买了一块足够长的木板（长度为 L_i 的总和，i=1,2,..., N），以便得到 N 块木板。切割时木屑损失的长度不计。

农夫唐向约翰收取切割费用。切割一块木板的费用与其长度相同。切割长度为 21 米的木板需要 21 美分。唐让约翰决定切割木板的顺序和位置。约翰知道以不同的顺序切割木板，将会产生不同的费用。帮助约翰确定他得到 N 块木板的最低金额。

输入：第 1 行包含一个整数 N，表示木板的数量。第 2～N+1 行，每行都包含一个所需木板的长度 L_i。

输出：一个整数，即进行 N-1 次切割的最低花费。

输入样例	输出样例
3 8 5 8	34

POJ1521

题目描述（POJ1521）：熵编码是一种数据编码方法，通过对去除“冗余”或“额外”信息的消息进行编码来实现无损数据压缩。为了能够恢复信息，编码字形的位模式不允许作为任何其他编码位模式的前缀，称之为“无前缀可变长度”编码。只允许逐位读取编码的比特流，并且每当遇到表示字形的一组比特时，都可以解码该字形。如果不强制使用无前缀约束，则不可能进行这种解码。

第 1 个例子，考虑文本“AAAAABCD”，对其使用 8 位 ASCII 编码需要 64 位。如果用“00”对“A”编码，用“01”对“B”编码，用“10”对“C”编码，用“11”对“D”编码，那么只需 16 位编码，得到的位模式将是“0000000000011011”。不过，这仍然是固定长度的编码；使用的是每个字形两位，而不是八位。既然字形“A”出现的频率更高，那么能用更少的位来编码它吗？实际上可以，但为了保持无前缀编码，其他一些位模式将变得比两位长。最佳编码是将“A”编码为“0”，将“B”编码为“10”，将“C”编码为“110”，将“D”编码为“111”（这显然不是唯一的最佳编码，因为 B、C 和 D 的编码可以在不增加最终编码消息大小的情况下自由地交换给任何给定的编码）。使用此编码，消息仅以 13 位编码到“0000010110111”，压缩比为 4.9:1（即最终编码消息中的每一位表示的信息与原始编码中的 4.9 位表示的信息相同）。从左到右阅读这个位模式，将看到无前缀编码使得将其解码为原始文本变得简单，即使代码的位长度不同。

第 2 个例子，考虑文本“THE CAT IN THE HAT”。字母“T”和空格字符都以最高频率出现，因此它们在最佳编码中显然具有最短的编码位模式。字母“C”、“I”和“N”只出现一次，因此它们的代码最长。有许多可能的无前缀可变长度位模式集可以产生最佳编码，也就是说，允许文本以最少的位进行编码。其中一种最佳编码是：空格:00、A:100、C:1110、E:1111、H:110、I:1010、N:1011、T:01。因此，这种最佳编码只需 51 位，与用 8 位 ASCII 编码对消息进行编码所需的 144 位相比，压缩比为 2.8:1。

输入：输入文件包含一个字符串列表，每行一个。字符串将只包含大写字母、数字字符和下画线（用于代替空格）。以字符串“END” 结尾，不应处理此行。

输出：对于每个字符串，都输出 8 位 ASCII 编码的位长度、最佳无前缀可变长度编码的位长度及精确到一个小数点的压缩比。

输入样例	输出样例
AAAAABCD THE_CAT_IN_THE_HAT END	64 13 4.9 144 51 2.8

UVA12676

题目描述（UVA12676）：静态哈夫曼编码是一种主要用于文本压缩的编码算法。给定一个由 N 个不同字符组成的特定长度的文本，算法选择 N 个编码，每个不同的字符都对应一个编码。使用这些编码压缩文本，当选择编码算法构建一个具有 N 个叶子的二叉树时，对于 N≥2，树的构建流程如下。

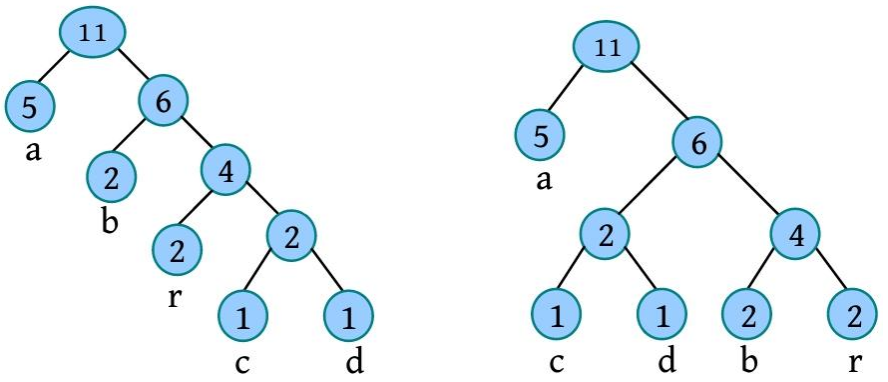
（1）对文本中的每个不同字符，都构建一个仅包含单个节点的树，其权值为该字符在文本中的出现次数。

（2）构建一个包含上述 N 棵树的集合 S。

（3）当 S 包含多于一棵树时：①选择最小的权值 $t_1 \in S$，并将其从 S 中删除；②选择最小的权值 $t_2 \in S$，并将其从 S 中删除；③构建一棵新树 t，t_1 为其左子树，t_2 为其右子树，t 的权值为 t_1、t_2 权值之和；④将 t 加入 S 集合。

（4）返回保留在 S 中的唯一一棵树。

对于文本“abracadabra”，由上述过程生成的树，可以像下面左图，其中每个叶子节点内都是该字符在文本中出现的次数（权值）。请注意获得的树不是唯一的，也可以像下面右图或其他，因为可能包含几个权值最小的树。



对文本中的每个不同字符，其编码都取决于最终树中从根到对应字符的叶子之间的路径，编码的长度是这条路径中的边数。假设该算法构建的是左侧的树，“r”的代码长度为 3，“d”的代码长度为 4。根据算法选择的 N 个代码的长度，找所有字符总数的最小值。

输入：输入包含多个测试用例，每个测试用例的第 1 行都包含一个整数 N（2≤N≤50），表示在文本中出现的不同字符数。第 2 行包含 N 个整数 L_i（$1 \le L_i \le 50$，i=1,2,…,N），表示由哈夫曼算法生成的不同字符的编码长度。假设至少存在一棵由上述算法构建的树，那么可以生成具有给定长度的编码。

输出：对每个测试用例都输出一行，表示所有字符总数的最小值。

输入样例	输出样例
2 1 1 4 2 2 2 2 10 8 2 4 7 5 1 6 9 3 9	2 4 89

UVA240

题目描述（UVA240）：哈夫曼编码是一种最优编码方法。根据已知源字母表中字符的出现频率，将源字母表中的字符编码为目标字母表中的字符，最优的意思是编码信息的平均长度最小。在该问题中，需要将 N 个大写字母（源字母 S_1...S_N、频率 f_1...f_N）转换成 R 进制数字（目标字母 T_1...T_R）。

当 R=2 时，编码过程分几个步骤。在每个步骤中都有两个最低频率的源字母 S_1 和 S_2，合并成一个新的“组合字母”，频率为 S_1 和 S_2 的频率之和。如果最低频率和次低频率相等，则字母表中最早出现的字母被选中。经过一系列步骤后，最后只剩两个字母合并，将每次合并的字母都分配一个目标字符，将较低频率的分配 0，将另一个分配 1。如果在一次合并中，每个字母都有相同的频率，则将最早出现的分配 0。出于比较的目的，组合字母的值为合并中最早出现的字母的值。源字母的最终编码由每次形成的目标字符组成。

目标字符以相反的顺序连接，最终编码序列中的第 1 个字符为分配给组合字母的最后一个目标字符。下面的两个图展示了 R=2 的过程。

Symbol	Frequency
A	5
B	7
C	8
D	15

Pass 1: A and B grouped
Pass 2: {A,B} and C grouped
Pass 3: {A,B,C} and D grouped
Resulting codes: A=110, B=111, C=10, D=0
Avg. length=(3*5+3*7+2*8+1*15)/35=1.91

Symbol	Frequency
A	7
B	7
C	7
D	7

Pass 1: A and B grouped
Pass 2: C and D grouped
Pass 3: {A,B} and {C,D} grouped
Resulting codes: A=00, B=01, C=10, D=11
Avg. length=(2*7+2*7+2*7+2*7)/28=2.00

当 R>2 时，对每一个步骤都分配 R 个字母。由于每个步骤都将 R 个字母或组合字母合并为一个组合字母，并且最后一次合并必须合并 R 个字母或组合字母，源字母必须包含 k×(R-1)+R 个字母，k 为整数。由于 N 可能不是很大，因此必须包括适当数量具有 0 频率的虚拟字母。这些虚拟字母不包含在输出中。进行比较时，虚拟字母晚于字母表中的任何字母。

哈夫曼编码的基本过程与 R=2 的过程相同。在每次合并中都将具有最低频率的 R 个字母合并，形成新的组合字母，其频率等于在组合字母中包括的字母频率的总和。被合并的字母被分配目标字母符号 0～R-1。0 被分配给具有最低频率的组合中的字母，1 被分配给下一个最低频率，等等。如果字母组合中的几个字母具有相同的频率，则字母表中最早出现的字母被分配较小的目标符号，以此类推。

下图说明了 R=3 的过程。

Symbol	Frequency
A	5
B	7
C	8
D	15

Pass 1: ? (fictitious symbol), A and B are grouped
Pass 2: {?,A,B}, C and D are grouped
Resulting codes: A=11, B=12, C=0, D=2
Avg. length=(2*5+2*7+1*8+1*15)/35=1.34

输入：输入将包含一个或多个数据集，每行一个。每个数据集都包含整数值 R（2≤R≤10）、整数值 N（2≤N≤26）和整数频率 f1...fN，每个都为 1～999。整个输入数据都以 R 为 0 结束，它不被认为是单独的数据集。

输出：对每个数据集都在单行上显示其编号（编号从 1 开始按顺序排列）和平均目标符号长度（四舍五入到小数点后两位）。然后显示 N 个源字母和相应的哈夫曼代码，每行都有一个字母和代码。在每个测试用例后都打印一个空行。

输入样例

```
2 5 5 10 20 25 40
2 5 4 2 2 1 1
3 7 20 5 8 5 12 6 9
4 6 10 23 18 25 9 12
0
```

输出样例

```
Set 1; average length 2.10
A: 1100
B: 1101
C: 111
D: 10
E: 0

Set 2; average length 2.20
A: 11
B: 00
C: 01
D: 100
E: 101

Set 3; average length 1.69
A: 1
B: 00
C: 20
D: 01
E: 22
F: 02
G: 21

Set 4; average length 1.32
A: 32
B: 1
C: 0
D: 2
E: 31
F: 33
```