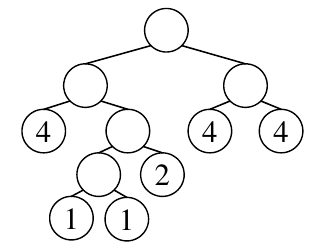
**题意**

二叉树上每个叶子结点都赋有一个整数作为权值。如果每个非叶子结点的左右子树的权值之和相同，那么这棵树就认为是平衡的，下图就是一个例子：



如果一个平衡树从左到右把所有叶子结点的权值全部列出来形成的序列是序列A的子序列(不一定连续)，那么我们就说这棵树隐藏在A中。上图的树就隐藏在序列3 4 1 3 1 2 4 4 6中，因为4 1 1 2 4 4是其子序列。

给出一个长度为N(1≤N≤1000)的整数序列*A1A2* . . . *AN*(1≤*Ai*≤500)，找出隐藏在其中的叶子个数最多的平衡树并输出其叶子个数。上图中的树就是上述序列中隐藏的叶子个数最多的平衡树。

**题解**

我们首先探究答案可能有的性质。可以发现答案树的权值存在一个或多个最小值，记为base，且答案树的其他权值都是base的2k倍。提供一个直观理解如下：

1. 假设base所在子树有k层，那么易得权和 = base \* 2k-1。这对于答案树也成立，所以答案树里的点权都是base的倍数。
2. 假设有些点权不是base的2k倍，且它们已经合成了另一棵子树，其最小权值为base1。由上面计算子树权和的公式，和base1 / base不是2的幂，可知这棵子树一定不能和base所在的子树合并。

于是对于每一个可以成为base的A[i]（即没有A[j]使A[i] / A[j] = 2k，k>0的A[i]），我们在A里依次取出其他权值，把它们和base的比值存到另一个容器B里。接下来只需要对B考虑合并问题，并更新最终答案。

我们定义dp[i,j]为只用B中前i个元素，且强制使用B[i]，所得树权值和为j的情况下的最大叶子个数。为叙述方便，定义B的前缀和数组为sum。由上面计算子树权和的公式可知，对B而言答案树权值和为2的幂，故答案是：

max(dp[B.size(),2x])，2x <= sum[B.size()]，x >= 0

我们可以采用打表和刷表两种方法。状态转移方程：

dp[i,j] = max(dp[i,j],dp[i – 1,j – B[i]] + 1)，j – B[i]是B[i]的倍数，x >= 0

记得还有一种方案是只选B[i]：

dp[i,B[i]] = max(dp[i,B[i]],1)

j的枚举次数是复杂度的瓶颈，它的一个上界是sum[i]，且只需枚举B[i]的倍数，因此我们实现时采用的上界是sum[i] / B[i] \* B[i]。

由于只需要用到i - 1的状态，我们可以借鉴01背包进行滚动优化。我们也发现这个算法和01背包的算法有比较多的相似性。

因为权值小于等于500，所以在B中每个元素最大为256。共有1000个元素，考虑极端情况是512个256合并，所以有效状态数不超过256 \* 512 = 217，其实远远小于此上界。但dp数组需要的空间大一些，256 \* 1000。

**总结**

上面对答案特性的探讨，使得以上dp过程、从dp寻找答案的过程由不可行变为可行。以上dp和01背包dp有一定相似性，比如，权值和和叶子数分别对应重量和价值。

**时空复杂度**

时间复杂度：O(N \* 217)

空间复杂度：O(217)

**代码**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <string>

#include <ctime>

#include <vector>

#include <cmath>

#include <queue>

#include <map>

#include <set>

#include <stack>

#include <algorithm>

*//#include <bits/stdc++.h>*

**using** **namespace** std;

**typedef** **long** **long** LL;

**const** **int** SZ = 1005,SZ0 = 256005;

**int** n,a[SZ],dp[SZ0];

**bool** bas[SZ];**bool** vis[SZ];

**int** sl,B[SZ],sum[SZ];

**template**<**typename** Type>**inline** **void** read(Type &xx){

Type f = 1;**char** ch;xx = 0;

**for**(ch = getchar();ch < '0' || ch > '9';ch = getchar()) **if**(ch == '-') f = -1;

**for**(;ch >= '0' && ch <= '9';ch = getchar()) xx = xx \* 10 + ch - '0';

xx \*= f;

}

**int** DP(){

**for**(**int** i = 1;i <= sl;i++) sum[i] = sum[i - 1] + B[i];

dp[0] = 0;

**for**(**int** i = 1;i <= sl;i++){

**int** mn = B[i];

**for**(**int** j = sum[i] / mn \* mn;j >= (mn << 1);j -= mn)

dp[j] = max(dp[j],dp[j - mn] + 1);

dp[mn] = max(dp[mn],1);

}

**int** ans = 0;

**for**(**int** i = 1;i <= sum[sl];i <<= 1) ans = max(ans,dp[i]);

memset(dp,0x8f,**sizeof** dp);

**return** ans;

}

**bool** is\_least(**int** idx){

**if**(vis[a[idx]]) **return** **false**;

vis[a[idx]] = **true**;

**for**(**int** j = 1;j <= n;++j)

**if**(a[idx] % a[j] == 0 && bas[a[idx] / a[j]] && a[idx] > a[j])

**return** **false**;

**return** **true**;

}

**int** main(**int** argc, **char**\*\* argv) {

**for**(**int** i = 1;i < SZ;i <<= 1) bas[i] = **true**;

**while**(scanf("%d",&n) != EOF && n){

memset(dp,0x8f,**sizeof** dp);memset(vis,0,**sizeof** vis);

**for**(**int** i = 1;i <= n;i++) read(a[i]);

**int** ans = 0;

**for**(**int** i = 1;i <= n;++i){

**if**(!is\_least(i)) **continue**;

sl = 0;

**for**(**int** j = 1;j <= n;++j)

**if**(a[j] % a[i] == 0 && bas[a[j] / a[i]])

B[++sl] = a[j] / a[i];

ans = max(ans,DP());

}

printf("%d\n",ans);

}

**return** 0;

}