A. Yukina with n-Power

算法：递推+矩阵快速幂

最容易想到的方法肯定是直接计算这个表达式的值，但是这样的精度是不够的。朴素的算法没有办法得到答案。但是根据分析可以发现这个问题不用求出 $\sqrt{5}$ 的值也可以得到答案。

可以发现，将 $(3+\sqrt{5})^n$ 这个式子展开后就是 $A_n + B_n\sqrt{5}$ 的形式。同样的，将 $(3-\sqrt{5})^n$ 这个式子展开后就是 $A_n - B_n\sqrt{5}$ 。

因此，$(3+\sqrt{5})^n + (3-\sqrt{5})^n = 2A_n$ 是一个整数，其中 $0 < (3-\sqrt{5})^n < 1$ ，是解题的关键。由于 $(3+\sqrt{5})^n = 2A_n - (3-\sqrt{5})^n$ ，所以 $(3+\sqrt{5})^n$ 的整数部分就是 $2A_n - 1$。

根据上面的推导，只要高效的求出 An 就可以解决这个问题了。由于 $(3+\sqrt{5})^{n+1} = (3+\sqrt{5})(3+\sqrt{5})^n = (3+\sqrt{5})(A_n + B_n\sqrt{5})$ ，可以得到 $A_n, B_n, A(n+1), B(n+1)$ 的递推关系。

$A(n+1) = 3An + 5Bn$

$B(n+1) = An + 3Bn$

令 $n = 0$ 可知：$A_0 = 1, B_0 = 0$

这里可以用矩阵表示这个递推关系，因此可以使用快速幂运算。因为结果要求的是后三位，所以最后取余 1000 就行。

B. Yukina with Three Points

算法：STL中set关联式容器的灵活运用+暴力计算

set就是数学上的集合——每个元素最多只出现一次，并且set中的元素已经**从小到大排好序**。

常用的函数：

insert(key_value)---将key_value插入到set中，返回值是pair<set::iterator,bool>，bool标志着插入是否成功，而iterator代表插入的位置，若key_value已经在set中，则iterator表示的key_value在set中的位置

begin()---返回set容器的第一个元素的地址

end()---返回set容器的最后一个元素地址

clear()---删除set容器中的所有的元素

empty()---判断set容器是否为空

max_size()---返回set容器可能包含的元素最大个数

size()---返回当前set容器中的元素个数

erase(it)---删除迭代器指针it处元素

count()---用来查找set中某个元素出现的次数。这个函数在set并不是很实用，因为一个键值在set只可能出现0或1次，这样就变成了判断某一键值是否在set出现过了。

find()---用来查找set中某个元素出现的位置。如果找到，就返回这个元素的迭代器，如果这个元素不存在，则返回s.end() 。 (最后一个元素的下一个位置，s为set的变量名)

本题中就可以将n个点的坐标插入到set集合内，因为包含两个元素，因此可以定一个数据类型：set<pair<int,int>> in。因为set默认数据从小到大排列，正好可以通过两个for循环去定义三个点组合里面的两个左右端点A和C，然后通过find函数在in集合里面去寻找是否存在B点，如果存在ans+1。首尾遍历结束输出ans即可。

C. Yukina with Courses

算法：分组背包问题

分组背包是 $LC(01)$ 背包的一种变形。分组背包中物品被分成几组，每组中只能挑选出一件物品加入背包，这是与 01 背包的区别。

在 01 背包中，以每一件物品作为动态规划的每一阶段，但是在分组背包中要以每一组作为每一阶段。 其实很简单，代码如下：

```
for(int i=1;i<=k;i++)
    for(int c=v;c>=0;c--)
        for( each 物品j in 第i组 )
            if(c>=w[i]) f[c]=max(f[c],f[c-w[j]+v[j]]);
```

先枚举每一组，再枚举体积，最后枚举每组中的物品。

本题核心代码：

```
for(int i=1;i<=n;i++)
    for(int j=m;j>=0;j--)
        for(int k=1;k<=m;k++)
            if(j>=k) dp[j]=max(dp[j],dp[j-k]+ai);
```

D. Yukina with NEWS

大力模拟，$O(n^2)$ 枚举子串，$O(n)$ 或 $O(n^2)$ 匹配该串在原串中出现次数。

ps : 我把枚举出来的子串丢进 $trie$ 里面建了个 AC自动机，这样找子串的出现次数只要自动机跑一次就行了，然而长度还是 $O(n^3)$ 的。如果有严格优于 $O(n^2log^2n)$ 的黑科技做法可以私聊我，如果正确我请喝奶茶。

E. Yukina with Virginia

各种 dp 姿势都可以过，标程写的是 $dp[i][0]$ 表示使用前i个数字的方案数，且当前没有待匹配的十位(1或2)，$dp[i][1]$ 和 $dp[i][2]$ 表示使用前 i 个数字，且有待匹配的十位(1或2)的情况。 显然新来一位数是 0，只能接在前面待匹配的 1 或 2 后面；若是 1 或 2，可以独立成 1 个字母，或者接在前面待匹配的 1 或 2 之后，或者成为新的待匹配的十位；若是 3456，可以独立，也可以接在待匹配 1 或 2 之后；若是789，可以独立，可以接在待匹配的 1 之后。讨论清楚分类转移即可。 应该有更简洁的做法。

F. Yukina with LST

L 对 ST 产生影响的唯一情况是， S 和 T 在一行或一列，且 L 在他们中间，此时需要额外的 2 步。

G. Yukina with a Simple Problem On LIS

考虑每个数在几个上升子序列上。

维护一下 f_i, f_i' 分别表示从左往右的和从右往左的以 i 结尾的上升子序列的个数。

用树状数组可以轻松的搞一搞。

由于这道题目是我从别的题上面魔改过来的，如果你爆踩标程了可以找我...

H. Yukina with Knights

正确做法，把包含每个圆的最小圆求出来，可以建成一棵树，包含关系就是树的边。

把包含每个点的最小圆求出来，那么这个点属于树上对应的点(树上对应的点就是包含它的圆)。

那么答案其实就是求树上两点的 LCA 。

但由于点数比较少，$1s$ 卡不掉暴力求祖先的做法。

但是如果我们把询问改成 $1e6$ 的话...

I. Yukina with Square

忽略标程的做法。(尽管这种做法有扩展性，但它不是本题的最好做法)

设 $dp[i][j]$ 表示以 (i,j) 为右下角的正方形最大能有多大。

转移方程

$dp[i][j] = max(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]+1) \ if \ a[i][j] == 1$

$dp[i][j] = 0 \ if \ a[i][j] == 0$

J. Yasaka and squares

形如第四天的 J 题(求一个数可以拆分成多少素数相加)，这里只是把素数换成了平方数，一模一样的完全背包思想：

把平方数看作物品的价值，然后完全背包计数。

设 $dp[i][j]$ 表示前 i 个平方数选出若干个和为 j 的方案数

初始状态 $dp[0][0] = 1, dp[i][j] = 0$

转移方程 $dp[i][j] = dp[i-1][j] + dp[i-1][j-p[i]], p[i]$ 为第 i 个平方数的值

答案 $dp[n\text{以内的平方数的个数}][n]$

可以滚掉物品的维度

K. Haruhi and Kotori

因为对原序列的加减是 $Kotonacci$ 加减，显然不能直接用差分搞。

但是发现 $Kotonacci$ 数列也是由类似于前缀和的方法定义出来的，我们思考能不能定义一个 $Kotori$ 差分，使得 $Kotori$ 差分是 $Kotonacci$ 递推的逆操作。

设 $a[i]$ 是 $Kotori$ 差分数组，考虑 $[l, r]$ 操作对 $Kotori$ 差分数组的影响。

$a[l-1] = a[l-1] + f[1]$

$a[l] = a[l] + f[2] - b * f[1]$

$a[r] = a[r] - f[r-l+2]$

$a[r+1] = a[r+1] - a * f[r-l+1]$

搞不明白? 手玩一下就懂啦:)

搞完之后可以按 $Kotonacci$ 数列的递推式还原操作，最后加上原来的 $c[i]$ 就是答案啦。

要注意 $f[r-l+1]$ 这种数的大小是可能达到 $mod * mod$ 的，所以减完以后一个 mod 是救不回来的，你需要加上 $mod * mod$ 。具体细节可以看标程嗷。

L. Kaguya with points

n 非常大，n^2 过不了。但是 d 非常小。题目是让我们求 $|X1 - Y1| + |X2 - Y2| + ... + |Xd - Yd|$。带有绝对值，不能整体算。所以我们可以分类讨论，而对于 $|a - b|$ 只有两种可能：$a - b$ 或 $b - a$。对于各个维度上的值，都可以有正负两种取法。而确定了 x 的符号，y 就必须与 x 异号。所以我们可以 dfs 枚举各个维度上是取正的还是负的。

由于绝对值已经拆掉了，所以利用加法交换律，根据枚举的符号，先求出每一个点各个维度的的总和 $X[i]$，再求出每个点取相反符号的总和 $Y[j]$。答案一定是 $X[i] + Y[j]$。（也就是让这两点匹配）而要让 $X[i] + Y[j]$ 尽量大，也就是让 $X[i]$ 尽量大，$Y[j]$ 尽量大。