

漳州OI复赛连测

CSP-J-S（一）

题目标准题解

厦门 ZZN01 · 信奥陪跑计划

含题：偶数平方数 / 课程安排 / 排队 / 士兵组队

题解仅供参考，欢迎讨论更优做法

偶数平方数 题解

A

考虑直接枚举 i 作为 $\sqrt{x}$ ，那么接下来只需要判定 i^2 的每一位是否为偶数即可。

我们只需要把这个数拆掉每一位单独去判就行，这个时间复杂度是 $O(\log_{10} i^2)$ 的。

总时间复杂度是 $O(\sqrt{n} \log_{10} x)$ 。

Code

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

using namespace std;

#define ll long long

#define fst ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);

int main()
{
    fst
    ll n; cin >> n;
    ll ans = 0;
    for (ll i = 0; i * i <= n; i += 2) {
        ll x = i * i; bool ok = 1;
        while (x) {
            if ((x % 10) & 1) {
                ok = 0;
                break;
            }
            x /= 10;
        }
        ans += ok;
    }
    cout << ans;
    return 0;
}
```

课程安排 题解

B

对于 10\%

直接 2^{2n} 枚举每天选的东西，然后求答案即可。

对于 40\%

考虑设 f_i, j 表示前 i 天，语文选了 j 次此时的最大价值。

然后转移就是 $f_i, j = \max(f_{i-1}, j-1 + a_i, f_{i-1}, j + b_i)$ 。

答案就是 f_{2n}, n 。

时间复杂度 $O(n^2)$ 。

对于 100\%

考虑一个贪心：

先全选语文，然后拿出一些换成数学。

那么每个换成数学的贡献就是 $b_i - a_i$ ，我们对这个排序，然后贪心取最大的 n 个即可。

时间复杂度 $O(n \log n)$ 。

Code

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

using namespace std;

#define ll long long

const int N = 5e5 + 10;

int n, a[N << 1], b[N << 1];

int main()
{
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    cin >> n;
    vector<int> q;
    ll ans = 0;
    for (int i = 1; i <= 2 * n; i++) cin >> a[i] >> b[i], q.push_back(b[i] - a[i]), ans += a[i];
    sort(q.begin(), q.end());
    for (int i = q.size() - 1, j = 1; i >= 0 && j <= n; i--, j++) ans += q[i];
    cout << ans;
    return 0;
}
```

排队 题解

D

注意到题目要求我们计算合法排列的数量，不难想到进行排列 DP。

排列 DP 的常用套路是，设 $f_{i,*}$ 表示将 $[1, i]$ 或 $[i, n]$ 插入序列时，符合某些条件的方案数。

于是我们设 $f_{i,j}$ 表示将 $[1, i]$ 插入序列时，**最靠前的上升子序列的末尾位置**为 j 时，符合条件的方案数。

为什么我们要这样设计状态呢？

假设一个序列为 $3, 2, 1, 4, 5$ ，当我们想要新加入一个数字 6 时，如果我们把这个数字插入在 4 的后面，由于 4 已经和之前的 $3, 2, 1$ 分别构成了长度为 2 的上升子序列，因此插入在 4 的后面显然是不合法的。

也就是说，只要我们将一个新的数字插入在原本长度为 2 的上升子序列后，就一定是不合法的。

我们不难发现，想要让排列合法，就必须把新的数字填入到**原本上升子序列末尾**之前，所以要这样设计状态。

状态设计完，我们考虑怎么样转移。

转移分三种情况：

— 前 i 个构成了一个递减的序列时，此时序列中没有上升子序列，因此对于每个 $j \in [2, i + 1]$ ，我们都可以放置在后面并且产生一个上升的子序列，这个子序列就是**排列中最靠前的子序列**，因此，我们得出第一个转移方程。

$$f_{i+1,j} = f_{i+1,j} + 1 \quad (j \in [2, i + 1])$$

—

我们新插入一个数字，插入到整个序列的最左边，此时不会对上升子序列数量与长度产生任何影响，但是由于插入到序列开头，因此所有**原来末尾位置都会向后移动 1**，方程直接累加方案数即可。

$$f_{i+1,j+1} = f_{i+1,j+1} + f_{i,j} \quad (j \in [2, i])$$

—

当我们把数字插入到除了序列最前面以外的合法位置，也就是位于最靠前上升子序列之前的位置时，最前的上升子序列末尾位置会变成**插入的位置**，我们外层枚举当前末尾位置 $j \in [2, i]$ ，内层枚举下一个数字填入的位置 $k \in [2, j]$ ，填入数字，即：

$$f_{i+1,k} = f_{i+1,k} + f_{i,j} \quad (j \in [2, i], k \in [2, j])$$

到这里，我们的方程已经推完了，但我们注意到题目说的是**最长上升子序列长度不超过 2 的排列方案数**，因此还需要注意： $n, n-1, n-2 \dots 2, 1$ 这种倒序的排列也是一种方案，因此答案应该是

$$1 + \sum_{i=2}^n f_{n,i}$$

记得在过程中及时取模（采用减法常数更小），最后输出答案即可。

优化

但是这个方程的复杂度为 $O(n^3)$ ，不足以通过本题，因此我们需要考虑优化。

不难发现，在方程中需要增加的部分都是连续的，也就是一段区间加法，因此我们可以考虑进行差分，在每一层转

移之后进行合并。

到这里，我们通过差分已经把时间复杂度从原本的 $O(n^3)$ 优化到了 $O(n^2)$ 。

我们还可以使用滚动数组优化这个 DP 的空间复杂度。

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 5010;
int f[2][maxn];
const int mod = 1e7 + 7;
inline void add(int i,int l,int r,int d)
{
    f[i][l] += d;
    if(f[i][l] >= mod) f[i][l] -= mod;
    f[i][r + 1] -= d;
    if(f[i][r + 1] < 0) f[i][r + 1] += mod;
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n;
    cin >> n;
    for(int i = 1;i <= n;i++)
    {
        for(int j = 0;j <= n;j++) f[(i + 1) & 1][j] = 0;
        for(int j = 2;j <= i + 1;j++) add((i + 1) & 1,j,j,1);
        for(int j = 2;j <= i;j++) add((i + 1) & 1,2,j + 1,f[i & 1][j]);
        for(int j = 1;j <= n;j++)
        {
            f[(i + 1) & 1][j] += f[(i + 1) & 1][j - 1];
            if (f[(i + 1) & 1][j] >= mod) f[(i + 1) & 1][j] -= mod;
        }
    }
    int ans = 0;
    for(int i = 2;i <= n;i++)
    {
        ans += f[n & 1][i];
        if(ans >= mod) ans -= mod;
    }
    cout << ans + 1 << '\n';
    return 0;
}
```

士兵组队 题解

20pts

$O(n^2)$ 枚举所有解。

+30pts

注意到可能的 (a_i, b_i) 至多 $O(V^2)$ 种情况，枚举 i, j 对应的 (a_i, b_i) 和 (a_j, b_j) ，时间复杂度为 $O(V^4)$ 。

+30pts

可能的 (a_i, b_i) 至多 $O(\log^2 V)$ 对，类比之前的做法，时间复杂度为 $O(\log^4 V)$

100pts

首先我们把这题的 $\min$ 给去掉，那么我们按位算贡献，可以求出和。这是这种式子的通用套路。

考虑加上 $\min$ ，那么我们先按照 (a_i, b_i) 的最高位分为：(1, 0)，(0, 1)，(1, 1)，(0, 0) 四种情况。

先考虑不同的情况，可以发现用贡献的不同的组如下：

- (0, 0), (0, 1) 贡献为 $a_i \wedge a_j$ 。
- (0, 0), (1, 0) 贡献为 $b_i \wedge b_j$ 。
- (1, 1), (1, 0) 贡献为 $a_i \wedge a_j$ 。
- (1, 1), (0, 1) 贡献为 $b_i \wedge b_j$ 。
- 以下几种可能最高位相同，无法通过最高位比较：
 - (0, 0) 与 (1, 1)
 - (0, 1) 与 (1, 0)

对于无法比较的两组，我们把两组合并起来，然后就变成了子问题，递归处理下一位。

再考虑相同的情况，发现同组内部贡献最高位一定为 0，因此可以划分成 4 个子问题继续递归下去。

每次考虑最高层，一共至多递归 $O(\log V)$ 层，时间复杂度为 $O(n \log^2 V)$ 。