第 13 届电子科技大学趣味赛正式赛第一场题解

A.单击？双击

题解

开启双击模式的键出现次数一定恰好是两倍，其他的则恰好是一倍。以此判断即可。

参考代码（Python）：

```python
T = int(input())

for ___ in range(T):
    s = input()
    t = input()
    cnt0 = []
    cnt1 = []
    for i in range(1000):
        cnt0.append(0)
        cnt1.append(0)
    for i in s:
        cnt0[ord(i) - ord('a')] += 1
    for i in t:
        cnt1[ord(i) - ord('a')] += 1
    ans = ""
    for i in range(26):
        if cnt0[i] > 0 and cnt0[i] * 2 == cnt1[i]:
            ans += chr(i + ord('a'))
    print(len(ans))
    print(ans if len(ans) != 0 else "None")
```

出题人： Vingying0

B.大小写切换

题解

除了权值为 0 的字母，其他的字母都选上一定不会亏，所以贪心地选择所有权值不为 0 的字符即可。

注意特判全为 0 的情况

时间复杂度 $O(n)$。

出题人： Vingying0

参考代码(python):

```python
for _ in range(int(input())):
    c = [int(x) for x in input().split(' ')]
    s = input()
    ans = 0
    for cc in s:
        ans = ans + (c[ord(cc) - ord('a')] > 0)
    if ans == 0:
        ans = len(s)
    print(ans)
```

C.骆驼刺

题解

首先，我们可以观察到最左边和最右边的节点都必须从上到下依次连接，因为这些节点只有这一种连接方式才能够使得其被连接到。

那么，我们考虑内部的一个节点，它必须至少和它左上方及右上方的两个节点中的一个连接。

注意到当根系交叉时，实质上是形成了回路。我们考虑，对于内部的一个节点，它的左上方及右上方的两个节点，一定可以通过某些根系互相连接。此时，如果我们将内部的一个节点同时向左上方和右上方连接，将会导致这两个节点之间存在两条互相连接的路径，即这两个节点之间形成了回路。

因此，对于每个内部节点，都有两种选择，一种是向左上方连接，一种是向右上方连接。并且我们可以发现，只向一个方向连接一定不会形成回路，这是因为新加入的连接不会使得原来存在根系连接的两点之间再次出现新的根系连接。

共有 $\frac{(n-1)(n-2)}{2}$ 个内部节点，每个节点有两种选择，因此答案为 $2^{\frac{(n-1)(n-2)}{2}}$。

参考代码 （C++）：

```cpp
#include <bits/stdc++.h>
#define ll long long

using namespace std;

const ll p = 276276276;

ll n;

inline ksm(ll aa,ll bb) {
    ll sum = 1;
    while(bb) {
        if(bb&1) sum = sum * aa %p;
        bb >>= 1; aa = aa * aa %p;
    }
    return sum;
}

int main() {
    cin>>n;
    cout<<ksm(2,(n-2)*(n-1)/2);
    return 0;
```

```
}
```

出题人：maple276

D.上三角矩阵

题解

我们可以证明，在一个 01 矩阵中，如果对于 $1 \le i \le n$，都满足从左到右第 i 列恰有 i 个 1，且从下到上第 i 行恰有 i 个 1，那么这个矩阵一定是一个完全上三角矩阵。

我们可以这样证明：首先，最上面那行一定是满足完全上三角矩阵的条件的，最右边那列也是同样，都是全为 1。因此我们直接把这两行 / 列删除，那么我们就得到，在删除这两行 / 列之后最左边那列应当是全为 0 的，因为这一列原本有 1 个 1，而我们删除的行中恰有一个 1 在这一列中，因此我们将这一列的 1 恰好删掉了，于是这一列变为全 0。最下面那行也是同样。因此我们继续把最下面那行和最左边那列删除，那么我们现在相当于还要证明一个大小为 $(n-2) \times (n-2)$ 的满足上述条件的 01 矩阵是完全上三角矩阵。于是继续使用同样的方法，我们会将矩阵一直缩小为 1×1 或 0×0 的大小，对于这两种基本情况是非常显然的。因此利用这种归纳法，我们证明了，对于任意大小的 01 矩阵，如果它满足从左到右第 i 列恰有 i 个 1，且从下到上第 i 行恰有 i 个 1，那么这个矩阵一定是一个完全上三角矩阵。

我们发现，当交换两行时，这两行中 1 的个数也发生了交换，而其他行、列中 1 的个数都不发生改变。对于列也是类似的。因此，只要我们能够将每行的 1 的个数重排为一个有序的 $1 \sim n$ 的排列，对于列也是同样，那么我们就一定能够将这个 01 矩阵变为完全上三角矩阵。

因此我们只需要检查每一行 / 列中 1 的个数是不是一个 $1 \sim n$ 的排列即可。

参考代码 （C++）：

```cpp
#include <cstdio>
using namespace std;
const int N = 102;

int n, r[N], c[N];
char s[N][N];

int main(){
    scanf("%d", &n);
    bool flag = true;
    for (int i = 1; i <= n; i++){
        scanf("%s", s[i] + 1);
        int cnt = 0;
        for (int j = 1; j <= n; j++)
            cnt += s[i][j] == '1';
        if(!cnt || r[cnt])
            flag = false;
        r[cnt] = true;
    }
    for (int j = 1; j <= n; j++){
        int cnt = 0;
        for (int i = 1; i <= n; i++)
            cnt += s[i][j] == '1';
        if(!cnt || c[cnt])
            flag = false;
        c[cnt] = true;
    }
```

```
    puts(flag ? "YES" : "NO");
    return 0;
}
```

出题人： Natsuzora

E.打怪兽

题解

(1) 二分 $O(nlog(n))$ 做法：

二分答案为 k，该情况下可以造成 k 次伤害为 1 的攻击和 k 次伤害为 2 的攻击，为避免伤害为 2 的攻击用在了血量为 1 的怪物身上造成浪费，优先把伤害为 2 的攻击用在血量大于等于 2 的怪物身上，直至所有怪兽血量均小于 2 或伤害为 2 的攻击次数用尽。若情况为前者，即所有怪物血量均小于 2 且伤害为 2 的攻击有剩余，假设剩 m 次，则可视为伤害为 1 的攻击次数为 $k+m$ 次；若情况为后者，伤害为 1 的攻击次数为 k 次。此时比较怪物剩余血量之和与伤害为 1 的攻击次数大小，若后者不小于前者，则说明答案不大于 k；否则，说明答案大于 k。

(2) 线性 $O(n)$ 做法：

将两把武器的使用次数单独考虑，不难发现最终用时为两者中的较大值，那么现在就是要在总伤害大于等于总血量的前提下最小化使用次数较大的那一个。考虑最平均的情况，即 2 和 1 相等，最多多出一个 2 或 1，但由于伤害溢出的存在，不一定有解；而伤害溢出只可能出现在需要用 2，但剩余血量为 1 的情况下，即 1 的次数不会造成溢出。那么在最优情况下，即无溢出、伤害全部有效，1 至少需要使用的次数为血量为奇数的怪的数量，定义为 $cnt1$ 初始值，剩余全部用 2，计算出次数 $cnt2$。进一步平均二者，即存在三种情况：相等时，直接输出；$cnt1 > cnt2$ 时，1 过剩，此时由于每个 1 攻击对象不同，难免会造成溢出，平均下来最优时间即为$(cnt1 + cnt2)/2$；$cnt1 < cnt2$，可以将一部分 2 的伤害用两次 1 替代，则平均下来最优情况下就需要 $cnt2 - (cnt2 - cnt1)/3$。

参考代码 （C++）：

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n,cnt1=0,cnt2=0,sum=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        int x;
        scanf("%d",&x);
        sum+=x;
        if(x%2)cnt1++;
    }
    cnt2=(sum-cnt1)/2;
    if(cnt1==cnt2)printf("%d",cnt1);
    else if(cnt1>cnt2)printf("%d",(cnt1+cnt2)/2+((cnt1+cnt2)%2>0));
    else {
        int x=(cnt2-cnt1)/3;
        printf("%d",max(cnt1+x*2,cnt2-x));
    }
    return 0;
}
```

出题人： Charming

F.电力调度

题目描述

给出三个 $n \times m$ 的矩阵，其中两个只经过正确操作，另一个经过了正确操作和错误操作。

正确操作为

$$a[x_1][y_1]+=1, a[x_2][y_2]+=1, a[x_1][y_2]-=1, a[x_2][y_1]-=1$$

错误操作为

$$a[x_1][y_1]+=1, a[x_2+k][y_2+k]+=1, a[x_1][y_2]-=1, a[x_2][y_1]-=1$$

求含错误操作的矩阵以及错误操作次数。

题解

对正确操作，注意到

$$(x_1+y_1)+(x_2+y_2)=(x_1+y_2)+(x_2+y_1)$$

于是考虑构造一个与操作无关的量。对每个策略维护

$$\sum a[i][j] \times (i+j)$$

正确操作不会改变这个值，一次错误操作会使这个值增加 $2 \times k$。对三个矩阵分别算出该值后，用最大值和最小值的差除以 $2 \times k$ 即可得到错误操作的次数，最大值对应的矩阵即为含错误操作的矩阵。

由于$|a[i][j]| <= 10^6$ (卑微出题人在此谢罪)，记得开 $long\ \ long$。

时间复杂度：$O(nm)$ 空间复杂度：$O(nm)$

参考代码 （C++）：

```
#include <bits/stdc++.h>
using namespace std;
const long long inf = 1e18;
int n,m,k;
long long val1=0,val2=0,val3=0,Mx=-inf,Mn=inf;
int mat1[1010][1010],mat2[1010][1010],mat3[1010][1010];
int main () {
    scanf("%d%d%d",&n,&m,&k);
    for (int i = 1; i <= n; i++){
        for (int j = 1; j <= m; j++){
            scanf("%d",&mat1[i][j]);
            val1 += 1ll*mat1[i][j]*(i+j);
        }
    }
    Mx = max(Mx,val1);
    Mn = min(Mn,val1);
    for (int i = 1; i <= n; i++){
        for (int j = 1; j <= m; j++){
            scanf("%d",&mat2[i][j]);
            val2 += 1ll*mat2[i][j]*(i+j);
```

```
        }
    }
    Mx = max(Mx,val2);
    Mn = min(Mn,val2);
    for (int i = 1; i <= n; i++){
        for (int j = 1; j <= m; j++){
            scanf("%d",&mat3[i][j]);
            val3 += 1ll*mat3[i][j]*(i+j);
        }
    }
    Mx = max(Mx,val3);
    Mn = min(Mn,val3);
    if (Mx==val1){
        printf("1");
    }else if (Mx==val2){
        printf("2");
    }else {
        printf("3");
    }
    printf(" %lld",(Mx-Mn)/(k*2));
    return 0;
}
```

出题人： an_interesting_name

G.小C的排序

题解

由于逆序对个数是固定的，所以我们希望i越小的$f[i]$越大，我们贪心地从前往后选择当前我们能选的最大值，将小的数留在后面，这样对于当前位的$f[i]$是最大的，本质上是求字典序最大的排列使其逆序对个数为m。简单构造几组数据可以发现答案为$n, n-1, n-2, \ldots n-p, k, 1, 2, 3, \ldots$求出$p, k$即可。复杂度$O(n)$.

参考代码 （C++）：

```
#include <bits/stdc++.h>
#define SZ(x) ((int)x.size())

#ifdef LOCAL_DEFINE
#include "bits/debug.h"
#else
#define DEB(...)
#endif

using namespace std;

void Main() {
    long long n, m;
    cin >> n >> m;
    vector<int> a(n);
    if (m > 1ll * n * (n - 1) / 2) {
        cout << -1 << endl;
        return;
```

```
    }
    for (int i = 0; i < n; i++) {
        if (m <= n - i - 1) {
            for (int j = 0; j < i; j++) {
                a[j] = n - j;
            }
            a[i] = m + 1;
            int w = 1;
            for (int j = i + 1; j < n; j++) {
                if (w == m + 1) w++;
                a[j] = w;
                w++;
            }
            for (int i = 0; i < n - 1; i++) cout << a[i] << ' ';
            cout << a[n - 1] << endl;
            return;
        } else {
            m -= n - i - 1;
        }
    }
}

signed main() {

    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int o_o;
    cin >> o_o;
    while (o_o--) {
        Main();
    }
    return 0;
}
```

出题人： justcyl