武汉工程大学程序设计新生赛题解

A 学长的神之一手
B 矩阵游戏
C 炮哥的树学题
D 炮哥的数数题
E 小白玩游戏
F 小白找好区间(easy version)
G 小白找好区间(hard version)
H 小白的小清新构造
I 小白的代数题II
J 六边形
K 小豪的快逃哇!!!
L miss you

A. 学长的神之一手

解题思路

提示1：二分答案

若初始血量为hp时可以相遇，则初始血量大于hp时一定能相遇，所以可以二分答案

提示2：枚举

现在需要知道在已知初始血量h的情况下，能否相遇。
分别考虑两人，假设我们已知每人在初始血量h时，到达每个房间所需的最少金币数。 那么，就可以枚举每一个房间作为相遇的房间，计算两人最少一共需要花费的金币数。
注意，在相遇的那个房间，如果两人都需要花费金币，那么一共只用花费1金币（需减去重复花费的1金币）。

提示3：反悔贪心

现在的问题就是计算一个人初始血量为h时，到达每个房间所需的最少金币数。
先考虑一种简单的情况，只有花费1金币的操作，没有花费2金币的操作。
我们可以直接遍历所有房间，只要血量足够，就直接扣血。 当血量小于1时，从前面扣血的房间中找出扣血最多的(这里可以使用堆)，使用1金币，把扣掉的血加回来。 最终可以知道到达每个房间所需的最少金币数。
我们再考虑有花费2金币的操作的情况。只需拆解成2次花费1金币，即可解决。

时间复杂度: $O(n \log n \log H)$ (二分和堆各一个log)
空间复杂度: $O(n)$

完整代码

```
class Solution {
    public long solve(long[] a) {
        // 二分答案
        long l = 1;
        long r = Arrays.stream(a).sum() + 5;
        while (l < r) {
            long mid = (l + r) >> 1;
            if (check(mid, a)) {
                r = mid;
            } else {
                l = mid + 1;
            }
        }
        return l;
    }

    private boolean check(long h, long[] a) {
        int n = a.length;
        int[] minCost1 = getMinCost(h, a); // 从1到n，到达每个房间所需的最少金币数
        int[] minCost2 = getReverseMinCost(h, a); // 从n到1，到达每个房间所需的最少
```

```
金币数
        int min = Math.min(minCost2[0], minCost1[n - 1]); // 1号房间相遇和n号房间相
遇的情况
        for (int i = 1; i < n - 1; i++) { // 枚举每一个房间作为相遇的房间
            min = Math.min(min, minCost1[i] + minCost2[i]
                    - (minCost1[i] > minCost1[i - 1] && minCost2[i] > minCost2[i +
1] ? 1 : 0) // 如果两人走到i都需要再多花费1金币，则减去一次重复花费
            );
        }
        return min <= n / 2;
    }

    private int[] getReverseMinCost(long h, long[] a) {
        int n = a.length;
        long[] b = new long[n];
        for (int i = 0; i < n; i++) {
            b[i] = a[n - 1 - i];
        }
        int[] minCost = getMinCost(h, b);
        int[] ans = new int[n];
        for (int i = 0; i < n; i++) {
            ans[i] = minCost[n - 1 - i];
        }
        return ans;
    }

    private int[] getMinCost(long h, long[] a) {
        int n = a.length;
        int[] ans = new int[n];
        int cost = 0;
        // 大顶堆
        PriorityQueue<Long> queue = new PriorityQueue<>((o1, o2) ->
Long.compare(o2, o1));
        for (int i = 1; i < n; i++) {
            queue.add(a[i]); // 花费1金币少扣a[i]的血
            queue.add(a[i]); // 花费2金币变为增加a[i]的血
            h -= a[i]; // 当前房间扣血
            if (h <= 0) {
                // 血量不足时花费1金币从堆顶取出一个
                h += queue.remove();
                cost++;
            }
            ans[i] = cost;
        }
        return ans;
    }
}
```

B. 矩阵游戏

解题思路

本题中。对于每第 i 列而言，我们可以发现当第 i 列和第 $(n-i+1)$ 的 1 的个数超过三是不可能通过翻转找到合法状态的。直接输出 No 。此外每行只有两种状态翻转或不翻转，我们可以使用二分图来进行求解。本题可以使用 bfs 或并查集来处理二分图。此处对并查集做法进行简单说明。

我们可以开一个 $2*n$ 的并查集。对于第 i 行，i 表示不反转，$i+n$ 表示翻转。假设第 i ，k 行在同一列都有 1 ,那必须从其中选择一个进行翻转。则将 i 和 $k+n$ 合并在同一连通块将 $i+n$ 和 k 和合并在同一个连通块。假设第 i 行第 j 列有 1 ，第 k 行第 $(n-j+1)$ 列有 1 。则两个要么都不翻转，要么都翻转。则将 i 和 k 合并在同一个连通块,将 $i+n$ 和 $k+n$ 和合并在同一个连通块。合并完之后。只要 i 和 $i+n$ 不在同一连通块则必然可以找到合法状态。然后遍历每个联通块。在每一个连通块中选择翻转和不反转中数量少的来确定方案。

完整代码

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
typedef pair<int, int> PII;
const int N = 2e6 + 10;
string a[N];
int n, m;
int f[N];
void init() {
    for (int i = 0; i <= 2 * n; i++) {
        f[i] = i;
    }
}
int find(int x) {
    if (x  != f[x]) f[x] = find(f[x]);
    return f[x];
}
void merge(int x, int y) {
    f[find(x)] = find(y);
}

void solve() {
    cin >> n >> m;
    init();
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    vector<int> g[m + 1];
    vector<int> s(m + 1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (a[i][j] == '1') {
```

```
                s[j]++;
            }
            if (s[j] + s[m - j - 1] > 2) {
                cout << -1 << endl;
                return;
            }
            if (a[i][j] == '1') {
                g[j].push_back(i);
            }
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (a[i][j] == '1') {
                for (auto v : g[j]) {
                    if (v == i) continue;
                    merge(i, v + n);
                    merge(i + n, v);
                }
                for (auto v : g[m - j - 1]) {
                    if (v == i) continue;
                    merge(i, v);
                    merge(i + n, v + n);
                }
            }
        }
    }
    vector<int> ans[2 * n + 1];
    for (int i = 0; i < n; i++) {
        if (find(i) == find(i + n)) {
            cout << "-1" << endl;
            return;
        }
        ans[find(i)].push_back(i);
    }
    int res = 0;
    for (int i = 0; i < n; i++) {
        if (ans[i].size() < ans[i + n].size()) {
            res+= ans[i].size();
        } else res += ans[i + n].size();
    }
    cout << res << endl;
    for (int i = 0; i < n; i++) {
        if (ans[i].size() < ans[i + n].size()) {
            for (int j = 0; j < ans[i].size(); j++) {
                cout << ans[i][j] << " ";
            }
        } else {
            for (int j = 0; j < ans[i + n].size(); j++) {
                cout << ans[i + n][j] << " ";
            }
        }
    }
    cout << endl;
```

```
    return;
}
signed main() {
#ifdef LOCAL
    freopen("in.txt", "r", stdin);
    freopen("out.txt", "w", stdout);
#endif
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int t;
    t = 1;
    cin >> t;
    while (t--) solve();
    return 0;
}
```

C.炮哥的树学题

题目大意

给定一棵有 n 个节点且根节点为 1 的树，每个结点都有一个权值，询问每个节点的子树内距离该节点小于等于 k 的节点的权值和。

解题思路

我们定义 sum_i 表示点 i 的子树内所有节点（包含该节点）的权值和， res_i 表示每个点的答案，res_i 初始化为 sum_i 。那么考虑如何求节点 i 的答案呢？其实就是 sum_i 减去点i的子树内与点 i 相距 $k+1$ 的点的 sum 。所以我们可以先跑一次 dfs 求出每个点的 sum_i ， 然后再跑一次 dfs ,用一个数组 dep 来记录每个深度的点是谁，具体来说 dep_i 表示当我们遍历到某个点时，根节点到该节点路径上距离根节点距离为 i 的点是谁。那么我们走到一个深度为 $depth$ 的点 j 时，我们就可以直接通过 dep 数组找到点 j 上方距离其距离为 $k+1$ 的点，并将其 res 减去sum_j ，那么 dfs 完成之后，每个点的答案便可以求出。

时间复杂度 $O(n)$ ，空间复杂度 $O(n)$ 。

完整代码

```
#include<bitsdc++.h>
#define int long long
using namespace std;
typedef pair<int,int> PII;
const int N=2e6+10;
int h[N],e[N],ne[N],idx;
int a[N];
int sum[N];//sum[i]表示节点i的子树权值和
int ans[N];
int n,k;
int dep[N];
void add(int a,int b)
{
    e[idx]=b,ne[idx]=h[a],h[a]=idx++;
}
void dfs1(int u,int fa1)
{
    sum[u]=a[u];
    for(int i=h[u];i!=-1;i=ne[i])
    {
        int j=e[i];
        if(j==fa1) continue;
        dfs1(j,u);
        sum[u]+=sum[j];
    }
}
void dfs2(int u,int fa1,int dis,int depth)
{
    dep[depth]=u;
```

```
    if(depth-k>=0) ans[depth-k]+=sum[u]-a[u];
    for(int i=h[u];i!=-1;i=ne[i])
    {
        int j=e[i];
        if(j==fa1) continue;
        dfs2(j,u,dis,depth+1);
    }
}
void solve()
{
    memset(h,-1,sizeof(h));
    cin >> n >> k;
    for(int i=1;i<=n;i++) scanf("%lld",&a[i]);
    for(int i=1;i<n;i++)
    {
        int u,v;
        scanf("%lld%lld",&u,&v);
        add(u,v);
        add(v,u);
    }
    dfs1(1,0);
    dfs2(1,0,k,1);
    for(int i=1;i<=n;i++) printf("%lld ",sum[i]-ans[i]);
}
signed main()
{
    int t;
    t=1;
//  cin >>t;
    while(t--) solve();
}
```

D.炮哥的数数题

题目大意

见题面，已经够简洁了

解题思路

公式中的 max 和 min 是可以拆开的，所以我们可以将公式变为

$$\sum_{i=1}^{n}\sum_{j=i}^{n}max(i,j)*sum(i,j)-\sum_{i=1}^{n}\sum_{j=i}^{n}min(i,j)*sum(i,j)$$

然后下面我们只考虑计算前一个式子，后一个式子和前一个式子同理。我们考虑每一个值对答案的贡献，对于值 a_{pos} ，假设其前一个比它大的元素位于 l ，其后一个比它大的元素位于 r （利用单调栈求得），那么值 a_{pos} 只会对区间 $[l-1,r-1]$ 的子区间中包含下标 pos 的区间产生贡献。在这些区间内，值 a_{pos} 产生的贡献为

$$a_{pos}*\sum_{i=l-1}^{pos}\sum_{j=pos}^{r-1}sum(i,j)$$

然后我们考虑如何计算

$$\sum_{i=l-1}^{pos}\sum_{j=pos}^{r-1}sum(i,j)$$

计算该式子的核心是要计算一个类似于

$$\sum_{i=l}^{r}a_i*(r-i+1)$$

将该式变形为

$$sum(l,l)+sum(l,l+1)+...sum(l,r)$$

利用前缀和将其变形为

$$(pre[l]-pre[l-1])+(pre[l+1]-pre[l-1])+...(pre[r]-pre[l-1])$$

化简得

$$(pre[l]+pre[l+1]+...pre[r])-(r-l+1)*pre[l-1]$$

前半部分可以利用前缀和的前缀和得到，则 a_{pos} 的贡献可以求出， 遍历 pos 从 1 到 n ， 则数组中所有元素的贡献全部算出。min 与 max 同理。

时间复杂度 $O(n)$ ， 空间复杂度 $O(n)$ 。

完整代码

```
#include<bits/stdc++.h>
#define int long long
#define x first
#define y second
using namespace std;
typedef long long LL;
typedef long long ll;
typedef pair<int,int> PII;
const int N=2e6+10;
const int M=1e3+10;
int mod=998244353;
ll a[N],b1[N],b2[N],b3[N],b4[N];//右大，右小，左大，左小;
int pre1[N],pre11[N];
void solve(){
    int n;cin>>n;
    for(int i=1;i<=n;i++) cin>>a[i];
    stack<ll>s1,s2,s3,s4;
    for(int i=n;i>=1;i--){
        while(!s1.empty()&&a[s1.top()]<=a[i]) s1.pop();
        if(!s1.empty()) b1[i]=s1.top();
        else b1[i]=n+1;
        s1.push(i);
        while(!s2.empty()&&a[s2.top()]>=a[i]) s2.pop();
        if(!s2.empty()) b2[i]=s2.top();
        else b2[i]=n+1;
        s2.push(i);
    }
    for(int i=1;i<=n;i++){
        while(!s3.empty()&&a[s3.top()]<a[i]) s3.pop();
        if(!s3.empty()) b3[i]=s3.top();
        else b3[i]=0;
        s3.push(i);
        while(!s4.empty()&&a[s4.top()]>a[i]) s4.pop();
        if(!s4.empty()) b4[i]=s4.top();
        else b4[i]=0;
        s4.push(i);
    }
    for(int i=1;i<=n;i++){
        pre1[i]=(pre1[i-1]+a[i])%mod;
        pre11[i]=(pre11[i-1]+pre1[i])%mod;
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        int l=b3[i]+1,r=b1[i]-1;
        int len1=i-l+1,len2=r-i+1;
        ans=(ans+a[i]*len2%mod*(pre1[i]*len1%mod-(pre11[i-1]-pre11[l-
2])%mod))%mod;
        ans=(ans+a[i]*len1%mod*((pre11[r]-pre11[i]+mod)%mod-(len2-
1)*pre1[i]%mod)%mod)%mod;
        l=b4[i]+1,r=b2[i]-1;
        len1=i-l+1,len2=r-i+1;
        ans=(ans-a[i]*len2%mod*(pre1[i]*len1%mod-(pre11[i-1]-pre11[l-
2])%mod))%mod;
```

```
        ans=(ans-a[i]*len1%mod*((pre11[r]-pre11[i]+mod)%mod-(len2-
1)*pre1[i]%mod)%mod)%mod;
    }
    ans=(ans%mod+mod)%mod;
    cout<<ans<<"\n";
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    int _;
    _=1;
    //cin>>_;
    while(_--){
        solve();
    }
}
```

E 小白玩游戏

解题思路

简单题。

考虑枚举所有可能的 (i,j) 组合，我们不妨先将此数组用 $sort$ 排序为一个单调递增的数组，我们考虑枚举 i ，然后用二分找出 $[i,n]$ 这个区间内第一个大于等于 $L-a[i]$ 的位置和第一个小于等于 $R-a[i]$ 的位置，显然在这两个位置之间的所有 j 都符合要求，使答案加上此时这两个位置之间的长度即可。

时间复杂度 $o(nlogn)$ 。

完整代码

```
#include<bits/stdc++.h>
#define int long long
#define x first
#define y second
using namespace std;
typedef long long ll;
typedef pair<int,int> PII;
const int N=3e5+10;
const int M=1e3+10;
int mod=1e9+7;
//char mp[M][M];
int a[N];

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(0),cout.tie(0);
    int n,l,r;cin>>n>>l>>r;
    for(int i=1;i<=n;i++) cin>>a[i];
    sort(a+1,a+n+1);
    int ans=0;
    for(int i=1;i<=n;i++){
        int k1=lower_bound(a+1+i,a+n+1,l-a[i])-a;
        int k2=upper_bound(a+1+i,a+n+1,r-a[i])-a-1;
        ans+=k2-k1+1;
    }
    cout<<ans<<"\n";
    return 0;
}
```

F.小白找好区间(easy version)

解题思路

本场的第一个签到题

由于本题的数据范围很小，因此可以暴力枚举左右端点，然后暴力 $check$ 枚举的区间内目标字符串的个数即可。

完整代码

```
#include<bits/stdc++.h>
#define int long long
#define x first
#define y second
using namespace std;
typedef long long LL;
typedef long long ll;
typedef pair<int,int> PII;
const int N=3e5+10;
const int M=1e3+10;
int mod=1e9+7;
string s[N];

void solve(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++) cin>>s[i];
    int ans=0;
    for(int i=1;i<=n;i++){
        for(int j=i;j<=n;j++){
            int cnt=0;
            for(int k=i;k<=j;k++){
                if(s[k]=="byl") cnt++;
            }
            if(cnt>j-i+1-cnt) ans++;
        }
    }
    cout<<ans<<"\n";
}

signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr),cout.tie(nullptr);
    int _;
    _=1;
    //cin>>_;
    while(_--){
        solve();
```

```
        }
    }
```

G.小白找好区间(hard version)

解题思路

中期题。

我们建立一个数组 a ，当第 i 个字符串是目标字符串时，令 $a_i = 1$ ，反之令 $a_i = 0$ ，同时建立一个数组 b ，当第 i 个字符串是其他字符串时，令 $b_i = 1$ ，反之令 $b_i = 0$ ，然后分别对数组 a 和 b 做前缀和操作，那么对于区间 $[l, r]$ ，目标字符串的个数就为 $a_r - a_{l-1}$ ，同理其他字符串的个数为 $b_r - b_{l-1}$ ，那么根据题意，如果这个区间是一个好区间，那么有：

$$a_r - a_{l-1} > k \times (b_r - b_{l-1})$$

进一步得到：

$$a_r - k \times b_r > a_{l-1} - k \times b_{l-1}$$

我们再建立一个数组 c ，并且令 $c_i = a_i - k \times b_i$ ，那么上式就变成了 $c_r > c_{l-1}$ ，题目转化为求有多少个 (i, j) 满足 $c_j > c_{i-1}$ ，这是一个很经典的题目，离散化树状数组即可。

时间复杂度 $o(nlogn)$ 。

完整代码

```
#include<bits/stdc++.h>
#define int long long
#define x first
#define y second
using namespace std;
typedef long long LL;
typedef long long ll;
typedef pair<int, int> PII;
const int N = 3e5 + 10;
const int M = 1e3 + 10;
int mod = 1e9 + 7;
int a[N], b[N], c[N];
ll tree[N];
ll lowbit(ll num) {
    return num & -num;
}
void build(ll s, ll num) {
    for (ll i = s; i <= 200010;
            i += lowbit(i)) tree[i] += num;
}
ll ask(ll s) {
    ll ans = 0;
    for (ll i = s; i >= 1; i -= lowbit(i)) ans += tree[i];
    return ans;
}
void solve() {
```

```
    int n, k;
    cin >> n >> k;
    map<int, int>mp1, mp2;
    for (int i = 1; i <= n; i++) {
        string s;
        cin >> s;
        if (s == "byl") a[i] = 1;
        else b[i] = 1;
        a[i] += a[i - 1], b[i] += b[i - 1];
        c[i] = a[i] - k * b[i];
        mp1[c[i]]++;
    }
    mp1[0]++;
    int idx = 0;
    for (auto it : mp1) mp2[it.x] = ++idx;
    for (int i = 0; i <= n; i++) {
        build(mp2[c[i]], 1);
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        int op = mp2[c[i - 1]];
        build(op, -1);
        ans += ask(200010) - ask(op);
    }
    cout << ans << "\n";
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int _;
    _ = 1;
    //cin>>_;
    while (_--) {
        solve();
    }
}
```

H.小白的小清新构造

解题思路

简单题。

首先判断当 $k=1$ 时，不难发现只有 $n=1$ 时才可能有解，其他的情况都是无解。

其次不难发现，当 k 或者 $k \geq n$ 时，采用类似 $1\ 2\ 1\ 2\ 1\ 2...$ 的构造方式是最优的 。

最后一种情况是 $k<n$ 并且 k 时，我们构造的时候以每 k 个作为一个循环节，所以只需构造前 k 个元素即可，最优的构造方式如下，$1\ 2\ 1\ 2\ 1\ 2...3$ ，即只有第 k 个数是 3 ，从 1 到 $k-1$ 都按 $1\ 2\ 1\ 2...$ 这样构造即可。

时间复杂度 $o(n)$ 。

完整代码

```
#include<bits/stdc++.h>
#define int long long
#define x first
#define y second
using namespace std;
typedef long long LL;
typedef long long ll;
typedef pair<int, int> PII;
const int N = 3e5 + 10;
const int M = 1e3 + 10;
int mod = 1e9 + 7;
int a[N];

void solve() {
    int n, k, s;
    cin >> n >> k >> s;
    if(k==1){
        if(n!=1){
            cout<<"NO\n";
            return ;
        }
    }
    if(k>=n||k%2==0){
        for(int i=1;i<=n;i++){
            if(i%2) a[i]=1;
            else a[i]=2;
        }
    }else{
        for(int i=1;i<=k;i++){
            if(i==k) a[i]=3;
            else{
                if(i%2) a[i]=1;
```

```
                else a[i]=2;
            }
        }
        for(int i=k+1;i<=n;i++) a[i]=a[i-k];
    }
    int sum=0;
    for(int i=1;i<=n;i++) sum+=a[i];
    if(sum<=s){
        cout<<"YES\n";
        for(int i=1;i<=n;i++) cout<<a[i]<<" ";
    }else cout<<"NO\n";
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int _;
    _ = 1;
    //cin>>_;
    while (_--) {
        solve();
    }
}
```

I.小白的代数题II

解题思路

中期题。

设小白拿了 x 次糖果，小曾拿了 y 次糖果，则容易列出方程 $ax+\lfloor\frac{x}{2}\rfloor\times3+by=n$ 。

不难发现，我们要对 x 的奇偶进行讨论。

当 x 为奇数时，方程变为 $ax+\frac{x-1}{2}\times3+by=n$ ，即为 $(2a+3)x+2by=2n+3$ 。

当 x 为偶数时，方程变为 $ax+\frac{x}{2}\times3+by=n$ ，即为 $(2a+3)x+2by=2n$ 。

用 $exgcd$ 分别解这两个二元一次不定方程，求出最接近 L 且大于等于 L 的符合条件的 x ，和 R 作比较即可，这里有一个细节就是要判断 x 取得目标值时 y 是否大于等于 0 ，因为小曾拿糖果的次数不能为负。

时间复杂度 $o(Tlogn)$ 。

完整代码

```
#include<bits/stdc++.h>
#define int long long
#define x first
#define y second
using namespace std;
typedef long long LL;
typedef long long ll;
typedef pair<int, int> PII;
const int N = 3e5 + 10;
const int M = 1e3 + 10;
int mod = 1e9 + 7;
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
void Exgcd(int a, int b, int& x, int& y) {
    if (!b) {
        x = 1, y = 0;
        return ;
    }
    int p;
    Exgcd(b, a % b, x, y);
    p = x;
    x = y;
    y = p - a / b * y;
    return ;
}
void solve() {
    int n, a, b, L, R;
```

```
    cin >> n >> a >> b >> L >> R;
    int aa = 2 * a + 3, bb = 2 * b, cc = 2 * n + 3;
    int f = 0;
    //x为奇数
    if (cc % __gcd(aa, bb) == 0) {
        int x0 = 0, y0 = 0;
        Exgcd(aa, bb, x0, y0);
        x0 *= cc / __gcd(aa, bb);
        int s, dx = bb / __gcd(aa, bb);
        if (dx > 0) s = ceil((L - x0) * 1.0 / dx);
        else s = floor((L - x0) * 1.0 / dx);
        x0 += s * dx;
        if (x0 % 2 == 0){
             if(dx%2) x0+=dx;
             else x0=R+1;
        }
        if (x0 * aa <= cc && x0 <= R) f++;
    }
    cc = 2 * n;
    //x为偶数
    if (cc % __gcd(aa, bb) == 0) {
        int x0 = 0, y0 = 0;
        Exgcd(aa, bb, x0, y0);
        x0 *= cc / __gcd(aa, bb);
        int s, dx = bb / __gcd(aa, bb);
        if (dx > 0) s = ceil((L - x0) * 1.0 / dx);
        else s = floor((L - x0) * 1.0 / dx);
        x0 += s * dx;
        if (x0 % 2){
             if(dx%2) x0+=dx;
             else x0=R+1;
        }
        if (x0 * aa <= cc && x0 <= R) f++;
    }
    if (f) cout << "YES\n";
    else cout << "NO\n";
}

signed main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int _;
    _ = 1;
    cin >> _;
    while (_--) {
        solve();
    }
}
```

J.六边形

题目大意

给定 7 个正六边形，每个正六边形的边上的值是 1 到 6 的排列，将这7个正六边形按照如图组合。你判断是否存在一种拼接方式使得这些六边形中相接的边的权值都相同，每个正六边形可以旋转。

解题思路

7! 枚举每个正六边形的位置，以正中间的正六边形为基准，其他六个六边形分别有一条边与中间的六边形的一条边相连，所以可以求出其他六个正六边形旋转了多少次，这样可以初步固定图形，再判断每个正六边形相连的边是否权值相同，如果所有相连的边权值相同，输出 YES，否则输出 NO。

```
#include <algorithm>
#include <iostream>

int t, p[7], a[7][6], ad[7], c[7][6];

int main() {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);

    for (int i = 0; i < 7; i++) {
        p[i] = i;
        ad[i] = -1;
        for (int j = 0; j < 6; j++) {
            std::cin >> a[i][j];
            a[i][j]--;
            c[i][a[i][j]] = j;
        }
    }
    bool solved = 0;
    do {
        ad[0] = c[p[0]][0];
        bool bad = false;
        for (int i = 1; i < 7; i++) {
            ad[i] = c[p[i]][a[p[0]][(ad[0] + i - 1) % 6]];
        }
        for (int i = 1; i < 6; i++) {
            if (a[p[i]][(ad[i] + 5) % 6] != a[p[i + 1]][(ad[i + 1] + 1) % 6]) {
                bad = true;
                break;
            }
        }
        if (a[p[6]][(ad[6] + 5) % 6] != a[p[1]][(ad[1] + 1) % 6]) {
            bad = 1;
        }
        if (!bad) {
            solved = 1;
            break;
```

```
            }
        } while (std::next_permutation(p, p + 7));
        if (solved) {
            std::cout << "YES\n";
        } else {
            std::cout << "NO\n";
        }
        return 0;
    }
```

K.小豪的快逃哇!!!

题目大意

n个点m条边的有向图，假设当前在u 号点，在所有能通向未到达点的边中**等概率**的选择进行逃亡。问你从1 号节点出发，算出走投无路停在i 号点$(1 \leq i \leq n)$ 的概率，输出每个答案对$1e9+7$ 取模的结果。

解题思路

做法： **状压dp**

$dp[i,j]$ 其中i 是一个二进制数，其中每一位的$1/0$ 表示该点是否走过，j 表示以i 的状态下当前在j 号点的概率。

状态转移为$dp[i<<1|k,k]$ += $\frac{dp[i,j]}{cnt}$ ，其中表示j 到k 有条边，且i 中第k 位为0 ，cnt 表示从j 出发还有多少条未到达点的边的条数。

当cnt 为0 的时候即走到第j 号点走投无路并记录答案。

时间复杂度 $O(2^n * m)$

完整代码

```
#include <bits/stdc++.h>
#define x first
#define y second
#define int long long
#define endl "\n"
#define all(v) v.begin(), v.end()
using namespace std;
const int N = 200010, INF = 0x3f3f3f3f, mod = 1000000007;

typedef long long LL;
typedef unsigned long long ULL;
typedef pair<int, int> PII;
typedef pair<PII, int> PIII;

int qmi(int a, int b) {
    int res = 1;
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

void solve(){
    int n, m;
    cin >> n >> m;
```

```
    vector<vector<int>> adj(n);
    for(int i = 0; i < m; i ++ ) {
        int u, v;
        cin >> u >> v;
        u --, v --;
        adj[u].emplace_back(v);
    }
    vector<int> inv(m + 1);
    inv[0] = 1;
    for(int i = 1; i <= m; i ++ ) inv[i] = qmi(i, mod - 2);
    vector dp(1 << n, vector<int>(n));
    dp[1][0] = 1;
    vector<int> ans(n);
    for(int i = 1; i < 1 << n; i ++ ) {
        for(int j = 0; j < n; j ++ ) {
            if(i >> j & 1) {  // i中有j 且当前走到j
                int cnt = 0;
                for(auto v: adj[j]) {
                    if((i & (1 << v)) == 0) {
                        cnt ++;
                    }
                }
                if(!cnt) {  // 表示没有路可以走了
                    ans[j] += dp[i][j];
                    ans[j] %= mod;
                }
                for(auto v: adj[j]) {
                    if((i & (1 << v)) == 0) {
                        dp[i | 1 << v][v] += dp[i][j] * inv[cnt] % mod;
                        dp[i | 1 << v][v] %= mod;
                    }
                }
            }
        }
    }
    for(int i = 0; i < n; i ++ ) cout << ans[i] << "\n";
}
signed main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int T = 1;
    //cin >> T;
    while(T -- ){
        solve();
    }
    return 0;
}
```

L.miss you

解题思路

可见数据范围为 $3 \leq n, m \leq 500$, $n \times m \geq 12$ ，所以直接 BFS 即可。

根据题意，InHng **只能破坏一个蓝色陷阱**，Cynthia **只能破坏一个红色陷阱**，**黄色陷阱**需要**两人共同破坏**，所有情况如下

- 可以逃离迷宫的情况：
  - 二人直接相遇
  - 二人之间只有一个**陷阱**(无论是什么颜色)
  - 二人之间有两个陷阱，InHng 一方的陷阱是**蓝色陷阱**，Cynthia 一方的陷阱是**红色陷阱**
- 不能逃离迷宫的情况：
  - 二人之间**被墙隔断**，无法相遇
  - 二人之间**有一个蓝色陷阱和一个红色陷阱**，但**蓝色陷阱在 Cynthia 一侧，红色陷阱在 InHng 一侧**
  - 二人之间有**两个及以上同色陷阱**

根据这个条件可能会想到如何判断两人是否破坏过陷阱，又如何判断两人能否走到同一个黄色陷阱的两侧，此时不妨~~更改规则~~换个思路，不如只让其中一个人移动，此时 ta 就拥有了**两个能力之一**：

- 直接破坏黄色陷阱的能力
- 先破坏自己原先就能破坏的颜色的陷阱，再破坏自己原先不能破坏的陷阱的能力

即以下两种情况进行了修改：

- 如果不经过蓝色陷阱和红色陷阱，只经过**一个**黄色陷阱到达另一个人所在的位置，那么根据题目原先的规则，两个人必定可以分别走到这个黄色陷阱的两侧，将这个黄色陷阱破坏
- 如果先后经过两个不同颜色的陷阱到达另一个人所在的位置**(先经过自己可以破坏的颜色的陷阱，再经过另一个人可以破坏的颜色的陷阱)**，那么根据题目原先的规则，两个人肯定可以各自走到自己可以破坏的陷阱旁破坏陷阱，然后通过一条没有任何陷阱的路，彼此相遇

以 InHng 的情况为例，以 InHng 的位置的东南西北进行 BFS ，初始状态的二进制表示为 `00`，其中 `00` 表示没有使用过魔法，`01` 表示已经破坏了一个蓝色陷阱，`10` 表示已经破坏了一个红色陷阱，`11` 表示破坏了一个黄色陷阱，或者破坏了蓝色陷阱和红色陷阱各一个，如果下一步是

- 墙壁：那么不能往这个方向走
- 蓝色陷阱：如果此时 InHng 没有破坏过陷阱，则可以破坏这个陷阱，状态置为 `01`，如果已经破坏过蓝色陷阱或黄色陷阱，根据题目原先的规则此时不能破坏陷阱，如果已经破坏过红色陷阱，当然也不能破坏这个陷阱，因为根据原来的规则这个陷阱只能由 Cynthia 破坏后才能 InHng 才能通过这里
- 红色陷阱：如果此时破坏过红色陷阱或者黄色陷阱，根据规则不能再破坏这个陷阱，其他情况则都可以，即将原来的状态对二进制 `10` 进行**位运算或操作**，即状态变为 `1X` (`X` 表示原先到达这个位置之前的状态)
- 黄色陷阱：如果 InHng 破坏过陷阱，则不能破坏这个陷阱，否则破坏陷阱，将状态置为 `11`

注意：在 BFS 时需要存储当前破坏陷阱的状态，如果没有以当前状态到达过下一个位置，则可以走，否则不能走.

完整代码

C++

```cpp
#include <iostream>
#include <queue>
#include <array>
#include <cstring>

const int dx[] = {0, 0, 1, -1};
const int dy[] = {1, -1, 0, 0};

int main() {
    int n, m;
    std::string ans = "There's nothing left!";
    std::cin >> n >> m;
    char grid[n][m];
    bool memory[n][m][4];
    memset(memory, false, sizeof(memory));
    std::pair<int, int> InHng, Cynthia; // 两个人的位置
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            std::cin >> grid[i][j];
            if (grid[i][j] == 'I') { // 将 InHng 设为起点
                InHng = {i, j};
            } else if (grid[i][j] == 'C') { // 将 Cythia 设为终点
                Cynthia = {i, j};
            }
        }
    }

    std::queue<std::array<int, 3>> q;
    q.push({InHng.first, InHng.second, 0}); // 将 InHng 的位置和初始状态放入
    memory[InHng.first][InHng.second][0] = true; // 对当前状态到达过的位置进行标记
    while (q.size()) {
        auto [x, y, z] = q.front();
        q.pop();
        if (std::pair<int, int>(x, y) == Cynthia) { // 到达了终点
            ans = "I miss you!";
            break;
        }
        for (int i = 0; i < 4; ++i) {
            int next_x = x + dx[i], next_y = y + dy[i];
            if (grid[next_x][next_y] == '#') { // 墙不能走
                continue;
            } else if (grid[next_x][next_y] == 'Y') { // 遇到黄色陷阱
                if (z) { // 但凡用过一次魔法就不能走
                    continue;
                }
                memory[next_x][next_y][z | 3] = true; // 同时用两个魔法
                q.push({next_x, next_y, z | 3});
            } else if (grid[next_x][next_y] == 'R') { // 遇到红色陷阱
```

```
                if (z > 1) { // 如果已经消除过红色陷阱了
                    continue;
                }
                memory[next_x][next_y][z | 2] = true;
                q.push({next_x, next_y, z | 2});
            } else if (grid[next_x][next_y] == 'B') { // 如果遇到了蓝色陷阱
                if (z) { // 如果已经消除过蓝色陷阱不能走，如果消除过红色陷阱也不能走，
因为 InHng 没法先消除红色陷阱，再消除蓝色陷阱
                    continue;
                }
                memory[next_x][next_y][z | 1] = true;
                q.push({next_x, next_y, z | 1});
            } else if (not memory[next_x][next_y][z]) { // 如果没有以当前状态到达这
个地方，则可以走
                memory[next_x][next_y][z] = true;
                q.push({next_x, next_y, z});
            }
        }
    }
    std::cout << ans;
    return 0;
}
```

Python

```
import queue

dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

ans = "There's nothing left!"
n, m = map(int, input().split())
grid = [input() for _ in range(n)]
memory = [[[False] * 4 for _ in range(m)] for _ in range(n)]
InHng, Cynthia = None, None  # 两个人的位置

for i in range(n):
    for j in range(m):
        if grid[i][j] == 'I':  # 将 InHng 设为起点
            InHng = (i, j)
        elif grid[i][j] == 'C':  # 将 Cythia 设为终点
            Cynthia = (i, j)

q = queue.Queue()
q.put((InHng[0], InHng[1], 0))  # 将 InHng 的位置和初始状态放入
memory[InHng[0]][InHng[1]][0] = True  # 对当前状态到达过的位置进行标记

while q.qsize():
    x, y, z = q.get()
    if (x, y) == Cynthia: # 到达了终点
        ans = "I miss you!"
```

```
            break
        for i in range(4):
            next_x, next_y = x + dx[i], y + dy[i]
            if grid[next_x][next_y] == '#':  # 墙不能走
                continue
            elif grid[next_x][next_y] == 'Y':  # 遇到黄色陷阱
                if z:  # 但凡用过一次魔法就不能走
                    continue
                memory[next_x][next_y][z | 3] = True  # 同时用两个魔法
                q.put((next_x, next_y, z | 3))
            elif grid[next_x][next_y] == 'R':  # 遇到红色陷阱
                if z > 1:  # 如果已经消除过红色陷阱了
                    continue
                memory[next_x][next_y][z | 2] = True
                q.put((next_x, next_y, z | 2))
            elif grid[next_x][next_y] == 'B':  # 如果遇到了蓝色陷阱
                if z:  # 如果已经消除过蓝色陷阱不能走，如果消除过红色陷阱也不能走，因为
InHng 没法先消除红色陷阱，再消除蓝色陷阱
                    continue
                memory[next_x][next_y][z | 1] = True
                q.put((next_x, next_y, z | 1))
            elif not memory[next_x][next_y][z]:  # 如果没有以当前状态到达这个地方，则可以
走
                memory[next_x][next_y][z] = True
                q.put((next_x, next_y, z))
    print(ans)
```

Go

```
package main

import (
    "bufio"
    . "fmt"
    "os"
)

var in = bufio.NewReader(os.Stdin)
var out = bufio.NewWriter(os.Stdout)

var dx = []int{0, 0, 1, -1}
var dy = []int{1, -1, 0, 0}

func solve() {
    var n, m int
    Fscan(in, &n, &m)
    ans := "There's nothing left!"
    grid := make([][]rune, n)
    memory := make([][][]bool, n)
    for i := range grid {
        memory[i] = make([][]bool, m)
```

```
        for j := range memory[i] {
            memory[i][j] = make([]bool, 8)
        }
    }
    var InHng, Cynthia [2]int // 两个人的位置
    for i := 0; i < n; i++ {
        grid[i] = make([]rune, m)
        input := ""
        Fscan(in, &input)
        for j, c := range input {
            grid[i][j] = c
            if grid[i][j] == 'I' { // 将 InHng 设为起点
                InHng = [2]int{i, j}
            } else if grid[i][j] == 'C' { // 将 Cythia 设为终点
                Cynthia = [2]int{i, j}
            }
        }
    }

    q := make([][3]int, 0)
    q = append(q, [3]int{InHng[0], InHng[1], 0}) // 将 InHng 的位置和初始状态放入
    memory[InHng[0]][InHng[1]][0] = true         // 对当前状态到达过的位置进行标记
    for len(q) > 0 {
        x, y, z := q[0][0], q[0][1], q[0][2]
        q = q[1:]
        if [2]int{x, y} == Cynthia { // 到达了终点
            ans = "I miss you!"
            break
        }
        for i := 0; i < 4; i++ {
            next_x, next_y := x+dx[i], y+dy[i]
            if grid[next_x][next_y] == '#' { // 墙不能走
                continue
            } else if grid[next_x][next_y] == 'Y' { // 遇到黄色陷阱
                if z != 0 { // 但凡用过一次魔法就不能走
                    continue
                }
                memory[next_x][next_y][z|3] = true // 同时用两个魔法
                q = append(q, [3]int{next_x, next_y, z | 3})
            } else if grid[next_x][next_y] == 'R' { // 遇到红色陷阱
                if z > 1 { // 如果已经消除过红色陷阱了
                    continue
                }
                memory[next_x][next_y][z|2] = true
                q = append(q, [3]int{next_x, next_y, z | 2})
            } else if grid[next_x][next_y] == 'B' { // 如果遇到了蓝色陷阱
                if z != 0 { // 如果已经消除过蓝色陷阱不能走，如果消除过红色陷阱也不能
走，因为 InHng 没法先消除红色陷阱，再消除蓝色陷阱
                    continue
                }
                memory[next_x][next_y][z|1] = true
                q = append(q, [3]int{next_x, next_y, z | 1})
            } else if !memory[next_x][next_y][z] { // 如果没有以当前状态到达这个地
方，则可以走
```

```
                memory[next_x][next_y][z] = true
                q = append(q, [3]int{next_x, next_y, z})
            }
        }
    }
    Println(ans)
}

func main() {
    defer out.Flush()
    solve()
}
```