Multiplication

签到题，直接根据题意写即可。

```cpp
# include <iostream>

using namespace std;

typedef long long LL;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T;
    cin >> T;
    while (T--) {
        LL n, k;
        cin >> n >> k;
        for (int i = 0; ; i++, k *= k)
            if (k >= n) {
                cout << i << endl;
                break;
            }
    }
    return 0;
}
```

Spread

并查集，在根结点的位置存一下该集合的人数即可。最后枚举一遍传染源，将其对应根结点互不相同的集合的人数相加即为答案。

（由于此题的查询并非在线操作，所以也可以用搜索来写

```cpp
# include <iostream>

using namespace std;

const int N = 1e5 + 5;
int n, m, t;
int source[N], cnt[N], pre[N];
bool vis[N];

int find(int x)
{
    return x == pre[x] ? x : pre[x] = find(pre[x]);
}

void merge(int a, int b)
{
    cnt[a] += cnt[b];
    pre[b] = a;
    return;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m >> t;
        for (int i = 1; i <= n; i++) {
            cnt[i] = 1;
            pre[i] = i;
            vis[i] = false;
        }
        for (int i = 0; i < m; i++)
            cin >> source[i];
        while (t--) {
            int a, b;
            cin >> a >> b;
            a = find(a), b = find(b);
            if (a != b)
                merge(a, b);
        }
        int res = 0;
        for (int i = 0; i < m; i++) {
            int root = find(source[i]);
            if (!vis[root]) {
                vis[root] = true;
                res += cnt[root];
```

```
            }
        }
        cout << res << endl;
    }
    return 0;
}
```

Transport

简单的 BFS，不过对于每组测试数据，需要跑两遍 BFS.

首先以指挥部为起点，找到到小区的最短距离；再以小区为起点，找到到医院的最短距离。

两者中任一一个不存在路径即为无解，否则，两者相加即为答案。

```
# include <iostream>
# include <cstring>
# include <queue>
# include <algorithm>

using namespace std;

typedef pair<int, int> PII;
typedef pair<PII, int> PIII;
const int N = 1010, INF = 0x3f3f3f3f;
const int dx[4] = { -1, 1, 0, 0 }, dy[4] = { 0, 0, -1, 1 };
int n, m;
char F[N][N];
bool vis[N][N];

int bfs(int stx, int sty, int edx, int edy)
{
    memset(vis, false, sizeof vis);
    queue<PIII> q;
    q.push({ {stx, sty}, 0 });
    vis[stx][sty] = true;
    while (!q.empty()) {
        auto t = q.front();
        q.pop();
        int a = t.first.first, b = t.first.second, dis = t.second;
        if (a == edx && b == edy)
            return dis;
        for (int i = 0; i < 4; i++) {
            int x = a + dx[i], y = b + dy[i];
            if (x < 1 || x > n || y < 1 || y > m || F[x][y] == '*' ||
vis[x][y])
                continue;
            q.push({ {x, y}, dis + 1 });
            vis[x][y] = true;
        }
    }
    return INF;
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m;
        for (int i = 1; i <= n; i++)
            cin >> F[i] + 1;
```

```
        int x1, y1, x2, y2, x3, y3;
        cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;
        int dis1 = bfs(x1, y1, x2, y2), dis2 = bfs(x2, y2, x3, y3);
        if (dis1 == INF || dis2 == INF)
            cout << -1 << endl;
        else
            cout << dis1 + dis2 << endl;
    }
    return 0;
}
```

OneTrunKill

模拟题（一开始的版本更加恶心，无奈之下，只能简化成这样）。

在读懂题意的基础上，模拟每一步的操作，最终取最大值即可。由于每一步操作可以选择不同的手牌打，这个过程可以用 DFS 来实现。

```cpp
# include <iostream>
# include <cstring>
# include <algorithm>

using namespace std;

int n, m, pos, res;
int card[5], q[11];

void dfs(int u, int fire, int damage, int attack, bool atk, bool sf, bool can_sf)
{
    res = max(res, damage);
    if (u == n + pos || !fire && (sf || !can_sf))
        return;
    // 用普攻
    if (!atk && fire)
        dfs(u, fire - 1, damage + attack, 0, true, sf, can_sf);
    // 用 1
    if (card[1]) {
        card[1]--;
        if (!sf && can_sf)
            dfs(u + 1, fire, damage + 4, attack, atk, true, false);
        else if (fire)
            dfs(u + 1, fire - 1, damage + 4, attack, atk, sf, true);
        card[1]++;
    }
    // 用 2
    if (card[2]) {
        card[2]--;
        if (!sf && can_sf)
            dfs(u + 1, fire, damage + 5, attack, atk, true, false);
        else if (fire)
            dfs(u + 1, fire - 1, damage + 5, attack, atk, sf, true);
        card[2]++;
    }
    // 用 3
    if (card[3]) {
        card[3]--;
        card[q[pos++]]++;
        if (!sf)
            dfs(u + 1, fire, damage, attack + 2, atk, true, can_sf);
        else if (fire)
            dfs(u + 1, fire - 1, damage, attack + 2, atk, sf, can_sf);
        card[3]++;
        card[q[--pos]]--;
    }
    // 用 4
    if (card[4]) {
```

```
        card[4]--;
        card[q[pos++]]++;
        if (!sf)
            dfs(u + 1, fire + 1, damage, attack, atk, true, can_sf);
        else if (fire)
            dfs(u + 1, fire, damage, attack, atk, sf, can_sf);
        card[4]++;
        card[q[--pos]]--;
    }
    return;
}

int main()
{
    int T;
    cin >> T;
    while (T--) {
        cin >> n >> m;
        memset(card, 0, sizeof card);
        memset(q, 0, sizeof q);
        for (int i = 0; i < n; i++) {
            int x;
            cin >> x;
            card[x]++;
        }
        for (int i = 0; i < m; i++)
            cin >> q[i];
        pos = res = 0;
        dfs(0, 2, 0, 3, false, false, false);
        cout << res << endl;
    }
    return 0;
}
```

SSR

一个简单的概率计算，公式为：

$$C_n^{\frac{n}{2}} \cdot (\frac{1}{2})^n$$

除法用快速幂求逆元代替即可。

```cpp
# include <iostream>

using namespace std;

const int N = 1e6 + 5, mod = 1e9 + 7;
int F[N], G[N];

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

void init()
{
    F[0] = G[0] = 1;
    for (int i = 1; i < N; i++) {
        F[i] = 1ll * F[i - 1] * i % mod;
        G[i] = 1ll * G[i - 1] * quickpow(i, mod - 2) % mod;
    }
    return;
}

int main()
{
    init();
    int T;
    cin >> T;
    while (T--) {
        int n;
        cin >> n;
        int k = n / 2;
        cout << 1ll * F[n] * G[k] % mod * G[n - k] % mod *
quickpow(quickpow(2, mod - 2), n) % mod << endl;
    }
    return 0;
}
```

非对称加密算法RSA

题意写的很清楚，用拓展欧几里得可以求 d.

接着由于 e 很小，所以直接暴力求密文也可以 AC.（当然，快速幂也没错

```cpp
# include <iostream>

using namespace std;

int exgcd(int a, int b, int& x, int& y)
{
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

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

int violence(int a, int b, int mod)
{
    int res = 1;
    while (b--)
        res = 1ll * res * a % mod;
    return res;
}

int main()
{
    int p, q, m, e = 65537;
    cin >> p >> q >> m;
    int n = p * q, phi = (p - 1) * (q - 1);
    int x, y;
    exgcd(e, phi, x, y);
    cout << (x % phi + phi) % phi << endl << violence(m, e, n) << endl;
    return 0;
}
```

网络线路选择

所求方案数可以转化为：

$$所有方案总数 - 任意两个相邻线路的维修规格均不相同的方案数$$

即：

$$m^n - m \cdot (m-1)^{n-1}$$

用快速幂求得上述答案即可。此题唯一的坑就是，取模数 $n \cdot p$ 是个 10^{10} 级别的数，而两个此级别的数相乘会爆 $long\ long$，所以快速幂中相乘的这一步需要转换成快速乘来求，或者转化为 $\_\_int128$ 也可。

```
# include <iostream>
# include <cstdio>

using namespace std;

typedef long long LL;

LL quickmul(LL a, LL b, LL mod)
{
    LL res = 0;
    while (b) {
        if (b & 1)
            res = (res + a) % mod;
        a = 2 * a % mod;
        b >>= 1;
    }
    return res;
}

LL quickpow(LL a, LL b, LL mod)
{
    LL res = 1;
    while (b) {
        if (b & 1)
            res = quickmul(res, a, mod);
        a = quickmul(a, a, mod);
        b >>= 1;
    }
    return res;
}

int main()
{
    int T;
    scanf("%d", &T);
    while (T--) {
        LL n, m, p;
        scanf("%lld %lld %lld", &n, &m, &p);
        p = n * p;
        printf("%lld\n", ((quickpow(m, n, p) - m * quickpow(m - 1, n - 1,
p) % p) % p + p) % p);
    }
    return 0;
```

```
43 }
```

神奇的硬币 II

由题意易知，此题需要一个能够支持区间求最值，同时能够区间修改的数据结构，很自然就能想到用线段树来维护。

那么，此问题就转化为，如何建立线段树，如何将题目所给信息以区间的形式更新到线段树上?

首先，我们来观察某硬币的能量值能够加到物品上的条件：$硬币的价值 \geq 物品的重量$，由神奇的硬币 I 的启发，我们可以先将物品按重量排个序，然后就能快速找到满足上述条件的物品的边界。

假设对于某个硬币，满足上述条件的边界的物品下标为 i（即：$该硬币的价值 \geq w_j(1 \leq j \leq i)$，且 $i+1$ 这个物品不存在或者这个物品不满足这个条件），那么，我们就可以在物品重量为 w_1 ~ w_i 的这个区间的物品的能量值全都加上该硬币的能量值。类似的，我们可以对每一枚硬币都做一次上述操作。此处的寻找边界物品就不能像神奇的硬币 I 中那样直接一个循环枚举，否则将变为 $O(n^2)$ 的时间复杂度，此处可以用二分来找，同时，线段树的区间修改操作的时间复杂度也为 $O(logn)$，对于每个硬币都先二分一次，再区间修改一次，所以该部分操作的时间复杂度为 $O(nlogn)$.

由于我们需要对某段物品的重量的区间进行修改操作，所以我们只需要以此建立线段树即可。而此题中重量的数据范围非常大，因此我们还需要事先做一遍离散化，将离散化后的区间用 $O(nlogn)$ 的时间建成一棵线段树。

至此，我们就能够利用线段树求出最大能量值，而此题需要我们求的并不是这个最大值，而是能取到这个最大值的物品中下标的最小值。

这并不难实现，只需要在线段树的信息上多加一个取到当前最大值的最小下标即可，随着最大值的变化一起更新这个下标；同时，由于离散化之后，所处的下标会发生变化，因此，需要将某个物品重量的原下标也一起记录下来，相同重量的下标只取最小值，将此信息在建线段树的时候填入叶节点。

至于查询操作，我们需要找的是整段区间的能取到能量值的最大值的最小下标，即为线段树中表示整个区间的结点所维护的最小下标的值，直接 $O(1)$ 输出即可。

综上，利用 $离散化 + 二分 + 线段树$ 即可以 $O(nlogn)$ 的时间复杂度完美解决此题。

Update:

上述是对于每一枚硬币，考虑它对于某段物品重量区间的影响。

在比赛过程中，有位大佬用了更简洁的方法 AC 了此题，这里也就简单说一下：

对于每一个物品，考虑能够将能量加到这件物品上的硬币。可以发现，在排完序之后，这段硬币区间即为：以第一枚能够将能量融入这件物品的硬币为起点的整个后缀。

所以，只需要在排序之后处理一次后缀能量和，再来依次枚举每一件物品，二分找到区间起点，用后缀来更新答案即可。

感谢 $华中师范大学$ 一 胡婧 小姐姐第一个以此简洁方法过题。（大家感兴趣的可以去榜单上找这位小姐姐的代码欣赏，下面给出的是蒟蒻出题人的线段树，也请不要嫌弃~~）

```cpp
# include <iostream>
# include <cstdio>
# include <cstring>
# include <algorithm>

using namespace std;

typedef pair<int, int> PII;
const int N = 1e6 + 5;
int n, m, pos;
int val[N], E[N];
```

```
PII w[N], nums[N];

struct segment_tree {
    int l, r;
    int add, max_val, idx;
}T[N << 2];

void update(int p)
{
    int lp = p << 1, rp = lp | 1;
    if (T[lp].max_val == T[rp].max_val) {
        T[p].max_val = T[lp].max_val;
        T[p].idx = min(T[lp].idx, T[rp].idx);
    }
    else if (T[lp].max_val > T[rp].max_val) {
        T[p].max_val = T[lp].max_val;
        T[p].idx = T[lp].idx;
    }
    else {
        T[p].max_val = T[rp].max_val;
        T[p].idx = T[rp].idx;
    }
    return;
}

void push_down(int p)
{
    if (!T[p].add || T[p].l == T[p].r)
        return;
    int lp = p << 1, rp = lp | 1;
    T[lp].max_val += T[p].add;
    T[rp].max_val += T[p].add;
    T[lp].add += T[p].add;
    T[rp].add += T[p].add;
    T[p].add = 0;
    return;
}

void build(int p, int l, int r)
{
    T[p].l = l, T[p].r = r;
    if (l == r) {
        T[p].max_val = 0;
        T[p].idx = nums[l].second;
        return;
    }
    int mid = l + r >> 1, lp = p << 1, rp = lp | 1;
    build(lp, l, mid);
    build(rp, mid + 1, r);
    update(p);
    return;
}

void change(int p, int l, int r, int c)
{
    if (l <= T[p].l && r >= T[p].r) {
        T[p].max_val += c;
        T[p].add += c;
```

```
        return;
    }
    push_down(p);
    int mid = T[p].l + T[p].r >> 1, lp = p << 1, rp = lp | 1;
    if (l <= mid)
        change(lp, l, r, c);
    if (r > mid)
        change(rp, l, r, c);
    update(p);
    return;
}

int get_idx(int val)
{
    int l = 0, r = pos;
    while (l < r) {
        int mid = l + r + 1 >> 1;
        if (nums[mid].first <= val)
            l = mid;
        else
            r = mid - 1;
    }
    return l;
}

int main()
{
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", val + i);
    for (int i = 1; i <= n; i++)
        scanf("%d", E + i);
    for (int i = 1; i <= m; i++) {
        scanf("%d", &w[i].first);
        w[i].second = i;
    }
    sort(w + 1, w + m + 1);
    nums[0].first = -2e9;
    for (int i = 1; i <= m; ) {
        nums[++pos] = w[i];
        int j = i;
        while (j <= m && w[i].first == w[j].first)
            j++;
        i = j;
    }
    build(1, 1, pos);
    for (int i = 1; i <= n; i++) {
        int p = get_idx(val[i]);
        if (!p)
            continue;
        change(1, 1, p, E[i]);
    }
    printf("%d\n", T[1].idx);
    return 0;
}
```

神奇的硬币 I

对于一枚硬币的价值 val，它所能购买的物品是所有物品中满足 $val \geq w_i (1 \leq i \leq m)$ 条件的。由于题目最终问的是数量，那么我们如何快速求出这个满足这个条件的数量？显然可以先将所有物品排个序，找到最后一个满足该条件的下标，就可以知道满足该条件的数量。

那么，如何求最受欢迎的物品？

对于任意两枚硬币 i, j，它们的价值分别为 val_i, val_j，若 $val_i \leq val_j$，则第 i 枚硬币能够购买的物品，第 j 枚硬币也一定可以购买。此时，第 i 枚硬币所能购买的物品的购买次数，就一定比只能用第 j 枚所能购买的物品的购买次数多。

因此，我们可以将硬币按价值也排个序，然后从左到右枚举每一枚硬币，对于第一枚能够购买物品的硬币，它所能购买的物品数量即为最终答案。

对于第一枚能够购买物品的硬币，如何找到最后一个满足购买要求的下标？当然可以用二分，也可以直接写个循环枚举，由于这个部分只会执行一次，所有这部分看似有两重循环，但时间复杂度仍然是 $O(n)$.

由于事先有排序操作，所以整个代码的时间复杂度为 $O(nlogn)$.

```cpp
# include <iostream>
# include <cstdio>
# include <algorithm>

using namespace std;

const int N = 1e6 + 5;
int n, m;
int val[N], w[N];

int main()
{
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", val + i);
    for (int i = 1; i <= m; i++)
        scanf("%d", w + i);
    sort(val + 1, val + n + 1);
    sort(w + 1, w + m + 1);
    int res = -1;
    for (int i = 1; i <= n; i++)
        if (val[i] >= w[1]) {
            int j = 1;
            while (j <= m && val[i] >= w[j])
                j++;
            res = j - 1;
            break;
        }
    printf("%d\n", res);
    return 0;
}
```