概述

	A. Orange的管道	B. 异色边	C. 建筑师	D. 你也想品尝俄式简餐吗?	E. Orange的阶乘	F. Orange的管道II
题目参考难度	1300	1300	1400	1900	1600	1800
涉及知识点	前缀和，双指针	构造，模拟，贪心，分类讨论	差分	DFS/BFS	数学，枚举，字符串	动态规划，前缀和，双指针，背包DP

> 也许题目很难？也许也没那么难？

题解报告

tips:如果有不懂的地方可以在评论区留言.

A. Orange的管道

不难发现，我们小球的质量一定为正数（~~这不是废话吗~~），因此最优的情况一定是拿走 k 个小球，那么从反方向的角度考虑，管道里面一定还剩下 $n-k$ 个小球，且**他们一定是连续的**（因为我们每次都是从两端拿），因为我们转换思路，发现连续的长度为 $n-k$ 的区间不超过 k 个，我们利用前缀和可以快速回答这一段连续区间的和，最后枚举所有的区间，取最小即可。

$$ans = \min_{i=1}^{n-k+1} pre_{i+k-1} - pre_{i-1}$$

```cpp
using i64 = long long;
const int N = 100010;
i64 pre[N];
int main()
{
    int n, k;
    cin >> n >> k;
    k = n - k;
    for(int i = 1; i <= n; i ++)
    {
        cin >> pre[i];
        pre[i] += pre[i-1];
    }
    i64 ans = 1e18;
    for(int i = 1; i + k - 1 <= n; i ++)
        ans = min(ans, pre[i + k - 1] - pre[i - 1]);
    cout << ans << '\n';
}
```

B. 异色边

我们记 R 为红色点的集合，B 为蓝色点的集合。

本题实际上主要考察贪心和分类讨论，我们首先考虑特殊的情况：

1. 若 $|R| = 0$ 或者 $|B| = 0$，那么答案就是0,我们只要把剩下的红色点或者蓝色点首尾相接即可，这样刚好用完 n 条边。
2. 如果 $|R| = 1$ 或者 $|B| = 1$，那么根据贪心的思想，我们假设 $|R| = 1$，那么我们只需要把所有的蓝色点全部连接到红色点上即可，这样会花费 $n-1$ 条边，最后剩下的一条边随意连接两个蓝色点即可。答案为 $n-1$。
3. 如果 $|R| > 1$ 且 $|B| > 1$，那么答案一定为 n，所有的边都可以成为异色边，考虑如下连接方案：

- 红色的蓝色的点交错连接，若点数相同，最后构成的链首尾一定异色，若一方的点数多于另一方(假设 $|R| > |B|$)，那么我们先取 $min(|R|, |B|)$ 个点构成一条环，再把剩下的 R 中的点随便找一个蓝色的点连接起来就好。

> ps：我的构造方法事实上还算比较复杂的，本题有更加简单的构造方法，可以自行思考

```
void solve()
{
    int n;
    cin >> n;
    string colors;
    cin >> colors;
    colors = '*' + colors;
    vector<int> red, blue;
    for (int i = 1; i <= n; i ++)
    {
        if(colors[i] == '1') red.push_back(i);
        else blue.push_back(i);
    }
    if(red.size() == 0 || blue.size() == 0)
    {
        if(red.size() > 0) swap(red, blue);
        for(int i = 0; i < n; i ++)
        {
            cout << blue[i] << ' ' << blue[(i + 1) % blue.size()] << '\n';
        }
        return;
    }
    if(red.size() == 1 || blue.size() == 1)
    {
        if(red.size() > blue.size()) swap(red, blue);
        for(int i = 0; i < blue.size(); i ++)
        {
            cout << blue[i] << ' ' << red[0] << '\n';
        }
        cout << blue.front() << ' ' << blue.back() << '\n';
        return;
    }
    if(red.size() != blue.size())
    {
        if(red.size() < blue.size()) swap(red, blue);
        for(int i = 0; i < blue.size(); i ++)
        {
            cout << red[i] << ' ' << blue[i] << '\n';
            cout << blue[i] << ' ' << red[i + 1] << '\n';
        }
        for(int i = blue.size(); i < red.size(); i ++)
        {
            cout << red[i] << ' ' << blue[0] << '\n';
        }
    }
    else
    {
        for(int i = 0; i < blue.size(); i ++)
        {
            cout << red[i] << ' ' << blue[i] << '\n';
            cout << blue[i] << ' ' << red[(i + 1) % red.size()] << '\n';
        }
    }
}
```

C. 建筑师

抽象一下题意，不难发现题意就是：
每次操作可以选择 $[a_l, a_r]$ 区间 $+1$，为至少多少次之后能得到 a_i。

看到区间加数，很容易想到差分算法，我们说过差分的本质就是选择差分数组的一个点 $+1$ 一个点 -1，我们钦定差分数组为 $d_i = a_i - a_{i-1}$，特别的规定 $a_0 = 0$。
那么一开始差分数组均为 0，我们每次操作相当于给 $d_l + 1$，$d_{r+1} - 1$，那么我们对原数组差分之后，要从 0 数组开始构造出 d_i，每次选择一个点 $+1$ 另一个点 -1，我们只需要每次选择 $d_i > 0$ 的位置 $+1$，$d_i < 0$ 的位置 -1，这样就是最优的方案，由于一开始 $d_i = 0$，因此在若干次操作之后一定有 d_i 所有的正数绝对值之和等于负数绝对值之和，所以答案就是 d_i 中正数或者负数绝对值之和。

$$ans = \sum_{i}^{n}[d_i > 0] \times d_i$$

```
using i64 = long long;
signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<int> a(n + 1), d(n + 1);
    for(int i = 1; i <= n; i ++)
    {
        cin >> a[i];
        d[i] = a[i] - a[i - 1];
    }
    i64 ans = 0;
    for(int i = 1; i <= n; i ++)
        ans += d[i] > 0 ? d[i] : 0;
    cout << ans << '\n';
    return 0;
}
```

D. 你也想品尝俄式简餐吗?

本题实际上是要去验证给定网格图中所有数字相同的连通块是否为 L 形或者 I 型。

我们这里定义一个连通块的 对边连通 ，当前仅当连通块满足其上下均存在连通块且左右不存在连通块或者相反，如下图所示：

```
1.      2.  #
# # #       #
            #
```

定义 邻边连通 ，当且仅当 # 中存在相邻的两条边与另一个 # 连通，如下图所示：

```
1. #   2. # #  3.   #  4. # #
   # #      #     # #     #
```

我们观察到，在 L 型的连通块中，一定存在一个 对边连通 和一个 邻边连通 ，在 I 型的连通块中，一定存在两个 对边连通 ，因此我们只要统计一个连通块所包含的 对边连通 和 邻边连通 的数量即可，若 L 型连通块包含一个 对边连通 和一个 邻边连通 ，I 型连通块包含两个 对边连通 ，那么这个连通块就是合法的。同时还要注意一个连通块大小一定为 4，如果超过或者不足 4 则直接返回 $False$。

更加细节的实现请看代码。

```cpp
// 偏移向量
int dx[] = {-1, 0, 1, 0};
int dy[] = {0, 1, 0, -1};


// bfs搜索函数，每次对图 graph 中以 (x, y) 为起点的连通块进行搜索，将结果存储在 set S 中
void bfs(vector<vector<int>> &graph, set<array<int, 2>> &s, int n,int m,int x,int y)
{
    queue<array<int, 2>> q;
    map<int, map<int,int>> vis;
    q.push({x, y});
    s.insert({x, y});
    while(q.size())
    {
        auto [nx, ny] = q.front();
        q.pop();
        for(int i = 0; i < 4; i ++)
        {
            int px = nx + dx[i];
            int py = ny + dy[i];
            if(px < 1 or py < 1 or px > n or py > m or vis[px][py] or graph[px][py] != graph[nx][ny]) continue;
            s.insert({px, py});
            vis[px][py] = 1;
            q.push({px, py});
        }
    }
}

bool check()
{
    int n;
    int m;
    cin >> n >> m;
    int max_idx = n * m / 4;
    auto mat = matrix(n + 1, m + 1, 0);
    bool flag = 1;
    for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= m; j ++)
        {
            cin >> mat[i][j];
            // 如果数字发生越界，可直接提前返回。
            if(mat[i][j] > max_idx or mat[i][j] < 1) flag = 0;
        }
    if(!flag) return 0;
    // 记录一个网格是否被搜索过
    auto vis = matrix(n + 1, m + 1, 0);
    for(int i = 1; i <= n; i ++)
        for(int j = 1; j <= m; j ++)
        {
            set<array<int, 2>> S;
            if(vis[i][j]) continue;
            bfs(mat, S, n, m, i, j);
            if(S.size() != 4) return 0;
            // A : 对边连通 B : 邻边连通
            int cntA = 0, cntB = 0;
            for(auto [x, y] : S)
            {
                vis[x][y] = 1;
                // 如果是对边连通，A++
                if(S.count({x + 1, y}) and S.count({x - 1, y}) or S.count({x, y + 1}) and S.count({x, y - 1}))
                    cntA ++;
                // 如果是邻边连通，B++
                for (int i = 0; i < 4; i ++)
                {
                    int px1 = x + dx[i], py1 = y + dy[i];
                    int px2 = x + dx[(i + 1) % 4], py2 = y + dy[(i + 1) % 4];
                    if(S.count({px1, py1}) and S.count({px2, py2}))
                        cntB++;
```

```
                }
            }
            // 判断形状，如果形状不符合，直接返回 False.
            if(!(cntA == 2 or (cntA == 1 and cntB == 1))) return 0;
        }
    return 1;
}

signed main(int argc, char** argv)
{
    cin.tie(0) -> sync_with_stdio(0);
    int T;
    cin >> T;
    while (T--)
    {
        if(check()) cout << "Accepted\n";
        else cout << "Wrong Answer\n";
    }
    return 0;
}
```

E.Orange的阶乘

很经典的题型，我们可以预处理出 1 到 5×10^6 中所有的阶乘以及 n?，在预处理过程中，我们发现 k 不超过 9，且要舍弃掉末尾所有的 0，因此我们事实上题目的意思就是相当于答案如果末尾是 0 (即为 10 的倍数)，我们就先除以 10 ,直到不是 10 的倍数，再对 10^8 取mod。我们预处理出所有的答案，每次 $O(1)$ 查询即可。

```cpp
#define int long long
const int N = 5e6 + 10, mod = 1e8;

// fact[i]表示 i! qfact[i] 表示 \prod_i^n{i!}
int fact[N], qfact[N];

void solve()
{
    int n, k;
    cin >> n >> k;
    string s = to_string(qfact[n]);
    while (s.size() < k)
        s = '0' + s;
    for (int i = s.size() - k; i < s.size(); i++)
        cout << s[i];
    cout << '\n';
}

signed main()
{
    IOS;
    fact[0] = qfact[0] = 1;
    for (int i = 1; i < N; i++)
    {
        qfact[i] = 1;
        fact[i] = fact[i - 1] * i;
        while (fact[i] % 10 == 0)
            fact[i] /= 10;
        fact[i] %= mod;
        qfact[i] = qfact[i - 1] * fact[i];
        while (qfact[i] % 10 == 0)
            qfact[i] /= 10;
        qfact[i] %= mod;
    }
    int _;
    cin >> _;
    while (_--)
        solve();
    return 0;
}
```

F. Orange的管道II

根据第A题，我们知道，每个一定是取满 k 个球才是最优的，现在本题管道数量从 1 变成了 n 个。我们需要找到一种最优的方案，使得管道剩余的质量最小/最大化拿出小球的质量。

我们可以发现，对于第 i 个管道，我们取 1 到 $\min(c_i, k)$ 中任意个数的小球都可以通过A题双指针的方法预处理出来。我们假设对于第 i 个管道，取出 j 个球，取出的最大价值我们可以用 $f_{i,j}$ 表示。换种角度看待，这可以看作第 i 个管道为一个分组背包问题的分组，里面包含 $\min(c_i, k)$ 个物品，每个物品的体积为 j，价值为 $f_{i,j}$，最后发现我们利用 A 中的方法处理所有的管道，即问题就转化成了一个 分组背包问题 。跑一次分组背包即可。

复杂度： $O(nk^2)$

```cpp
int dp[105][10005];
// 预处理的物品
int pak[105][105];

signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    vector<vector<int>> a(n + 1);
    vector<int> c(n + 1);
    for(int i = 1; i <= n; i ++)
    {
        int t;
        cin >> t;
        c[i] = t;
        a[i].assign(t + 1, 0);
        for(int j = 1; j <= t; j ++) cin >> a[i][j], a[i][j] += a[i][j - 1];
        int sum = a[i][t];
        for (int j = 0; j <= t; j++)
        {
            int res = 0;
            for (int k = 1; k + j - 1 <= t; k++)
            {
                res = max(res, sum - (a[i][k + j - 1] - a[i][k - 1]));
            }
            pak[i][t - j] = res;
        }
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= m; j++)
        {
            for (int k = 0; k <= min(j, c[i]); k++)
            {
                dp[i][j] = max(dp[i][j], dp[i - 1][j - k] + pak[i][k]);
            }
        }
    }
    cout << dp[n][m];
    return 0;
}
```