A 小红的博弈

如果小红可以一次拿完，就是小红赢，否则小紫赢。

```
void solve() {
    int n;
    cin >> n;
    if (n <= 2) cout << "red" << "\n";
    else cout << "purple" << "\n";
}
```

Fence 1

B 小红选点

n 只有 1000，暴力即可。

```
void solve() {
    int n;
    cin >> n;
    vector<PII> a(n);
    PII x, y;
    for (int i = 0; i < n; ++i) cin >> a[i].fi >> a[i].se;
    int ans = -1;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            PII u = a[i], v = a[j];
            int dis = (u.fi - v.fi) * (u.fi - v.fi) + (u.se - v.se) * (u.se - v.se);
            if (dis > ans) {
                ans = dis;
                x = u, y = v;
            }
        }
    }
    cout << x.fi << " " << x.se << " " << y.fi << " " << y.se << "\n";
}
```

Fence 2

C 小红的矩形

如果横/纵坐标相等就往纵/横方向延伸出一格，否则取 (x_1, y_2) 和 (x_2, y_1) 即可。

```
void solve() {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    if (x1 == x2) {
        cout << x1 + 1 << " " << y1 << " " << x2 + 1 << " " << y2 << "\n";
    } else if (y1 == y2) {
        cout << x1 << " " << y1 + 1 << " " << x2 << " " << y2 + 1 << "\n";
    } else {
        cout << x1 << " " << y2 << " " << x2 << " " << y1 << "\n";
    }
}
```

Fence 3

D 小红拿石子1.0

对于小红来说，所有剩下堆的“已被紫方拿掉的量”都是相同的（都是 d ），所以当前拿走石子最多的堆即是最优的。

```
void solve() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) cin >> a[i];
    sort(all(a), [&](int a, int b) {
        return a > b;
    });
    int d = 0;
    ll ans = 0;
    for (int i = 0; i < n; ++i) {
        if (a[i] <= d) continue;
        ans += a[i] - d;
        d++;
    }
    cout << ans << "\n";
}
```

Fence 4

E 小红玩树

记 $db[u]$ 为从紫方起点 b 到 u 的最短边数。

对红方从 a 出发的一条路径 $a = p_0, p_1, p_2, \ldots, p_t = leaf$（红方第 i 步到达 p_i）：

- 红方一旦走到叶子 p_t，立即获胜，所以只要红能在到达前不被捕获就能赢。
- 在红方到达中间点 p_i（$i < t$）之后，紫方会进行其第 i 次回合并可走 2 步，因此若 $db[p_i] <= 2 * i$，紫可以在该回合到达 p_i 并抓住红（紫胜）。
  因此要继续向下走到更远的节点，必须保证对到达的中间点 p_i 有 $db[p_i] > 2 * i$。
- 对叶子 p_t，红在第 t 步到达并立即获胜；紫在那之前只进行了 $t-1$ 次回合，紫能否在红到达前已占据该叶子由 $db[p_t] <= 2 * (t-1)$ 判断。 因此只要 $db[p_t] > 2 * (t-1)$，红到达该叶子时紫尚未占据，它就获胜。

```cpp
void solve() {
    int n;
    cin >> n;
    int a, b;
    cin >> a >> b;
    vector<vector<int>> g(n + 1);
    for (int i = 1; i < n; ++i) {
        int u, v;
        cin >> u >> v;
        g[u].pb(v);
        g[v].pb(u);
    }

    vector<int> db(n + 1, inf);
    queue<int> q;

    db[b] = 0;
    q.push(b);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : g[u]) {
            if (db[v] == inf) {
                db[v] = db[u] + 1;
                q.push(v);
            }
        }
    }

    if (g[a].size() <= 1) {
        cout << "red" << "\n";
        return;
    }

    vector<bool> vis(n);
    queue<PII> qu;
    vis[a] = 1;
    qu.push({a, 0});
    bool f = false;
```

```
40
41      while (!qu.empty() && !f) {
42          auto [u, d] = qu.front();
43          qu.pop();
44          for (int v : g[u]) {
45              if (vis[v]) continue;
46              if (g[v].size() == 1) {
47                  if (db[v] > 2 * d) {
48                      f = true;
49                      break;
50                  } else {
51                      vis[v] = 1;
52                      continue;
53                  }
54              }
55              if (db[v] > 2 * (d + 1)) {
56                  vis[v] = 1;
57                  qu.push({v, d + 1});
58              } else {
59                  vis[v] = 1;
60              }
61          }
62      }
63
64      if (f) cout << "red" << "\n";
65      else cout << "purple" << "\n";
66  }
```

Fence 5

F 小红拿石子2.0

打表看出来的：

- 记 $mx = max(a)$，$cnt =$ 数组中等于mx的堆数
- 若 $cnt >= mx + 1$，则小紫获胜；否则小红获胜

证明如下：

- 如果把石子堆画成柱状图，相当于小红每次可以移走一列，小紫每次可以移走最底下一层。
- 如果 $cnt > mx$ 小紫必胜：
  - 任一初始高度不超过 mx 的列，若没有被小红在之前某次彻底删除，则在第 mx 次小紫的动作后会被减到 0 。
  - 小红在这 mx 轮内至多可以删除 mx 列。
  - 但最开始就有 $cnt > mx$ 列高度为 mx 。小红在 mx 次行动中最多能把其中 mx 列删掉，仍至少留下一列初始为 mx 的列没有被小红删除；该列在第 mx 次小紫的操作后会被减到 0 。再考虑任意其它初始高度 $< mx$ 的列，要么被小红删掉，要么也在第 mx 次小紫操作之前被减到 0 。于是在第 mx 次小紫操作结束时，所有列都已为空。
  - 故小紫必胜。
- 如果 $cnt \leq mx$ 小红必胜：
  - 小红的第一步按两种情况选一堆删除：

    1. 若 $cnt = mx$ ： 小红删除一堆高度为 mx 的堆。
    2. 若 $cnt < mx$ ：
       - 若存在高度 $< mx$ 的堆，小红删除任意这样一堆；
       - 否则所有堆都是高度 mx（此时 $cnt = n < mx$ ），小红随便删除一堆。

    然后小紫把剩余所有堆高度都减 1 。
  - 记“红删 + 紫减 1 ”后的新最大高度为 $mx\prime$ 及新最大高度的堆数为 $cnt\prime$ ，然后我们证明：
    - $mx' \leq mx - 1$:
      - 无论小红删哪堆，小紫把所有剩余堆减 1 后，原来高度为 k 的堆变为高度 $k-1$ 。因此新的最大高度 $mx' \leq mx - 1$ 。（若小红没有删掉所有原来高度为 mx 的堆，则 $mx\prime = mx - 1$；若小红删掉了所有原来为 mx 的堆，则 $mx' < mx - 1$ 。总之 $mx' \leq m - 1$ 。）
    - $cnt' \leq mx'$:
      - 若原来 $cnt = mx$ ： 小红删一个高度为 mx 的堆，剩下高度为 mx 的堆数变为 $mx - 1$ 。减 1 后它们变为高度 $mx - 1$，于是新的最大高度 $mx' = mx - 1$ 且 $cnt\prime = mx - 1 = mx\prime$ 。所以 $cnt' \leq m'$ 成立。
  - 基底：若 $mx = 1$ ，且 $c \leq 1$ ，小红第一步删掉唯一的 1（若存在），小紫回合无子可取，红胜成验证。
  - 归纳：由上面的构造，若命题在所有最大高度 $< mx$ 时成立，那么在高度 mx 且 $c \leq m$ 时，小红能做一步把局面变为高度 $mx\prime < mx$ 且仍满足条件，按归纳假设，小红必胜。
- 证毕

```cpp
void solve() {
    int n, mx = 0;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) cin >> a[i], mx = max(mx, a[i]);

    int cnt = 0;
    for (int i = 0; i < n; ++i) cnt += a[i] == mx;

    if (cnt >= mx + 1) cout << "purple" << "\n";
    else cout << "red" << "\n";
}
```

Fence 6

头文件

```cpp
//Anoth3r
#include<bits/stdc++.h>
#include<bits/extc++.h>
#define pb push_back
#define eb emplace_back
#define fi first
#define se second
#define all(a) a.begin(), a.end()
#define rall(a) a.rbegin(), a.rend()
using namespace std;

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef __int128 i128;
typedef pair<int, int> PII;
typedef pair<ll, ll> PLL;
typedef tuple<ll, ll, ll> TLLL;
typedef __gnu_pbds::tree<PLL, __gnu_pbds::null_type, less<PLL>, __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update> Tree;
// typedef __gnu_pbds::tree<ll, __gnu_pbds::null_type, less<ll>, __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update> Tree;

constexpr int inf = (ll)1e9 + 7;
constexpr ll INF = (ll)2e18 + 9;
// constexpr ll INF = (ll)4e18;
// constexpr ll MOD = 1e9 + 7;
constexpr ll MOD = 998244353;
constexpr ld PI = acos(-1.0);
constexpr ld eps = 1e-10;

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
ull randint(ull l, ull r) {uniform_int_distribution<unsigned long long> dist(l, r); return dist(rng);}

void init() {

}

void solve() {

}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    init();

    int t = 1;
```

```
    // cin >> t;
    for (int _ = 1; _ <= t; ++_) {
        solve();
    }
    return 0;
}
```

Fence 7