

计蒜客信息学 2023 年 2 月提高组模拟赛题解

题目	知识点
蒜头君的抽象画	简单图论、连通性、并查集、环
拍照留念	动态规划
蒜头君的困扰	高精度、快速幂、扩展欧拉定理、逆元
蒜头君的选择	图论、Tarjan、主席树优化建图

蒜头君的抽象画

题解

因为合法的图中只有一个环，且其余部分均为树，并且图连通。则合法的图一定满足：

- 点数等于边数（ $n = m$ ）；
- 图连通；
- 环上结点的个数 ≥ 5 。

因此可以通过并查集或搜索判连通，再求简单环上结点的数量即可。

标程

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1010;
int n, m, len;
int ph[maxn], vis[maxn], f[maxn];
vector<int> G[maxn];
void init() {
    len = 0;
    for (int i = 1; i < maxn; i++) {
        f[i] = i;
        vis[i] = 0;
        G[i].clear();
    }
}
int getf(int x) {
    if (f[x] == x) {
        return x;
    }
    return f[x] = getf(f[x]);
}
void merge(int x, int y) {
    x = getf(x), y = getf(y);
    if (x != y) {
        f[y] = x;
    }
}
void dfs(int u, int fa, int cnt) {
    if (!len) {
        return;
    }
    if (vis[u]) {
        for (int i = cnt - 1; i >= 0; i--) {
            len++;
            if (ph[i] == u) {
                break;
            }
        }
        return;
    }
    ph[cnt] = u;
    vis[u] = 1;
    for (int i = 0; i < G[u].size(); i++) {
        int v = G[u][i];
        if (v != fa) {
            dfs(v, u, cnt + 1);
        }
    }
}
bool solve() {
    if (n != m) {
        return false;
    }
    for (int i = 2; i <= n; i++) {
        if (getf(i) != getf(1)) {
            return false;
        }
    }
    dfs(1, -1, 0);
    if (len < 5) {
        return false;
    }
    return true;
}
int main() {
    freopen("draw.in", "r", stdin);
    freopen("draw.out", "w", stdout);
    int T;
    cin >> T;
    while (T--) {
        init();
        cin >> n >> m;
        for (int i = 1; i <= m; i++) {
            int u, v;
            cin >> u >> v;
```

```
        merge(u, v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    if (solve()) {
        cout << "perfect" << endl;
    } else {
        cout << "sorry" << endl;
    }
}
return 0;
}
```

拍照留念

题解

题目类型：贪心，动态规划

首先注意到，如果连续两个人的奖品加起来超过 m ，则方案肯定不存在。否则，把环断开为链，需要考虑到第一个人和最后一个人不能有重复的。所以贪心地做，需要让倒数第二个人和第一个人重复的奖品尽量多。这提示我们使用动态规划，用 $f[i]$ 、 $g[i]$ 表示第 i 个人**最多 / 最少**与第一个人分到多少重复的奖品数量。

用 $f[i-1]$ 更新 $g[i]$ ，用 $g[i-1]$ 更新 $f[i]$ ，最后判断 $g[n]$ 是否大于 0 即可。

标程

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 200010;
int c[maxn], g[maxn], f[maxn];
int main() {
    freopen("photograph.in", "r", stdin);
    freopen("photograph.out", "w", stdout);
    int T;
    scanf("%d", &T);
    while (T--) {
        bool flag = true;
        int n, m;
        scanf("%d %d", &n, &m);
        for (int i = 1; i <= n; i++) {
            scanf("%d", &c[i]);
        }
        if (c[1] > m) {
            flag = false;
        }
        f[1] = c[1];
        g[1] = c[1];
        for (int i = 2; i <= n; i++) {
            if ((c[i] + c[i - 1]) > m) {
                flag = false;
            } else {
                f[i] = min(c[i], c[1] - g[i - 1]);
                g[i] = max(0, c[i] - ((m - c[i - 1]) - (c[1] - f[i - 1])));
            }
        }
        if (g[n] > 0) {
            flag = false;
        }
        if (flag) {
            printf("%d\n", 1);
        } else {
            printf("%d\n", -1);
        }
    }
    return 0;
}
```

蒜头君的困扰

题解

算法零

把题面里的程序加个输入输出交上去，期望得分 0 分。

算法一

思考一下 `solve` 函数的意义：求区间 $[l, r]$ 中与 p 互质的数乘积模 p 意义下的值，而题面里的内层循环实质上就是在判 p 和 j 是否互质。用 `gcd` 把这一步复杂度优化到 $\log$ ，期望得分 20 分。

复杂度： $\mathcal{O}(r \log p)$

算法二

在模 p 意义下， $[l, r]$ 区间中每 p 个数为一个循环节，其答案相同。因此我们处理出 $1 \sim p$ 答案的前缀/后缀积，用于计算 $[1, k_1p)$ 和 $[k_2p, r]$ 。而 $[k_1p, k_2p)$ 的部分是 $1 \sim p$ 答案 $k_2 - k_1$ 的次幂。

如果你没有使用快速幂，期望得分是 30 分。

如果你使用快速幂，期望得分是 60 分。

复杂度： $\mathcal{O}(p \log p + \log_p^x)$

算法三

在 l, r 很大的情况下，用高精度模拟上述过程。

如果你直接按照高精除低精写快速幂里的部分，期望得分 80 分。

复杂度： $\mathcal{O}(\lg \frac{r}{p} + \log \frac{r}{p} + p \log p)$

如果把除以 2 写成高精位运算，期望得分 100 分。

复杂度： $\mathcal{O}(\lg \frac{r}{p} + \log \frac{r}{p} + p \log p)$

算法四

预处理时顺便把 φ § 算出来。运用扩展欧拉定理，次数对 φ § 取模，就可以在算法二基础上（连快速幂都不需要）AC 本题。期望得分 100 分。

复杂度： $\mathcal{O}(\lg r + p \log p)$

算法五

对于 $[k_1p, k_2p)$ 的部分，只有模 p 意义下的逆元等于该数本身的数才有贡献。因此我们只需要考虑这些数的乘积，以及中间整段数的奇偶性。期望得分 100 分。复杂度： $\mathcal{O}(\lg r + p \log p)$

标程

```
#include <bits/stdc++.h>
#define N 1000005
using namespace std;
typedef long long LL;
const int V = 100000000;
int p, ph, L, R, eq;
LL ans = 1, s = 1;
char S[N];
int gcd(int x, int y) {
    return x && y ? gcd(y, x % y) : x + y;
}
struct big {
    int d;
    LL a[N >> 2];
    void rd() {
        scanf("%s", S), d = strlen(S);
        for (int i = 1; i <= d; i++) {
            a[d - i >> 3] = a[d - i >> 3] * 10 + S[i - 1] - '0';
        }
        d = d - 1 >> 3;
    }
    void upd() {
        while (d && !a[d]) {
            d--;
        }
    }
    void div() {
        for (int i = d; i >= 0; i--) {
            i ? a[i - 1] += a[i] % p * V : 0, a[i] /= p;
        }
        upd();
    }
    int mod(int k) {
        LL ret = 0;
        for (int i = d; i >= 0; i--) {
            ret = (ret * V + a[i]) % k;
        }
        return ret;
    }
} l, r, ll, rr, tmp;
int main() {
    freopen("challenge.in", "r", stdin);
    freopen("challenge.out", "w", stdout);
    ll.rd(), rr.rd(), scanf("%d", &p);
    l = ll, r = rr, L = l.mod(p), R = r.mod(p);
    l.div(), r.div();
    if (L) {
        l.a[0]++;
        for (int i = 0; l.a[i] >= V; i++) {
            l.a[i + 1]++, l.a[i] -= V;
        }
        l.d += !!l.a[l.d + 1];
    }
    for (int i = r.d; i >= 0 && l.a[i] >= r.a[i] && !eq; i++) {
        eq |= l.a[i] > r.a[i];
    }
    if (eq) {
        R += L > R ? p : 0;
        for (int i = L; i <= R; i++) {
            i && gcd(i, p) < 2 ? ans = ans * i % p : 0;
        }
    } else {
        for (int i = 1; i < p; i++) {
            gcd(i, p) < 2 ? ph++, s = s * i % p, i <= R ? ans = ans * i % p : 0, i >= L ? ans = ans * i % p : 0 : 0;
        }
        for (int i = 0; i <= r.d; i++) {
            r.a[i] -= l.a[i], r.a[i] < 0 ? r.a[i + 1]--, r.a[i] += V : 0;
        }
        r.upd(), ph = r.mod(ph);
        while (ph--) {
            ans = ans * s % p;
        }
    }
    printf("%lld\n", ans);
}
```

```
    return 0;  
}
```

蒜头君的选择

题解

题意可以转换为一个图论模型，将员工抽象为结点，图中共有 $N + \text{事件 1 的数量}$ 个点。

- 对于事件 1：新增一个点，替换到第 x 个工位上。
- 对于事件 2，由 x 向工位区间 $l \sim r$ 的员工结点连接一条有向边（注意结点会根据事件 1 变化）。

每种方案的厌烦值之和为：从某一个结点出发，沿着有向边可以走到的所有结点的厌烦值之和。

因此 30% 的解法：建图后直接 `dfs` 搜索即可。

另外 30% 的解法：因为图中可能存在大量的环，可以使用 `Tarjan` 缩点。

满分解法：因为朴素建图就会超时，向区间连边可以想到用主席树优化。每次修改在原来的基础上新增一条链，父亲向儿子连边。每次规定用当前最后一个在位置 x 的点编号向区间连边。复杂度 $\mathcal{O}(n \log n)$ 。

【注意】

1. 每次新建链都需要向左右儿子连边，不要连向 0。
2. 边数组开 3 倍。
3. 有向图的 `Tarjan` 算法不用判断反向边。

标程

```
#include <cstdio>
#include <algorithm>
#define ll long long
using namespace std;
bool isdigit(char c) {
    return c >= '0' && c <= '9';
}
int read() {
    int s = 0, t = 1;
    char c;
    while (!isdigit(c = getchar())) {
        if (c == '-') {
            t = -1;
        }
    }
    do {
        s = s * 10 + c - '0';
    } while (isdigit(c = getchar()));
    return s * t;
}
const int maxn = 100010, N = 1800010;
int dfn[N], low[N], st[N], dfsnum, cnt, tot, first[N], col[N], sz, v[maxn], top, n, m, rt, ou[N];
ll num[N];
struct edge {
    int u, v, from;
} e[N * 3]; // three
void ins(int u, int v) {
    tot++;
    e[tot].u = u;
    e[tot].v = v;
    e[tot].from = first[u];
    first[u] = tot;
}
struct tree {
    int l, r, s;
} t[N];
void insert(int& x, int y, int l, int r, int k, int p) {
    x = ++sz;
    t[x] = t[y];
    if (l == r) {
        t[x].s = p;
        v[k] = x;
        return;
    }
    int mid = (l + r) >> 1;
    if (k <= mid) {
        insert(t[x].l, t[y].l, l, mid, k, p);
    } else {
        insert(t[x].r, t[y].r, mid + 1, r, k, p);
    }
    if (t[x].l) {
        ins(x, t[x].l);
    }
    if (t[x].r) {
        ins(x, t[x].r);
    }
}
void tarjan(int x) {
    dfn[x] = low[x] = ++dfsnum;
    st[++top] = x;
    for (int i = first[x]; i; i = e[i].from) { // fan???
        if (!dfn[e[i].v]) {
            tarjan(e[i].v);
            low[x] = min(low[x], low[e[i].v]);
        } else if (!col[e[i].v]) {
            low[x] = min(low[x], dfn[e[i].v]);
        }
    }
    if (dfn[x] == low[x]) {
        cnt++;
        while (st[top] != x) {
            col[st[top--]] = cnt;
        }
        col[st[top--]] = cnt;
    }
}
```

```

    }
}
void modify(int k, int l, int r, int L, int R, int x) {
    if (L <= l && r <= R) {
        return void(ins(x, k));
    }
    int mid = (l + r) >> 1;
    if (L <= mid) {
        modify(t[k].l, l, mid, L, R, x);
    }
    if (R > mid) {
        modify(t[k].r, mid + 1, r, L, R, x);
    }
}
int main() {
    freopen("choose.in", "r", stdin);
    freopen("choose.out", "w", stdout);
    n = read();
    for (int i = 1; i <= n; i++) {
        insert(rt, rt, 1, n, i, read());
    }
    m = read();
    while (m--) {
        int kind = read();
        if (kind == 1) {
            int x = read(), y = read();
            insert(rt, rt, 1, n, x, y);
        } else {
            int x = read(), l = read(), r = read();
            modify(rt, 1, n, l, r, v[x]);
        }
    }
    for (int i = 1; i <= sz; i++) {
        if (!dfn[i]) {
            tarjan(i);
        }
    }
    for (int i = 1; i <= sz; i++) {
        num[col[i]] += t[i].s;
    }
    for (int i = 1; i <= tot; i++) {
        if (col[e[i].u] != col[e[i].v]) {
            ou[col[e[i].u]]++;
        }
    }
    ll ans = 1ll << 60;
    for (int i = 1; i <= cnt; i++) {
        if (!ou[i] && ans > num[i]) {
            ans = num[i];
        }
    }
    printf("%lld", ans);
    return 0;
}

```