带花树算法

带花树算法是为了解决一般图的匹配问题的，注意这种问题是网络流构图无法可解的。

效率为$O(|V|^2|E|)$实际表现会好得多。

下面为牛客2020多校一的I题代码，精彩的构图+带花树。

```
#include <bits/stdc++.h>

using namespace std;
const int maxn = 500;
struct struct_edge {
    int v;
    struct_edge *n;
};
typedef struct_edge *edge;
struct_edge pool[maxn * maxn * 2];
edge top = pool, adj[maxn];
int V, match[maxn], qh, qt, q[maxn], father[maxn], base[maxn];
bool inq[maxn], inb[maxn], ed[maxn][maxn];

void addEdge(int u, int v) {
    --u, --v;
    if (!ed[u][v]) {
        ed[u][v] = ed[v][u] = true;
        top->v = v, top->n = adj[u], adj[u] = top++;
        top->v = u, top->n = adj[v], adj[v] = top++;
    }
}

int LCA(int root, int u, int v) {
    static bool inp[maxn];
    memset(inp, 0, sizeof(inp));
    while (1) {
        inp[u = base[u]] = true;
        if (u == root) break;
        u = father[match[u]];
    }
    while (1) {
        if (inp[v = base[v]]) return v;
        else v = father[match[v]];
    }
}

void mark_blossom(int lca, int u) {
    while (base[u] != lca) {
        int v = match[u];
        inb[base[u]] = inb[base[v]] = true;
        u = father[v];
```

```
        if (base[u] != lca) father[u] = v;
    }
}

void blossom_contraction(int s, int u, int v) {
    int lca = LCA(s, u, v);
    memset(inb, 0, sizeof(inb));
    mark_blossom(lca, u);
    mark_blossom(lca, v);
    if (base[u] != lca)
        father[u] = v;
    if (base[v] != lca)
        father[v] = u;
    for (int u = 0; u < V; u++)
        if (inb[base[u]]) {
            base[u] = lca;
            if (!inq[u])
                inq[q[++qt] = u] = true;
        }
}

int find_augmenting_path(int s) {
    memset(inq, 0, sizeof(inq));
    memset(father, -1, sizeof(father));
    for (int i = 0; i < V; i++) base[i] = i;
    inq[q[qh = qt = 0] = s] = true;
    while (qh <= qt) {
        int u = q[qh++];
        for (edge e = adj[u]; e; e = e->n) {
            int v = e->v;
            if (base[u] != base[v] && match[u] != v)
                if ((v == s) || (match[v] != -1 && father[match[v]] != -1))
                    blossom_contraction(s, u, v);
                else if (father[v] == -1) {
                    father[v] = u;
                    if (match[v] == -1)
                        return v;
                    else if (!inq[match[v]])
                        inq[q[++qt] = match[v]] = true;
                }
        }
    }
    return -1;
}

int augment_path(int s, int t) {
    int u = t, v, w;
    while (u != -1) {
        v = father[u];
        w = match[v];
```

```
        match[v] = u;
        match[u] = v;
        u = w;
    }
    return t != -1;
}

int edmonds() {
    int matchc = 0;
    memset(match, -1, sizeof(match));
    for (int u = 0; u < V; u++)
        if (match[u] == -1)
            matchc += augment_path(u, find_augmenting_path(u));
    return matchc;
}

int N, M, d[55];

int main() {
    while (scanf("%d%d", &N, &M) == 2) {
        V = N * 2 + M * 2;
        int sum = 0;
        memset(adj, 0, sizeof(adj));
        memset(ed, 0, sizeof(ed));
        for (int i = 1; i <= N; i++)
            scanf("%d", &d[i]), sum += d[i];
        if (sum & 1) {
            puts("No");
            continue;
        }
        for (int i = 1, u, v; i <= M; i++) {
            scanf("%d%d", &u, &v);
            if (d[u] == 2 && d[v] == 2) {
                sum += 2;
                addEdge(u, i + 2 * N);
                addEdge(u + N, i + 2 * N);
                addEdge(v, i + 2 * N + M);
                addEdge(v + N, i + 2 * N + M);
                addEdge(i + 2 * N, i + 2 * N + M);
            } else {
                addEdge(u, v);
                if (d[u] == 2)
                    addEdge(u + N, v);
                if (d[v] == 2)
                    addEdge(u, v + N);
            }
        }
        puts(edmonds() * 2 == sum ? "Yes" : "No");
    }
    return 0;
}
```