```cpp
// Max Flow:
// Adjacency list implementation of Dinic's blocking flow algorithm.
// This is very fast in practice, and only loses to push-relabel flow.
//
// Running time:
//     O(|V|^2 |E|)
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - maximum flow value
//     - To obtain the actual flow values, look at all edges with
//       capacity > 0 (zero capacity edges are residual edges).
#include <bits/stdc++.h>
using namespace std;
#define clr(a) (a.clear())
#define sz(x) (int)x.size()
#define pb push_back
const int INF = 2000000000;

struct Edge {
    int from, to, cap, flow, index;
    Edge(int from, int to, int cap, int flow, int index) :
        from(from),to(to),cap(cap),flow(flow),index(index) {}
};
struct Dinic{
    int N;
    vector< vector<Edge> > G;
    vector<Edge *> dad;
    vector<int> Q;

    Dinic(int N) : N(N), G(N), dad(N), Q(N) {}

    void AddEdge(int from, int to, int cap) {
        G[from].push_back(Edge(from,to,cap,0,G[to].size()));
        if (from == to) G[from].back().index++;
        G[to].push_back(Edge(to,from,0,0,G[from].size()-1));
    }

    long long BlockingFlow(int s, int t) {
        fill(dad.begin(), dad.end(), (Edge *) NULL);
        dad[s] = &G[0][0] - 1;

        int head = 0, tail = 0;
        Q[tail++] = s;
        while (head < tail) {
            int x = Q[head++];
            for (int i = 0; i < G[x].size(); i++) {
                Edge &e = G[x][i];
                if (!dad[e.to] && e.cap - e.flow > 0) {
                    dad[e.to] = &G[x][i];
                    Q[tail++] = e.to;
                }
            }
        }
        if (!dad[t]) return 0;

        long long totflow = 0;
        for (int i = 0; i < G[t].size(); i++) {
            Edge *start = &G[G[t][i].to][G[t][i].index];
            int amt = INF;
            for(Edge *e=start; amt&&e!=dad[s]; e=dad[e->from]){
                if (!e) {
                    amt = 0;
                    break;
                }
                amt = min(amt, e->cap - e->flow);
            }
```

```
            if (amt == 0) continue;
            for(Edge *e=start; amt&&e!=dad[s]; e=dad[e->from]){
                e->flow += amt;
                G[e->to][e->index].flow -= amt;
            }
            totflow += amt;
        }
        return totflow;
    }

    long long GetMaxFlow(int s, int t) {
        long long totflow = 0;
        while (long long flow = BlockingFlow(s, t))
            totflow += flow;
        return totflow;
    }
};
int main(){
    int n, m;  cin >> n >> m;
    vector <int> a(n + 1), b(n + 1);

    int tot1 = 0;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        tot1 += a[i];
    }

    int tot2 = 0;
    for (int i = 1; i <= n; i++) {
        cin >> b[i];
        tot2 += b[i];
    }
    int src = 0, snk = n + n + 1;

    Dinic din(n + n + 2);

    for (int i = 1; i <= n; i++) {
        din.AddEdge(src, i, a[i]);
        din.AddEdge(i, i + n, INF);
    }
    for (int i = 1; i <= n; i++) {
        din.AddEdge(i + n, snk, b[i]);
    }
    for(int i = 0; i < m; i++) {
        int from, to; cin >> from >> to;
        din.AddEdge(from, to + n, INF);
        din.AddEdge(to, from + n, INF);
    }

    int Flow = din.GetMaxFlow(src, snk);

    if (Flow == tot1 && Flow == tot2) {
        cout << "YES" << endl;
        int ans[n + 1][n + 1];
        memset (ans, 0, sizeof ans);
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j < sz(din.G[i]); j++) {
                int to = din.G[i][j].to - n;
                if (to >= 1 && to <= n) {
                    ans[i][to] = din.G[i][j].flow;
                }
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                cout << ans[i][j] << ' ';
            }
            cout << endl;
        }
    }else{
        cout << "NO" << endl;
    }
}
```