费用流

```
struct CostFlow{
#define inf 0x3f3f3f3f
    vector<int> to, flow, cost, nxt;
    vector<int> head, dis, vis, now;
    int cnt, s, t, n, m;
    CostFlow(int S, int T, int cntn, int cnte){
        head.resize(cntn + 1, -1);
        to.resize(cnte << 1);
        flow.resize(cnte << 1);
        cost.resize(cnte << 1);
        nxt.resize(cnte << 1);
        vis.resize(cntn + 1);
        s = S, t = T;
        cnt = 0, n = cntn, m = cnte;
    }
    void addEdge(int u, int v, int f, int c){
        to[cnt] = v, flow[cnt] = f, cost[cnt] = c, nxt[cnt] = head[u], head[u] = cnt++;
        to[cnt] = u, flow[cnt] = 0, cost[cnt] = -c, nxt[cnt] = head[v], head[v] = cnt++;
    }
    bool SPFA(){
        queue<int> q;
        dis.clear(), dis.resize(n + 1, inf);
        dis[s] = 0;
        now = head;
        q.push(s);
        while(!q.empty()){
            int node = q.front(); q.pop();
            vis[node] = 0;
            for(int i = head[node];~i;i = nxt[i]){
                if(flow[i] && dis[to[i]] > dis[node] + cost[i]){
                    dis[to[i]] = dis[node] + cost[i];
                    if(!vis[to[i]]){
                        vis[to[i]] = 1;
                        q.push(to[i]);
                    }
                }
            }
        }
        return dis[t] != inf;
    }
    int dfs(int x, int f, int& c){
        if(x == t){
            c += dis[t] * f;
            return f;
        }
        vis[x] = 1;
        int rest = f;
        for(int i = now[x];rest && ~i;i = nxt[i]){
            if(flow[i] && dis[to[i]] == dis[x] + cost[i] && !vis[to[i]]){
                now[x] = i;
                int tmp = dfs(to[i], min(flow[i], rest), c);
                if(!tmp) dis[to[i]] = inf;
                rest -= tmp;
                flow[i] -= tmp;
                flow[i ^ 1] += tmp;
            }
        }
        vis[x] = 0;
        return f - rest;
    }
```

```
        array<int, 2> Dinic(){
            int f = 0, c = 0;
            while(SPFA()) f += dfs(s, inf, c);
            return {f, c};
        }
    };
```

使用$spfa$寻找最短路然后用$dinic$计算费用流，返回顺序为流量+费用