G 贪心

先考虑如果不用删边，最短的路径应该怎么求。

一个点到下一个点，要么顺时针走，要么逆时针走，统计两者的较小值即可。

我们不去想删除一条边后的路径该怎么求，而要去想删哪条边的代价最小。

我们假设不删边时，顺时针走总是比逆时针走更优。

那么我们如果改成走逆时针，那么就会多出这两者之差的代价。

我们删掉顺时针走的路径上的任意一条边，都会导致我们只能转而去走逆时针。

显而易见的，我们可以使用差分，将更优的路径上的所有边都加上两个选择的差。

最后被加的数最小的边就是最优解，答案就是不删边的答案加上那条边被加的数。

```
void solve() {
    cin >> n >> m;
    vector<ll> a(m + 1);
    repi(i, 1, m + 1) { cin >> a[i]; }
    ll ans = 0;
    vector<ll> pre(n + 2, 0);
    for (int i = 2; i <= m; i++) {
        int x = max(a[i], a[i - 1]);
        int y = min(a[i], a[i - 1]);
        int aa = x - y;
        int b = y + n - x;
        if (aa < b) {
            ans += aa;
            pre[y] += b - aa;
            pre[x] -= (b - aa);
        } else {
            ans += b;
            pre[1] += aa - b;
            pre[y] -= (aa - b);
            pre[x] += aa - b;
        }
    }
    ll tot = INF;
    for (int i = 1; i <= n; i++) {
        pre[i] += pre[i - 1];
        tot = min(tot, pre[i]);
    }
    cout << tot + ans;
}
```

H 暴力/dij

因为范围很小,记录边的下标,暴力枚举每条边走不通依次跑Dij即可.

```
#define Pa pair<ll, ll>
struct node{
    ll to, next, w;
```

```
    node() {}
    node(ll w, ll to) : to(to), next(0), w(w) {}
    friend bool operator<(node a, node b){
        return a.w > b.w;
    }
};
ll nxt, rnxt;
node e[maxn * 2];
ll h[N];
bool vis[N];
ll dis[N];
void add(ll u,ll v,ll w){
    e[++nxt].next=h[u];
    e[nxt].w=w;
    e[nxt].to=v;
    h[u] = nxt;
}
void init(){
    nxt=1;
    for (int i = 0; i <= n; i++)//网络流使用2*n+7
    {
        h[i] = -1;
    }
}
Pa pre[N];
void dij(ll st){
    for (int i = 0; i < n+1; i++){
        dis[i] = INF;
        vis[i] = 0;
    }
    dis[st] = 0;
    priority_queue<node> q;
    q.push({0, st});
    while (!q.empty()){
        node tt = q.top();
        q.pop();
        if (vis[tt.to])
            continue;
        vis[tt.to] = 1;
        for (int i = h[tt.to]; ~i; i = e[i].next){
            ll v = e[i].to, w = e[i].w;
            if (dis[v] > tt.w + w){
                pre[v] = Pa(tt.to,i);
                dis[v] = tt.w + w;
                q.push({dis[v], v});
            }
        }
    }
};
void solve(){
    cin >>n ;
    for (int i = 0; i < n+1; i++){
        pre[i] = {0,0};
    }
    cin >>m;
    init();
```

```
    vector<ll> a;
    for (int i = 0; i < m; i++){
        cin >>x>>y;
        add(x,y,1);
    }
    set<ll>b;
    ll now = n;
    vector<ll> ans(m+1);
    dij(1);
    while (now>1){
        b.insert(pre[now].second);
        now = pre[now].first;
    }
    // 枚举用不到边
    repi(i,2,2+m){
        if (!b.count(i)){
            ans[i-1] = dis[n];
        }
        else {
            a.push_back(i);
        }
    }
    for(auto i:a){
        ll tt = e[i].w;
        e[i].w = INF;// 把边去掉
        // e[i^1].w = INF;
        dij(1);
        ans[i-1]= dis[n];
        e[i].w=tt;
    }
    repi(i,1,1+m){
        if (ans[i] ==INF) ans[i] = -1;
        cout << ans[i]<<'\n';
    }
    cout << endl;
}
```

I Dij

假设 x 到 y 有一条边，则从 x 到 y 与从 y 到 y 所得的幸福感不一定相同，所以，对于一个无向边，将其拆成连个有向边，并考虑边的权值。

对于每条边，只需判断两个端点的大小关系，然后确定权值即可。

```
ll ans = 0;
void dij(ll st){
    memset(dis, 0x3f, sizeof(dis));
    dis[st] = 0;
    priority_queue<node> q;
    q.push({0, st});
    while (!q.empty()){
        node tt = q.top();
        q.pop();
        if (vis[tt.to])
```

```
            continue;
        vis[tt.to] = 1;
        for (int i = h[tt.to]; ~i; i = e[i].next){
            ll v = e[i].to, w = e[i].w;
            if (dis[v] > tt.w + w){
                dis[v] = tt.w + w;
                q.push({dis[v], v});
            }
        }
    }
}
void solve(){
    cin >>n >>m;
    init();
    vector<ll> a(n+1);
    repi(i,1,n+1){
        cin >> a[i];
    }
    ans =0 ;
    FOR(m){
        cin >>x>>y;
        ll sum = abs(a[x]-a[y]);
        if( a[x]>a[y]){
            add(x,y,0);
            add(y,x,sum);
        }else if (a[x]==a[y]){
            add(x,y,0);
            add(y,x,0);
        }else {
            add(x,y,sum);
            add(y,x,0);
        }
    }
    dij(1);
    ans =0 ;
    repi(i,1,n+1){
        ans = max(ans,a[1]-a[i]-dis[i]);
    }
    cout <<ans;
}
```