二分图最大权匹配

Kuhn-Munkres Algorithm

先把两个集合中点数较少的点补上，使得两集合点数相同，不存在的边权取 0 或 $-\mathrm{INF}$（依题目而定）。这样问题转化成**最大权完美匹配**。

Concepts：

- **可行顶标**：给每个节点 i 分配一个权值 $l(i)$，对所有边 (u,v) 满足 $w(u,v)\leqslant l(u)+l(v)$.
- **相等子图**：在一组可行顶标下，原图中所有点以及满足 $w(u,v)=l(u)+l(v)$ 的边构成的子图。

Theorem：对于某组可行顶标，如果其相等子图存在**完美匹配**，那么该匹配就是原二分图的**最大权完美匹配**。

证明：考虑原二分图的任意一组完美匹配 M，其边权和为 $val(M)=\sum_{(u,v)\in M}w(u,v)\leqslant\sum_{(u,v)\in M}l(u)+l(v)\leqslant\sum_{i=1}^n l(i)$. 而任意一组可行顶标的相等子图的完美匹配 M' 的边权和 $val(M')=\sum_{(u,v)\in M'}w(u,v)=\sum_{(u,v)\in M'}l(u)+l(v)=\sum_{i=1}^n l(i)$. 故任意一组完美匹配边权和都不会大于 $val(M')$，即 M' 是最大权完美匹配。

Algorithm：根据上述定理，我们不断调整可行顶标，使得相等子图是完美匹配即可。（可以理解为匈牙利算法+不断调整可行顶标）

Complexity：$O(n^3)$

Code：

`w[][]`：邻接矩阵存图（存边权）；

`lx[i],rx[i]`：左/右边点的顶标；

`visx[i],visy[i]` 左/右边点访问标记；

`matchx[i],matchy[i]` 左边点匹配的右边点，右边点匹配的左边点；

`slack[i]`：松弛数组，表示对于指向右边点 i 的所有边的 $\min\{lx[u]+ly[i]-w[u][i]\}$；

`pre[i]`：记录交错路径。

和匈牙利算法一样，**只用建从左向右的边**。

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

using namespace std;

typedef long long LL;

const LL INF = 1e14;
const int N = 505;

namespace KM{
    int n;
    LL w[N][N];
    int matchx[N], matchy[N];
    LL lx[N], ly[N];
    LL slack[N];
    bool visx[N], visy[N];

    queue<int> q;
    int pre[N];

    bool check(int cur){
        visy[cur] = true;
        if(matchy[cur]){
            if(!visx[matchy[cur]]){
                q.push(matchy[cur]);
```

```
                visx[matchy[cur]] = true;
            }
            return false;
        }
        while(cur)  swap(cur, matchx[matchy[cur] = pre[cur]]);
        return true;
    }
    void bfs(int s){
        fill(visx, visx+n+1, false);
        fill(visy, visy+n+1, false);
        fill(slack, slack+n+1, INF);
        while(!q.empty())   q.pop();
        q.push(s), visx[s] = true;
        while(1){
            while(!q.empty()){
                int cur = q.front(); q.pop();
                for(int i = 1; i <= n; i++){
                    LL diff = lx[cur] + ly[i] - w[cur][i];
                    if(!visy[i] && diff <= slack[i]){
                        slack[i] = diff;
                        pre[i] = cur;
                        if(diff == 0)
                            if(check(i))    return;
                    }
                }
            }
            LL delta = INF;
            for(int i = 1; i <= n; i++)
                if(!visy[i] && slack[i])
                    delta = min(delta, slack[i]);
            for(int i = 1; i <= n; i++){
                if(visx[i]) lx[i] -= delta;
                if(visy[i]) ly[i] += delta;
                else    slack[i] -= delta;
            }
            while(!q.empty())   q.pop();
            for(int i = 1; i <= n; i++)
                if(!visy[i] && !slack[i] && check(i))
                    return;
        }
    }
    void solve(){
        fill(matchx, matchx+n+1, 0);
        fill(matchy, matchy+n+1, 0);
        fill(ly, ly+n+1, 0);
        for(int i = 1; i <= n; i++){
            lx[i] = 0;
            for(int j = 1; j <= n; j++)
                lx[i] = max(lx[i], w[i][j]);
        }
        for(int i = 1; i <= n; i++) bfs(i);
    }
}

int n, m;

int main(){
    scanf("%d%d", &n, &m);
    KM::n = n;
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            KM::w[i][j] = -INF;
    for(int i = 1; i <= m; i++){
        int y, c, h; scanf("%d%d%d", &y, &c, &h);
        KM::w[y][c] = h;
    }
    KM::solve();
```

```
    LL ans = 0;
    for(int i = 1; i <= n; i++) ans += KM::w[i][KM::matchx[i]];
    printf("%lld\n", ans);
    for(int i = 1; i <= n; i++) printf("%d ", KM::matchy[i]);
    return 0;
}
```

转换为费用流模型

Idea：左边所有点接原点，右边所有点接汇点，容量为 1，权值为 0；原来的边从左往右连边，容量为 1，权值为边权。跑最大费用最大流即可。