费用流

Min-cost Max-flow

Idea：将 EK 算法中的 bfs 改为 spfa，每次寻找费用最少的增广路。

Complexity：$O(VE^2)$

ATT：链式前向星存储时，`edgeNum` 初始化为1；建图时建流为 `0`、费用为 `-f` 的反向边。

Code：

```
int minCost[N], minFlow[N], pre[N];
bool inq[N];
int spfa(){
    for(int i = 1; i <= n; i++){
        minCost[i] = minFlow[i] = INF;
        pre[i] = 0;
        inq[i] = 0;
    }
    queue<int> q;
    q.push(src);
    inq[src] = 1;
    minCost[src] = 0;
    while(!q.empty()){
        int cur = q.front(); q.pop();
        inq[cur] = 0;
        for(int i = head[cur]; i; i = edge[i].nxt){
            if(edge[i].flow && minCost[edge[i].to] > minCost[cur] + edge[i].cost){
                minCost[edge[i].to] = minCost[cur] + edge[i].cost;
                minFlow[edge[i].to] = min(minFlow[cur], edge[i].flow);
                pre[edge[i].to] = i;
                if(!inq[edge[i].to]){
                    q.push(edge[i].to);
                    inq[edge[i].to] = 1;
                }
            }
        }
    }
    if(pre[dst] == 0)   return -1;
    return minFlow[dst];
}

void EK(int &maxflow, int &mincost){
    maxflow = mincost = 0;
    int flow = 0;
    while((flow = spfa()) != -1){
        int t = dst;
        while(t != src){
            edge[pre[t]].flow -= flow;
            edge[pre[t]^1].flow += flow;
            t = edge[pre[t]^1].to;
        }
        maxflow += flow;
        mincost += flow * minCost[dst];
    }
}
```