```cpp
class Mincostflow {
private:
  struct edge {
    int eid, from, to;
    ll cap, cost;
  };
  struct node {
    int id; ll d; int from_eid; vector<int> to_eids;
  };
  struct pq_t {
    int id; ll d;
    bool operator<(const pq_t & another) const {
      return d != another.d ? d > another.d : id > another.id;
    }
  };
  int dual_eid(int eid) {
    if (eid < m) return eid + m;
    else return eid - m;
  }
  vector<node> nodes;
  vector<edge> edges;
  int n, m;
  int source, sink;
  bool overflow;
public:
  Mincostflow(const vvi &lst, const vvll &cap, const vvll &cst, int s, int t) {
    n = lst.size();
    nodes.resize(n);
    Loop(i, n) nodes[i] = { i, LLONG_MAX, -1, {} };
    int eid = 0;
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[i].to_eids.push_back(eid);
        edges.push_back({ eid, i, lst[i][j], cap[i][j], cst[i][j] });
        eid++;
      }
    }
    m = eid;
    Loop(i, n) {
      Loop(j, lst[i].size()) {
        nodes[lst[i][j]].to_eids.push_back(eid);
        edges.push_back({ eid, lst[i][j], i, 0, -cst[i][j] });
        eid++;
      }
    }
    source = s;
    sink = t;
    overflow = false;
  }
  bool add_flow(ll f) {
    if (overflow) return false;
    while (f > 0) {
      Loop(i, n) {
        nodes[i].d = LLONG_MAX;
        nodes[i].from_eid = -1;
      }
      // Bellmanford
      nodes[source].d = 0;
      Loop(k, n) {
        Loop(i, n) {
          int a = i;
          if (nodes[a].d == LLONG_MAX) continue;
          Foreach(eid, nodes[a].to_eids) {
            if (edges[eid].cap == 0) continue;
            int b = edges[eid].to;
            if (nodes[a].d + edges[eid].cost < nodes[b].d) {
              nodes[b].d = nodes[a].d + edges[eid].cost;
              nodes[b].from_eid = eid;
              if (k == n - 1) {
                return false;
              }
```

```
72              }
73            }
74          }
75        }
76        if (nodes[sink].d == LLONG_MAX) return false;
77        int a = sink;
78        ll df = f;
79        while (a != source) {
80          df = min(df, edges[nodes[a].from_eid].cap);
81          a = edges[nodes[a].from_eid].from;
82        }
83        a = sink;
84        while (a != source) {
85          edges[nodes[a].from_eid].cap -= df;
86          edges[dual_eid(nodes[a].from_eid)].cap += df;
87          a = edges[nodes[a].from_eid].from;
88        }
89        f -= df;
90      }
91      return true;
92    }
93    vll get_eid_flow() {
94      vll ret(m, -1);
95      if (overflow) return ret;
96      Loop(i, m) {
97        ret[i] = edges[i + m].cap;
98      }
99      return ret;
100   }
101   ll get_flow() {
102     if (overflow) return -1;
103     ll ret = 0;
104     Foreach(eid, nodes[sink].to_eids) {
105       if (eid >= m) ret += edges[eid].cap;
106     }
107     return ret;
108   }
109   ll get_cost() {
110     if (overflow) return -1;
111     ll ret = 0;
112     Loop(i, m) {
113       ret += edges[i].cost * edges[i + m].cap;
114     }
115     return ret;
116   }
117 };
```