Week 37: Advanced Flow Variants – Min-Cost Circulation & Flow with Lower Bounds

Topics: - Minimum Cost Circulation Problem - Circulation with Demands and Lower Bounds - Flow with Lower/Upper Capacity Constraints - Successive Shortest Path Algorithm - Cycle Canceling Method - Applications: Task Scheduling with Costs, Supply-Demand Networks, Transportation Problems

Weekly Tips: - Convert circulation with lower/upper bounds to a standard flow problem by adjusting demands. - Successive shortest path repeatedly augments along shortest paths w.r.t. reduced costs. - Cycle canceling detects negative cycles and augments flow until no cycle remains. - Always check feasibility when lower bounds are involved. - Min-cost flow is widely applied in logistics, scheduling, and network optimization.

Problem 1: Min-Cost Max-Flow (Successive Shortest Path) Link: CSES Task Assignment **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct Edge{int v, cap, cost, rev;};
const int INF=1e9;
vector<vector<Edge>> adj;
int n;
void addEdge(int u,int v,int cap,int cost){
    adj[u].push_back({v,cap,cost,(int)adj[v].size()});
    adj[v].push_back({u,0,-cost,(int)adj[u].size()-1});
}
pair<int,int> minCostMaxFlow(int s,int t){
    int flow=0,cost=0;
    vector<int> dist, parentV, parentE;
    while(true){
        dist.assign(n,INF); parentV.assign(n,-1); parentE.assign(n,-1);
        dist[s]=0; vector<bool> inq(n,false);
        queue<int> q; q.push(s); inq[s]=true;
        while(!q.empty()){
            int u=q.front(); q.pop(); inq[u]=false;
            for(int i=0;i<(int)adj[u].size();i++){
                Edge &e=adj[u][i];
                if(e.cap>0 && dist[e.v]>dist[u]+e.cost){
                    dist[e.v]=dist[u]+e.cost;
                    parentV[e.v]=u; parentE[e.v]=i;
                    if(!inq[e.v]){ q.push(e.v); inq[e.v]=true; }
                }
            }
        }
        if(dist[t]==INF) break;
```

```
        int f=INF;
        for(int v=t;v!=s;v=parentV[v]){
            f=min(f,adj[parentV[v]][parentE[v]].cap);
        }
        flow+=f; cost+=f*dist[t];
        for(int v=t;v!=s;v=parentV[v]){
            Edge &e=adj[parentV[v]][parentE[v]];
            e.cap-=f; adj[v][e.rev].cap+=f;
        }
    }
    return {flow,cost};
}
int main(){
    int jobs,workers; cin>>jobs>>workers; n=jobs+workers+2;
    int s=jobs+workers,t=s+1; adj.assign(n,{});
    for(int i=0;i<jobs;i++) addEdge(s,i,1,0);
    for(int j=0;j<workers;j++) addEdge(jobs+j,t,1,0);
    for(int i=0;i<jobs;i++){
        for(int j=0;j<workers;j++){
            int cost; cin>>cost;
            addEdge(i,jobs+j,1,cost);
        }
    }
    auto [f,c]=minCostMaxFlow(s,t);
    cout<<c<<endl;
}
```

Explanation Comments: - Build bipartite graph between jobs and workers with cost edges. - Apply successive shortest path to find augmenting paths. - Accumulate flow and cost until no more augmenting path exists. - Returns optimal assignment with minimum cost.

Problem 2: Flow with Lower Bounds Conceptual Overview: - Each edge has capacity `[L, U]`. - Subtract L from both ends and adjust node demands accordingly. - Add super-source/sink to balance demands. - Solve standard max-flow/min-cost flow. - If all demands are satisfied, solution exists.

Applications: - Scheduling with minimum requirements. - Transportation problems with mandatory shipments. - Network routing with guaranteed throughput.

End of Week 37 - Master advanced flow problems with costs and lower bounds. - Practice min-cost max-flow, circulation, and feasibility checks. - Essential for optimization-heavy ACM-ICPC problems.