Week 26: Advanced Flow & Matching Problems

Topics: - Min-Cost Max-Flow Algorithm - Circulation Problems with Demands - Bipartite Matching and Maximum Weight Matching - Flow with Lower and Upper Bounds - Applications: Job Assignment, Network Optimization, Matching in Graphs

Weekly Tips: - Min-Cost Max-Flow combines maximum flow computation with shortest path cost optimization. - Use SPFA or Dijkstra with potentials to handle costs efficiently. - Circulation problems involve balancing demands and supplies in a network. - Lower bounds can be transformed to standard max-flow problems. - Practice job assignment and weighted matching problems to solidify concepts.

Problem 1: Min-Cost Max-Flow Link: CSES Min-Cost Max-Flow **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct Edge{ int to, rev; long long cap, cost; };
const long long INF = 1e18;
vector<Edge> adj[505];
void addEdge(int u,int v,long long c,long long w){
    adj[u].push_back({v,(int)adj[v].size(),c,w});
    adj[v].push_back({u,(int)adj[u].size()-1,0,-w});
}
pair<long long,long long> minCostMaxFlow(int s,int t,int n){
    long long flow=0,cost=0;
    while(true){
        vector<long long> dist(n,INF); dist[s]=0;
        vector<int> prevv(n), preve(n);
        bool updated=true;
        for(int k=0;k<n && updated;k++){
            updated=false;
            for(int u=0;u<n;u++){
                if(dist[u]==INF) continue;
                for(int i=0;i<adj[u].size();i++){
                    Edge &e=adj[u][i];
                    if(e.cap>0 && dist[e.to]>dist[u]+e.cost){
                        dist[e.to]=dist[u]+e.cost; prevv[e.to]=u;
preve[e.to]=i; updated=true;
                    }
                }
            }
        }
        if(dist[t]==INF) break;
        long long d=INF;
        for(int v=t;v!=s;v=prevv[v]) d=min(d,adj[prevv[v]][preve[v]].cap);
        flow+=d; cost+=d*dist[t];
        for(int v=t;v!=s;v=prevv[v]){
            Edge &e=adj[prevv[v]][preve[v]];
            e.cap-=d; adj[v][e.rev].cap+=d;
        }
    }
    return {flow,cost};
}
int main(){
    int n,m; cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v; long long c,w; cin>>u>>v>>c>>w; addEdge(u,v,c,w);
    }
    auto res=minCostMaxFlow(0,n-1,n);
    cout<<res.first<<' '<<res.second<<endl;
}
```

Explanation Comments: - Each edge stores capacity and cost. - Repeatedly find shortest path (by cost) with available capacity and augment flow. - SPFA or Bellman-Ford used to handle negative costs safely. - Flow and total cost are updated after each augmentation.

Problem 2: Maximum Weight Bipartite Matching Link: HackerEarth Weighted Matching **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
const int INF=1e9;
int n,m;
vector<vector<int>> cost;
vector<int> u_match,v_match; vector<int> u_dist,v_dist;
bool bfs(){
    queue<int> q;
    fill(u_dist.begin(),u_dist.end(),INF);
    for(int u=0;u<n;u++) if(u_match[u]==-1){ u_dist[u]=0; q.push(u); }
    bool found=false;
    while(!q.empty()){
        int u=q.front(); q.pop();
        for(int v=0;v<m;v++){
            if(v_match[v]==-1) found=true;
            else if(u_dist[v_match[v]]==INF){ u_dist[v_match[v]]=u_dist[u]+1;
q.push(v_match[v]); }
        }
    }
    return found;
}
bool dfs(int u){
    for(int v=0;v<m;v++){
        if(v_match[v]==-1 || (u_dist[v_match[v]]==u_dist[u]+1 &&
dfs(v_match[v]))){
            u_match[u]=v; v_match[v]=u; return true;
        }
    }
    u_dist[u]=INF; return false;
}
int main(){
    cin>>n>>m; cost.assign(n,vector<int>(m,0));
    u_match.assign(n,-1); v_match.assign(m,-1);
    u_dist.assign(n,INF);
    for(int i=0;i<n;i++) for(int j=0;j<m;j++) cin>>cost[i][j];
    int match_count=0;
    while(bfs()) for(int u=0;u<n;u++) if(u_match[u]==-1 && dfs(u)) match_count+
+;
    cout<<match_count<<endl;
}
```

Explanation Comments: - Bipartite matching finds maximum pairing between two sets. - BFS layers the graph, DFS finds augmenting paths. - Maximum weight matching involves adjusting costs and potentials. - Useful for job assignment and network optimization problems.

End of Week 26 - Advanced flow and matching problems are critical in ACM-ICPC contests. - Min-Cost Max-Flow, Circulation, and Weighted Matching have many real-world applications. - Practice different variations to strengthen understanding and implementation skills.