Week 31: Network Flow – Advanced Applications & Techniques

Topics: - Maximum Flow Algorithms (Dinic, Push-Relabel) - Flow with Capacity Scaling - Min-Cost Max-Flow Revisited and Optimizations - Circulation with Demands and Lower/Upper Bounds - Applications: Project Selection, Baseball Elimination, Edge-Disjoint Paths - Flow Decomposition and Flow with Multiple Sources/Sinks

Weekly Tips: - Dinic's algorithm is efficient for unit and general capacities (O(V^2 E) or better for sparse graphs). - Push-Relabel is good for dense graphs. - Capacity scaling improves performance for large capacities. - Min-Cost Max-Flow can be optimized using potentials and successive shortest paths. - Understanding flow decomposition is useful for extracting actual paths from max-flow.

Problem 1: Dinic Maximum Flow Link: CSES Maximum Flow **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct Edge{ int to, rev; long long cap; };
vector<Edge> adj[505]; int level[505], ptr[505];
void addEdge(int u,int v,long long c){
    adj[u].push_back({v,(int)adj[v].size(),c});
    adj[v].push_back({u,(int)adj[u].size()-1,0});
}
bool bfs(int s,int t){
    fill(level,level+505,-1); level[s]=0;
    queue<int> q; q.push(s);
    while(!q.empty()){
        int u=q.front(); q.pop();
        for(auto &e:adj[u]) if(e.cap>0 && level[e.to]==-1){
            level[e.to]=level[u]+1; q.push(e.to);
        }
    }
    return level[t]!=-1;
}
long long dfs(int u,int t,long long f){
    if(u==t) return f;
    for(int &i=ptr[u];i<adj[u].size();i++){
        Edge &e=adj[u][i];
        if(e.cap>0 && level[e.to]==level[u]+1){
            long long pushed=dfs(e.to,t,min(f,e.cap));
            if(pushed){ e.cap-=pushed; adj[e.to][e.rev].cap+=pushed; return
pushed; }
        }
    }
    return 0;
```

```
}
long long dinic(int s,int t){
    long long flow=0;
    while(bfs(s,t)){
        fill(ptr,ptr+505,0);
        while(long long pushed=dfs(s,t,LLONG_MAX)) flow+=pushed;
    }
    return flow;
}
int main(){
    int n,m; cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v; long long c; cin>>u>>v>>c; addEdge(u,v,c);
    }
    cout<<dinic(0,n-1)<<endl;
}
```

Explanation Comments: - BFS builds level graph. - DFS pushes flow respecting levels. - Repeat until no augmenting path exists. - Efficient for sparse and medium-sized networks.

Problem 2: Min-Cost Max-Flow with Successive Shortest Path Link: CP-Algorithms 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; };
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,LLONG_MAX); 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]==LLONG_MAX) 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]==LLONG_MAX) break;
        long long d=LLONG_MAX;
        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: - Successive shortest path augments flow along minimum cost paths. - Recompute shortest paths after each augmentation. - Handles costs and capacities efficiently. - Useful in project selection, job assignment, and optimization problems.

End of Week 31 - Advanced flow techniques are fundamental in ACM-ICPC contests. - Dinic, Push-Relabel, and Min-Cost Max-Flow algorithms with applications solve real-world network problems efficiently. - Practice multiple variations and problem applications to master these concepts.