Week 21: Advanced Graph Techniques & Flow Algorithms

Topics: - Maximum Flow: Ford-Fulkerson, Edmonds-Karp, Dinic's Algorithm - Minimum Cut & Max Flow-Min Cut Theorem - Bipartite Matching using Hopcroft-Karp - Network Flow Applications: Circulation, Assignment Problem - Flow with Costs: Min-Cost Max-Flow

Weekly Tips: - Max Flow: Understand residual graph and augmenting paths. - Use BFS for Edmonds-Karp to ensure shortest augmenting paths. - Dinic's algorithm optimizes flow calculation with level graphs. - Min-Cost Max-Flow combines shortest paths with flow. - Bipartite matching can be solved efficiently with network flow.

Problem 1: Maximum Flow (Edmonds-Karp) Link: CSES Flight Routes **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
const int INF=1e9;
struct Edge{int to,rev,cap;};
vector<Edge> adj[505];
void addEdge(int u,int v,int c){
    adj[u].push_back({v,(int)adj[v].size(),c});
    adj[v].push_back({u,(int)adj[u].size()-1,0});
}
int bfs(int s,int t,vector<int>& level){
    fill(level.begin(),level.end(),-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]<0){
                level[e.to]=level[u]+1;q.push(e.to);
            }
        }
    }
    return level[t];
}
int dfs(int u,int t,int f,vector<int>& level,vector<int>& ptr){
    if(u==t) return f;
    for(;ptr[u]<adj[u].size();ptr[u]++){
        Edge &e=adj[u][ptr[u]];
        if(e.cap>0 && level[e.to]==level[u]+1){
            int pushed=dfs(e.to,t,min(f,e.cap),level,ptr);
            if(pushed){ e.cap-=pushed; adj[e.to][e.rev].cap+=pushed; return
pushed; }
```

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

Explanation Comments: - Dinic's algorithm uses BFS to build level graph and DFS for blocking flow. - Efficient for large graphs compared to basic Edmonds-Karp. - Residual capacities are updated after each augmenting path.

Problem 2: Bipartite Matching (Hopcroft-Karp) Link: CSES Matching **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
const int INF=1e9;
vector<int> adj[505];
int match[505],dist[505];
int n,m;
bool bfs(){
    queue<int> q;
    for(int i=1;i<=n;i++){
        if(match[i]==0){ dist[i]=0;q.push(i); }
        else dist[i]=INF;
    }
    dist[0]=INF;
    while(!q.empty()){
        int u=q.front(); q.pop();
        if(u!=0){
            for(int v:adj[u]){
                if(dist[match[v]]==INF){
                    dist[match[v]]=dist[u]+1;q.push(match[v]);
                }
            }
        }
    }
    return dist[0]!=INF;
}
bool dfs(int u){
    if(u!=0){
        for(int v:adj[u]){
            if(dist[match[v]]==dist[u]+1 && dfs(match[v])){
                match[u]=v; match[v]=u; return true;
            }
        }
        dist[u]=INF; return false;
    }
    return true;
}
int hopcroft_karp(){
    fill(match,match+505,0);
    int res=0;
    while(bfs()){
        for(int i=1;i<=n;i++) if(match[i]==0 && dfs(i)) res++;
    }
    return res;
}
int main(){
    cin>>n>>m;
    int edges; cin>>edges;
    for(int i=0;i<edges;i++){
        int u,v; cin>>u>>v;
        adj[u].push_back(v+n); // shift for bipartite
    }
    cout<<hopcroft_karp()<<endl;
}
```

Explanation Comments: - Hopcroft-Karp alternates BFS and DFS to find maximum matching in bipartite graphs. - BFS layers the graph to find shortest augmenting paths. - DFS augments along these paths for efficiency. - Time complexity: O(sqrt(V) * E) for bipartite graphs.

End of Week 21 - Advanced flow and matching algorithms are crucial for many ACM-ICPC network and assignment problems. - Practice different flow variants: max flow, min-cost flow, bipartite matching. - Understand implementation nuances to avoid TLE in contests.