Week 36: Advanced Graph Matching – Bipartite & General Matching Algorithms

Topics: - Bipartite Matching using Hopcroft-Karp Algorithm - Maximum Cardinality Matching in General Graphs (Edmonds' Blossom Algorithm) - Matching in Weighted Graphs (Hungarian Algorithm) - Applications: Job Assignment, Task Scheduling, Network Pairing - Vertex Cover, Edge Cover, and Kőnig's Theorem - Matching Augmenting Paths and Alternating Trees

Weekly Tips: - Hopcroft-Karp uses BFS/DFS to find multiple augmenting paths, O(sqrt(V) * E). - Edmonds' Blossom Algorithm handles odd-length cycles in general graphs. - Hungarian Algorithm solves weighted bipartite matching in O(n^3). - Augmenting paths are key to increasing matching size. - Visualize alternating paths and blossoms to understand complex augmentations.

Problem 1: 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<vector<int>> adj;
vector<int> pairU, pairV, dist;
int n,m;
bool bfs(){
    queue<int> q;
    for(int u=0;u<n;u++){
        if(pairU[u]==-1){ dist[u]=0; q.push(u); }
        else dist[u]=INF;
    }
    dist[INF]=INF;
    while(!q.empty()){
        int u=q.front(); q.pop();
        if(dist[u]<dist[INF]){
            for(int v:adj[u]){
                if(dist[pairV[v]]==INF){
                    dist[pairV[v]]=dist[u]+1; q.push(pairV[v]);
                }
            }
        }
    }
    return dist[INF]!=INF;
}
bool dfs(int u){
    if(u!=INF){
        for(int v:adj[u]){
            if(dist[pairV[v]]==dist[u]+1 && dfs(pairV[v])){
                pairV[v]=u; pairU[u]=v; return true;
```

```
            }
        }
        dist[u]=INF; return false;
    }
    return true;
}
int hopcroftKarp(){
    pairU.assign(n,-1); pairV.assign(m,-1); dist.assign(n+1,INF);
    int result=0;
    while(bfs()){
        for(int u=0;u<n;u++) if(pairU[u]==-1 && dfs(u)) result++;
    }
    return result;
}
int main(){
    cin>>n>>m; adj.assign(n,{});
    int e; cin>>e;
    for(int i=0;i<e;i++){ int u,v; cin>>u>>v; u--; v--; adj[u].push_back(v); }
    cout<<hopcroftKarp()<<endl;
}
```

Explanation Comments: - BFS finds multiple shortest augmenting paths. - DFS augments along paths to increase matching. - O(sqrt(V)*E) efficient for large bipartite graphs.

Problem 2: Weighted Bipartite Matching (Hungarian Algorithm) Link: CP-Algorithms Hungarian **Difficulty:** Advanced

C++ Solution Overview: - Maintain potential for left/right vertices. - Iteratively improve matching along minimal slack edges. - Update potentials to maintain reduced cost zero edges. - Extract maximum weight matching from final potentials.

Applications: - Job assignments to maximize profit. - Network pairing problems. - Minimizing total cost in task scheduling.

End of Week 36 - Mastering graph matching algorithms is essential for ACM-ICPC contests. - Practice bipartite and general matchings, augmenting paths, and Hungarian algorithm for weighted assignments.