Week 1: Basics & Implementation

Topics: - Input/Output, Loops, Conditionals - Arrays, Strings, Basic Math - Simple sorting

Weekly Tips: - Focus on writing clean, readable code. - Always test edge cases (0, 1, negative numbers, large numbers). - Use online judge IDE or local compiler to verify behavior.

Week 2: Ad-hoc & Simulation

Topics: - Simulation - Ad-hoc logic problems - Greedy basics

Weekly Tips: - Think step by step, simulate processes on paper first. - Carefully read problem constraints to optimize loops. - Greedy approach works if problem guarantees local optimality leads to global optimality.

Week 3: Sorting & Searching

Topics: - Sorting algorithms: QuickSort, MergeSort, STL sort - Binary Search & Ternary Search - Two-pointer technique

Weekly Tips: - Always check if STL sort suffices before implementing manually. - Binary search can be applied to sorted arrays or answer space. - Two-pointer technique is useful for finding pairs, sums, or sliding windows.

Week 4: Strings & Pattern Matching

Topics: - String searching: KMP, Rabin-Karp - Palindromes & substrings - Prefix/Suffix techniques

Weekly Tips: - Understand failure function in KMP for linear-time matching. - Use rolling hash for fast substring comparison. - Practice manipulating strings efficiently with STL.

Week 5: Recursion & Backtracking

Topics: - Recursion basics - Backtracking: N-Queens, subsets, combinations - Depth-First Search (DFS) for combinatorial problems

Weekly Tips: - Draw recursion trees to understand problem flow. - Watch stack usage and avoid unnecessary deep recursion. - Memoization can be applied to optimize repetitive recursive calls.

Week 6: Graph Theory Basics

Topics: - Graph representation: adjacency list & matrix - BFS & DFS traversal - Connected components - Shortest paths (Dijkstra, BFS for unweighted)

Weekly Tips: - Always check graph type: directed, undirected, weighted, unweighted. - Use visited array to avoid revisiting nodes. - For unweighted shortest paths, BFS is sufficient.

Week 7: Dynamic Programming (DP)

Topics: - Introduction to DP: memoization & tabulation - Classic problems: Fibonacci, Knapsack, LIS - Grid DP, state compression

Weekly Tips: - Identify overlapping subproblems and optimal substructure. - Start with recursive solution, then memoize or tabulate. - Practice simple to complex DP to build intuition.

Week 8: Advanced Graph Algorithms

Topics: - Minimum Spanning Trees: Prim, Kruskal - Bellman-Ford for negative weights - Floyd-Warshall for all-pairs shortest paths - Strongly Connected Components (Kosaraju, Tarjan)

Weekly Tips: - MST: Focus on edge selection and cycle prevention. - Bellman-Ford: Detect negative cycles. - Floyd-Warshall: Use DP-like approach for all-pairs shortest path. - SCC: Identify components and condensation graph.

Problem 1: Minimum Spanning Tree

Link: CSES MST **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Edge { int u,v,w; };
int n,m;
vector<Edge> edges;
vector<int> parent;

int find(int x) {
    if (parent[x] != x) parent[x] = find(parent[x]);
    return parent[x];
}
```

```
bool unite(int a,int b) {
    a = find(a); b = find(b);
    if (a==b) return false;
    parent[a]=b;
    return true;
}

int main() {
    cin >> n >> m;
    parent.resize(n+1);
    for(int i=1;i<=n;i++) parent[i]=i;
    for(int i=0;i<m;i++) {
        int u,v,w; cin>>u>>v>>w;
        edges.push_back({u,v,w});
    }
    sort(edges.begin(),edges.end(),[](Edge a, Edge b){return a.w<b.w;});
    long long cost=0;
    for(auto e:edges){
        if(unite(e.u,e.v)) cost+=e.w;
    }
    cout << cost << endl;
}
```

Explanation Comments: - Kruskal's algorithm using union-find. - Sort edges by weight, add if no cycle. - Classic MST implementation.

Problem 2: Bellman-Ford

Link: CSES Shortest Routes II **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n,m; cin>>n>>m;
    vector<tuple<int,int,int>> edges(m);
    for(int i=0;i<m;i++) {
        int u,v,w; cin>>u>>v>>w;
        edges[i]={u,v,w};
    }
    int start; cin>>start;
    vector<long long> dist(n+1,1e18);
    dist[start]=0;
```

```
    for(int i=1;i<=n-1;i++){
        for(auto [u,v,w]:edges){
            if(dist[u]+w<dist[v]) dist[v]=dist[u]+w;
        }
    }
    // Optional: check for negative cycle
    cout << "Distances from start node:" << endl;
    for(int i=1;i<=n;i++) cout<<dist[i]<<" "; cout<<endl;
}
```

Explanation Comments: - Relax all edges n-1 times. - Detects negative weight cycles if additional iteration decreases distance. - Suitable for graphs with negative weights.

Problem 3: Floyd-Warshall

Link: CSES All-Pairs Shortest Path **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n,m; cin>>n>>m;
    vector<vector<long long>> dist(n+1, vector<long long>(n+1,1e18));
    for(int i=1;i<=n;i++) dist[i][i]=0;
    for(int i=0;i<m;i++){
        int u,v,w; cin>>u>>v>>w;
        dist[u][v]=min(dist[u][v],(long long)w);
    }
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);
    cout << "All-pairs distances:" << endl;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++) cout<<dist[i][j]<<" "; cout<<endl;
    }
}
```

Explanation Comments: - Iteratively updates shortest path using intermediate nodes. - Handles all-pairs shortest path in O(n^3). - Demonstrates DP-like approach on graphs.

Problem 4: Strongly Connected Components

Link: Kattis SCC **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
vector<vector<int>> adj, radj;
vector<bool> visited;
vector<int> order, component;
void dfs1(int u){ visited[u]=true; for(int v:adj[u]) if(!visited[v]) dfs1(v);
order.push_back(u); }
void dfs2(int u){ visited[u]=true; component.push_back(u); for(int v:radj[u])
if(!visited[v]) dfs2(v); }
int main(){
    int n,m; cin>>n>>m;
    adj.resize(n); radj.resize(n); visited.assign(n,false);
    for(int i=0;i<m;i++){ int u,v; cin>>u>>v; adj[u].push_back(v);
radj[v].push_back(u); }
    for(int i=0;i<n;i++) if(!visited[i]) dfs1(i);
    fill(visited.begin(),visited.end(),false);
    reverse(order.begin(),order.end());
    for(int u:order){
        if(!visited[u]){
            component.clear(); dfs2(u);
            // component now contains one SCC
            for(int x:component) cout<<x<<" "; cout<<endl;
        }
    }
}
```

Explanation Comments: - Kosaraju's algorithm for SCC. - First DFS to get finishing times, second DFS on reversed graph. - Demonstrates component identification in directed graphs.

End of Week 8 - Focus on MST, shortest paths, and SCCs. - Understand when to use Kruskal vs Prim, Bellman-Ford vs Dijkstra. - Practice on weighted and directed graphs.