1) Kruskal + backtracking to print all possible MSTs

Approach:

1. Run a standard Kruskal once to get the minimum possible MST weight.
2. Sort edges by weight. Perform a backtracking search over the sorted edge list; at each step try either taking an edge (if it doesn't make a cycle) or skipping it, but prune branches whose current weight + minimum-possible remaining cannot reach the MST weight or which exceed the MST weight. Use union-find (disjoint set) for cycle checks. This follows the Kruskal + disjoint-set description in your lecture notes.

greedy

```
// AllMSTsKruskal.java
import java.util.*;


public class AllMSTsKruskal {
  static class Edge implements Comparable<Edge>{
    int u,v,w,idx;
    Edge(int u,int v,int w,int idx){this.u=u;this.v=v;this.w=w;this.idx=idx;}
    public int compareTo(Edge o){ return Integer.compare(this.w,o.w); }
    public String toString(){return "("+(u+1)+","+(v+1)+","+w+")";}
  }


  static class DSU {
    int[] p;
    DSU(int n){ p = new int[n]; for(int i=0;i<n;i++) p[i]=i;}
    int find(int x){ return p[x]==x?x:(p[x]=find(p[x])); }
    boolean union(int a,int b){
      int pa=find(a), pb=find(b);
      if(pa==pb) return false;
      p[pb]=pa; return true;
    }
```

```
    DSU copy(){ DSU c=new DSU(p.length); c.p = p.clone(); return c; }
}

int n;
List<Edge> edges;
long targetWeight;
Set<String> solutions = new LinkedHashSet<>();

public AllMSTsKruskal(int n, List<Edge> edges){
    this.n=n;
    this.edges = new ArrayList<>(edges);
    Collections.sort(this.edges);
    this.targetWeight = computeKruskalWeight();
}

private long computeKruskalWeight(){
    DSU d = new DSU(n);
    long w=0; int cnt=0;
    for(Edge e: edges){
        if(d.union(e.u,e.v)){
            w += e.w; cnt++;
            if(cnt==n-1) break;
        }
    }
    if(cnt != n-1) return Long.MAX_VALUE; // not connected
    return w;
}
```

```java
public void findAll(){
   if(targetWeight==Long.MAX_VALUE){
      System.out.println("Graph not connected.");
      return;
   }
   backtrack(0, new DSU(n), 0, new ArrayList<>());
   System.out.println("Total MST(s): " + solutions.size());
   int id=1;
   for(String s: solutions){
      System.out.println("MST#" + (id++) + " : " + s);
   }
}

private void backtrack(int idx, DSU dsu, long curW, List<Edge> curEdges){
   // prune: if too many edges or too heavy
   if(curEdges.size() > n-1) return;
   if(curW > targetWeight) return;
   // if we have n-1 edges, check weight and connectivity
   if(curEdges.size() == n-1){
      if(curW == targetWeight){
         // format canonical string to avoid duplicates (sort by edge idx)
         List<Integer> ids = new ArrayList<>();
         for(Edge e: curEdges) ids.add(e.idx);
         Collections.sort(ids);
         solutions.add(ids.toString());
      }
      return;
   }
```

```
        if(idx >= edges.size()) return;

        // fast bound: estimate minimal extra weight if we pick the smallest possible weights
available
        // (simple lower bound) -> optional, but helps
        long minPossibleExtra = 0;
        int need = (n-1) - curEdges.size();
        for(int i=idx;i<edges.size() && need>0;i++){
            minPossibleExtra += edges.get(i).w;
            need--;
        }
        if(curW + minPossibleExtra > targetWeight) return;

        Edge e = edges.get(idx);

        // Option 1: take edge if it doesn't create cycle
        DSU dsuCopy = dsu.copy();
        if(dsuCopy.union(e.u, e.v)){
            curEdges.add(e);
            backtrack(idx+1, dsuCopy, curW + e.w, curEdges);
            curEdges.remove(curEdges.size()-1);
        }
        // Option 2: skip edge
        backtrack(idx+1, dsu, curW, curEdges);
    }

    // Example run
    public static void main(String[] args){
```

```
        // Example graph: change to input-reading as needed
        int n=5;
        List<Edge> E = new ArrayList<>();
        E.add(new Edge(0,1,1,0));
        E.add(new Edge(0,2,3,1));
        E.add(new Edge(1,2,1,2));
        E.add(new Edge(2,3,4,3));
        E.add(new Edge(2,4,2,4));
        E.add(new Edge(3,4,5,5));
        AllMSTsKruskal solver = new AllMSTsKruskal(n,E);
        solver.findAll();
    }
}
```

Notes: algorithm and disjoint-set operations follow your lecture slides on Kruskal and DSU.

greedy

---

2) Print all spanning trees from adjacency matrix (undirected) using backtracking

Approach: enumerate edges, backtrack choosing edges such that no cycle forms and finally check connectivity (or use DSU to ensure edges chosen form a tree when count==n-1). This is a direct use of backtracking techniques from your notes.

backtracking

```
// AllSpanningTrees.java
import java.util.*;


public class AllSpanningTrees {
    static class Edge { int u,v; Edge(int u,int v){this.u=u;this.v=v;} public String toString(){return (u+1)+"-"+(v+1);} }
    int n;
    List<Edge> edges;
```

```
Set<String> solutions = new LinkedHashSet<>();

public AllSpanningTrees(int[][] adj){
    n = adj.length;
    edges = new ArrayList<>();
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(adj[i][j] == 1) edges.add(new Edge(i,j));
        }
    }
}

void findAll(){
    backtrack(0, new int[n], 0, new ArrayList<>());
    System.out.println("Total spanning trees: " + solutions.size());
    int id=1;
    for(String s: solutions) System.out.println("Tree#"+(id++)+" : "+s);
}

// union-find helper for cycle detection
int find(int[] p, int x){ return p[x]==x?x:(p[x]=find(p,p[x])); }

void backtrack(int idx, int[] parent, int chosen, List<Edge> cur){
    if(chosen > n-1) return;
    if(idx == edges.size()){
        if(chosen == n-1){
            // check connectivity via DSU built from cur
            int[] p = new int[n]; for(int i=0;i<n;i++) p[i]=i;
```

```
        int comps=n;
        for(Edge e: cur){
            int a=find(p,e.u), b=find(p,e.v);
            if(a!=b){ p[b]=a; comps--; }
        }
        if(comps==1){
            List<String> repr = new ArrayList<>();
            for(Edge e: cur) repr.add(e.toString());
            Collections.sort(repr);
            solutions.add(repr.toString());
        }
    }
    return;
}
// Option: include edge if it doesn't create cycle (fast check with a local DSU)
Edge e = edges.get(idx);
int[] ptemp = new int[n]; for(int i=0;i<n;i++) ptemp[i]=i;
for(Edge ed: cur){
    int a=find(ptemp, ed.u), b=find(ptemp, ed.v);
    if(a!=b) ptemp[b]=a;
}
if(find(ptemp,e.u) != find(ptemp,e.v)){
    cur.add(e);
    backtrack(idx+1, parent, chosen+1, cur);
    cur.remove(cur.size()-1);
}
// Option: skip
backtrack(idx+1, parent, chosen, cur);
```

```java
    }

    public static void main(String[] args){
        // sample adjacency matrix for a connected graph
        int[][] adj = {
            {0,1,1,0,0},
            {1,0,1,1,0},
            {1,1,0,0,1},
            {0,1,0,0,1},
            {0,0,1,1,0}
        };
        AllSpanningTrees alg = new AllSpanningTrees(adj);
        alg.findAll();
    }
}
```

Reference: backtracking slides for enumeration-of-solutions style algorithms.

backtracking

---

3) Topological sorts — print all topological orders (from adjacency matrix)

Use indegree array + backtracking: pick any current node with indegree 0, take it, reduce indegrees, recurse, then backtrack. This is exactly the backtracking technique used in your slides to print all solutions (applied to topological order here).

backtracking

```java
// AllTopologicalSorts.java
import java.util.*;

public class AllTopologicalSorts {
    int n;
    int[][] adj;
```

```
boolean[] used;
int[] indeg;
List<Integer> order = new ArrayList<>();

public AllTopologicalSorts(int[][] adj){
    this.adj = adj; n = adj.length;
    used = new boolean[n];
    indeg = new int[n];
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) if(adj[i][j]==1) indeg[j]++;
}

void allTopos(){
    backtrack();
}

void backtrack(){
    boolean flag=false;
    for(int i=0;i<n;i++){
        if(!used[i] && indeg[i]==0){
            flag=true;
            used[i]=true;
            order.add(i);
            // reduce indeg for neighbors
            for(int j=0;j<n;j++) if(adj[i][j]==1) indeg[j]--;
            backtrack();
            // backtrack
            for(int j=0;j<n;j++) if(adj[i][j]==1) indeg[j]++;
            order.remove(order.size()-1);
```

```
            used[i]=false;
        }
    }
    if(!flag){
        if(order.size()==n){
            System.out.println(orderToString(order));
        }
    }
}

String orderToString(List<Integer> o){
    StringBuilder sb = new StringBuilder();
    for(int x: o) sb.append((x+1)).append(" ");
    return sb.toString().trim();
}

public static void main(String[] args){
    int[][] adj = {
        // example DAG of 5 nodes (0..4). change to your adjacency matrix input.
        {0,1,1,0,0},
        {0,0,0,1,0},
        {0,0,0,1,1},
        {0,0,0,0,0},
        {0,0,0,0,0}
    };
    new AllTopologicalSorts(adj).allTopos();
}
}
```

Reference: backtracking slides and the sample question in your images.

backtracking

---

4) Tri-diagonal matrix (TDM) multiplication — functions requested: readTDM, printTDM, mulTDMs, printRES

Representation: for an n×n tri-diagonal matrix store only n×3 array: col 0 = sub-diagonal (a[i][0] holds A[i][i-1] for i>=1; set 0 for i==0), col1 = diagonal A[i][i], col2 = super-diagonal A[i][i+1] (0 for last). I provide the four functions requested and code that multiplies two TDMs producing a full dense result (but you can optimize to store minimal band if desired). This follows matrix multiplication ideas from your matrix lectures.

algorithm-analysis-and-divide-n…

```
// TDMultiply.java

import java.util.*;


public class TDMultiply {

   // TDM stored as [n][3]: [i][0]=A[i][i-1], [i][1]=A[i][i], [i][2]=A[i][i+1]

   public static int[][] readTDM(Scanner sc, int n){

      int[][] a = new int[n][3];

      System.out.println("Enter TDM elements row by row (sub, diag, super). Use 0 where
absent:");

      for(int i=0;i<n;i++){

         a[i][0] = sc.nextInt(); // sub (for i=0 should be 0)

         a[i][1] = sc.nextInt(); // diag

         a[i][2] = sc.nextInt(); // super (for i=n-1 should be 0)

      }

      return a;

   }


   public static void printTDM(int[][] a){

      int n = a.length;
```

```java
        System.out.println("TDM (full view):");
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                int val = 0;
                if(j==i) val = a[i][1];
                else if(j==i-1) val = a[i][0];
                else if(j==i+1) val = a[i][2];
                System.out.print(val + "\t");
            }
            System.out.println();
        }
    }

    public static int[][] mulTDMs(int[][] A, int[][] B){
        int n = A.length;
        // naive: compute full result but exploit sparsity: each row of A has nonzeros only in i-
1,i,i+1
        int[][] C = new int[n][n];
        for(int i=0;i<n;i++){
            for(int k=Math.max(0,i-1); k<=Math.min(n-1,i+1); k++){
                int aik = (k==i?A[i][1] : (k==i-1?A[i][0] : A[i][2]));
                if(aik==0) continue;
                // B has nonzeros in k-1,k,k+1
                for(int j=Math.max(0,k-1); j<=Math.min(n-1,k+1); j++){
                    int bkj = (j==k?B[k][1] : (j==k-1?B[k][0] : B[k][2]));
                    if(bkj==0) continue;
                    C[i][j] += aik * bkj;
                }
```

```
        }
      }
      return C;
    }

    public static void printRES(int[][] C){
      System.out.println("Resultant matrix:");
      for(int i=0;i<C.length;i++){
        for(int j=0;j<C.length;j++) System.out.print(C[i][j] + "\t");
        System.out.println();
      }
    }

    public static void main(String[] args){
      Scanner sc = new Scanner(System.in);
      int n = 4;
      System.out.println("Sample TDMs created in code for demo.");
      int[][] A = { {0,2,3}, {1,4,5}, {0,6,7}, {0,8,0} }; // example rows
      int[][] B = { {0,1,0}, {2,3,4}, {0,5,6}, {0,7,0} };
      printTDM(A);
      printTDM(B);
      int[][] C = mulTDMs(A,B);
      printRES(C);
    }
}
```

Reference: matrix multiplication fundamentals and optimized storage ideas from your matrix lecture.

algorithm-analysis-and-divide-n…

---

5) printActualByte(int n) (Java) — print the actual byte value of an integer n without using byte type

Explanation: emulate the conversion ((n % 256) + 256) % 256 then map to signed range −128..127. This follows simple modular arithmetic (no special PDF citation needed, but chain-of-thought of types is basic Java behavior).

```
// PrintActualByte.java

public class PrintActualByte {

    public static int printActualByte(int n){

        int unsigned = ((n % 256) + 256) % 256; // 0..255

        int signed = (unsigned > 127) ? unsigned - 256 : unsigned; // -128..127

        System.out.println("Actual byte value (simulated) of " + n + " is: " + signed);

        return signed;

    }

    public static void main(String[] args){

        printActualByte(300); // prints 44

        printActualByte(-130); // prints 126

    }

}
```

---

6) Chain Matrix Multiplication (Dynamic Programming) — print optimal order & cost

I implemented the standard DP to compute M[i][j] and P[i][j] (split points). This follows the chain multiplication DP slides in your PDF.

dynamic-programming-approach

```
// MatrixChainOrder.java

import java.util.*;


public class MatrixChainOrder {

    // d[] stores dimensions: Ai is d[i-1] x d[i]
```

```
static void matrixChainOrder(int[] d){
    int n = d.length - 1;
    long[][] m = new long[n+1][n+1];
    int[][] s = new int[n+1][n+1];
    for(int i=1;i<=n;i++) m[i][i]=0;
    for(int L=2; L<=n; L++){
        for(int i=1;i<=n-L+1;i++){
            int j=i+L-1;
            m[i][j]=Long.MAX_VALUE;
            for(int k=i;k<j;k++){
                long q = m[i][k] + m[k+1][j] + (long)d[i-1]*d[k]*d[j];
                if(q < m[i][j]) { m[i][j]=q; s[i][j]=k; }
            }
        }
    }
    System.out.println("Minimum multiplications: " + m[1][n]);
    System.out.print("Optimal order: ");
    printOpt(s,1,n);
    System.out.println();
}

static void printOpt(int[][] s, int i, int j){
    if(i==j) System.out.print("A" + i);
    else{
        System.out.print("(");
        printOpt(s,i,s[i][j]);
        System.out.print(" x ");
        printOpt(s,s[i][j]+1,j);
```

```
            System.out.print(")");
        }
    }

    public static void main(String[] args){
        int[] d = {5,2,3,4,6}; // example from PDF
        matrixChainOrder(d);
    }
}
```

Reference: dynamic programming slides on chained matrix multiplication.

dynamic-programming-approach

---

7) 0–1 Knapsack (DP) — classic bottom-up DP

I wrote a standard DP O(nW) solution. (Slides on branch-and-bound show knapsack for branch/BB; DP is standard.)

```
// Knapsack01.java
import java.util.*;

public class Knapsack01 {
    // weights w[], profits p[], capacity W
    static int knapsack(int[] w, int[] p, int W){
        int n = w.length;
        int[][] dp = new int[n+1][W+1];
        for(int i=1;i<=n;i++){
            for(int wt=0; wt<=W; wt++){
                dp[i][wt] = dp[i-1][wt];
                if(wt >= w[i-1]) dp[i][wt] = Math.max(dp[i][wt], dp[i-1][wt - w[i-1]] + p[i-1]);
            }
```

```
        }
        return dp[n][W];
    }

    public static void main(String[] args){
        int[] p = {40,30,50,10};
        int[] w = {2,5,10,5};
        int W = 16;
        System.out.println("Max profit = " + knapsack(w,p,W));
    }
}
```