```
// selection sort

import java.util.*;

public class SelectionSort{

public static void selectionSort(int num[]) {
int len = num.length;

for(int i=0;i<len-1;i++) {
int index = i;
for(int j=i+1; j<len; j++) {
if(num[index] > num[j]) {
index = j;
}
}
int temp = num[index];
num[index] = num[i];
num[i] = temp;
}
}

public static void main(String args[]) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter size of array: ");
int size = sc.nextInt();

System.out.println("Enter numbers into the array-");
int num[] = new int[size];
for(int i=0; i<size; i++) {
System.out.print("num["+i+"] = ");
num[i] = sc.nextInt();
}

System.out.println("Unsorted array - ");
for(int i=0; i<size; i++) {
System.out.print(" "+num[i]);
}
System.out.println();

selectionSort(num);
System.out.println("Sorted array - ");
for(int i=0; i<size; i++) {
System.out.print(" "+num[i]);
}
}
}

// job scheduling


import java.util.*;

class Job{
int id,deadline,profits;

public Job(int id, int deadline, int profits) {
this.id = id;
this.deadline = deadline;
this.profits = profits;
}
}

public class JobScheduling{

static int getMaxDeadline(Job jobs[]) {
int max = 0;
for(int i=0; i<jobs.length; i++) {
Job job = jobs[i];
max = Math.max(max, job.deadline);
}
return max;
}
```

```java
static void scheduleJob(Job jobs[], Job scheduleJobs[]) {
Arrays.sort(jobs, (a,b)->(b.profits - a.profits));

for(int i=0;i<jobs.length;i++) {
Job job = jobs[i];
for(int j=job.deadline-1;j>=0;j--) {
if(scheduleJobs[j] == null) {
scheduleJobs[j] = job;
break;
}
}
}
}

static int calculateProfit(Job jobs[]) {
int total = 0;
for(int i=0; i<jobs.length; i++) {
Job job = jobs[i];
total += job.profits;
}
return total;
}

public static void main(String args[]) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter number of jobs: ");
int n = sc.nextInt();

Job jobs[] = new Job[n];
for(int i=0;i<n; i++) {
System.out.println("Enter details for job "+(i+1));
System.out.print("ID - ");
int id = sc.nextInt();
System.out.print("Deadline - ");
int deadline = sc.nextInt();
System.out.print("Profits - ");
```

```
 int profits = sc.nextInt();

 jobs[i] = new Job(id, deadline, profits);

 }


 int maxDeadline = getMaxDeadline(jobs);

 Job scheduleJobs[] = new Job[maxDeadline];


 scheduleJob(jobs, scheduleJobs);


 int totalProfit = calculateProfit(scheduleJobs);

 System.out.println("Schedule jobs: ");

 for(int i=0; i<scheduleJobs.length; i++) {

 Job job = scheduleJobs[i];

 System.out.println("Job ID - "+job.id+", Deadline - "+job.deadline+", Profits - "+job.profits);

 }


 System.out.println("Total Profits = "+totalProfit);

 sc.close();

 }

}
```

```
// Kruskal


 import java.util.*;


 public class KruskalMST{

  public class Edge implements Comparable<Edge>{

  int src,dest,weight;


  public int compareTo(Edge other) {

  return this.weight - other.weight;

  }

  }


  public void kruskal(int graph[][], int vertices) {

  List<Edge> edges = new ArrayList<>();
```

```
for(int i=0; i<vertices; i++) {
for(int j=i+1; j<vertices; j++) {
if(graph[i][j] != 0) {
Edge edge = new Edge();
edge.src = i;
edge.dest = j;
edge.weight = graph[i][j];
edges.add(edge);
}
}
}

int parent[] = new int[vertices];
for(int i=0; i<vertices; i++) {
parent[i] = i;
}

List<Edge> mst = new ArrayList<>();
for(Edge edge:edges) {
int srcParent = find(parent, edge.src);
int destParent = find(parent, edge.dest);
if(srcParent != destParent) {
mst.add(edge);
parent[srcParent] = destParent;
}
}

int sum = 0;
System.out.println("Edges in MST :");
for(Edge edge:mst) {
System.out.println(edge.src+" - "+edge.dest+" : "+edge.weight);
sum += edge.weight;
}
System.out.println("Weight of MST is : "+sum);
}

private int find(int parent[], int i) {
 if(parent[i] != i) {
 parent[i] = find(parent, parent[i]);
 }
 return parent[i];
 }

 public static void main(String args[]) {
 Scanner sc = new Scanner(System.in);
 System.out.println("enter number of vertices: ");
 int n = sc.nextInt();

 int graph[][] = new int[n][n];
 for(int i=0; i<n; i++) {
 for(int j=0; j<n; j++) {
 System.out.println("Enter the weight "+i+" ->"+j+" of the graph");
 graph[i][j] = sc.nextInt();
 }
 }

 KruskalMST k = new KruskalMST();
 k.kruskal(graph, n);
 }
}

//prims


 import java.util.*;

 public class PrimsMST{
  static class Edge{
  String v1, v2;
  int w;

  Edge(String v1,String v2,int w){
  this.v1 = v1;
  this.v2 = v2;
this.w = w;
}

public String toString() {
return "("+ v1 + ", "+v2+", "+w+")";
}
}

static class MstResult{
HashSet<Edge> edges;
int totalWeight = 0;
MstResult(HashSet<Edge> e, int tw){
this.edges = e;
this.totalWeight = tw;
}
}

static MstResult prims(HashMap<String,HashMap<String, Integer>> graph, String startVertex) {
HashSet<String> visited = new HashSet<>();
visited.add(startVertex);
int totalWeight = 0;
HashSet<Edge> mstEdges = new HashSet<>();

while(visited.size() < graph.size()) {
Edge minEdge = null;
int minWeight = Integer.MAX_VALUE;

for(String vertex:visited) {
for(Map.Entry<String, Integer> entry:graph.get(vertex).entrySet()) {
String neighbour = entry.getKey();
int weight = entry.getValue();
if(!visited.contains(neighbour) && weight < minWeight) {
minWeight = weight;
minEdge = new Edge(vertex,neighbour,weight);
}
}
}
```

```
 mstEdges.add(minEdge);
 totalWeight += minEdge.w;
 visited.add(minEdge.v2);
 }
 return new MstResult(mstEdges, totalWeight);
 }

 public static void main(String args[]) {
 Scanner sc = new Scanner(System.in);
 System.out.println("Enter number of edges: ");
 int n = sc.nextInt();
 sc.nextLine();

 HashMap<String, HashMap<String, Integer>> graph = new HashMap<>();
 for(int i=0; i<n; i++) {
 System.out.println("Enter edge and weight (vertex1 vertex2 weight):");
 String input[] = sc.nextLine().split(" ");
 String v1 = input[0];
 String v2 = input[1];
 int w = Integer.parseInt(input[2]);

 graph.computeIfAbsent(v1, k ->new HashMap<>()).put(v2, w);
 graph.computeIfAbsent(v2, k ->new HashMap<>()).put(v1, w);
 }

 System.out.println("Enter the Starting Vertex: ");
 String startVertex = sc.nextLine();

 MstResult p = prims(graph, startVertex);
 System.out.println("Minimal spanning tree: ");
 for(Edge edge:p.edges) {
 System.out.println(edge);
 }
 System.out.println("total weight of tree: "+p.totalWeight);
 }
}
```

```java
// graphColoring


import java.util.*;

public class GraphColoring{
 private final int graph[][];
 private final int vertices;
 private int colors[];
 private int minColors;

 public GraphColoring(int vertices, int graph[][]) {
 this.vertices = vertices;
 this.graph = graph;
 this.colors = new int[vertices];
 this.minColors = Integer.MAX_VALUE;
 }

 public void solve() {
 Arrays.fill(colors, -1);
 tryColoring(0,1);
 }

 private void tryColoring(int vertex, int numColors) {
 if(numColors >= minColors) {
 return; //branch and bound
 }

 if(vertex == vertices) {
 minColors = numColors;
 printSolution(minColors);
 return;
 }

 for(int i=1;i<=numColors;i++) {
 if(isSafe(vertex, i)) {
```

```
colors[vertex] = i;
tryColoring(vertex + 1, numColors);
colors[vertex] = -1;
}
}

colors[vertex] = numColors+1;
tryColoring(vertex + 1, numColors+1);
colors[vertex] = -1;
}

private boolean isSafe(int vertex , int color) {
for(int i=0; i<vertices; i++) {
if(graph[vertex][i]==1 && colors[i] == color) {
return false;
}
}
return true;
}

private void printSolution(int numColors) {
System.out.println("solution found with "+numColors+" colors");
for(int i=0; i<vertices ; i++) {
System.out.println("Vertex "+i+" --> color "+colors[i]);
}
}

public static void main(String args[]) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter number of vertices: ");
int vertices = sc.nextInt();

int graph[][] = new int[vertices][vertices];
System.out.println("Enter adjacency matrix: ");

for(int i=0;i<vertices;i++) {
for(int j=0;j<vertices;j++) {
 graph[i][j] = sc.nextInt();
 }
 }

 GraphColoring G1 = new GraphColoring(vertices, graph);
 G1.solve();
 }
}
```

```
//Dijkastra


import java.util.*;

public class DijkstraMST {

    private int numVertices;
    private int[] dist;
    private boolean[] visited;
    private int[][] graph;

    public DijkstraMST(int[][] graph, int numVertices) {
        this.numVertices = numVertices;
        this.graph = graph;
        this.dist = new int[numVertices];
        this.visited = new boolean[numVertices];
    }

    public void dijkstra(int startVertex) {
        for (int i = 0; i < numVertices; i++) {
            dist[i] = Integer.MAX_VALUE;
            visited[i] = false;
        }

        dist[startVertex] = 0;
        for (int i = 0; i < numVertices - 1; i++) {
            int u = minDistance(dist, visited);
        visited[u] = true;

        for (int v = 0; v < numVertices; v++) {
            if (!visited[v] && graph[u][v] != 0 && dist[u] != Integer.MAX_VALUE
                    && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    printMST(startVertex);
}

private int minDistance(int[] dist, boolean[] visited) {
    int minDist = Integer.MAX_VALUE;
    int minIndex = -1;

    for (int i = 0; i < numVertices; i++) {
        if (!visited[i] && dist[i] <= minDist) {
            minDist = dist[i];
            minIndex = i;
        }
    }

    return minIndex;
}

private void printMST(int startVertex) {
    System.out.println("Vertex \t Distance from Source");
    for (int i = 0; i < numVertices; i++) {
        System.out.println(i + "\t" + dist[i]);
    }
}

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Enter the  size of the graph: ");
```

```
        int n = in.nextInt();

        int[][] graph = new int[n][n];

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

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

                System.out.print("Enter the weight " + i + "-> " + j + " of the graph: ");

                graph[i][j] = in.nextInt();

            }

        }

        DijkstraMST dijkstra = new DijkstraMST(graph, n);

        System.out.print("Enter the starting vertex of the graph: ");

        int vertex = in.nextInt();

        dijkstra.dijkstra(vertex);

    }

}
```

```
// DFS BFS
```

```
graph = {
    'A' : ['E' , 'B'],
    'B' : ['C' , 'D' , 'A'],
    'C' : ['E' , 'D'],
    'D' : ['C' , 'B'],
    'E' : ['A', 'C']
}

visited = set()

def dfs(visited,graph,s):
    if s not in visited:
        print(s)
        visited.add(s)

        for child in graph[s]:
            dfs(visited,graph,child)

print("dfs: ")
dfs(visited,graph,'A')

vis = []
queue = []

def bfs(vis,graph,s):
    vis.append(s)
    queue.append(s)

    while queue:
        element = queue.pop(0)
        print(element)

        for child in graph[element]:
            if child not in vis:
                vis.append(child)
                queue.append(child)
```

```
print("bfs: ")
bfs(vis,graph,'A')



// chatbot



import random

def greet():
    greetings = ["Hi there!Welcome to Shopizo!", "Hello!Welcome to Shopizo!", "Hey!Welcome to Shopizo!"]
    return random.choice(greetings)

def ask_name():
    return input("What's your name? ")

def chat():
    print(greet())
    name = ask_name()
    print("Nice to meet you, " + name + "! How can I help you?")
    while True:
        user_input = input("You: ")
        if user_input.lower() == 'exit':
            print("Shopizo: Goodbye!")
            break
        else:
            response = generate_response(user_input)
            print("Shopizo:", response)

def generate_response(user_input):
    responses = {
        "i want to track my order": "ok!Can you tell me order number?",
        "i want to update my contact number": "Sure!Enter your new contact number",
        "i want to have a call with your authority": "OK.You will receive a call soon!",
        "when Will I receive my order": "Tell me your Order number",
        "i want to update my adress": "Sure sir,tell us your new address.",
        "what is the procedure for exchanging order": "Go to my orders,go to your recent order for exchange,select the option for exhange",
        "what is the processs for return?": "Go to my orders,go to your recent order for exchange,select the option for return",
        "my money after return hasn't deposited yet,please  let me know": "Can I know your account number?",
        "in how many days is product generally delivered": "Generally in less than 4-5 days",
        "will I get exchange in case of damaged products": "Which products can be exchanged is specified at the begining of product.So you will get an exchange at only those products.
        "thank you": "You're welcome!",
        "bye": "Goodbye!",
        "exit": "Goodbye!",

    }
    response = responses.get(user_input.lower(), "I'm sorry, I didn't understand that.")
    return response

if __name__ == "__main__":
    chat()
```