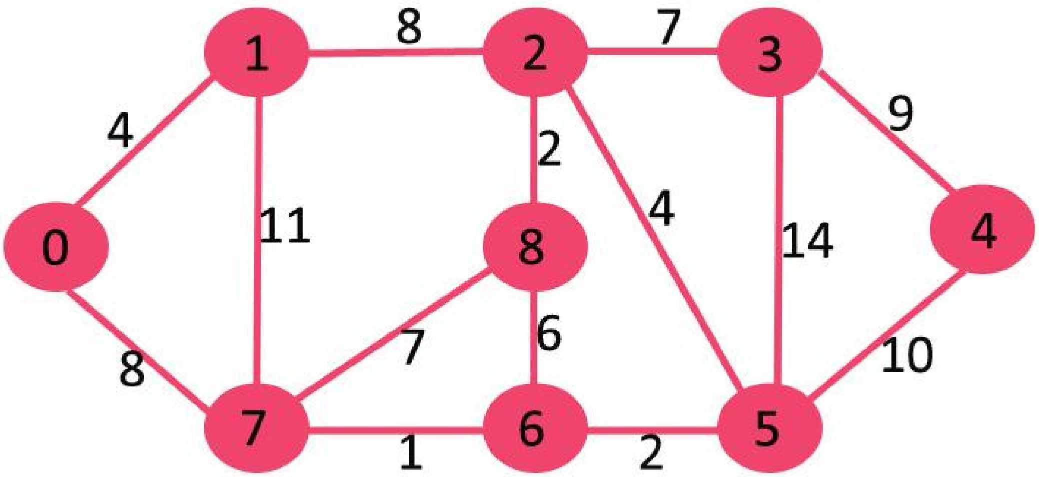
```java
import java.io.*;
import java.lang.*;
import java.util.*;


class Job {
   char job_id;
   int deadline, profit;
   Job(char job_id, int deadline, int profit) {
       this.deadline = deadline;
       this.job_id = job_id;
       this.profit = profit;
   }
}


class PrimsMST {
   private int V, graph[][];

   PrimsMST(int V, int graph[][]) {
       this.V = V;
       this.graph = graph;
   }

   int minKey(int key[], Boolean mstSet[]) {
       int min = Integer.MAX_VALUE, min_index = -1;
       for (int v = 0; v < V; v++)
           if (mstSet[v] == false && key[v] < min) {
               min = key[v];
               min_index = v;
           }
       return min_index;
   }

   void primMST() {
       int parent[] = new int[V];
       int key[] = new int[V];
       Boolean mstSet[] = new Boolean[V];

       for (int i = 0; i < V; i++) {
           key[i] = Integer.MAX_VALUE;
           mstSet[i] = false;
       }

       key[0] = 0;
       parent[0] = -1;

       for (int count = 0; count < V - 1; count++) {
           int u = minKey(key, mstSet);
           mstSet[u] = true;
```

```
            for (int v = 0; v < V; v++)
                if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
        }

        System.out.println("\n\n\nPrim’s Minimum Spanning Tree:\nEdge \tWeight");
        int minimumCost = 0;
        for (int i = 1; i < V; i++){
            System.out.printf("%d -- %d == %d\n", parent[i], i, graph[i][parent[i]]);
            minimumCost += graph[i][parent[i]];
        }
        System.out.printf("Minimum Cost: %d", minimumCost);
    }
}


class KruskalsMST {
    class Edge implements Comparable<Edge> {
        int src, dest, weight;
        public int compareTo(Edge compareEdge) {
            return this.weight - compareEdge.weight;
        }
    };

    class subset {
        int parent, rank;
    };

    private int V, E;
    private Edge edge[];

    KruskalsMST(int v, int e, int graph[][]) {
        this.V = v;
        this.E = e;
        this.edge = new Edge[E];
        int i = -1;
        for (int x = 0; x < v; x++)
            for (int y = x; y < v; y++)
                if (graph[x][y] != 0) {
                    edge[++i] = new Edge();
                    edge[i].src = x;
                    edge[i].dest = y;
                    edge[i].weight = graph[x][y];
                }
    }

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

    void Union(subset subsets[], int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        if (subsets[xroot].rank < subsets[yroot].rank)
            subsets[xroot].parent = yroot;
        else if (subsets[xroot].rank > subsets[yroot].rank)
            subsets[yroot].parent = xroot;
        else {
            subsets[yroot].parent = xroot;
            subsets[xroot].rank++;
        }
    }

    void KruskalMST() {
        Edge result[] = new Edge[V];
        int e = 0, i = 0;

        for (i = 0; i < V; ++i)
            result[i] = new Edge();
        Arrays.sort(edge);

        subset subsets[] = new subset[V];
        for (i = 0; i < V; ++i)
            subsets[i] = new subset();

        for (int v = 0; v < V; ++v) {
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }

        i = 0;

        while (e < V - 1) {
            Edge next_edge = edge[i++];

            int x = find(subsets, next_edge.src);
            int y = find(subsets, next_edge.dest);

            if (x != y) {
                result[e++] = next_edge;
                Union(subsets, x, y);
            }
        }
```

```
       System.out.println("\n\n\nKruskal’s Minimum Spanning Tree:\nEdge \tWeight");
       int minimumCost = 0;
       for (i = 0; i < e; ++i) {
           System.out.printf("%d -- %d == %d\n", result[i].src, result[i].dest,
result[i].weight);
           minimumCost += result[i].weight;
       }
       System.out.printf("Minimum Cost: %d", minimumCost);
   }
}


class GreedySearchAlgorithms {

   static void selectionSort(int[] A) {
       int[] U = A.clone();
       int n = A.length;
       for (int i = 0; i < n - 1; i++) {
           int min_idx = i;
           for (int j = i + 1; j < n; j++) {
               if (A[j] < A[min_idx]) {
                   min_idx = j;
               }
           }
           int tmp = A[i];
           A[i] = A[min_idx];
           A[min_idx] = tmp;
       }

       System.out.printf("Selection Sort:\nUnsorted array: %s\nSorted array: %s",
Arrays.toString(U), Arrays.toString(A));
   }

   static void jobScheduling(ArrayList<Job> arr) {
       int n = arr.size();
       Collections.sort(arr, (a, b) -> {
           return a.deadline - b.deadline;
       });

       ArrayList<Job> result = new ArrayList<>();
       PriorityQueue<Job> maxHeap = new PriorityQueue<>((a, b) -> {
           return b.profit - a.profit;
       });

       for (int i = n - 1; i > -1; i--) {
           int slot_available;
           if (i == 0)
               slot_available = arr.get(i).deadline;
           else
               slot_available = arr.get(i).deadline - arr.get(i - 1).deadline;
```

```
            maxHeap.add(arr.get(i));

            while (slot_available > 0 && maxHeap.size() > 0) {
                Job job = maxHeap.remove();
                slot_available--;
                result.add(job);
            }
        }

        Collections.sort(result, (a, b) -> {
            return a.deadline - b.deadline;
        });

        System.out.print("\n\n\nJob Scheduling Problem:\nFollowing is maximum profit sequence
of jobs: \n");
        for (Job job : result) {
            System.out.printf("[%s, %d, %d] -> ", job.job_id, job.deadline, + job.profit);
        }
        System.out.print("Finish");
    }

    static int minDistance(int[] dist, boolean[] visited) {
        int min = Integer.MAX_VALUE;
        int min_index = -1;

        for (int i = 0; i < dist.length; i++) {
            if (visited[i] == false && dist[i] <= min) {
                min = dist[i];
                min_index = i;
            }
        }
        return min_index;
    }

    static void dijkstra(int[][] graph, int src, int dest) {
        int n = graph.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        HashMap<Integer, ArrayList<Integer>> parent = new HashMap<>();
        List<Integer> path = new ArrayList<>();
        path.add(dest);

        for (int i = 0; i < n; i++) {
            dist[i] = Integer.MAX_VALUE;
            visited[i] = false;
        }

        dist[src] = 0;
        parent.put(src, new ArrayList<>());
```

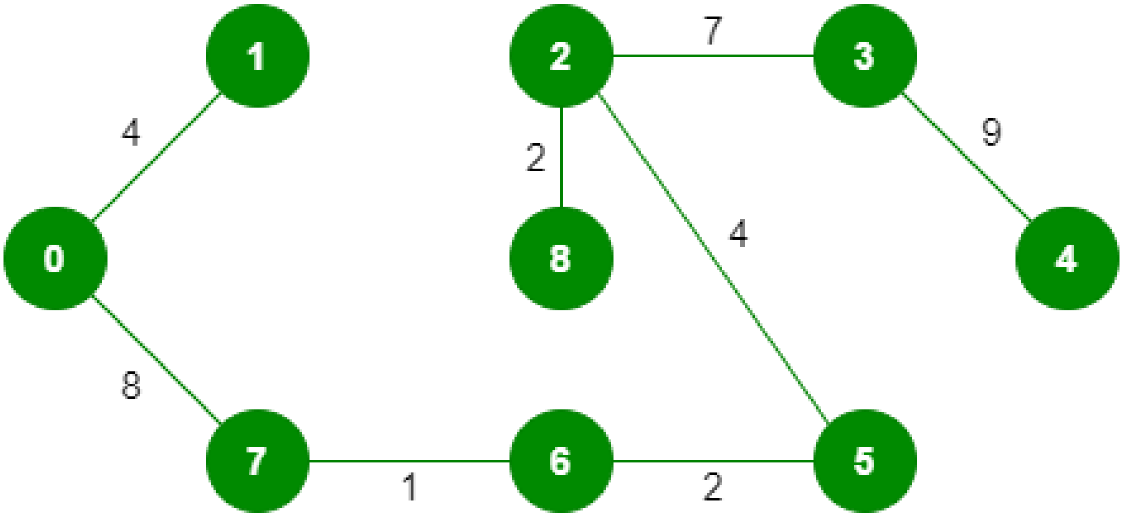
```java
        for (int i = 0; i < n - 1; i++) {
            int u = minDistance(dist, visited);
            visited[u] = true;

            for (int v = 0; v < n; 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];

                    if (!parent.containsKey(v))
                        parent.put(v, new ArrayList<Integer>());
                    parent.get(v).add(u);
                }
            }
        }

        int key = dest;
        while (parent.get(key).size() > 0) {
            int elem = parent.get(key).get(parent.get(key).size()-1);;
            path.add(elem);
            key = elem;
        }
        Collections.reverse(path);

        System.out.printf("\n\n\nDijkstra Single-Source Shortest Path::\nPath: %s\nMinimum Cost: %d", path.toString(), dist[dest]);
    }

    public static void main(String args[]) {

        selectionSort(new int[]{64, 25, 12, 22, 11});

        jobScheduling(new ArrayList<Job>(Arrays.asList(
            new Job('A', 2, 100),
            new Job('B', 1, 19),
            new Job('C', 2, 27),
            new Job('D', 1, 25),
            new Job('E', 3, 15)
        )));
```



```
        int graph[][] = new int[][] {
            { 0, 4, 0, 0, 0, 0, 0, 8, 0},
            { 4, 0, 8, 0, 0, 0, 0,11, 0},
            { 0, 8, 0, 7, 0, 4, 0, 0, 2},
            { 0, 0, 7, 0, 9,14, 0, 0, 0},
            { 0, 0, 0, 9, 0,10, 0, 0, 0},
            { 0, 0, 4,14,10, 0, 2, 0, 0},
            { 0, 0, 0, 0, 0, 2, 0, 1, 6},
            { 8,11, 0, 0, 0, 0, 1, 0, 7},
            { 0, 0, 2, 0, 0, 0, 6, 7, 0}
        };

        PrimsMST primsMST = new PrimsMST(9, graph);
        primsMST.primMST();

        KruskalsMST kruskalsMST = new KruskalsMST(9, 14, graph);
        kruskalsMST.KruskalMST();

        dijkstra(graph, 0, 4);
    }
}
```

OUTPUT:

```
Selection Sort:
Unsorted array: [64, 25, 12, 22, 11]
Sorted array: [11, 12, 22, 25, 64]


Job Scheduling Problem:
Following is maximum profit sequence of jobs:
[A, 2, 100] -> [C, 2, 27] -> [E, 3, 15] -> Finish
```



```
Prim's Minimum Spanning Tree:
Edge    Weight
0 -- 1 == 4
1 -- 2 == 8
2 -- 3 == 7
3 -- 4 == 9
2 -- 5 == 4
5 -- 6 == 2
6 -- 7 == 1
2 -- 8 == 2
Minimum Cost: 37
```

```
Kruskal’s Minimum Spanning Tree:
Edge    Weight
6 -- 7 == 1
2 -- 8 == 2
5 -- 6 == 2
0 -- 1 == 4
2 -- 5 == 4
2 -- 3 == 7
0 -- 7 == 8
3 -- 4 == 9
Minimum Cost: 37


Dijkstra Single-Source Shortest Path::
Path: [0, 7, 6, 5, 4]
Minimum Cost: 21
```