**Rubric**

| Checking if neighbours exists for a vertex | **2** |
| --- | --- |
| Deadend condition handled | **2** |
| Finding max/min edge | **5** |
| Checking if enough points available for next jump | **2** |
| Updating variables for next iteration | **2** |
| Returning result | **2** |
| **Total** | **15** |

**Set A**

**For Adjacency Lis**t

| private static int findMaxEdgeIndex(int[] destinations, int[] weights) {  int maxIndex = -1;  int maxWeight = Integer.MIN\_VALUE;  for (int i = 0; i < destinations.length; i++) {  if (destinations[i] != -1 && weights[i] > maxWeight) {  maxWeight = weights[i];  maxIndex = i;  }  }  return maxIndex;  }  private static int traverseWithMaxEdge(int[][] adjDestinations, int[][] adjWeights, int s, int p) {  int currentVertex = s;  while (true) {  int[] destinations = adjDestinations[currentVertex];  int[] weights = adjWeights[currentVertex];  boolean deadEnd = true;  for (int dest : destinations) {  if (dest != -1) {  deadEnd = false;  break;  }  }  if (deadEnd) {  System.out.println("Reached a dead end ");  break;  }  int maxEdgeIndex = findMaxEdgeIndex(destinations, weights);  if (maxEdgeIndex == -1) break;  int destination = destinations[maxEdgeIndex];  int weight = weights[maxEdgeIndex];  if (weight > p) {  break;  }  p -= weight;  currentVertex = destination;  }  return currentVertex;  } |
| --- |

**For Adjacency Matrix**

| private static int findMaxEdge(int[] neighbors) {  int maxDestination = -1;  int maxWeight = Integer.MIN\_VALUE;  for (int i = 0; i < neighbors.length; i++) {  if (neighbors[i] > 0 && neighbors[i] > maxWeight) {  maxWeight = neighbors[i];  maxDestination = i;  }  }  return maxDestination;  }  private static int traverseMaxEdge(int[][] adjMatrix, int s, int p) {  int currentVertex = s;  int vertices = adjMatrix.length;  while (true) {  int[] neighbors = adjMatrix[currentVertex];  boolean deadEnd = true;  for (int weight : neighbors) {  if (weight > 0) {  deadEnd = false;  break;  }  }  if (deadEnd) {  System.out.println("Reached a dead end");  break;  }  int destination = findMaxEdge(neighbors);  if (destination == -1) break;  int weight = adjMatrix[currentVertex][destination];  if (weight > p) {  break;  }  p -= weight;  currentVertex = destination;  }  return currentVertex;  } |
| --- |

**Set B**

**For Adjacency Lis**t

| private static int findMinEdgeIndex(int[] destinations, int[] weights) {  int minIndex = -1;  int minWeight = Integer.MAX\_VALUE;  for (int i = 0; i < destinations.length; i++) {  if (destinations[i] != -1 && weights[i] < minWeight) {  minWeight = weights[i];  minIndex = i;  }  }  return minIndex;  }  private static int traverseMinEdge(int[][] adjDestinations, int[][] adjWeights, int s, int p) {  int currentVertex = s;  while (true) {  int[] destinations = adjDestinations[currentVertex];  int[] weights = adjWeights[currentVertex];  boolean deadEnd = true;  for (int dest : destinations) {  if (dest != -1) {  deadEnd = false;  break;  }  }  if (deadEnd) {  System.out.println("Reached a dead end");  break;  }  int minEdgeIndex = findMinEdgeIndex(destinations, weights);  if (minEdgeIndex == -1) break;  int destination = destinations[minEdgeIndex];  int weight = weights[minEdgeIndex];  if (weight > p) {  break;  }  p -= weight;  currentVertex = destination;  }  return currentVertex;  } |
| --- |

**For Adjacency Matrix**

| private static int findMinEdge(int[] neighbors) {  int minDestination = -1;  int minWeight = Integer.MAX\_VALUE;  for (int i = 0; i < neighbors.length; i++) {  if (neighbors[i] > 0 && neighbors[i] < minWeight) {  minWeight = neighbors[i];  minDestination = i;  }  }  return minDestination;  }  private static int traverseMinEdge(int[][] adjMatrix, int s, int p) {  int currentVertex = s;  int vertices = adjMatrix.length;  while (true) {  int[] neighbors = adjMatrix[currentVertex];  boolean deadEnd = true;  for (int weight : neighbors) {  if (weight > 0) {  deadEnd = false;  break;  }  }  if (deadEnd) {  System.out.println("Reached a dead end");  break;  }  int destination = findMinEdge(neighbors);  if (destination == -1) break;  int weight = adjMatrix[currentVertex][destination];  if (weight > p) {  break;  }  p -= weight;  currentVertex = destination;  }  return currentVertex;  } |
| --- |