# **Algorithm**

1. START
2. Read input from “storage.txt”.
3. Store data from first line into M and N variables.
4. Create two arrays for values and position of size N.
5. Create a list of edges of size N.
6. Read next line from the file and store data in position array.
7. Read further lines and create new edges.
8. Declare variable high and low with low’s value being 0 and high having Integer.MAX\_VALUE which is 232.
9. Finding values of high, low and mid using binary search algorithm.
10. Check if value of mid is value by calling isValid() method.
11. Use solve() method to recursively find the solution.
12. Assign v[curr] = value.
13. Return true or false based on this value.
14. Finish iterations and store value of low in file “storageout.txt”.
15. END

# **Pseudocode**

**Step 1:** Start

**Step 2:** Read input from “storage.txt”.

**Step 3:** Store data from first line into M and N variables.

**Step 4:** Create two arrays for values and position of size N**.**

**Step 5:** Create a list of edges of size N.

**Step 6:** Initialize all bandwidths from file into position array.

**Step 7:** Initialize while subtracting by 1.

**Step 8:** Create edges and store them in the list according to the bandwidths.

**Step 9:** Declare variable high and low with low’s value being 0 and high having Integer.MAX\_VALUE which is 232­ and use the following method:

while (low != high) {  
 mid = (low + high + 1) / 2;  
 if (*isValid*(mid, v, p, edges)) {  
 low = mid;  
 } else {  
 high = mid - 1;  
 }  
}

**Step 10:** call isValid() method to check if the file is in correct position and if not then new position is assigned to it using following method:

public static void solve(int curr, int value, int low, int[] v, List<Edge>[] edges) {  
 if (v[curr] == value)  
 return;  
 v[curr] = value;  
 for (Edge e : edges[curr]) {  
 if (e.getWeight() >= low) {  
 *solve*(e.getPath(), value, low, v, edges);  
 }  
 }  
}

**Step 11:** Repeat step 9 and 10, until all files are stored correctly.

**Step 12:** Save the value of low in file.

# **Implemented code in Java**

import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Scanner;  
  
public class Lab3 {  
 static class Edge {  
 private final int path, weight;  
  
 public Edge(int path, int weight) {  
 this.path = path;  
 this.weight = weight;  
 }  
  
 public int getPath() {  
 return path;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
 }  
  
 public static void readFile(Scanner sc, int[] p, List<Edge>[] edges, int N, int M) {  
 for (int i = 0; i < N; i++) {  
 edges[i] = new ArrayList<>();  
 }  
  
 String[] tokens = sc.nextLine().split(" ");  
  
 for (int i = 0; i < N; i++) {  
 p[i] = Integer.*parseInt*(tokens[i]);  
 p[i]--;  
 }  
  
 for (int i = 0; i < M; i++) {  
 tokens = sc.nextLine().split(" ");  
 edges[Integer.*parseInt*(tokens[0]) - 1].add(new Edge(Integer.*parseInt*(tokens[1]) - 1, Integer.*parseInt*(tokens[2])));  
 edges[Integer.*parseInt*(tokens[1]) - 1].add(new Edge(Integer.*parseInt*(tokens[0]) - 1, Integer.*parseInt*(tokens[2])));  
 }  
  
 sc.close();  
 }  
  
  
 public static boolean isValid(int l, int[] v, int[] p, List<Edge>[] edges) {  
 Arrays.*fill*(v, -1);  
 int counter = 0;  
  
 for (int i = 0; i < v.length; i++) {  
 if (v[i] < 0) {  
 *solve*(i, counter++, l, v, edges);  
 }  
 }  
  
 for (int i = 0; i < p.length; i++) {  
 if (v[i] != v[p[i]]) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public static void solve(int curr, int value, int low, int[] v, List<Edge>[] edges) {  
 if (v[curr] == value)  
 return;  
 v[curr] = value;  
 for (Edge e : edges[curr]) {  
 if (e.getWeight() >= low) {  
 *solve*(e.getPath(), value, low, v, edges);  
 }  
 }  
 }  
  
 public static void writeFile(int output) throws FileNotFoundException {  
 PrintWriter pw = new PrintWriter(new File("storageout.txt"));  
 pw.println(output);  
 pw.close();  
 }  
  
 //main method for execution  
 public static void main(String[] args) throws Exception {  
 Scanner sc = new Scanner(new File("storage.txt"));  
 String[] tokens = sc.nextLine().split(" ");  
  
 int N = Integer.*parseInt*(tokens[0]);  
 int M = Integer.*parseInt*(tokens[1]);  
  
 int[] p = new int[N];  
 int[] v = new int[N];  
 ArrayList<Edge>[] edges = new ArrayList[N];  
  
 *readFile*(sc, p, edges, N, M);  
  
 int low = 0;  
 int high = Integer.*MAX\_VALUE*;  
 int mid;  
 while (low != high) {  
 mid = (low + high + 1) / 2;  
 if (*isValid*(mid, v, p, edges)) {  
 low = mid;  
 } else {  
 high = mid - 1;  
 }  
 }  
  
 *writeFile*(low);  
 }  
}

# **Complexity**

|  |  |
| --- | --- |
| **Code** | **Complexity** |
| public static void main(String[] args) throws Exception {  Scanner sc = new Scanner(new File("storage.txt"));  String[] tokens = sc.nextLine().split(" ");   int N = Integer.*parseInt*(tokens[0]);  int M = Integer.*parseInt*(tokens[1]);   int[] p = new int[N];  int[] v = new int[N];  ArrayList<Edge>[] edges = new ArrayList[N];   *readFile*(sc, p, edges, N, M);   int low = 0;  int high = Integer.*MAX\_VALUE*;  int mid;  while (low != high) {  mid = (low + high + 1) / 2;  if (*isValid*(mid, v, p, edges)) {  low = mid;  } else {  high = mid - 1;  }  }   *writeFile*(low); } | Nlog2N, including the Read and Write methods. |
| **solve() method:**  public static void solve(int curr, int value, int low, int[] v, List<Edge>[] edges) {  if (v[curr] == value)  return;  v[curr] = value;  for (Edge e : edges[curr]) {  if (e.getWeight() >= low) {  *solve*(e.getPath(), value, low, v, edges);  }  } } | N |
| **isValid() method:**  public static boolean isValid(int l, int[] v, int[] p, List<Edge>[] edges) {  Arrays.*fill*(v, -1);  int counter = 0;   for (int i = 0; i < v.length; i++) {  if (v[i] < 0) {  *solve*(i, counter++, l, v, edges);  }  }   for (int i = 0; i < p.length; i++) {  if (v[i] != v[p[i]]) {  return false;  }  }  return true; } | N2, since complexity of solve() method is N. |
| Total Time complexity= | N+N2+Nlog­2N |

# **Conclusion**

In this project, the job was to find if all the files are in their optimal position and it was possible to do so. The code is provided above and explanation is also provided regarding the algorithm used. DFS search was mainly used which is implemented in solve() method which is recursively searching for optimal solution.