1. using check this vector is be checked or not in the Boolean[] is the True of False.

```
private int V;
private ArrayList<ArrayList<Integer> > adj;
Graph(int v) {
    V = v;
    adj = new ArrayList<>(v);
    for (int i = 0; i < v; ++i)
        adj.add(new ArrayList<>());
}
void addEdge(int v, int w) { adj.get(v).add(w); }
void DAG_tool(int v, boolean[] T_or_F, Stack<Integer> stack){
    T_or_F[v] = true;
    for (Integer integer : adj.get(v)) if (!T_or_F[integer]) DAG_tool(integer, T_or_F, stack);
    stack.push(v);
}
String DAG_topologicalSort() {
    String topologicalSort="";
    Stack<Integer> stack = new Stack<>();
    boolean[] T_or_F = new boolean[V];// meaning is that element visited or not
                                      // if not visited that will be False
    for (int i = 0; i < V; i++) T_or_F[i] = false;
    for (int i = 0; i < V; i++) if (!T_or_F[i]) DAG_tool(i, T_or_F, stack);
    while (!stack.empty()) topologicalSort+=stack.pop()+" ";
    return topologicalSort;
}
```

2. this the Kruskal' s MST is begin connect with the lightest and then keep connect with weight sort ,if have will continue else at to the tree.

```
static class Edge implements Comparable<Edge> {...}
static class subset {...}
int vertices,edges;
Edge[] edge_list;
Graph(int v, int e) {
    vertices = v;
    edges = e;
    edge_list = new Edge[edges];
    for (int i = 0; i < e; ++i)
        edge_list[i] = new Edge();
}
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_for_MST(subset[] subsets, int x, int y) {
    int X_point = find(subsets, x);
    int Y_point = find(subsets, y);
    if (subsets[X_point].rank < subsets[Y_point].rank) subsets[X_point].parent = Y_point;
    else if (subsets[X_point].rank > subsets[Y_point].rank) subsets[Y_point].parent = X_point;
    else {
        subsets[Y_point].parent = X_point;
        subsets[X_point].rank++;
    }
}
void KruskalMST() {
    Edge[] answer = new Edge[vertices];
    for (int i = 0; i < vertices; ++i) answer[i] = new Edge();
    Arrays.sort(edge_list);
    subset[] subsets = new subset[vertices];
    for (int i = 0; i < vertices; ++i) subsets[i] = new subset();
    for (int v = 0; v < vertices; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;}
    int i = 0;
    int buffer_number = 0;
    while (buffer_number < vertices - 1) {
        Edge buffer = edge_list[i++];
        int x = find(subsets, buffer.src);
        int y = find(subsets, buffer.dest);
        if (x != y) {
            answer[buffer_number++] = buffer;
            Union_for_MST(subsets, x, y);
```

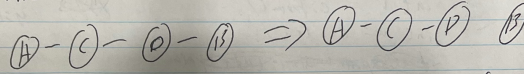
This is the Prim’ s MST at the begin need to change graph to 2D list
For example, 1 don’ t connect with himself that [1][1] is 0 and 1 is connect to 2 weight is 3 thus [1][2] =3. To get all point of graph have thus use 2D list make MST.

```
int minKey(int[] key, Boolean[] T_F_set, int length) {
    int min_value = Integer.MAX_VALUE, min_index = -1;
    for (int i = 0; i < length; i++)
        if (!T_F_set[i] && key[i] < min_value) {
            min_value = key[i];
            min_index = i;
        }

    return min_index;
}
void printMST(int[] parent, int[][] graph) {...}
void primMST(int[][] graph) {
    int length=graph.length;
    int[] parent_list = new int[length];
    int[] key_value = new int[length];
    Boolean[] T_F_set = new Boolean[length];
    for (int i = 0; i < length; i++) {
        key_value[i] = Integer.MAX_VALUE;
        T_F_set[i] = false;
    }
    key_value[0] = 0;
    parent_list[0] = -1;
    for (int count = 0; count < length - 1; count++) {
        int x = minKey(key_value, T_F_set,length);
        T_F_set[x] = true;
        for (int y = 0; y < length; y++)
            if (graph[x][y] != 0 && !T_F_set[y] && graph[x][y] < key_value[y]) {
                parent_list[y] = x;
                key_value[y] = graph[x][y];
            }
    }
    printMST(parent_list, graph);
}
```

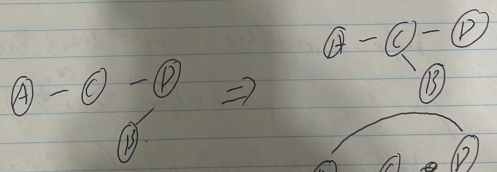
the best way to solve this question is iterative method.
And my logic is:

① if the way of A to B is the shortest MST right now, and remove the last one edge between A and B like this

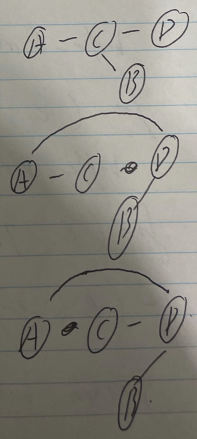


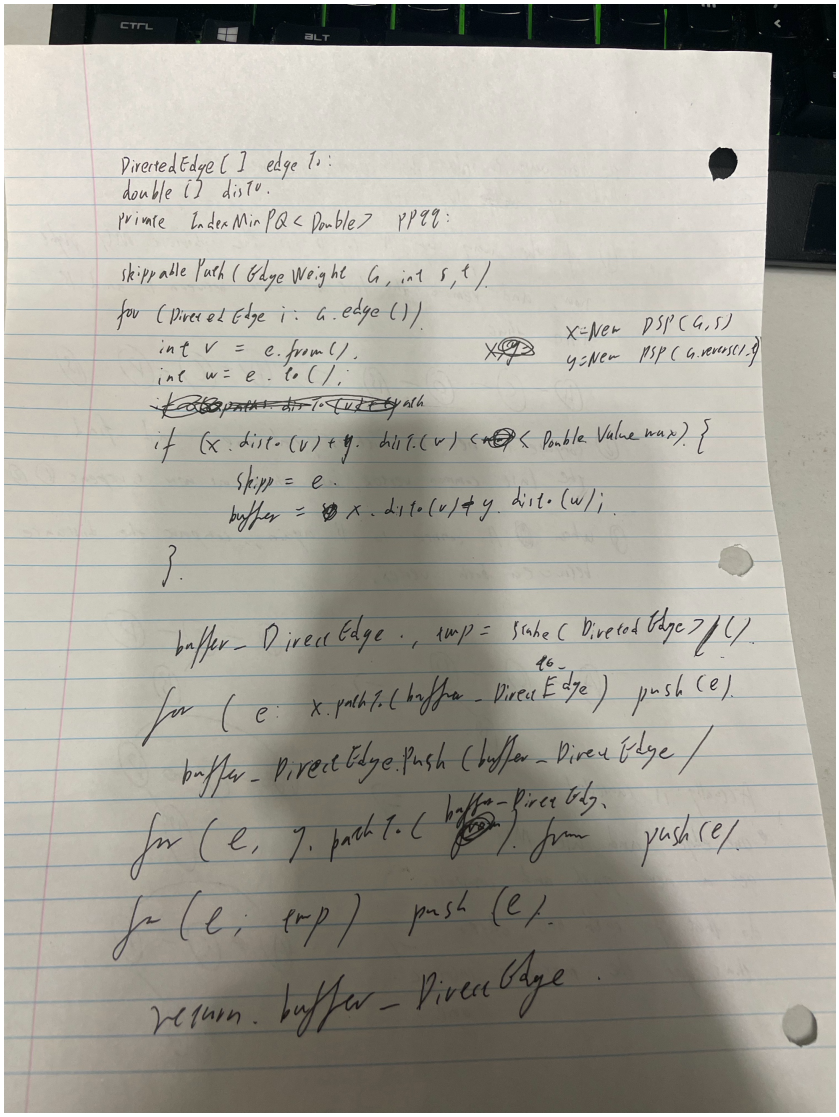
② Compare the new one with original one, and find the last common vertex, use that as new t, repeat ①,②.

③ when A connect to B again, compare the distance between both vertex,



Actually is cancelled one edges and use MST to get a new graph and compare. do that for each one edges then get the maximum increase one.





DirectedEdge[] edgeTo;
double[] distTo;
private IndexMinPQ<Double> pq;

skippable Path(EdgeWeight G, int s, t);
for (DirectedEdge i: G.edge()){
  int v = e.from();
  int w = e.to();
  ~~relax distTo[v] path~~
  if (x.distTo(v) + y.distTo(v) < Double.Value max){
    skipp = e;
    buffer = x.distTo(v) + y.distTo(w);
  }

x=New DSP(G,s)
y=New DSP(G.reverse(),s)

buffer_DirectEdge., tmp = stake(DirectedEdge)();
for (e: x.pathTo(buffer_DirectEdge)) push(e);
buffer_DirectEdge.Push(buffer_DirectEdge);
for (e, y.pathTo(buffer_DirectEdge). from pushe);
for (e; tmp) push(e);
return. buffer_DirectEdge.

4. that is O(|m|+|n|) because that must check each one elements of the String have and make the largest one out.
Like I use “dasdasdasdasd123456721” and
“123456781dasdasdasdasdasdasdas”
He will output 21=1 that meaning at the located 21

```
static HashMap<Integer,String> search(String A, String B){
    HashMap<Integer,String> answer=new HashMap<>();
    for (int i=1;i<Math.min(A.length(),B.length());i++){
        KMPSearch(A.substring(0,i),B,answer);
    }
    return answer;
}
static void KMPSearch(String A, String B,HashMap<Integer,String> answer) {
    int[] buffer_int_list = new int[A.length()];
    LPSintlist(A, A.length(), buffer_int_list);
    int i=0,j = 0;
    while (i < B.length()) {
        if (A.charAt(j) == B.charAt(i)) {
            j++;
            i++;
        }
        if (j == A.length()){
            if ((i-j)+A.length()==B.length()){// check is that is suffix of String B
            answer.put((i-j),B.substring((i-j),(i-j)+A.length()));
            //Integer show the located of String B and String show the same part of String B
            }
            j = buffer_int_list[j - 1];
        }
        else if (i < B.length() && A.charAt(j) != B.charAt(i)) {
            if (j != 0) j = buffer_int_list[j - 1];
            else i ++;
        }
    }
}

static void LPSintlist(String buffer_string, int length, int[] buffer_int_list) {
    int buffer_number = 0,i=1;
    buffer_int_list[0] = 0;
    while (i< length) {
        if (buffer_string.charAt(i) == buffer_string.charAt(buffer_number)) {
            buffer_number++;
            buffer_int_list[i] = buffer_number;
            i++;
        }
        else{
            if (buffer_number != 0) buffer_number = buffer_int_list[buffer_number - 1];
            else {
                buffer_int_list[i] = buffer_number;
                i++;
```

```
{21=1}
```

And I think the easy way of I show also is O(|m|+|n|)

```java
class Longest_suffix_check {
    static int check(String x, String y){
        if (x.equals(y)) return x.length();
        int answer=0;
        for (int i=1;i<Math.min(x.length(),y.length())+1;i++){
            if (x.substring(0,i).equals(y.substring(y.length()-i,y.length()))) answer=i;
        }
        return answer;
    }
```