```
M={5}
Algorithm reduceL2M(x)
        if VL(x) = yes then
                return 5
        else return 1

Algorithm verifyM(x)
        if x = 5 then
                return yes
        else return no

String --> (S, min, max)
Algorithm reduceM2SS(x)
        if verifyM(x) = yes then
                return ({2}, 2, 2)
        else return ({2}, 1, 1)
x --> (G, max)
Algorithm reduceM2MST(x)
        G := new Graph
        v := G.insertVertex("v")
        u := G.insertVertex("u")
        e := G.insertEdge(v, u, 4)
        if verifyM(x) = yes then
                return (G, 4)
        else return return (G, 3)

(G, u, v, max) --> (G, max)
SP(G, u, v, max) ->
Randomly pick edges from G and put into T
Algorithm verifySP(G, u, v, max, T)
        ShortestPath(G, u, v)
        e := getParent(v)
        if e = null then
                return no
        curr := v
        sum := 0
        while curr != u do
                sum := sum + weight(e)
                curr := G.opposite(curr, e)
                e := getParent(curr)
        if sum > max then
                return no
        else return yes
```

```
Algorithm verifySP(G, u, v, max, T)
        ShortestPath(G, u, v)
        if getDistance(v) > max then
                return no
        else return yes


HP --> LP
(G, u, v) -->  (G, u, v, min)

Algorithm reduceHP2LP(G, u, v)
        for each e in G.edges() do
                setWeight(e, 1)
        return (G, u, v, G.numVertices()-1)

randomly pick edges from G and put into T
Algorithm verifyLP(G, u, v, min, T)
        for each e in G.edges() do
                setInT(e, no)
        for each e in T.elements() do
                setInT(e, yes)
        start := u  // subclass field
        setDistance(u, 0)
        BFS(G)
        if getParent(v) = null then
                return no
        if getDistance(v) < min then
                return no
        else return yes


Algorithm isNextComponent(G, v)
        return start = v
Algorithm preDiscEdgeVisit(G, v, e, w)
        setParent(w, e)
        setDistance(w, getDistance(v)+weight(e))
Algorithm postInitVertex(v)
        setDistance(v, -infinity)
        setParent(v, null)

Algorithm postInitEdge(e)
      if getInT(e) = NO then
        setLabel(e, SKIP)
```

```
randomly pick a sequence of vertices and put into T
Algorithm verifyLP(G, u, v, min, T)
        if T.elemAtRank(0) ! = u then
                return NOT_A_SOLN
        if T.last().element() != v then
                return NOT_A_SOLN
        sum := 0
        curr := u
        for i := 1 to T.size()-1 do
                next := T.elemAtRank(i)
                found := false
                for each e in G.incidentEdges(curr) dof
                        if G.opposite(curr, e) = next then
                                sum := sum + weight(e)
                                found := true
                                break
                if !found then
                        return no
                curr := next
        if sum < min then
                return no
        else return yes
```