Aufgabe 2:

Das Nechrichtenalphabet M ist ein Tupel aus stage x UID x N

states_i:

```
u from UID, initially i's UID
stage from {'exploring',
            'test_relax',
            'evaluate_test'
            'convergecasting',
            'preparing_for_merge',
            'merging',
            'leader_relax',
            'updating'}, initially 'exploring'
tree_edges from nbrs, initially {}   //speichert die MST-Kanten
myleader from UID, initially u
UID_to_send from UID, initially u    //UID, die weitergeschickt wird
to_test from nbrs, initially {}   //teste diese auf Komponentenzugehörigkeit
response_for_test from nbrs, initially {}   //erwarten Antwort auf test
broadcast from nbrs, initially {}   //an diese wird broadcast (weiter)geschickt
converge from nbrs, initially {}   //an diese wird convergecast (weiter)geschikt
MWOE from N, initially infinity   //bis jetzt bekannte minimum-Kante
candidate_for_merge from nbrs, initially null
to_merge from nbrs, initially null   //an diesem wird die Merge-Nachfrage geschickt
incoming_converge from N, initially 0   //von wieviel Knoten erwarte ich
                                        //noch convergecast
```

msgs_i:

```
//frage, ob meine MWOE auch für die andere Seite minimal ist
if stage == 'merging' then
   send ('request_for_merge', u, null) to to_merge
   to_merge := null

//teste auf Komponentenzugehörigkeit
if stage == 'test_relax' then
   send ('test', null, null) to all j from to_test
   to_test := {}

//schicke Antwort an alle Knoten, die mich auf Komponentenzugehörigkeit testen
send ('response_for_test', myleader, null) to all j from response_for_test
response_for_test := {}

//Broadcast, bzw. Broadcastweiterleitung
send ('broadcast', UID_to_send, MWOE) to all j from broadcast
broadcast := {}

//Convergecast, bzw. Convergecastweiterleitung
if stage == 'preparing_for_converge'
   send ('converge', UID_to_send, MWOE) to all j from converge   //ist maximal 1
   converge := {}
```

trans_i:

```
//auf Komponentenzugehörigkeit-Anfragen wird imer geantwortet
//zweite und dritte Variablen vom Tupel sind irrelevant.
for each incoming message ('test', -, -)_j do
   response_for_test = response_for_test VEREIN {j}
```

```
switch stage

   //EXPLORING STAGE:
   //Der Lieder informiert durch Broadcast alle Knoten in der Komponente,
   //dass sie mit der Suche nach der minimum-Kante beginnen sollen
   //Zweite und dritte Variablen in der Broadcast-Nachricht sind irrelevant.
   case 'exploring':
      MWOE = infinity
      if u = myleader then
         broadcast := tree_edges   //Leader startet broadcast
         to_test := nbrs - tree_edges   //teste auf Komponentenzugehörigkeit
         stage := 'test_relax'
      else
         //leite broadcast weiter und prüfe auf Komponentenzugehörigkeit
         for each message ('broadcast', -, -)_j do   //ist maximal 1
            broadcast := tree-edges - j
            to_test := nbrs - tree_edges
            stage := 'test_relax'



   //TEST_RELAX STAGE:
   //Wird benötigt, da die test-Nachrichten eine Runde brauchen
   //bis sie von den Nachbarn beantwortet werden.
   case 'test_relax':
      stage := 'evaluate_test'



   //EVALUATE_TEST STAGE:
   //Aus allen außen-Kanten wird die minimale ermittelt.
   //Die Baum-Blätter starten das Convergecast
   //Die dritte Variable der Response-Nachricht ist irrelevant, da
   //jedem Knoten seine Kantengewichte bekannt sind.
   case 'evaluate_test':
      for each incoming message ('response_for_test', UID, -)_j
         if UID_j != myleader   //nicht aus der selben Komponente?
            if weight_ij < MWOE then
               MWOE := weight_ij
               candidte_for_merge = UID_j
      //wenn ich nicht der Leader binn und nur einen MST-Nachbarn
      //habe, dann bin ich ein Blatt und starte convergecast
      if myleader != u AND |tree_edges| == 1 then
         converge := tree_edges
      incoming_converge := |tree_edges|
      stage := 'convergecasting'
```

```
//CONVERGECASTING STAGE:
//Die Information über die minimale Kante wird bis zum Root hochgeschickt
//Die inneren MST-Knoten warten, dass sie von allen außer einem Knoten
//Nachrichten bekommen und schicken seinerseits Nachricht an diesem
//Der Leader wartet bis er Nachrichten von allen Vorgängern bekommt, dann
//broadcastet er das Minimum zurück an allen.
case 'convergecasting':
   if u = myleader then
      if (incoming_converge > 0) then   //Leader wartet auf alle Nachrichten
         for each incoming message ('converge', UID, weight)_j do
            if weight_j < MWOE then
               MWOE := weight_j
               UID_to_send := UID_j
            incoming_converge = incoming_converge - 1
      else
         if UID_to_send != u then   //minimale Kante ist keine Kante vom Leader
            stage := 'updating'
         else
            to_merge := candidate_for_merge
            stage := 'merging'
         broadcast := tree_edges
   //innere Knoten
   else
      if (incoming_converge > 1) then
         for each incoming message ('converge', UID, weight)_j do
            if weight_j < MWOE then
               MWOE := weight_j
               UID_to_send := UID_j
            incoming_converge = incoming_converge - 1
            converge := converge - {j}   //schicke Nachricht hoch
      else
         stage := 'preparing_for_merge'



//PREPARING_FOR_MERGE STAGE:
//Die Information über die MWOE wird „gebroadcastet". Der Knoten, der
//diese Kante besitzt beginnt mit dem Verschmelzen.
case 'preparing_for_merge':
   for each message ('broadcast', UID, weight)_j do // might be max. one
   if UID_j = u then   //meine Kante
        to_merge := candidate_for_merge
        stage := 'merging'
     else
        //schick die Nachricht runter
        UID_to_send := UID_j
        MWOE := weight_j
        broadcast := tree_edges - {j}
        stage := 'updating'
```

```
//MERGING STAGE:
//Wenn die Kante auch für die andere seite minimal ist, werden beide
//Komponenten verschmolzen. Neuer Leader wird der von beiden „Verbindungs-"
//Knoten, der die größere UID hat. Auschliessend wird die Information über
//den neuen Leader an allen Knoten (aus beiden Komponenten) „gebroadcastet".
//Die dritte Variable in der Request-for-merge-Nachricht wier nicht
//benötigt und ist somit irrelevant.
case 'merging':
   for each incoming message ('request_for_merge', UID, -)_j do
      if UID_j = candidate_for_merge then   //Kante ist auch für mich minimal
         to_merge := candidate_for_merge
         tree_edges := tree_edges VEREIN {j}   //Verschmelzen
         //neuen Leader bestimmen und broadcasten
         if u > UID_j then
            myleader := u
            broadcast := tree_edges
            UID_to_send := u
            stage := 'leader_relax'
         else
            stage := 'updating'



//LEADER_RELAX STAGE:
//Wird benötigt, da der Leader eine Runde warten soll, bevor er die
//„Suche-MWOE"-Nachricht broadcastet. Ansonsten könnte sie mit der
//„New-Leader"-Nachricht kollidieren. Die zwei aufeinanderfolgende
//Brodacasts könnten theoretisch auch als Einen implementiert werden,
//der Algorithmus ist aber im Buch mit zwei Broadcasts beschrieben,
//deswegen haben wir es auch so gemacht.
case 'leader_relax'
   to_merge := null
   stage := 'exploring'



//UPDATING STAGE:
//Hier wird die Information über den neuen Leader „gebroadcastet"
//und „upgedatet". Die dritte Variable in der broadcast-Nachricht
//ist irrelevant.
case 'updating':
   for each incoming message ('broadcast', UID, -)_j   //ist maximal 1
      myleader := UID_j
      broadcast := tree_edges - {j}
      stage := 'exploring'
```