Algorithmus zur Bestimmung der minimalen Kantenanzahl zwischen allen erreichbaren Knoten $MINE_G(S)$ für alle N Knoten in S mit Kantengewichten == 1

```
MINERREICHBAR(S, V) {
 Input: Gerichteter Graph S = (P, R, α, ω) in Adjazenmatrixdarstellung (P: Menge der
        Knoten, R: Menge gerichteter Kanten, α: R→V und ω: R→V sind Abbildungen
        (α(r) ist Anfangsknoten und ω(r) ist Endknoten der gerichteten Kante r)); eine
        Kopie V der Adjazenzmatrixdarstellung von S.

 Output: Matrix V mit minimaler Anzahl der Kanten für Weg zwischen Quellknoten
         (Zeilennummer) und Zielknoten (Spaltennummer)

 for d:= 1, ..., N-1 { /* Abstand d als minimale Kantenanzahl zwischen 2 Knoten, die durch
                         Weg verbunden sind (Wellenfront) */
   for each i∈P(S) do {        /* Von allen Knoten i∈P(S) */
      for each j∈P(S) do {     /* Nach allen Knoten j∈P(S) */
         if V[i][j] == d {     /* minimaler Weg mit d Kanten von i nach j existiert ? */
            for each k∈P(S) do {  /* Alle Knoten k∈P(S) */
               if (j,k) ∈ R(S) und v[i][k] == 0  /* Kante (j,k) existiert in S und bisher
                                                    wurde Knoten k nicht erreicht ? */
                  Setze V[i][k] := d + 1  /* Knoten k von i ueber d + 1 Kanten erreichbar */
            }
         }
      }
   }
 }
 return Matrix V mit minimaler Anzahl der Kanten zwischen Quell- und Zielknoten
}
```

Algorithmus zum Kopieren einer Matrix

```
Copy(m[N][N], n[N][N]) {

Input:  Zielmatrix m[N][N],  Quellmatrix n[N][N], N>0

Output: Zielmatrix m[N][N] mit Werten der Quellmatrix n[N][N]

   for each i∈P(S) do {        /* Von allen Knoten i∈P(S) */
      for each j∈P(S) do {     /* Nach allen Knoten j∈P(S) */
         m[i][j] := n[i][j]
      }
   }
   return Matrix m[N][N]
}
```

Algorithmus zur Erreichbarkeit aller Knoten eines Graphen seitens aller Knoten

```
int Check(v[N][N]) {

  Input:   Matrix v[N][N] minimaler Anzahl Kanten für Weg von Knoten i nach j,  N>0,

  Output: 0, wenn mindestens ein Knoten nicht erreichbar, sonst 1

    for each i∈P(S) do {          /* Von allen Knoten i∈P(S) */
        for each j∈P(S) do {      /* Nach allen Knoten j∈P(S) */
            if v[i][j] == 0       /* Zwischen Knoten i und j existiert kein Weg ? */
               return 0
        }
    }
    return 1                      /* Zwischen allen Knoten existiert ein Weg */
}
```

Algorithmus zur Ermittlung der maximalen Kantenzahl

```
int longest(v[N][N]) {

  Input:   Matrix v[N][N] minimaler Anzahl Kanten für Weg von Knoten i nach j,  N>0,

  Output: Maximale Kantenzahl in v
    Setze l: = 0
    for each i∈P(S) do {          /* Von allen Knoten i∈P(S) */
        for each j∈P(S) do {      /* Nach allen Knoten j∈P(S) */
            if v[i][j] > l        /* Zwischen Knoten i und j existiert längerer Weg ? */
               Setze l := v[i][j] /* Aktualisierung maximale Kantenzahl */
        }
    }
    return l                      /* Maximale Kantenanzahl */
}
```