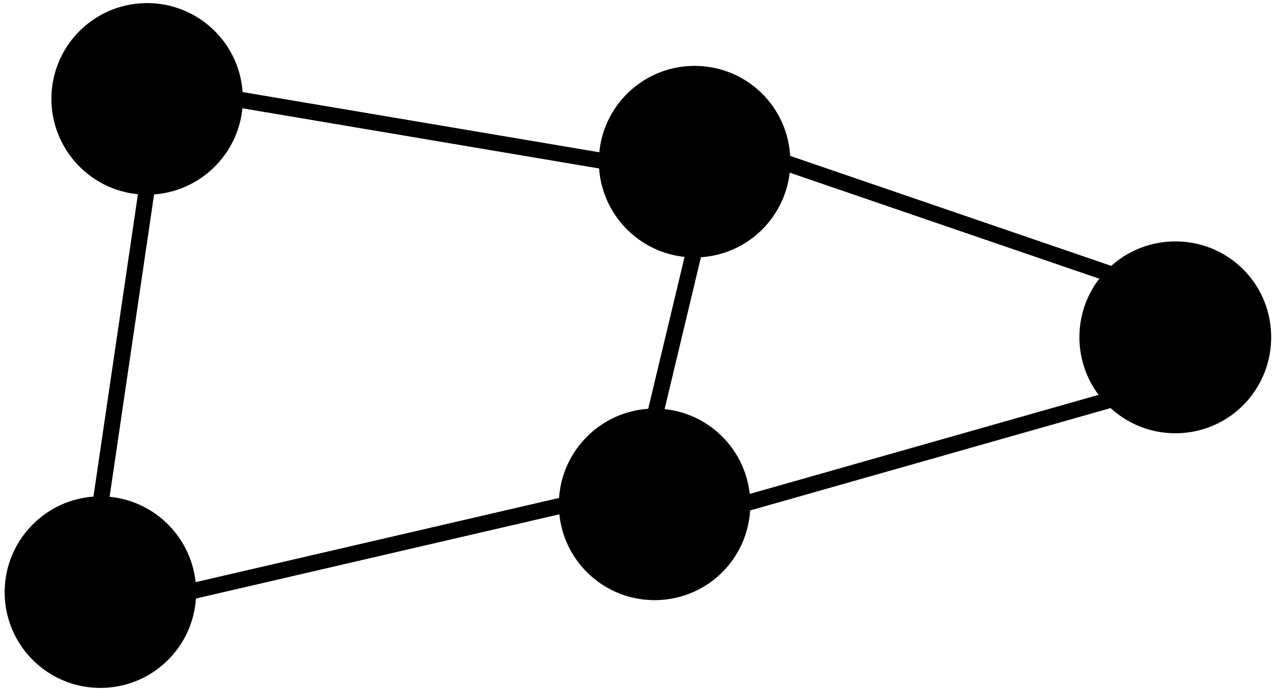
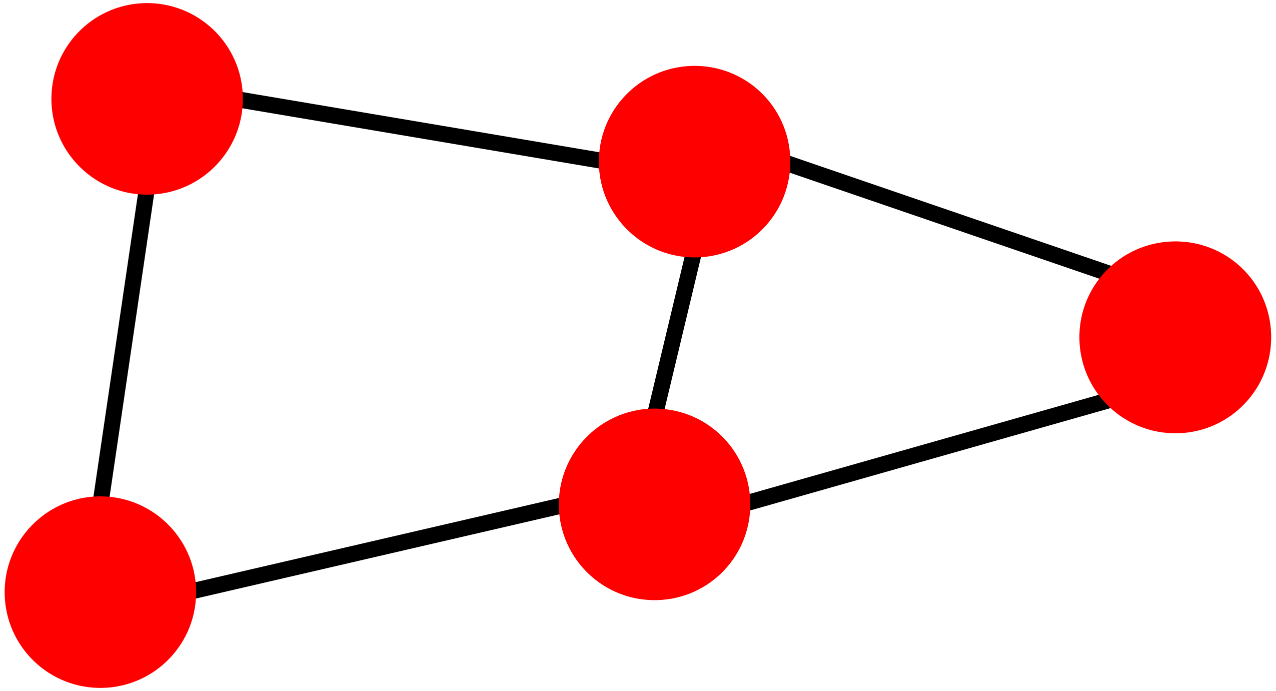
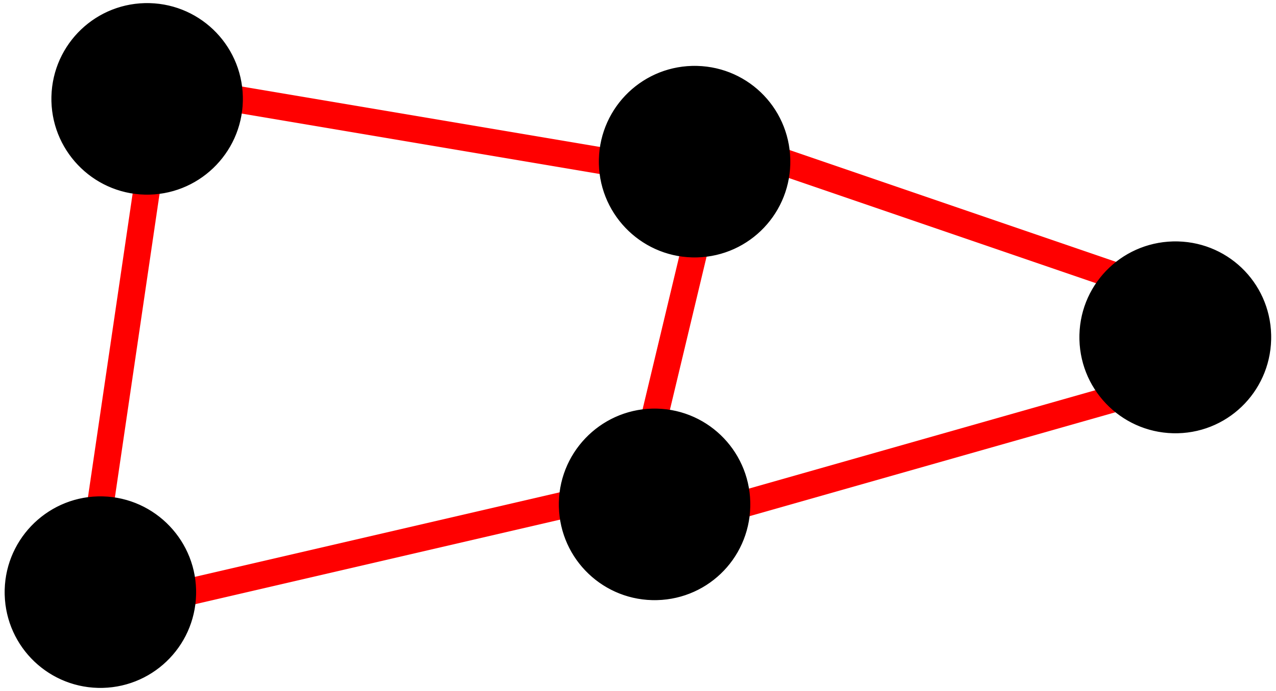
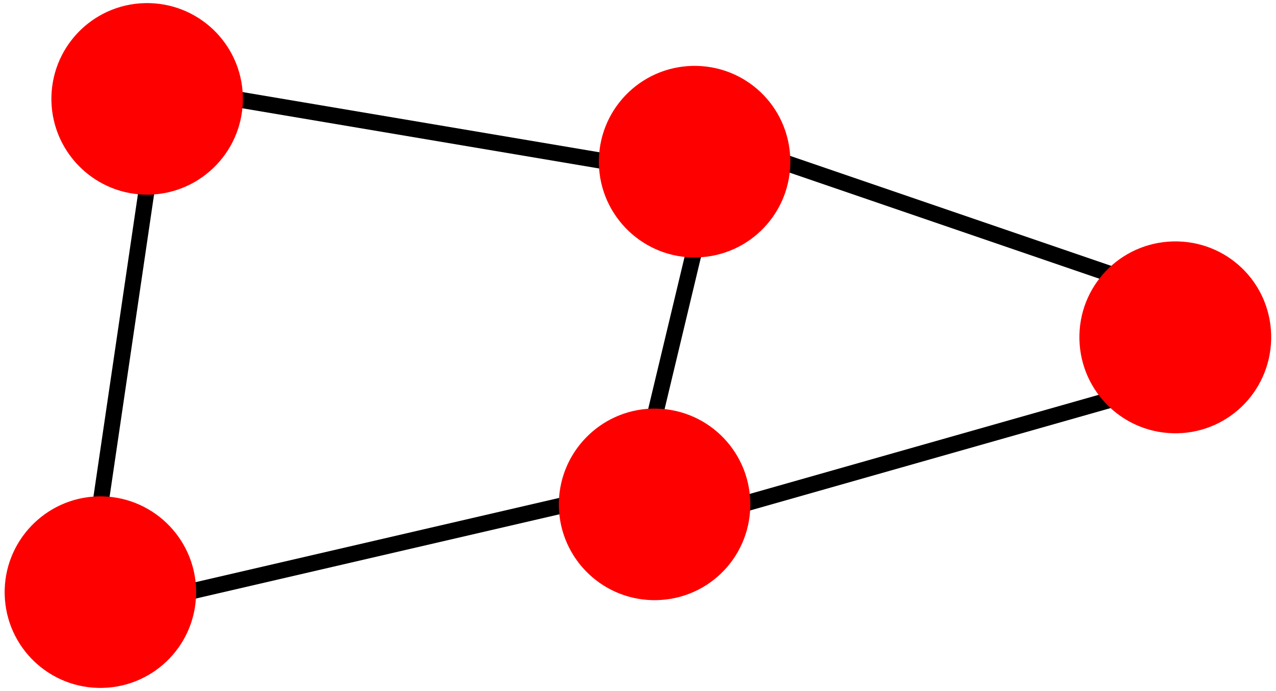
Graphentheorie
und
Routenplanung

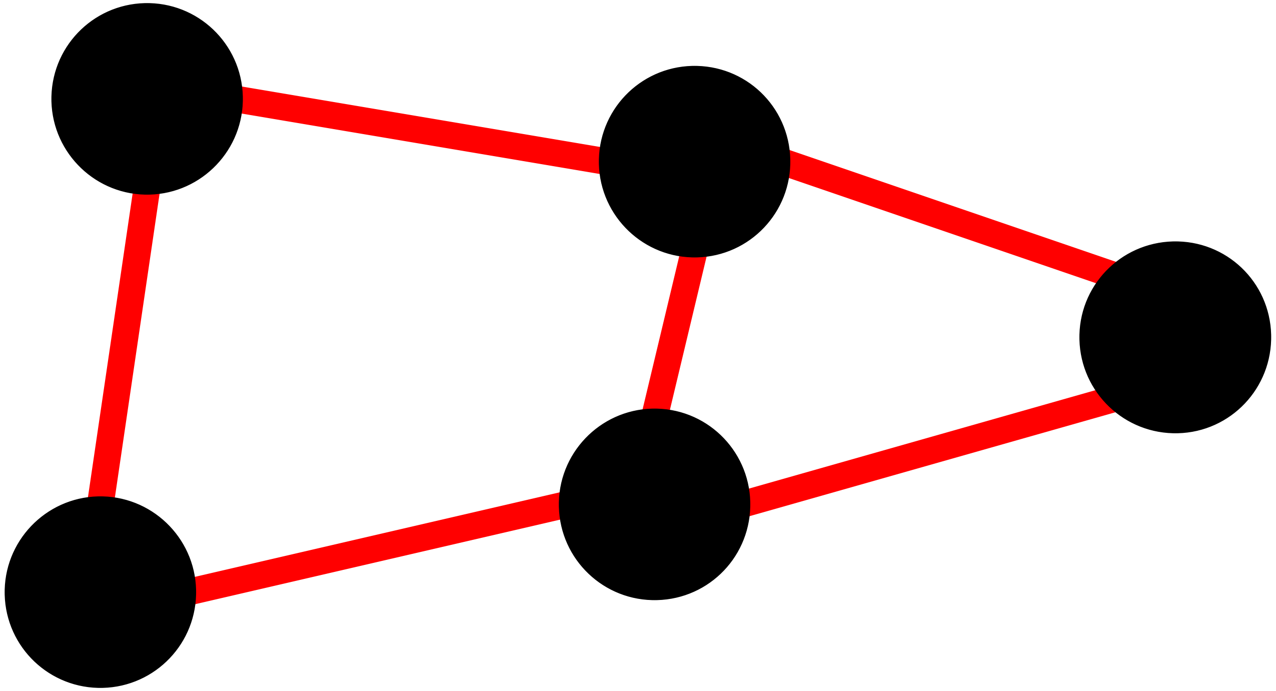
Graphen

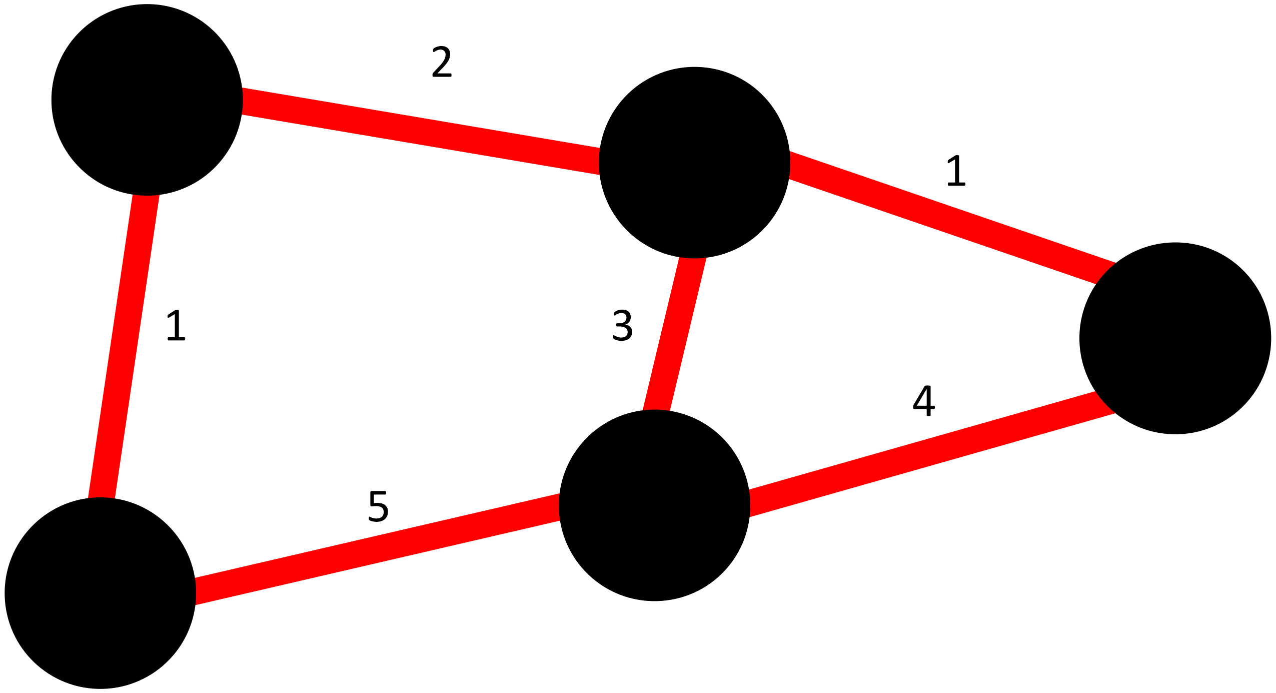




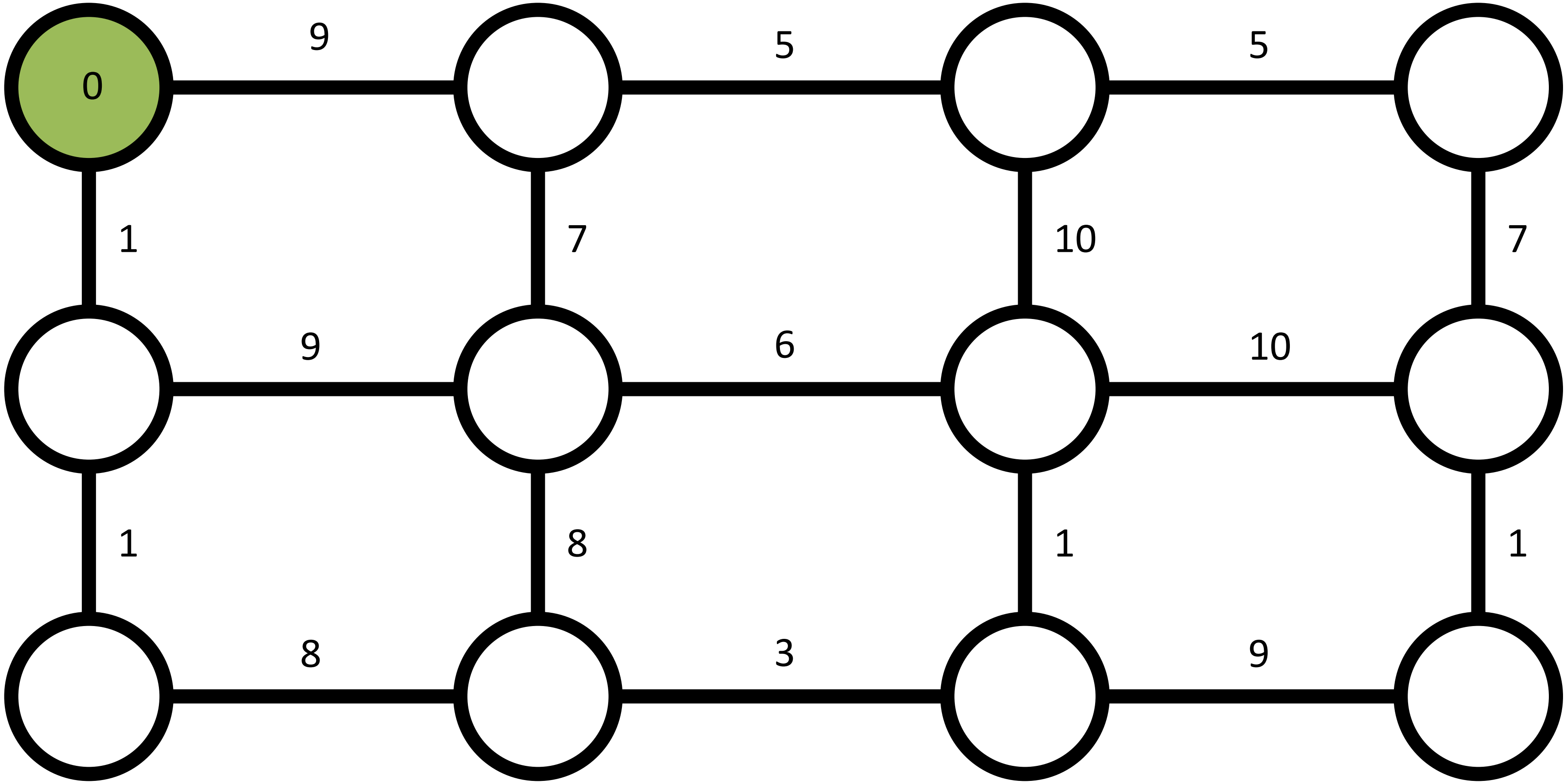


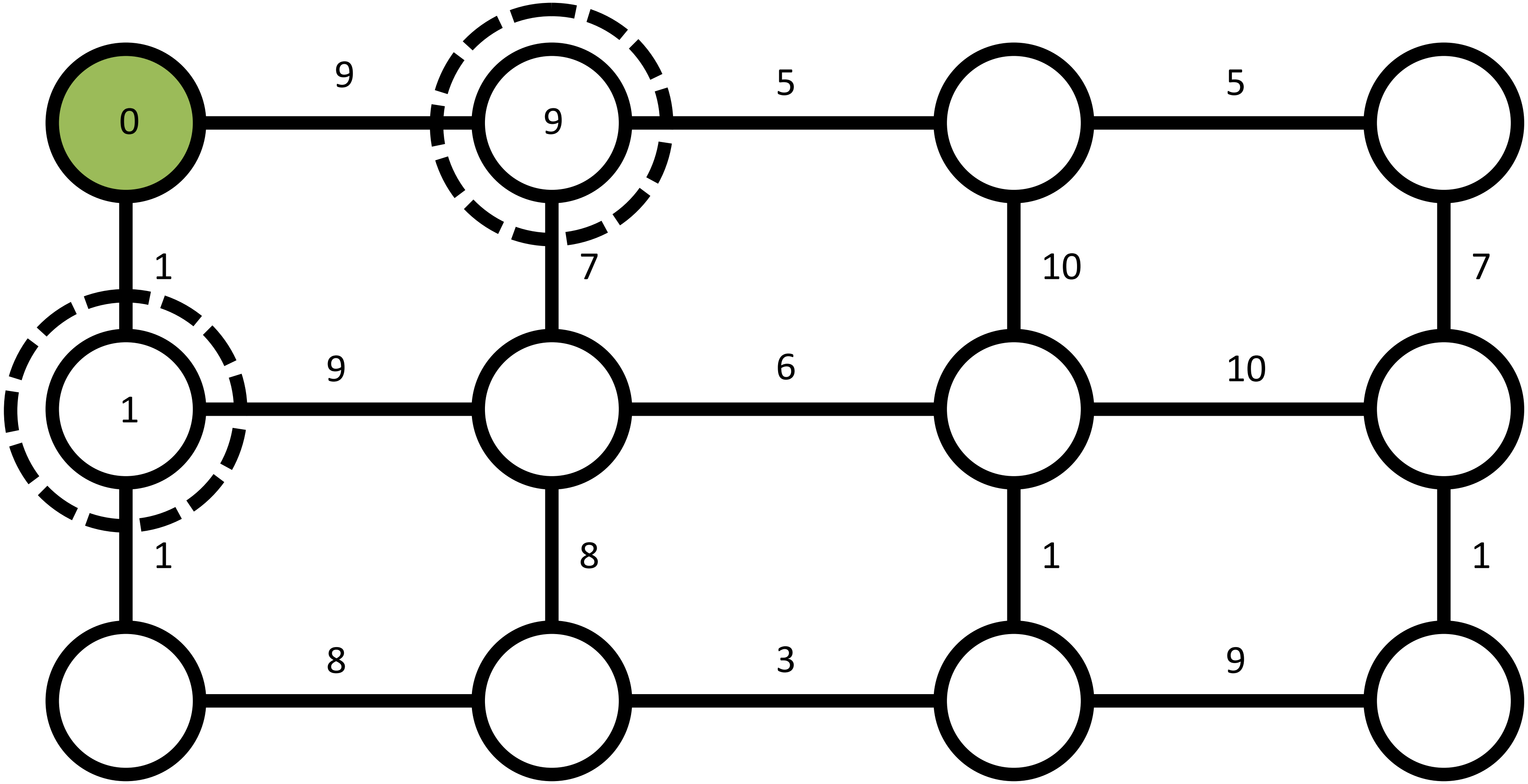


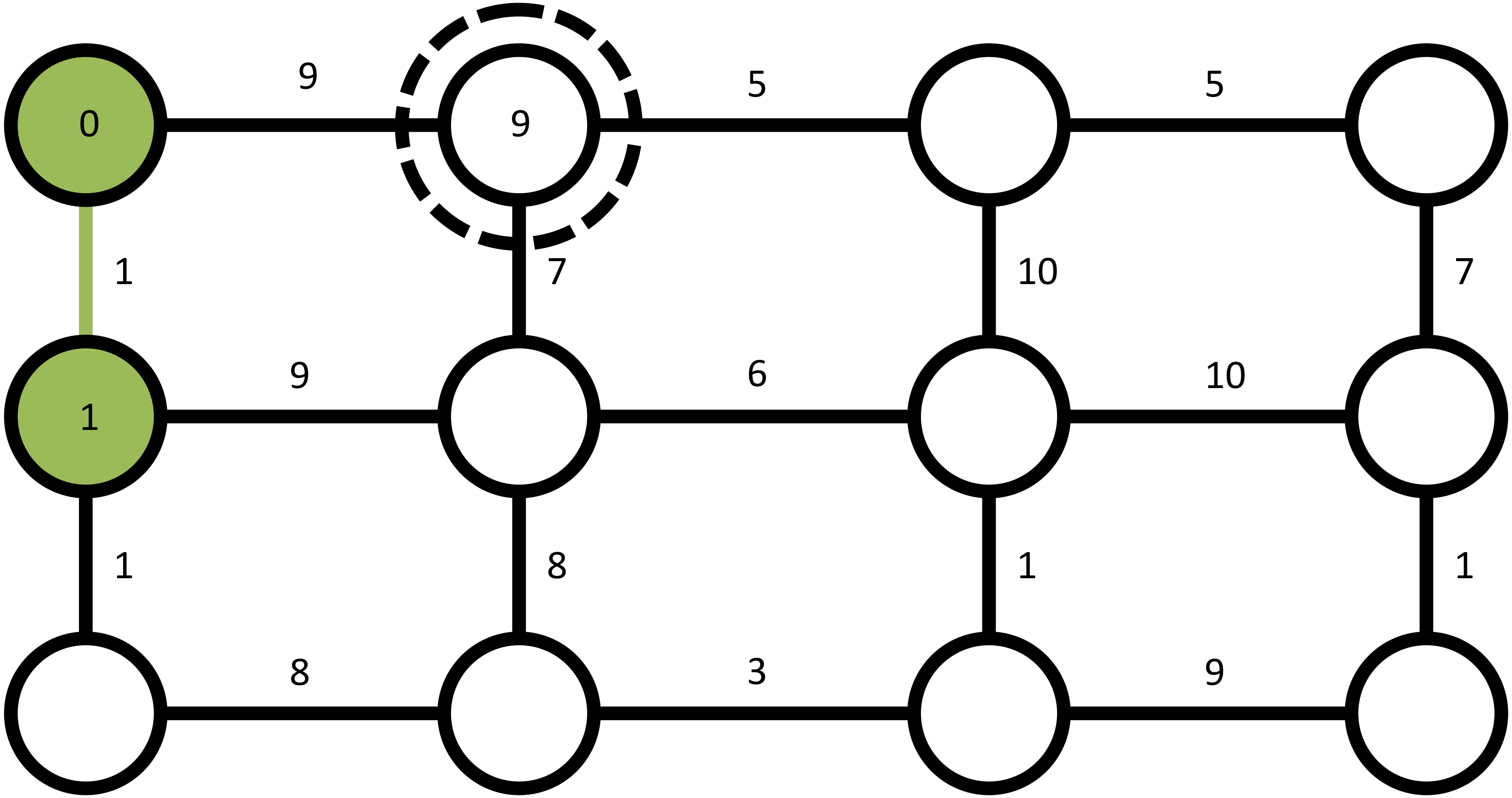


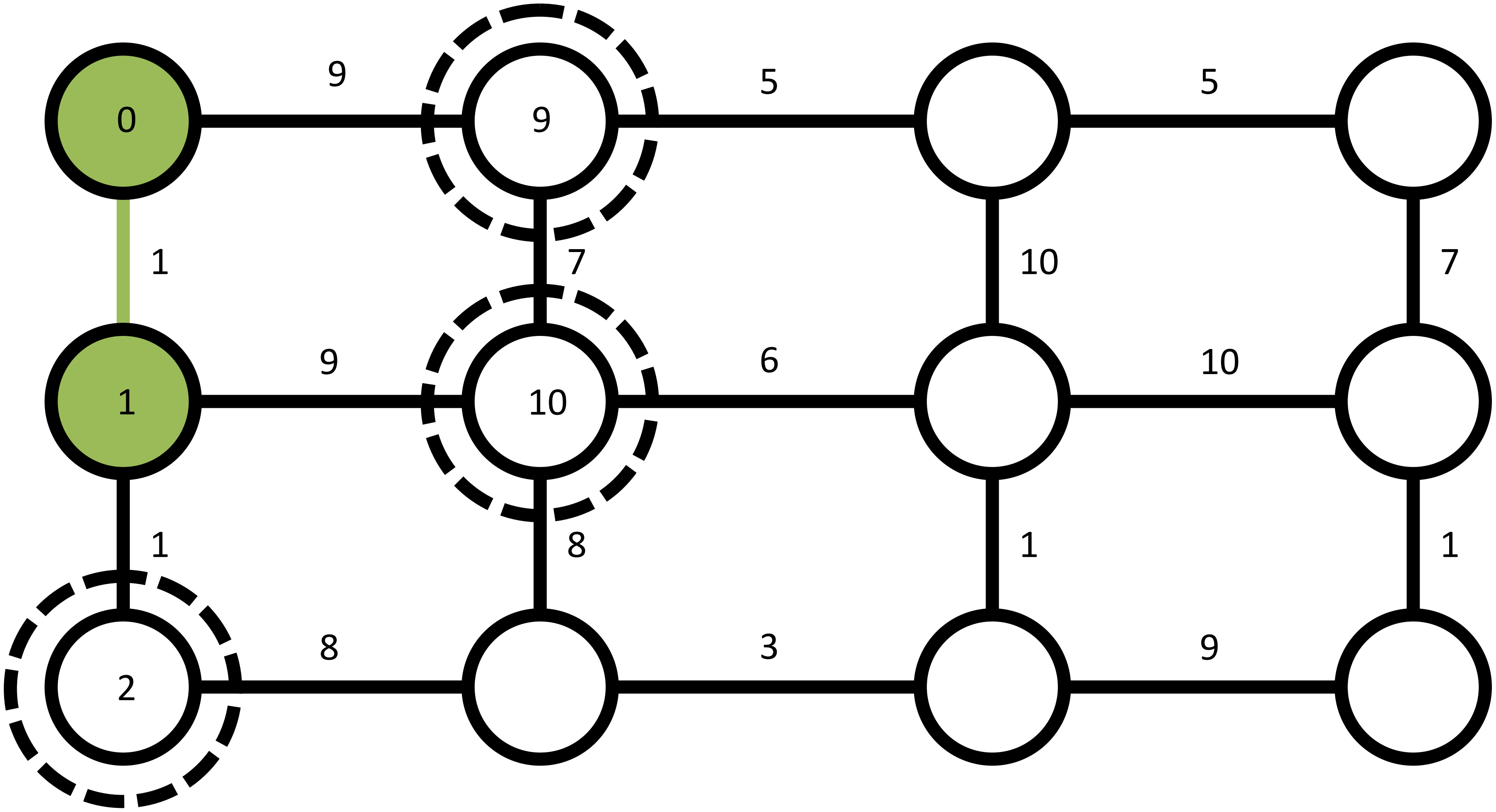

2
1
1
3
4
5

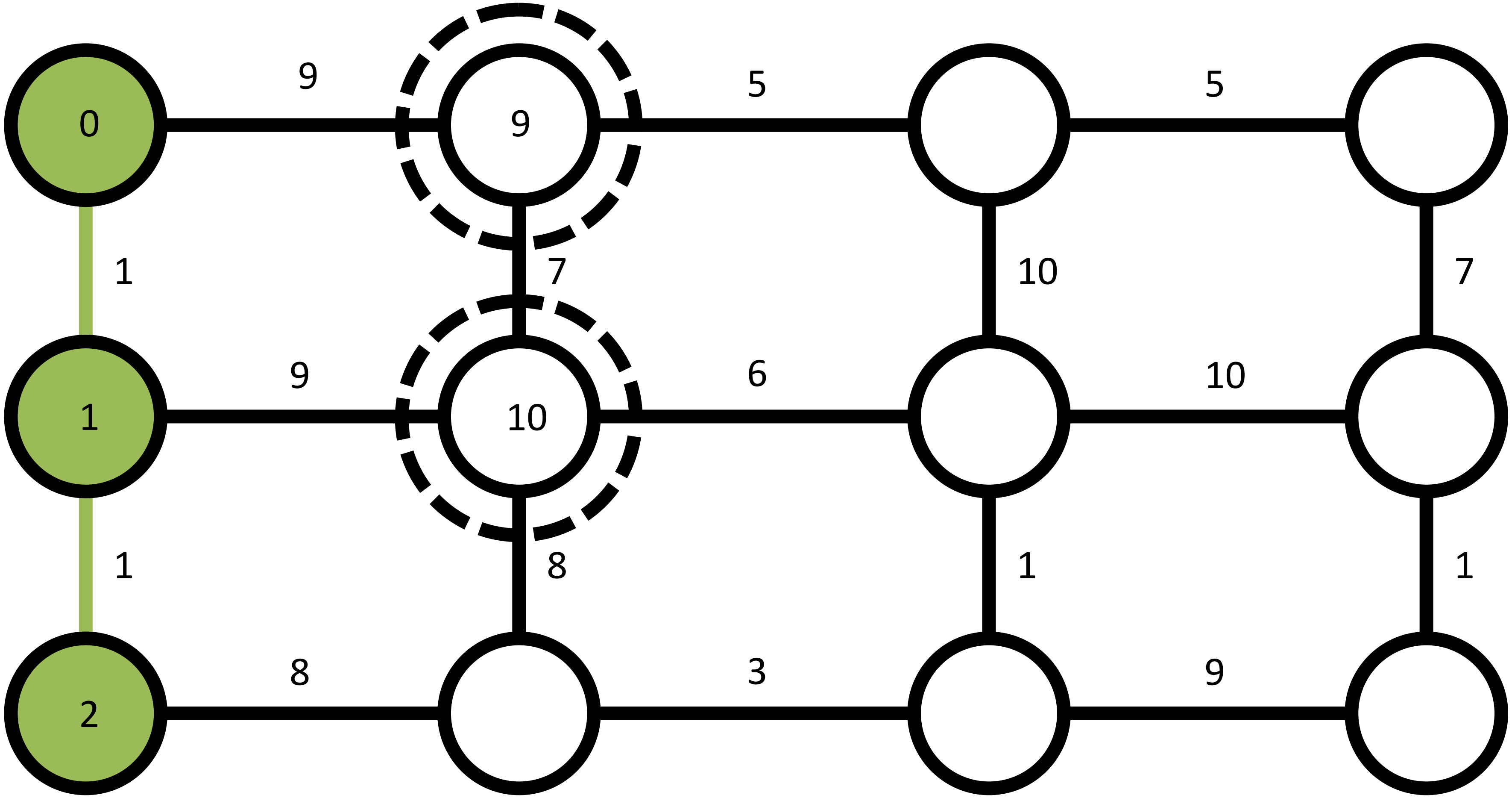
Dijkstra Algorithmus

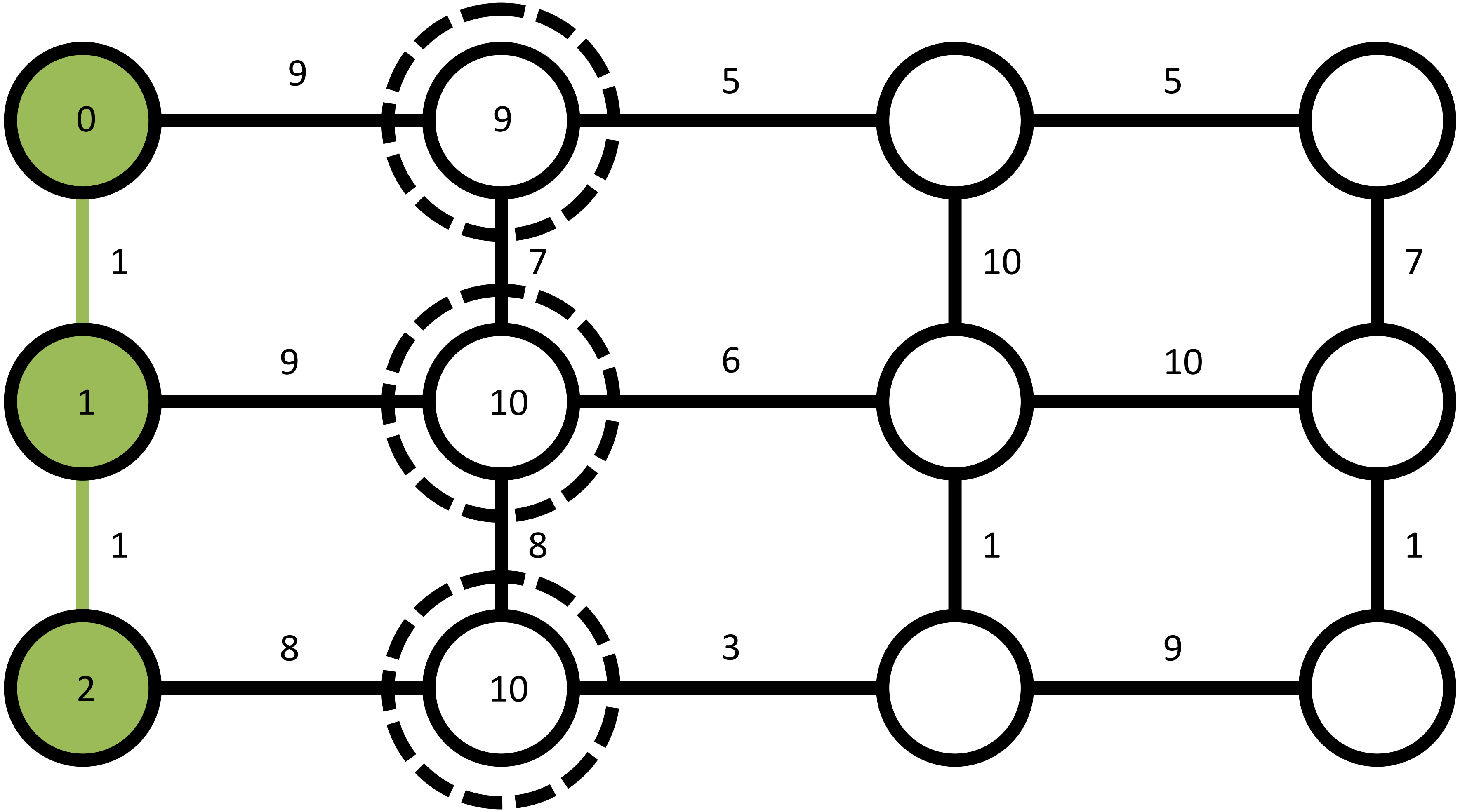

0
9
5
5
1
7
10
7
9
6
10
1
8
1
1
8
3
9

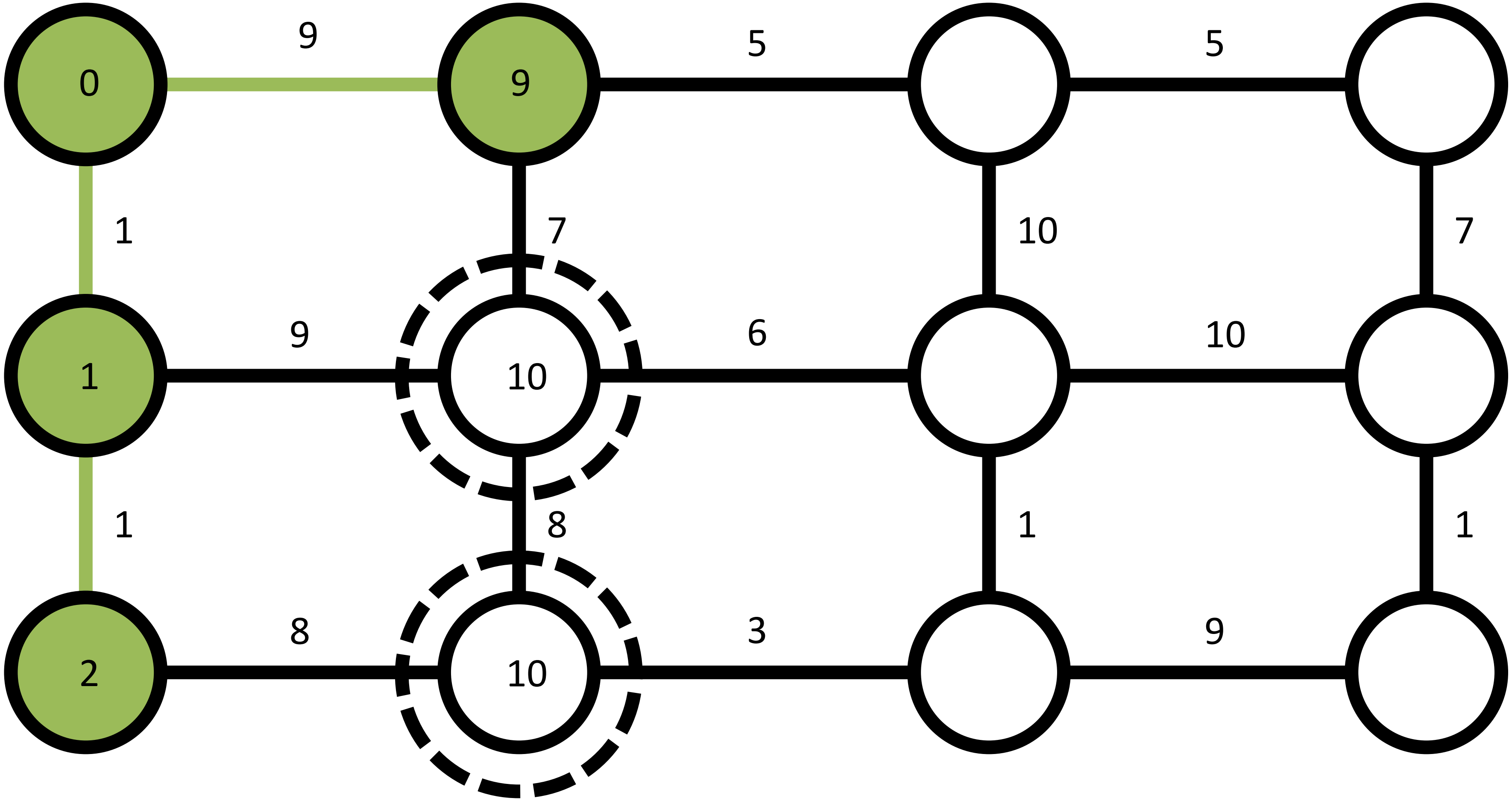

0
9
9
5
5
1
7
10
7
1
9
6
10
1
8
1
1
8
3
9

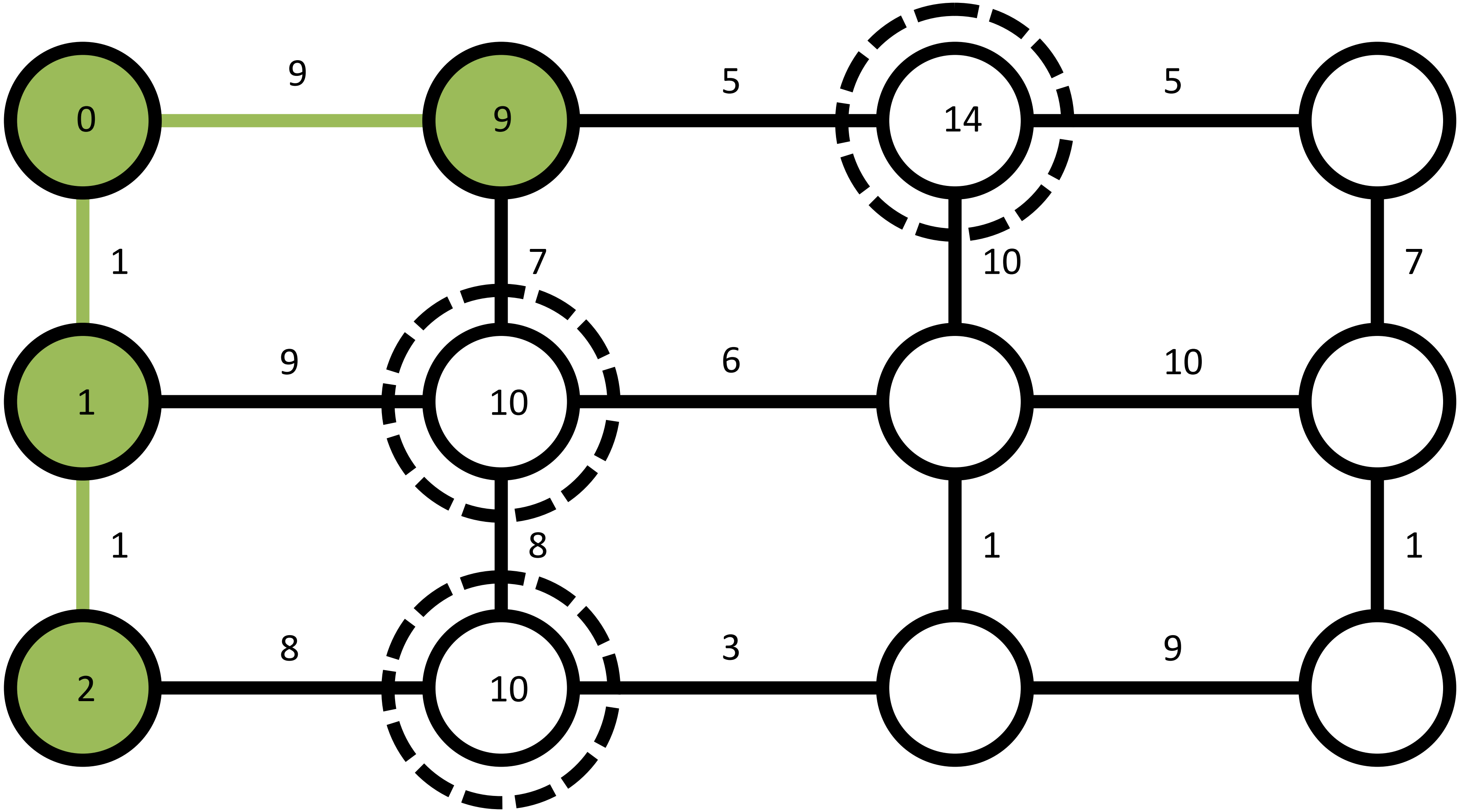

0
9
9
5
5
1
7
10
7
1
9
6
10
1
8
1
1
8
3
9

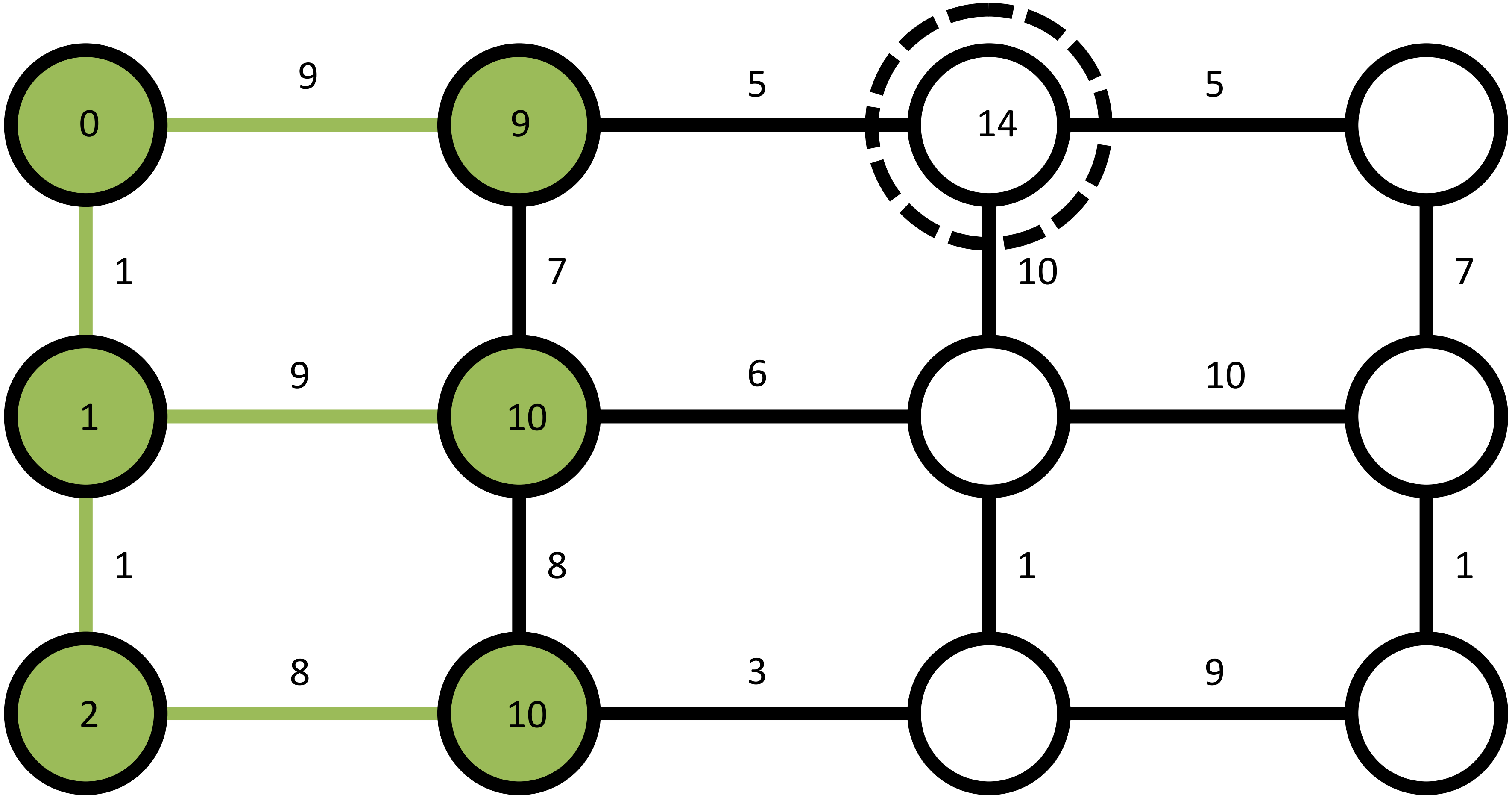

0
9
9
5
5
1
7
10
7
1
9
10
6
10
1
8
1
1
2
8
3
9

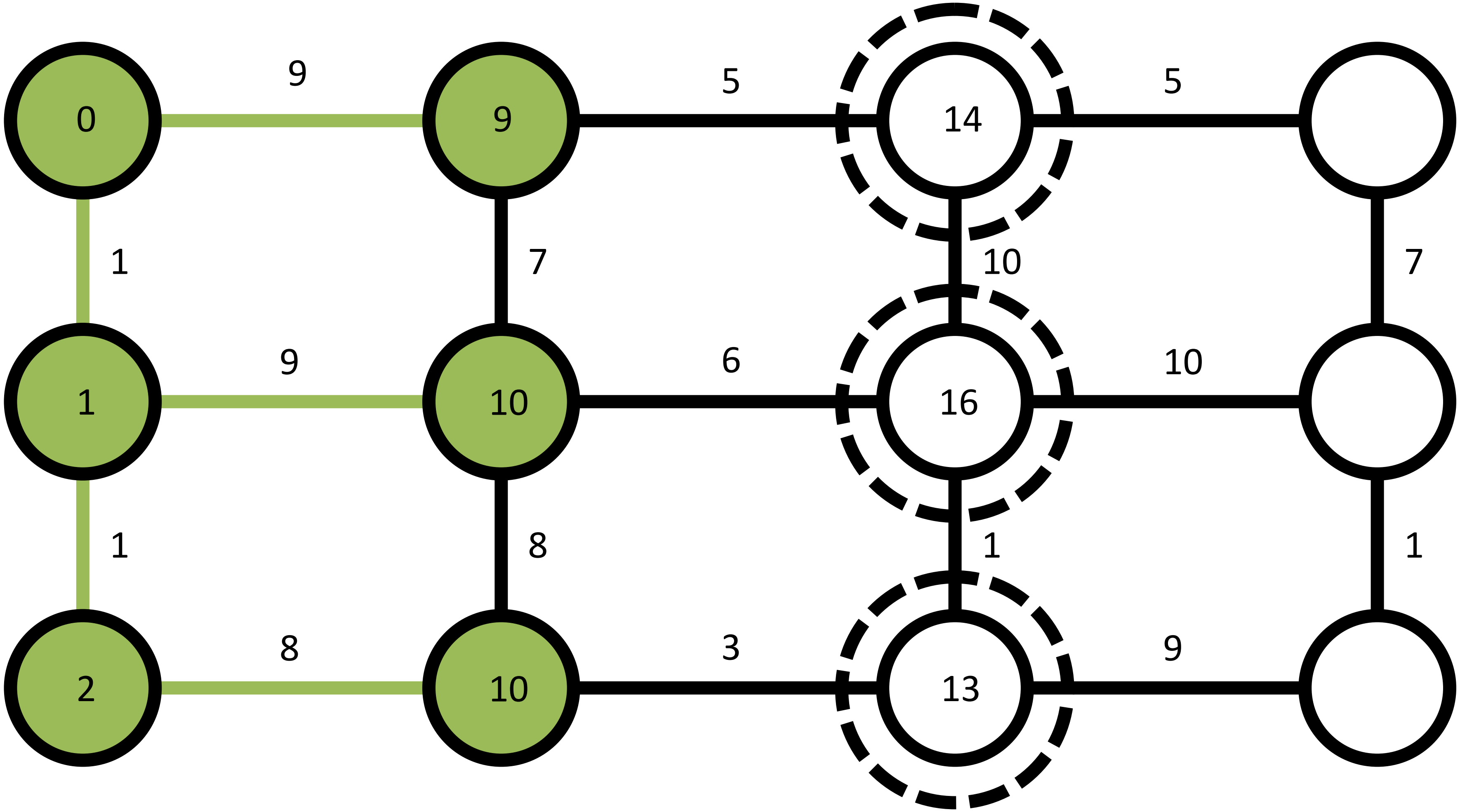

0
9
9
5
5
1
7
10
7
1
9
10
6
10
1
8
1
1
2
8
3
9

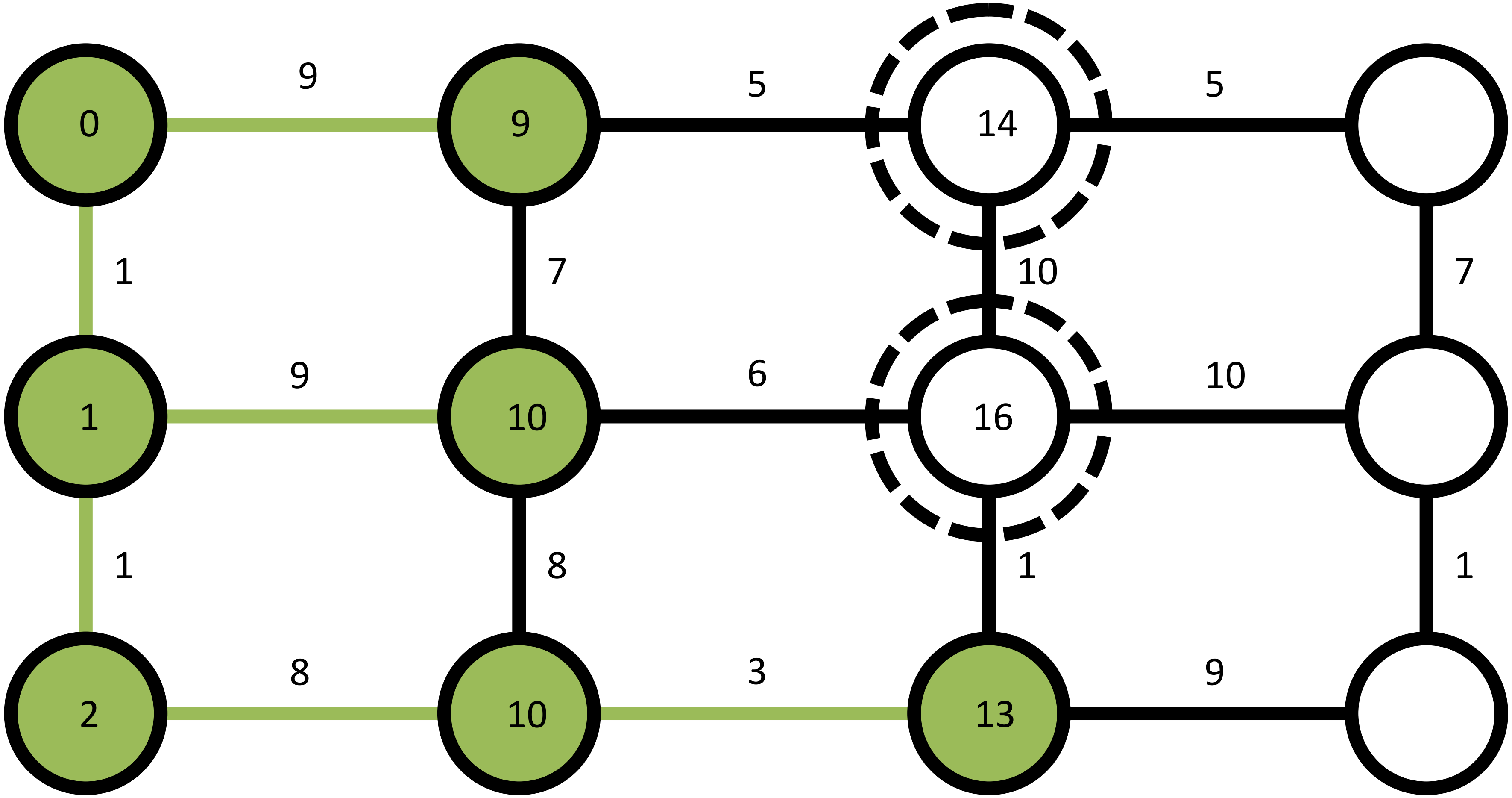

0
9
5
5
9
1
7
10
7
1
9
10
6
10
1
8
1
1
2
8
10
3
9

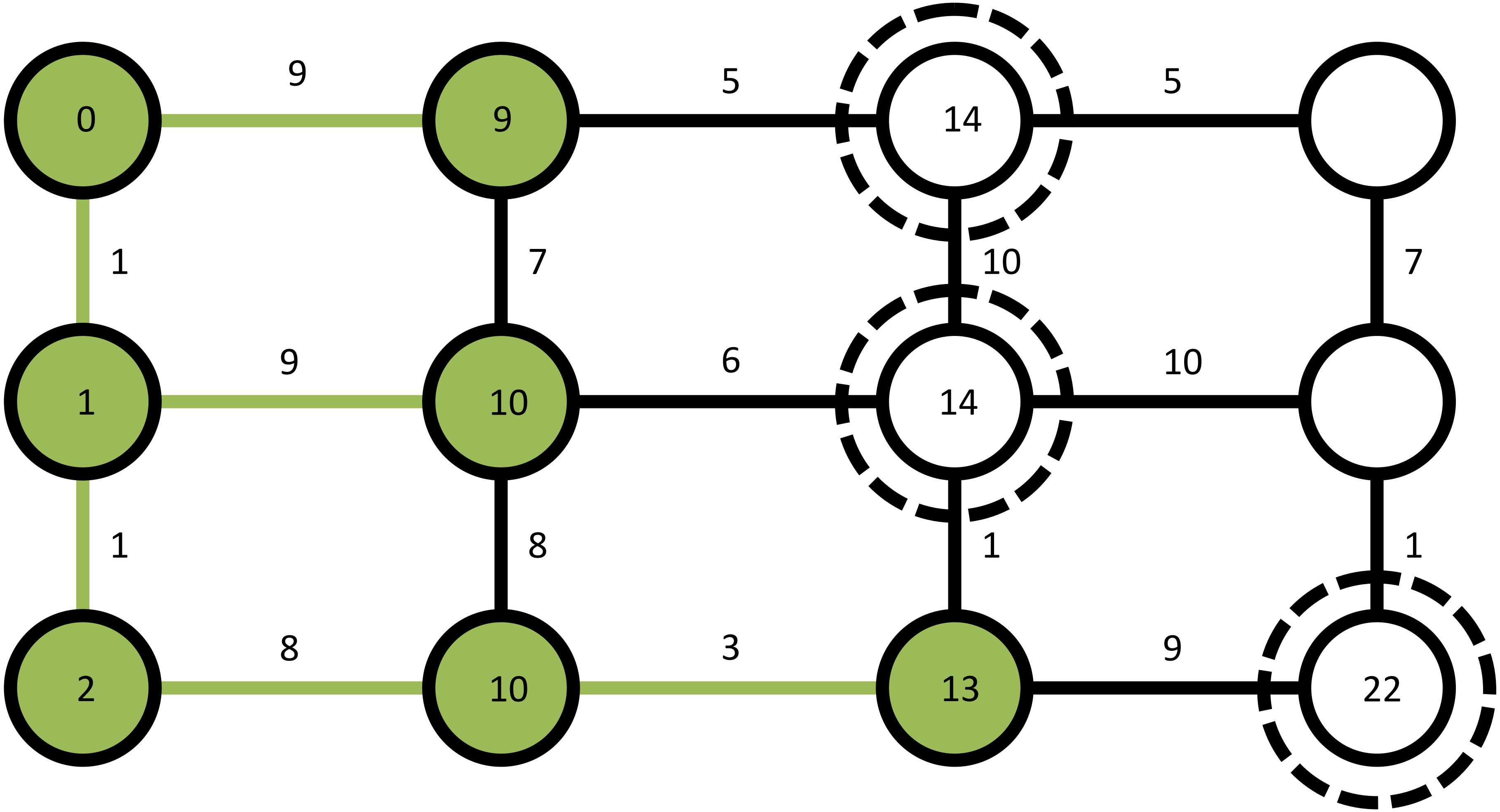

0
9
9
5
5
1
7
10
7
1
9
10
6
10
1
8
1
1
2
8
10
3
9

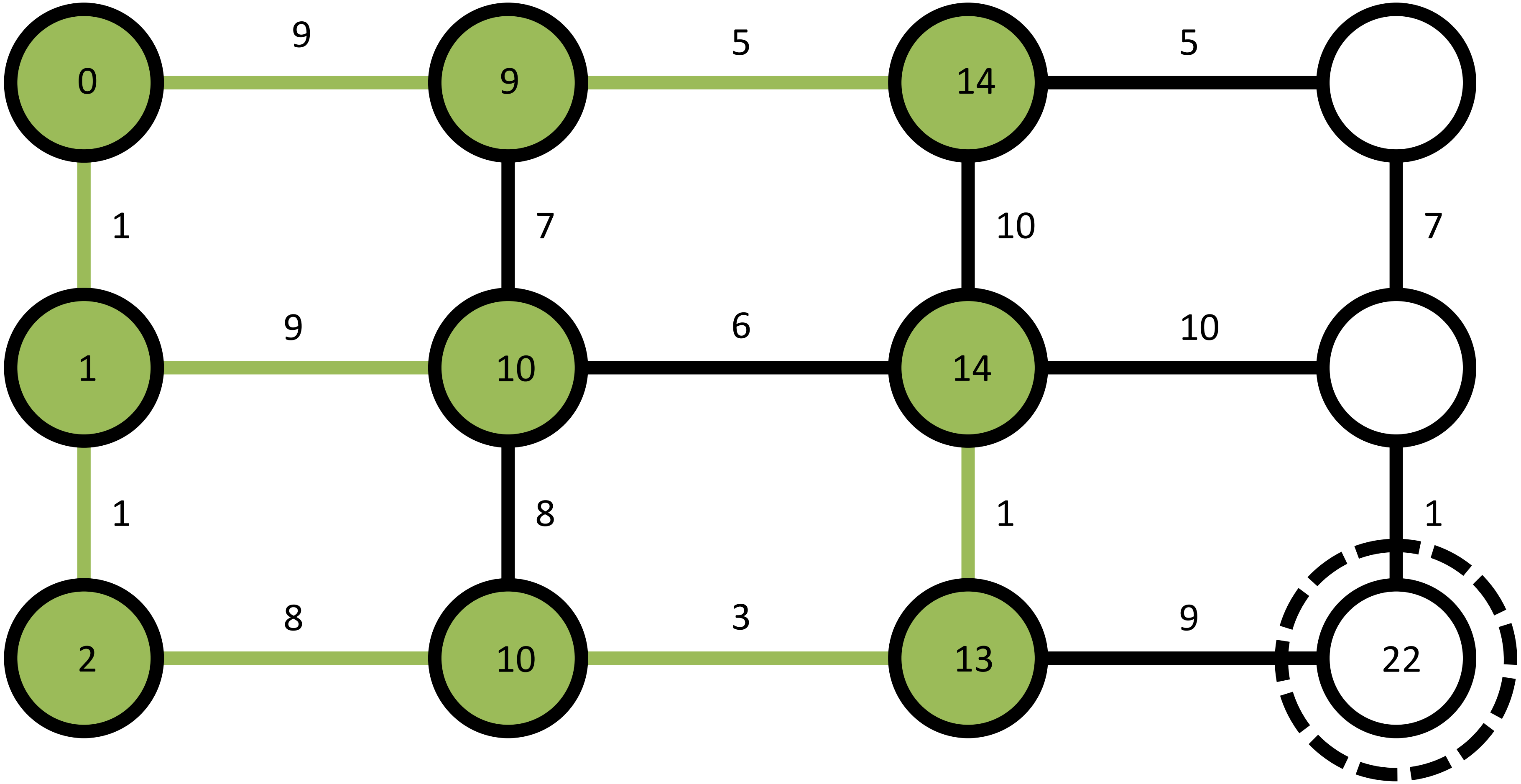

0
9
9
5
14
5
1
7
10
7
1
9
10
6
10
1
8
1
1
2
8
10
3
9

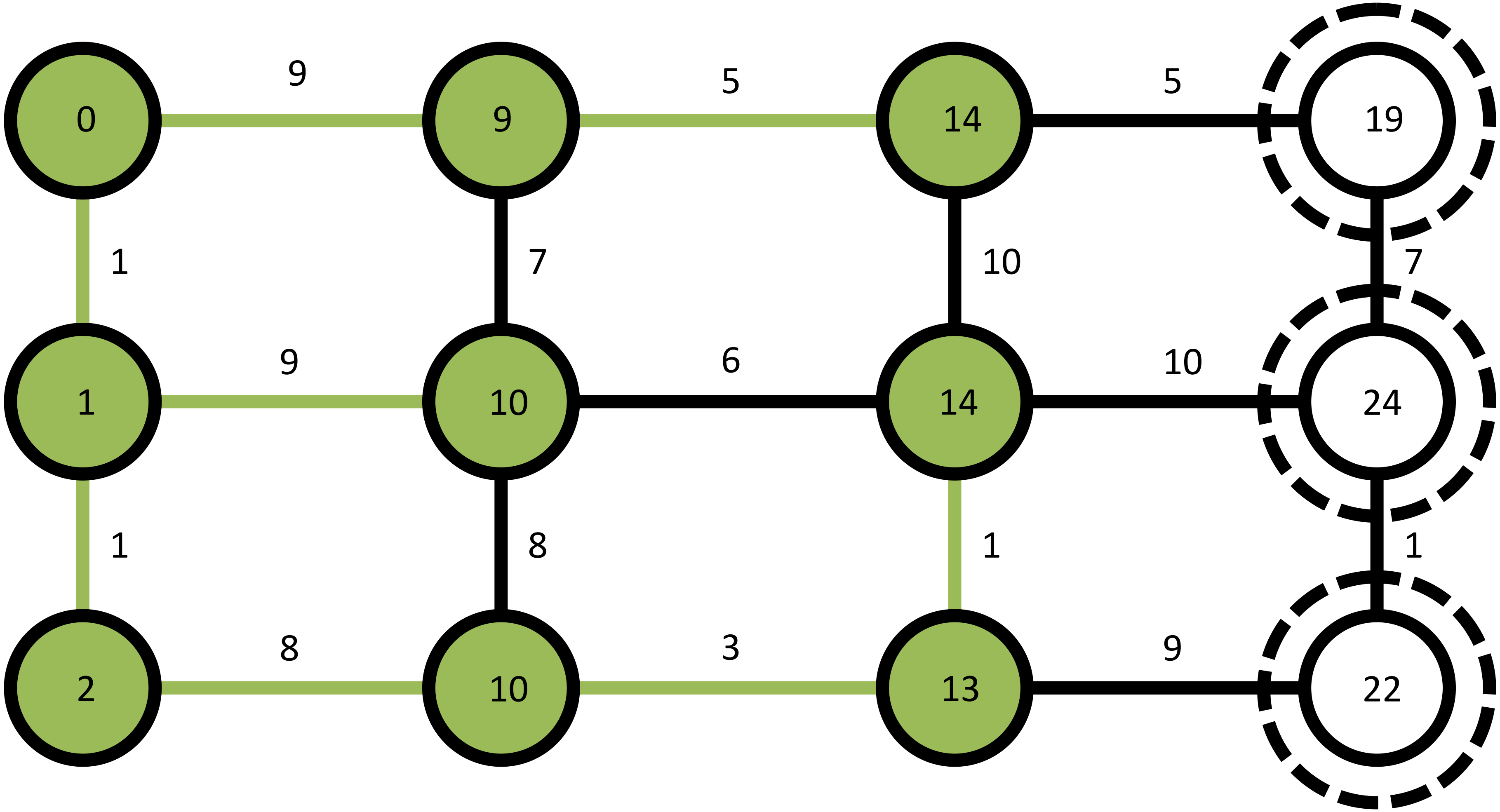

0
9
9
5
14
5
1
7
10
7
1
9
10
6
10
1
8
1
1
2
8
10
3
9

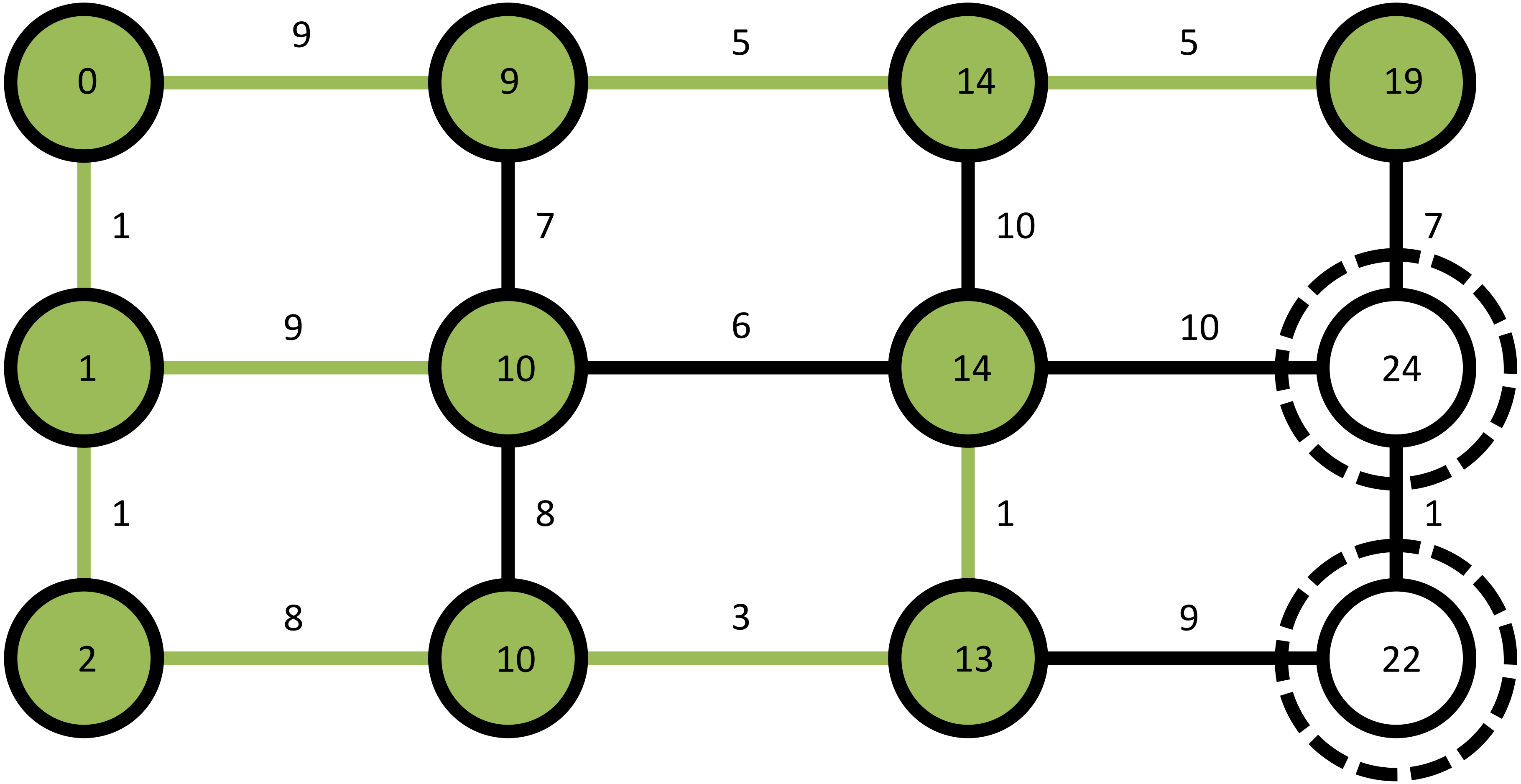

9
5
5
0
9
14
1
7
10
7
9
6
10
1
10
16
1
8
1
1
8
3
9
2
10
13

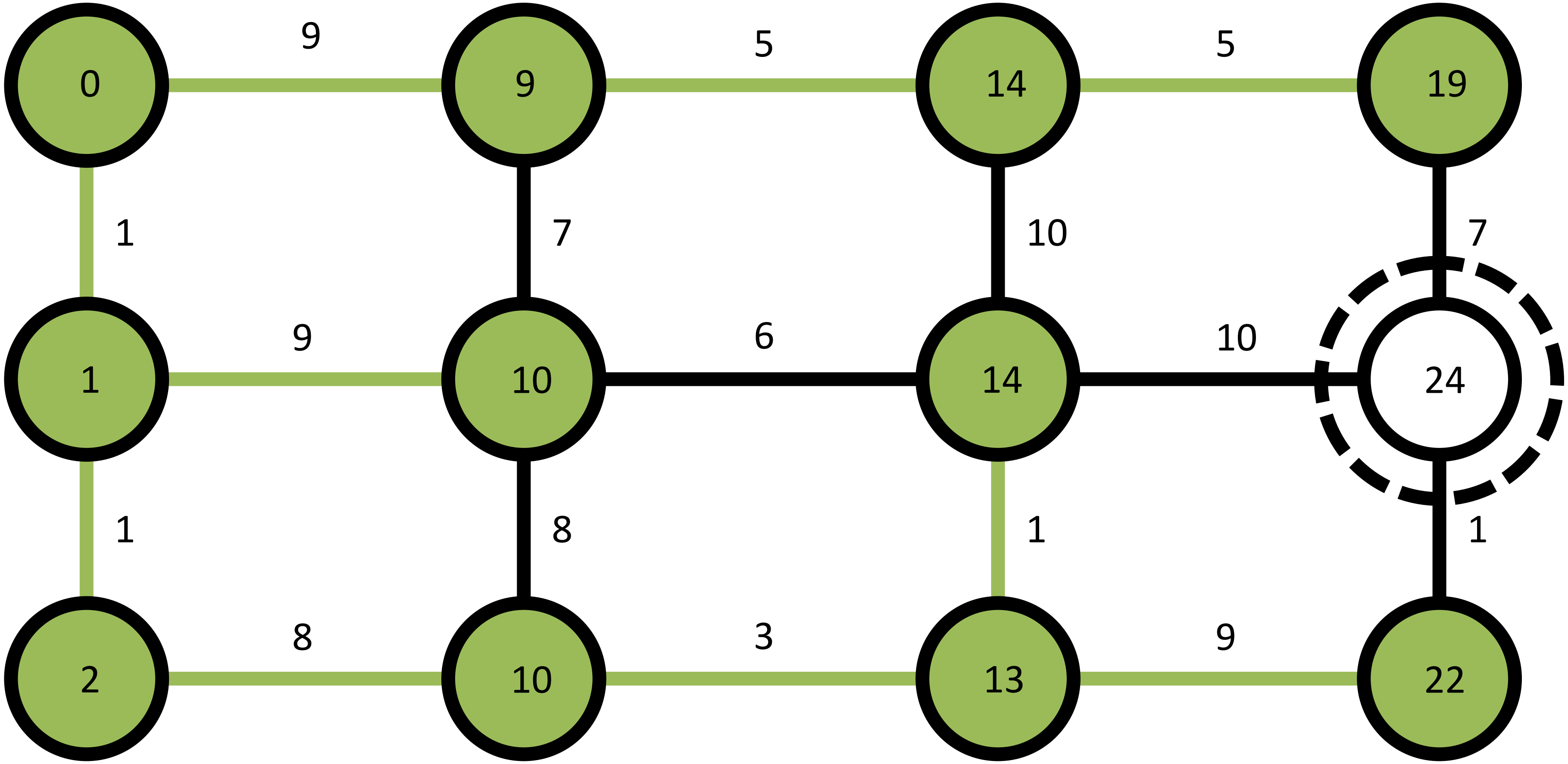

0
9
9
5
14
5
1
7
10
7
1
9
10
6
16
10
1
8
1
1
2
8
10
3
13
9

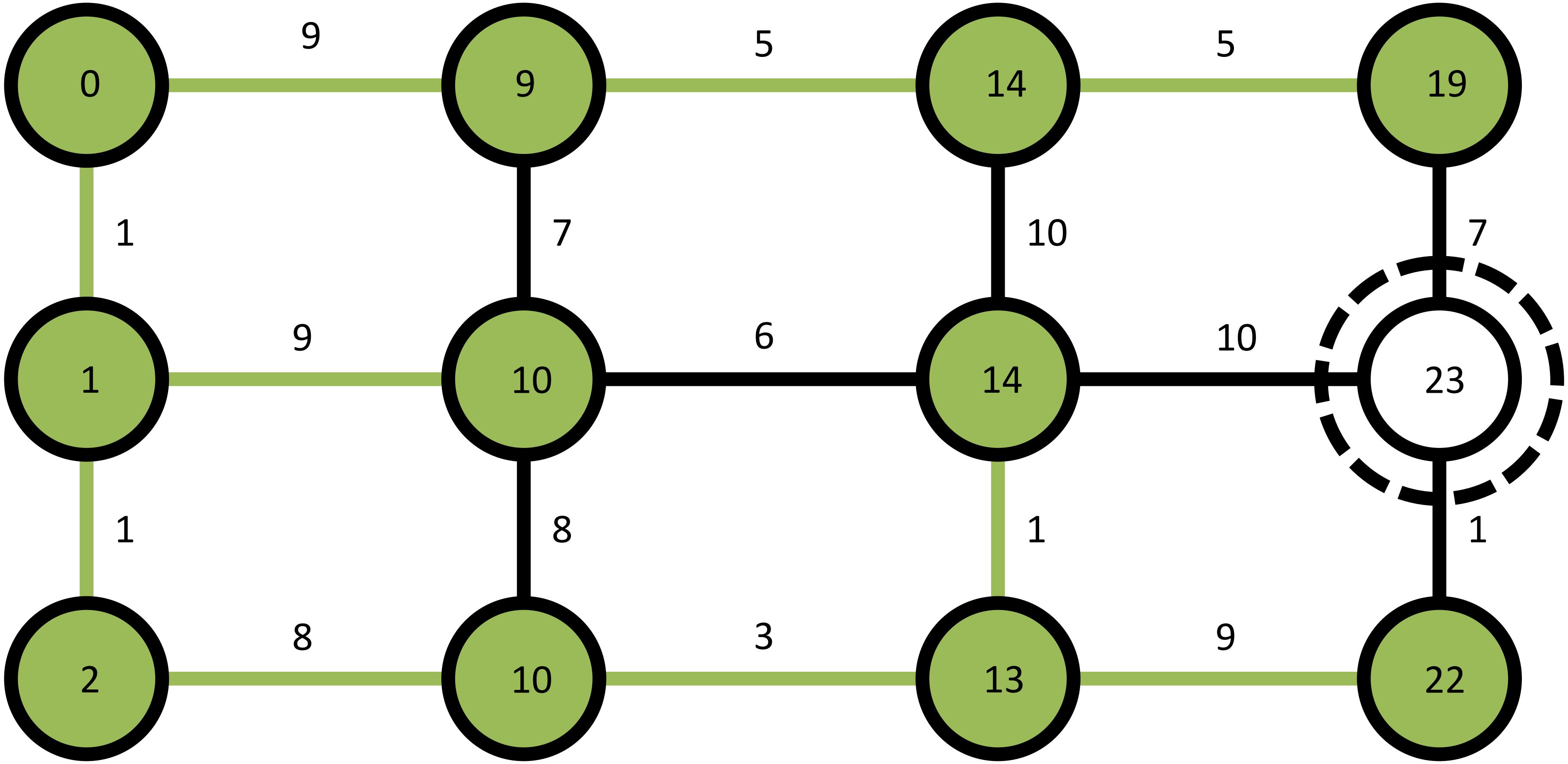

0
9
14
9
5
5
1
7
10
7
1
10
14
9
6
10
1
8
1
1
2
10
13
22
8
3
9

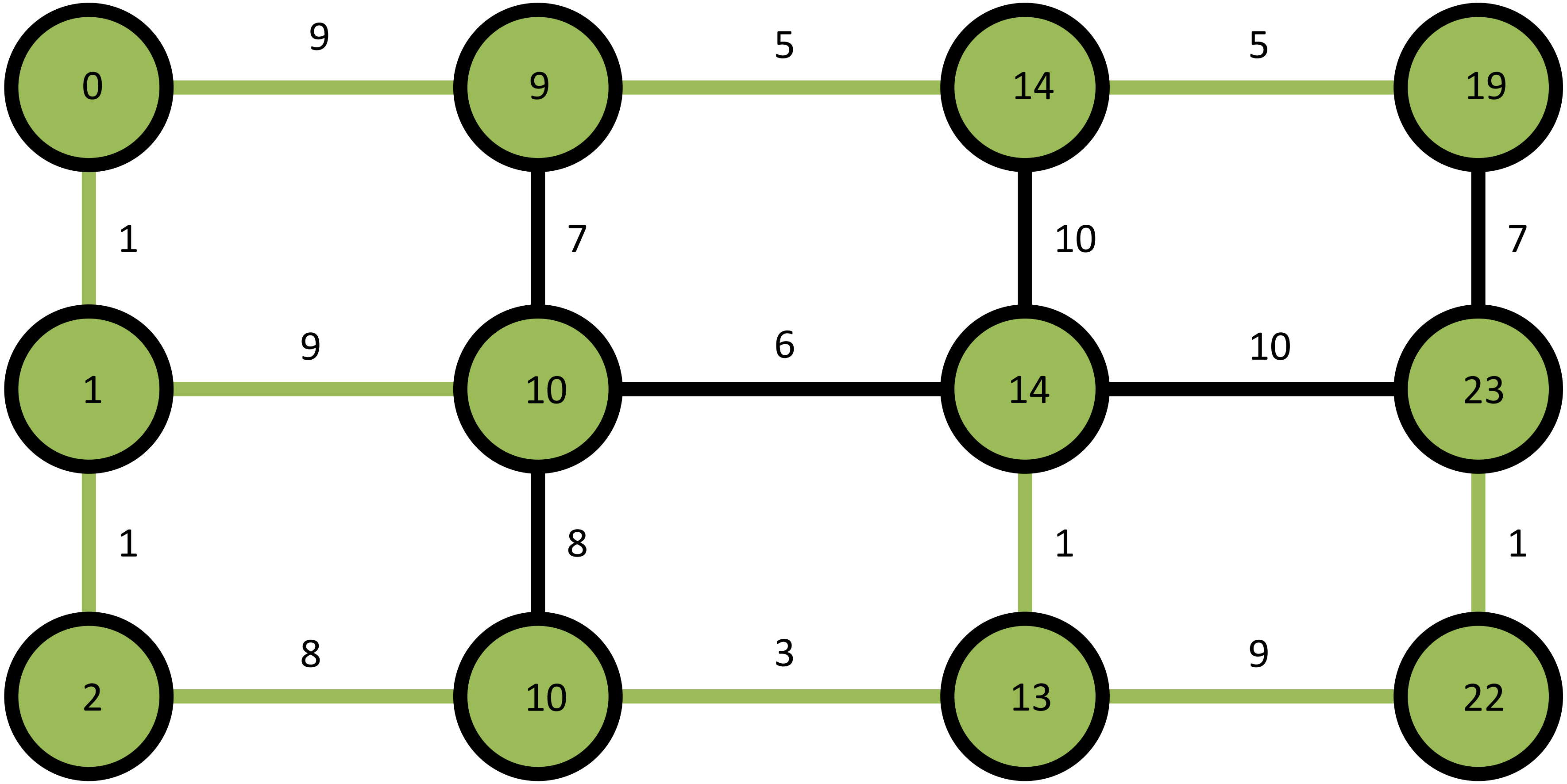

9
5
5
0
9
14
1
7
10
7
9
6
10
1
10
14
1
8
1
1
8
3
9
2
10
13
22


9
5
5
0
9
14
19
1
7
10
7
9
6
10
1
10
14
24
1
1
1
8
8
3
9
2
10
13
22

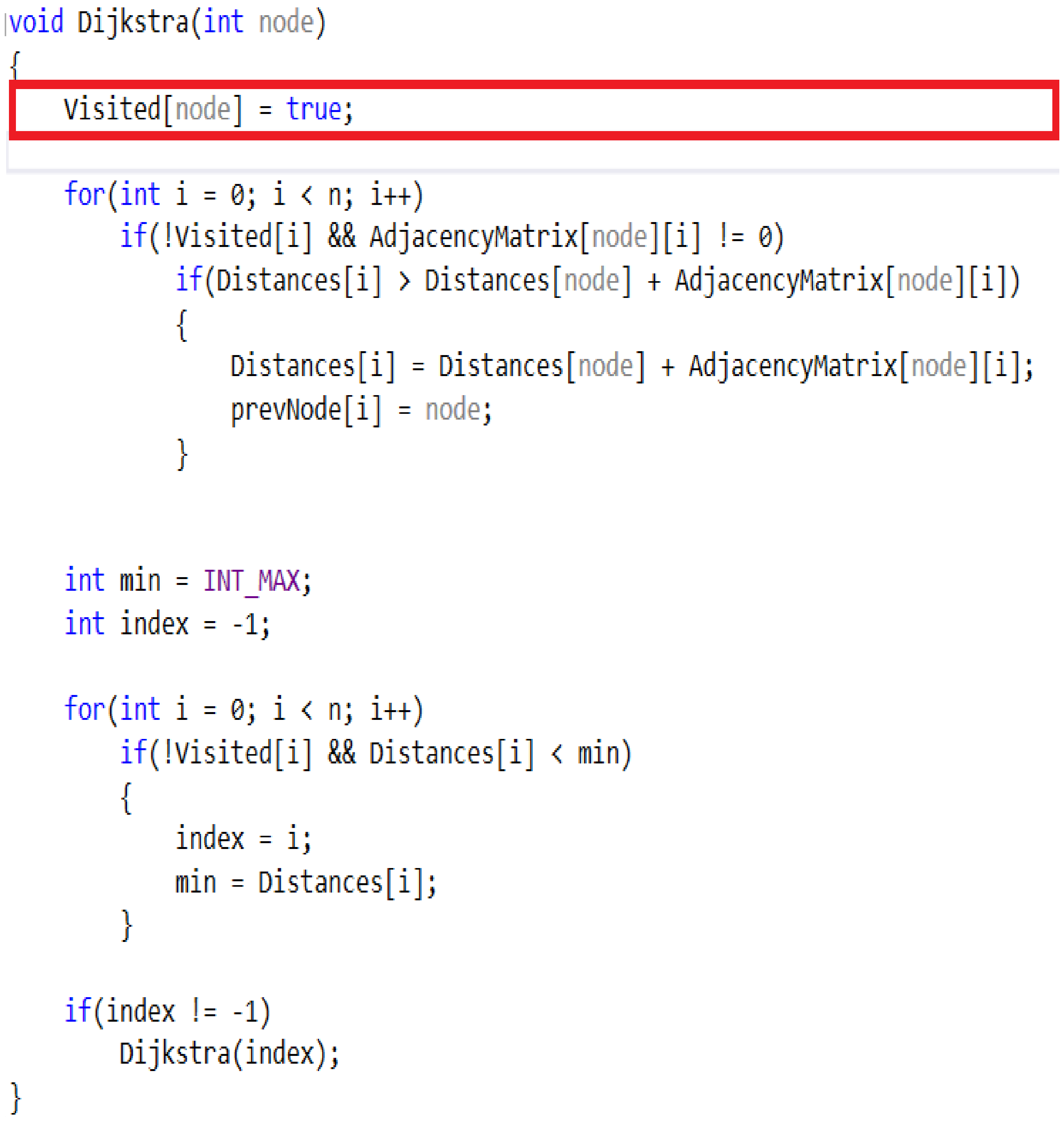

9
5
5
0
9
14
19
1
7
10
7
9
6
10
1
10
14
24
1
8
1
1
8
3
9
2
10
13
22


9
5
5
0
9
14
19
1
7
10
7
9
6
10
1
10
14
24
1
8
1
1
8
3
9
2
10
13
22

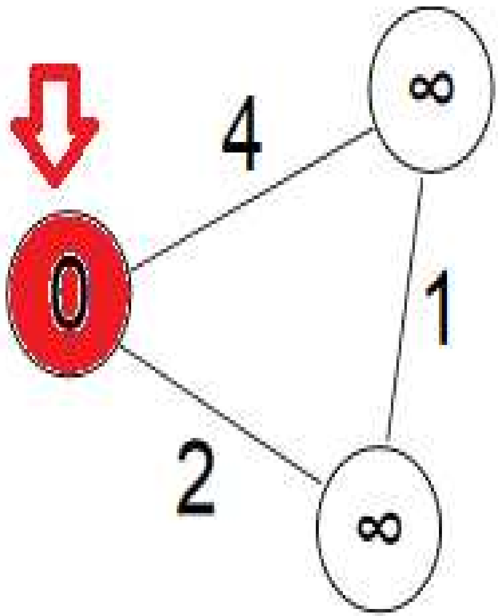

9
5
5
0
9
14
19
1
7
10
7
9
6
10
1
10
14
23
1
8
1
1
8
3
9
2
10
13
22

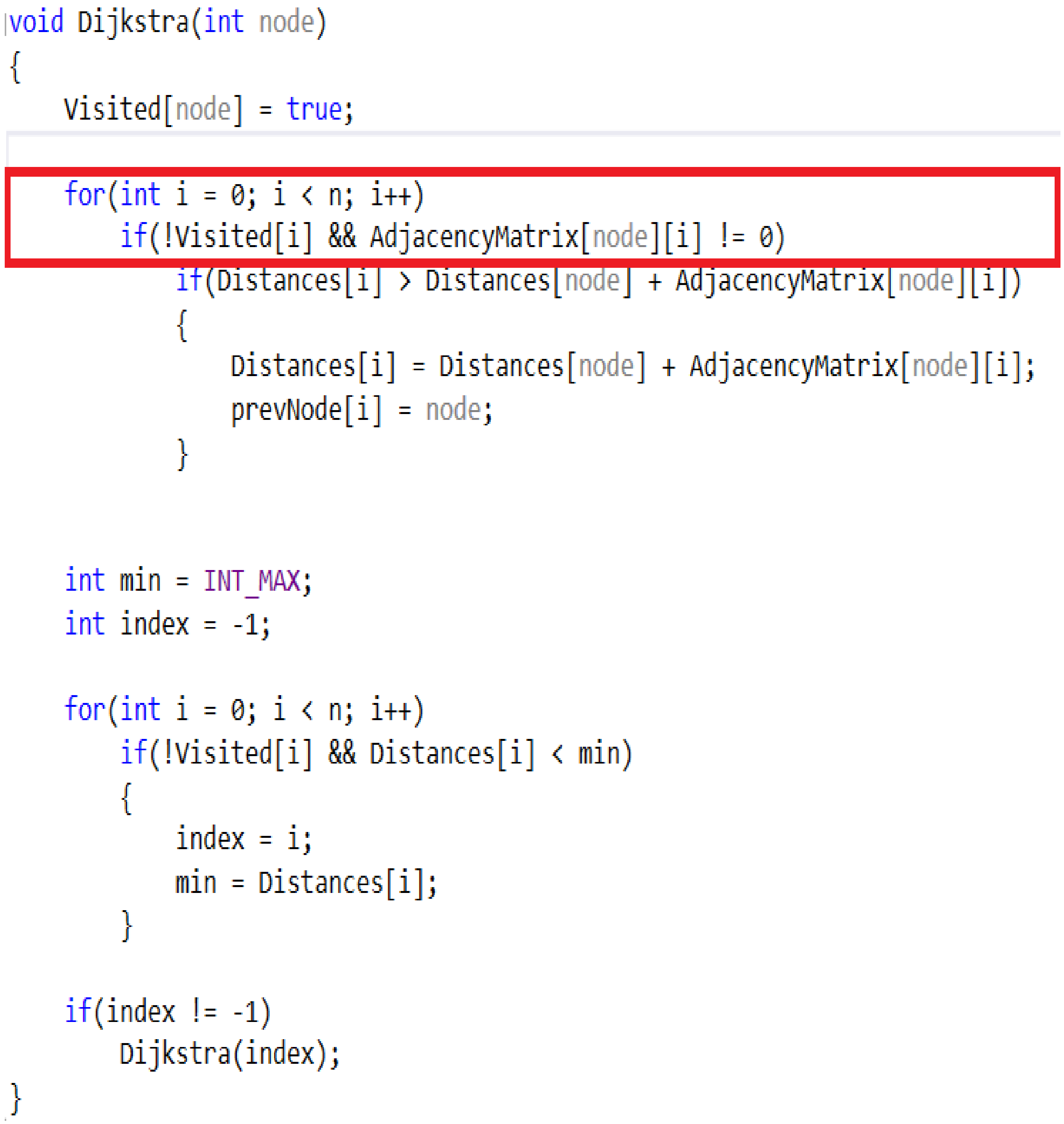

0
9
9
5
14
5
19
1
7
10
7
1
9
10
6
14
10
23
1
8
1
1
2
8
10
3
13
9
22

Implementierung in C++

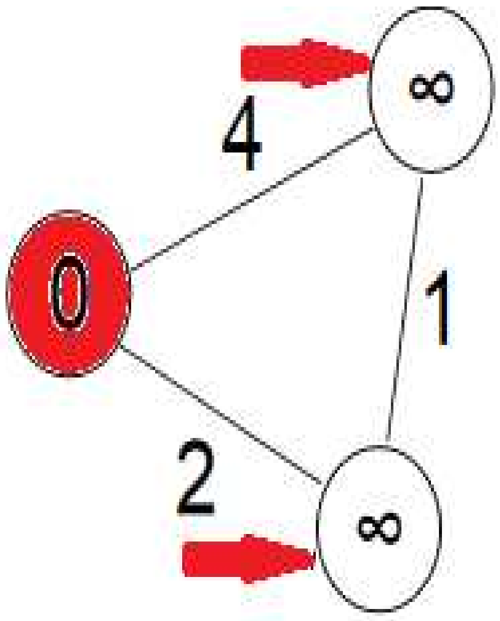


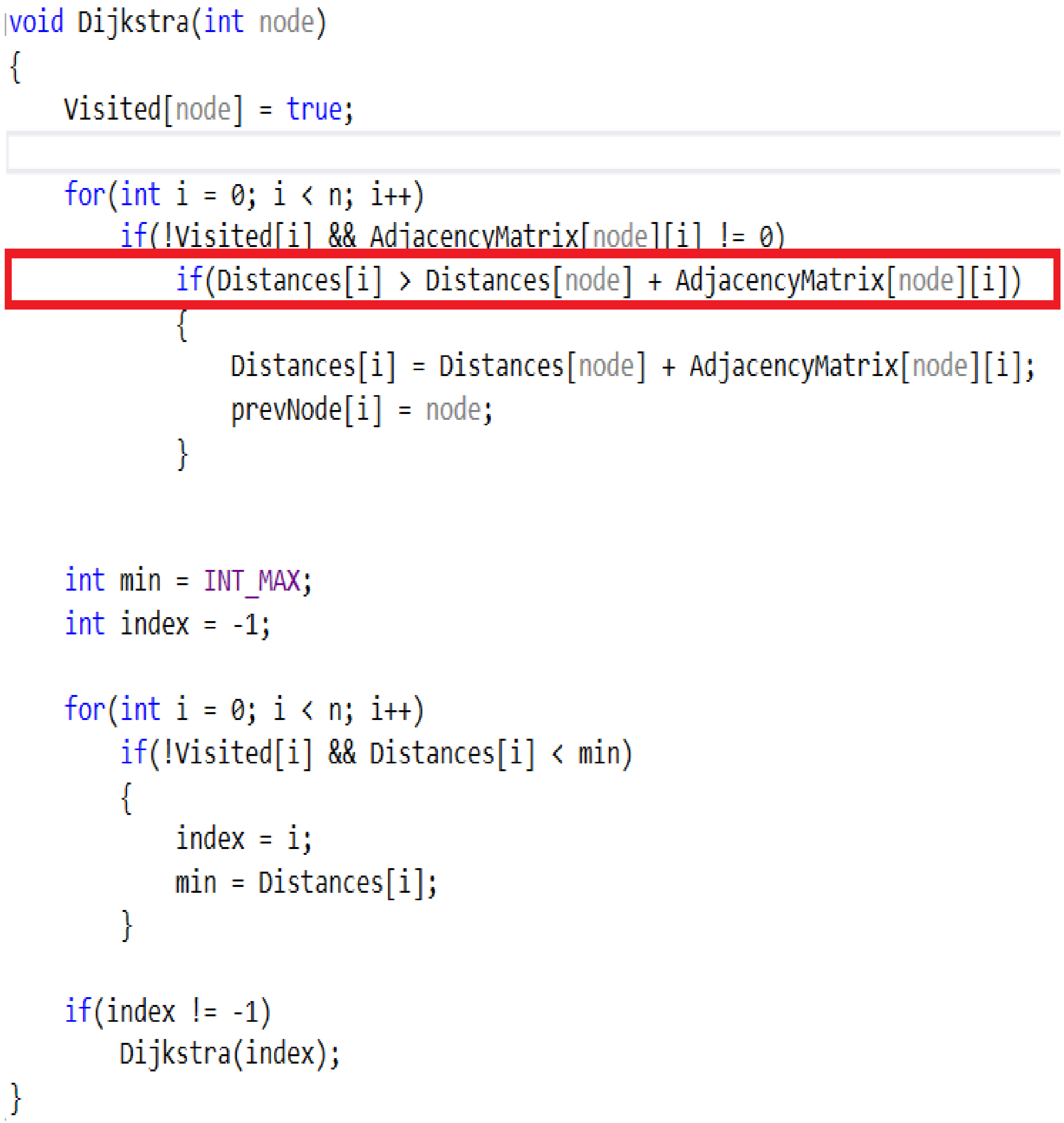
```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```



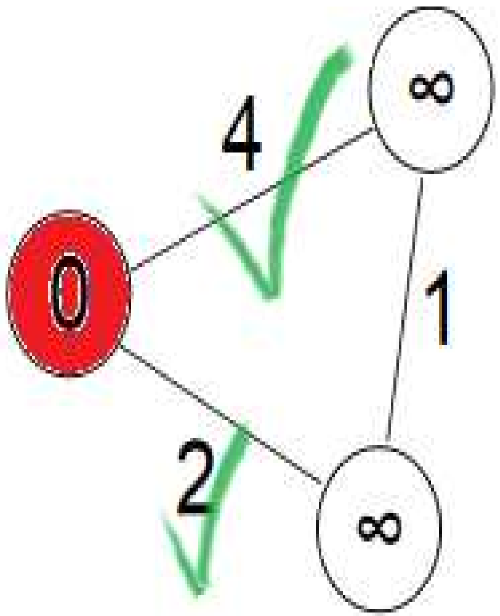


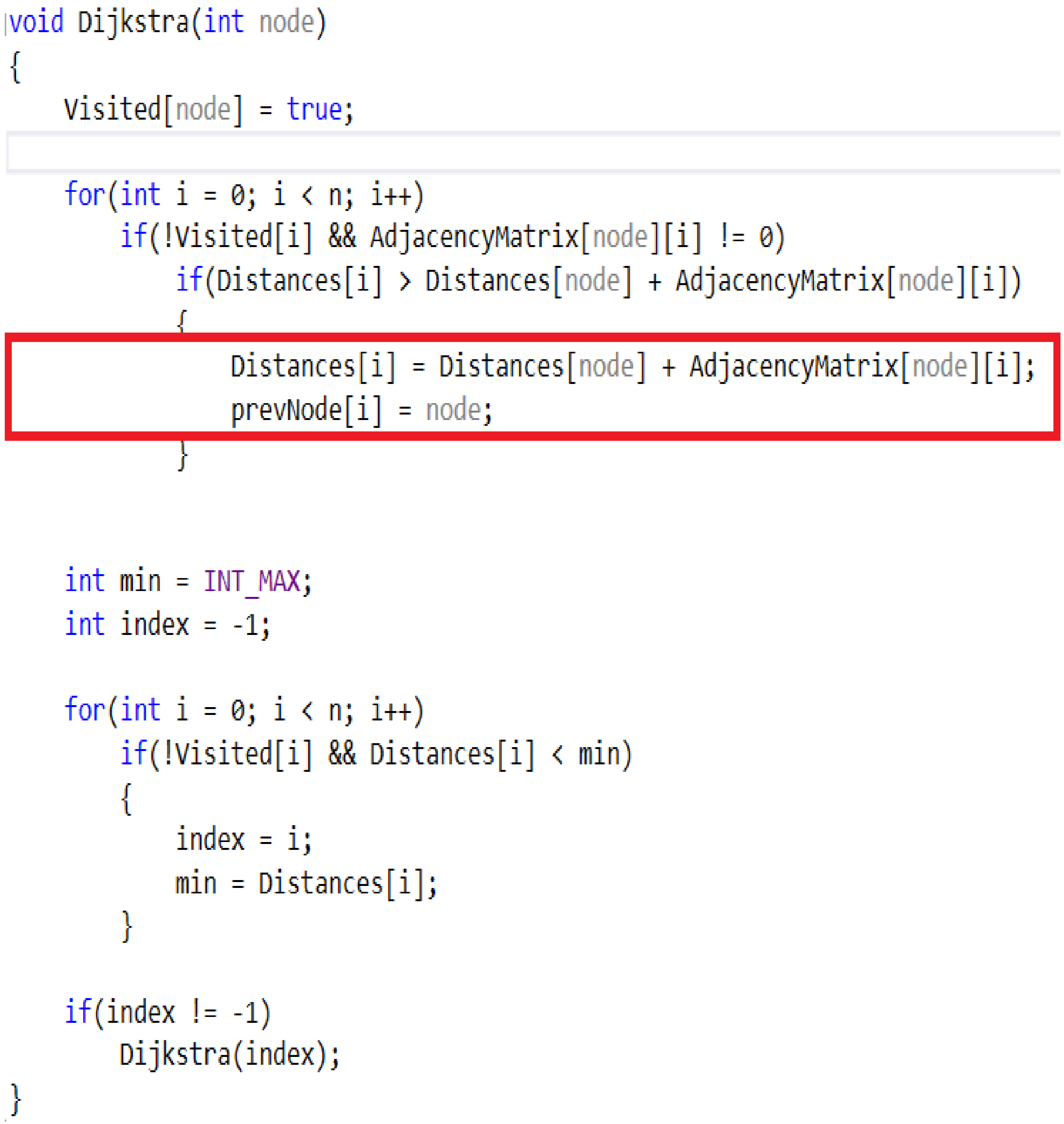
```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```



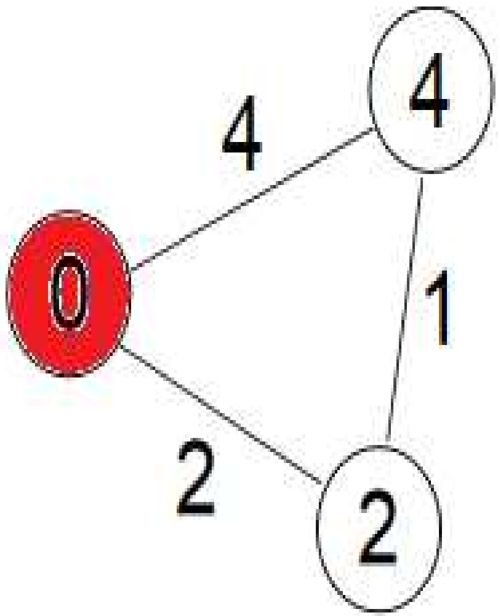


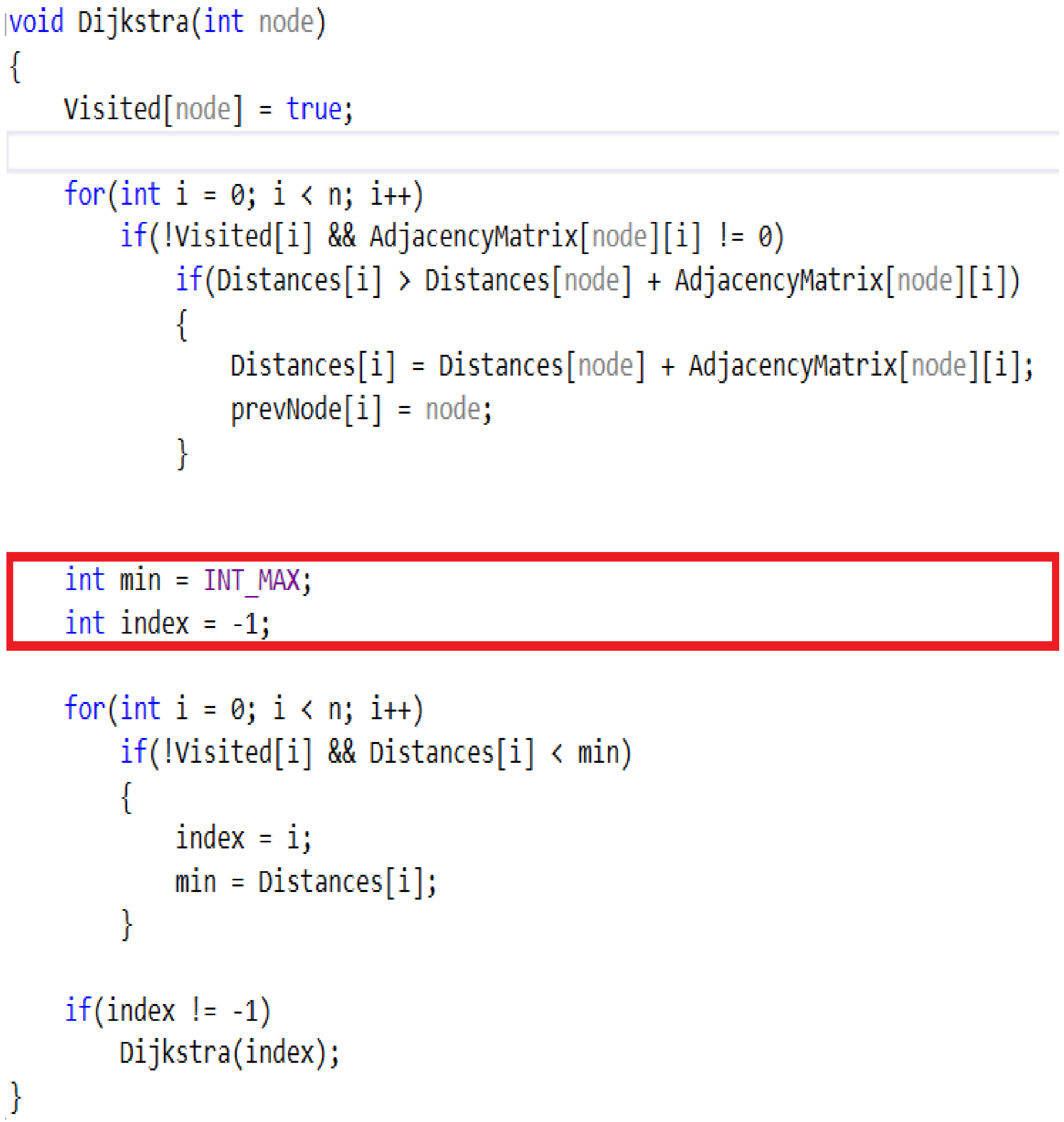
```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```



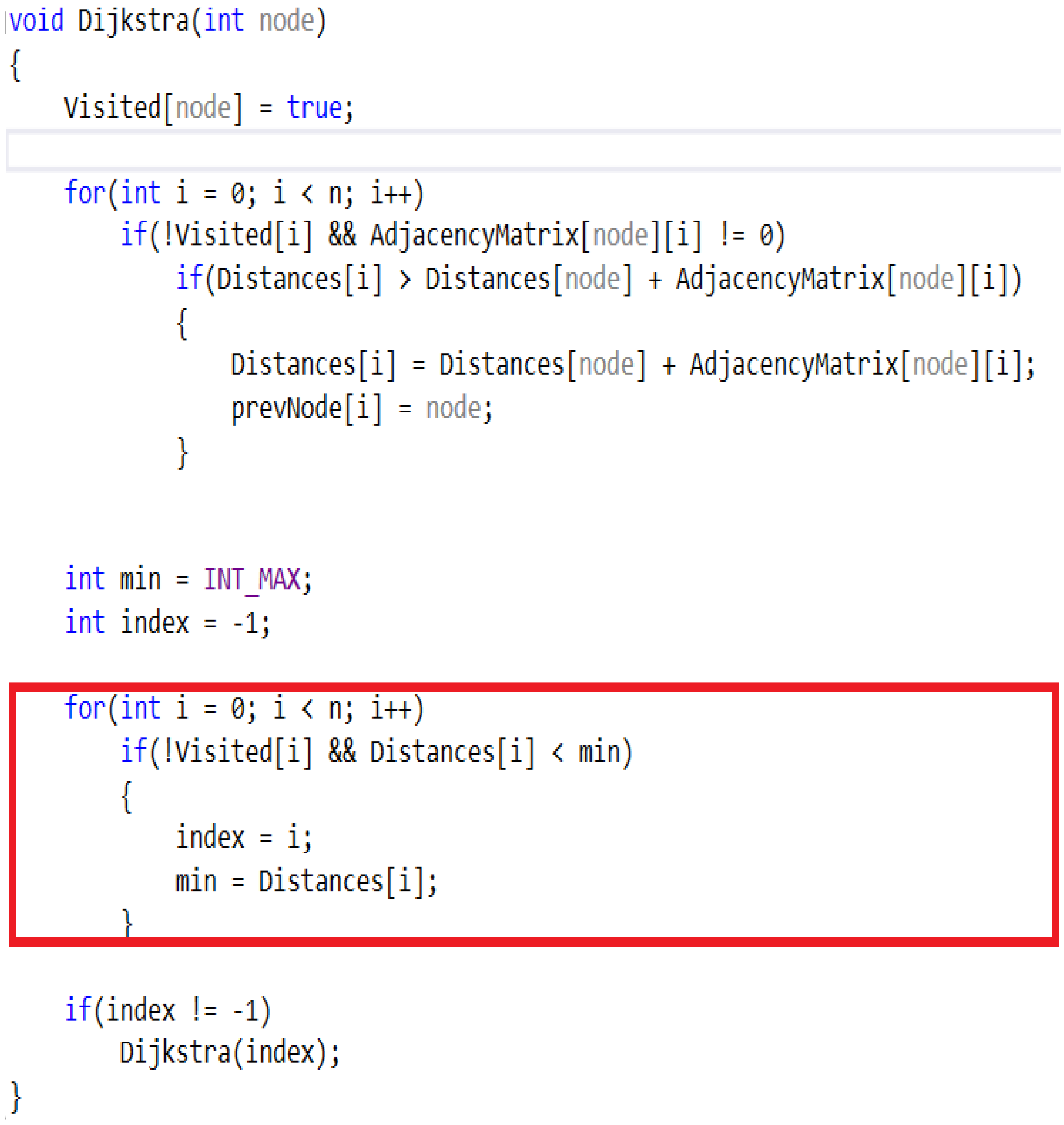


```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```

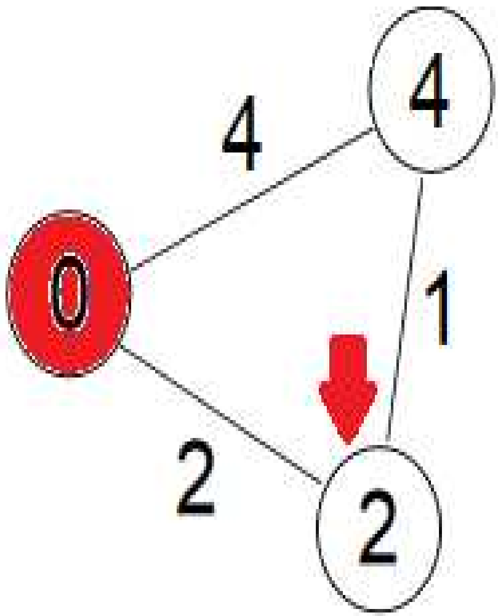


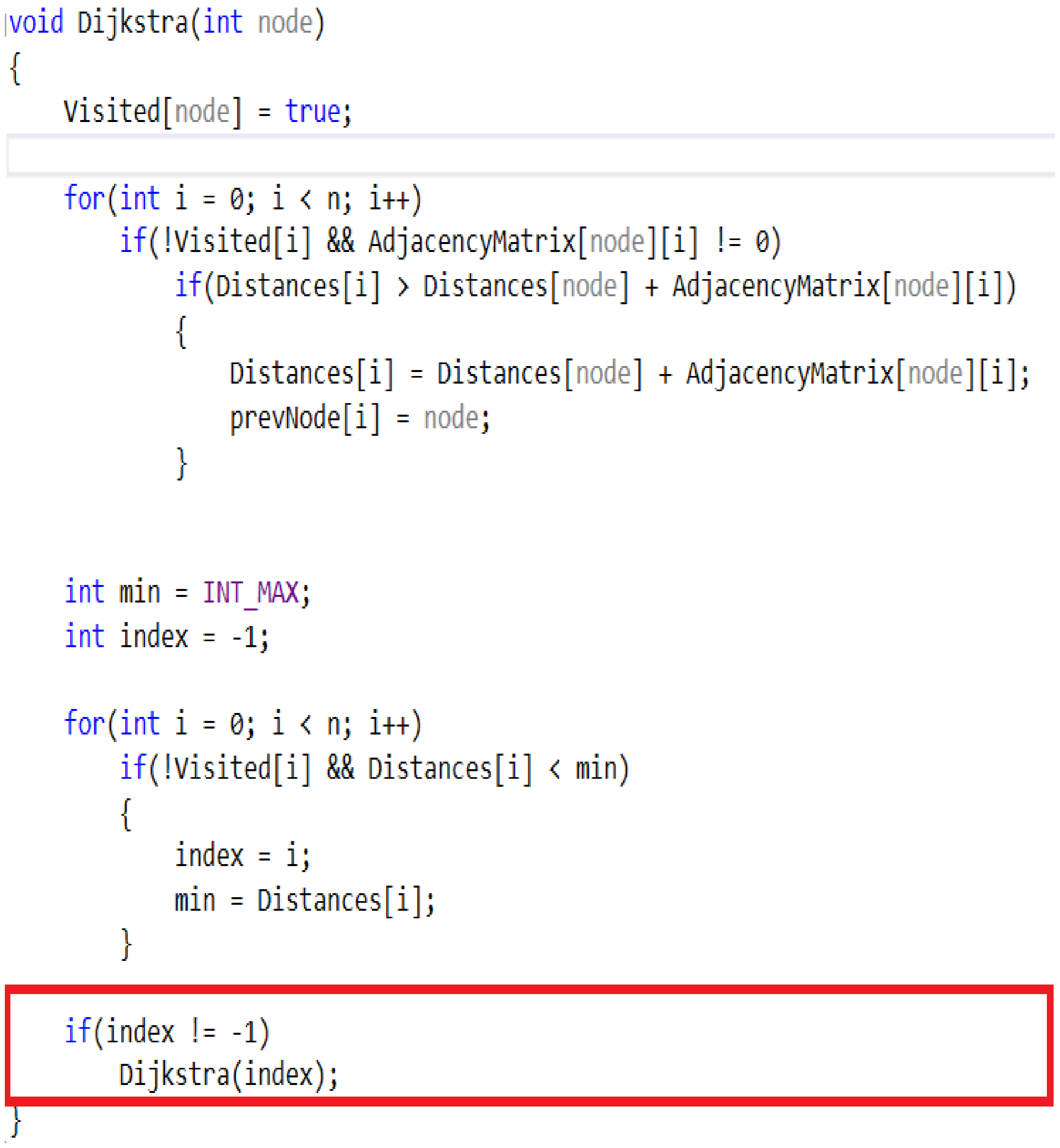


```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```

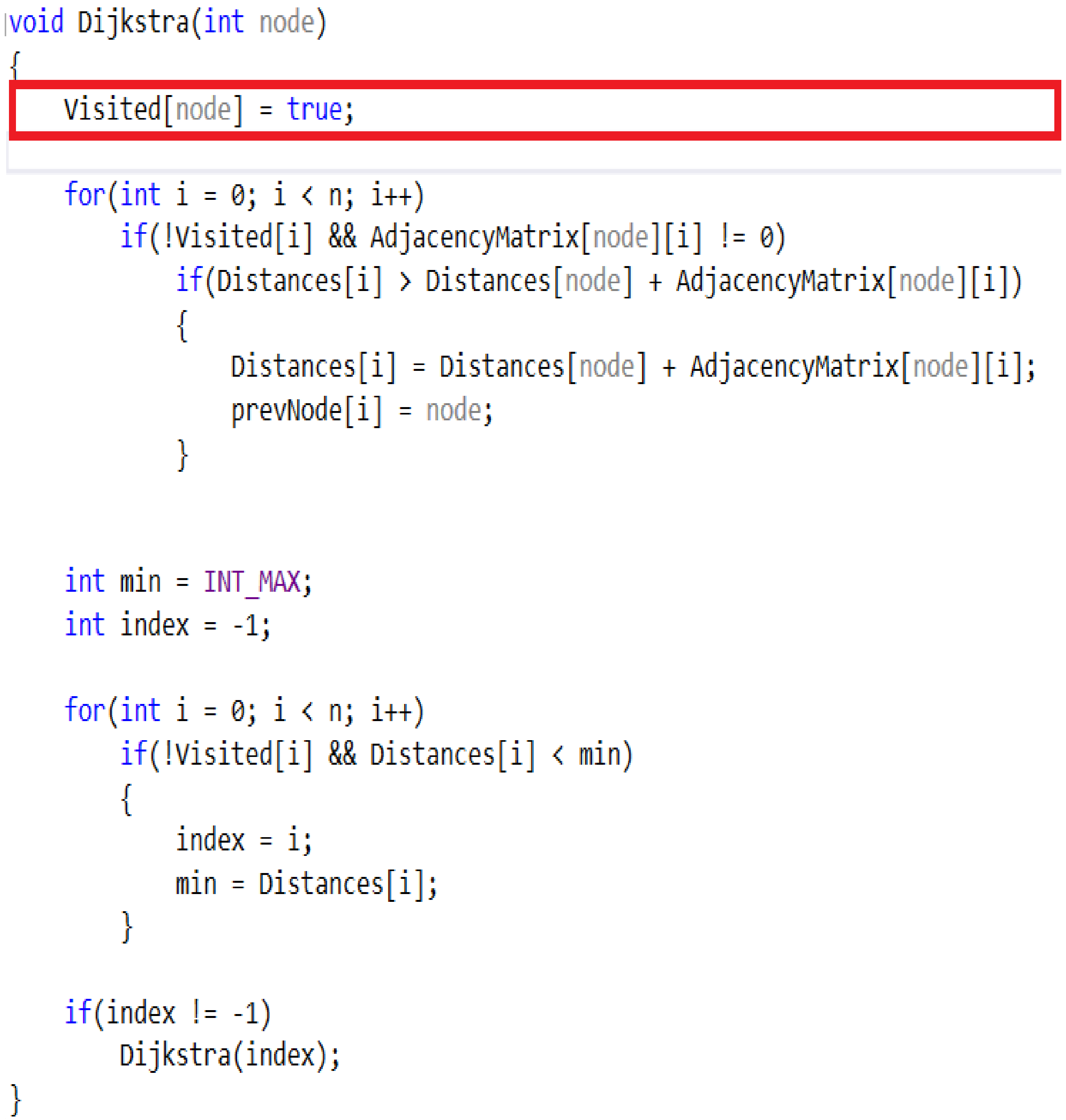


```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```

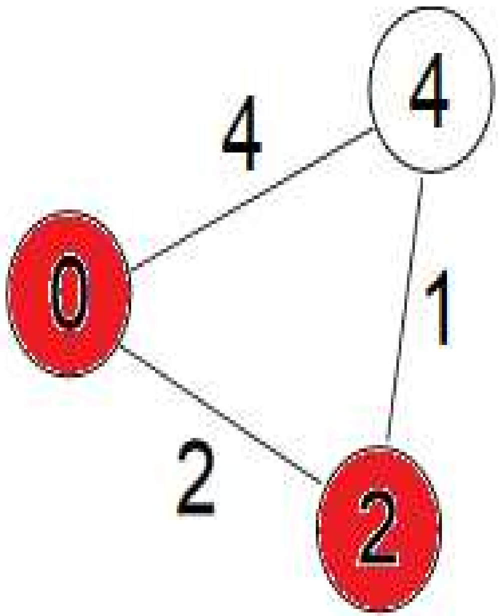


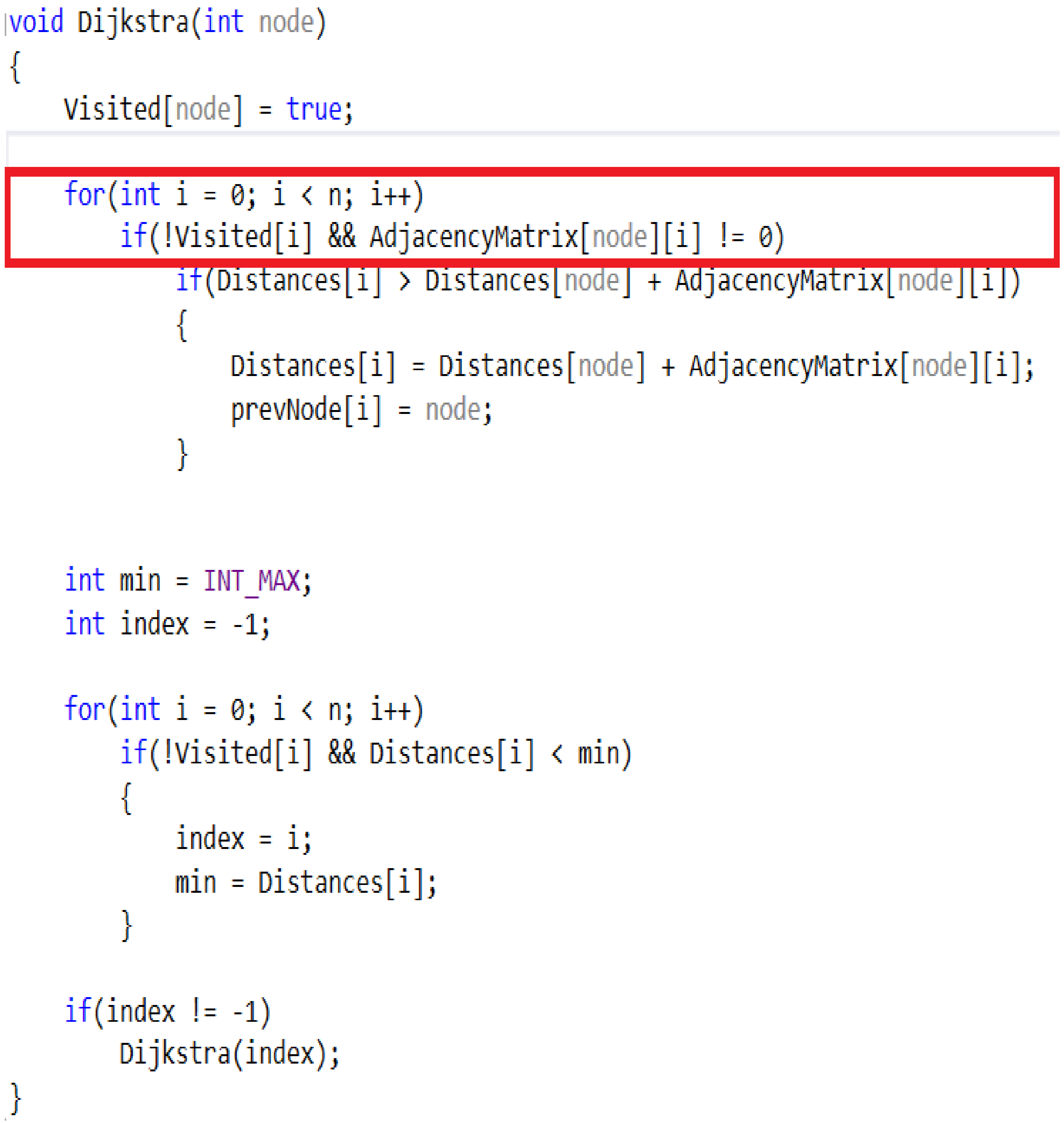


```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```

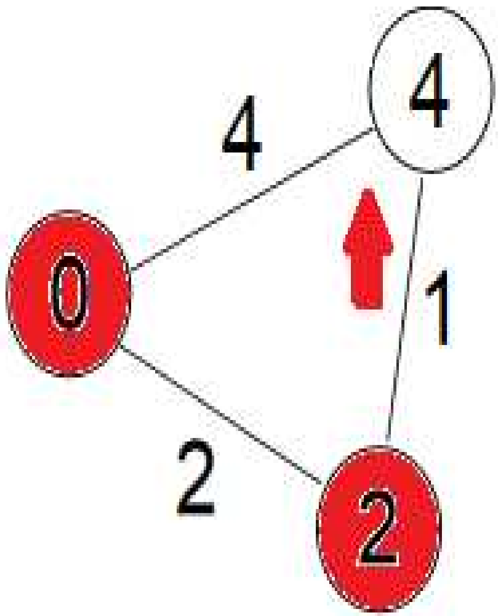


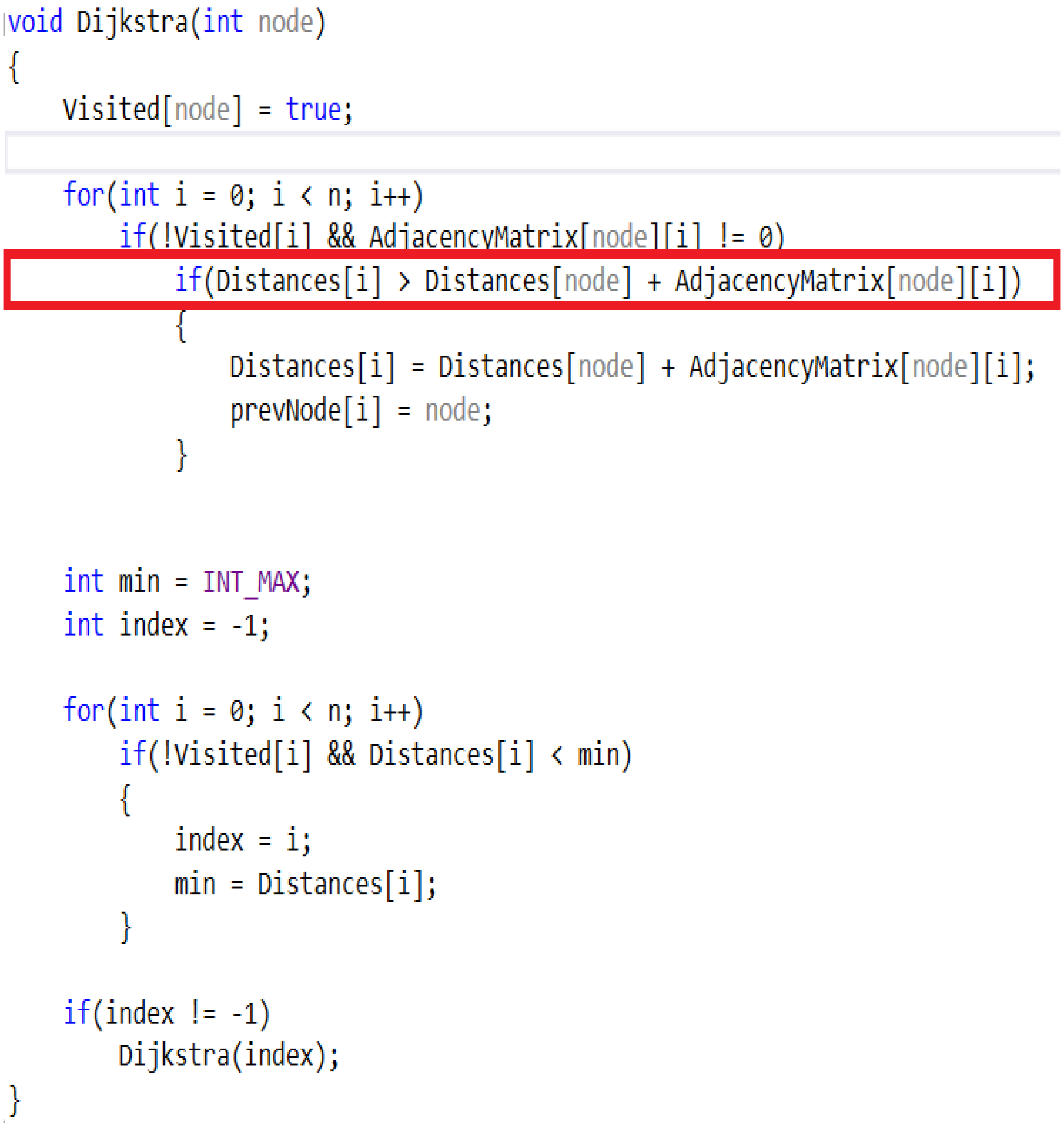
```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```



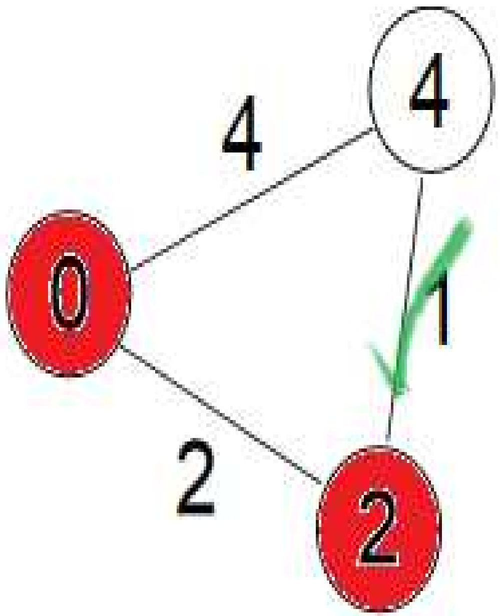


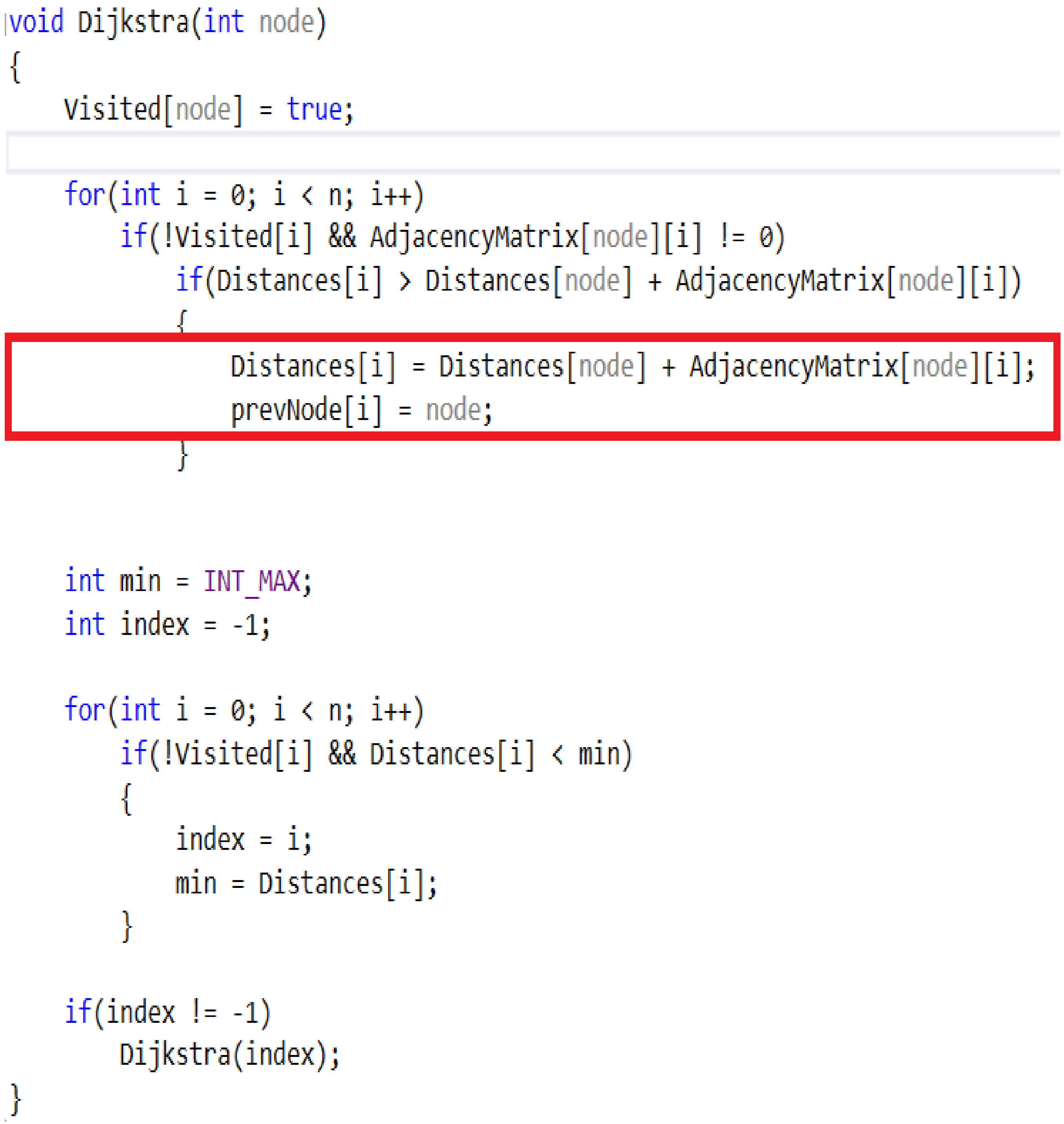
```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```



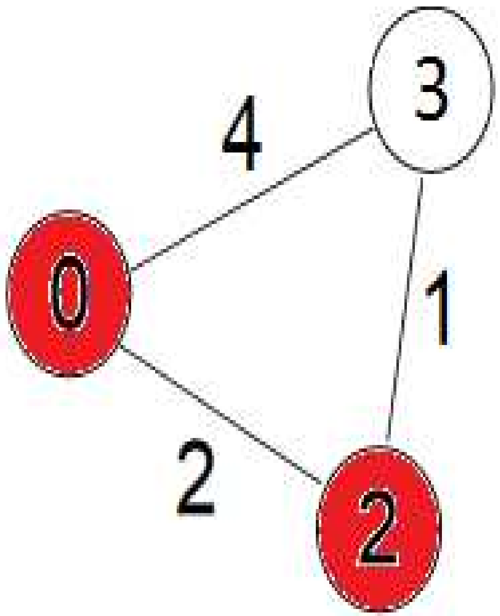


```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```



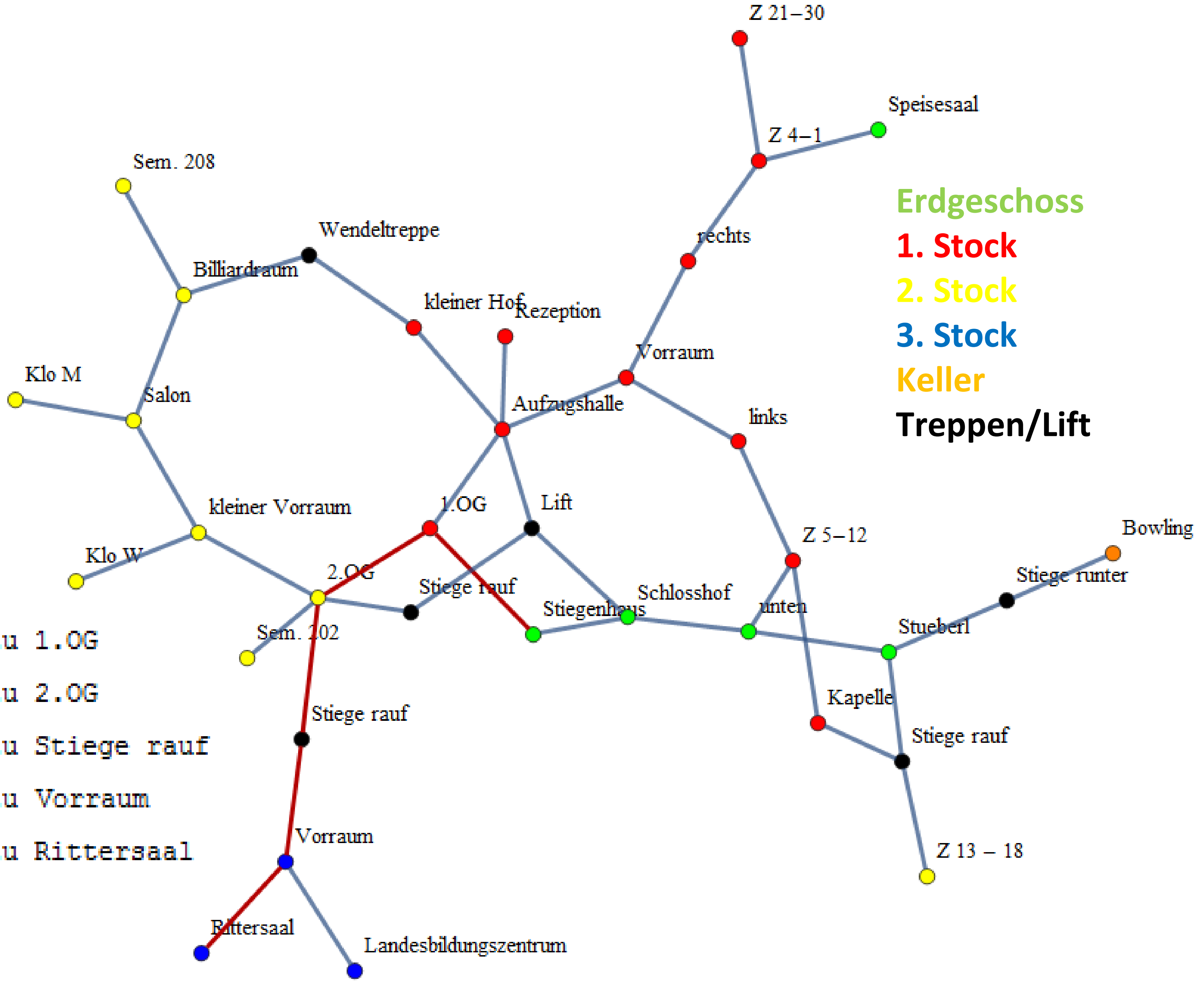


```
void Dijkstra(int node)
{
    Visited[node] = true;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && AdjacencyMatrix[node][i] != 0)
            if(Distances[i] > Distances[node] + AdjacencyMatrix[node][i])
            {
                Distances[i] = Distances[node] + AdjacencyMatrix[node][i];
                prevNode[i] = node;
            }


    int min = INT_MAX;
    int index = -1;

    for(int i = 0; i < n; i++)
        if(!Visited[i] && Distances[i] < min)
        {
            index = i;
            min = Distances[i];
        }

    if(index != -1)
        Dijkstra(index);
}
```



Implementierung in Mathematica

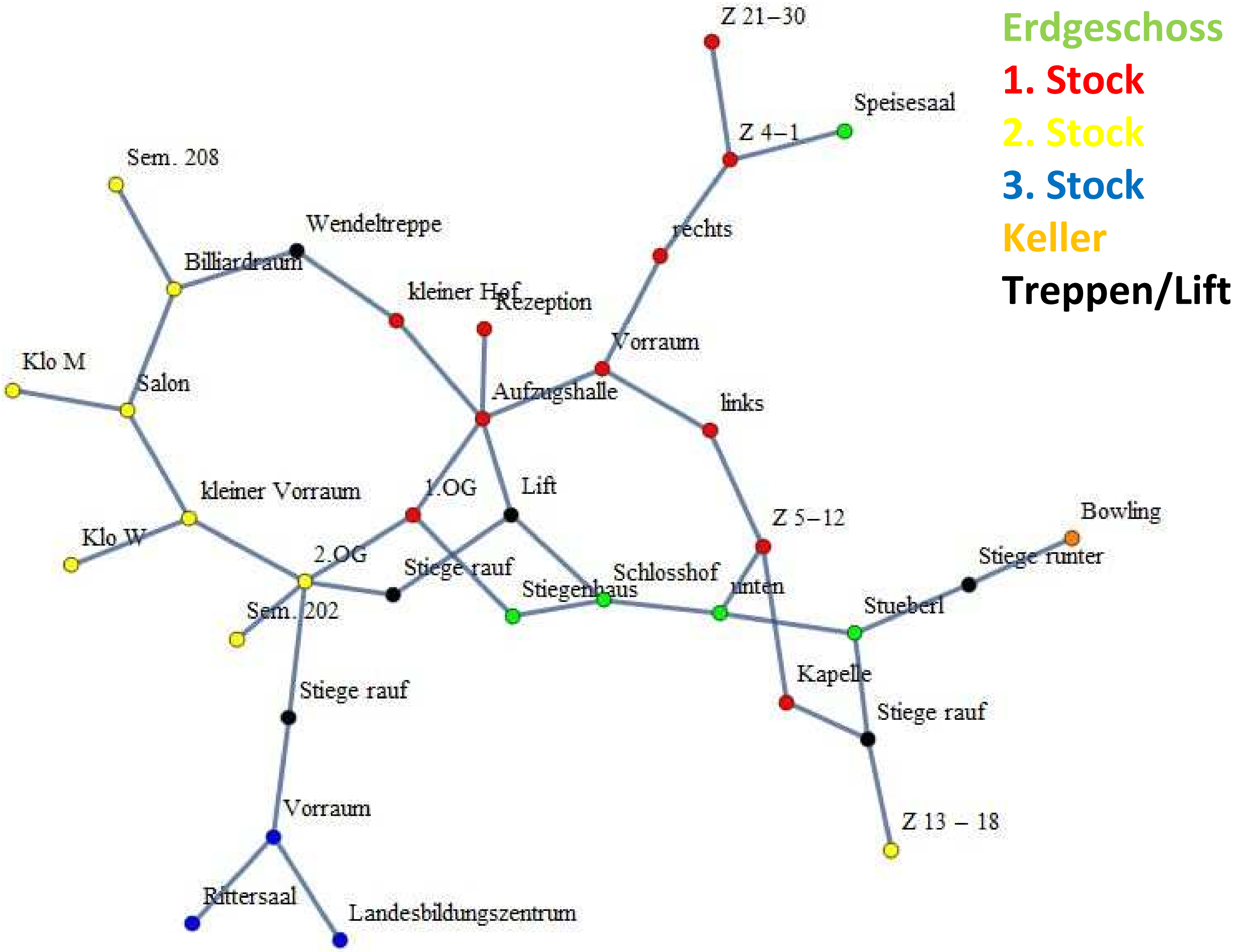
```
(*Speichert den gefundenen Wert in die fertige Liste*)

Actual[kand_] := {min = kand[[1]], value[[kand[[2]]]] = kand[[1]]}


(*Vereint die obigen Schritte in einer Funktion und gibt die fertige Liste aus*)

WholeStep[g_] := (mat = CreateMat[g]; defvalue[mat]; For[k = 1, k < length, k++, {SetPoint[mat, value]; Actual[kand];}])

(*speichert die Punkte die man entlanggehen muss um zum Punkt p zu kommen in eine Liste und die begangenen Wege in eine Liste*)
Way[g_, p_] := (WholeStep[g]; specway = {p}; For[i = 1, specway[[i]] ≠ 1, i++, AppendTo[specway, way[[(specway[[i]])]]]]; specway = Reverse[specway]; elist = {};
  For[i = 1, specway[[i]] ≠ p, i++, AppendTo[elist, specway[[i]] ⟷ specway[[i + 1]]]])

(*Erzeugt eine Wegbeschreibung mit Karte zum gewünschten Punkt*)
(*Wegbeschreibung2[g_,p_,start_]:=(Way[g,p,start];For[i=2,specway[[i]]≠p,i++,Print["Go to Point ",specway[[i]]]];Print["Go to Point ",p];HighlightGraph[g,elist])*)

Wegbeschreibung[g_, p_] := (Way[g, p]; For[i = 2, specway[[i]] ≠ p, i++, Print["Go to Point ", specway[[i]]]]; Print["Go to Point ", p]; HighlightGraph[g, elist])

WegbeschreibungS[g_, p_] := (Way[g, p]; For[i = 2, specway[[i]] ≠ p, i++, Print["Gehe zu ", PropertyValue[{g, specway[[i]]}, VertexLabels]]];
  Print["Gehe zu ", PropertyValue[{g, p}, VertexLabels]]; HighlightGraph[g, elist])

WegbeschreibungSchloss[punkt_] := Magnify[WegbeschreibungS[schloss, (Flatten[Position[labelsonly, punkt]][[1]])], 1.5]

WegbeschreibungSchloss2[punkt_, position_] := WegbeschreibungS[schloss, (Flatten[Position[labelsonly, punkt]][[position]])]
```



```
Gehe zu 1.OG

Gehe zu 2.OG

Gehe zu Stiege rauf

Gehe zu Vorraum

Gehe zu Rittersaal
```

Das Schloss
als Graph

Schloss als Graph



Erdgeschoss

1. Stock

2. Stock

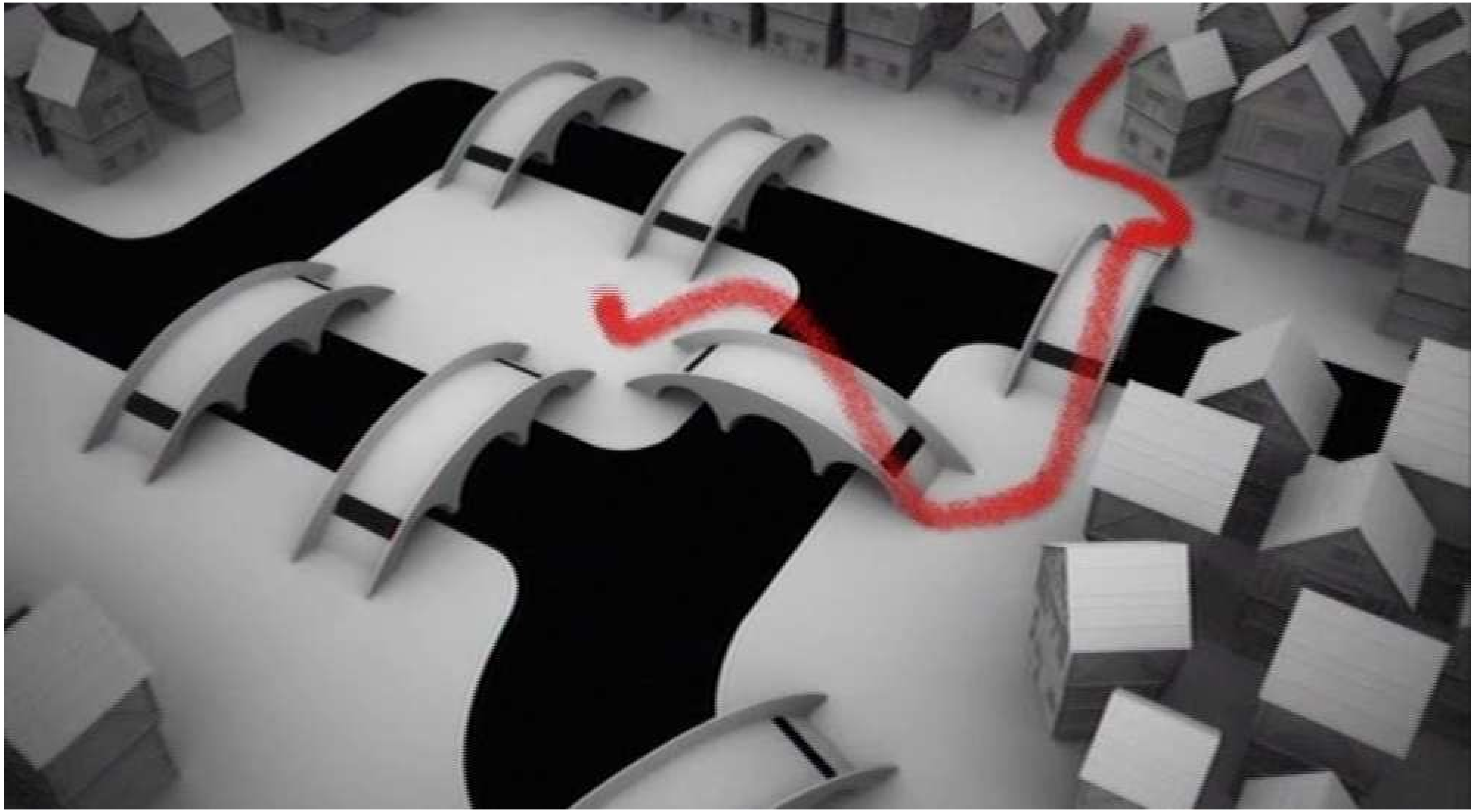
3. Stock

Keller

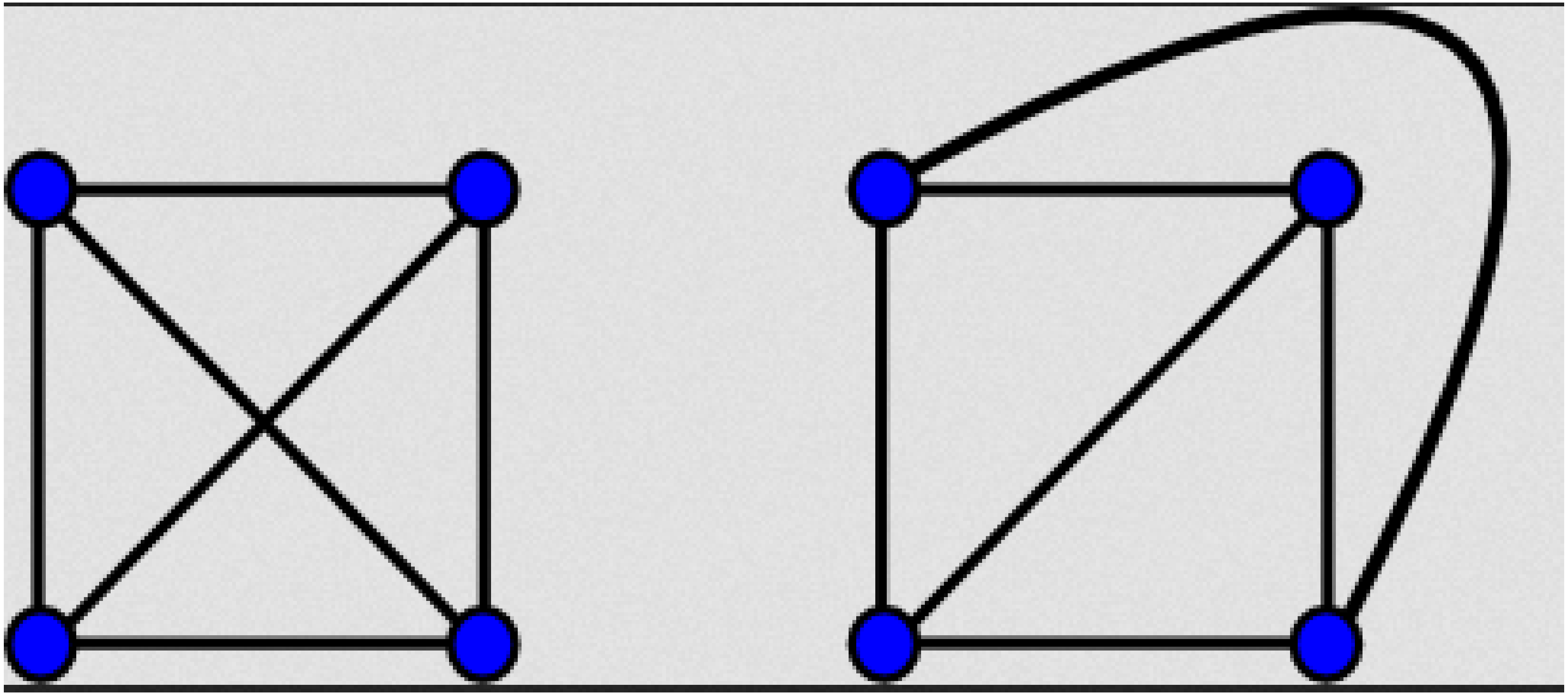
Treppen/Lift

Die Brücken
von Königsberg

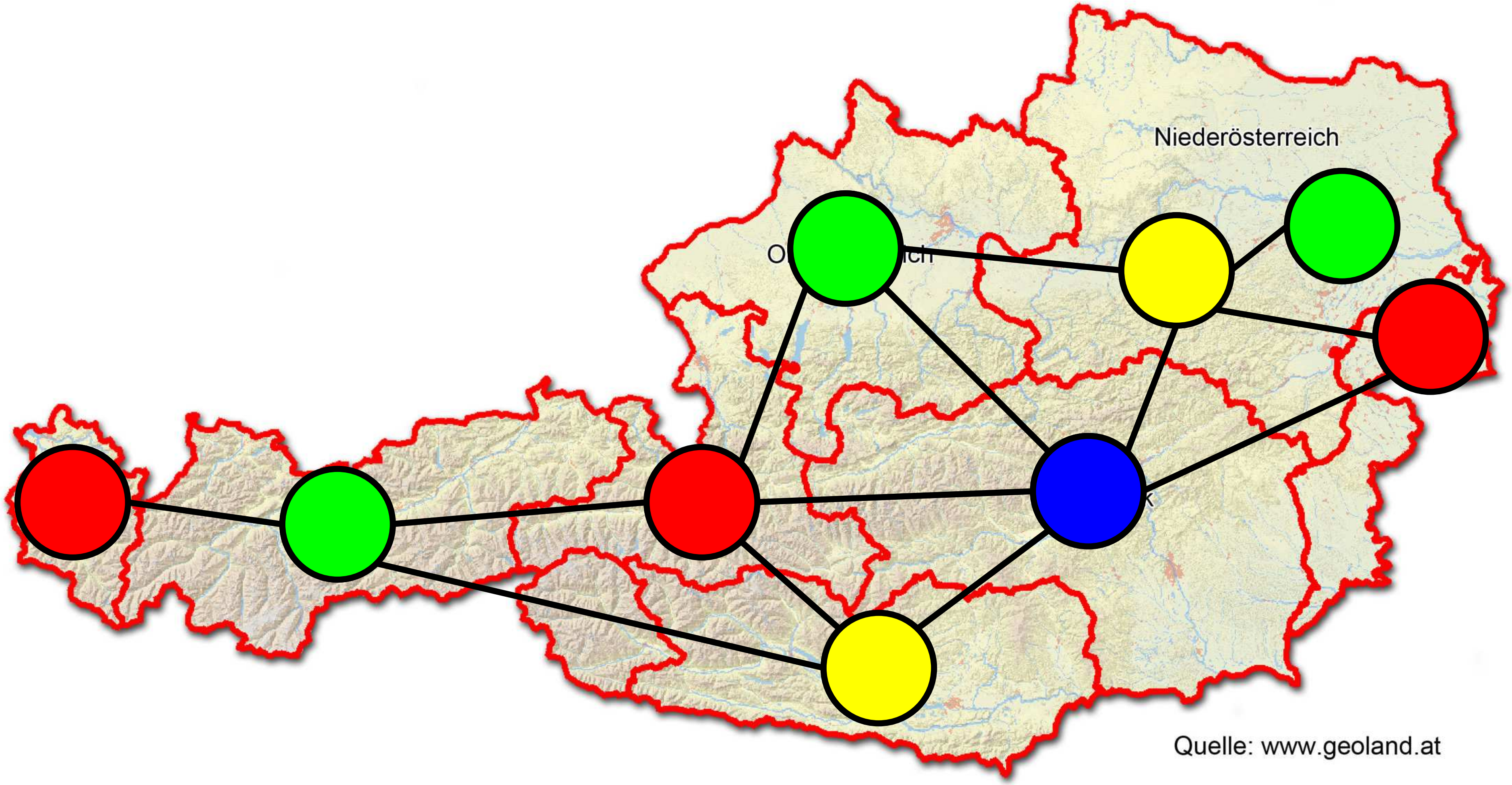
Brücken von Königsberg



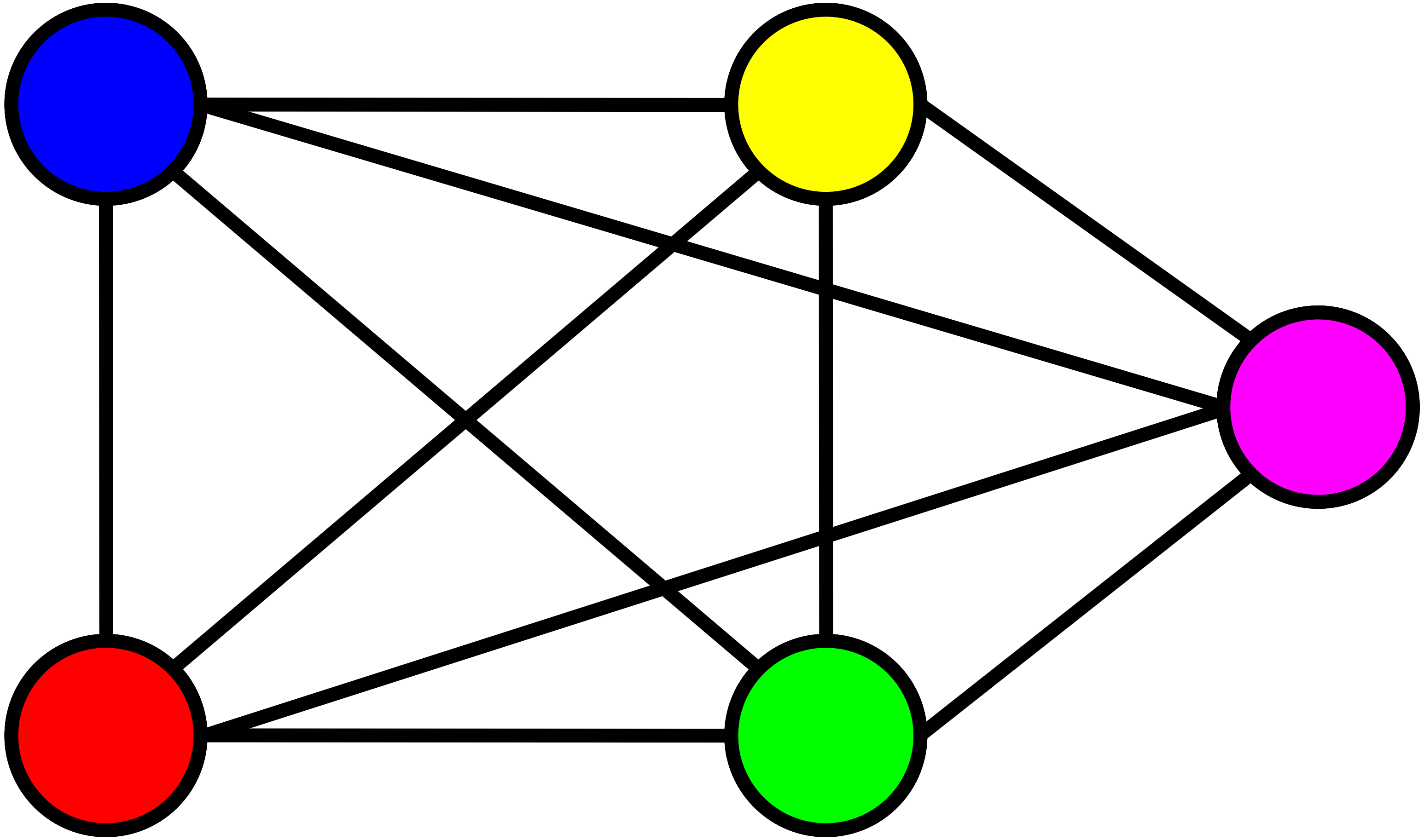
Planare Graphen



Knotenfärbung



Quelle: www.geoland.at



Was nehmen wir mit?

- Graphen
- Karten → Graphen darstellen
- kürzesten Weg
- Prinzip von Navigationssystemen
- Arbeiten mit Mathematica
- Dijkstra Algorithmus
- Implementierung in Programmiersprachen

Vielen Dank für eure Aufmerksamkeit!