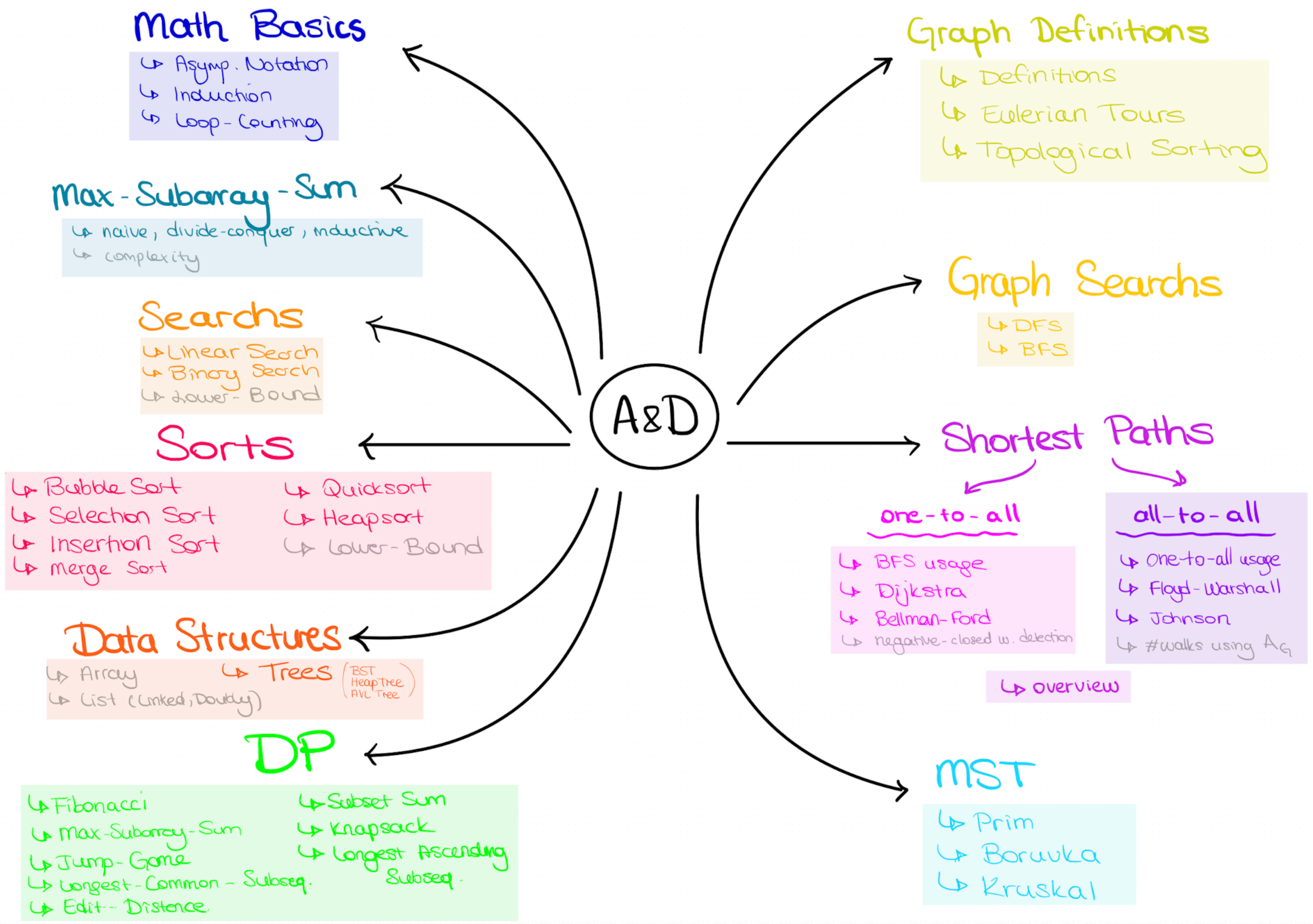
A&D

Exercise Session 12

Nil Ozer

A&D Overview


A&D
Math Basics
Asymp. Notation
Induction
Loop-Counting
Max-Subarray-Sum
naive, divide-conquer, inductive
complexity
Searchs
Linear Search
Binary Search
Lower-Bound
Sorts
Bubble Sort
Selection Sort
Insertion Sort
Merge Sort
Quicksort
Heapsort
Lower-Bound
Data Structures
Array
List (Linked, Doubly)
Trees
BST
Heap Tree
AVL Tree
DP
Fibonacci
Max-Subarray-Sum
Jump-Game
Longest-Common-Subseq.
Edit-Distance
Subset Sum
Knapsack
Longest Ascending Subseq.
Graph Definitions
Definitions
Eulerian Tours
Topological Sorting
Graph Searchs
DFS
BFS
Shortest Paths
one-to-all
BFS usage
Dijkstra
Bellman-Ford
negative-closed w. detection
all-to-all
One-to-all usage
Floyd-Warshall
Johnson
#walks using A_G
overview
MST
Prim
Boruvka
Kruskal

Outline

- Quiz
- Exercise Sheets
- MST
- Last week organization

Quiz

Exercise Sheet 7

Bonus Feedback

- 7.1 : 👏🏻👏🏻👏🏻
- 7.4 : 👏🏻👏🏻👏🏻
- 7.5 : 👏🏻👏🏻👏🏻

- Recursion Justification :(

Exercise Sheet 9

Bonus Feedback

- 9.2 : short questions about graphs 👏🏻👏🏻👏🏻
- 9.4 : 👏🏻
  - Recursion Justification :(
  - Watch out for the calculation order :
    - path starting at i : n to 1
    - path ending at i : 1 to n
- 9.5 : 👏🏻👏🏻👏🏻
  - Pre-order , revision from the dfs slides

Peergrading and rest

- Exercise Sheet 11 peergrading
  - 11.1 this week
  - Emails will be sent

- If urgent feedback is needed, send me an email !

MST

MST
Definition

Tree

- no cycles
- A tree with k vertices has k-1 edges

MST
Definition

Spanning Tree

- Spans all the vertices in the graph
- Every vertex is included in the tree.

- no cycles
- |V| - 1 edges

MST

Definition

Minimum Spanning Tree

- Among all spanning trees, MST has the minimum total weight

  (smallest possible sum of edge weights)

- Spans all the vertices in the graph
- Every vertex is included in the tree.

- no cycles
- |V| - 1 edges

Recap

Dijkstra's Algorithm

Algorithm 6 Dijkstra(s)

```
d[s] ← 0;   d[v] ← ∞  ∀v ∈ V \ {s}
S ← ∅
H ← make-heap(V); decrease-key(H, s, 0)
while S ≠ V do
    v* ← extract-min(H)
    S ← S ∪ {v*}
    for (v*, v) ∈ E, v ∉ S do
        d[v] ← min{d[v], d[v*] + c(v*, v)}
        decrease-key(H, v, d[v])
```

Runtime : O ((|V| + |E|) * log n)

weighted , positive edge weights

$c(e) \geq 0$

d[] : distance array S : visited set

H : min-heap

make-heap(V) :

Create a min heap of the vertices

extract-min(H) :

Extract (= remove and assign) the node with the minimum distance from the heap

decrease-key(H, v, k) :

Update the distance of v in heap H to the key k

MST

Prim's Algorithm - Dijkstra approach

Runtime : O ((|V| + |E|) * log n)

Algorithm 6 Dijkstra(s)

$\mathrm{d}[s] \leftarrow 0; \quad \mathrm{d}[v] \leftarrow \infty \quad \forall v \in V \setminus \{s\}$
$S \leftarrow \varnothing$
$H \leftarrow$ make-heap(V); decrease-key($H, s, 0$)
while $S \neq V$ **do**
  $v^* \leftarrow$ extract-min(H)
  $S \leftarrow S \cup \{v^*\}$
  for $(v^*, v) \in E,\ v \notin S$ **do**
    $\mathrm{d}[v] \leftarrow \min\{\mathrm{d}[v], \mathrm{d}[v^*] + c(v^*, v)\}$
    decrease-key($H, v, \mathrm{d}[v]$)

Algorithm 10 Prim(G, s) (mit min-heap)

$H \leftarrow$ make-heap(V, ∞), $S \leftarrow \varnothing$
$\mathrm{d}[s] \leftarrow 0; \quad \mathrm{d}[v] \leftarrow \infty \quad \forall v \in V \setminus \{s\}$
decrease-key($H, s, 0$)
while $H \neq \varnothing$ **do**
  $v^* \leftarrow$ extract-min(H)
  $S \leftarrow S \cup \{v^*\}$
  for $v^*v \in E,\ v \notin S$ **do**
    $\mathrm{d}[v] \leftarrow \min\{\mathrm{d}[v], c(v^*, v)\}$
    decrease-key($H, v, \mathrm{d}[v]$)

MST

Prim's Algorithm - connected components approach

Runtime : O ((|V| + |E|) * log n)

Algorithm 9 Prim(G, s) (allgemeine Form)

$F \leftarrow \varnothing$
$S \leftarrow \{s\}$
while F nicht Spannbaum **do**
    $u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
    $F \leftarrow F \cup \{u^*v^*\}$
    $S \leftarrow S \cup \{v^*\}$

MST

Prim's Algorithm - connected components approach

Runtime : O ((|V| + |E|) * log n)

Algorithm 9 Prim(G, s) (allgemeine Form)

$F \leftarrow \varnothing$
$S \leftarrow \{s\}$
while F nicht Spannbaum **do**
    $u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
    $F \leftarrow F \cup \{u^*v^*\}$
    $S \leftarrow S \cup \{v^*\}$

F : edges of the MST

S : connected component set

MST

Prim's Algorithm - connected components approach

Runtime : O ((|V| + |E|) * log n)

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t. u* is in S but v* is not

MST

Prim's Algorithm

F : edges of the MST

S : connected component set

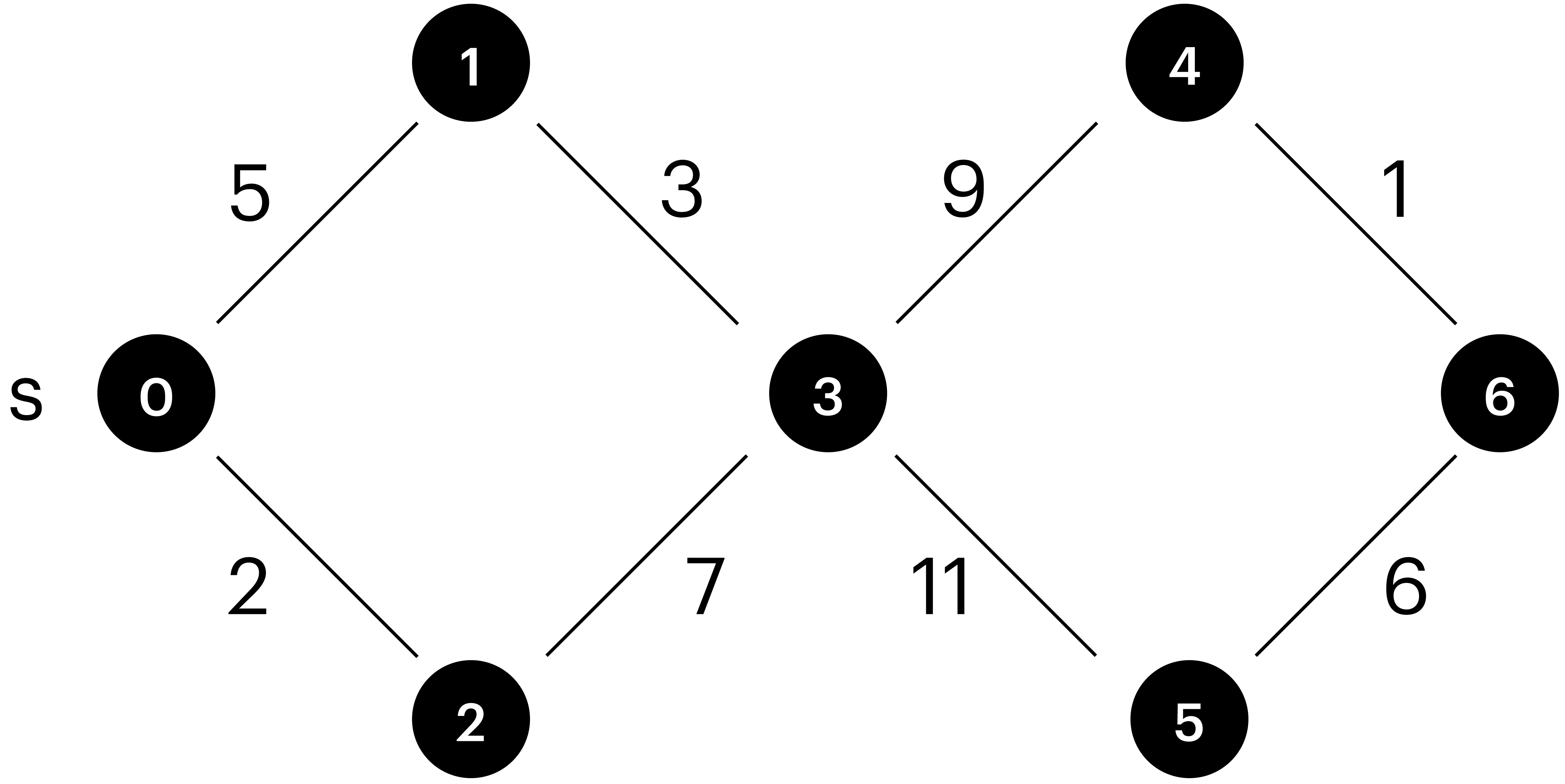
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

```
1: F ← ∅
2: S ← {s}
3: while F nicht Spannbaum do
4:     u*v* ← minimale Kante an S    (u* ∈ S, v* ∉ S)
5:     F ← F ∪ {u*v*}
6:     S ← S ∪ {v*}
```

F :

S :



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

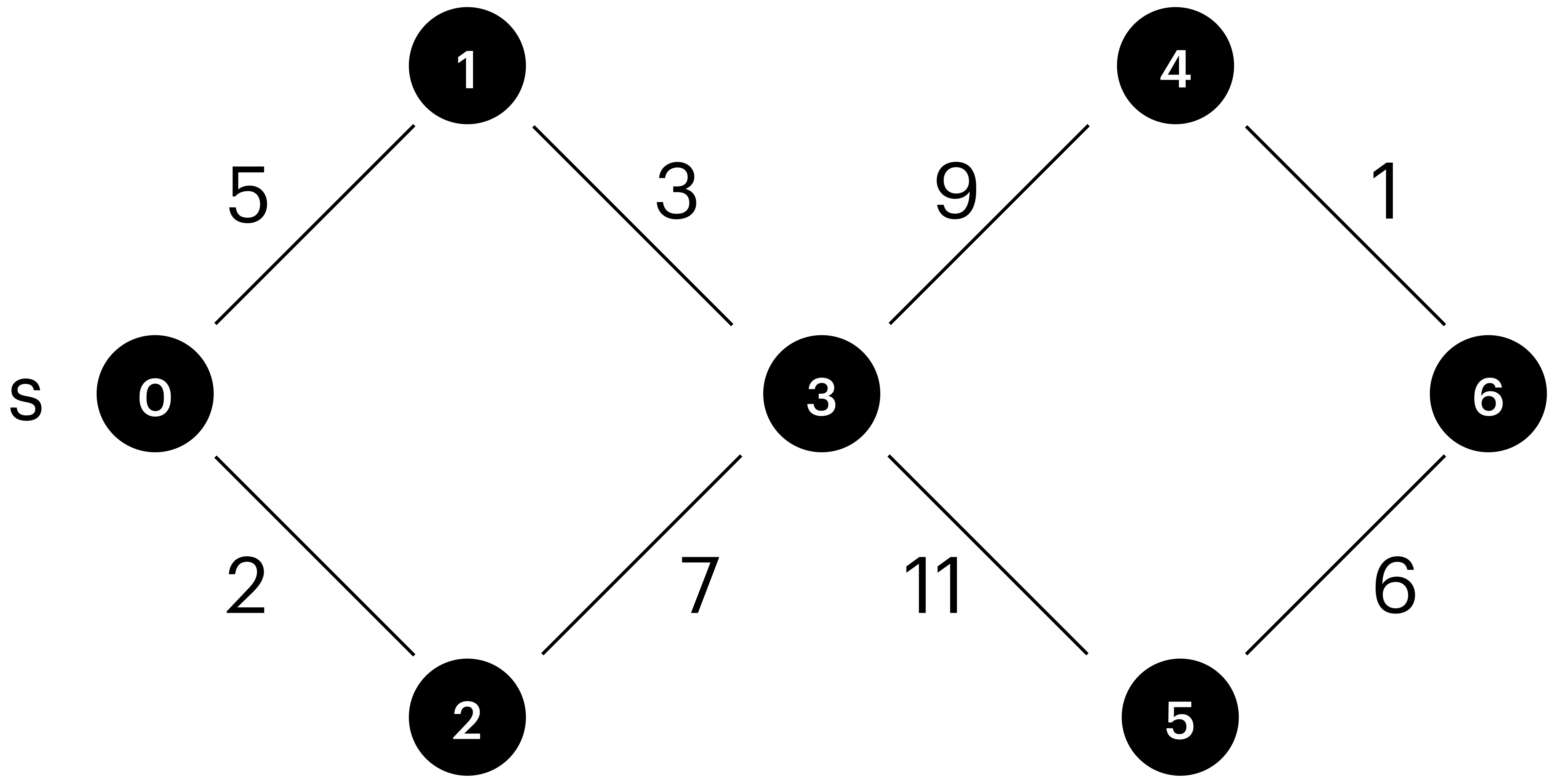
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
5: $F \leftarrow F \cup \{u^*v^*\}$
6: $S \leftarrow S \cup \{v^*\}$

F : $\varnothing$

S : { 0 }



MST

Prim’s Algorithm

F : edges of the MST

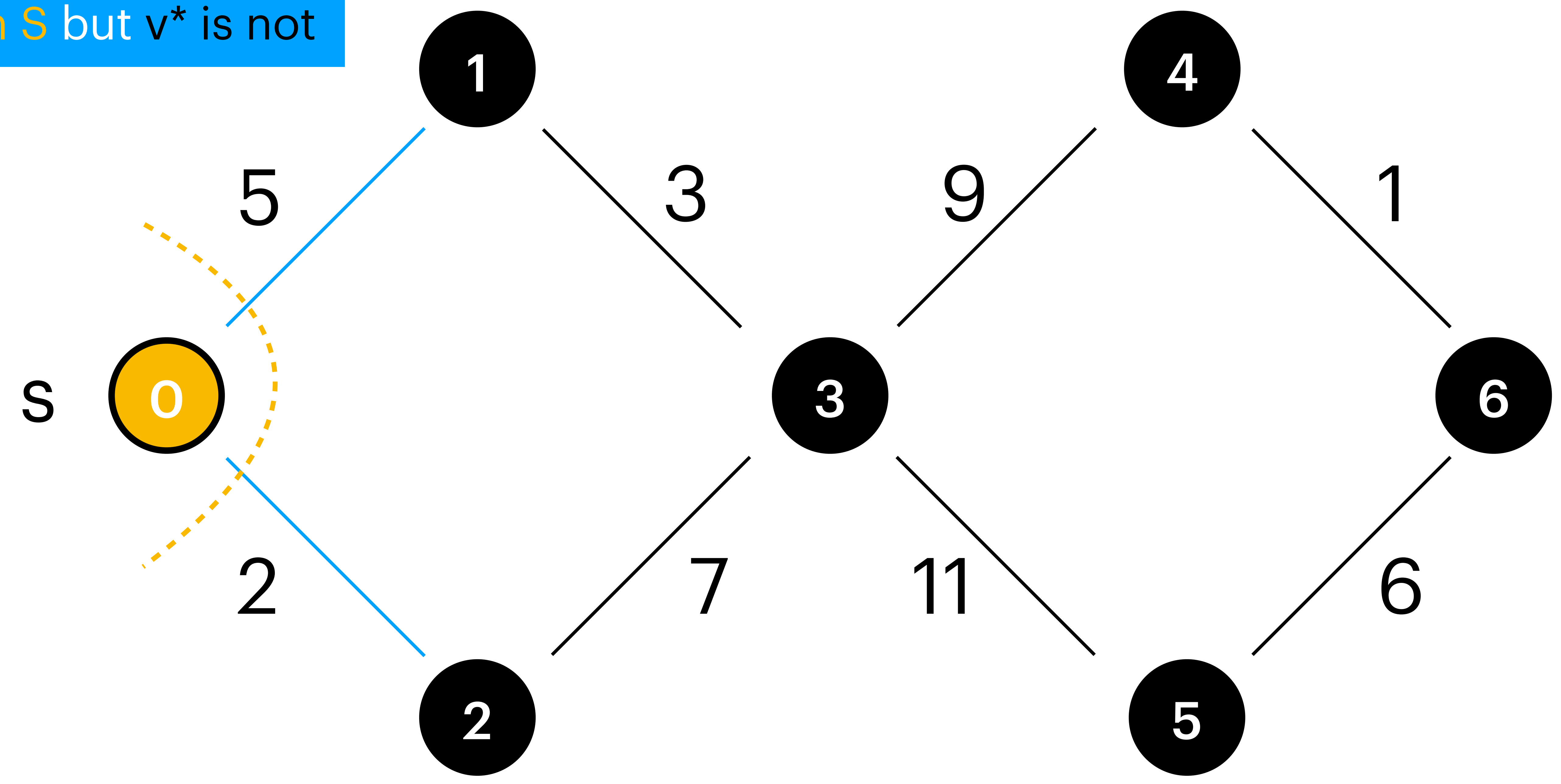
S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
5: $F \leftarrow F \cup \{u^*v^*\}$
6: $S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not



F : $\varnothing$

S : { 0 }

MST

Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t. u* is in S but v* is not

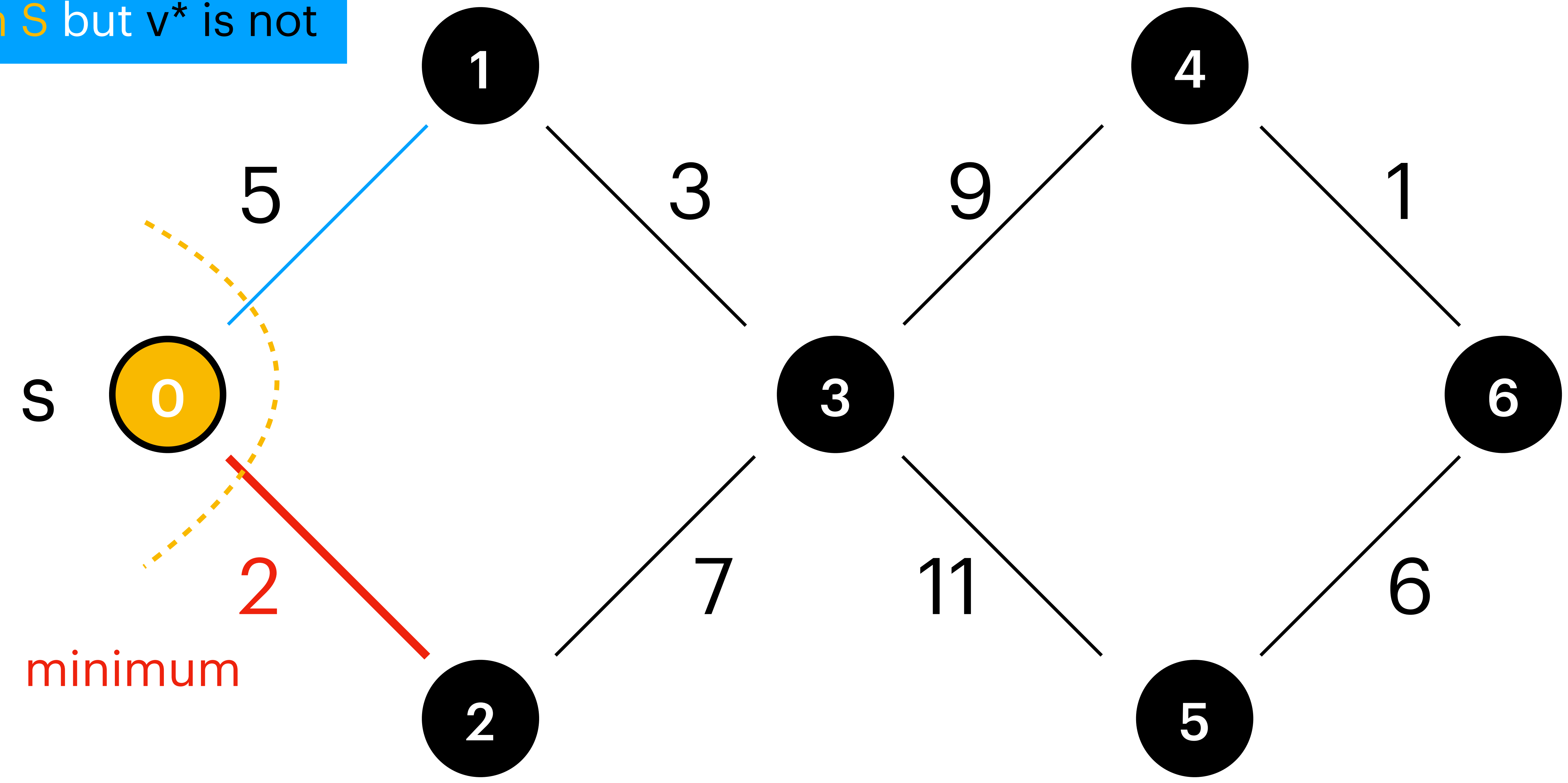
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not

F : $\varnothing$

S : { 0 }



MST

Prim’s Algorithm

F : edges of the MST

S : connected component set

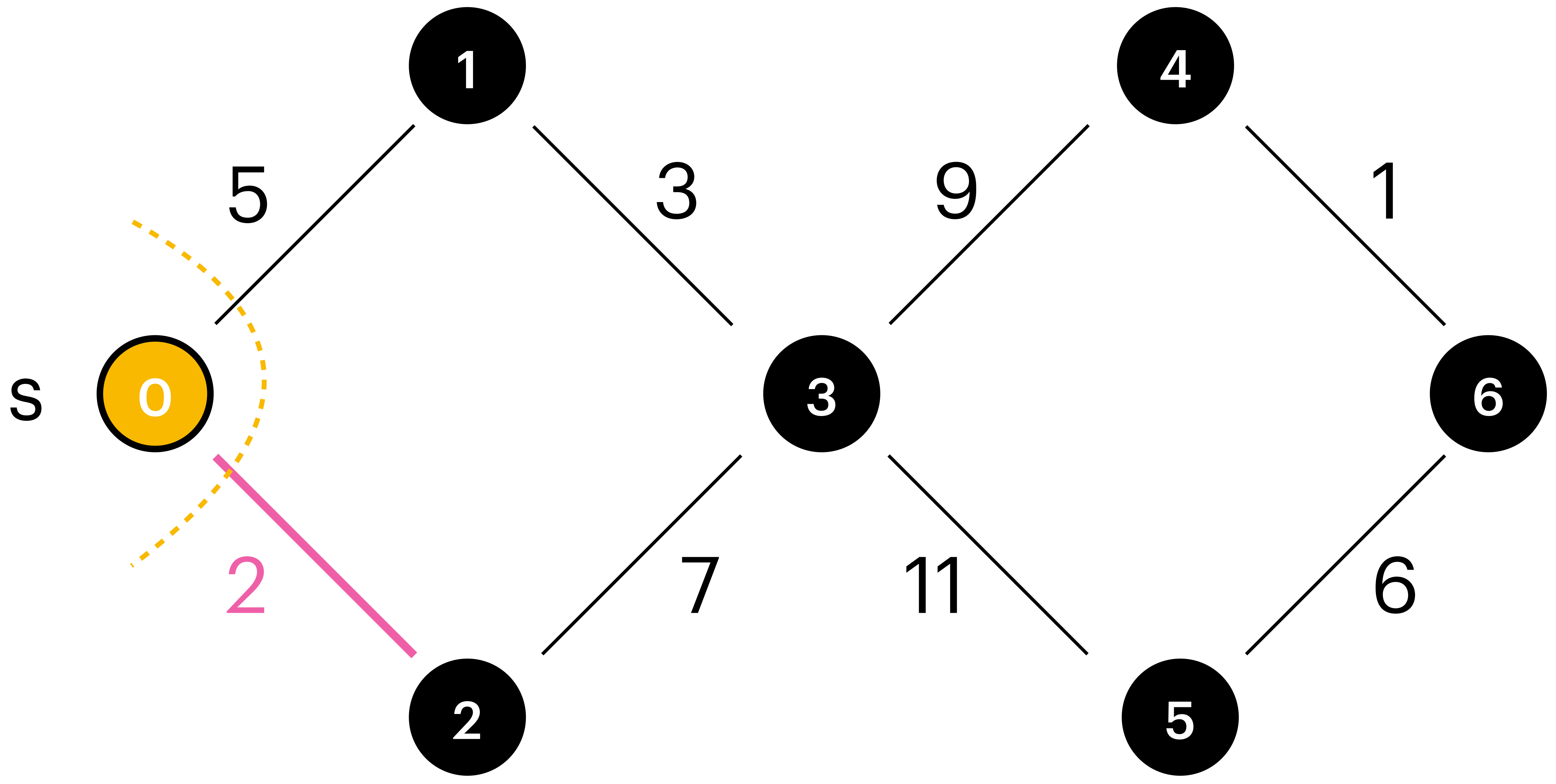
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} }

S : { 0 }



MST
Prim's Algorithm

F : edges of the MST

S : connected component set

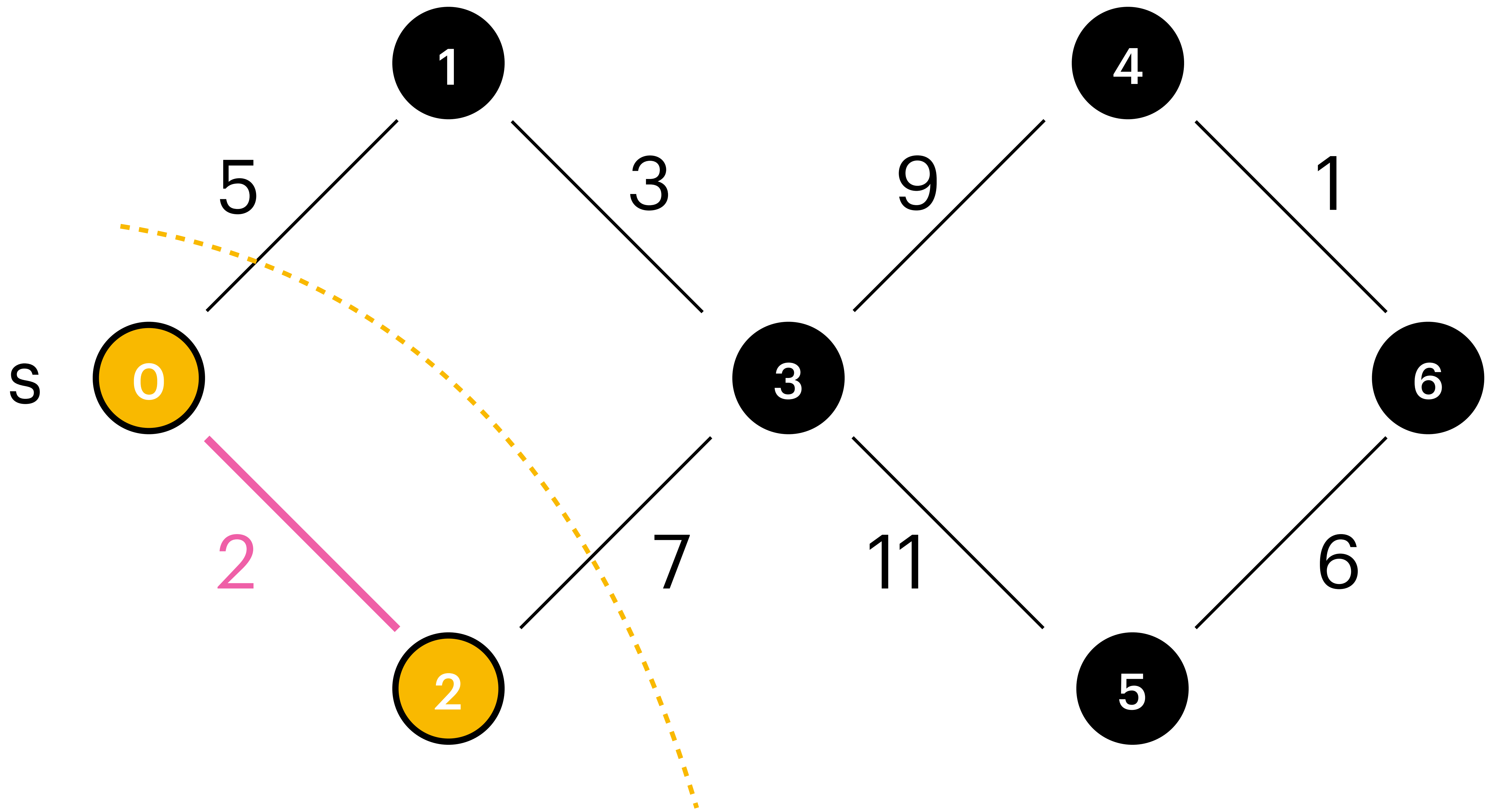
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} }

S : { 0 , 2 }



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

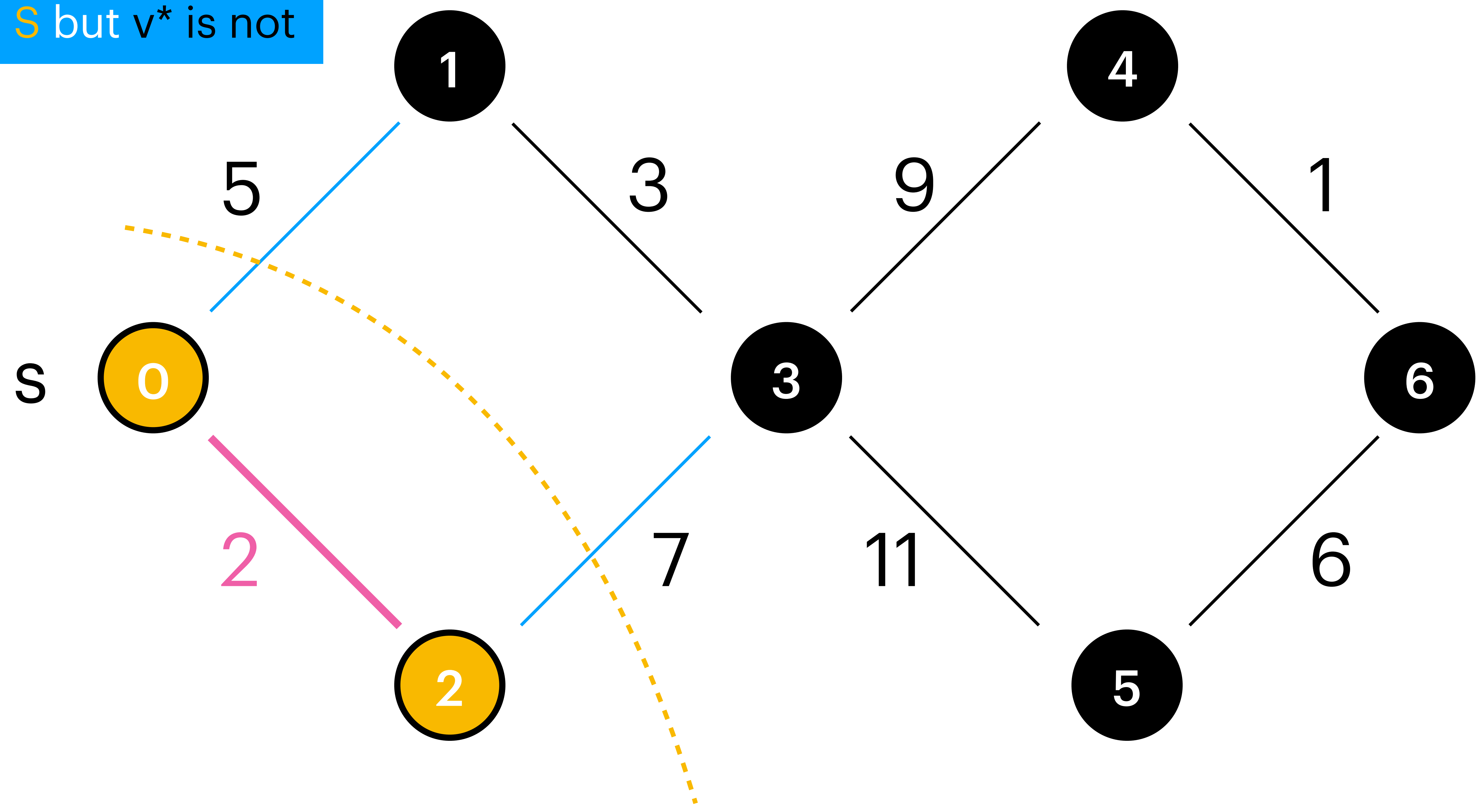
Algorithm 9 Prim(G, s) (allgemeine Form)

```
1: F ← ∅
2: S ← {s}
3: while F nicht Spannbaum do
4:     u*v* ← minimale Kante an S   (u* ∈ S, v* ∉ S)
5:     F ← F ∪ {u*v*}
6:     S ← S ∪ {v*}
```

edges {u* v*} s.t.
u* is in S but v* is not

F : { {0,2} }

S : { 0 , 2 }



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

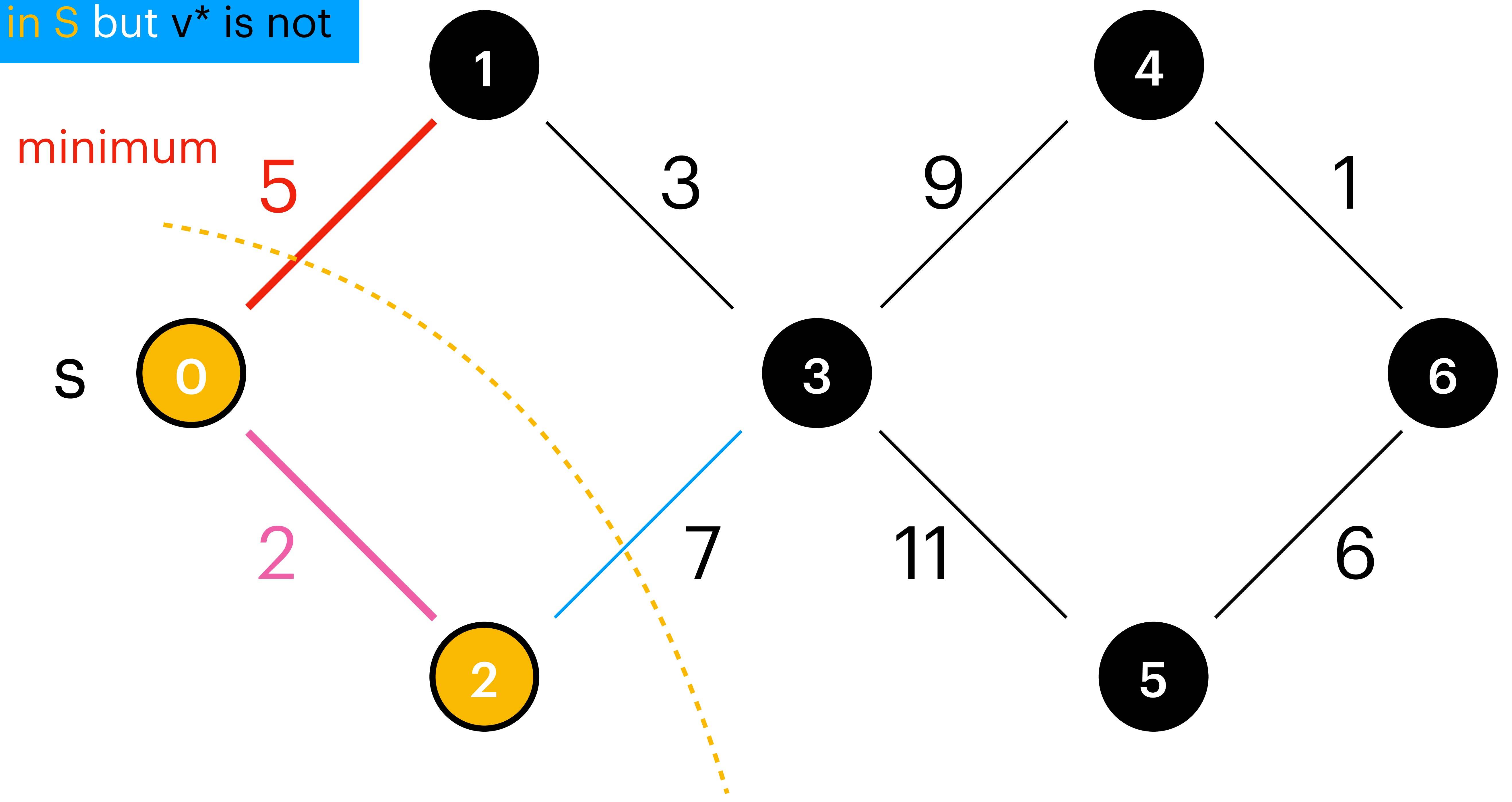
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^* v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^* v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} }

S : { 0 , 2 }

edges {u* v*} s.t.
u* is in S but v* is not



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

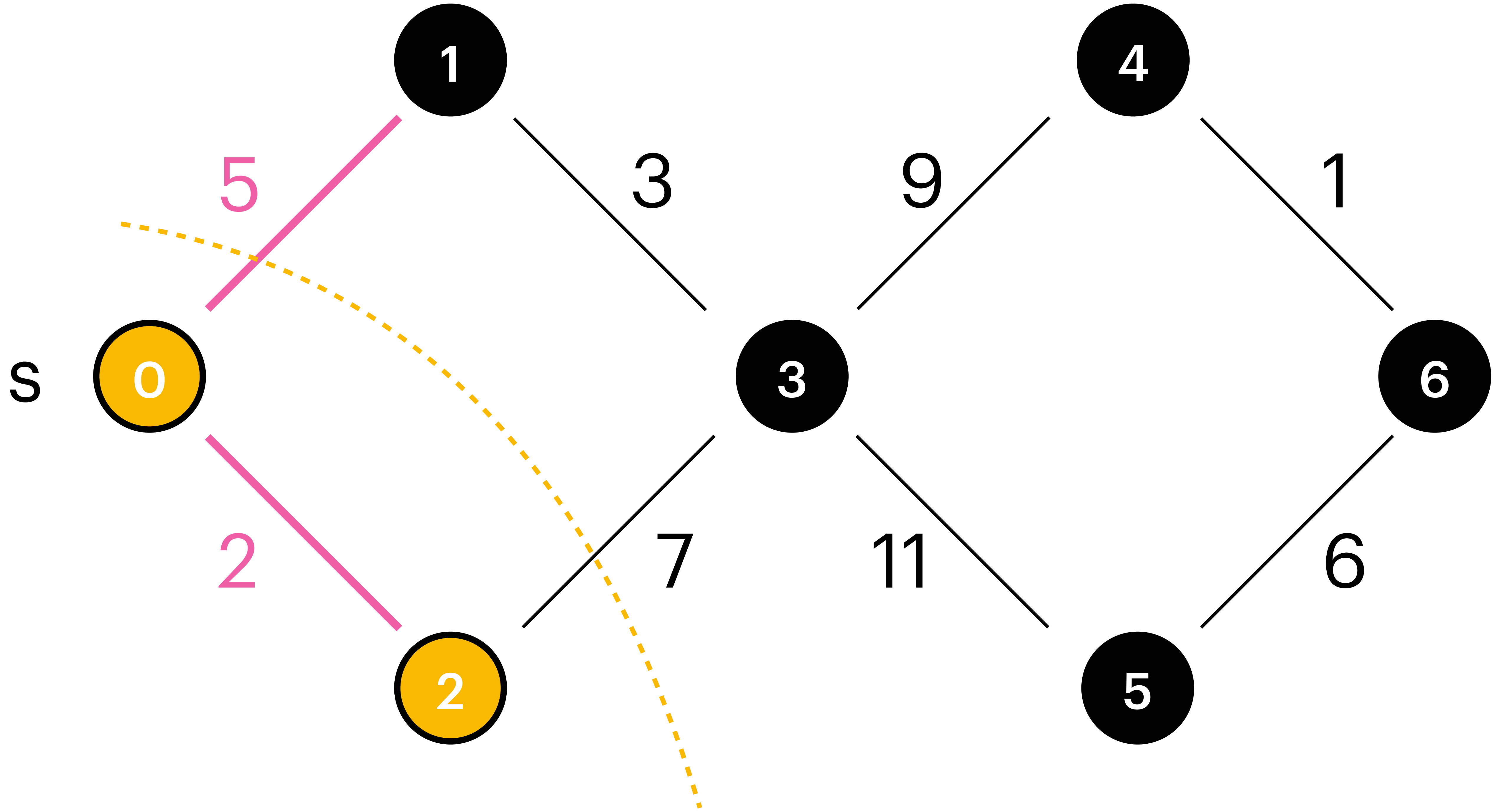
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
5: $F \leftarrow F \cup \{u^*v^*\}$
6: $S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} }

S : { 0 , 2 }



MST
Prim's Algorithm

F : edges of the MST

S : connected component set

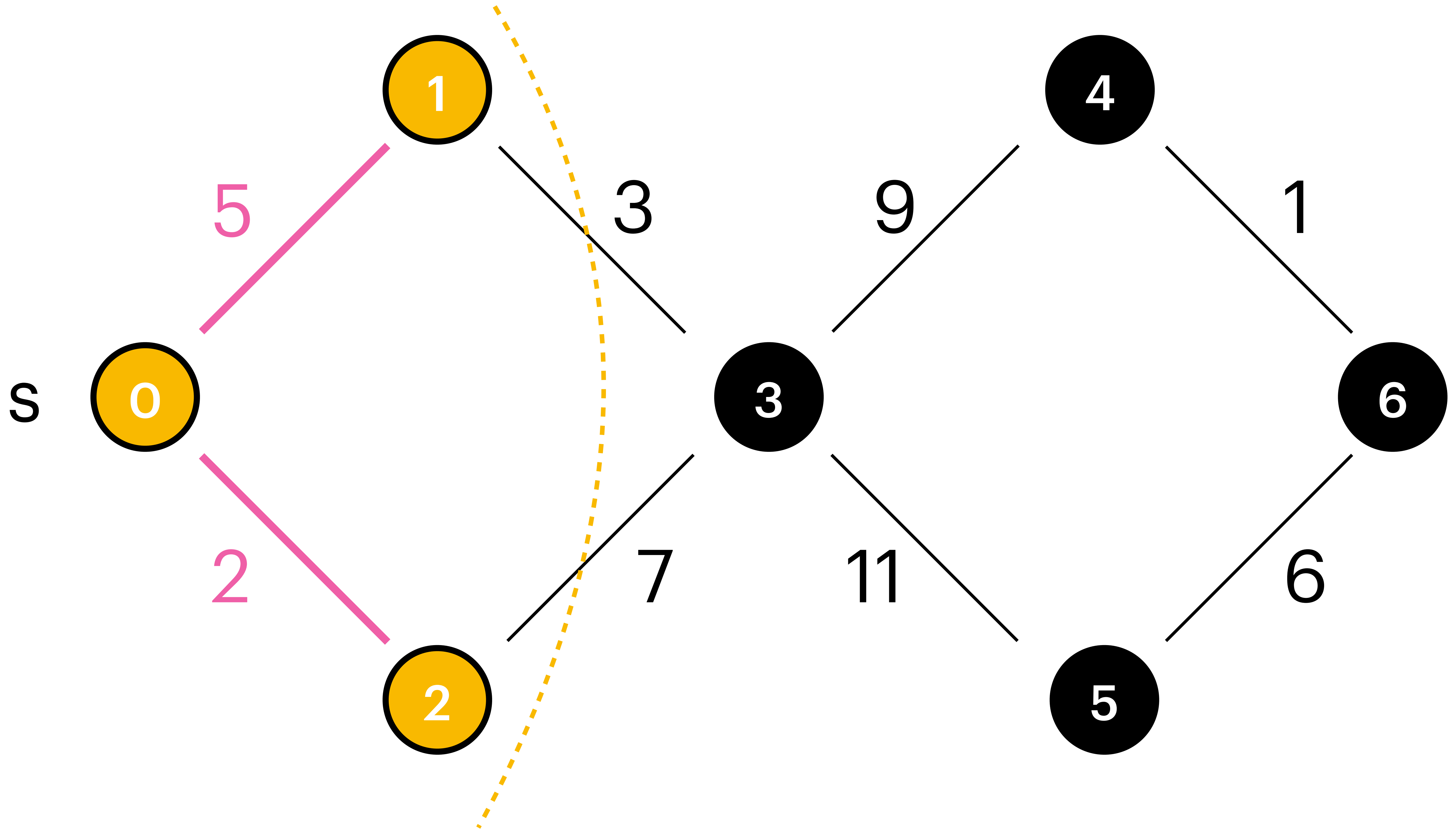
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \emptyset$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} }

S : { 0 , 2 , 1 }



MST

Prim's Algorithm

F : edges of the MST

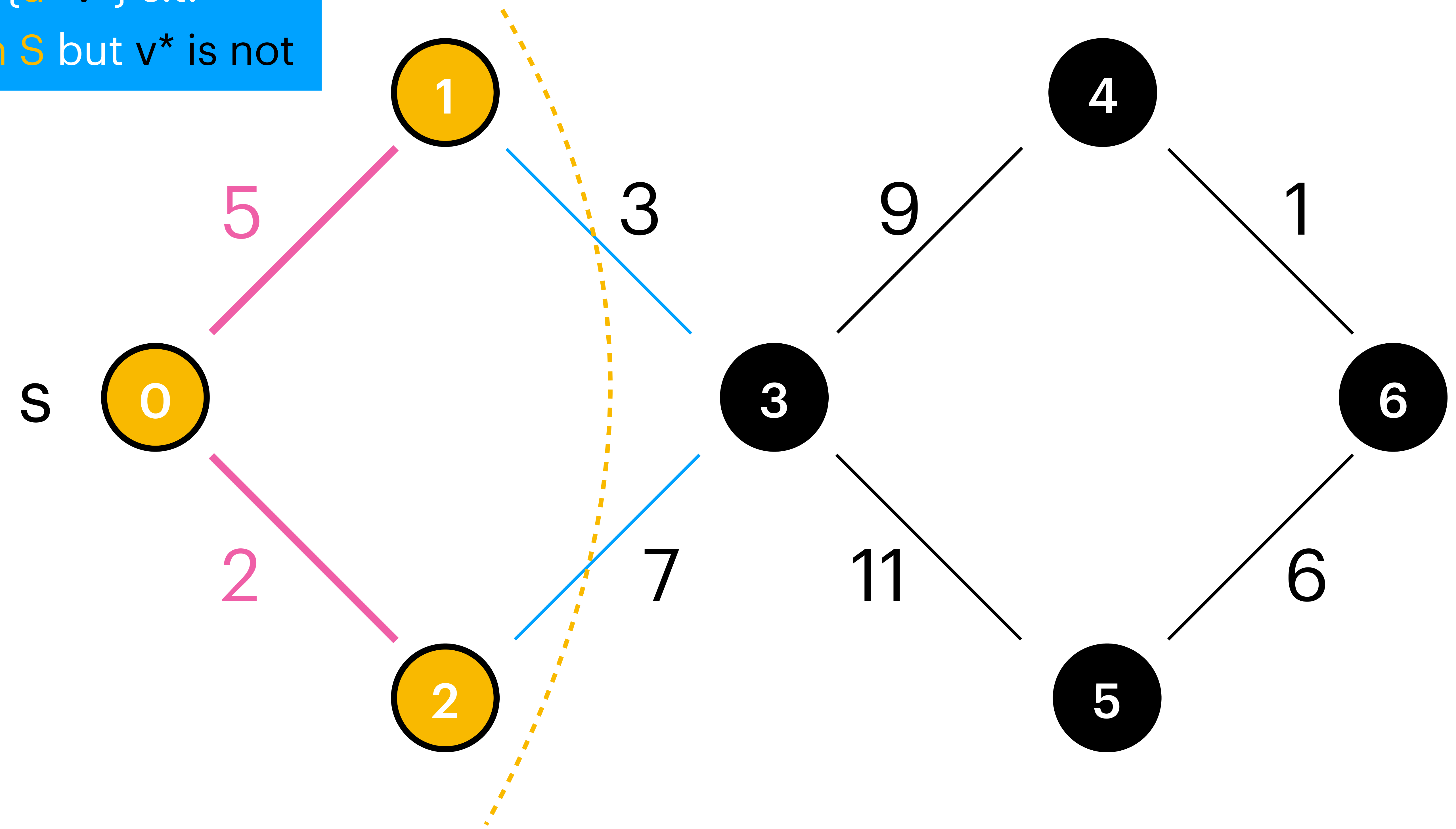
S : connected component set

4 : find the minimum edge {u*,v*} s.t. u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t. u* is in S but v* is not



F : { {0,2} , {0,1} }

S : { 0 , 2 , 1 }

MST

Prim's Algorithm

F : edges of the MST

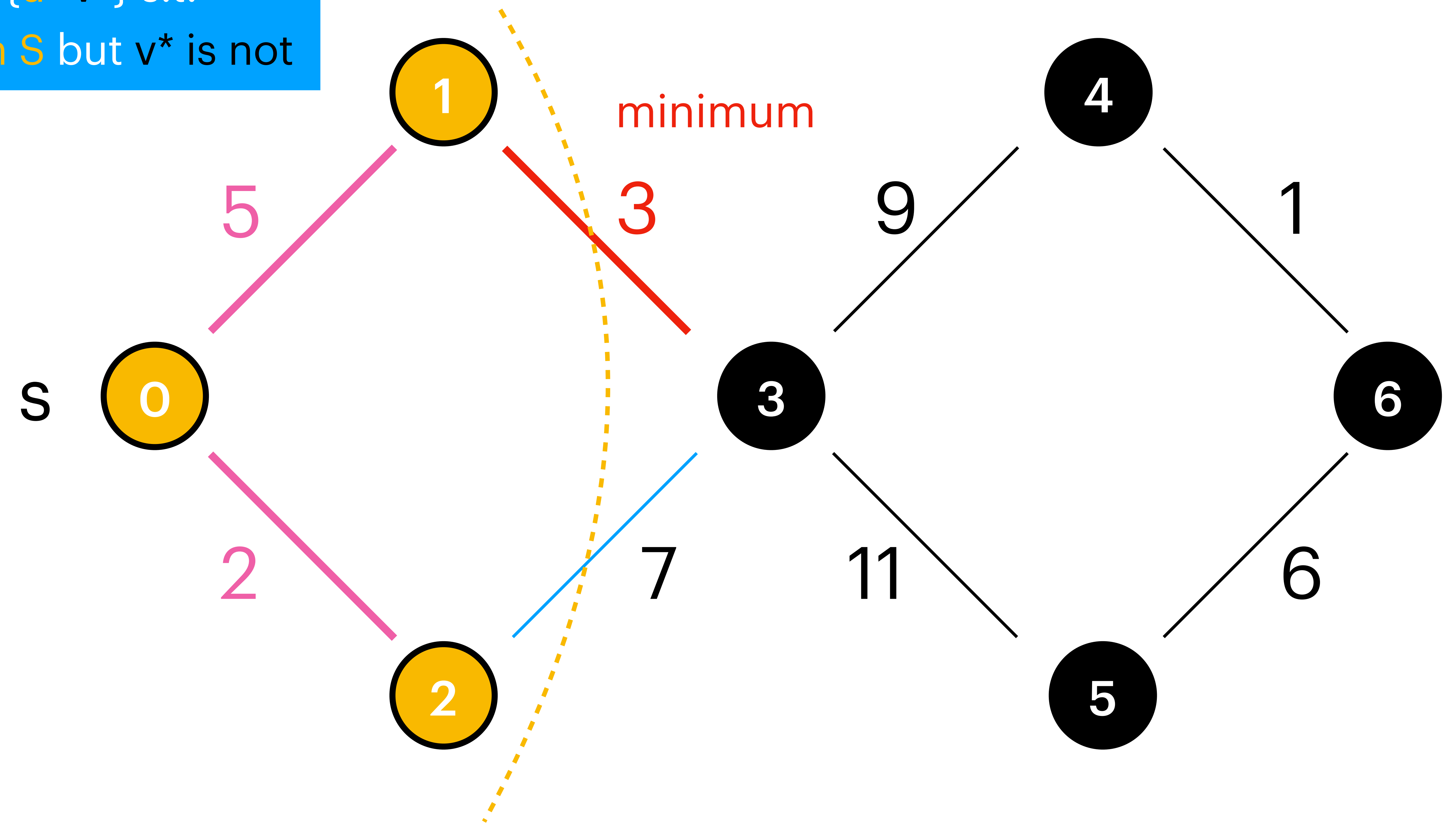
S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
5: $F \leftarrow F \cup \{u^*v^*\}$
6: $S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not



F : { {0,2} , {0,1} }

S : { 0 , 2 , 1 }

MST

Prim's Algorithm

F : edges of the MST

S : connected component set

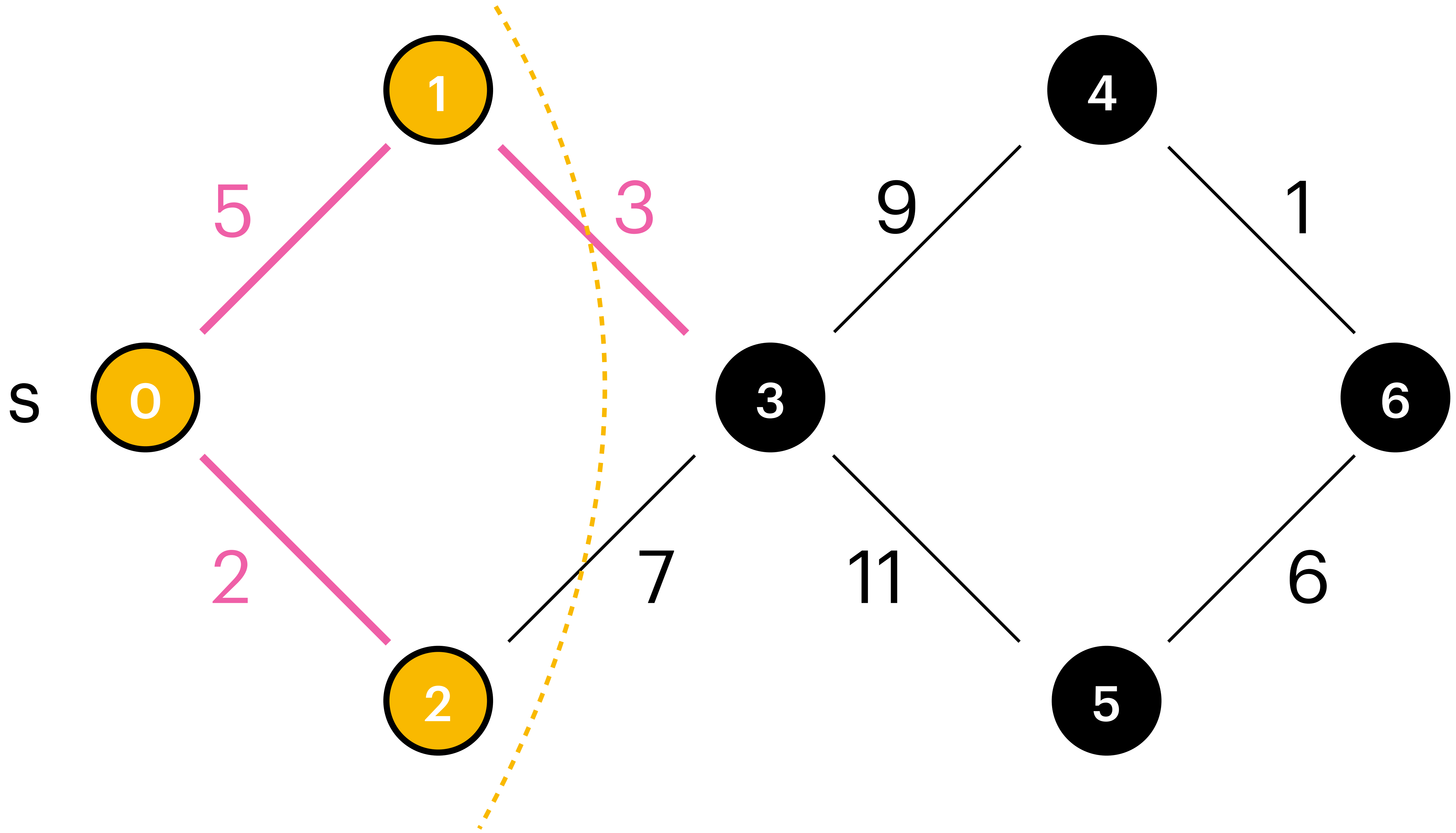
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
5: $F \leftarrow F \cup \{u^*v^*\}$
6: $S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} }

S : { 0 , 2 , 1 }



MST
Prim's Algorithm

F : edges of the MST

S : connected component set

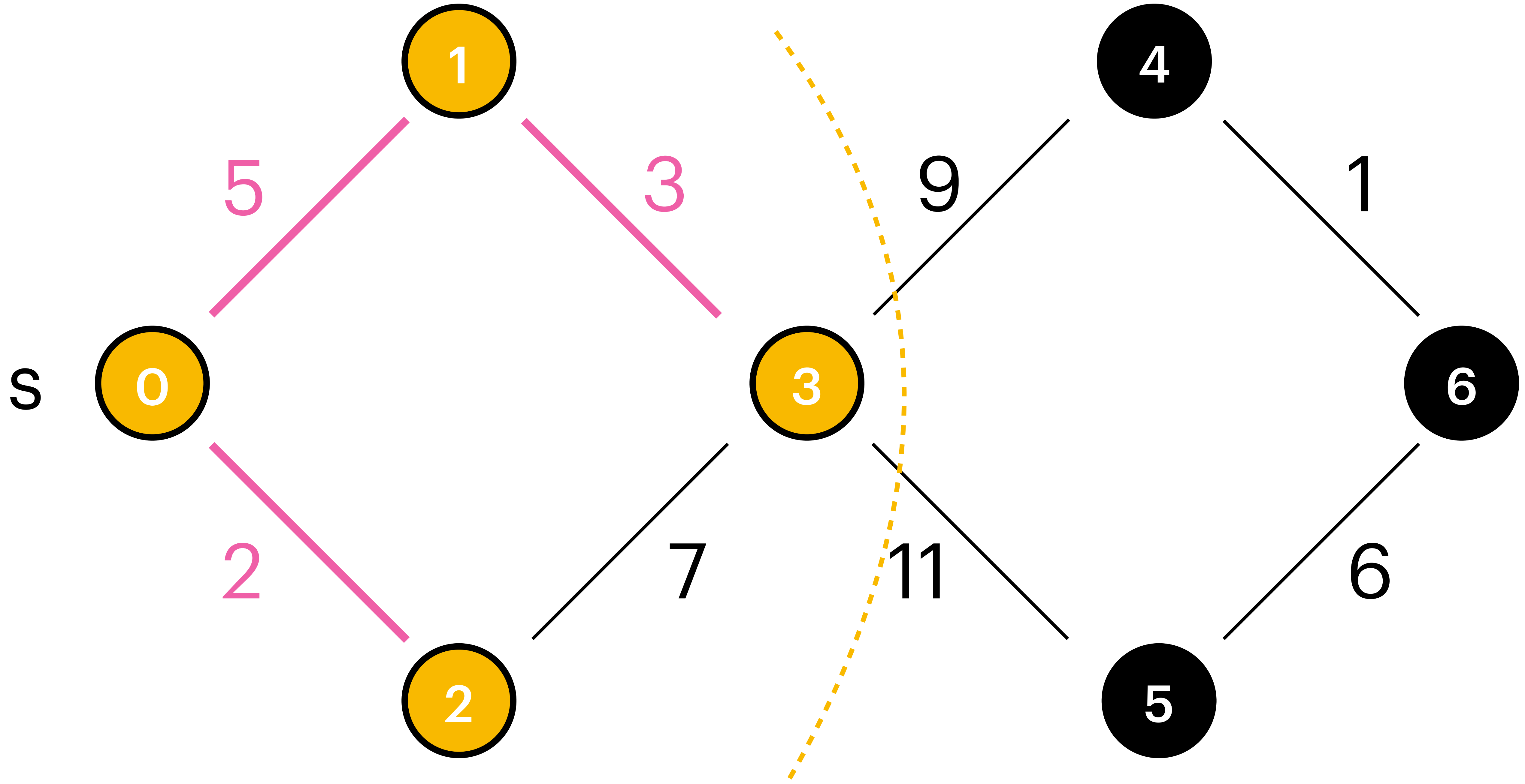
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \emptyset$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} }

S : { 0 , 2 , 1 , 3}



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t. u* is in S but v* is not

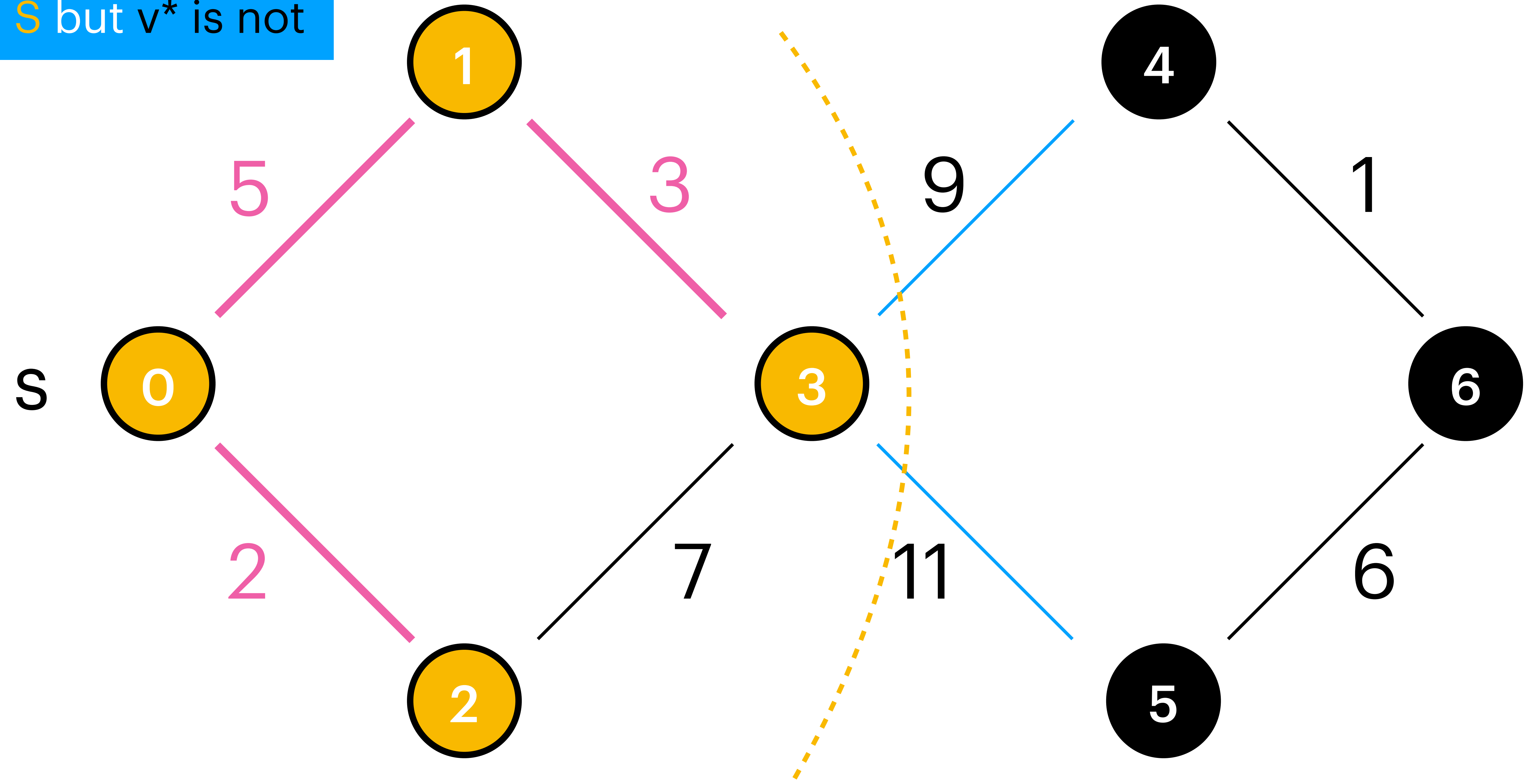
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not

F : { {0,2} , {0,1} , {1,3} }

S : { 0 , 2 , 1 , 3}



MST

Prim’s Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

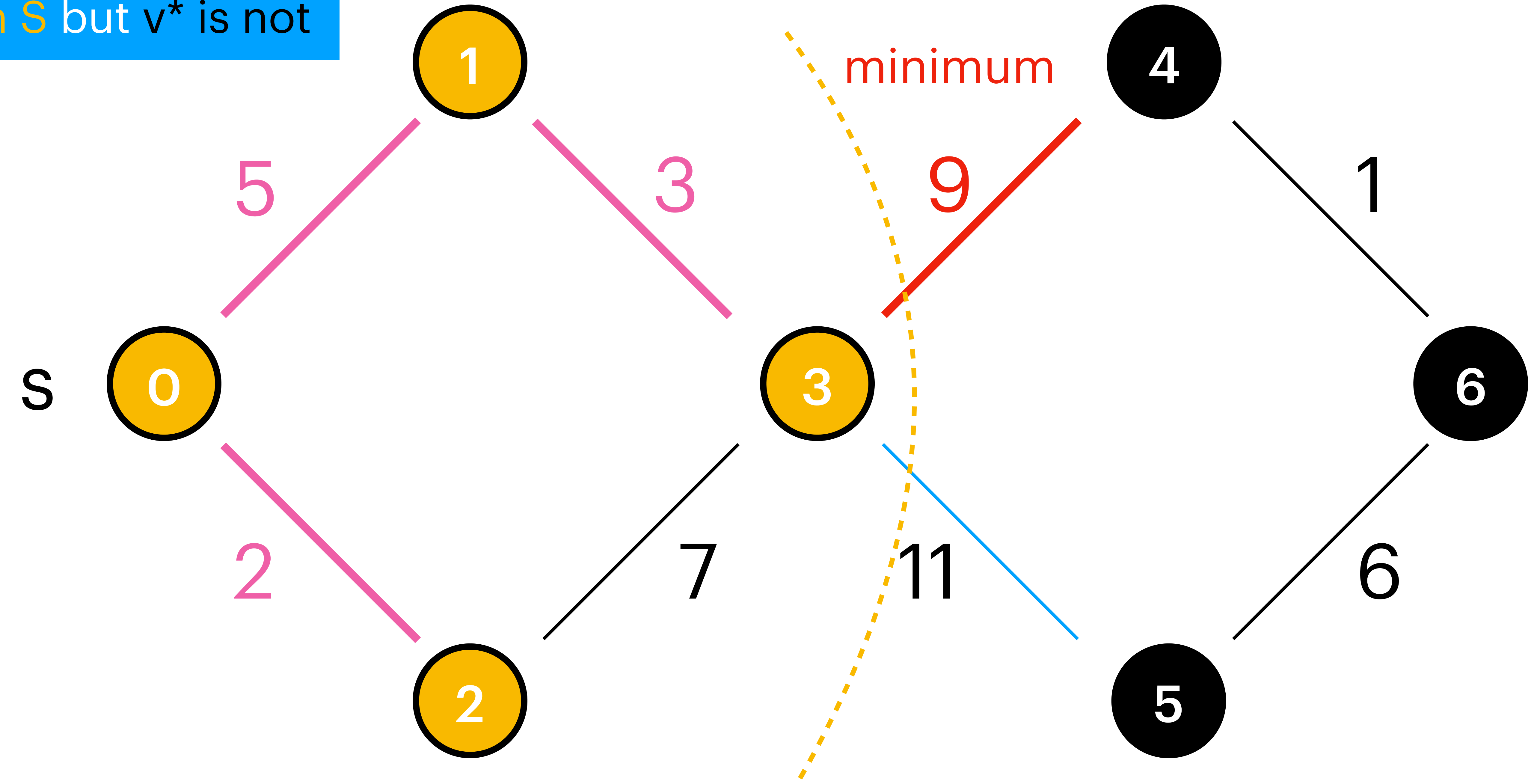
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not

F : { {0,2} , {0,1} , {1,3} }

S : { 0 , 2 , 1 , 3}



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

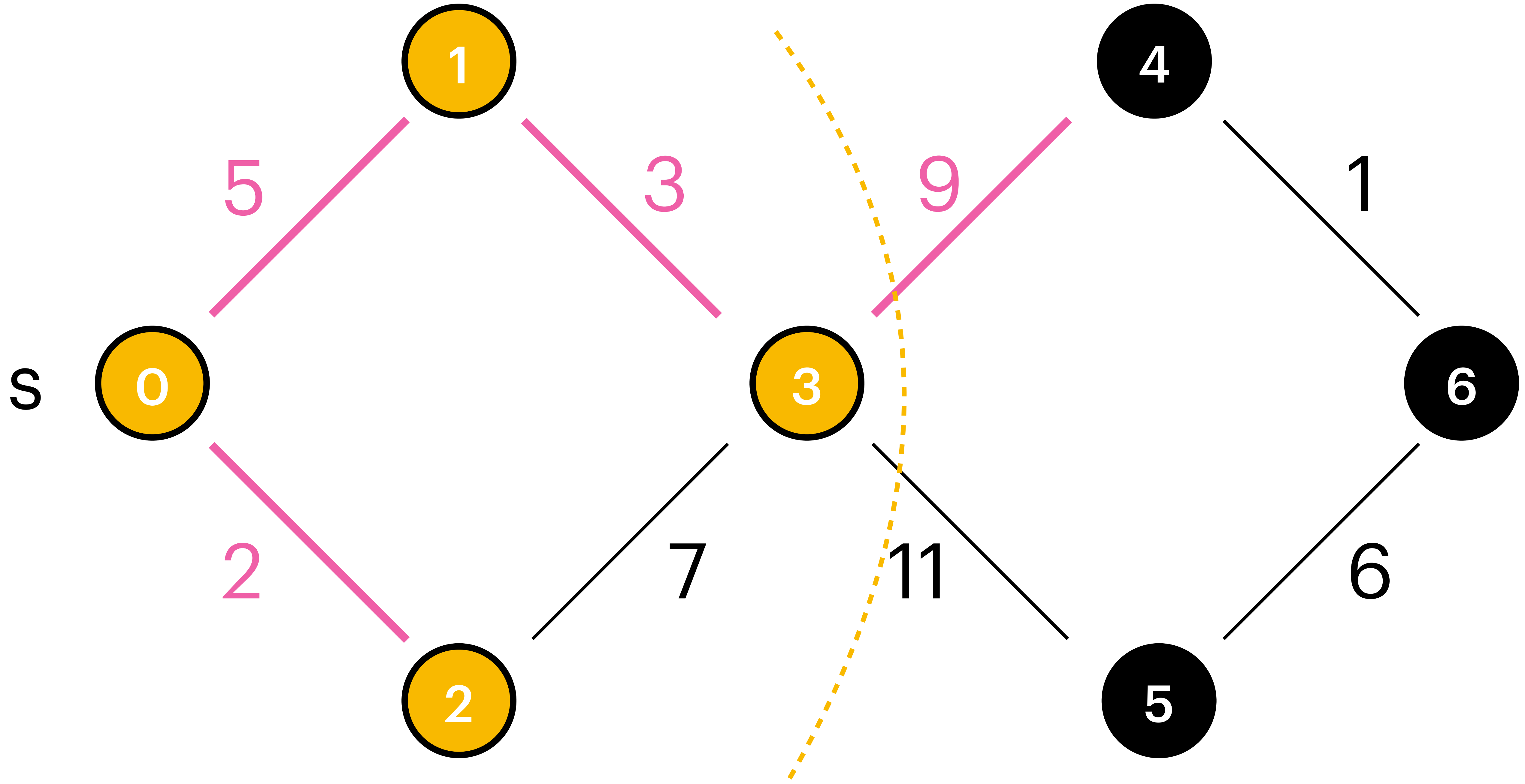
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} }

S : { 0 , 2 , 1 , 3}



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

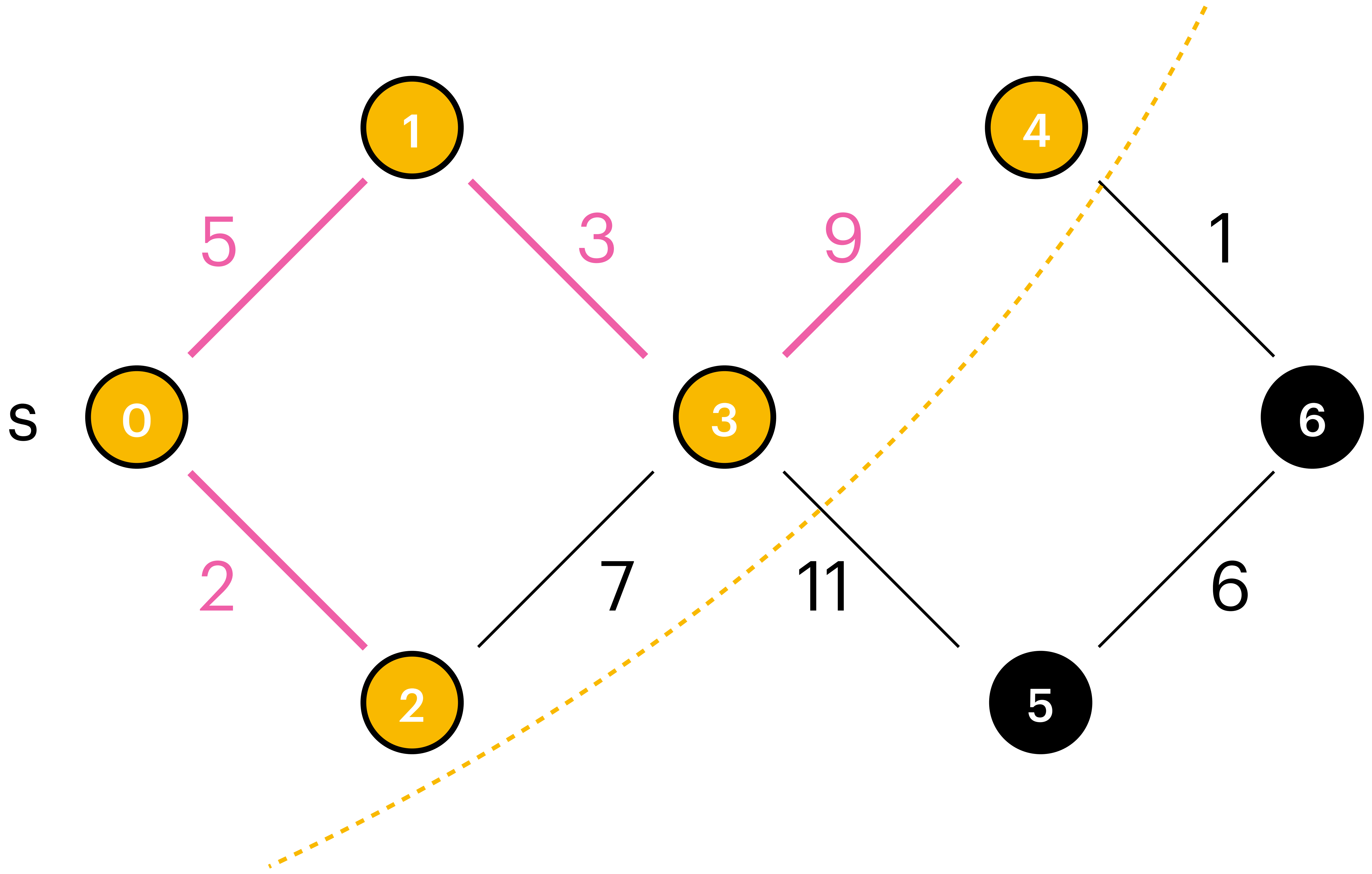
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} }

S : { 0 , 2 , 1 , 3 , 4 }



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

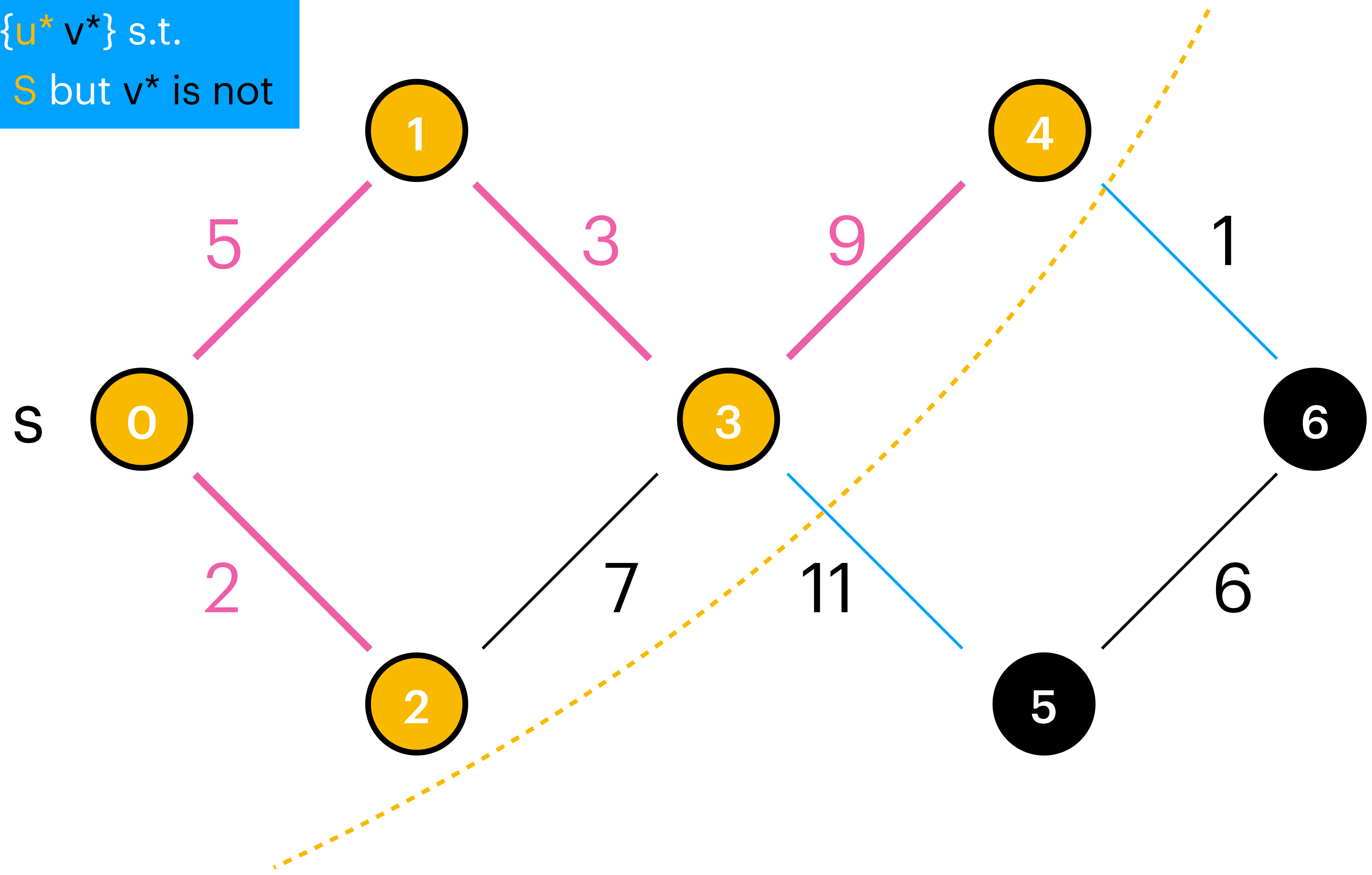
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not

F : { {0,2} , {0,1} , {1,3} , {3,9} }

S : { 0 , 2 , 1 , 3 , 4 }



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

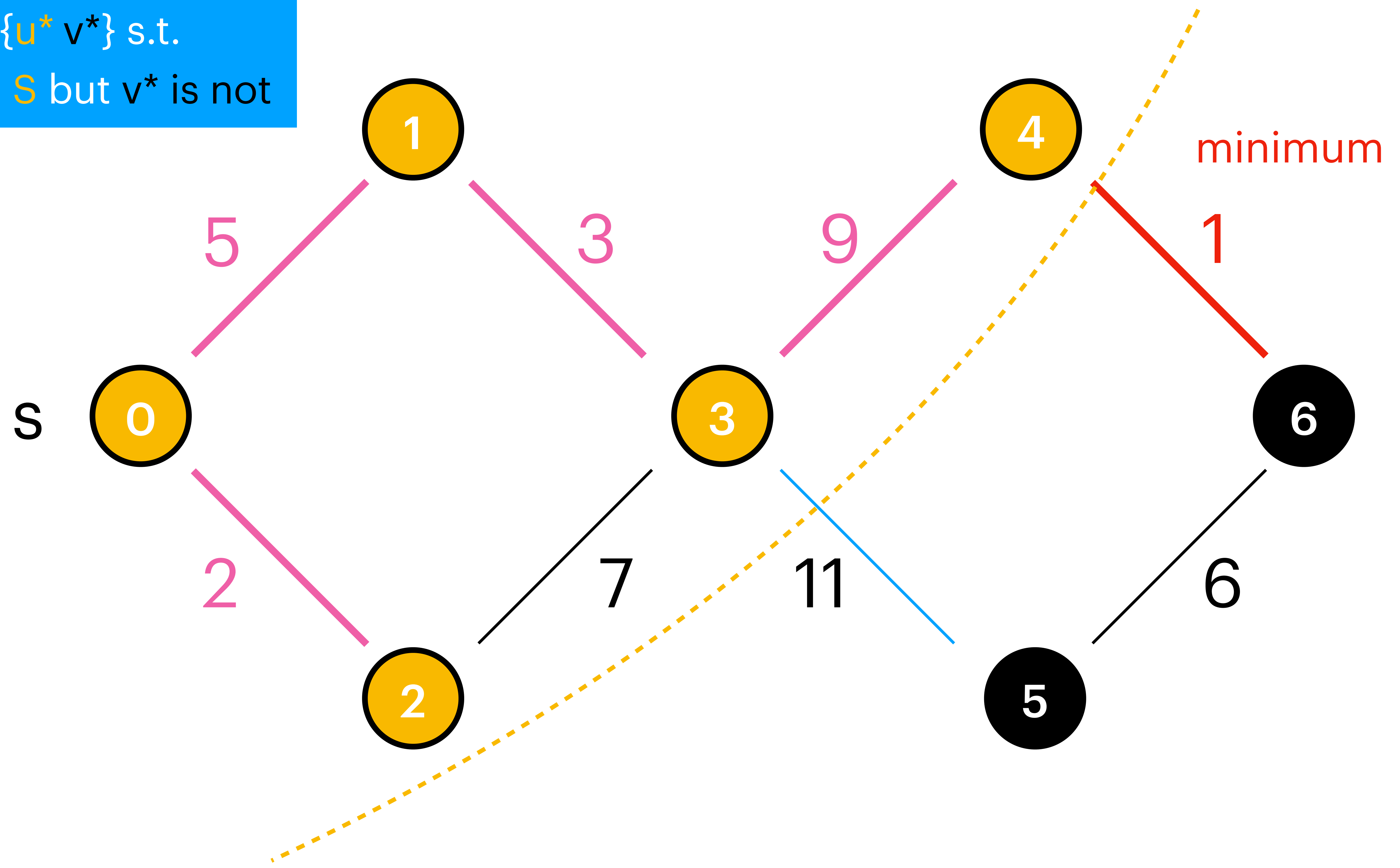
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not

F : { {0,2} , {0,1} , {1,3} , {3,9} }

S : { 0 , 2 , 1 , 3 , 4 }



MST

Prim's Algorithm

F : edges of the MST

S : connected component set

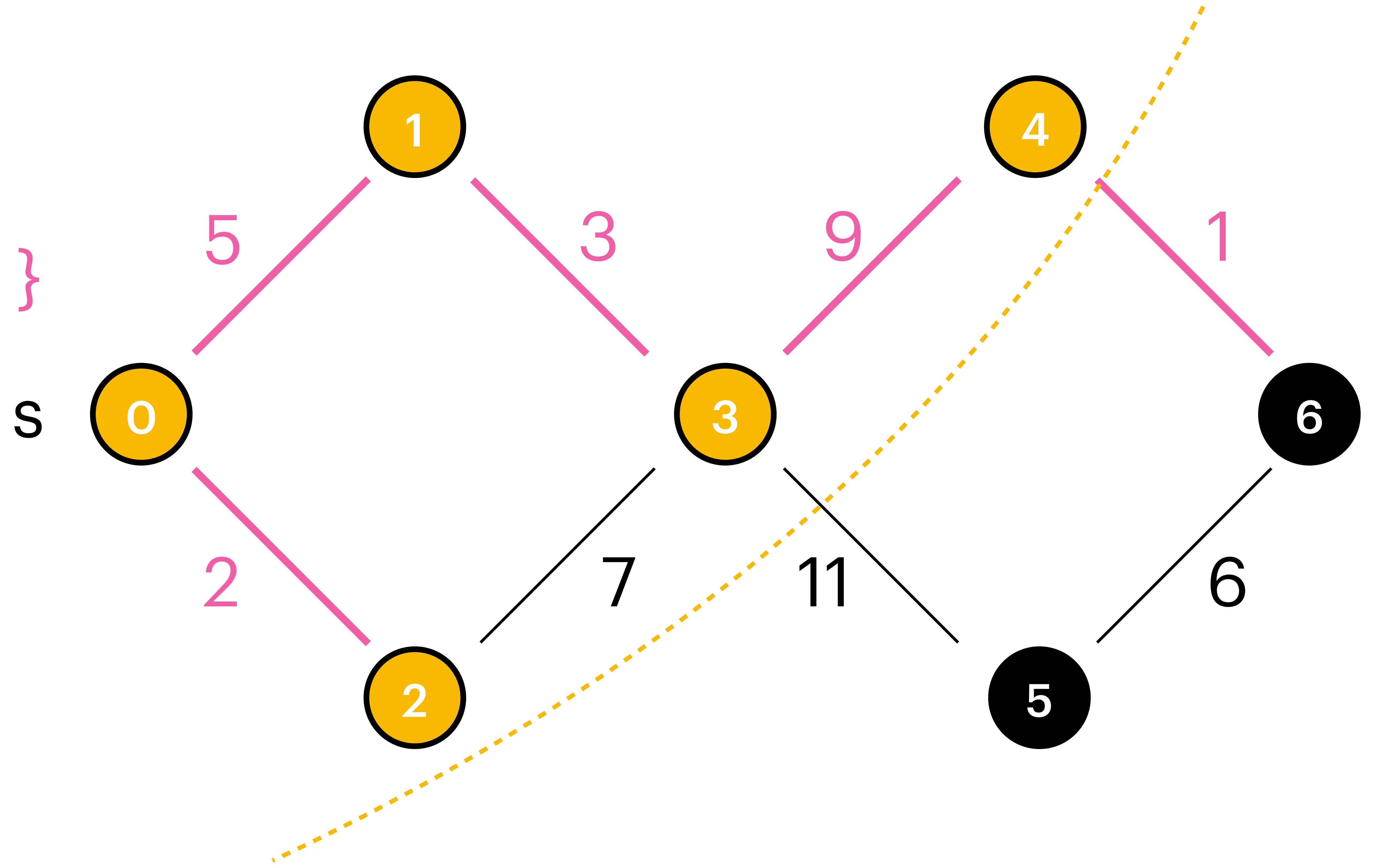
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} }

S : { 0 , 2 , 1 , 3 , 4 }



MST
Prim's Algorithm

F : edges of the MST

S : connected component set

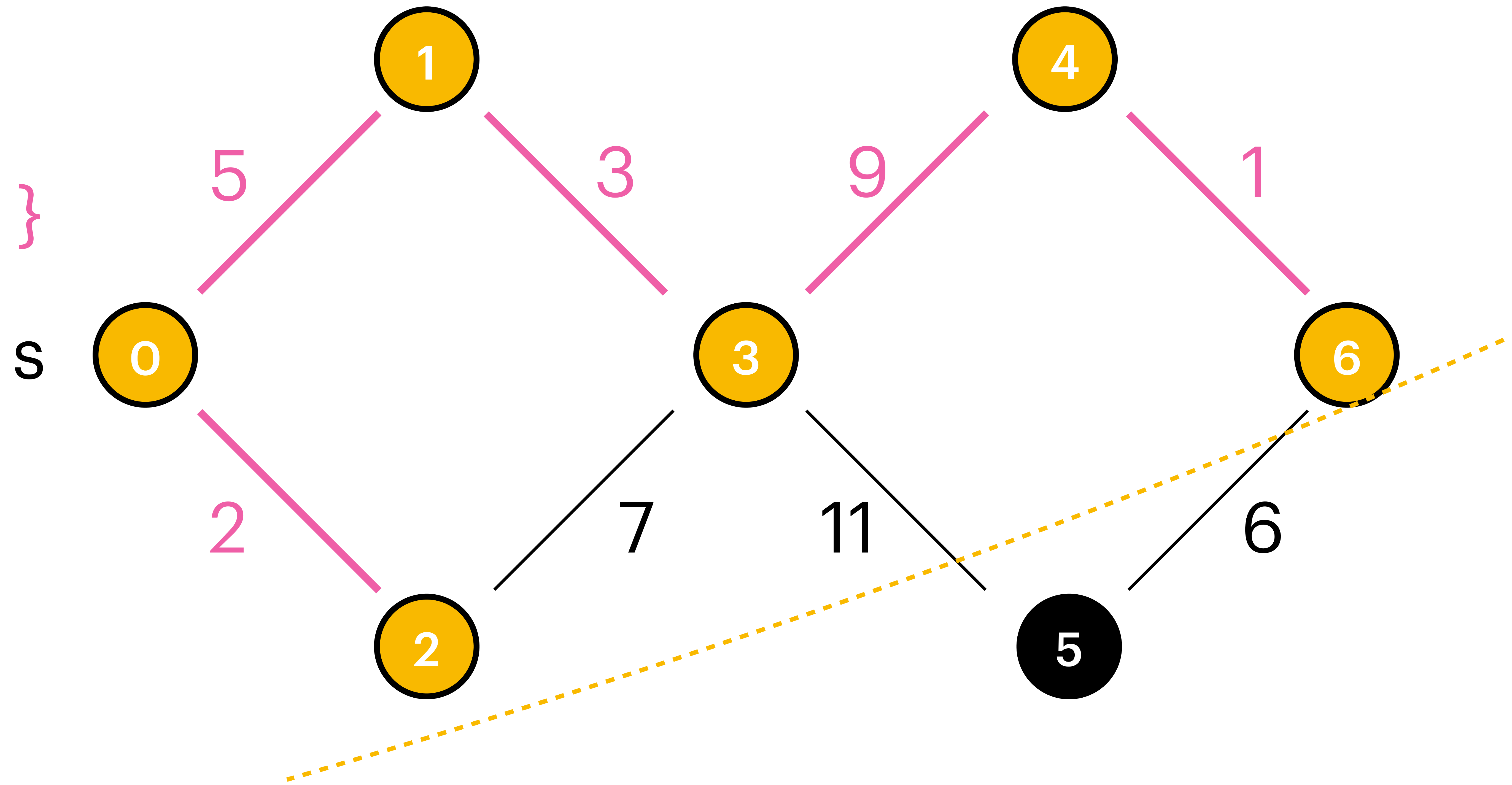
4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} }

S : { 0 , 2 , 1 , 3 , 4 , 6 }



MST

Prim's Algorithm

F : edges of the MST

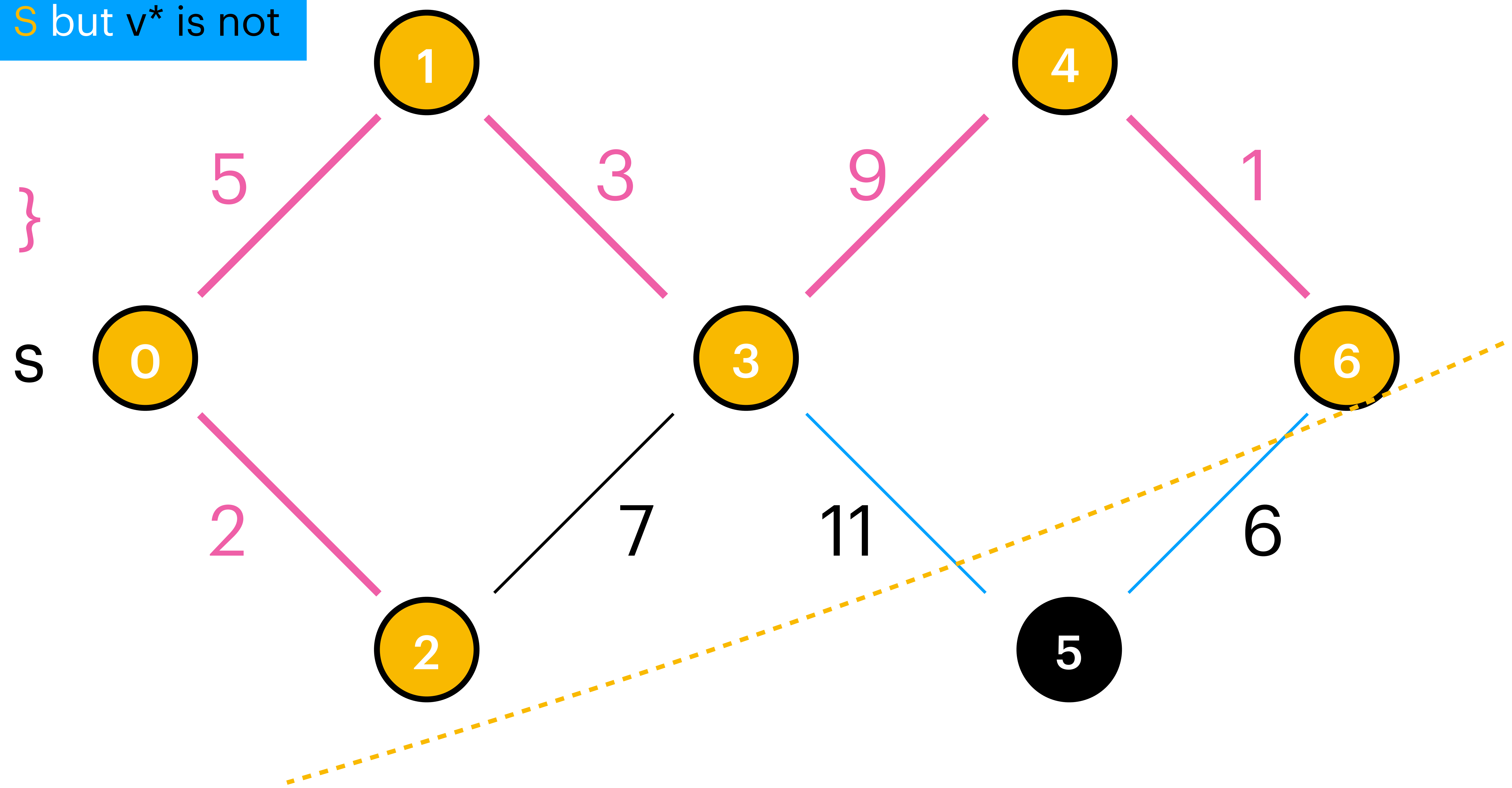
S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not



F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} }

S : { 0 , 2 , 1 , 3 , 4 , 6 }

MST
Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

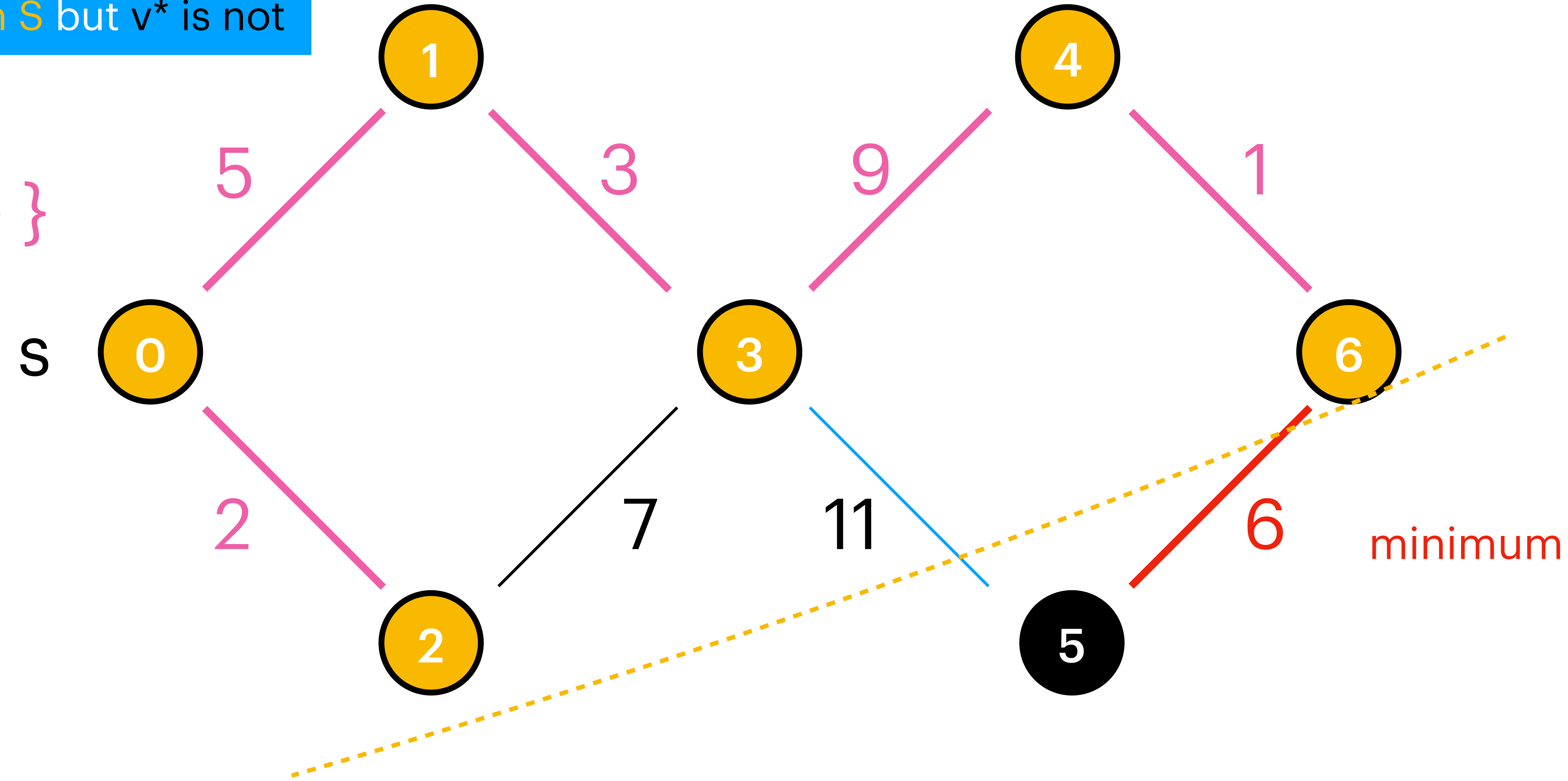
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} }

S : { 0 , 2 , 1 , 3 , 4 , 6 }



MST
Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

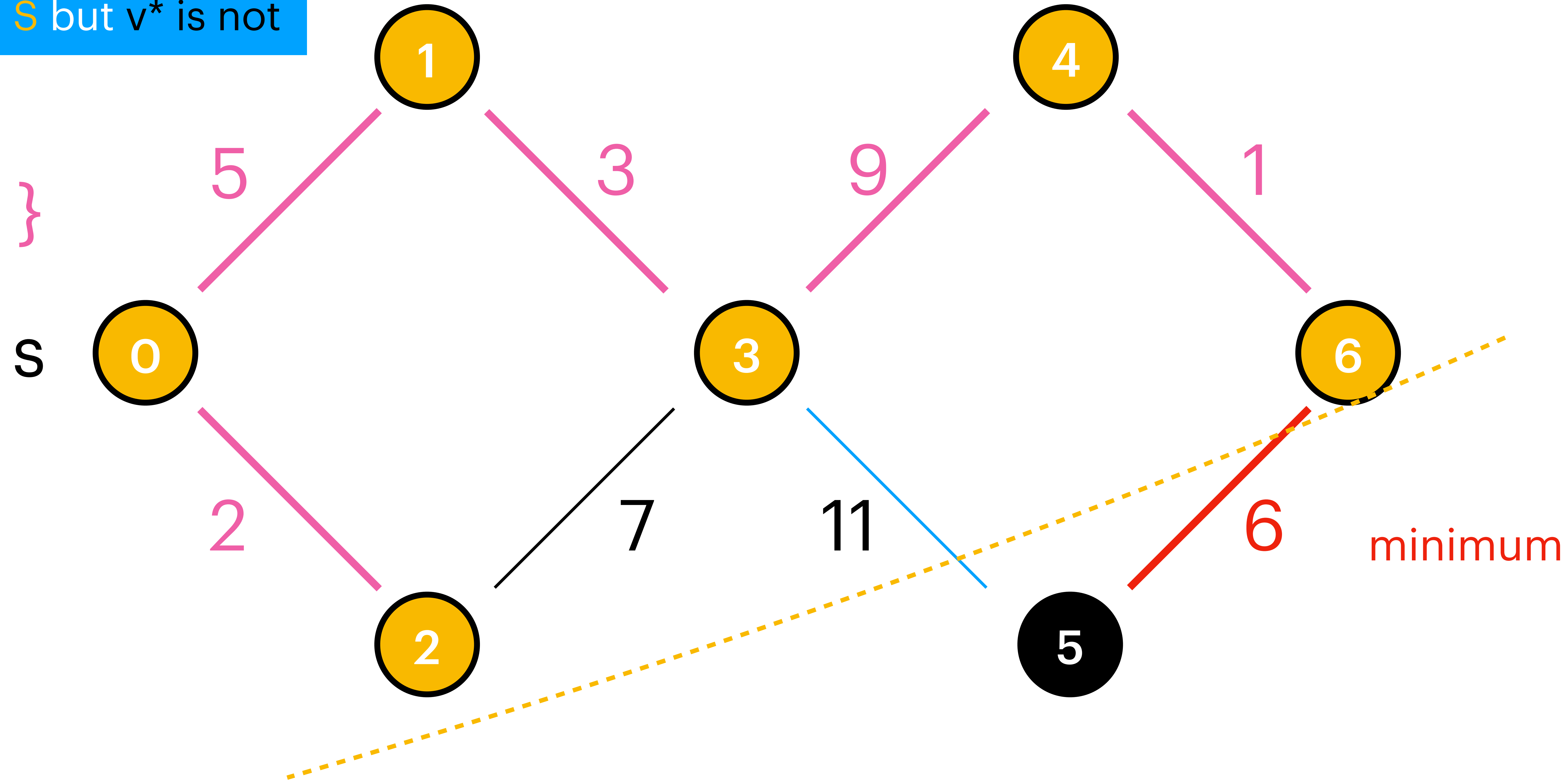
Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
5: $F \leftarrow F \cup \{u^*v^*\}$
6: $S \leftarrow S \cup \{v^*\}$

edges {u* v*} s.t.
u* is in S but v* is not

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} }

S : { 0 , 2 , 1 , 3 , 4 , 6 }



MST
Prim's Algorithm

F : edges of the MST

S : connected component set

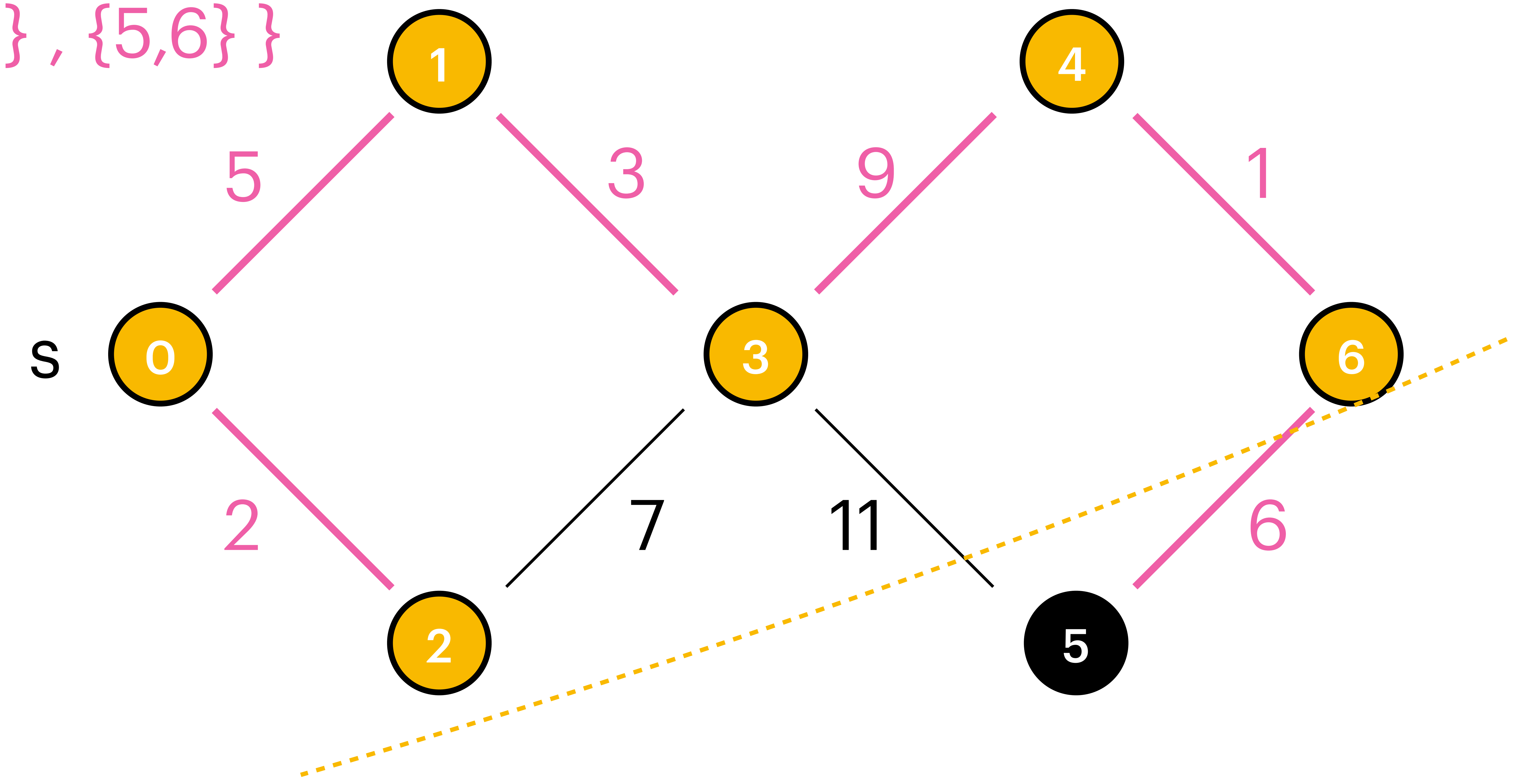
4 : find the minimum edge {u*,v*} s.t. u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} , {5,6} }

S : { 0 , 2 , 1 , 3 , 4 , 6 }



MST
Prim's Algorithm

F : edges of the MST

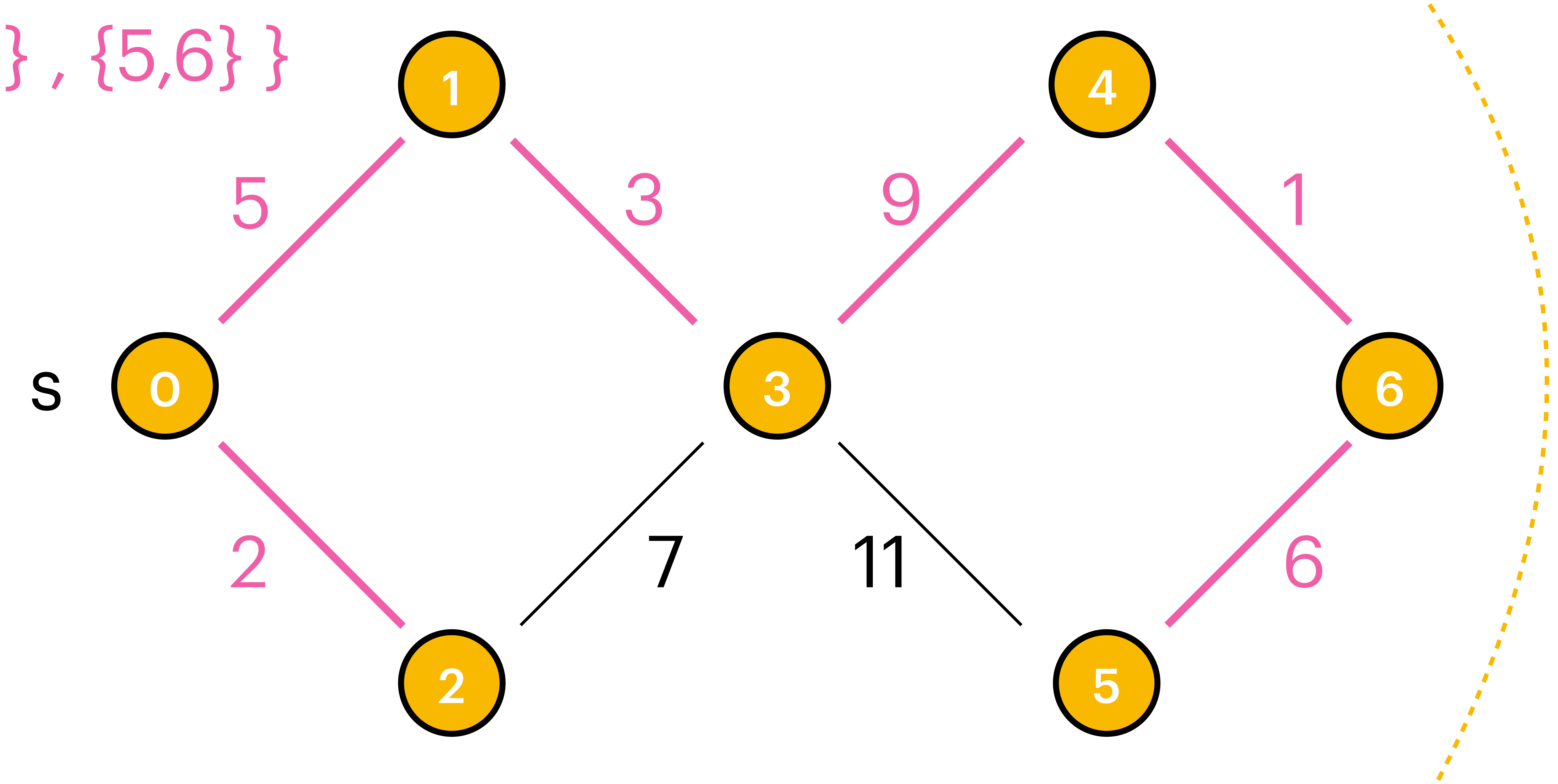
S : connected component set

4 : find the minimum edge {u*,v*} s.t. u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an S $\quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} , {5,6} }



S : { 0 , 2 , 1 , 3 , 4 , 6 , 5 }

MST

Prim's Algorithm

F : edges of the MST

S : connected component set

4 : find the minimum edge {u*,v*} s.t.
u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

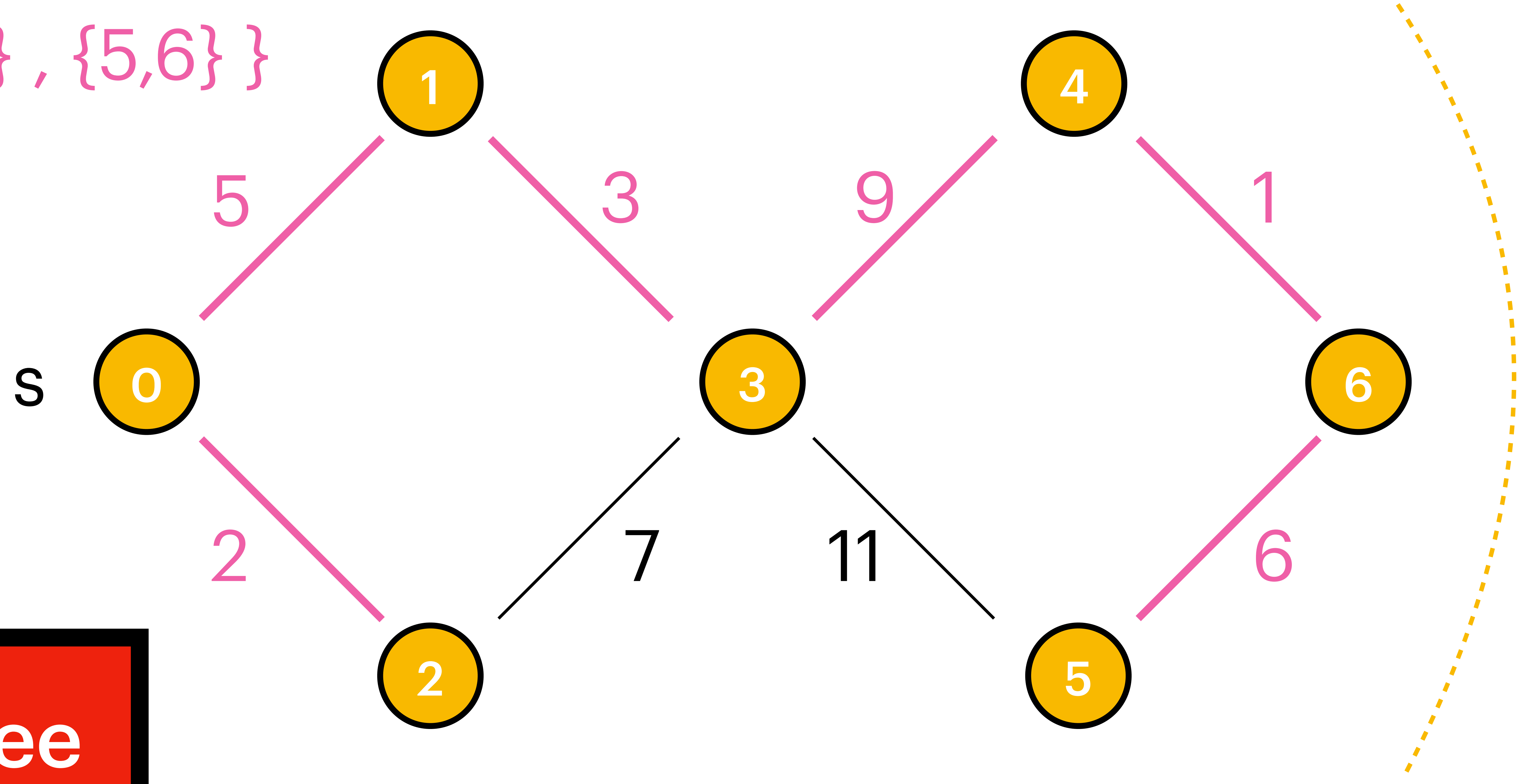
1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $\quad u^*v^* \leftarrow$ minimale Kante an $S \quad (u^* \in S, v^* \notin S)$
5: $\quad F \leftarrow F \cup \{u^*v^*\}$
6: $\quad S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} , {5,6} }

S : { 0 , 2 , 1 , 3 , 4 , 6 , 5 }

S = V

F is a spanning tree



MST

Prim's Algorithm

F : edges of the MST

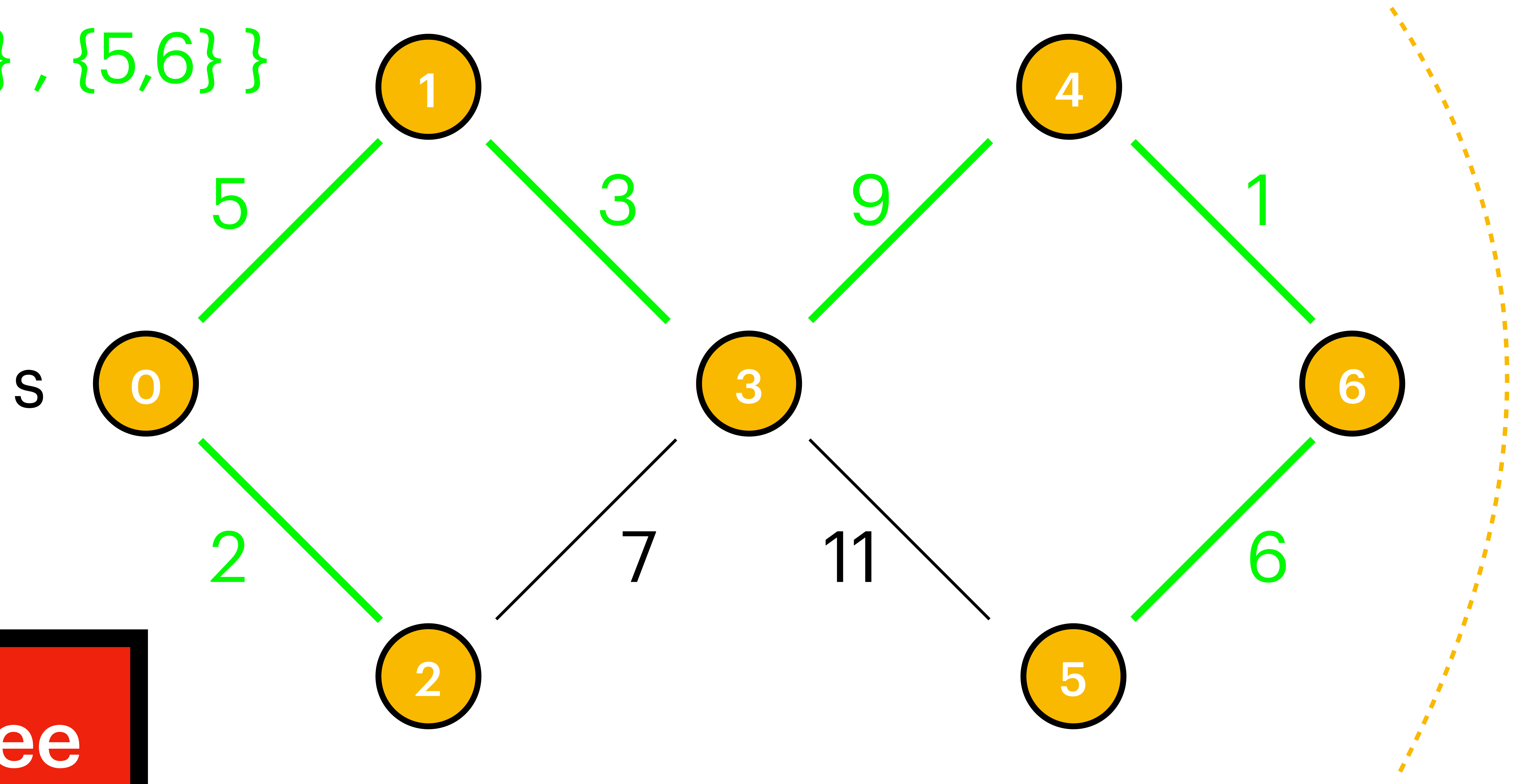
S : connected component set

4 : find the minimum edge {u*,v*} s.t. u* is in S but v* is not

Algorithm 9 Prim(G, s) (allgemeine Form)

1: $F \leftarrow \varnothing$
2: $S \leftarrow \{s\}$
3: **while** F nicht Spannbaum **do**
4: $u^*v^* \leftarrow$ minimale Kante an S $(u^* \in S, v^* \notin S)$
5: $F \leftarrow F \cup \{u^*v^*\}$
6: $S \leftarrow S \cup \{v^*\}$

F : { {0,2} , {0,1} , {1,3} , {3,9} , {4,6} , {5,6} }



S : { 0 , 2 , 1 , 3 , 4 , 6 , 5 }

S = V

F is a spanning tree

MST

Boruvka's Algorithm

Runtime : O ((|V| + |E|) * log n)

Algorithm 8 Boruvka(G)

$F \leftarrow \emptyset$
while F nicht Spannbaum **do**
    $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
    $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
    $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

connected component of F

minimum edges near Ss

MST

Boruvka's Algorithm

Runtime : O ((|V| + |E|) * log n)

Algorithm 8 Boruvka(G)

$F \leftarrow \varnothing$
while F nicht Spannbaum **do**
    $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
    $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
    $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

find with DFS only using the edges in F !

connected components with edges from F

minimum edges near Ss

F : edges of the MST

MST

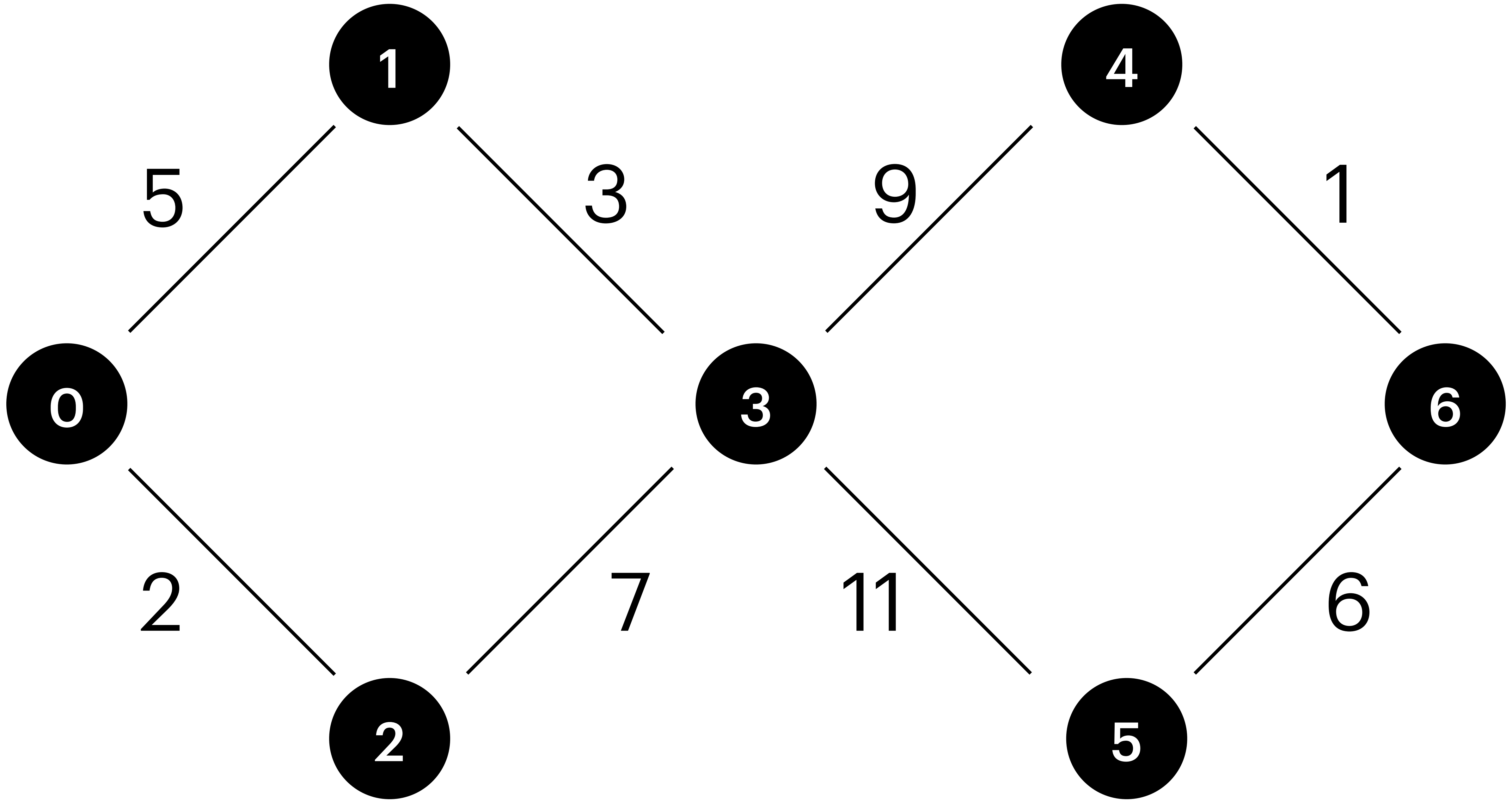
Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

$F \leftarrow \varnothing$
while F nicht Spannbaum **do**
    $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
    $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
    $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F :



MST

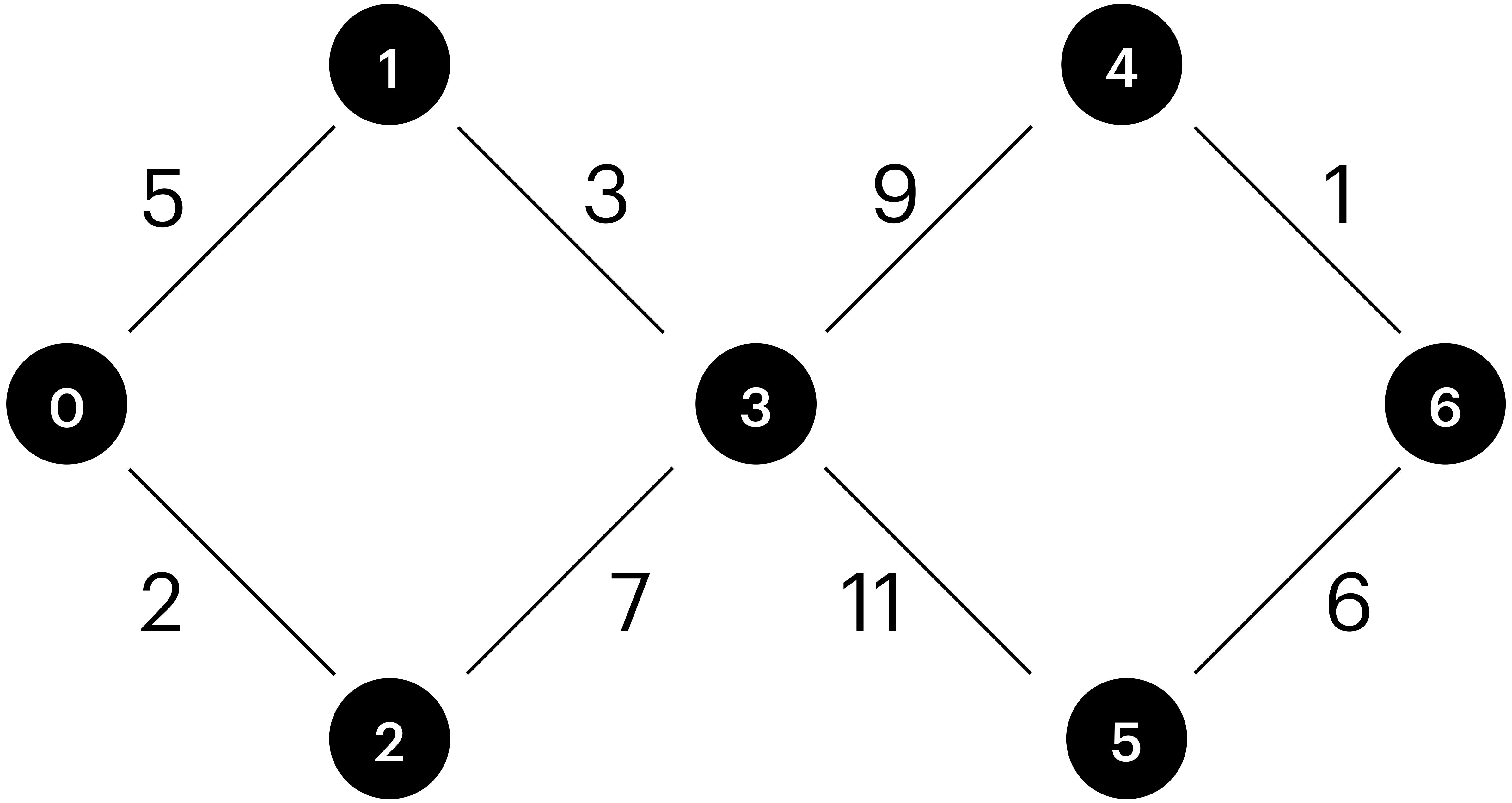
Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : $\varnothing$



MST

Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : $\varnothing$

$S_1 = \{0\}$

$S_2 = \{1\}$

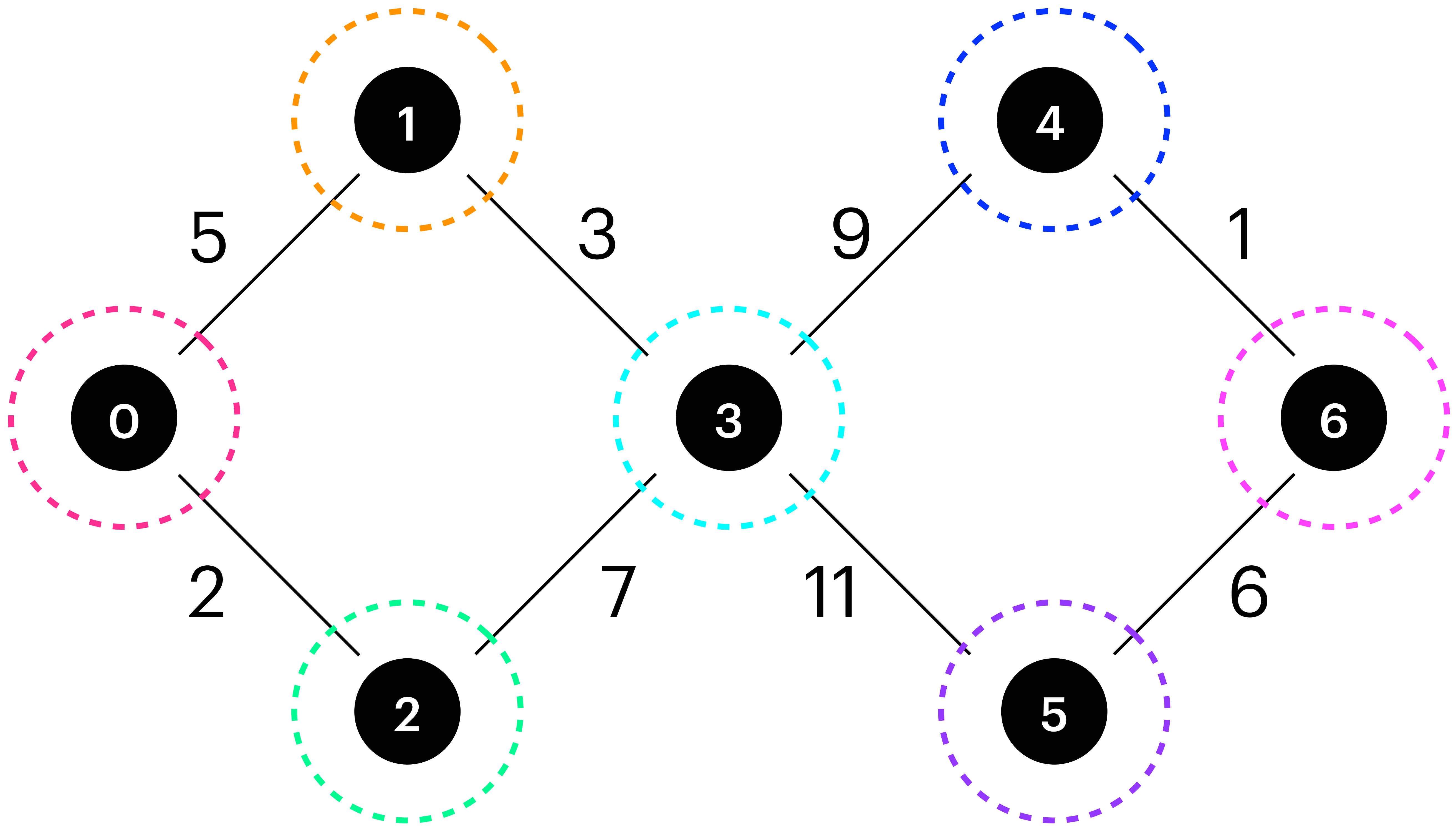
$S_3 = \{2\}$

$S_4 = \{3\}$

$S_5 = \{4\}$

$S_6 = \{5\}$

$S_7 = \{6\}$



MST

Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : $\varnothing$

$S_1 = \{0\}$

$S_2 = \{1\}$

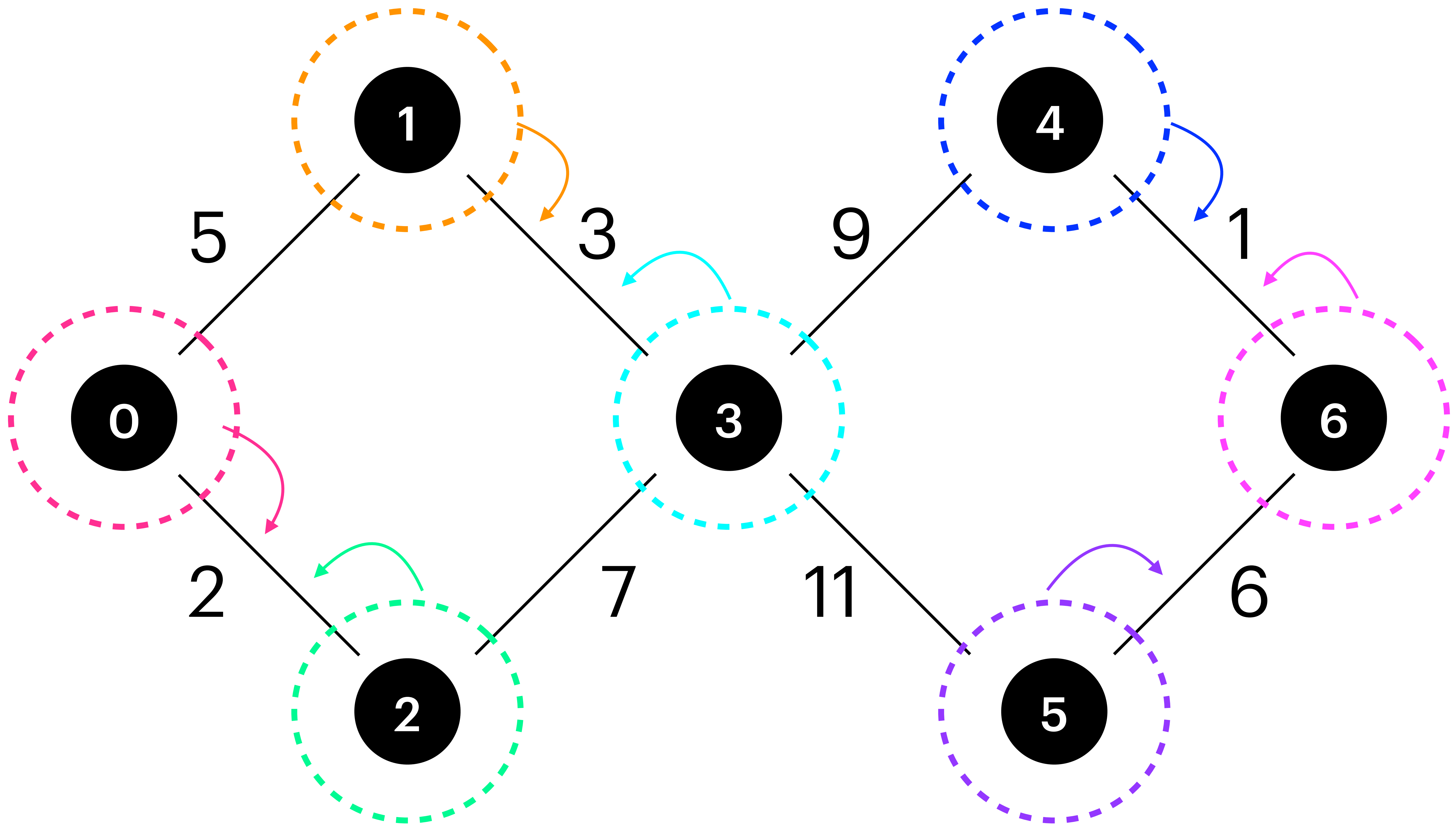
$S_3 = \{2\}$

$S_4 = \{3\}$

$S_5 = \{4\}$

$S_6 = \{5\}$

$S_7 = \{6\}$



MST
Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : ∅

$S_1 = \{0\}$

$S_2 = \{1\}$

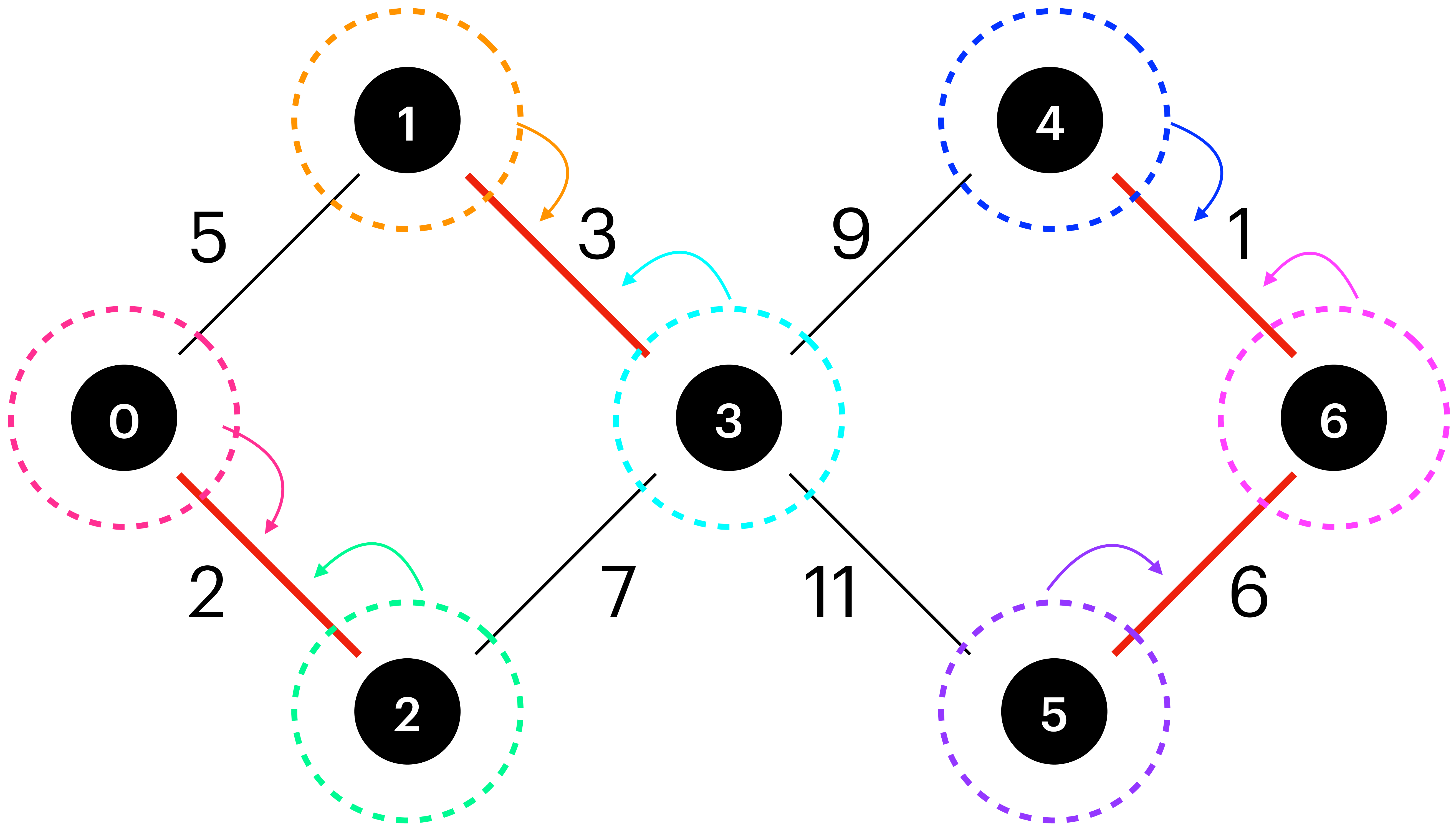
$S_3 = \{2\}$

$S_4 = \{3\}$

$S_5 = \{4\}$

$S_6 = \{5\}$

$S_7 = \{6\}$



MST

Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : { {0,2} , {1,3} , {4,6} , {5,6} }

$S_1 = \{0\}$

$S_2 = \{1\}$

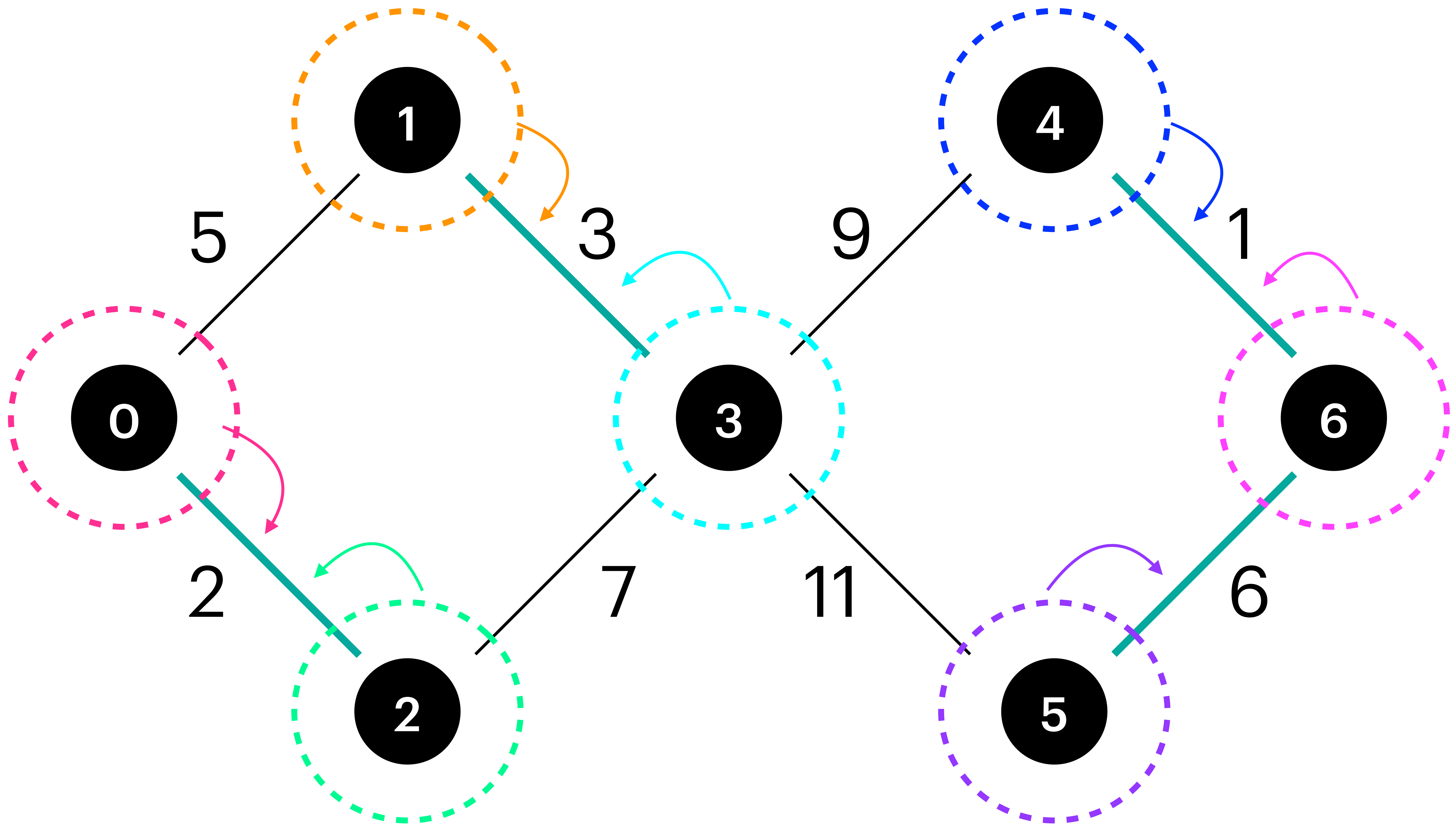
$S_3 = \{2\}$

$S_4 = \{3\}$

$S_5 = \{4\}$

$S_6 = \{5\}$

$S_7 = \{6\}$



MST

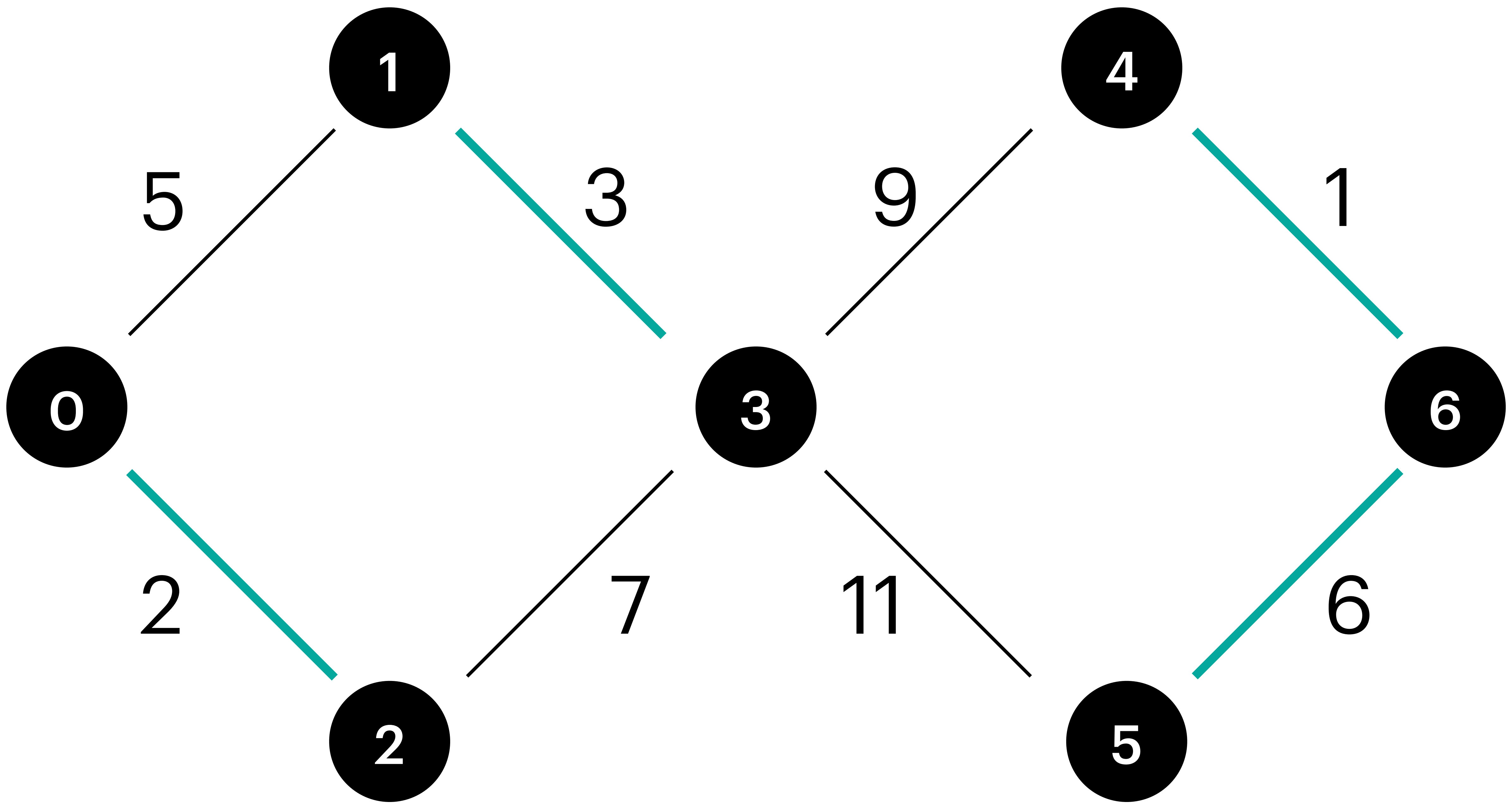
Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : { {0,2} , {1,3} , {4,6} , {5,6} }



MST

Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

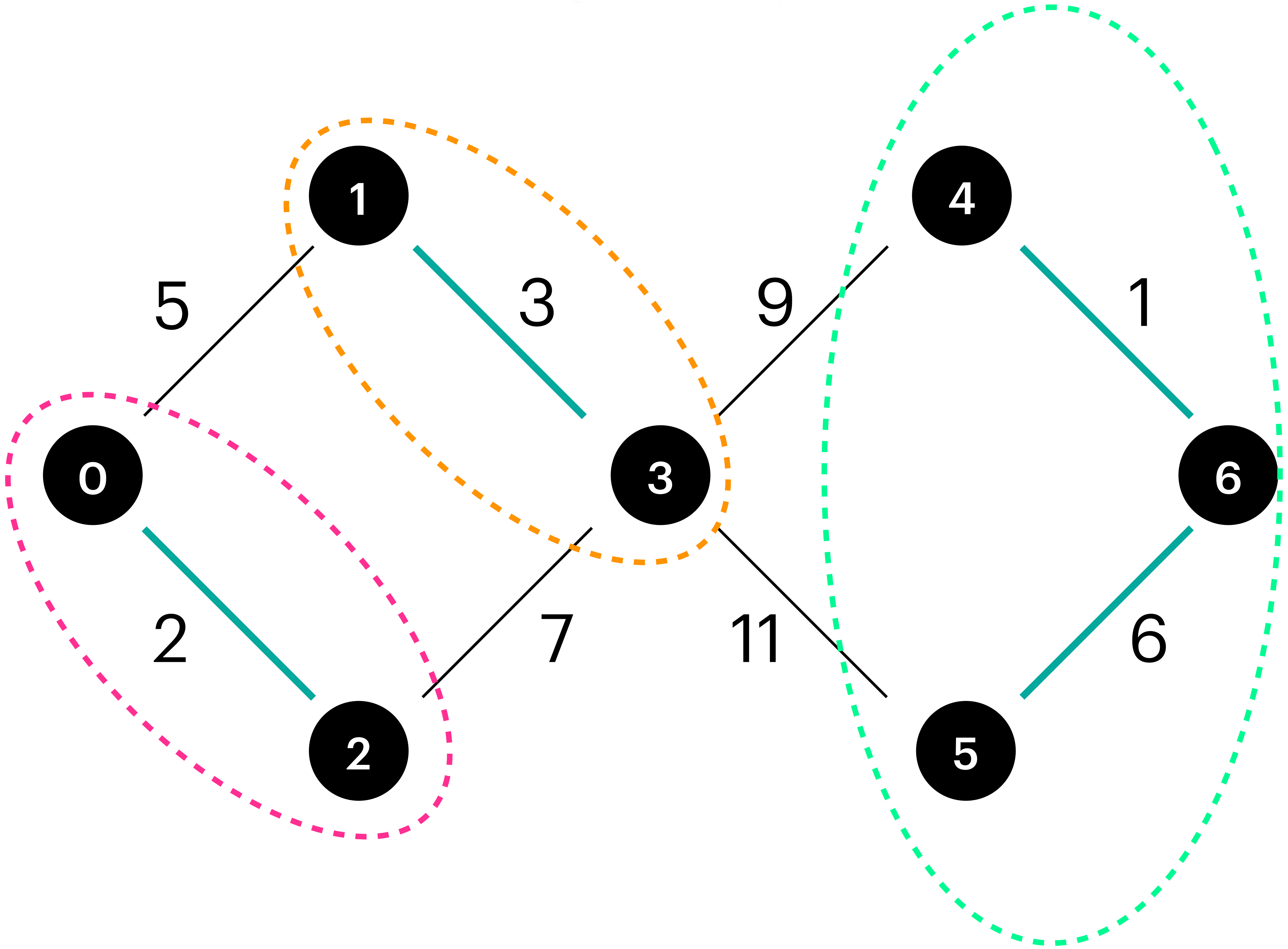
1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : { {0,2} , {1,3} , {4,6} , {5,6} }

$S_1 = \{0,2\}$

$S_2 = \{1,3\}$

$S_3 = \{4,5,6\}$



MST
Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

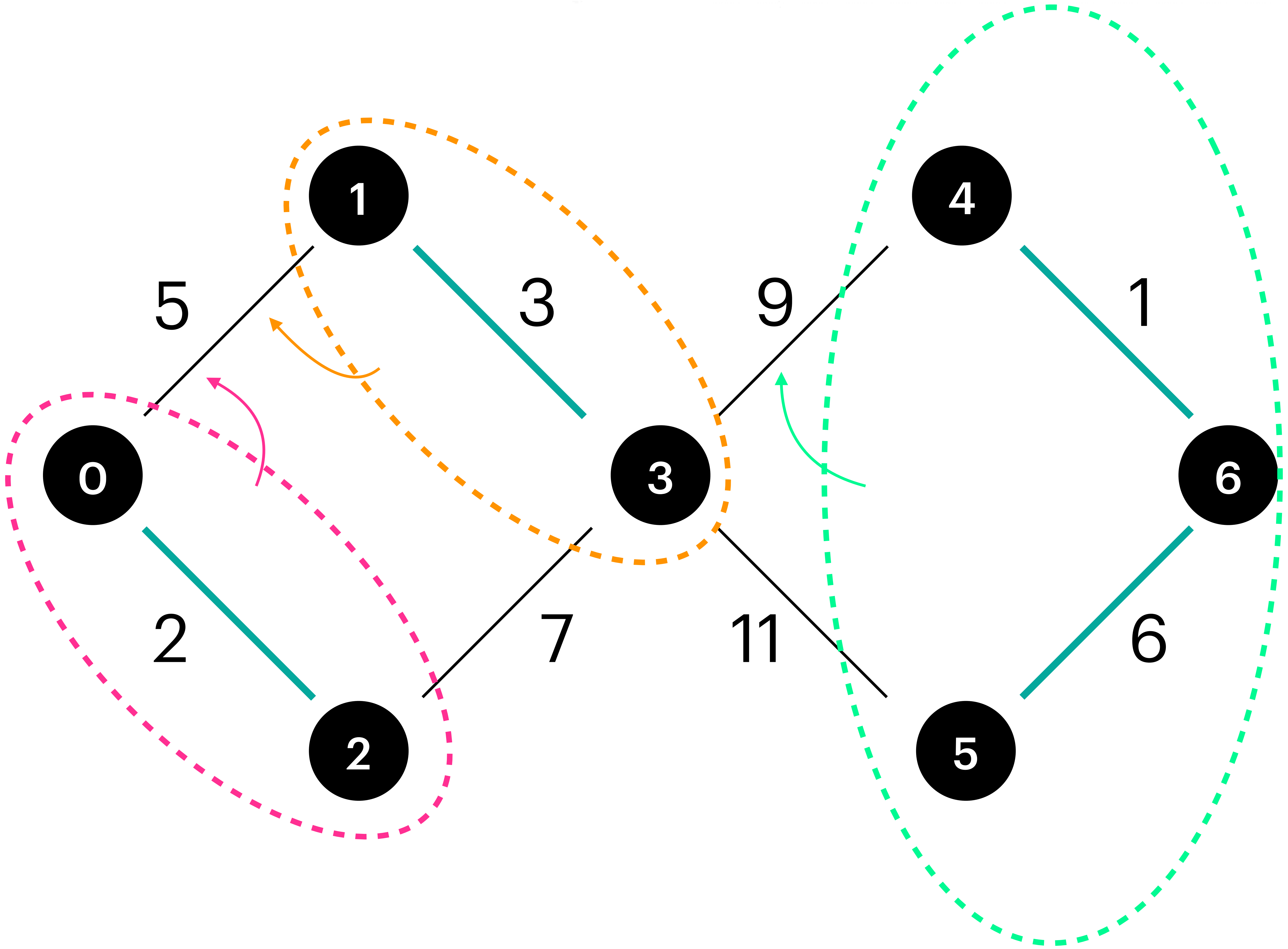
1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : { {0,2} , {1,3} , {4,6} , {5,6} }

S_1 = {0,2}

S_2 = {1,3}

S_3 = {4,5,6}



MST

Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

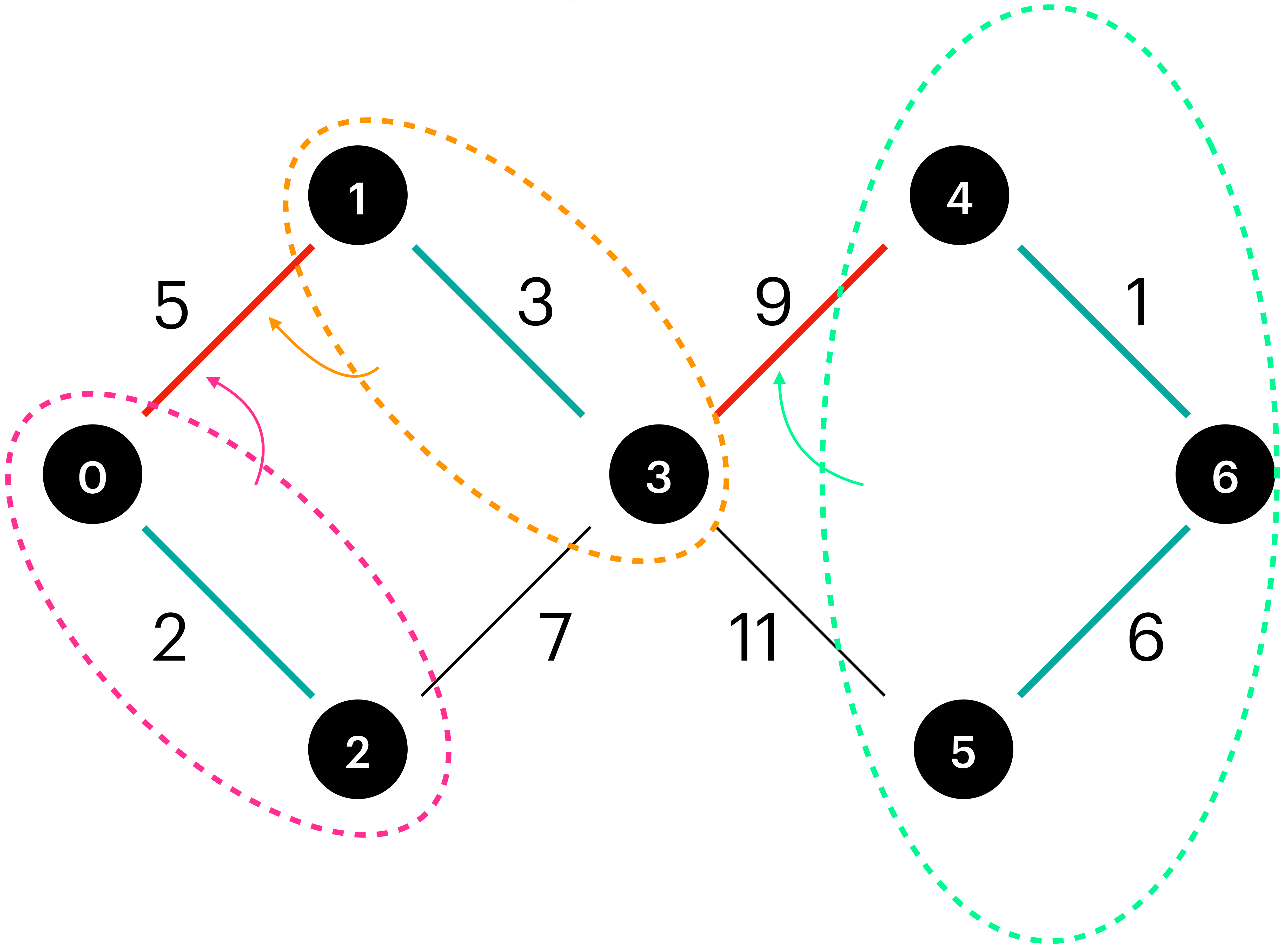
```
F ← ∅
while F nicht Spannbaum do
    (S_1, ..., S_k) ← ZHKs von F
    (e_1, ..., e_k) ← minimale Kanten an S_1, ..., S_k
    F ← F ∪ {e_1, ..., e_k}
```

F : { {0,2} , {1,3} , {4,6} , {5,6} }

$S_1 = \{0,2\}$

$S_2 = \{1,3\}$

$S_3 = \{4,5,6\}$



MST

Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

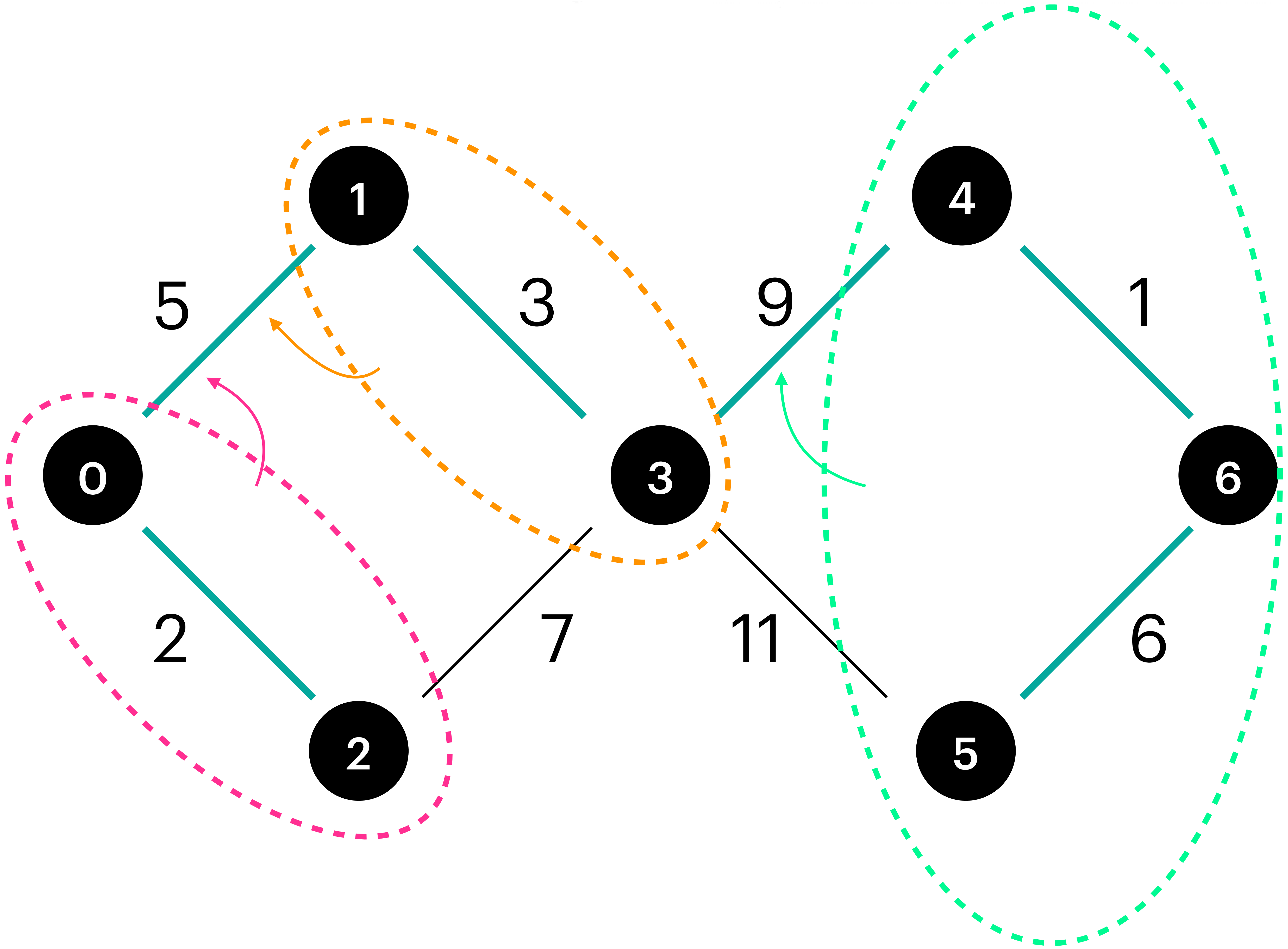
1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : { {0,2} , {1,3} , {4,6} , {5,6} ,
{0,1} , {4,3} }

$S_1 = \{0,2\}$

$S_2 = \{1,3\}$

$S_3 = \{4,5,6\}$



MST

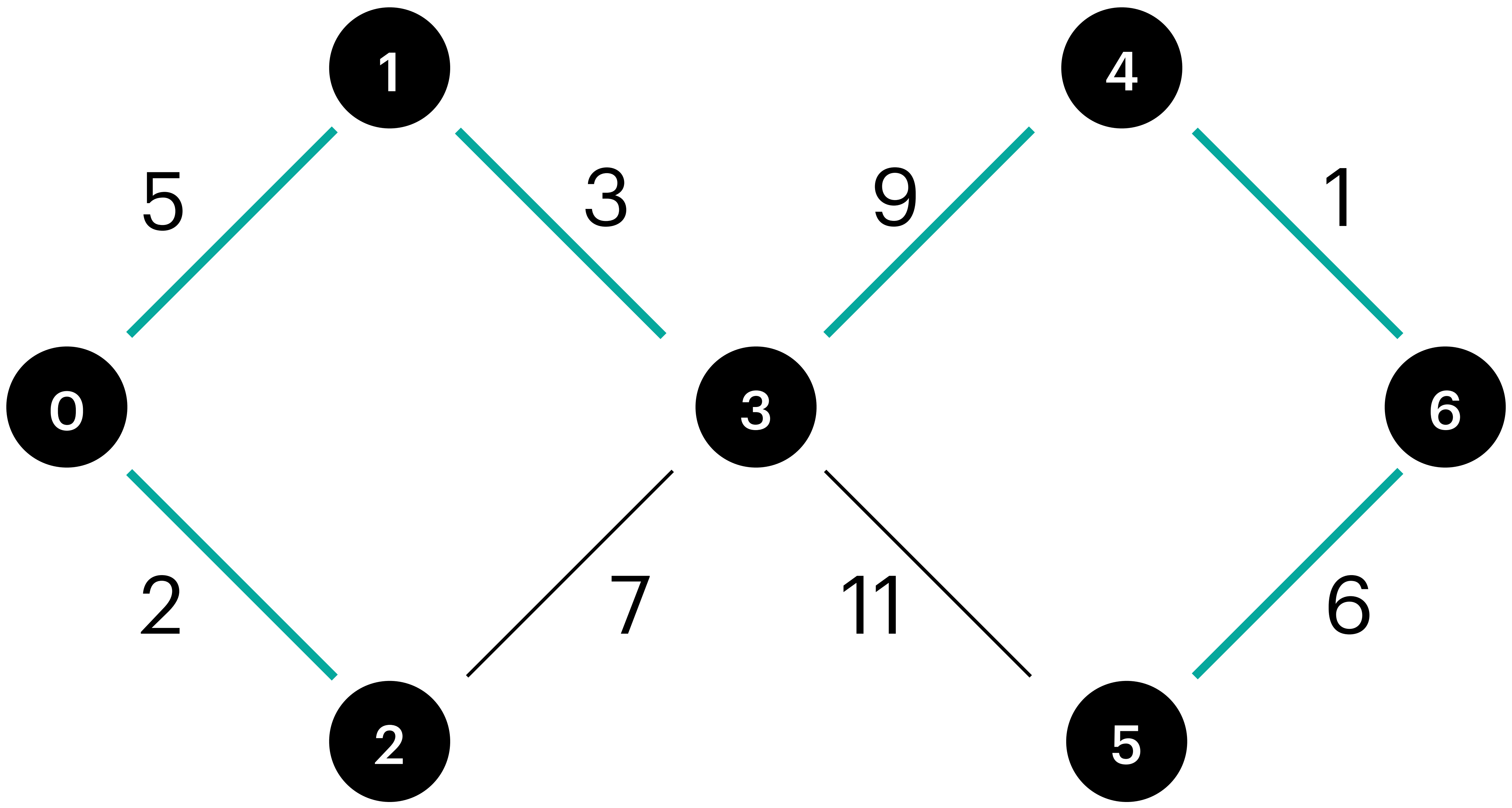
Boruvka's Algorithm

F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : { {0,2} , {1,3} , {4,6} , {5,6} ,
{0,1} , {4,3} }



MST

Boruvka's Algorithm

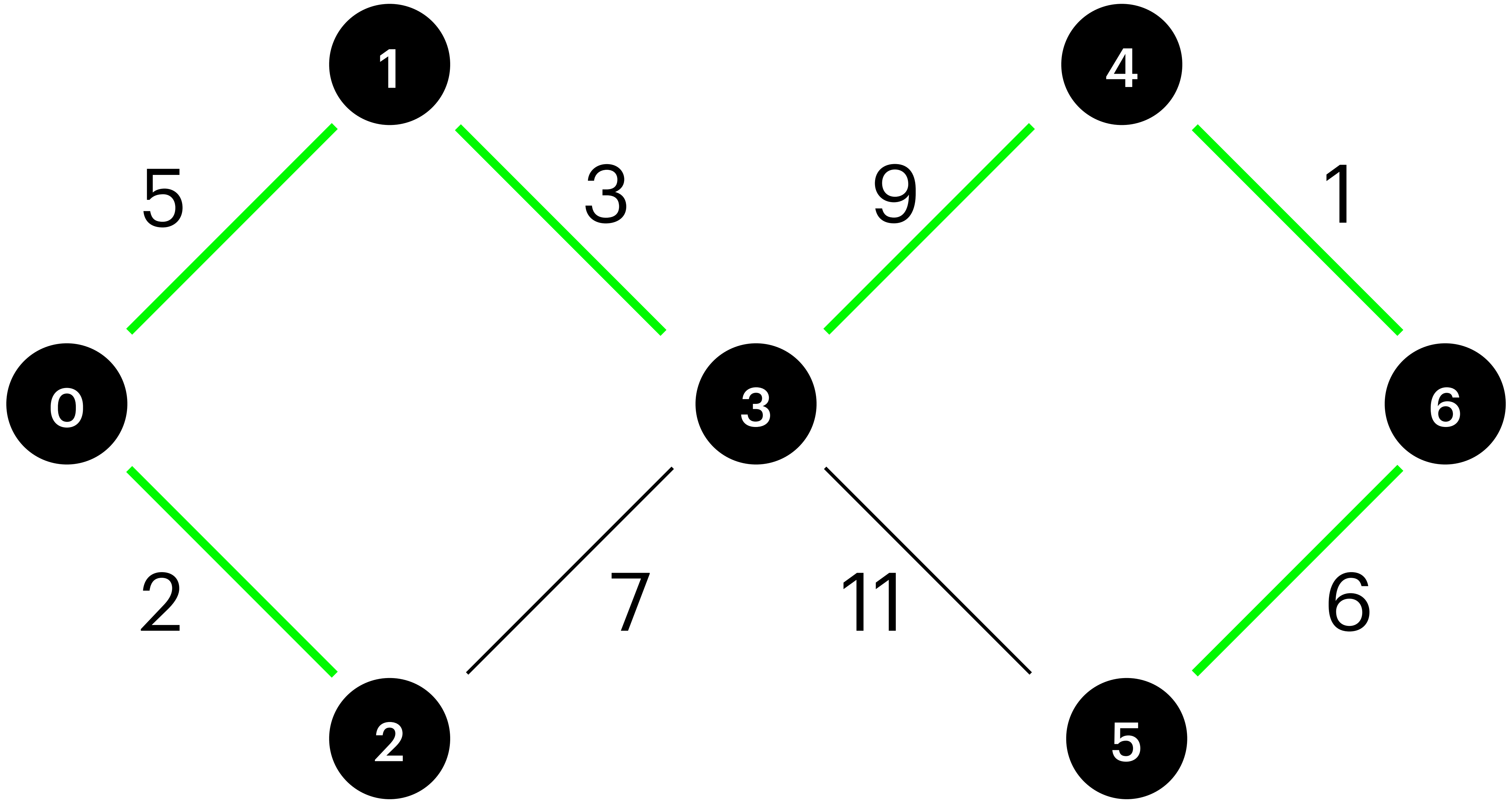
F : edges of the MST

Algorithm 8 Boruvka(G)

1: $F \leftarrow \varnothing$
2: **while** F nicht Spannbaum **do**
3: $(S_1, \ldots, S_k) \leftarrow$ ZHKs von F
4: $(e_1, \ldots, e_k) \leftarrow$ minimale Kanten an $S_1, \ldots, S_k$
5: $F \leftarrow F \cup \{e_1, \ldots, e_k\}$

F : { {0,2} , {1,3} , {4,6} , {5,6} , {0,1} , {4,3} }

F is a spanning tree



MST

Kruskal's Algorithm

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V     members[v] <- {v}          ▷ O(n)

SAME(u,v):   teste ob rep[u] = rep[v]                          ▷ O(1)

UNION(u,v):                                                    ▷ O(|ZHK(u)|)
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

MST

Kruskal's Algorithm

Runtime : O ((|V| + |E|) * log n)

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \emptyset$ edges of the MST
$UF \leftarrow$ MAKE(V) UF : Union Find
SORT(E) sort the edges in increasing order
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then** if its not in the same concomp
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

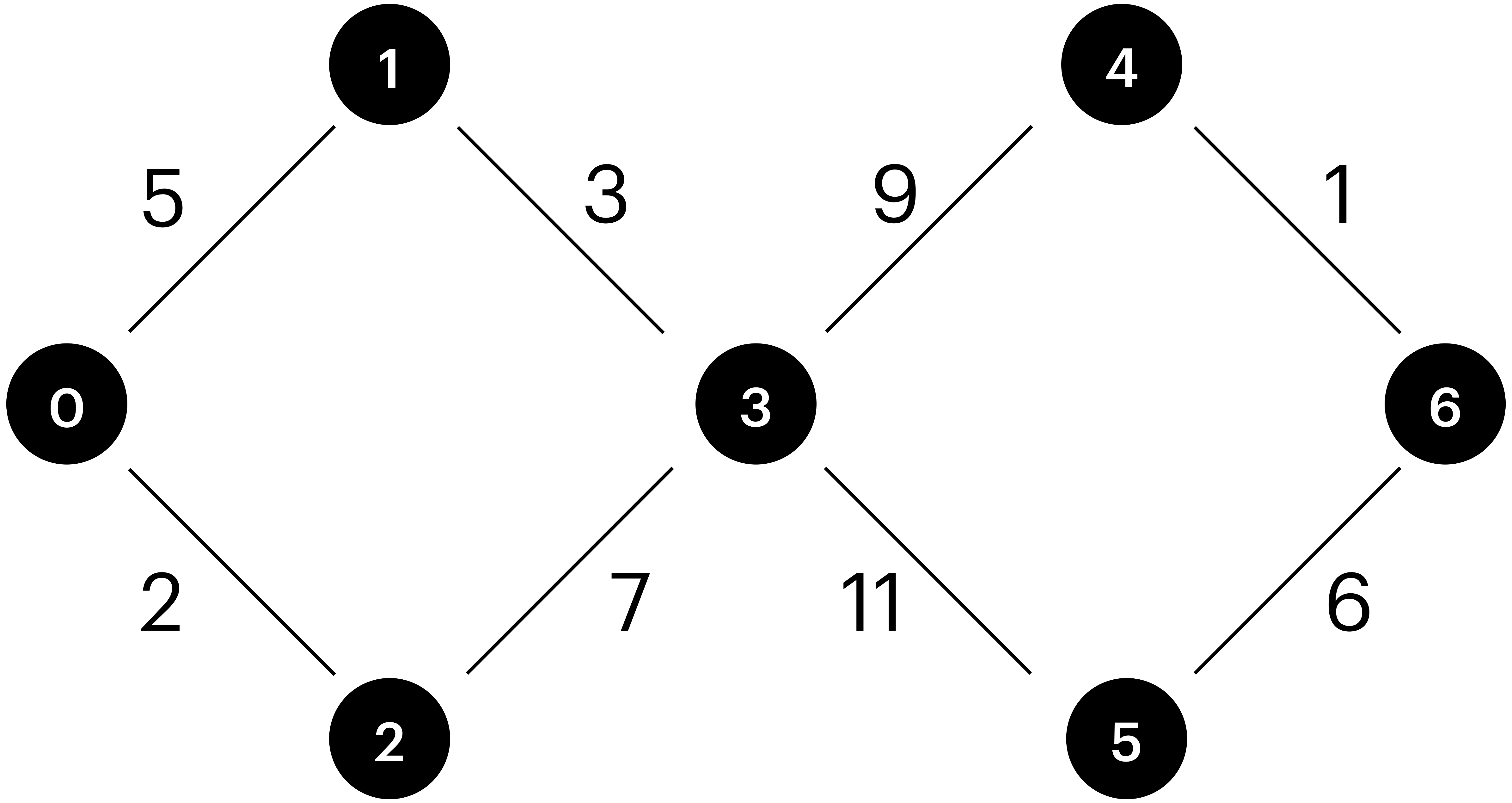
F :

rep[] :

0	1	2	3	4	5	6

members[] :

0	
1	
2	
3	
4	
5	
6	



MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

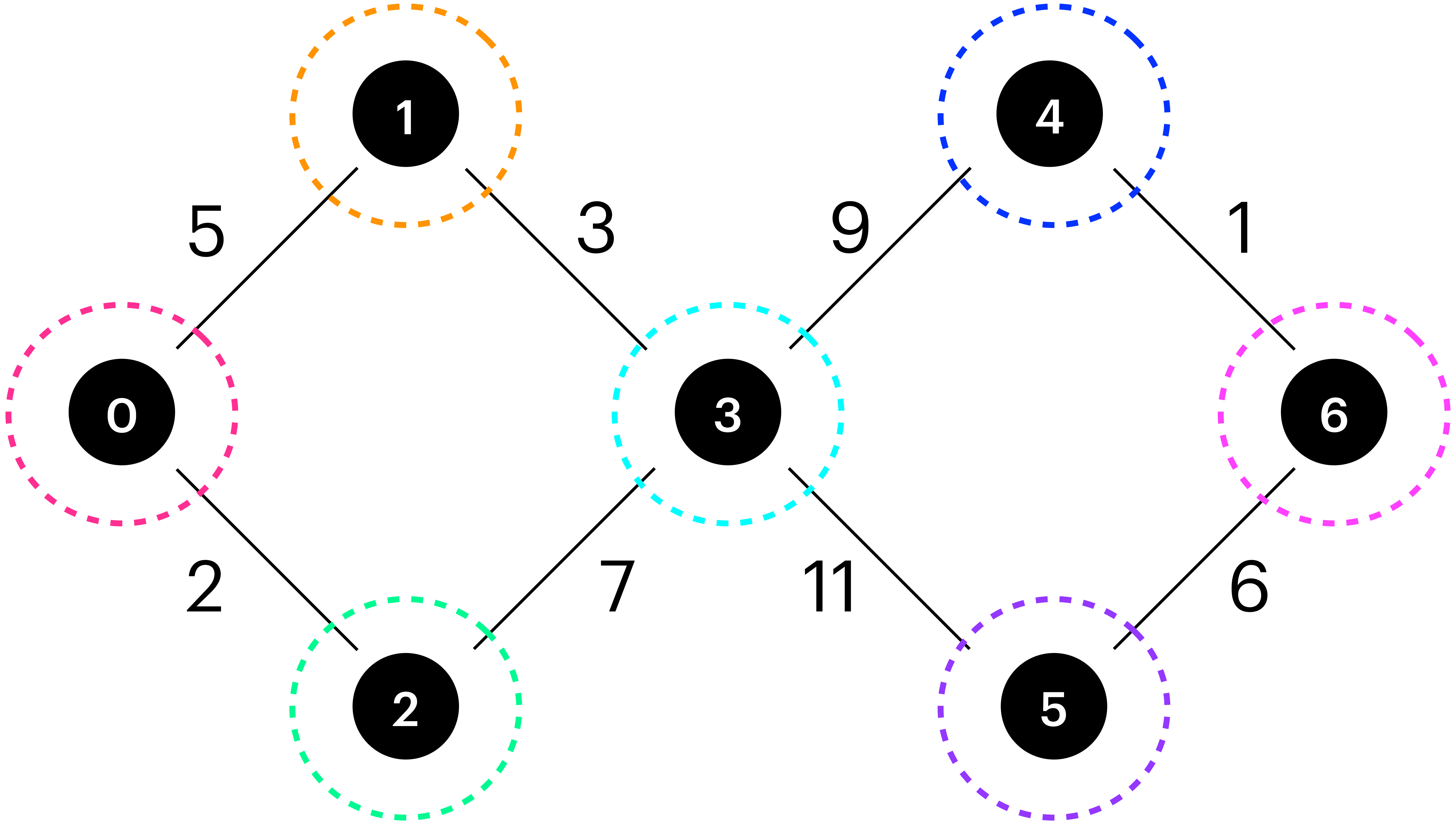
F : $\varnothing$

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	6

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):   teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

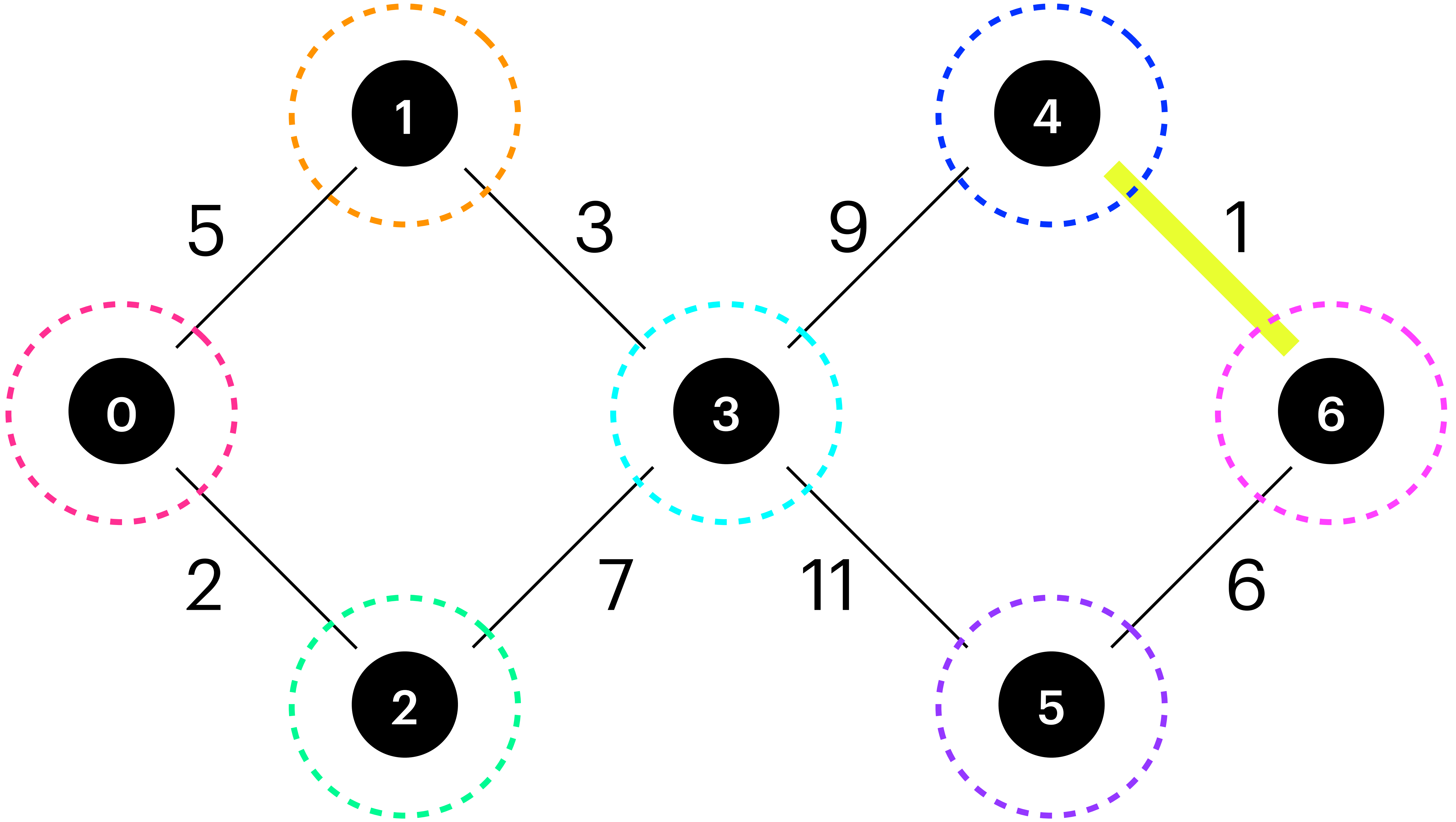
F : ∅

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	6

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \ \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

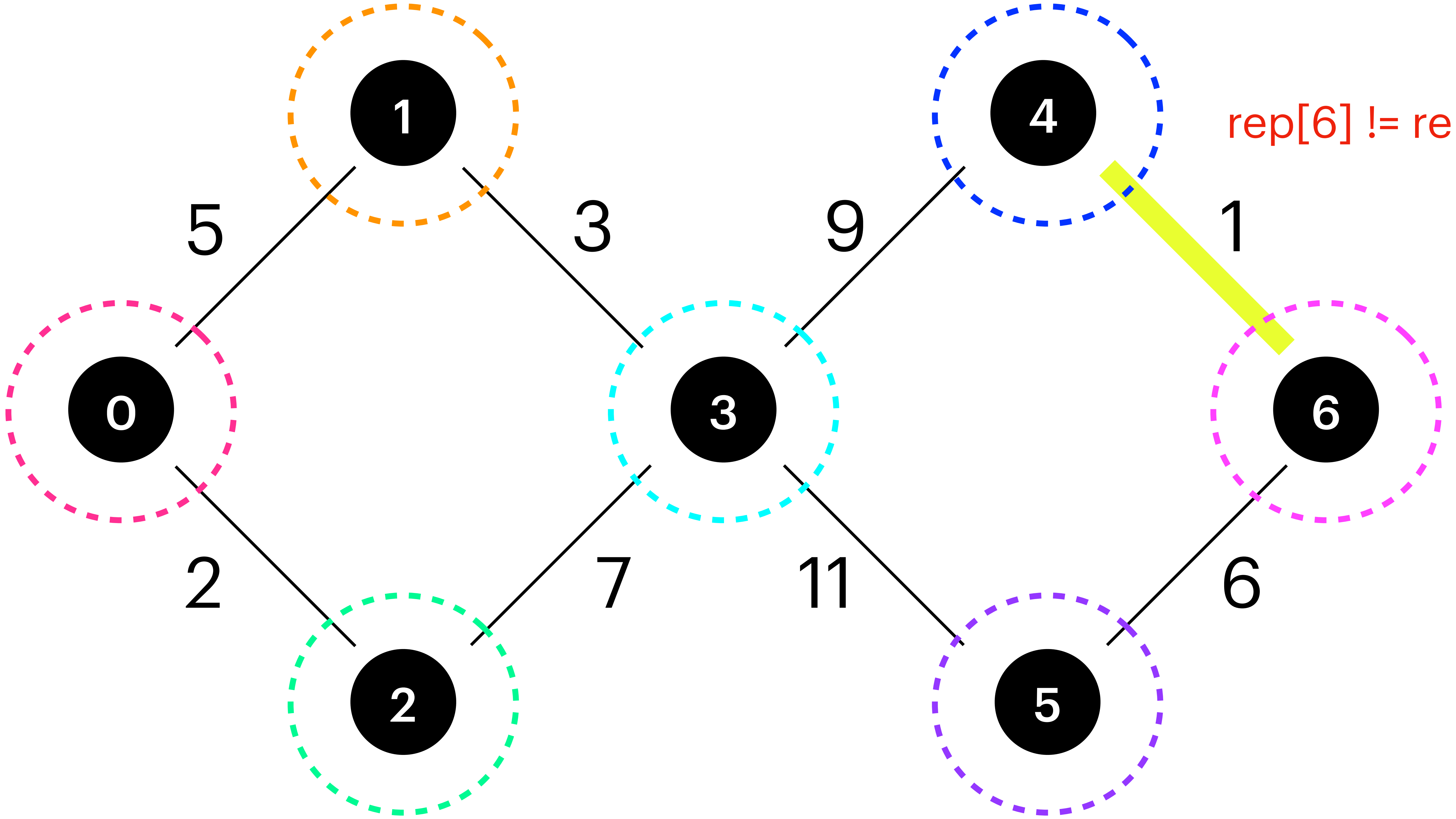
F : $\varnothing$

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	6

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 }
5	{ 5 }
6	{ 6 }



rep[6] != rep[4]

SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

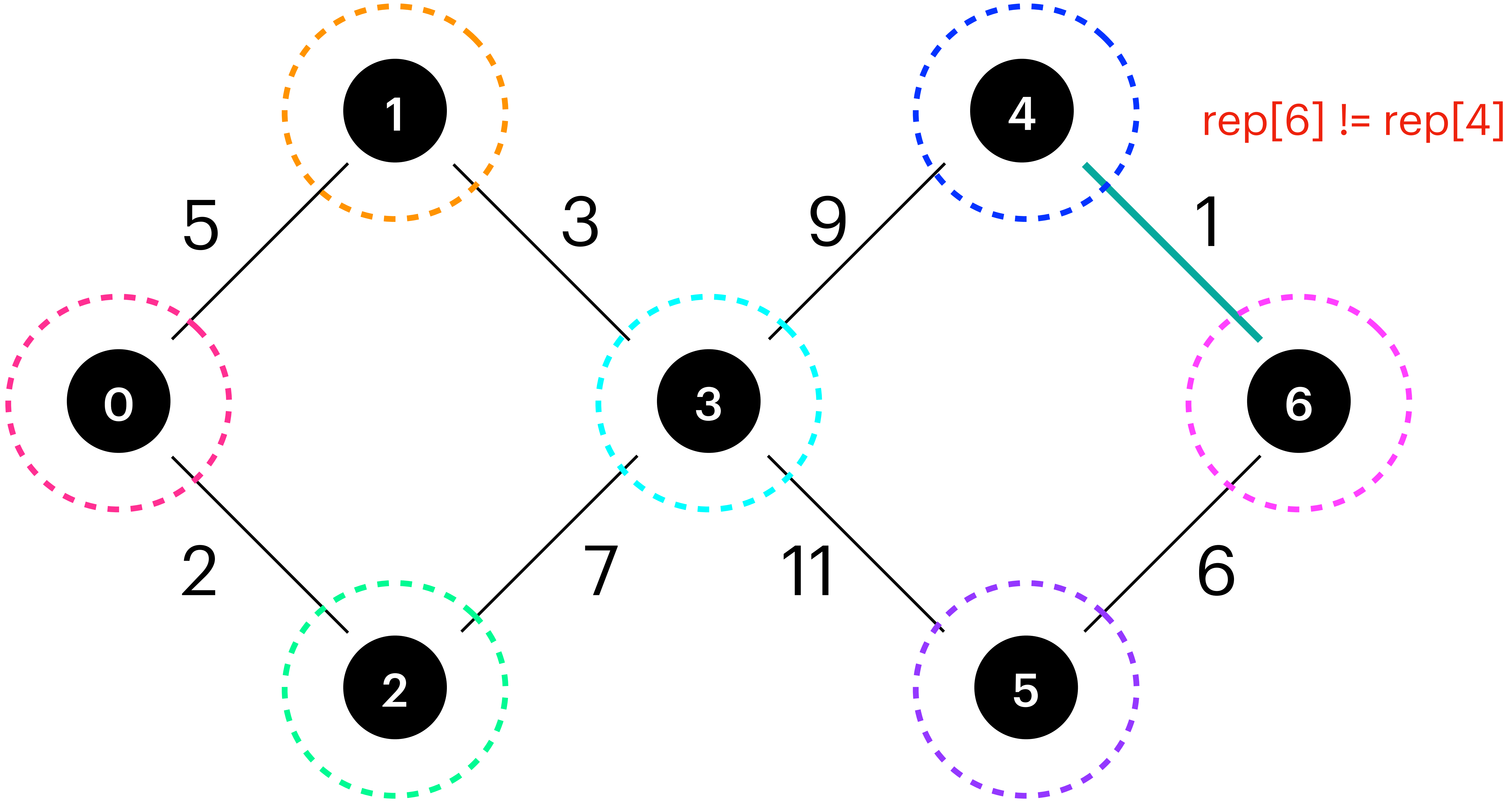
F : { {6,4} }

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	6

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

F : { {6,4} }

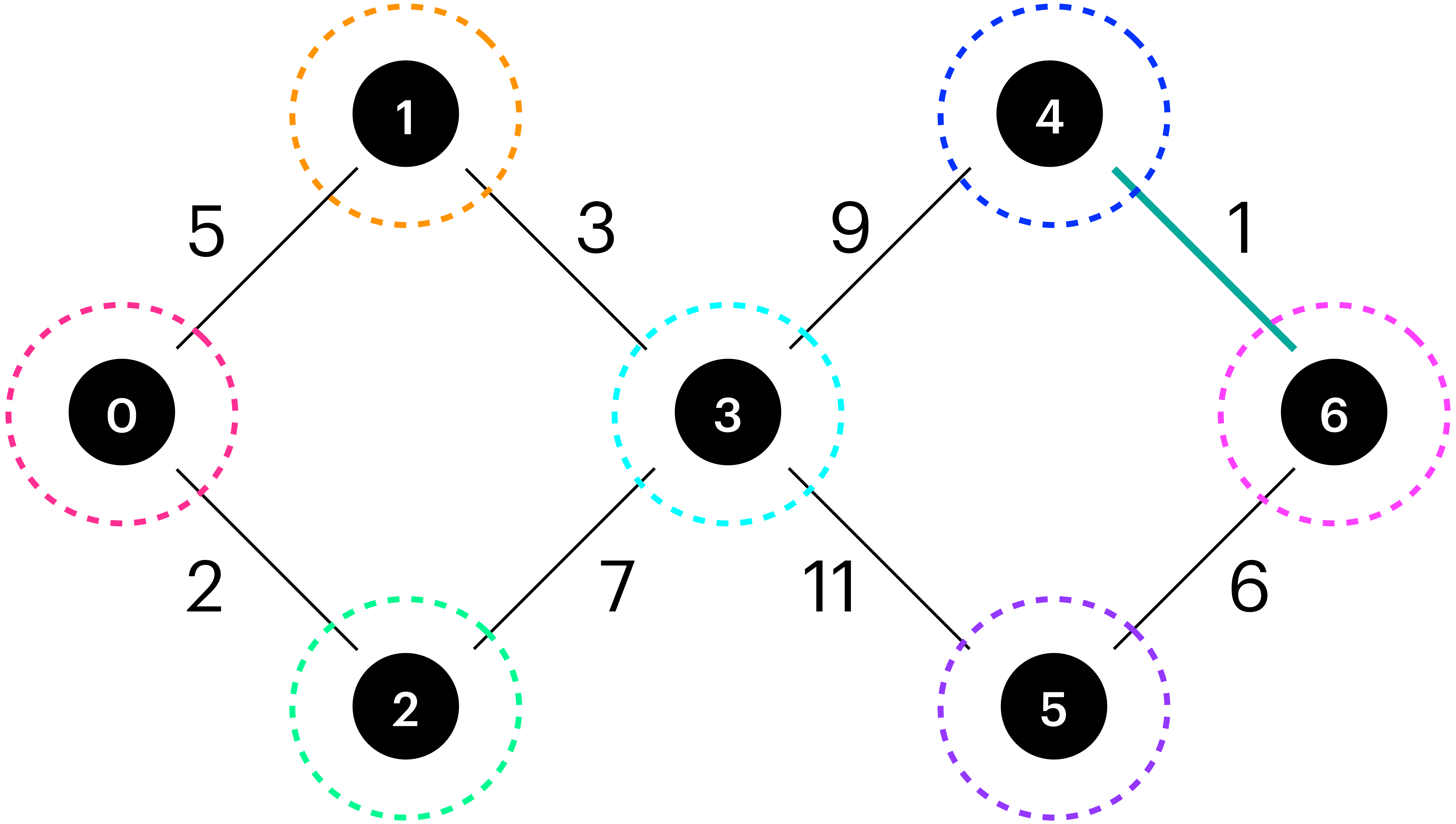
UNION(6,4)

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	6

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

F : { {6,4} }

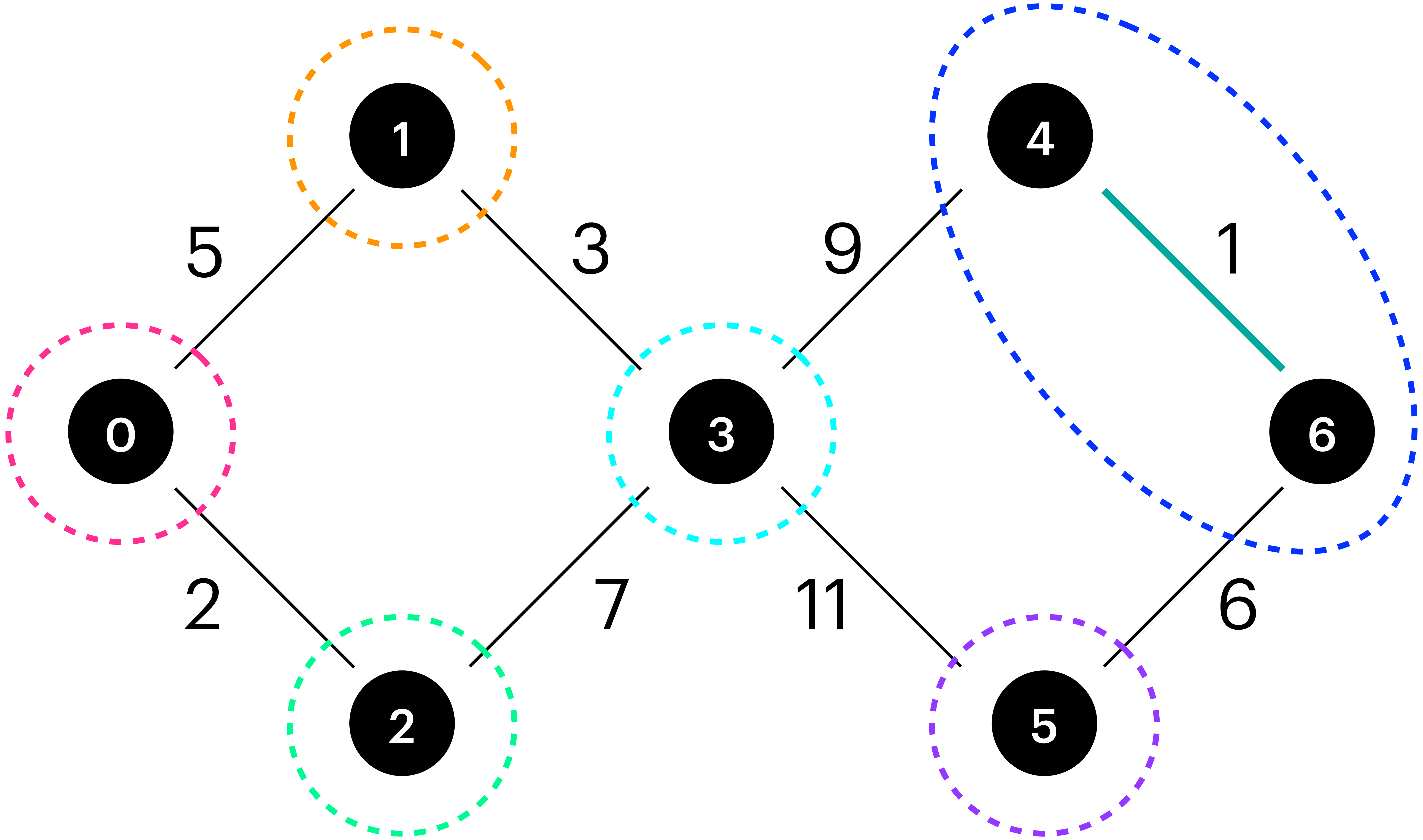
UNION(6,4)

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	4

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \ \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
rep[x] $\leftarrow$ rep[v]
members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
if SAME(u,v) = false **then**
$F \leftarrow F \cup \{uv\}$
UNION(u,v)

F : edges of the MST

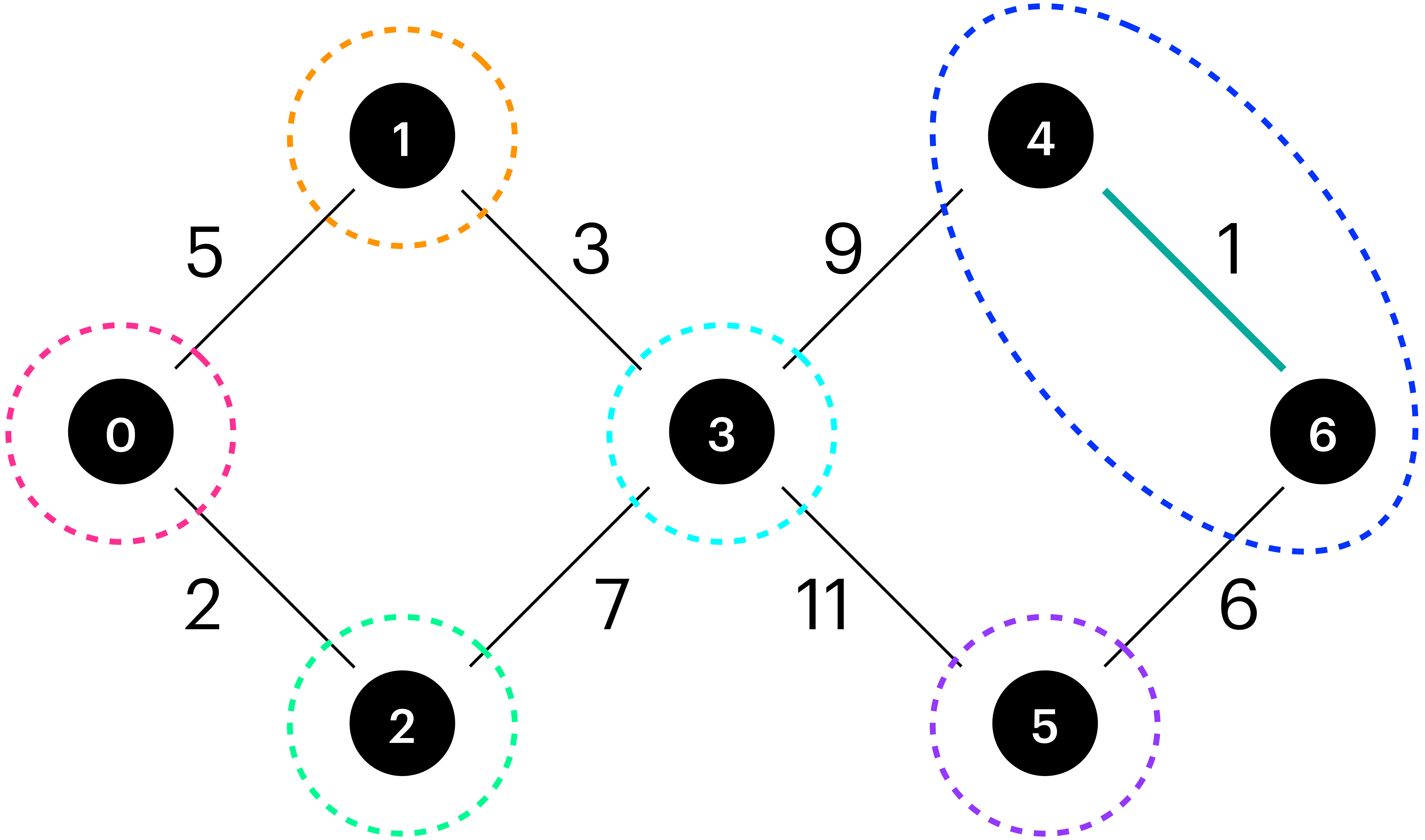
F : { {6,4} }

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	4

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

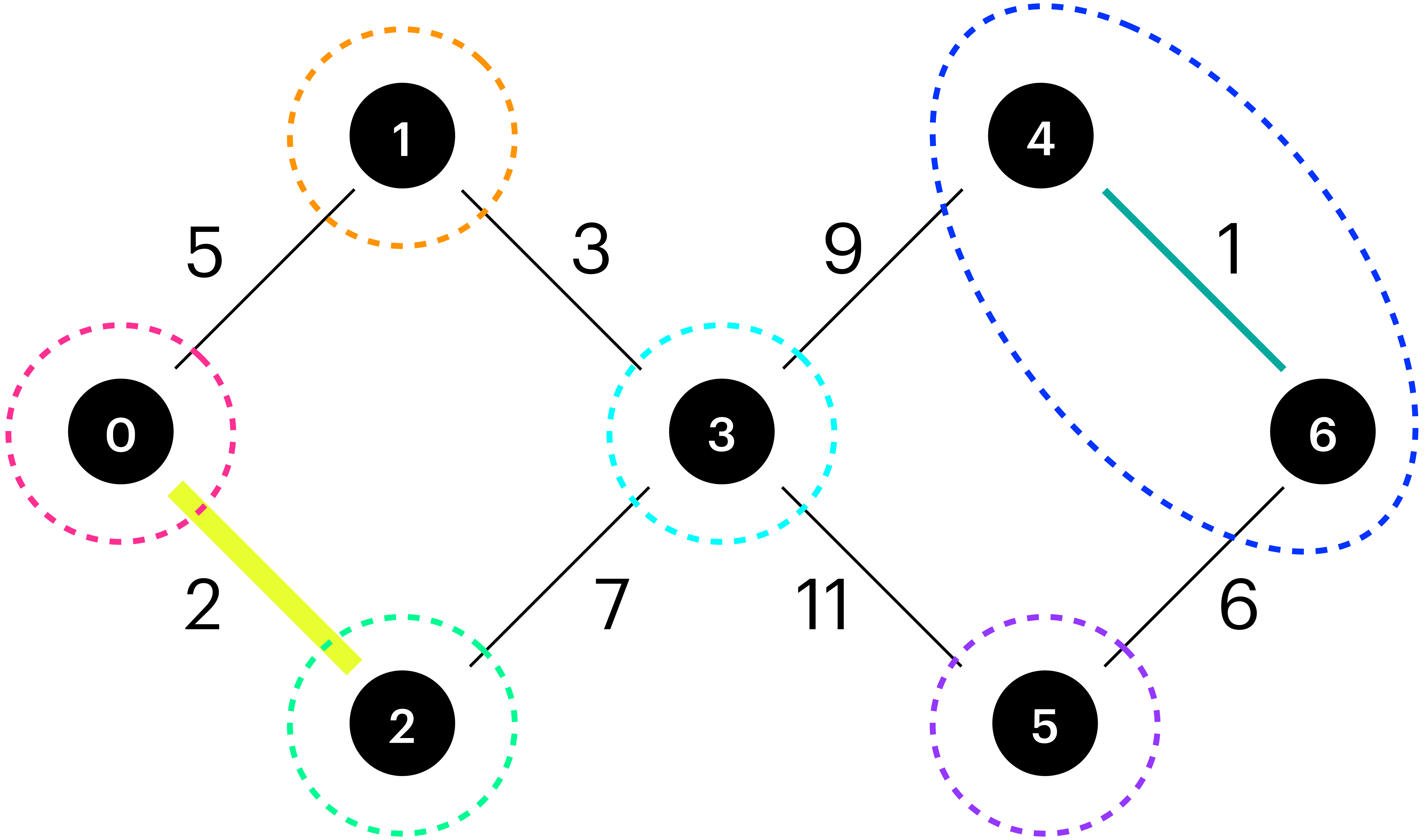
F : { {6,4} }

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	4

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

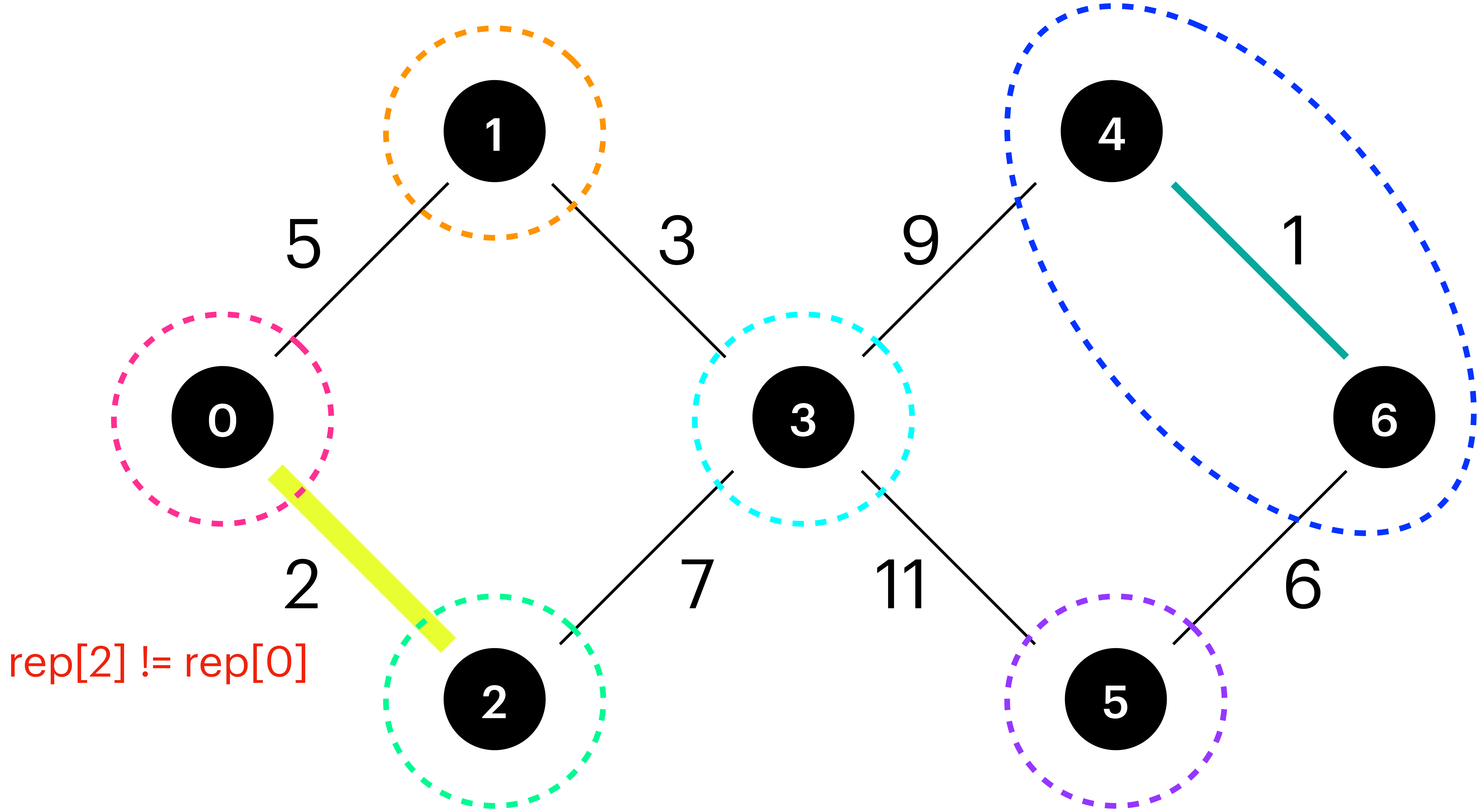
F : { {6,4} }

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	4

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \emptyset$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

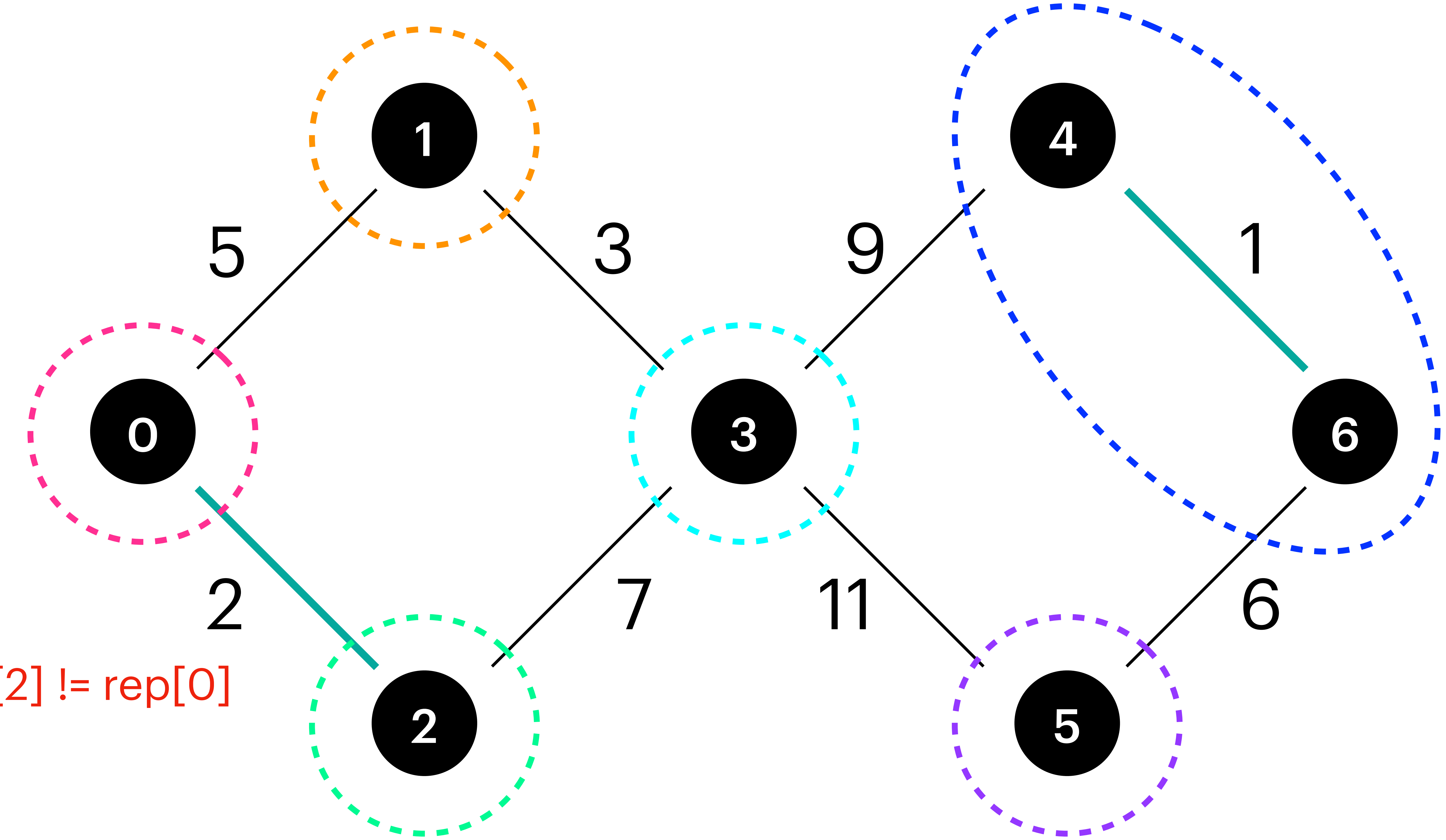
F : { {6,4} , {2,0} }

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	4

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



rep[2] != rep[0]

SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

1: **Implementierung:**
2: MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$
3:
4: SAME(u,v): teste ob rep[u] = rep[v]
5:
6: UNION(u,v):
7: **for** $x \in$ members[rep[u]] **do**
8: rep[x] $\leftarrow$ rep[v]
9: members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

1: $F \leftarrow \varnothing$
2: $UF \leftarrow$ MAKE(V)
3: SORT(E)
4: **for** $uv \in E$, aufsteigend sortiert **do**
5: **if** SAME(u,v) = false **then**
6: $F \leftarrow F \cup \{uv\}$
7: UNION(u,v)

F : edges of the MST

F : { {6,4} , {2,0} }

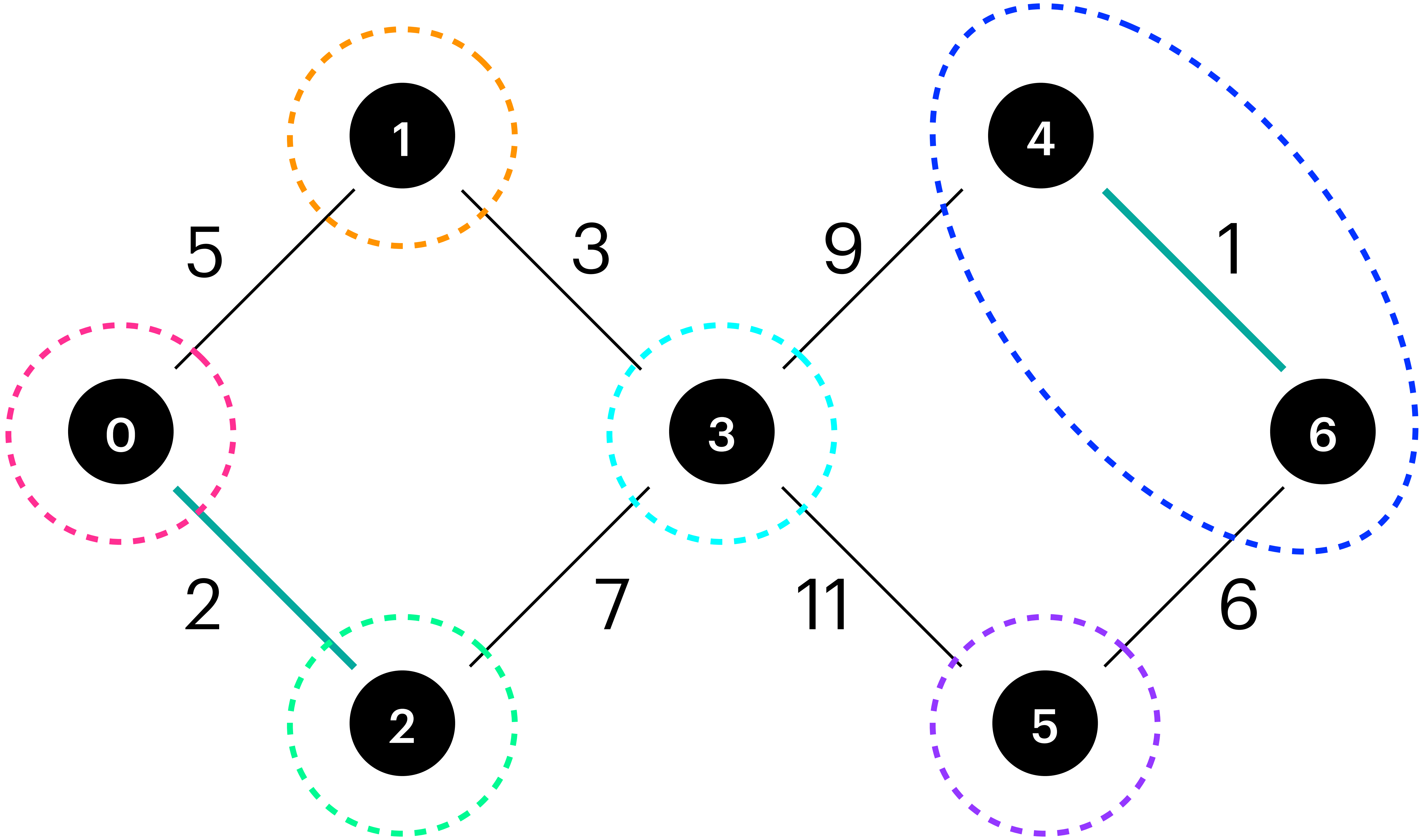
UNION(2,0)

rep[] :

0	1	2	3	4	5	6
0	1	2	3	4	5	4

members[] :

0	{ 0 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

F : { {6,4} , {2,0} }

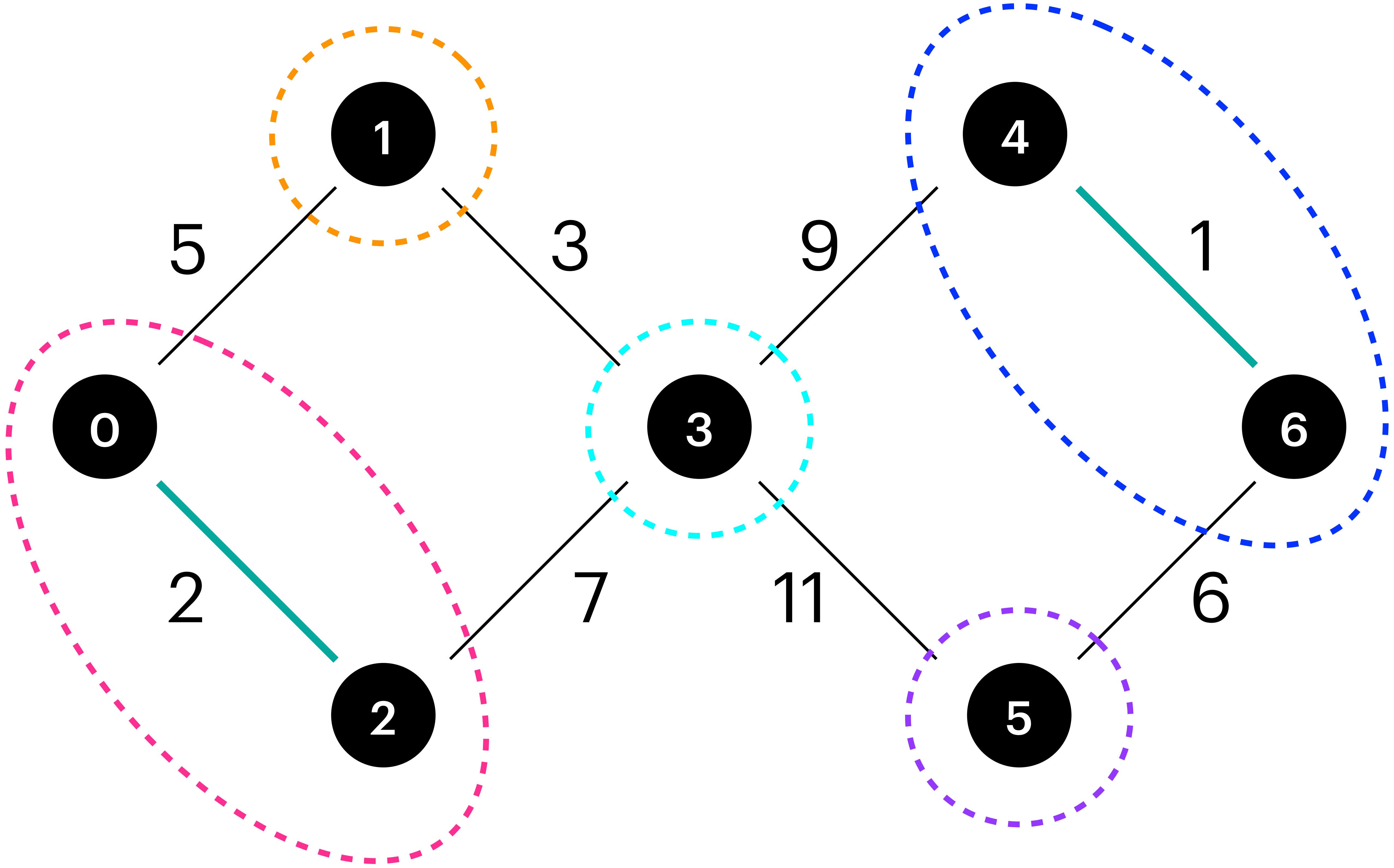
UNION(2,0)

rep[] :

0	1	2	3	4	5	6
0	1	0	3	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \;\; \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

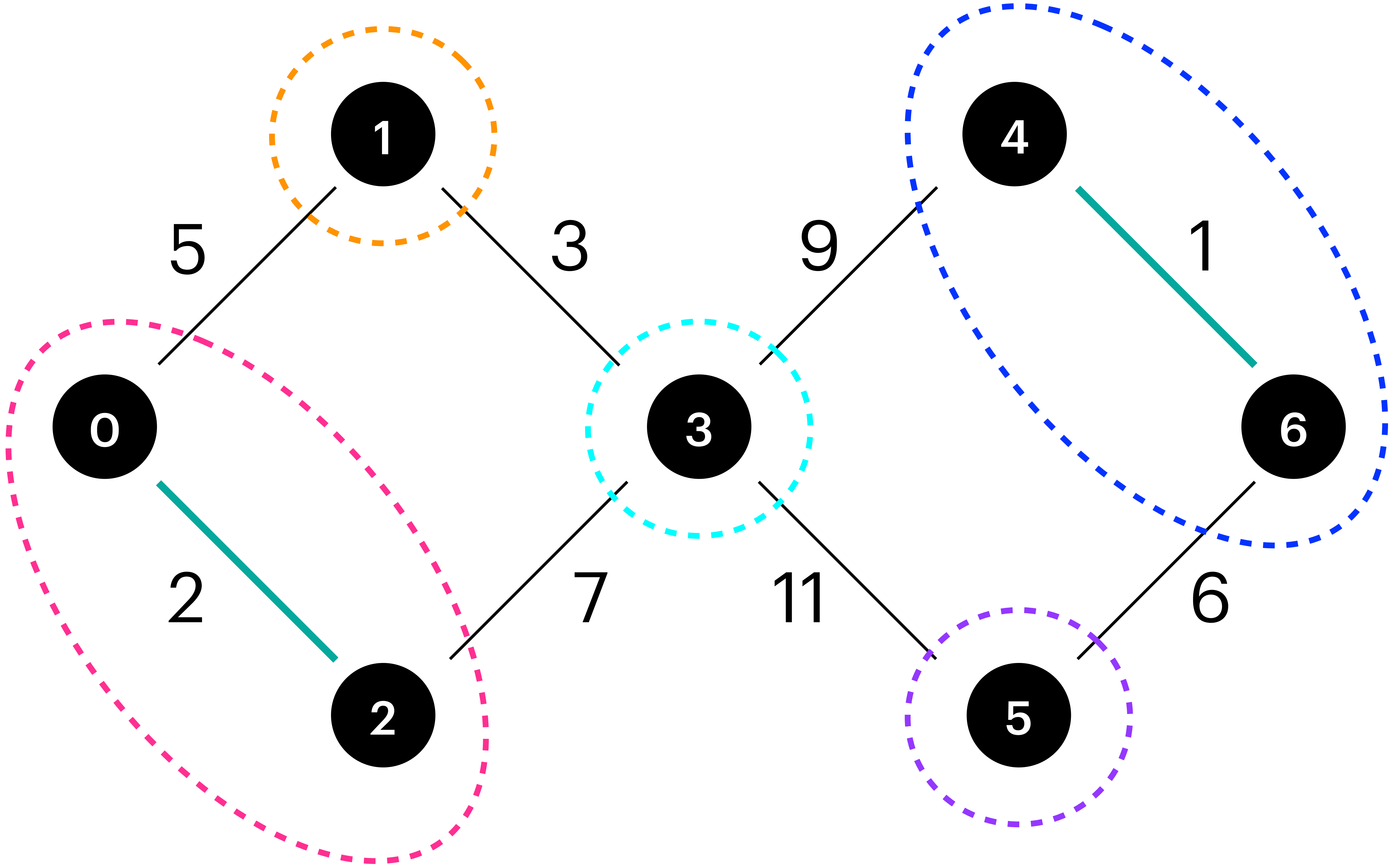
F : { {6,4} , {2,0} }

rep[] :

0	1	2	3	4	5	6
0	1	0	3	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \;\; \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
    rep[x] $\leftarrow$ rep[v]
    members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
    if SAME(u,v) = false **then**
        $F \leftarrow F \cup \{uv\}$
        UNION(u,v)

F : edges of the MST

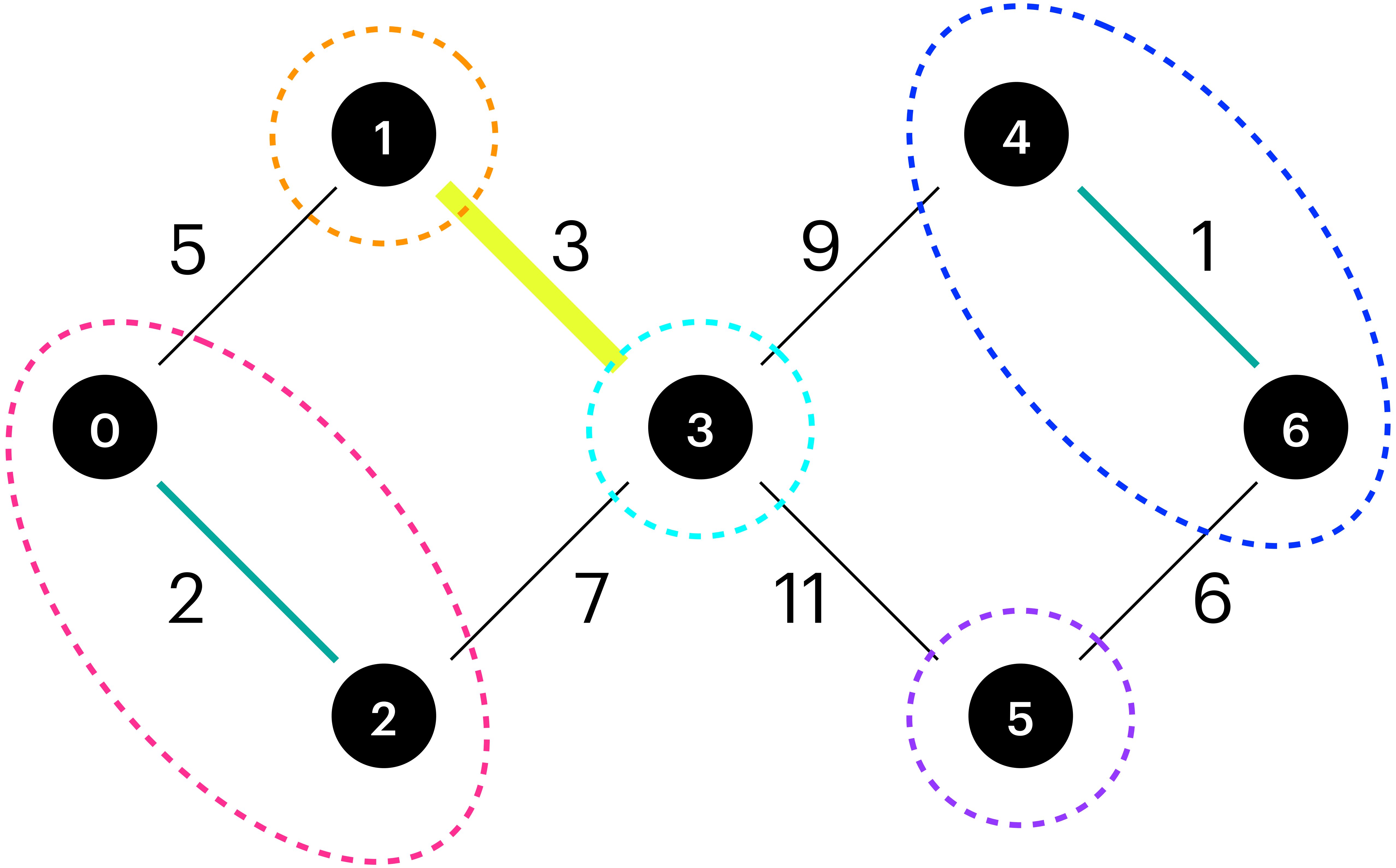
F : { {6,4} , {2,0} }

rep[] :

0	1	2	3	4	5	6
0	1	0	3	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

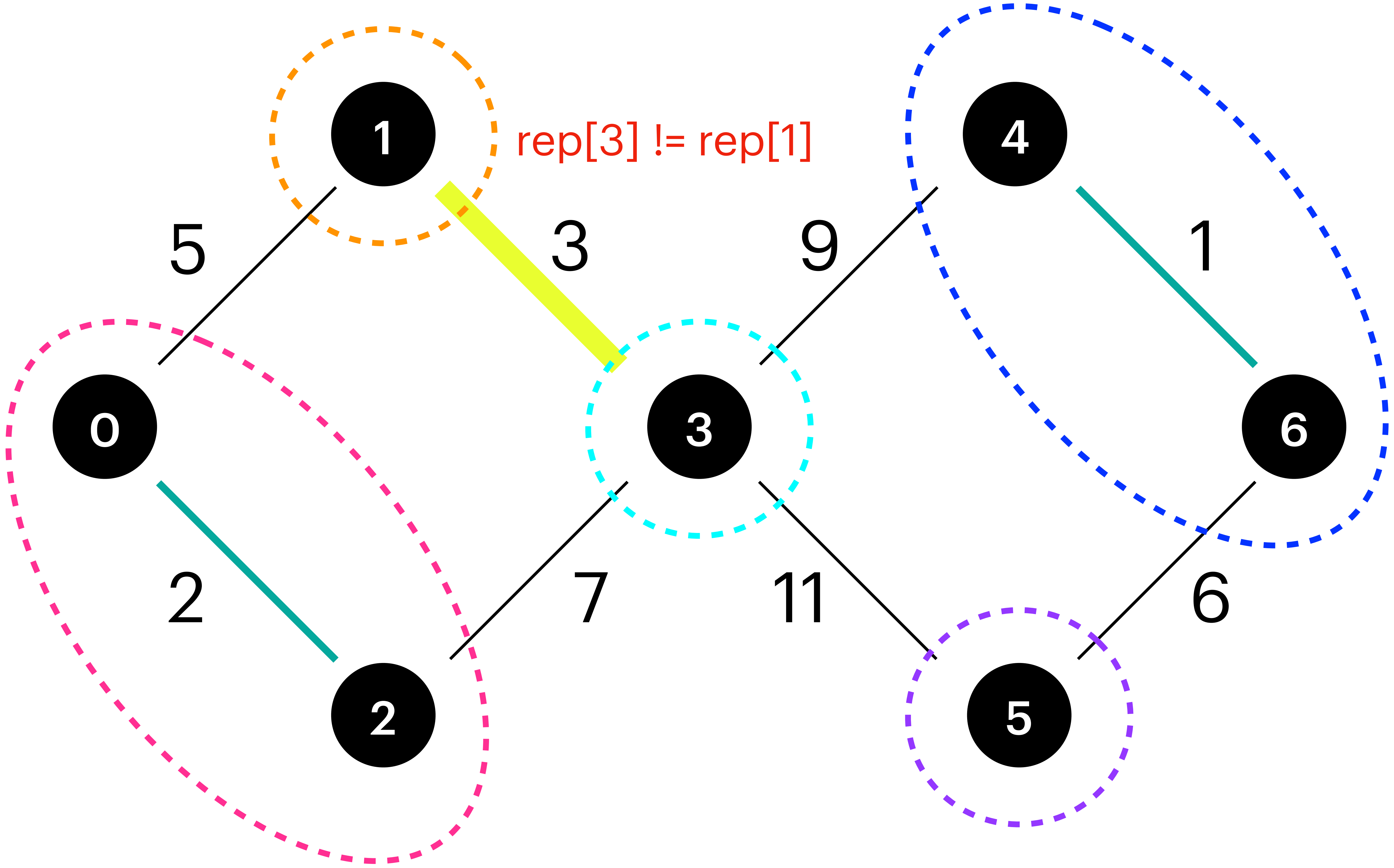
F : { {6,4} , {2,0} }

rep[] :

0	1	2	3	4	5	6
0	1	0	3	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

F : { {6,4} , {2,0} , {3,1} }

rep[] :

0	1	2	3	4	5	6
0	1	0	3	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

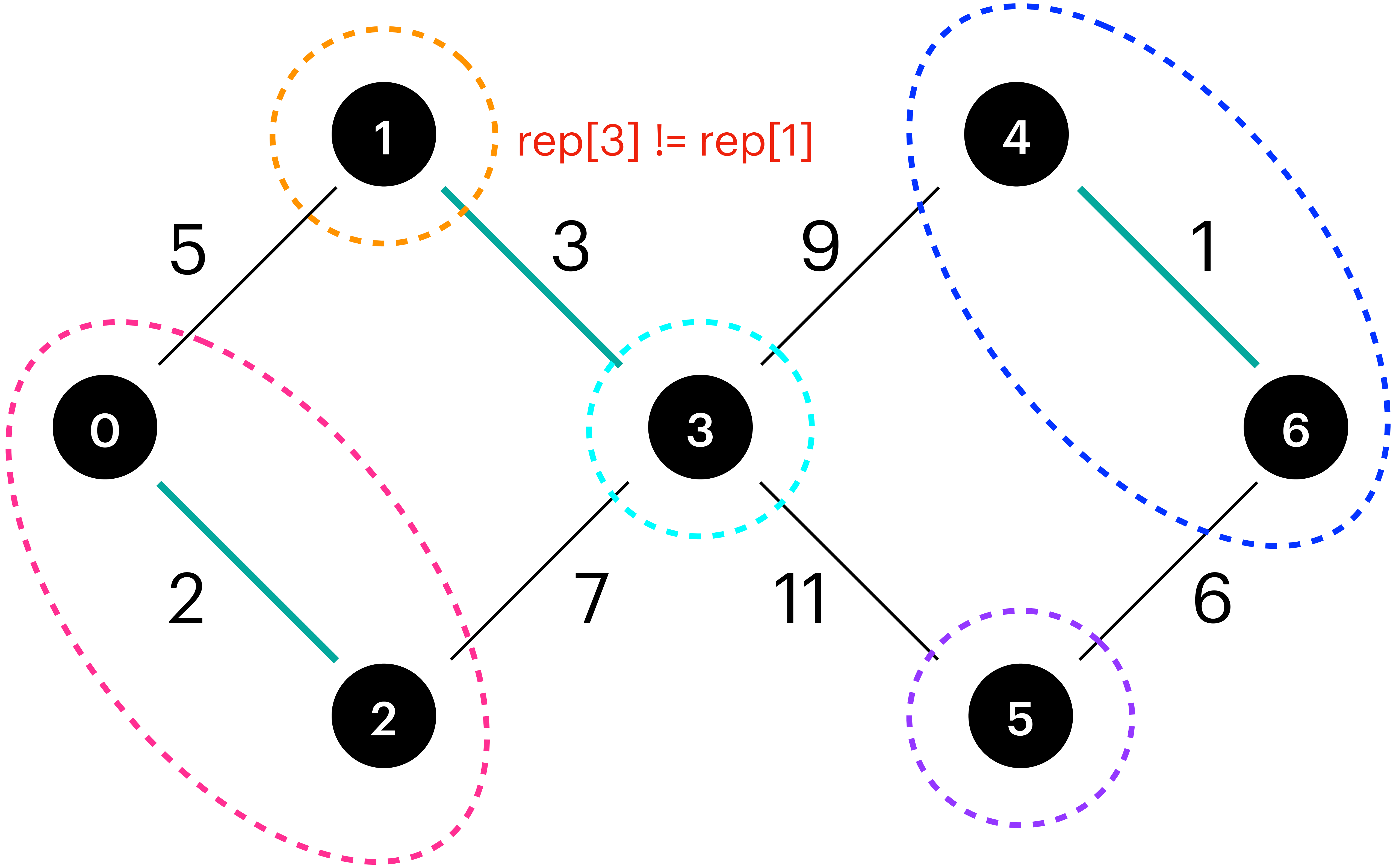
rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} }

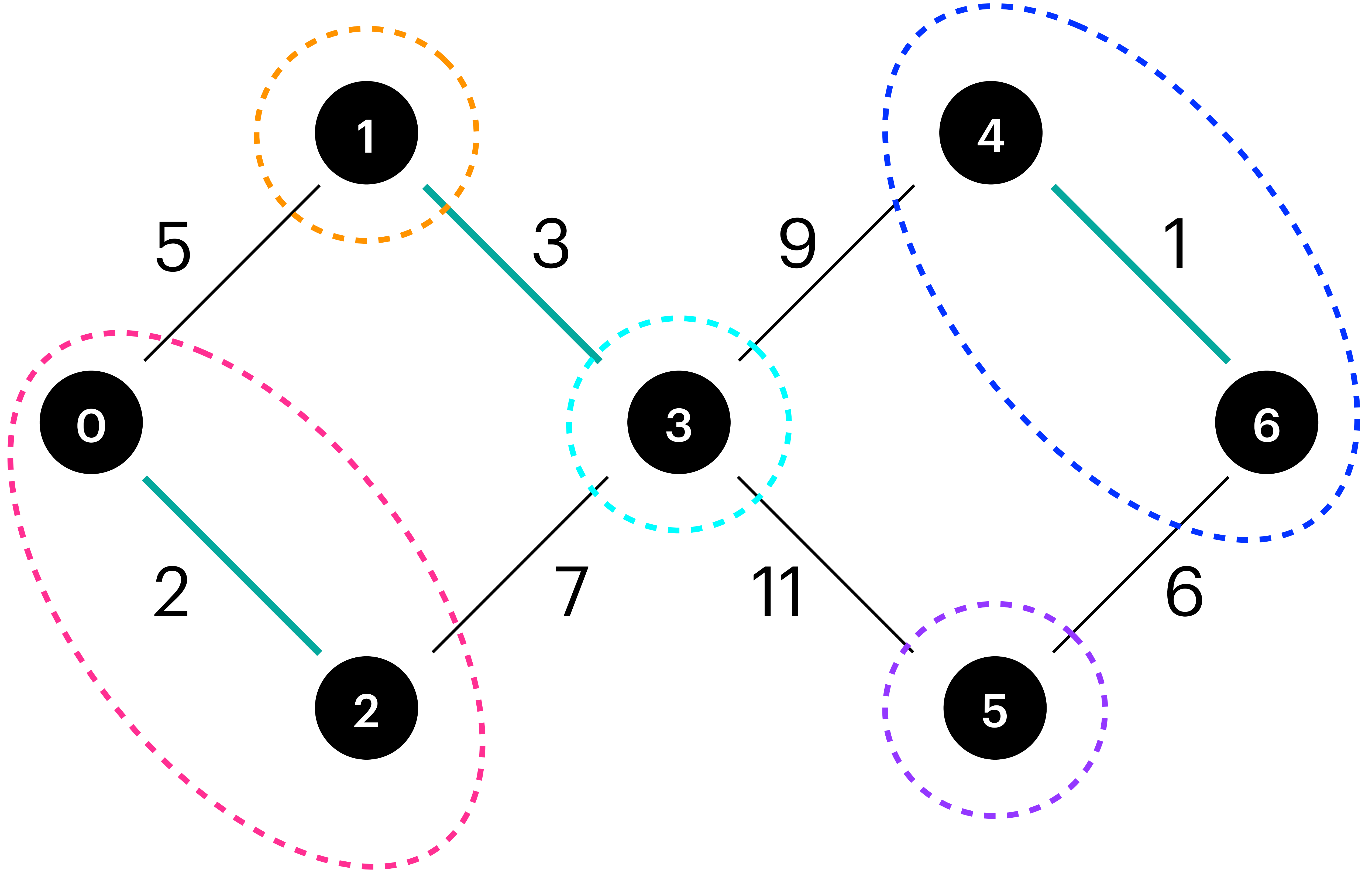
UNION(3,1)

rep[] :

0	1	2	3	4	5	6
0	1	0	3	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
rep[x] $\leftarrow$ rep[v]
members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} }

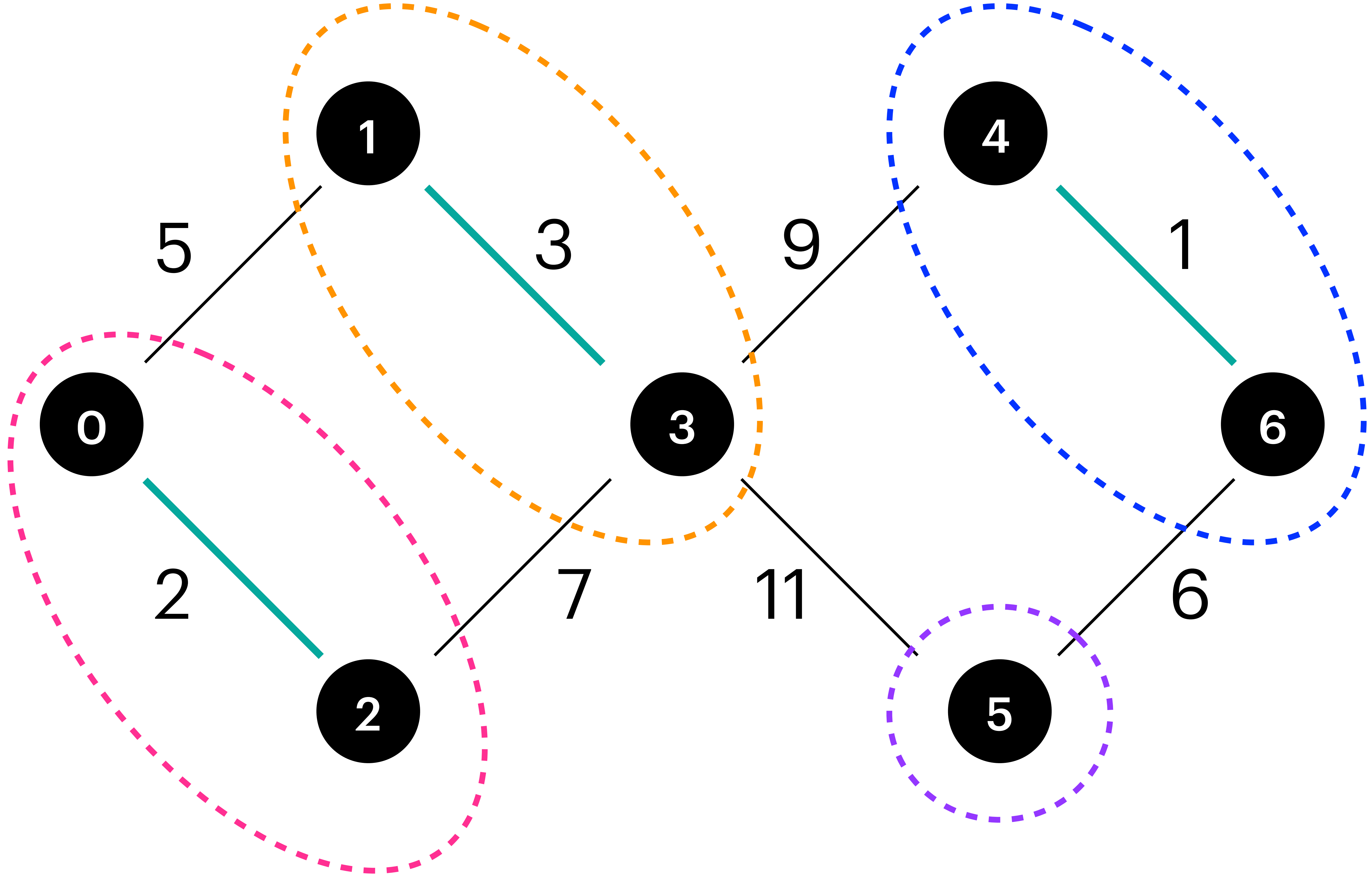
UNION(3,1)

rep[] :

0	1	2	3	4	5	6
0	1	0	1	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:

MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

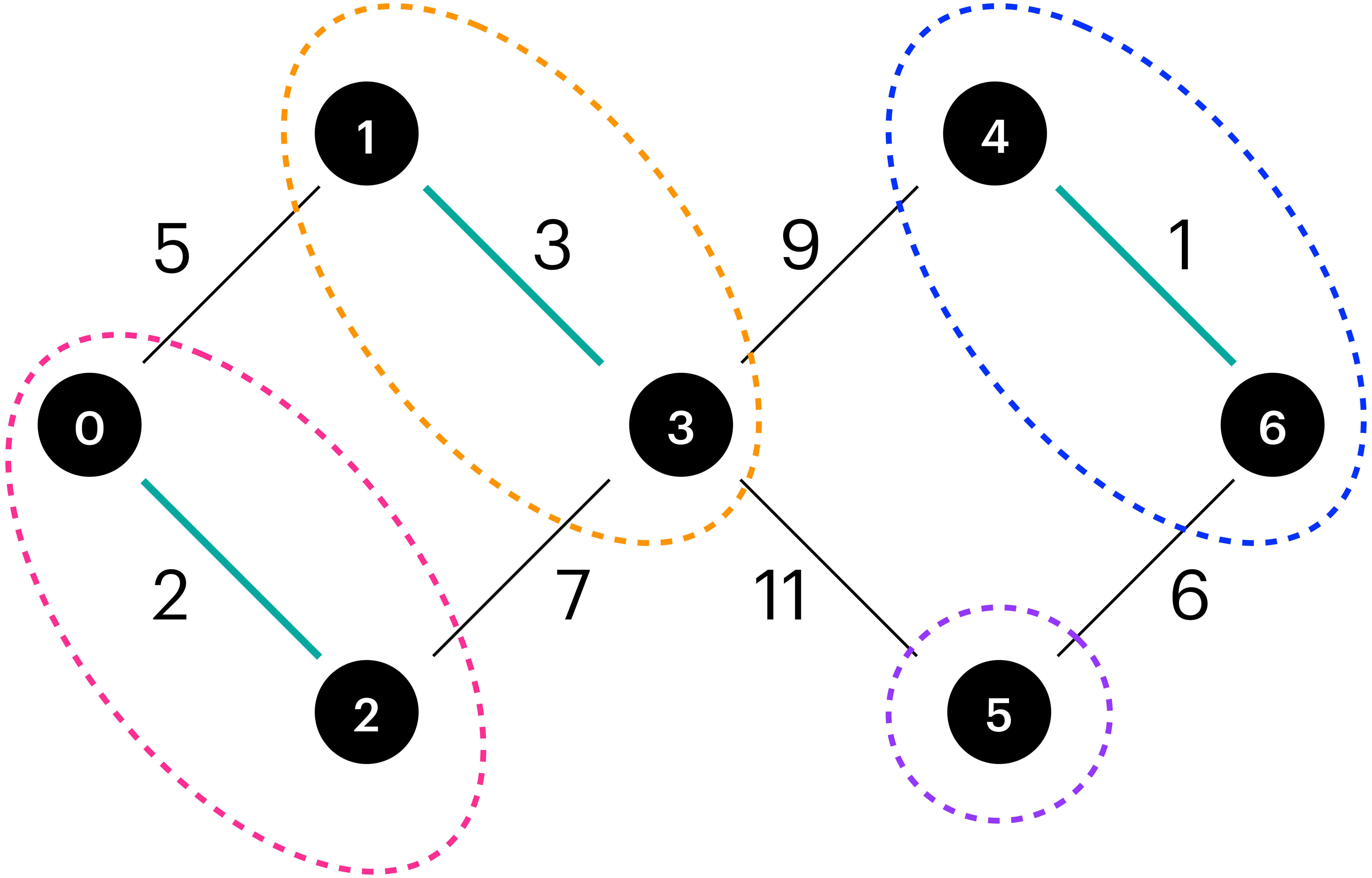
F : { {6,4} , {2,0} , {3,1} }

rep[] :

0	1	2	3	4	5	6
0	1	0	1	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

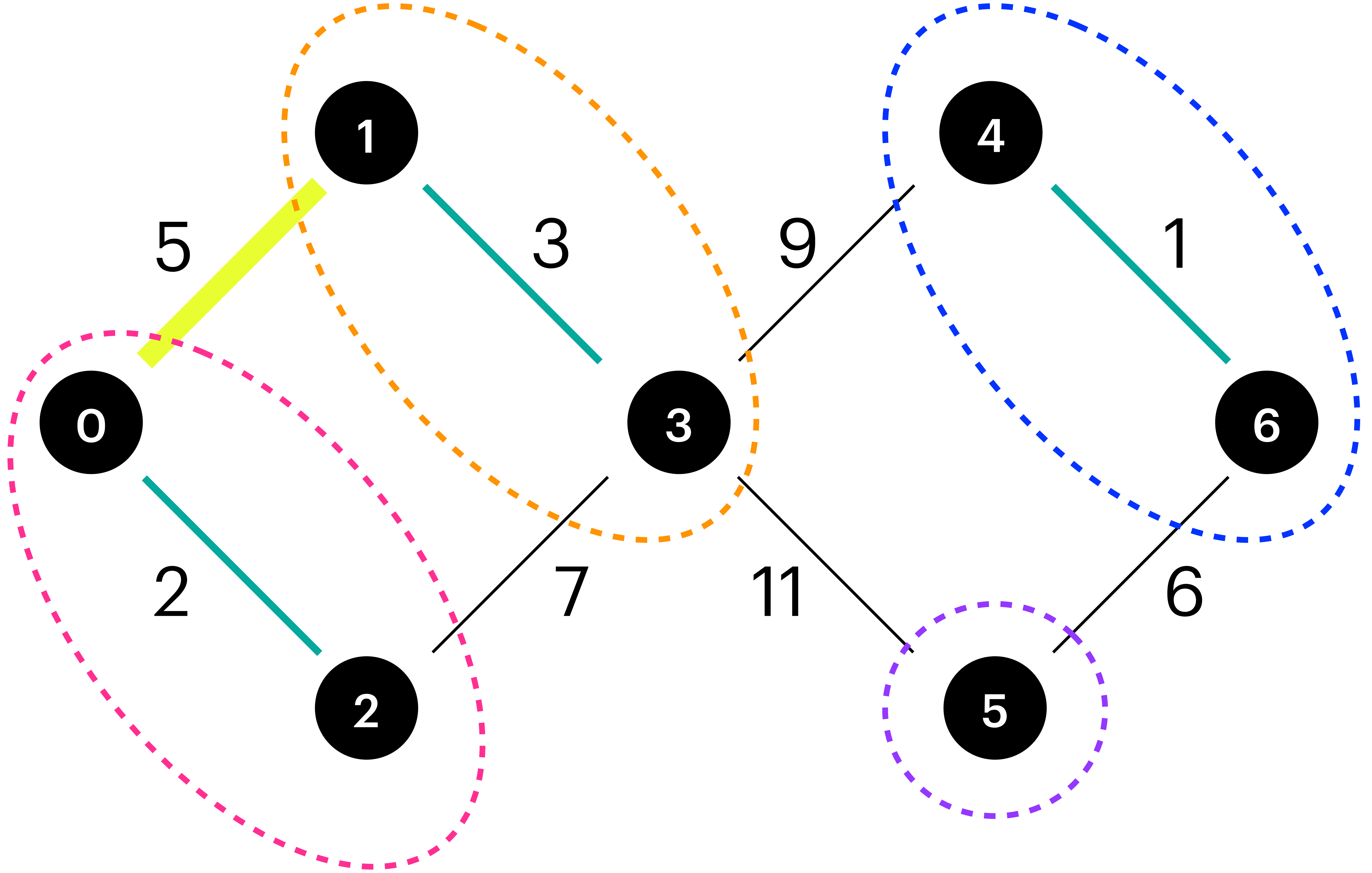
F : { {6,4} , {2,0} , {3,1} }

rep[] :

0	1	2	3	4	5	6
0	1	0	1	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

F : { {6,4} , {2,0} , {3,1} }

rep[] :

0	1	2	3	4	5	6
0	1	0	1	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \ \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

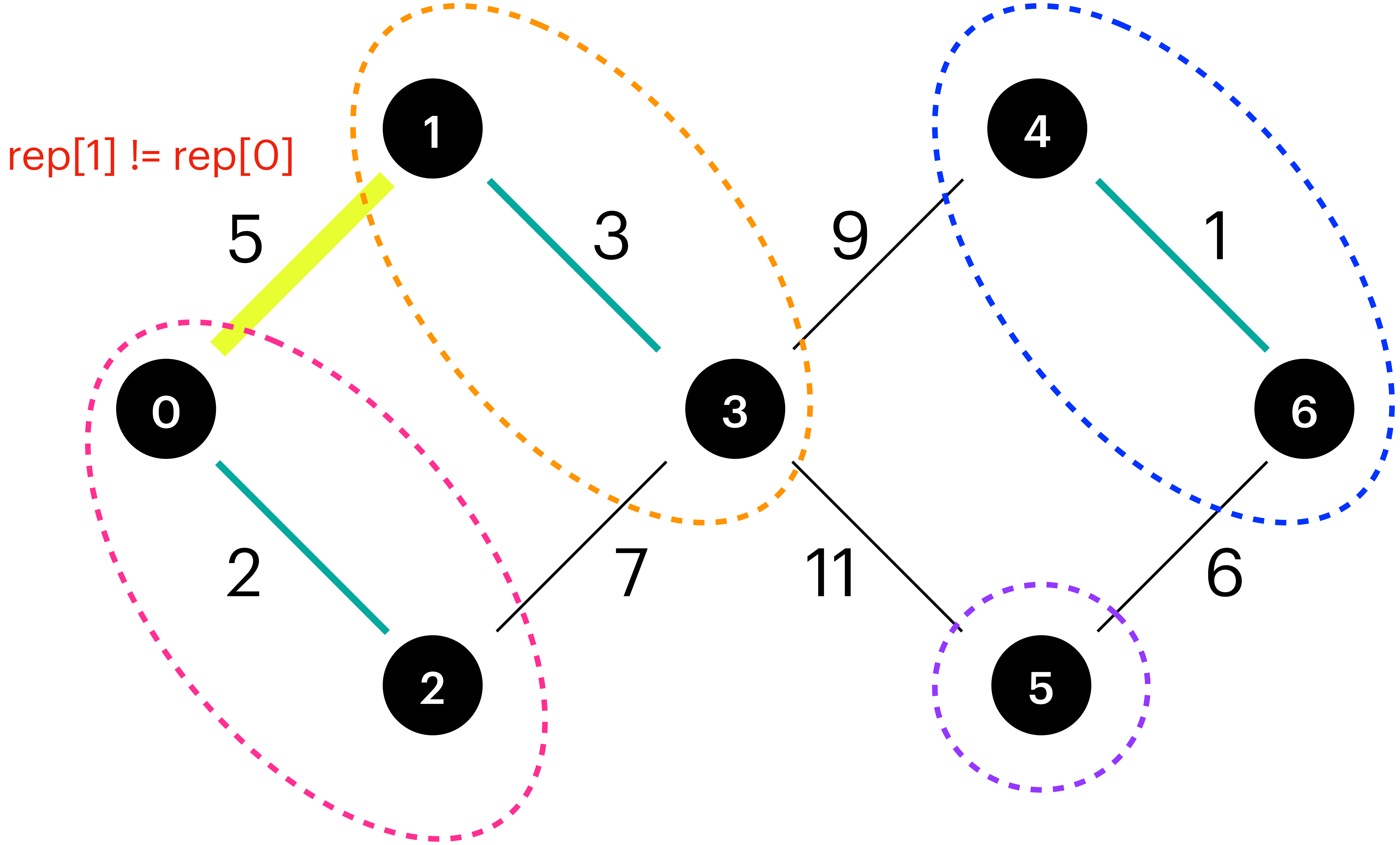
Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] ← v ∀$v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] ← rep[v]
  members[rep[v]] ← members[rep[v]] ∪ {x}

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

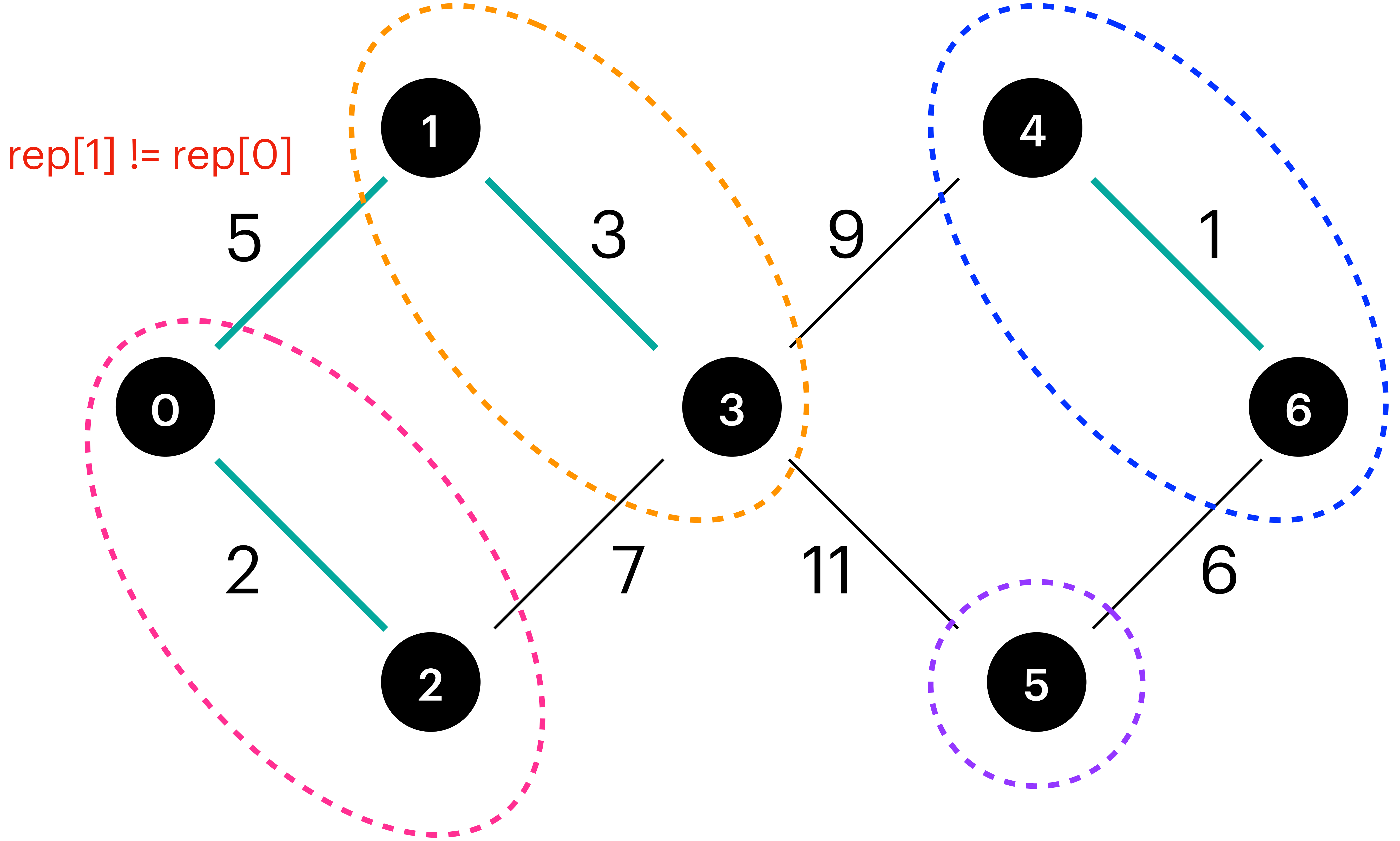
F : { {6,4} , {2,0} , {3,1} , {1,0} }

rep[] :

0	1	2	3	4	5	6
0	1	0	1	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \quad \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} }

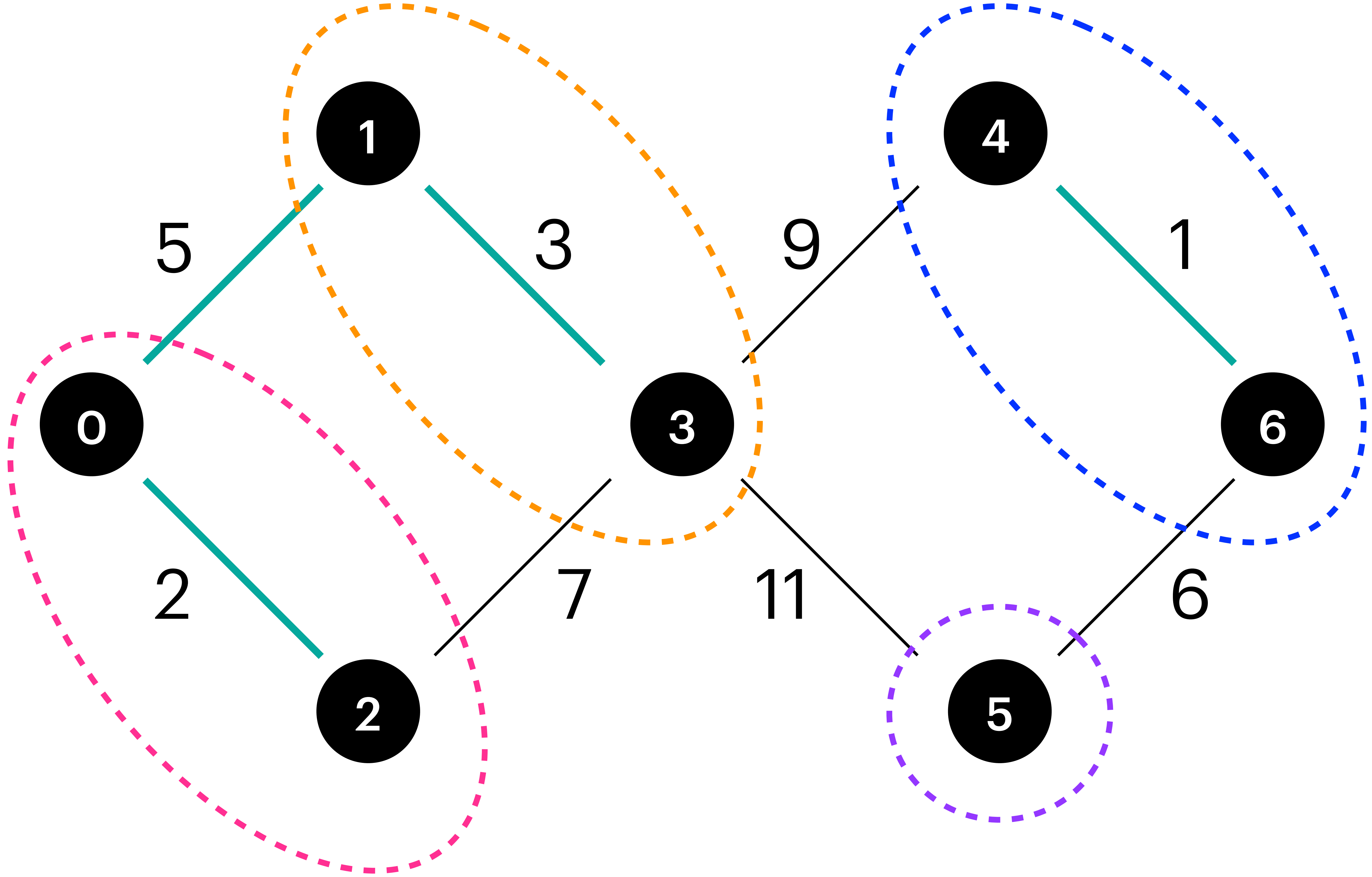
UNION(1,0)

rep[] :

0	1	2	3	4	5	6
0	1	0	1	4	5	4

members[] :

0	{ 0 , 2 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} }

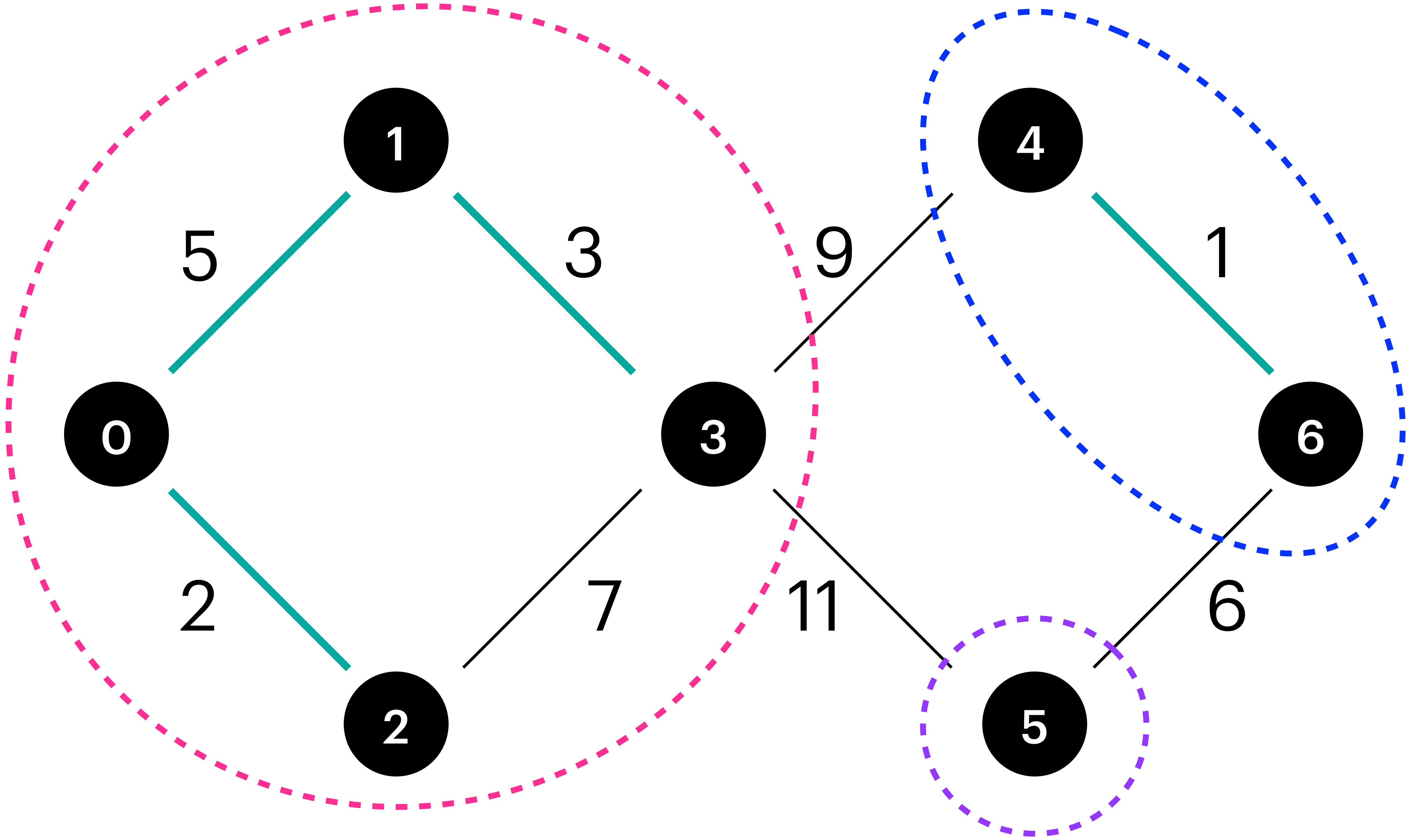
UNION(1,0)

rep[] :

0	1	2	3	4	5	6
0	0	0	0	4	5	4

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:

MAKE(V): rep[v] $\leftarrow v \ \ \forall v \in V$

SAME(u,v): teste ob rep[u] $=$ rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) $=$ false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

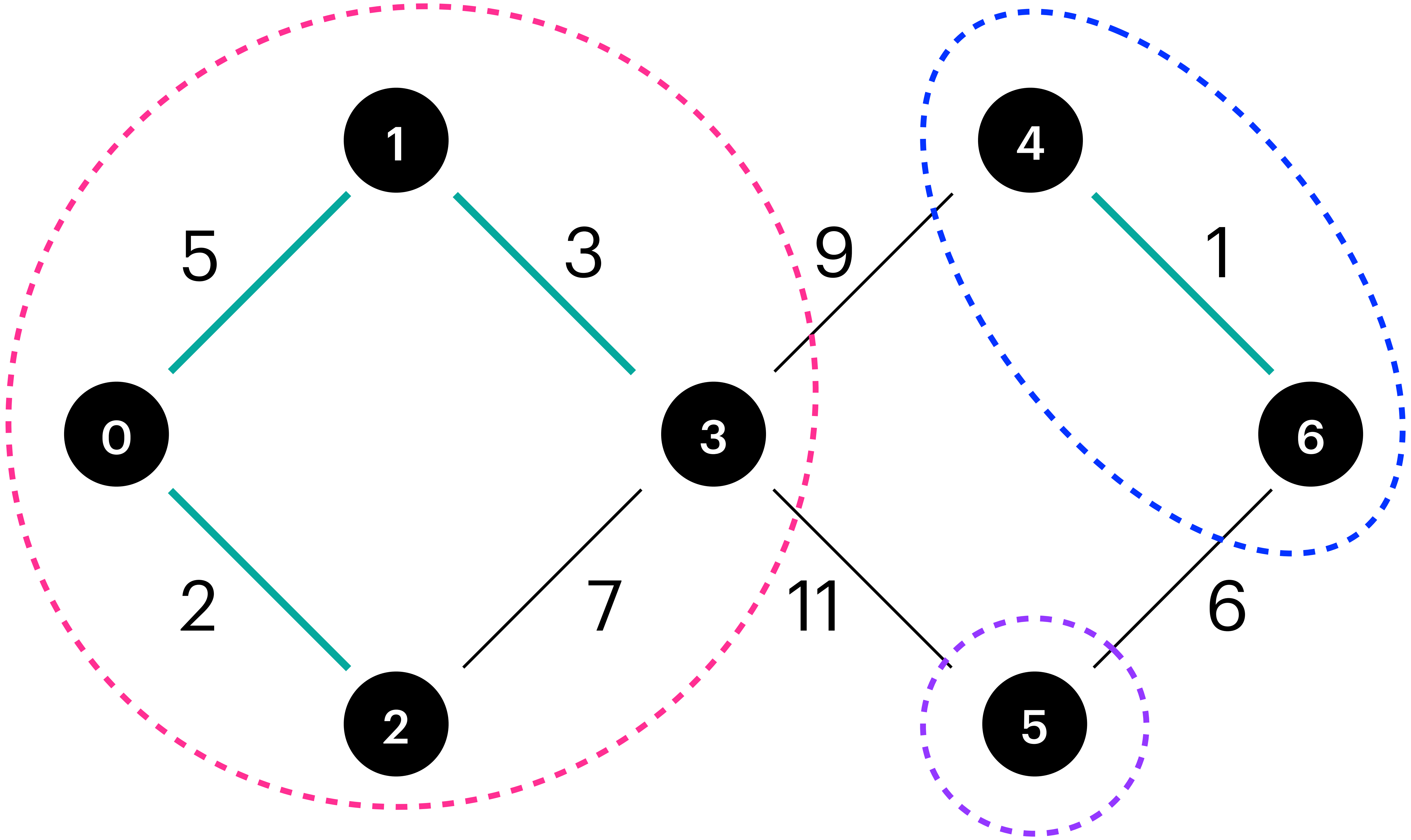
F : { {6,4} , {2,0} , {3,1} , {1,0} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	4	5	4

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \ \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

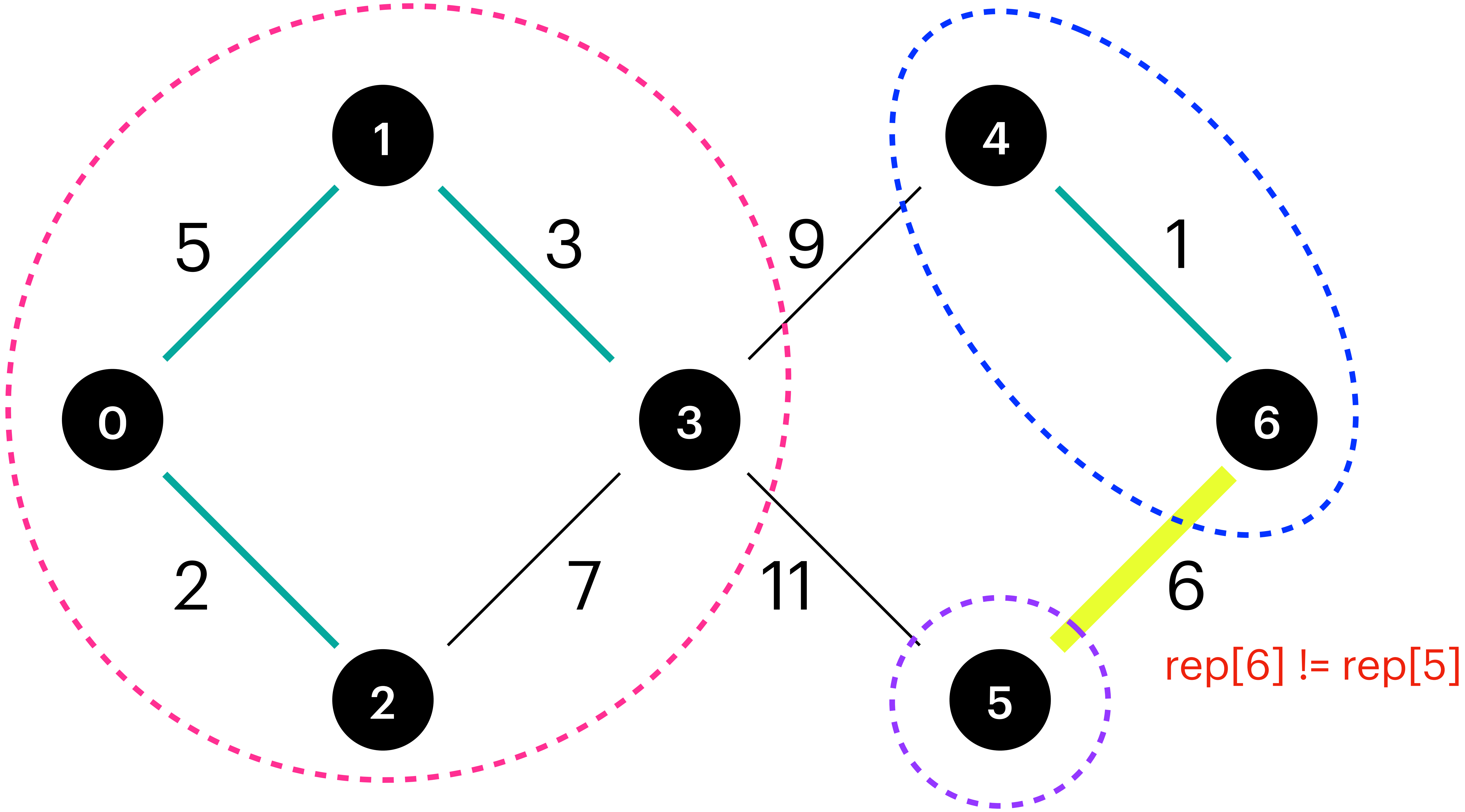
F : { {6,4} , {2,0} , {3,1} , {1,0} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	4	5	4

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

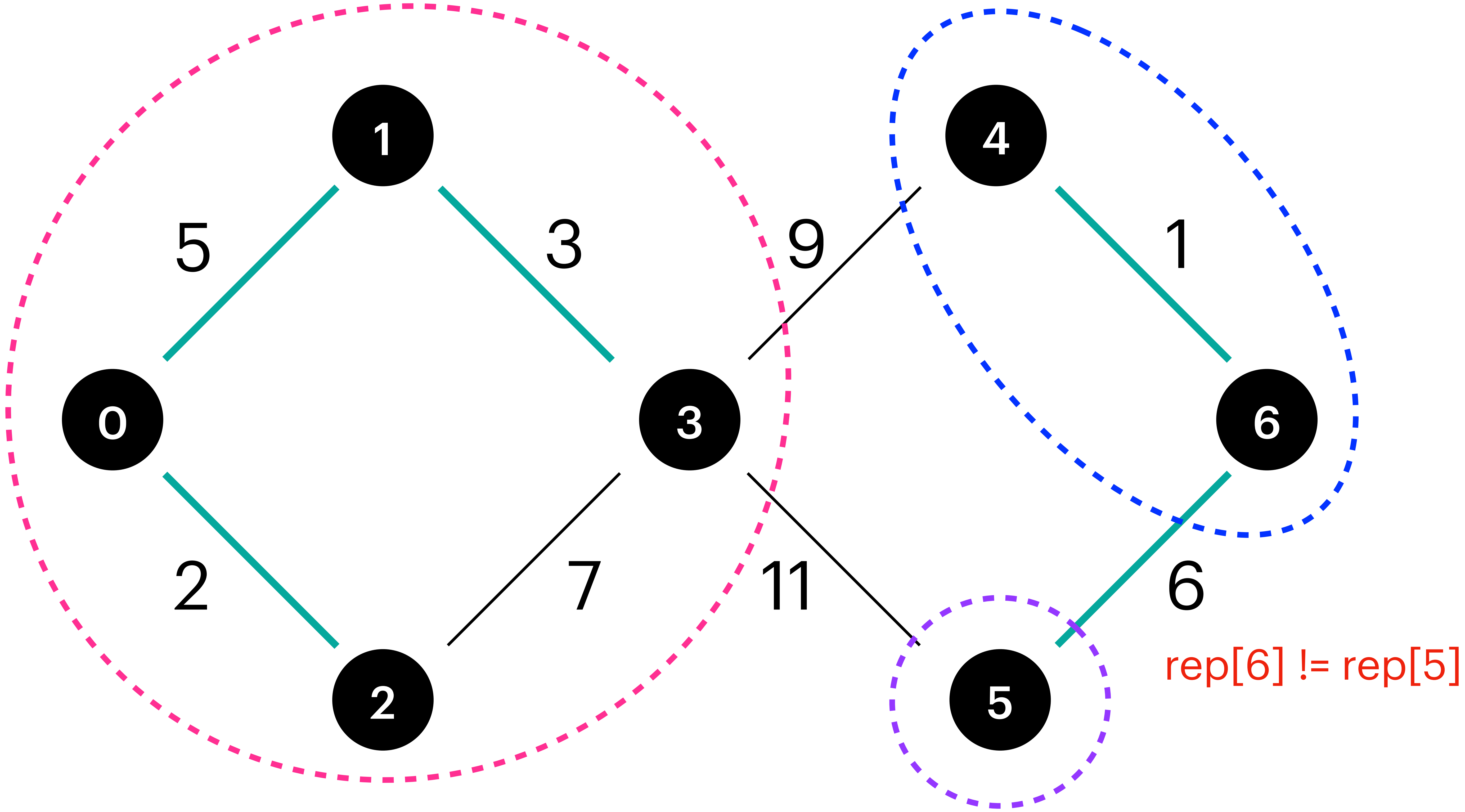
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	4	5	4

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

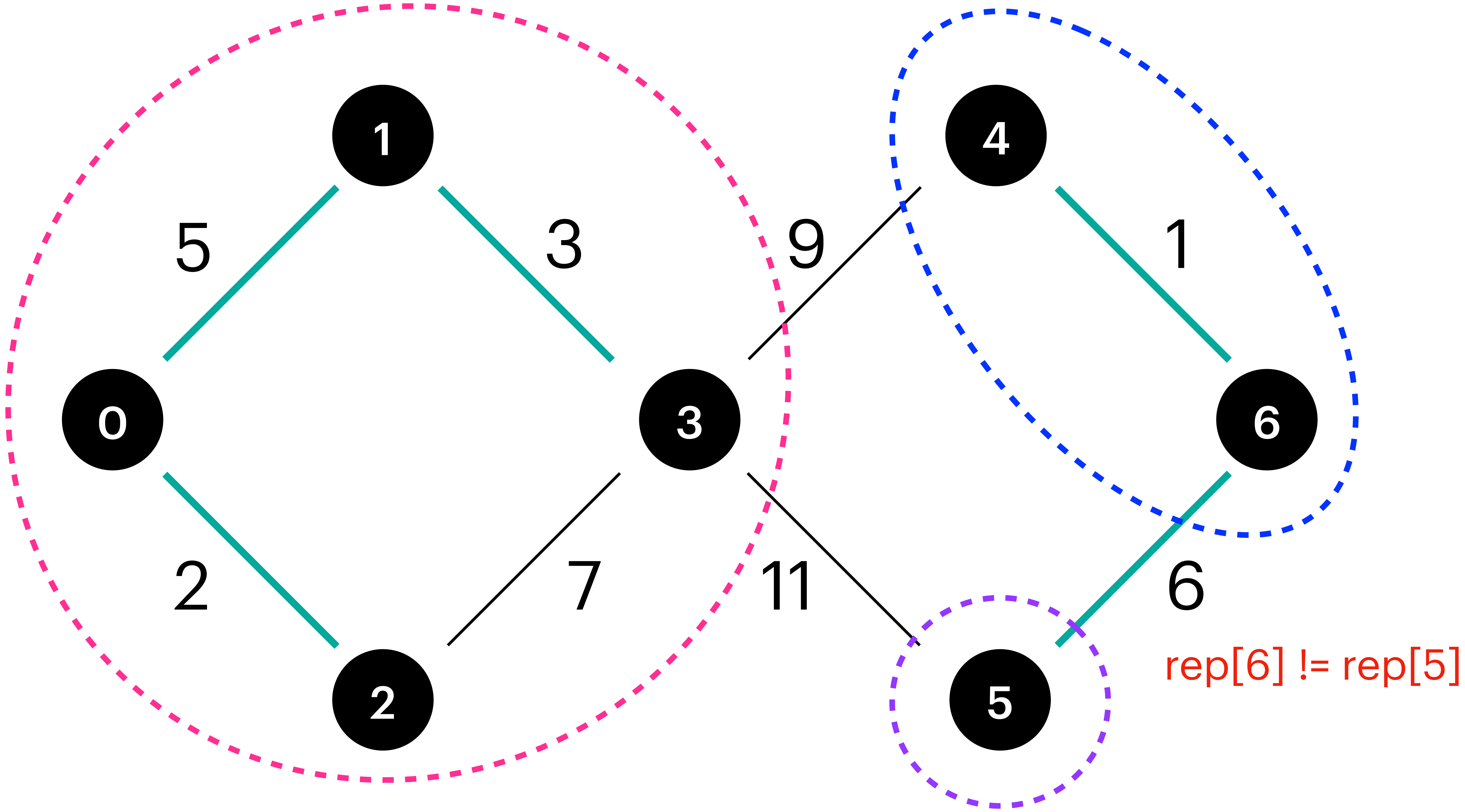
UNION(6,5)

rep[] :

0	1	2	3	4	5	6
0	0	0	0	4	5	4

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):   teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

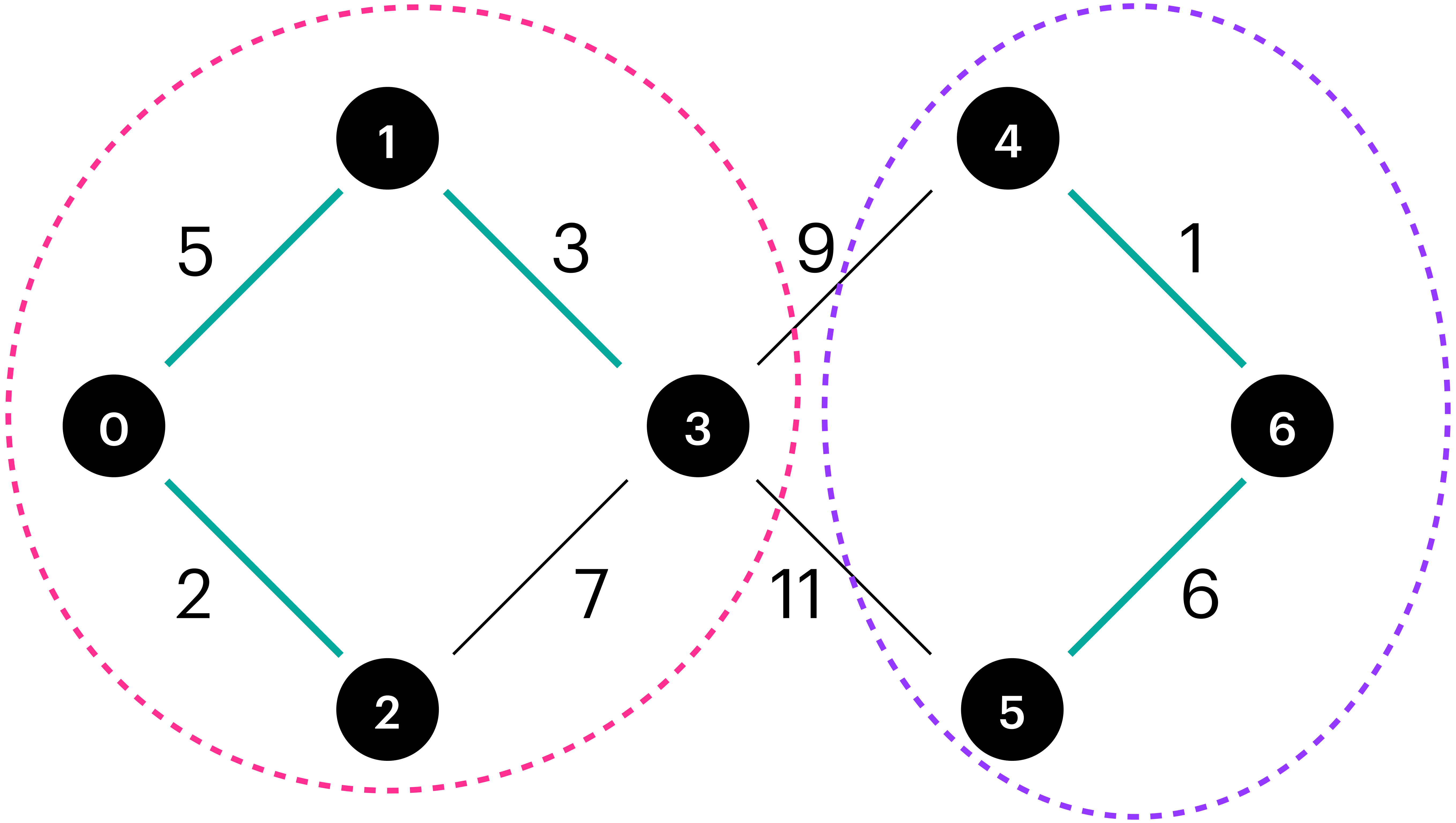
UNION(6,5)

rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

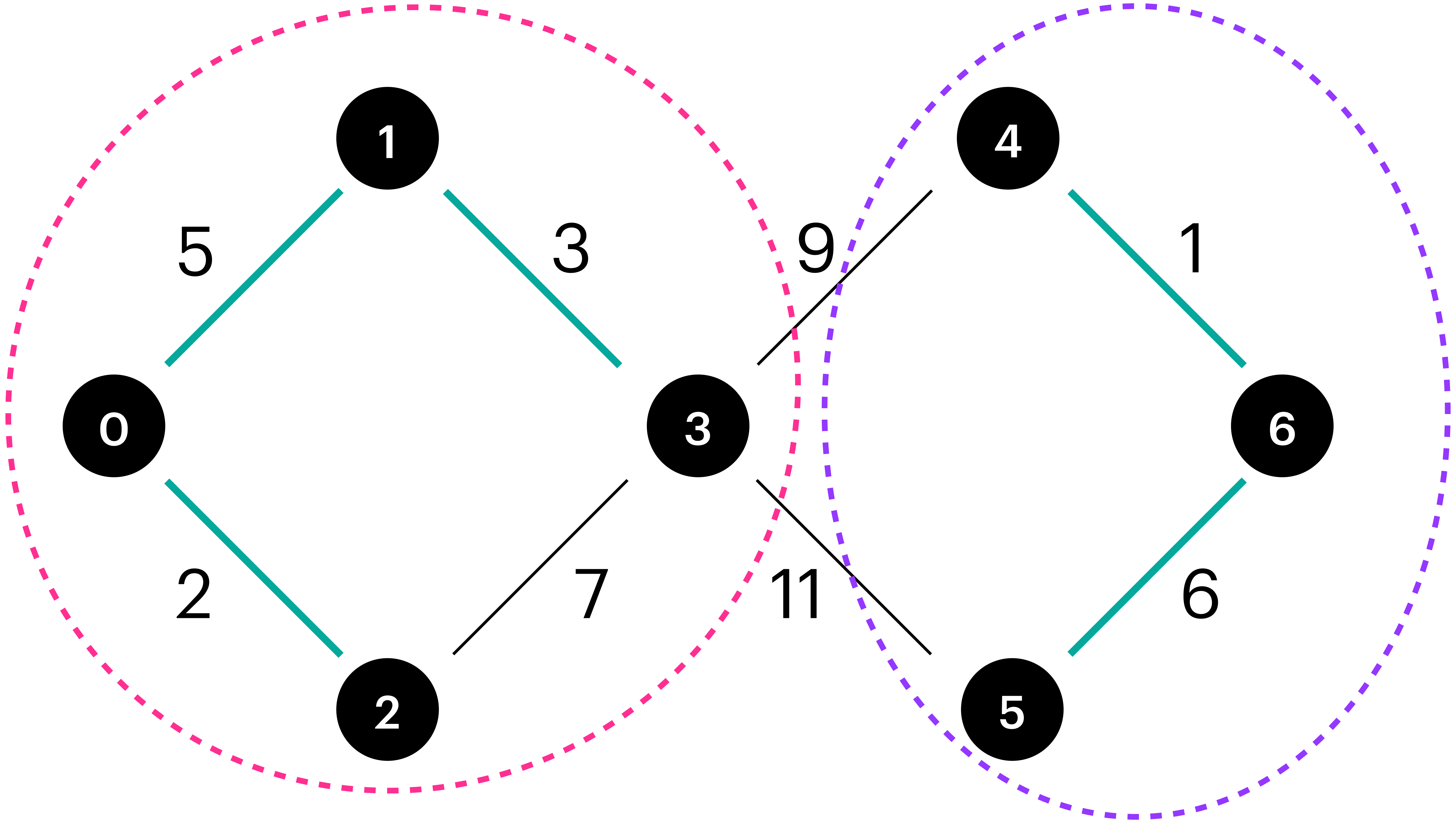
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

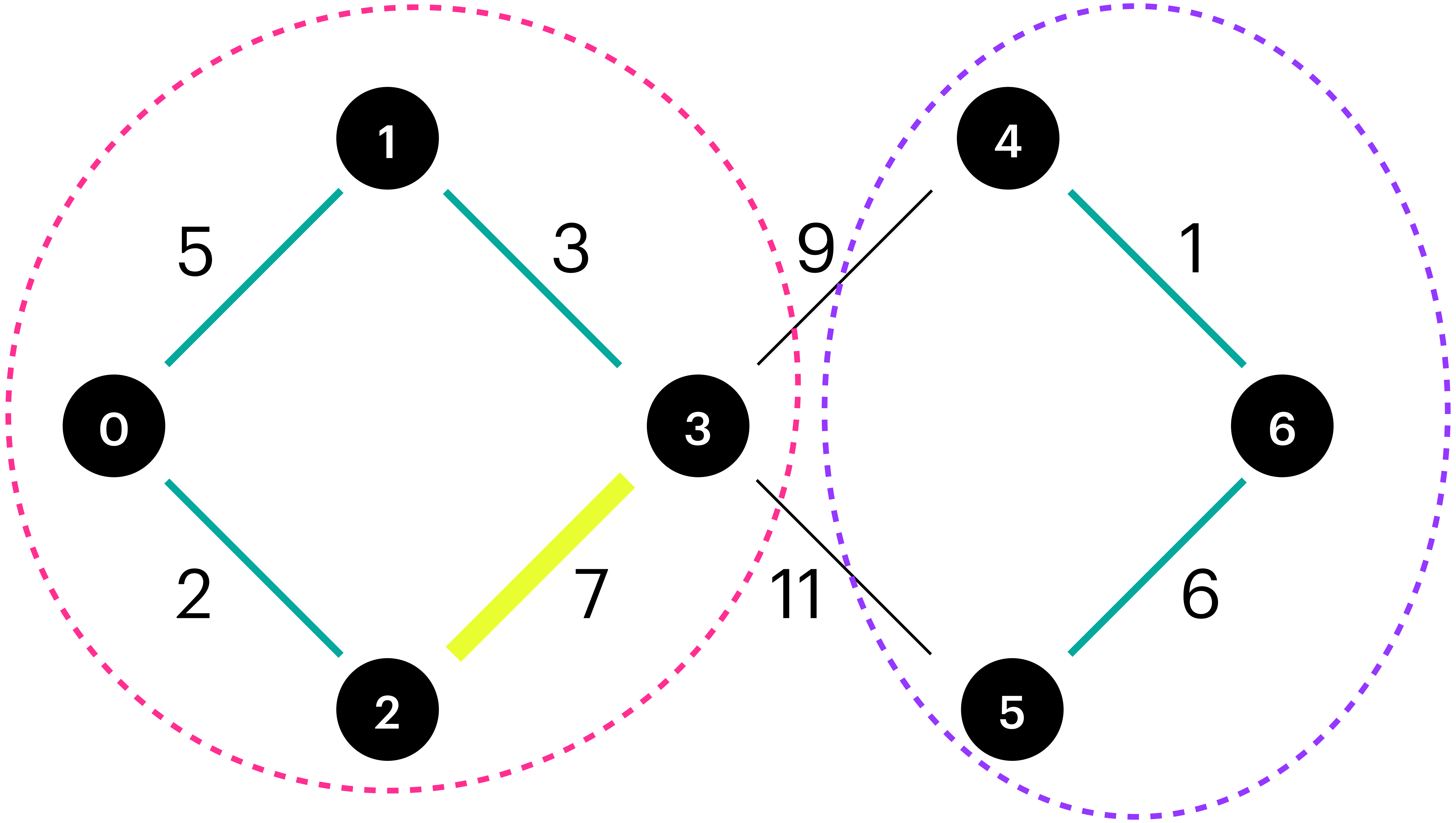
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

1: **Implementierung:**
2: MAKE(V): rep[v] $\leftarrow v$ $\forall v \in V$
3:
4: SAME(u,v): teste ob rep[u] = rep[v]
5:
6: UNION(u,v):
7: **for** $x \in$ members[rep[u]] **do**
8: rep[x] $\leftarrow$ rep[v]
9: members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

1: $F \leftarrow \varnothing$
2: $UF \leftarrow$ MAKE(V)
3: SORT(E)
4: **for** $uv \in E$, aufsteigend sortiert **do**
5: **if** SAME(u,v) = false **then**
6: $F \leftarrow F \cup \{uv\}$
7: UNION(u,v)

F : edges of the MST

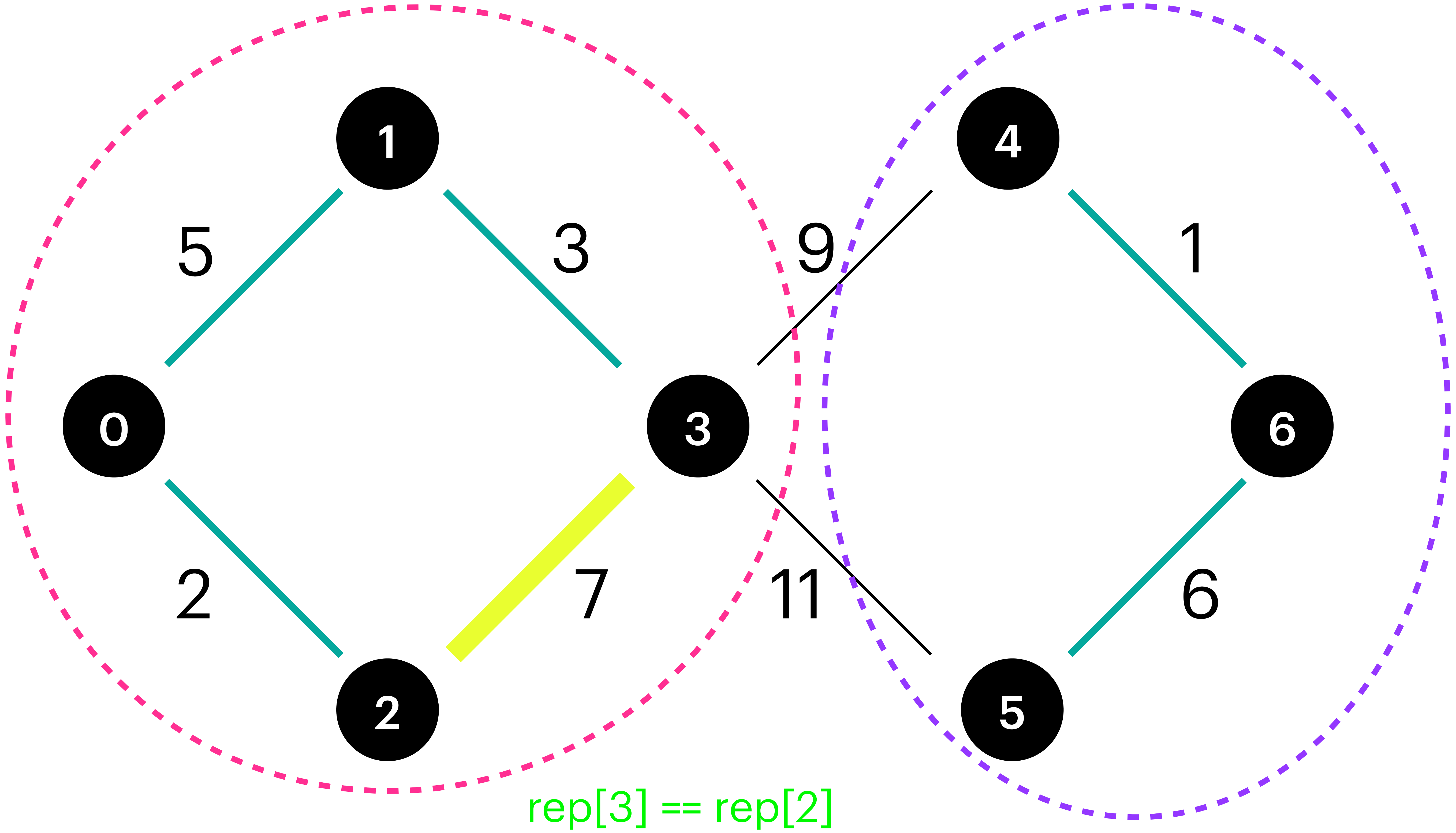
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



rep[3] == rep[2]

SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

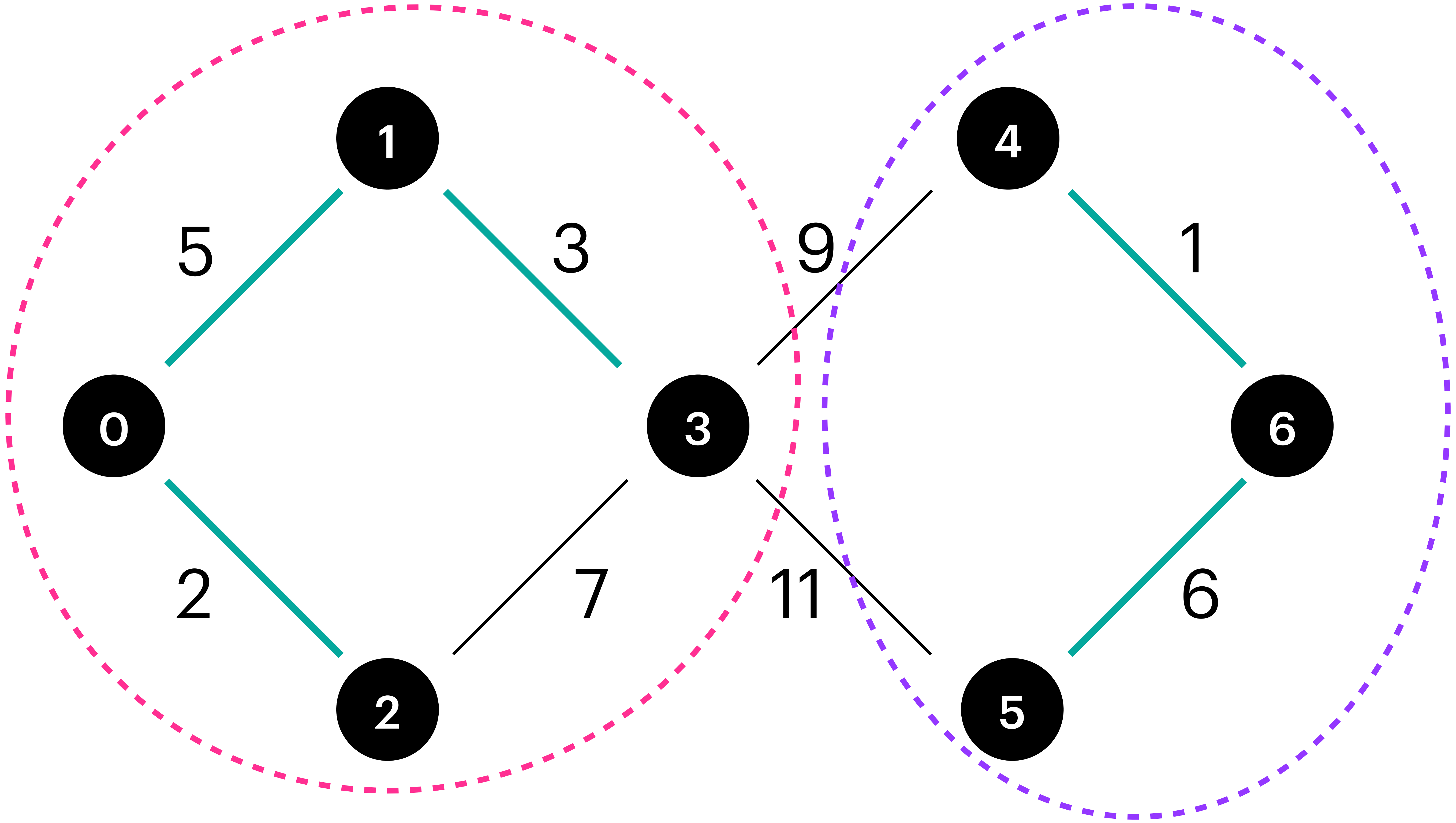
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

Implementierung:
MAKE(V): rep[v] $\leftarrow v \ \forall v \in V$

SAME(u,v): teste ob rep[u] = rep[v]

UNION(u,v):
for $x \in$ members[rep[u]] **do**
  rep[x] $\leftarrow$ rep[v]
  members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

$F \leftarrow \varnothing$
$UF \leftarrow$ MAKE(V)
SORT(E)
for $uv \in E$, aufsteigend sortiert **do**
  if SAME(u,v) = false **then**
    $F \leftarrow F \cup \{uv\}$
    UNION(u,v)

F : edges of the MST

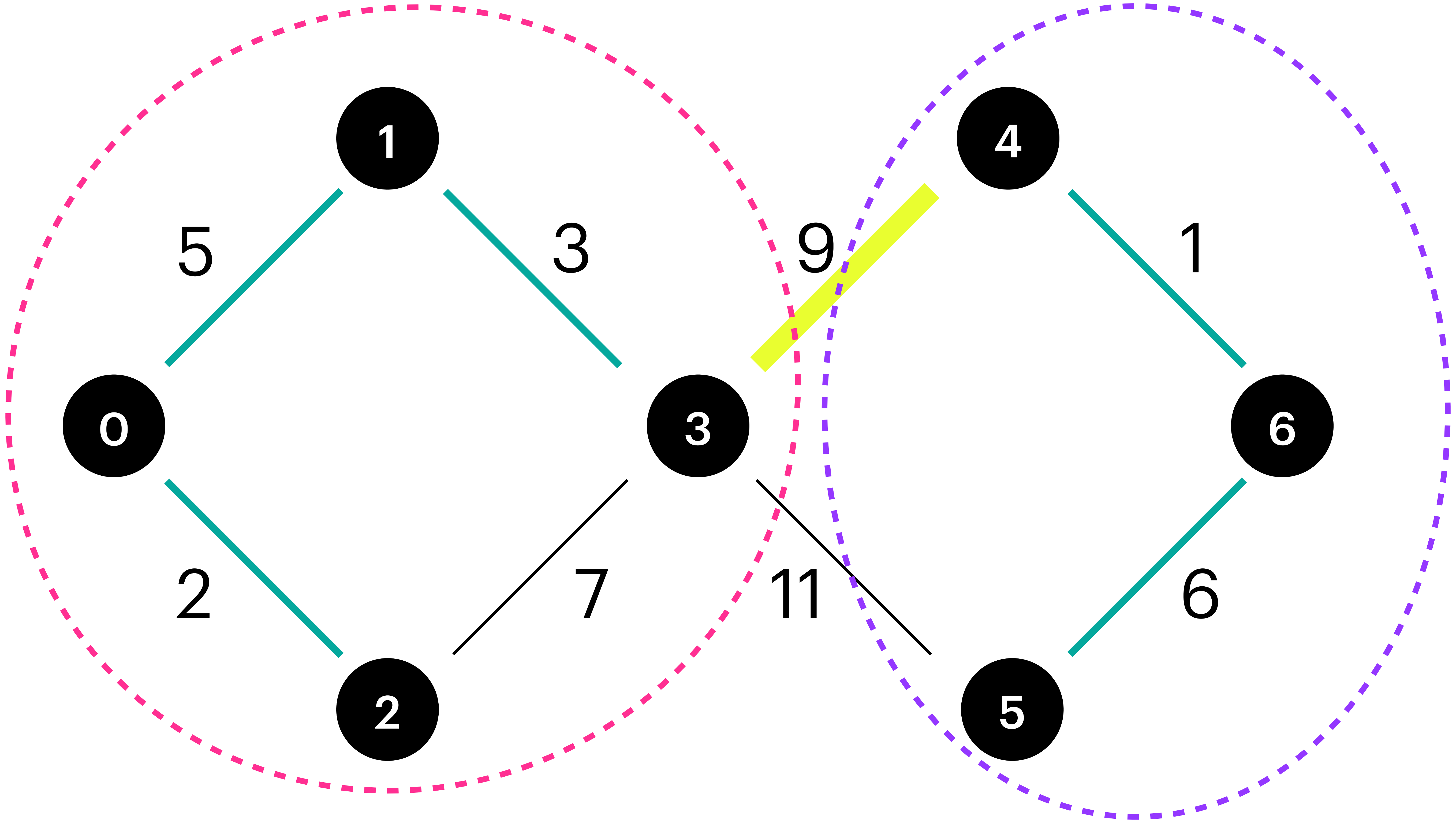
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} }

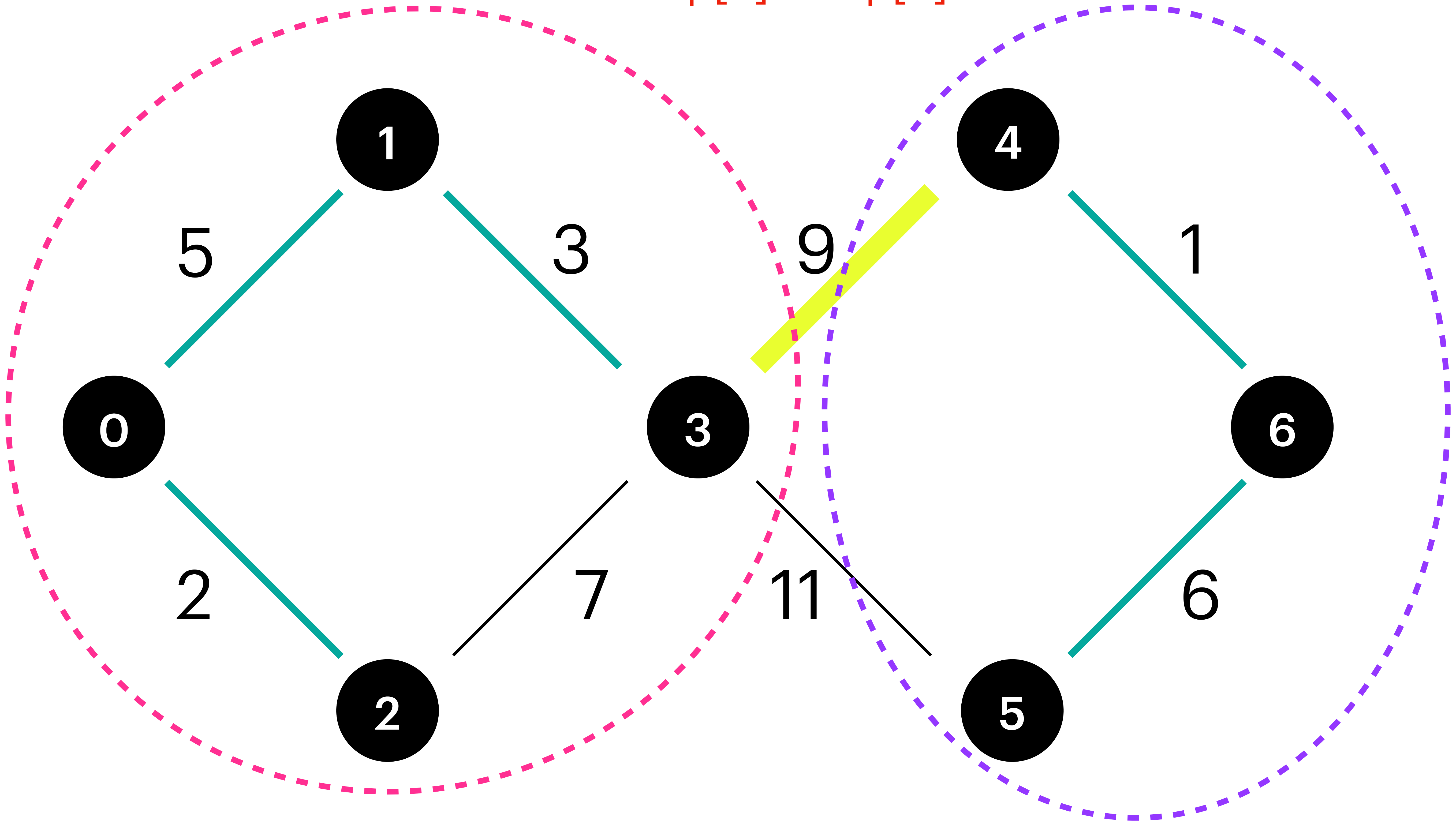
rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }

rep[4] != rep[3]



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {4,3} }

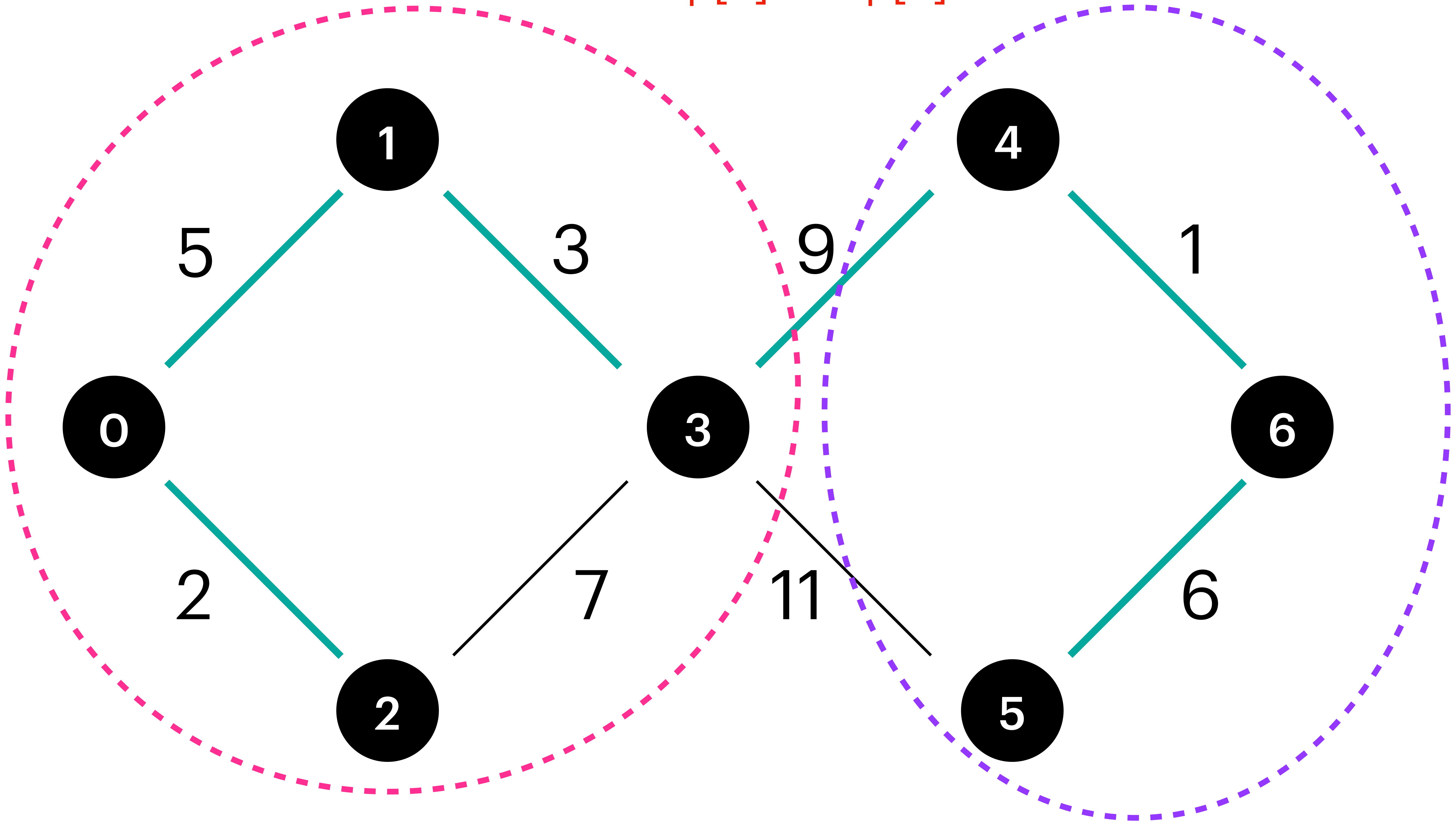
rep[4] != rep[3]

rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

1: **Implementierung:**
2: MAKE(V): rep[v] $\leftarrow v \ \forall v \in V$
3:
4: SAME(u,v): teste ob rep[u] = rep[v]
5:
6: UNION(u,v):
7: **for** $x \in$ members[rep[u]] **do**
8: rep[x] $\leftarrow$ rep[v]
9: members[rep[v]] $\leftarrow$ members[rep[v]] $\cup \{x\}$

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

1: $F \leftarrow \varnothing$
2: $UF \leftarrow$ MAKE(V)
3: SORT(E)
4: **for** $uv \in E$, aufsteigend sortiert **do**
5: **if** SAME(u,v) = false **then**
6: $F \leftarrow F \cup \{uv\}$
7: UNION(u,v)

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {4,3} }

UNION(4,3)

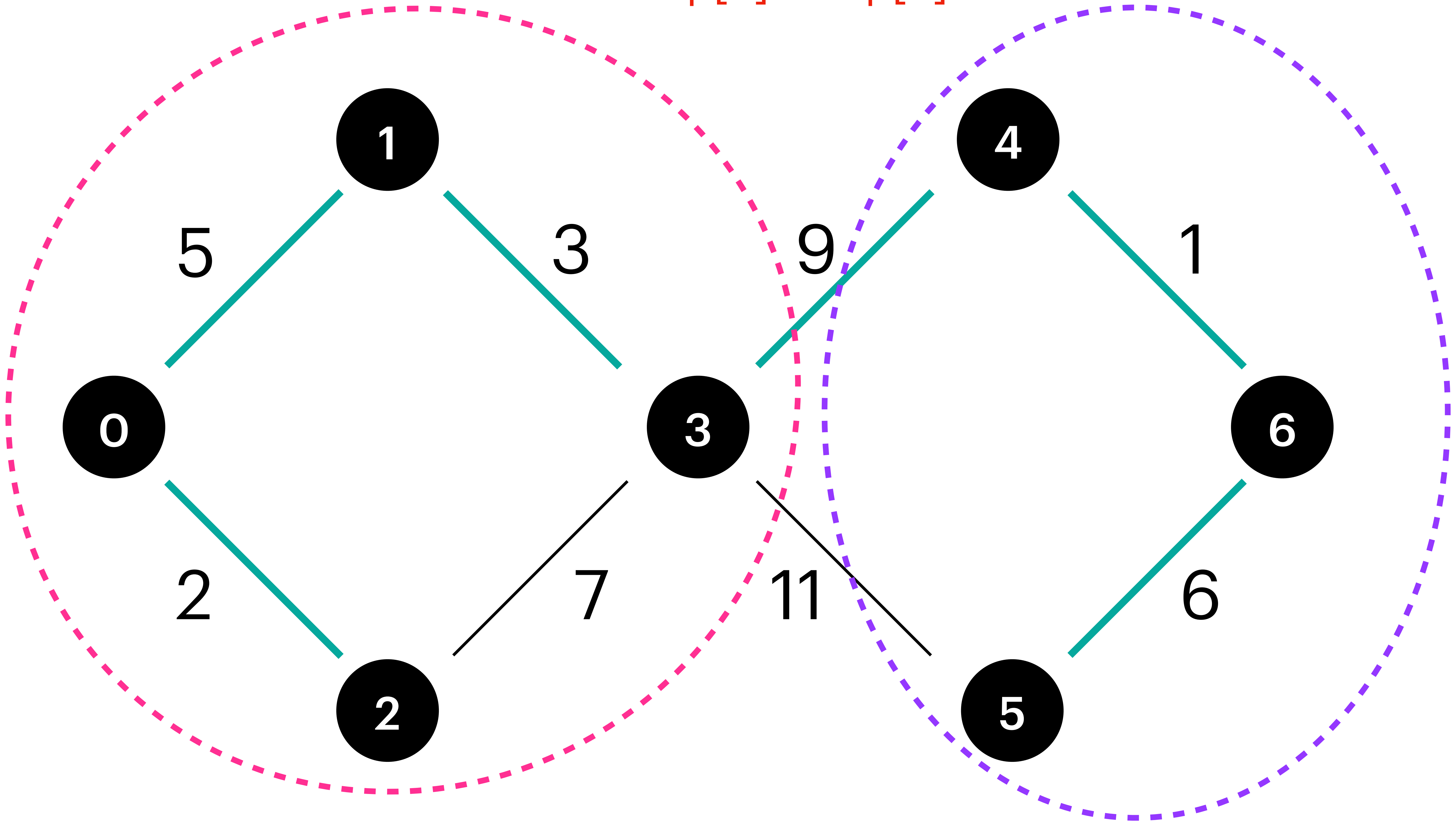
rep[] :

0	1	2	3	4	5	6
0	0	0	0	5	5	5

members[] :

0	{ 0 , 2 , 1 , 3 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }

rep[4] != rep[3]



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {4,3} }

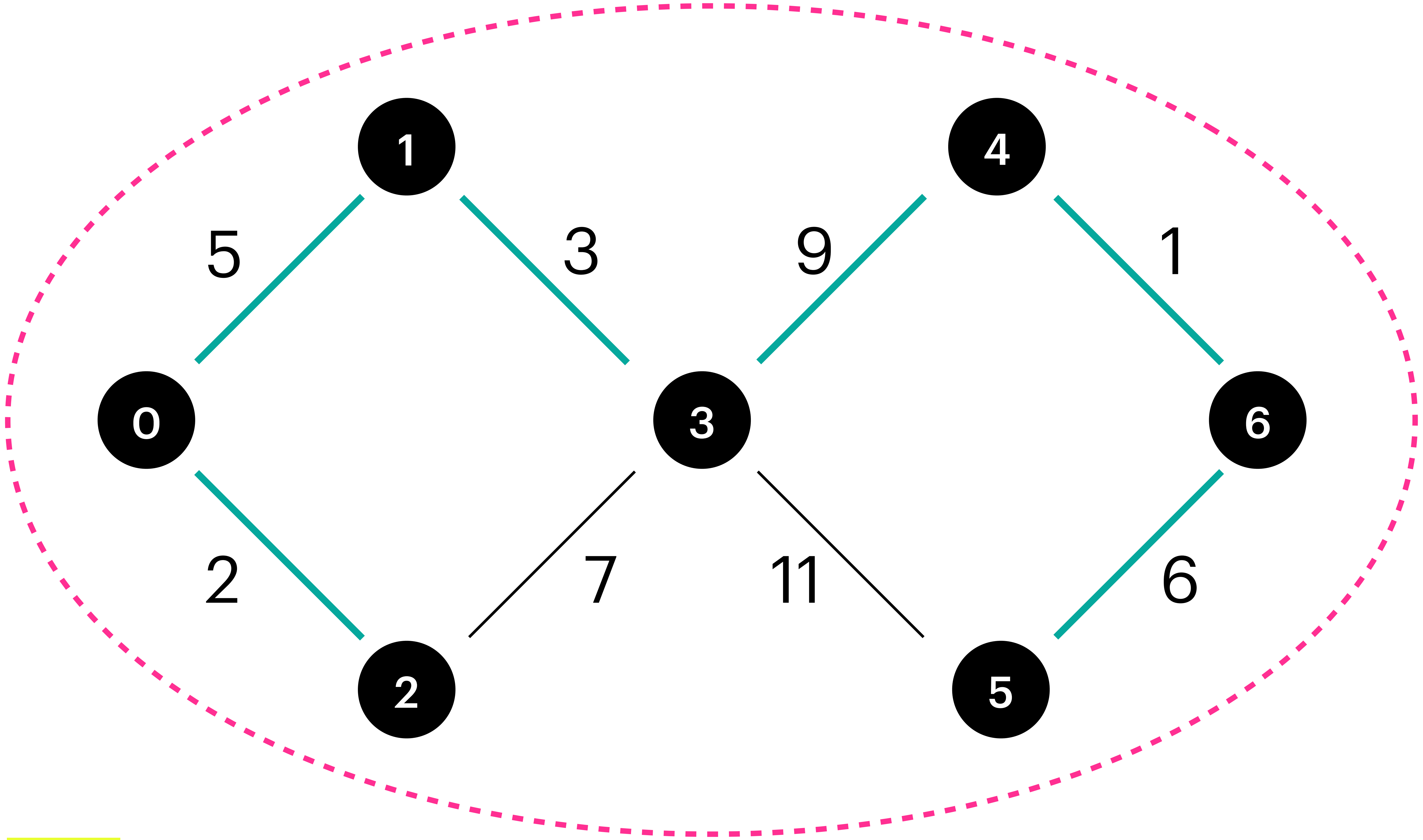
UNION(4,3)

rep[] :

0	1	2	3	4	5	6
0	0	0	0	0	0	0

members[] :

0	{ 0 , 2 , 1 , 3 , 5 , 4 , 6 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):    rep[v] ← v  ∀v ∈ V

SAME(u,v):    teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

F : edges of the MST

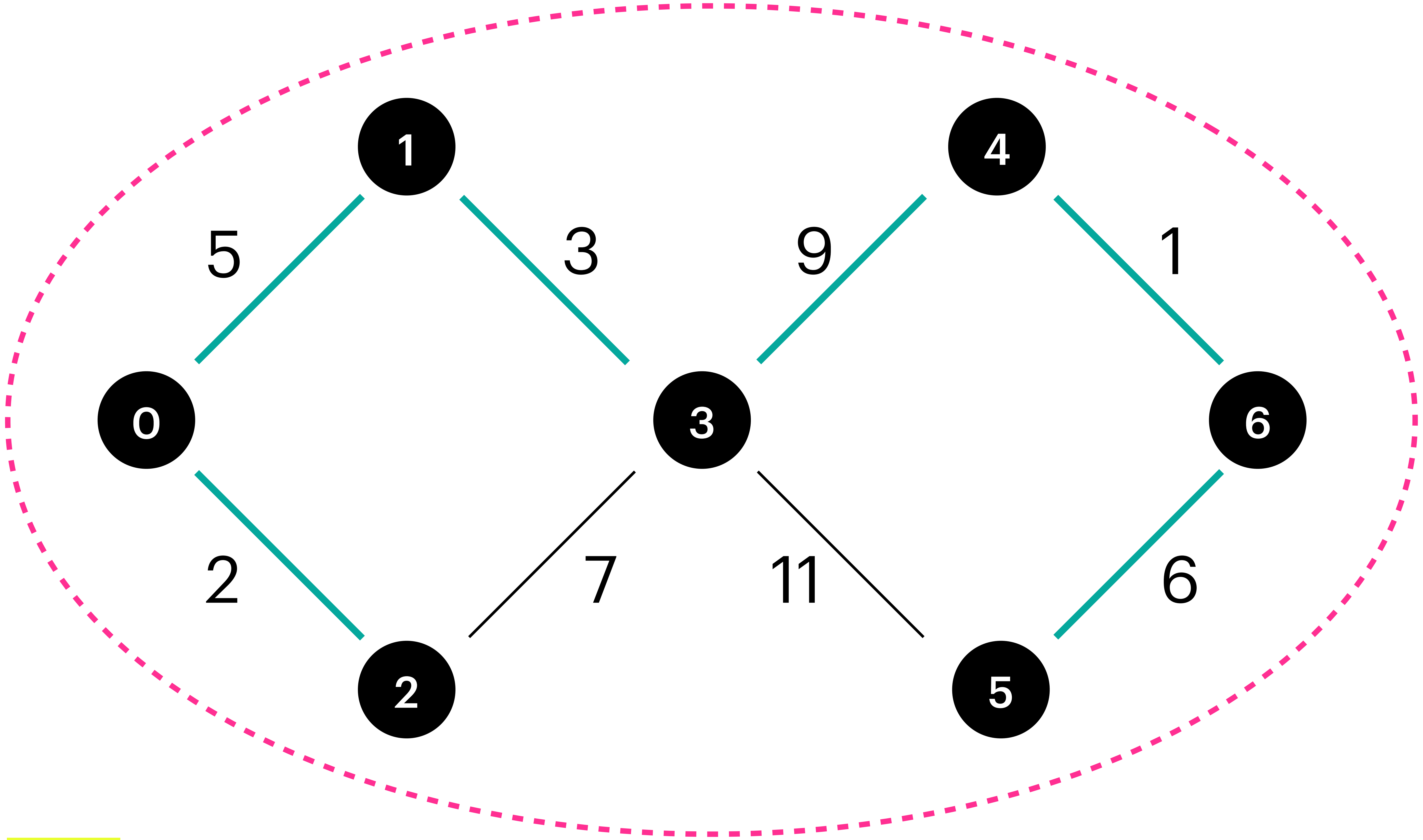
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {4,3} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	0	0	0

members[] :

0	{ 0 , 2 , 1 , 3 , 5 , 4 , 6 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

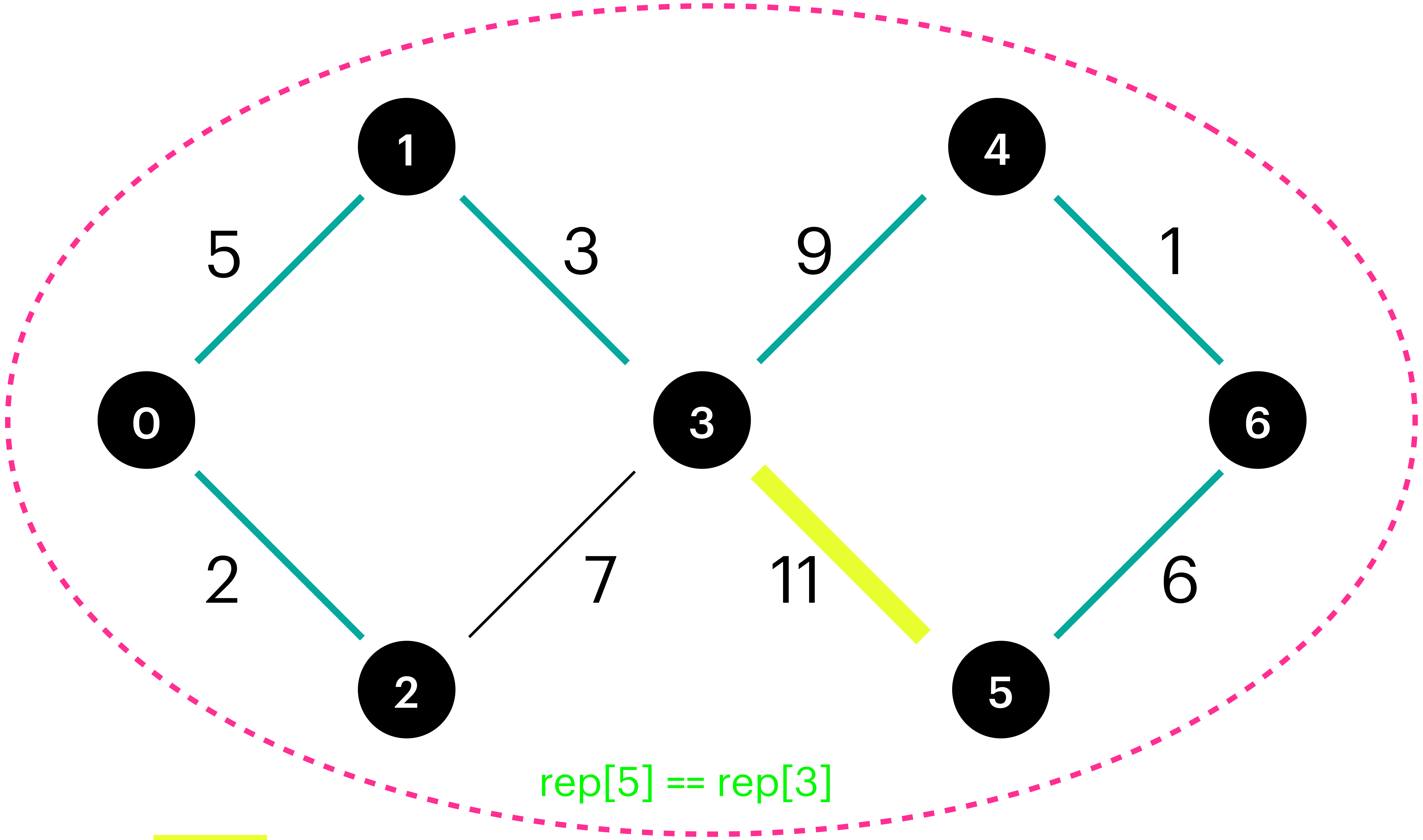
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {4,3} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	0	0	0

members[] :

0	{ 0 , 2 , 1 , 3 , 5 , 4 , 6 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} }

MST
Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)
members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

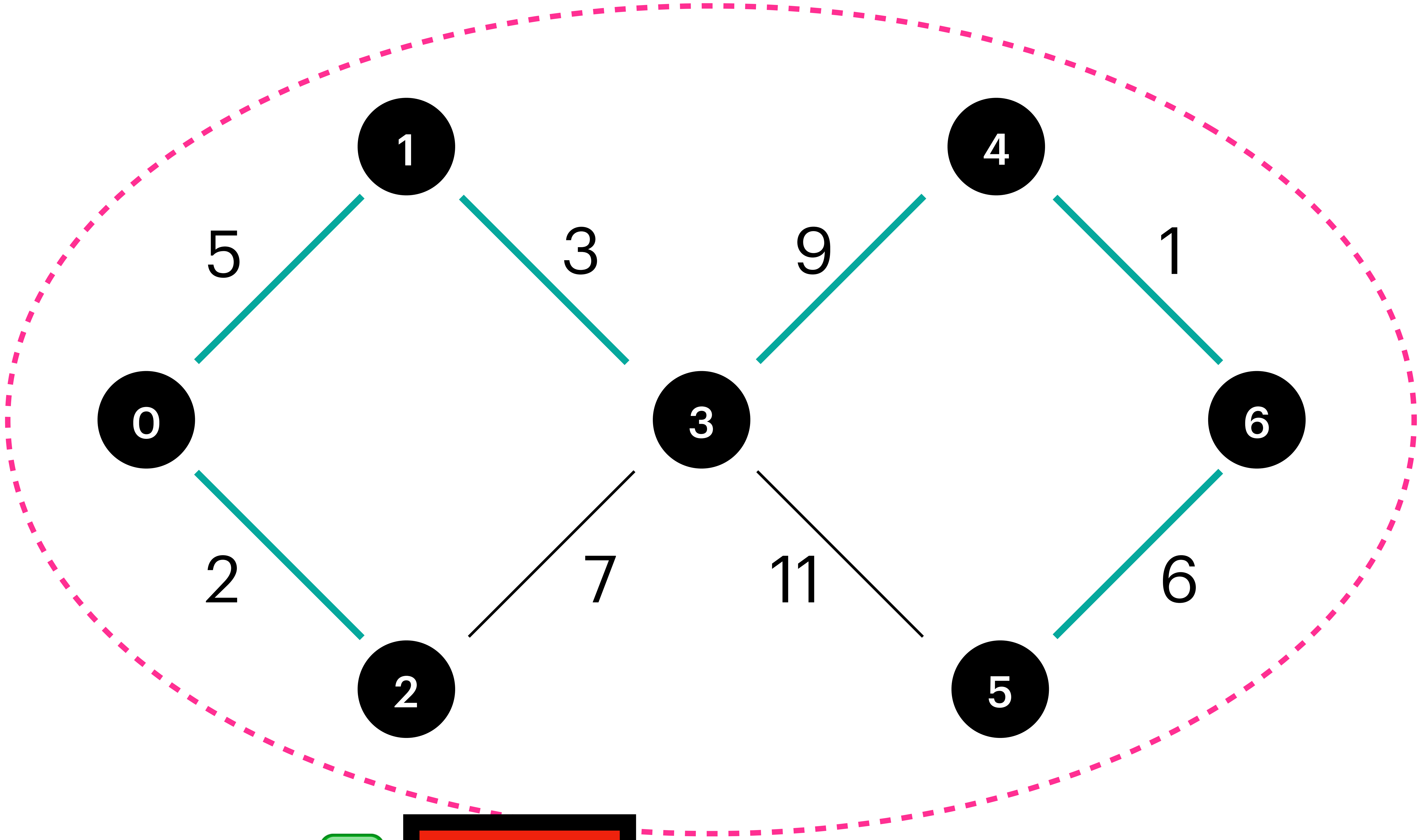
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {4,3} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	0	0	0

members[] :

0	{ 0 , 2 , 1 , 3 , 5 , 4 , 6 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} } ✅

All edges are done

MST

Kruskal's Algorithm

Algorithm 11 Union-Find(G)

```
Implementierung:
MAKE(V):     rep[v] ← v  ∀v ∈ V

SAME(u,v):     teste ob rep[u] = rep[v]

UNION(u,v):
for x ∈ members[rep[u]] do
    rep[x] ← rep[v]
    members[rep[v]] ← members[rep[v]] ∪ {x}
```

rep[v] : unique representative of ConComp(v)

members[rep[v]] : list of the nodes in ConComp(rep[v])

Algorithm 12 Kruskal(G) (mit UF-Datenstruktur)

```
F ← ∅
UF ← MAKE(V)
SORT(E)
for uv ∈ E, aufsteigend sortiert do
    if SAME(u,v) = false then
        F ← F ∪ {uv}
        UNION(u,v)
```

F : edges of the MST

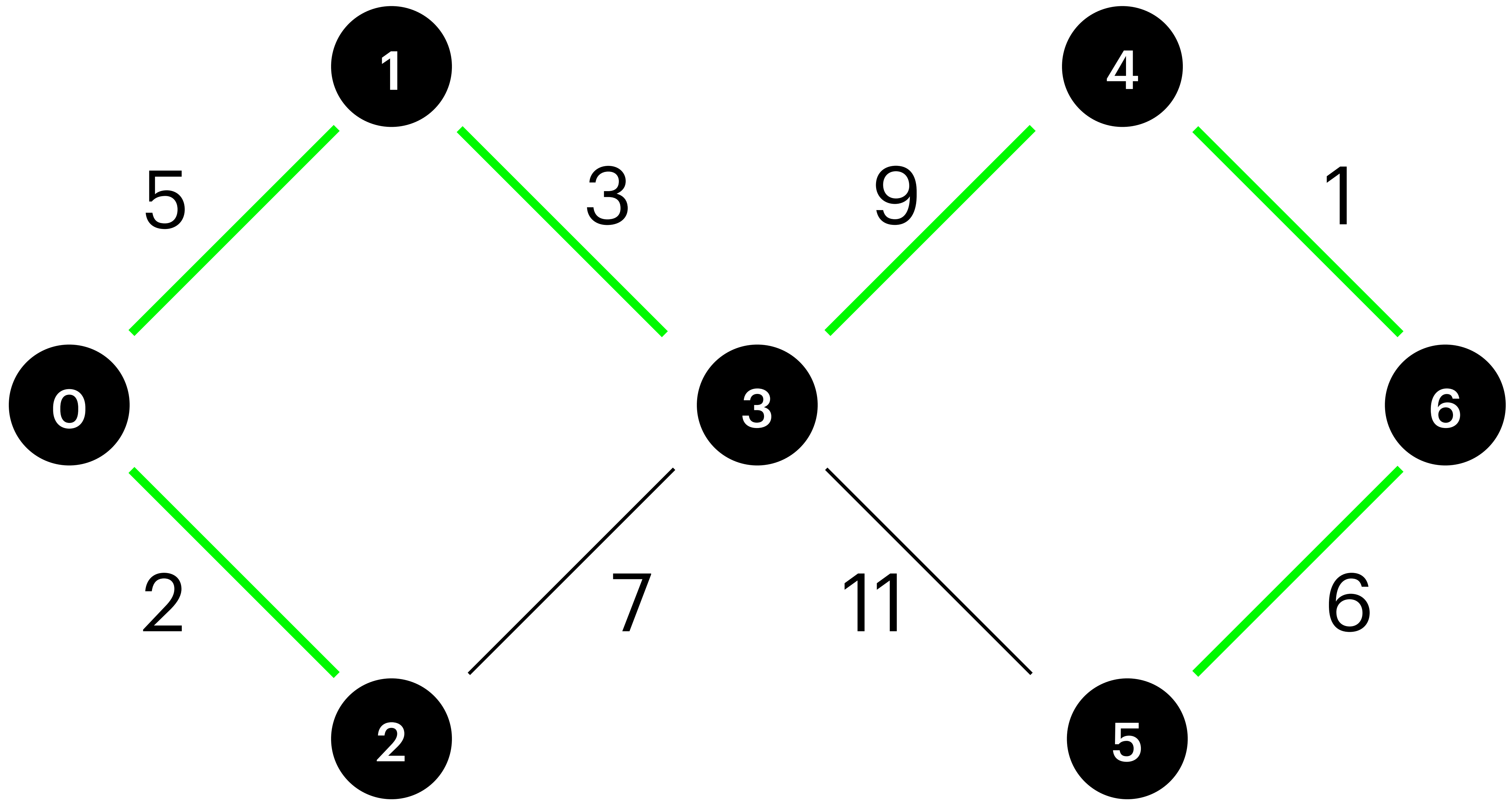
F : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {4,3} }

rep[] :

0	1	2	3	4	5	6
0	0	0	0	0	0	0

members[] :

0	{ 0 , 2 , 1 , 3 , 5 , 4 , 6 }
1	{ 1 , 3 }
2	{ 2 }
3	{ 3 }
4	{ 4 , 6 }
5	{ 5 , 4 , 6 }
6	{ 6 }



SORT(E) : { {6,4} , {2,0} , {3,1} , {1,0} , {6,5} , {3,2} , {4,3} , {5,3} } ✅

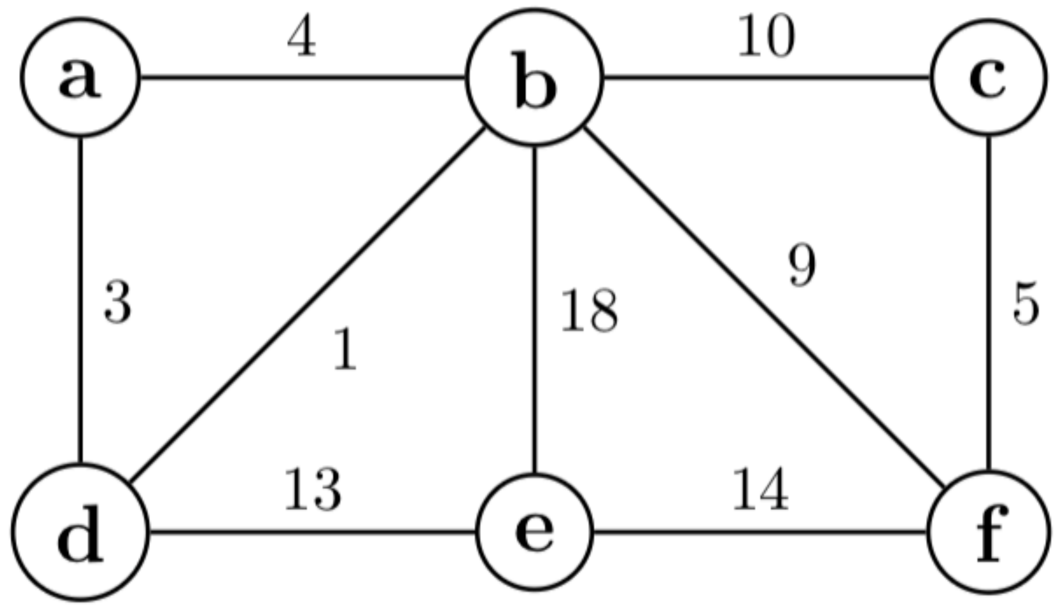
All edges are done

MST

Exam Question

/ 2 P

e) *Minimum Spanning Tree:* Consider the following graph:



i) Highlight the edges that are part of the minimum spanning tree. (Either in the picture above, or you can recreate the graph below).

ii) Write out all positive integers x such that if we replace the weight 1 of edge $\{b, d\}$ in the above graph with x, then edge $\{b, d\}$ would be in at least one minimum spanning tree of the resulting graph.

My To-Do List

Remaining things from sessions

- Graph Sets Code Expert
- Graph Modelling exam question

- Exercise Sheet Corrections
- Quiz Templates

Last Weeks
Organization

- Extra session on friday 13 Dec 14:15 - 17:00 , **CAB H52**
  - All-to-all paths
  - Exercise Correction
- Last session on monday 16 Dec
  - Exam Preparation Session
    - Exam tipps, lernphase tipps, mock exam
  - Recap topics
    - Let me know your specific topics till 23:59 today !
  - Additional things let me know till 23:59 today

Please fill the poll !!!

Questions

Feedbacks , Recommendations

Nil Ozer