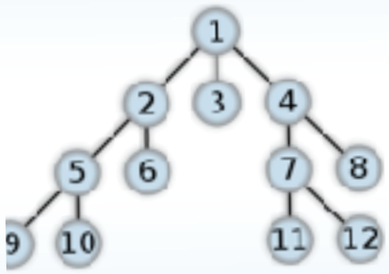


Nachtrag Kapitel 3: Adjazenzliste und Graphenscan

Algorithmen und Datenstrukturen
WS 2020/21

Prof. Dr. Sándor Fekete

Algorithmus 3.7

INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w ∈ V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w ∈ V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP

Algorithmus 3.7

INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w ∈ V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w ∈ V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP

Algorithmus 3.7

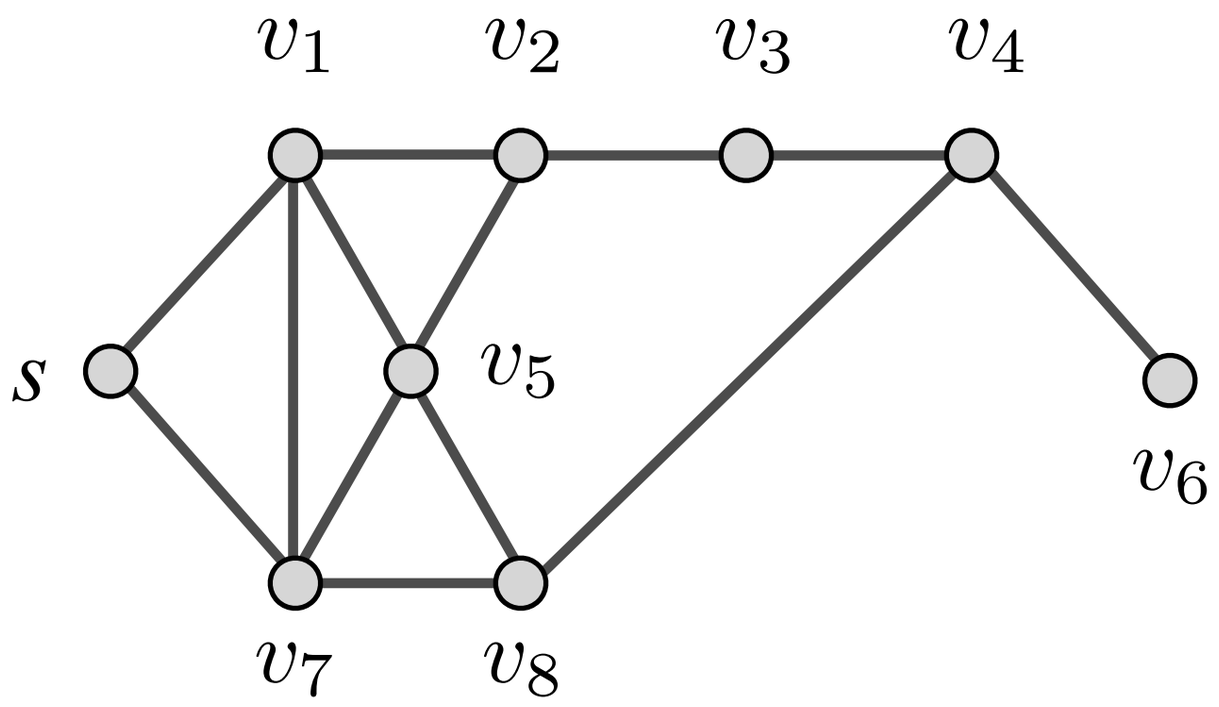
INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP



Algorithmus 3.7

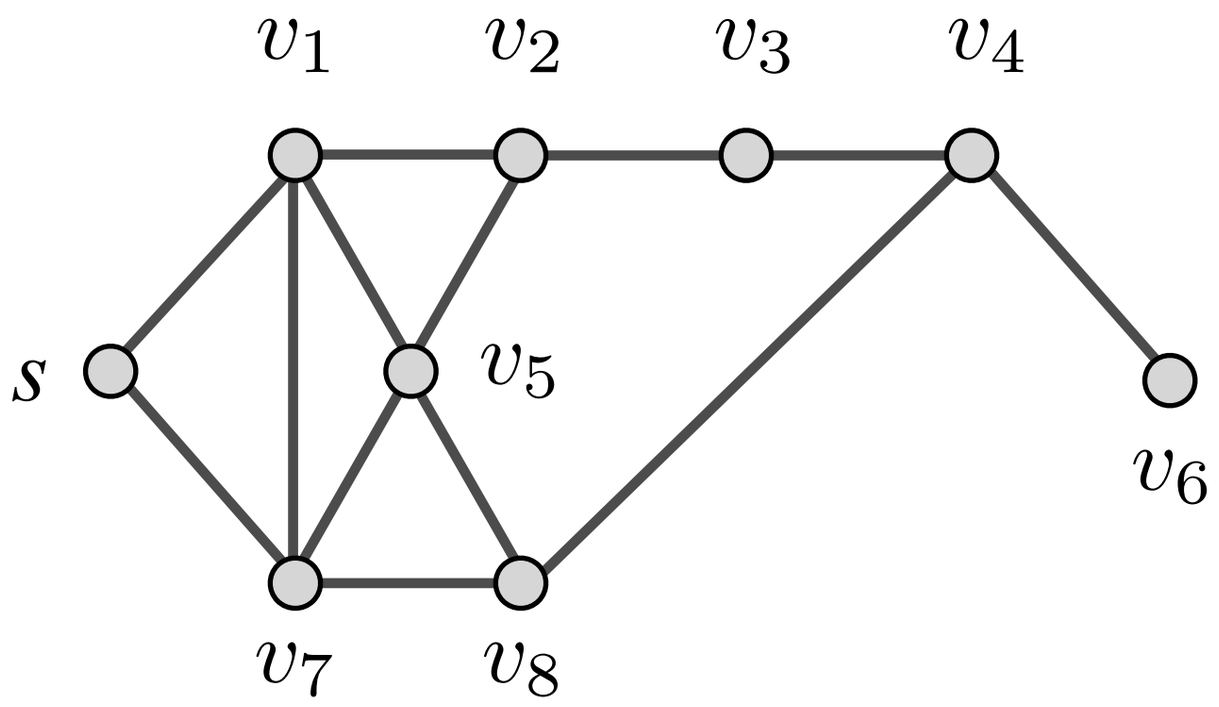
INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP



Algorithmus 3.7

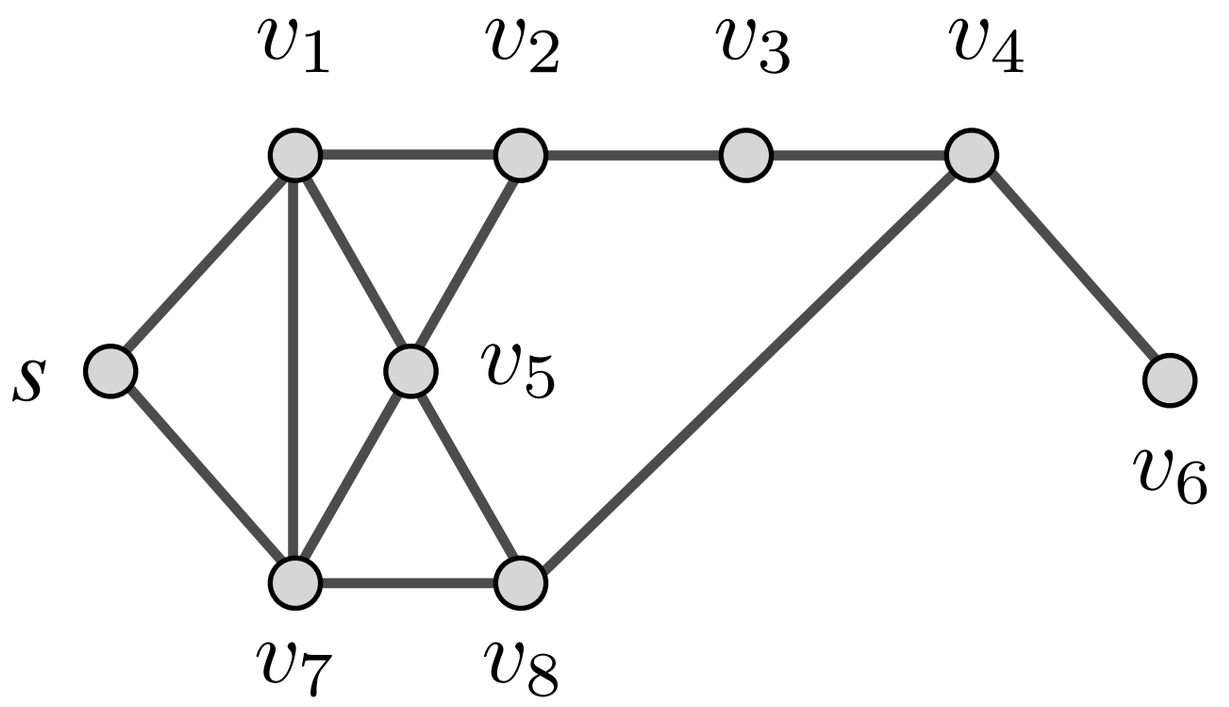
INPUT: Graph G = (V,E), Kraten s

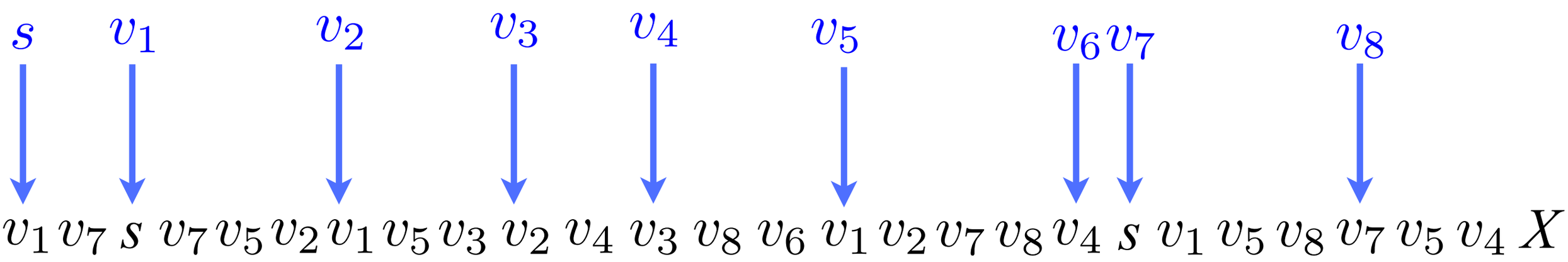
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

INPUT: Graph G = (V,E), Kraten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

Sei R:={s}, Y:={s}, T:=∅

2. WHILE (R≠∅) DO {

   2.1. Wähle v∈ R

   2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN

      2.2.1. R:=R\{v}

   2.3. ELSE {

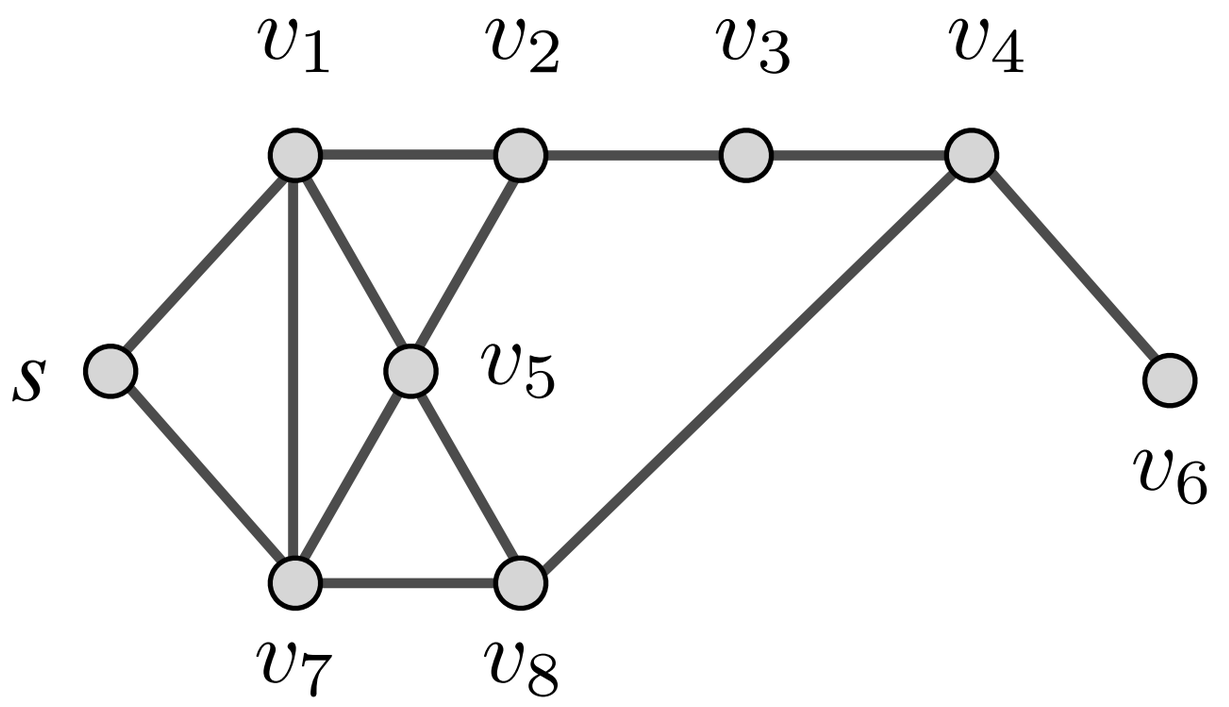
      2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E

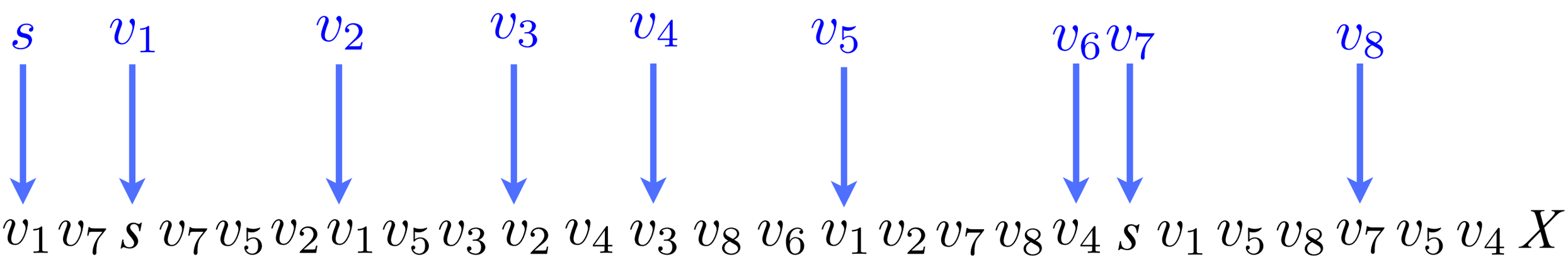
      2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}

      }

   }

3. STOP





Algorithmus 3.7

INPUT: Graph G = (V,E), Kraten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

Sei R:={s}, Y:={s}, T:=∅

2. WHILE (R≠∅) DO {

  2.1. Wähle v∈ R

  2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN

    2.2.1. R:=R\{v}

  2.3. ELSE {

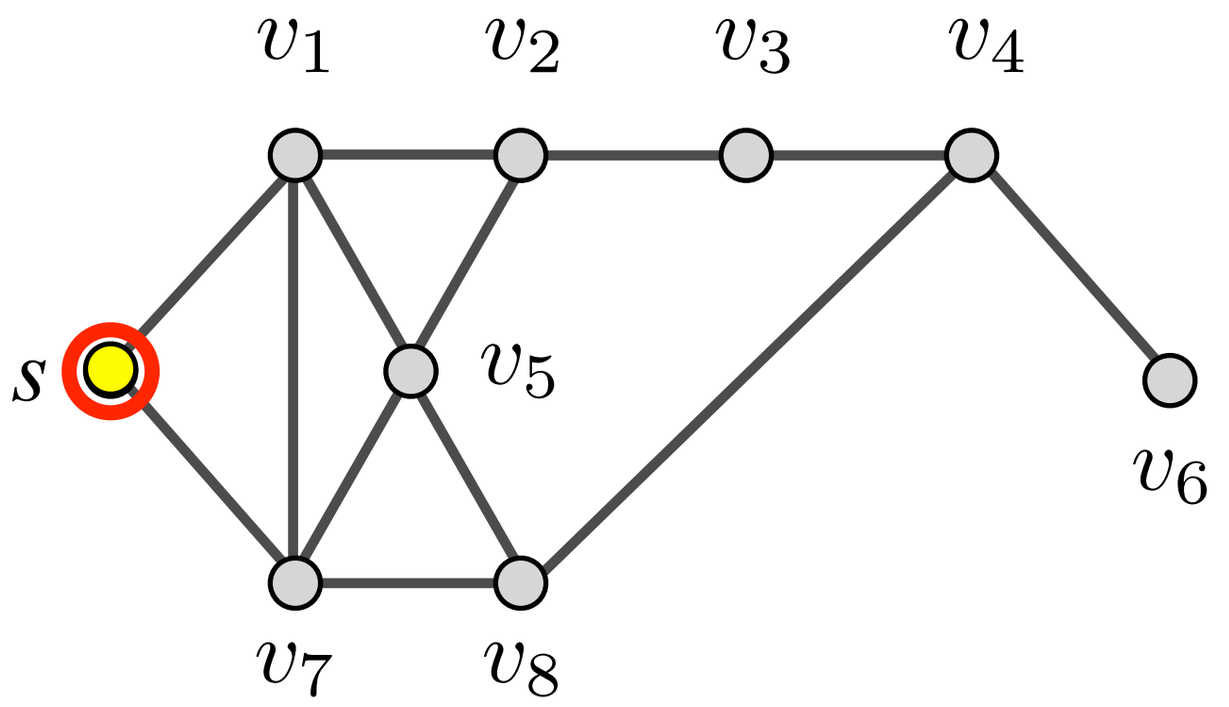
    2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E

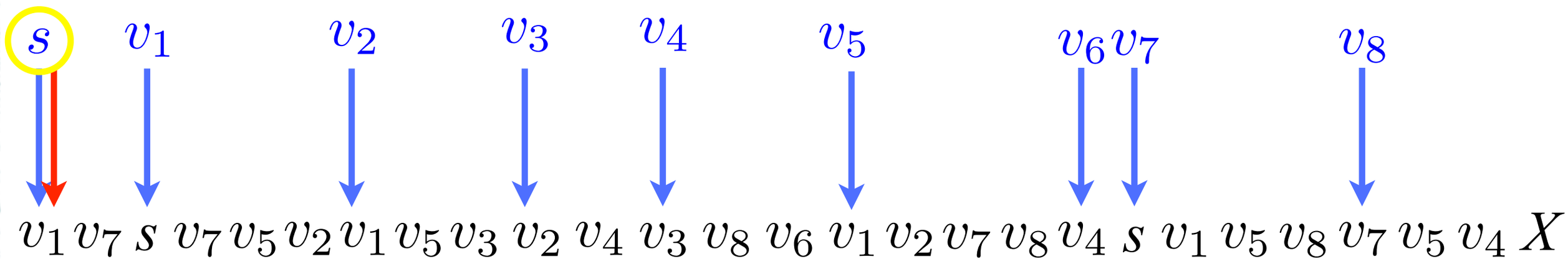
    2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}

    }

  }

3. STOP





Algorithmus 3.7

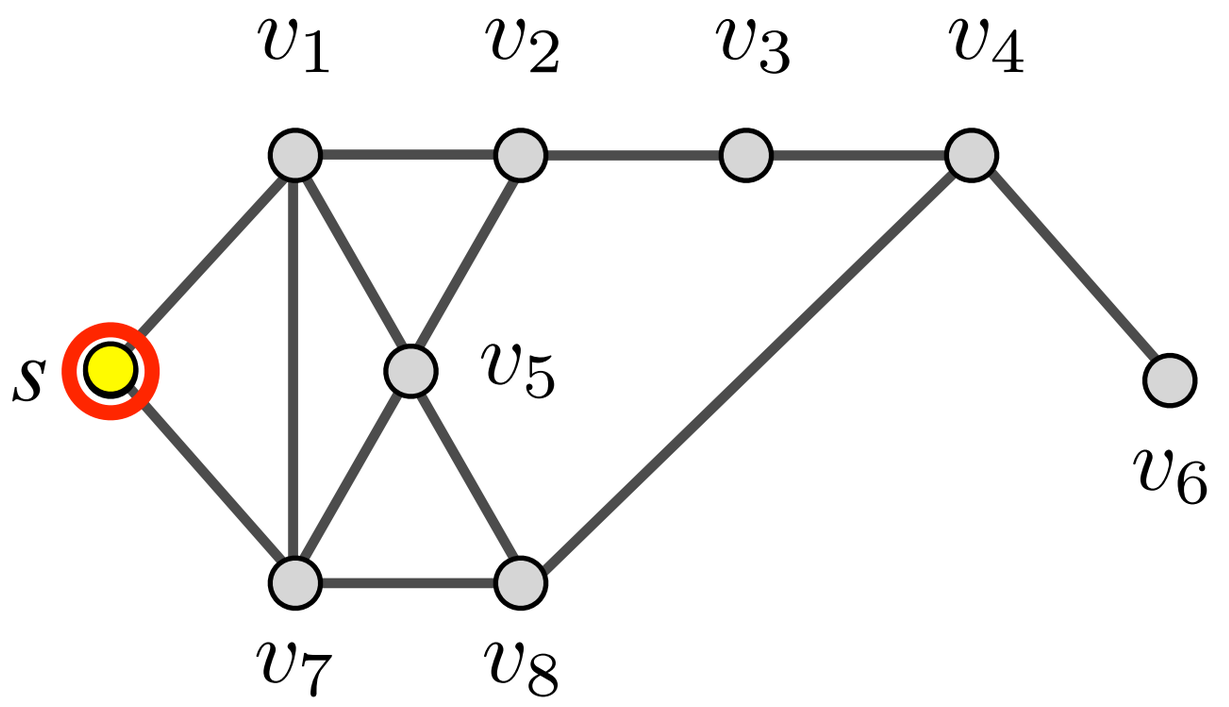
INPUT: Graph G = (V,E), Kraten s

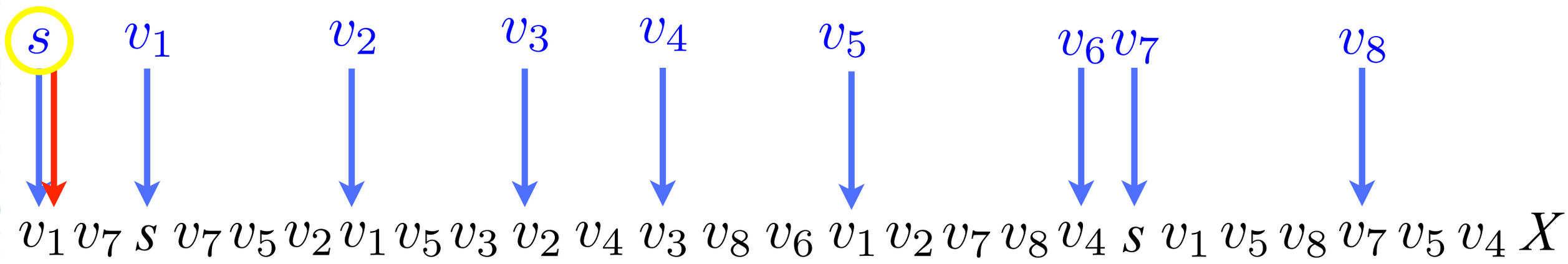
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

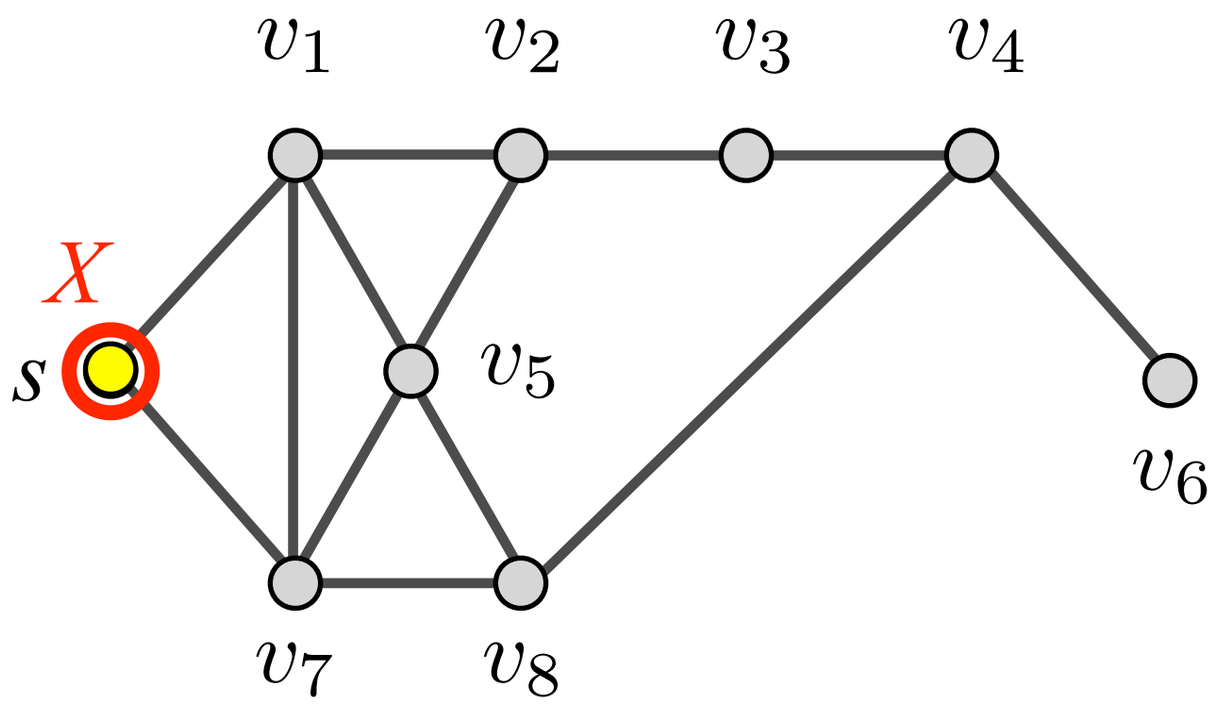
INPUT: Graph G = (V,E), Kraten s

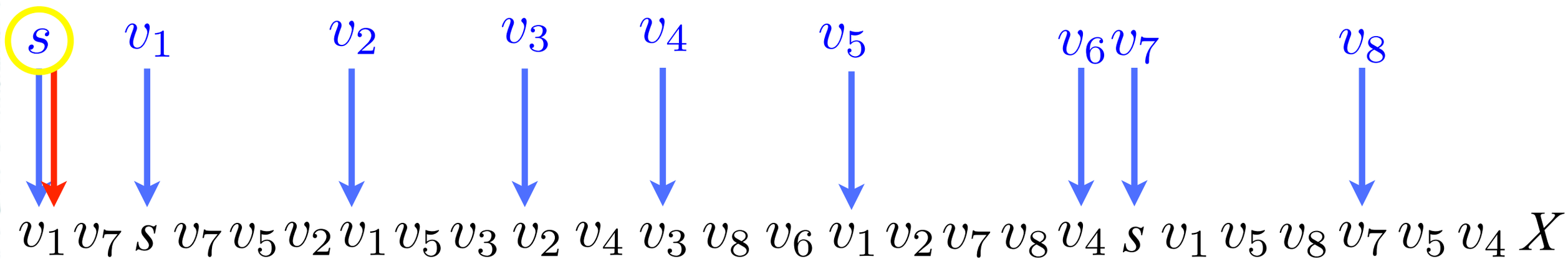
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

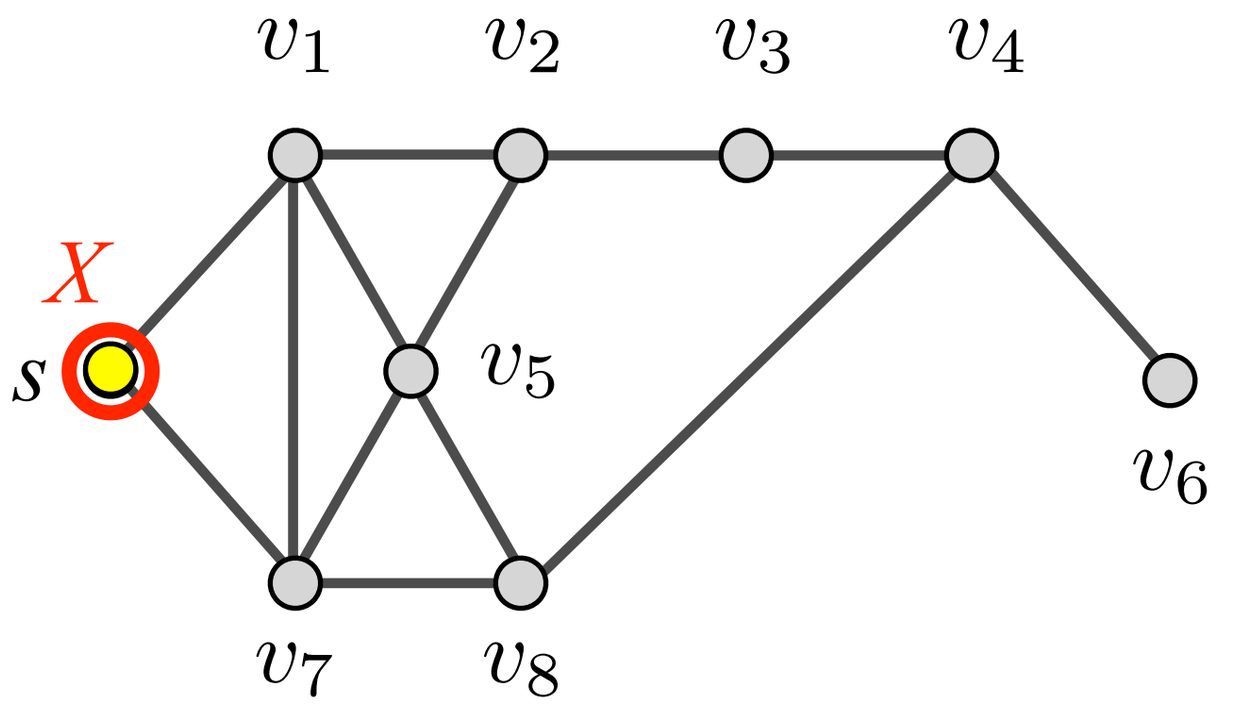
INPUT: Graph G = (V,E), Kraten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP



s v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

$v_1\ v_7\ s\ v_7\ v_5\ v_2\ v_1\ v_5\ v_3\ v_2\ v_4\ v_3\ v_8\ v_6\ v_1\ v_2\ v_7\ v_8\ v_4\ s\ v_1\ v_5\ v_8\ v_7\ v_5\ v_4\ X$

Algorithmus 3.7

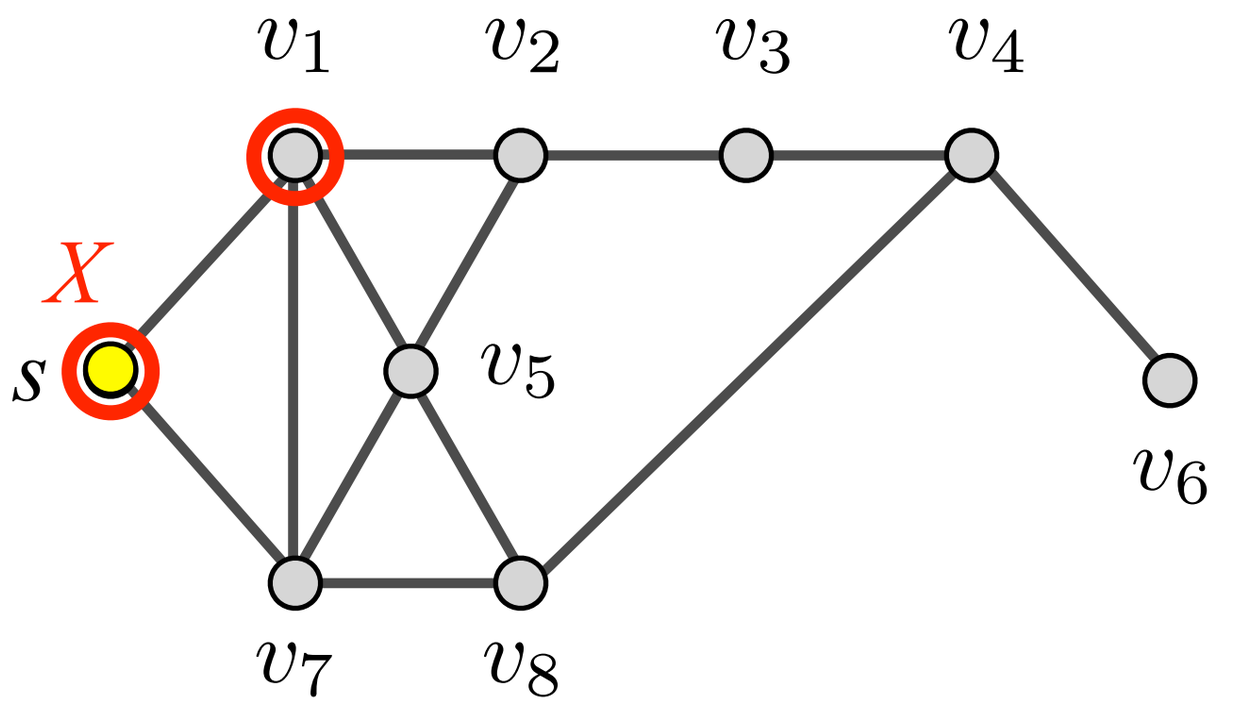
INPUT: Graph G = (V,E), Kraten s

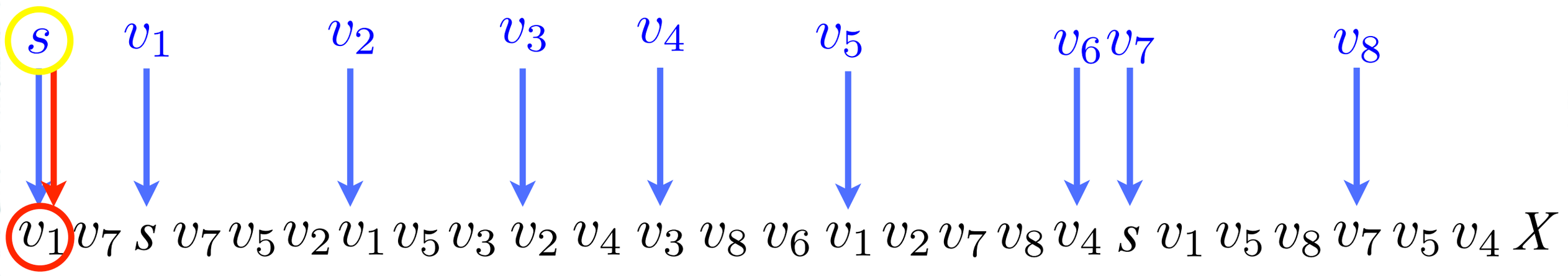
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

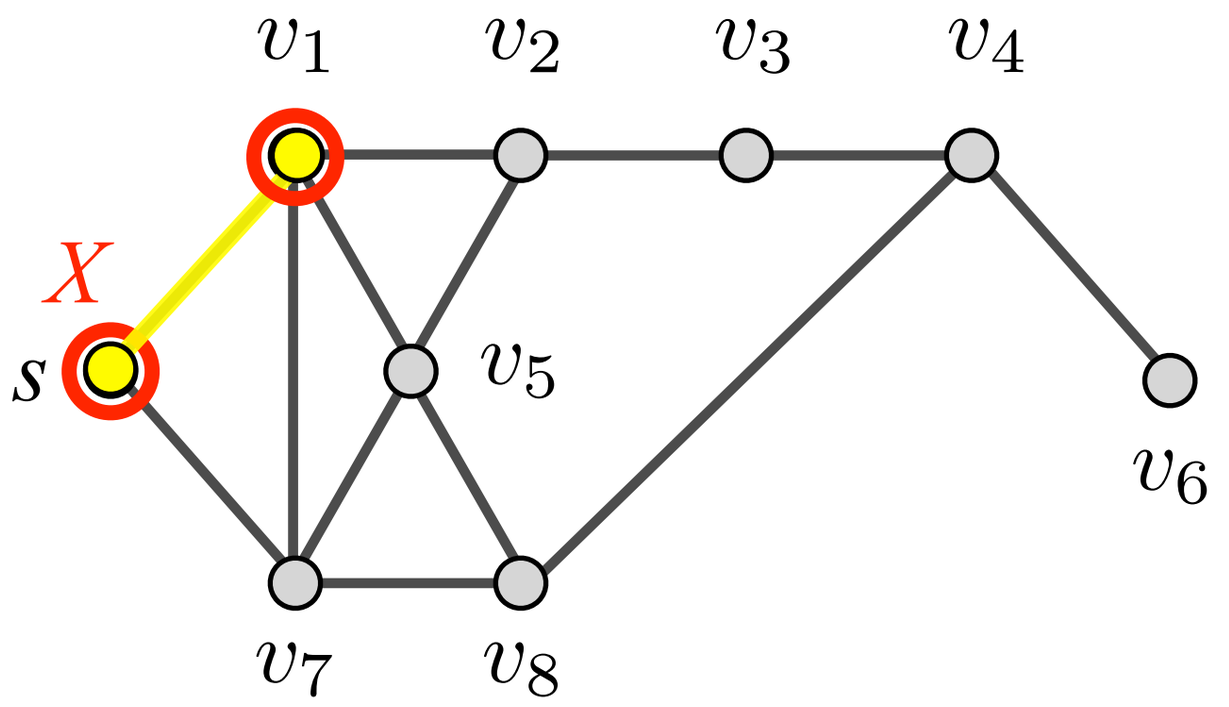
INPUT: Graph G = (V,E), Kraten s

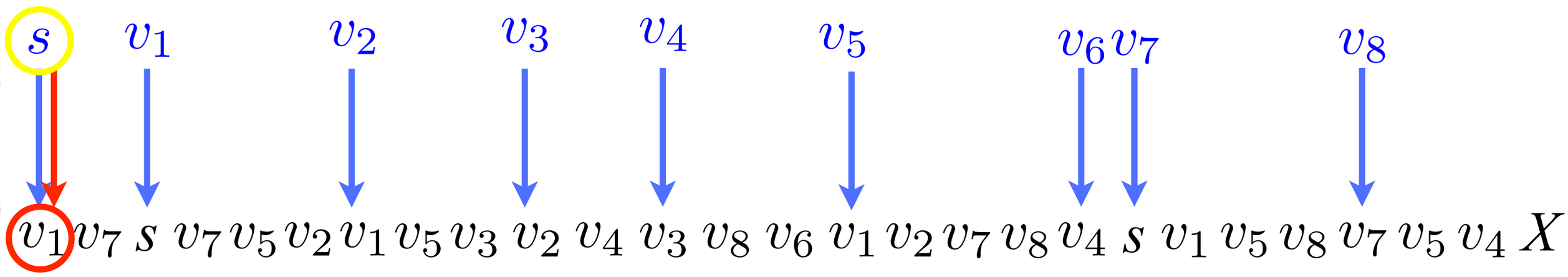
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

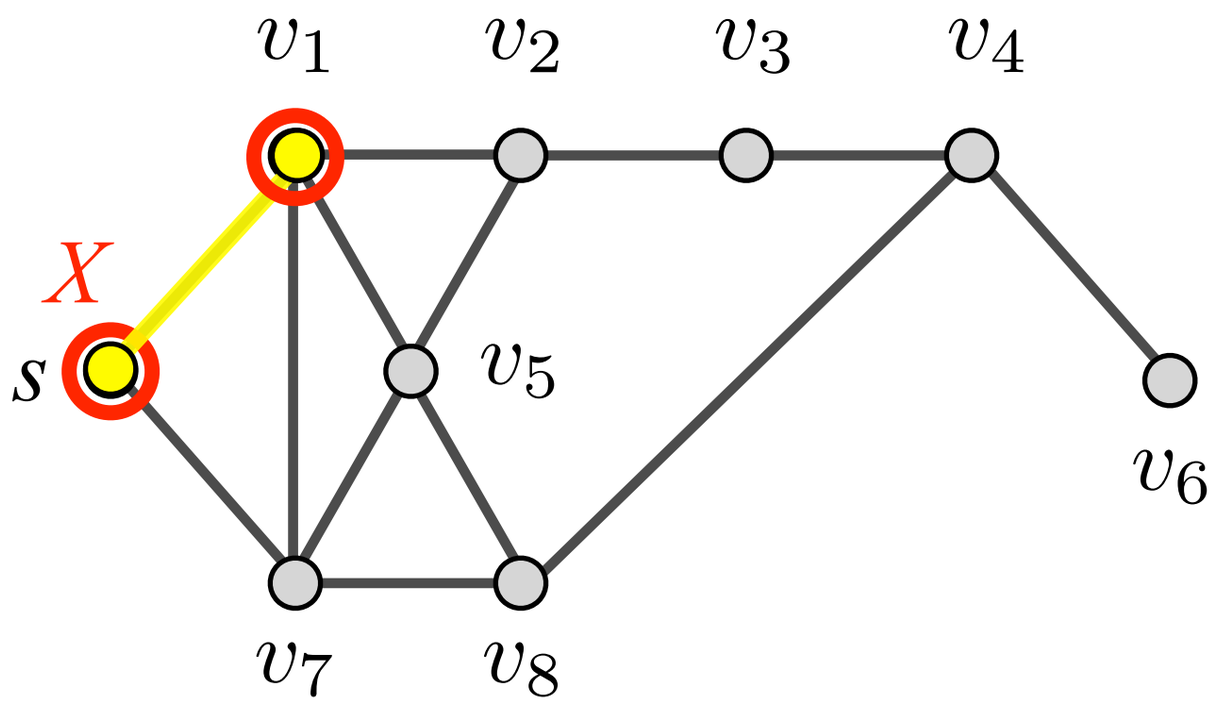
INPUT: Graph G = (V,E), Kraten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP



s v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8

$v_1\, v_7\, s\, v_7\, v_5\, v_2\, v_1\, v_5\, v_3\, v_2\, v_4\, v_3\, v_8\, v_6\, v_1\, v_2\, v_7\, v_8\, v_4\, s\, v_1\, v_5\, v_8\, v_7\, v_5\, v_4\, X$

Algorithmus 3.7

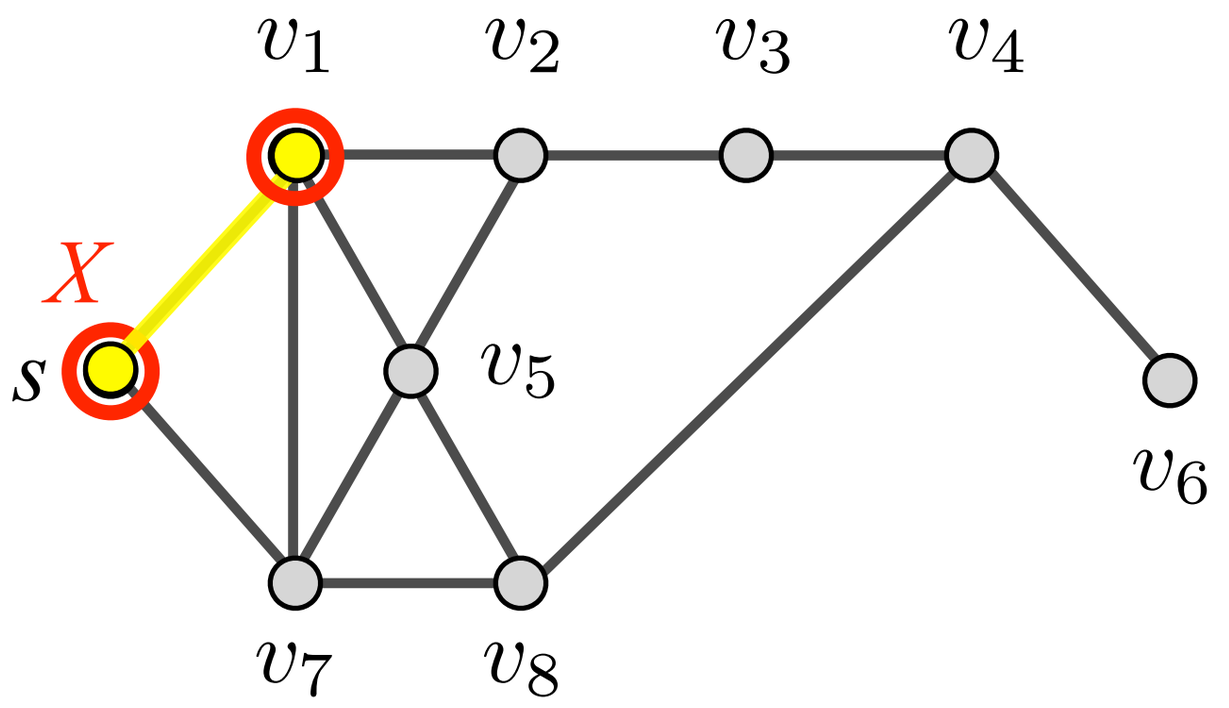
INPUT: Graph G = (V,E), Kraten s

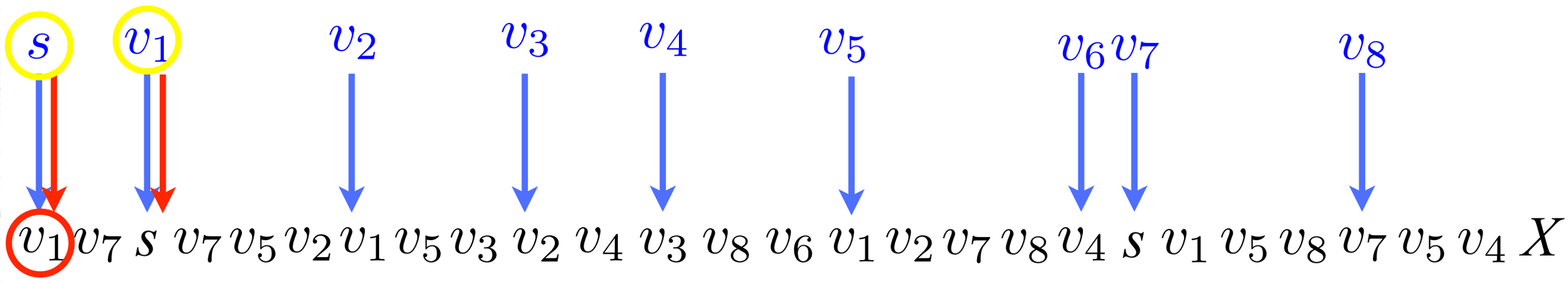
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

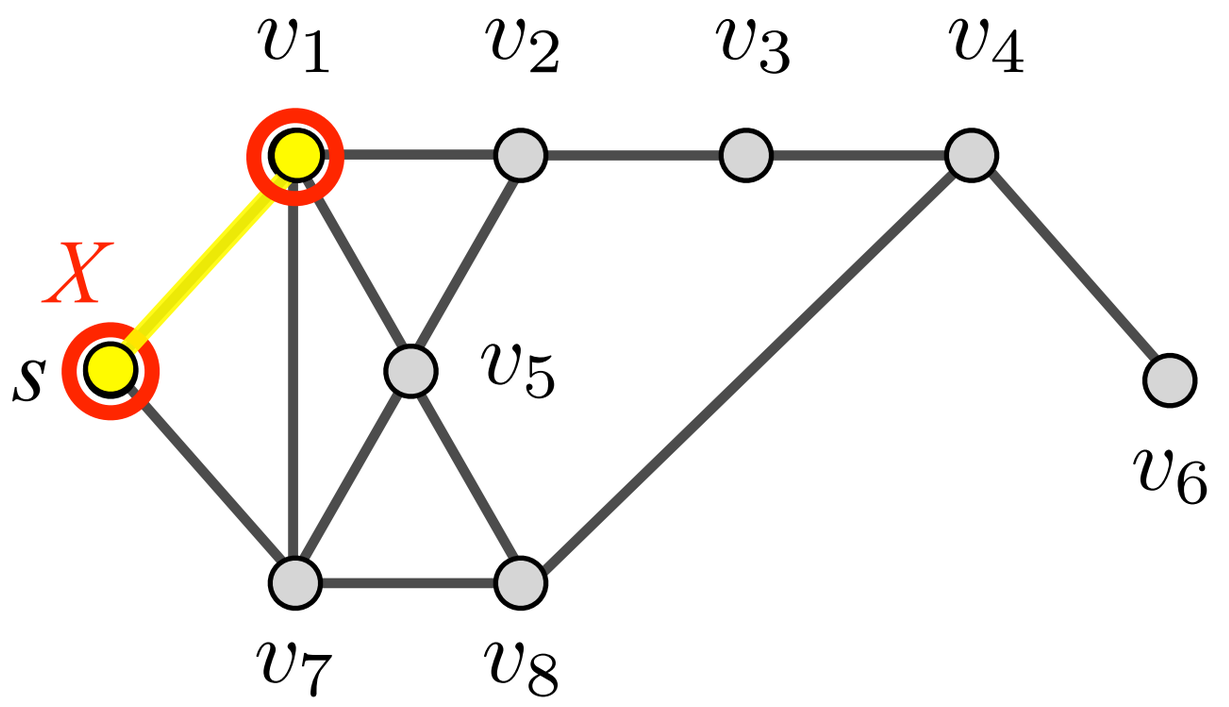
INPUT: Graph G = (V,E), Kraten s

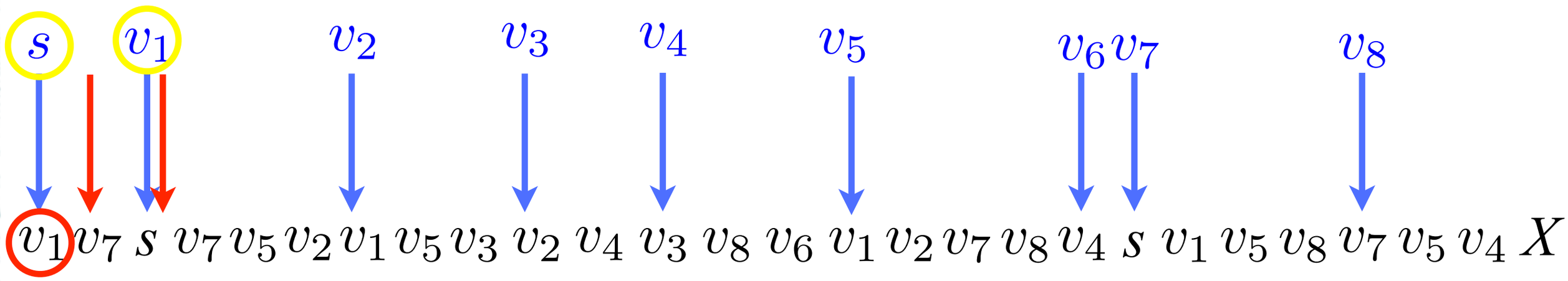
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

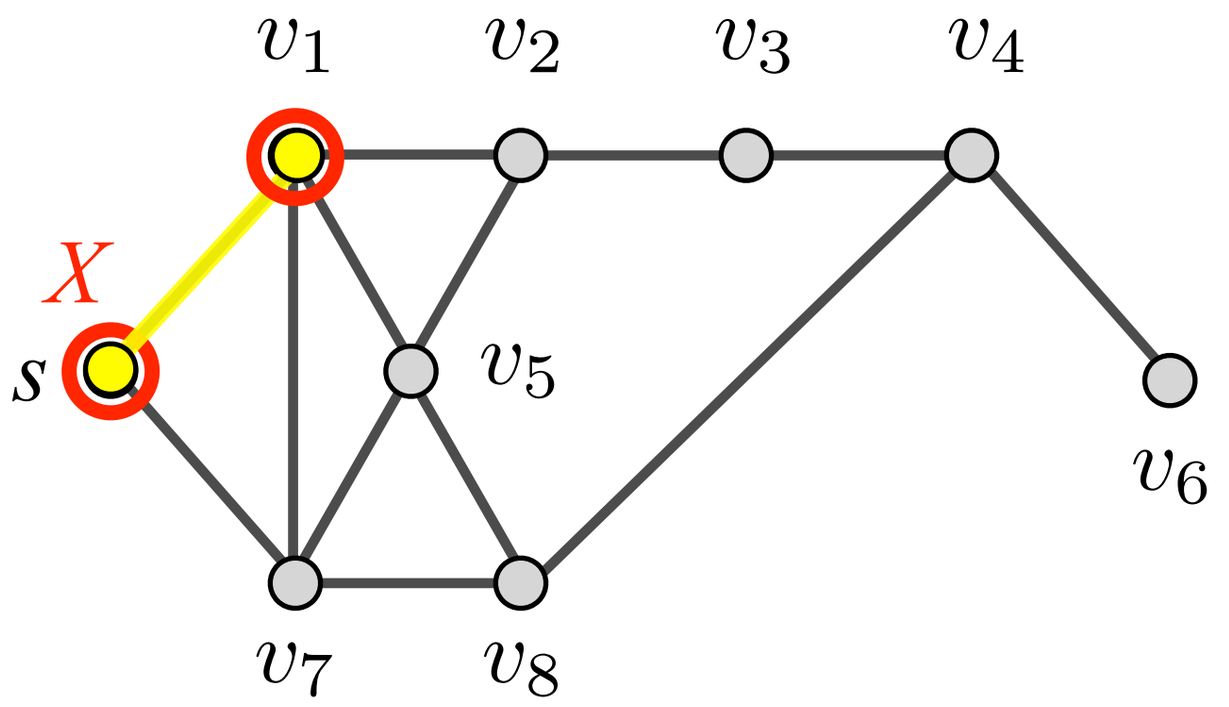
INPUT: Graph G = (V,E), Kraten s

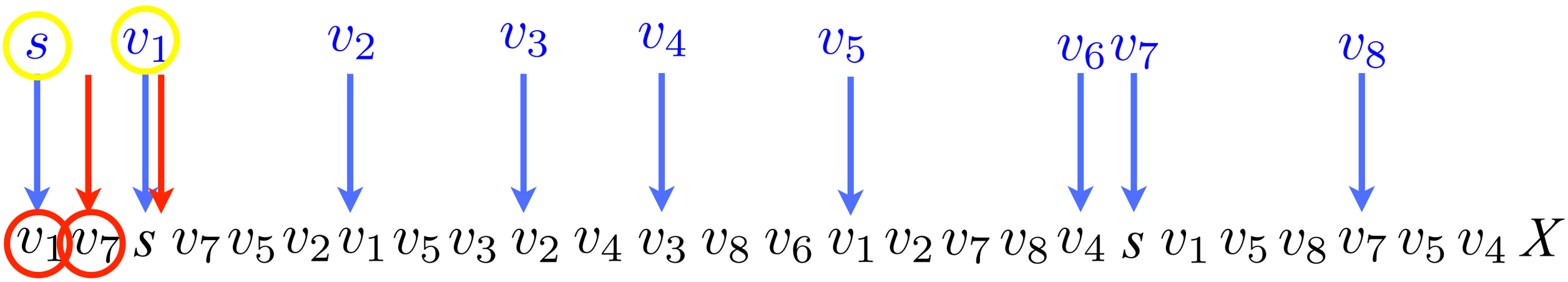
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

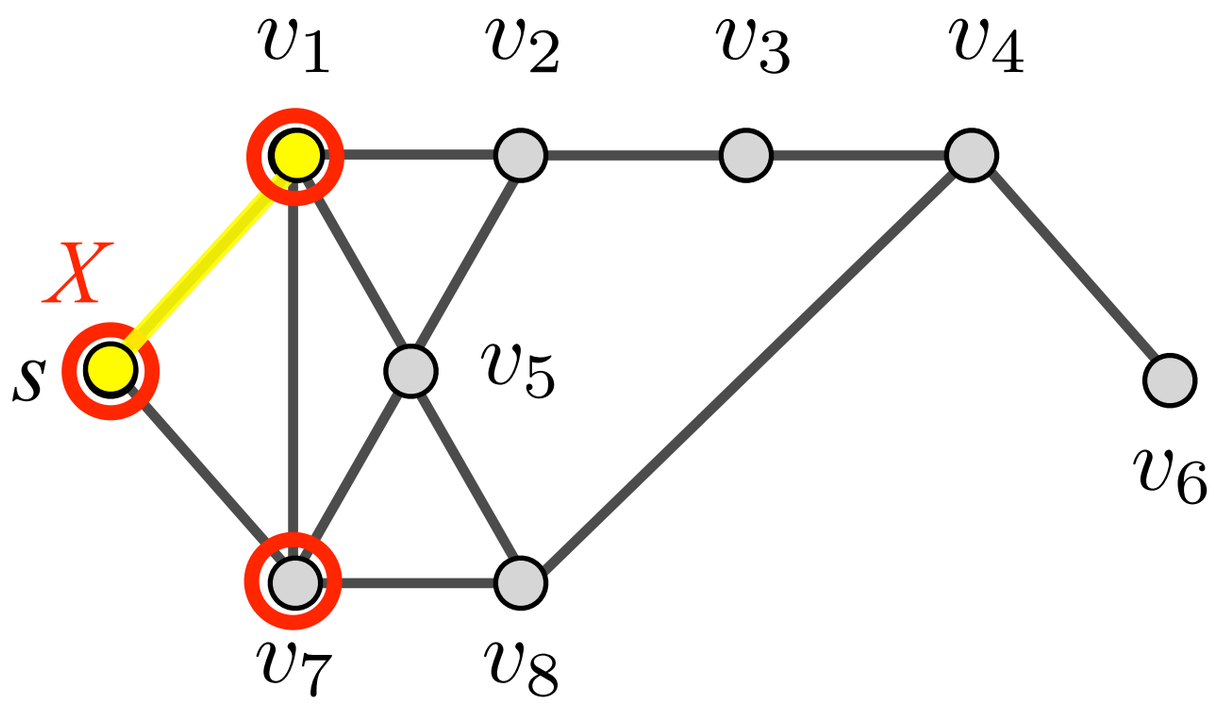
INPUT: Graph G = (V,E), Kraten s

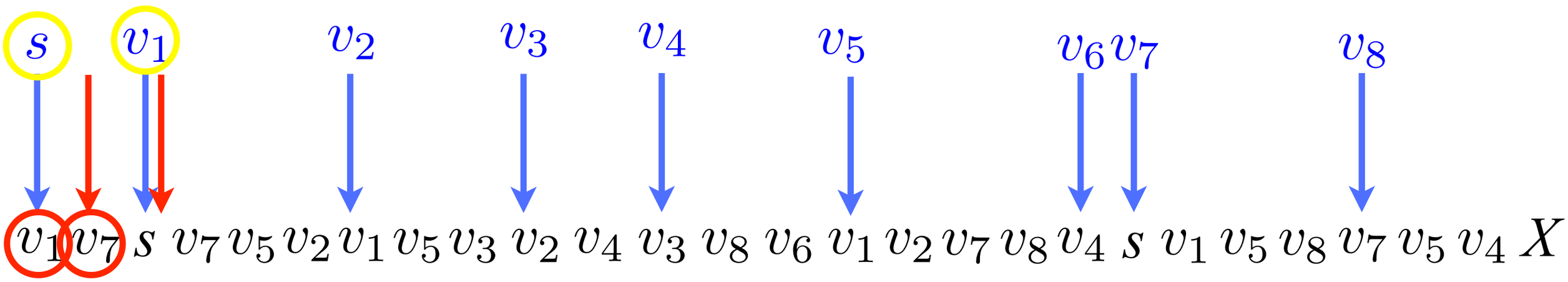
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

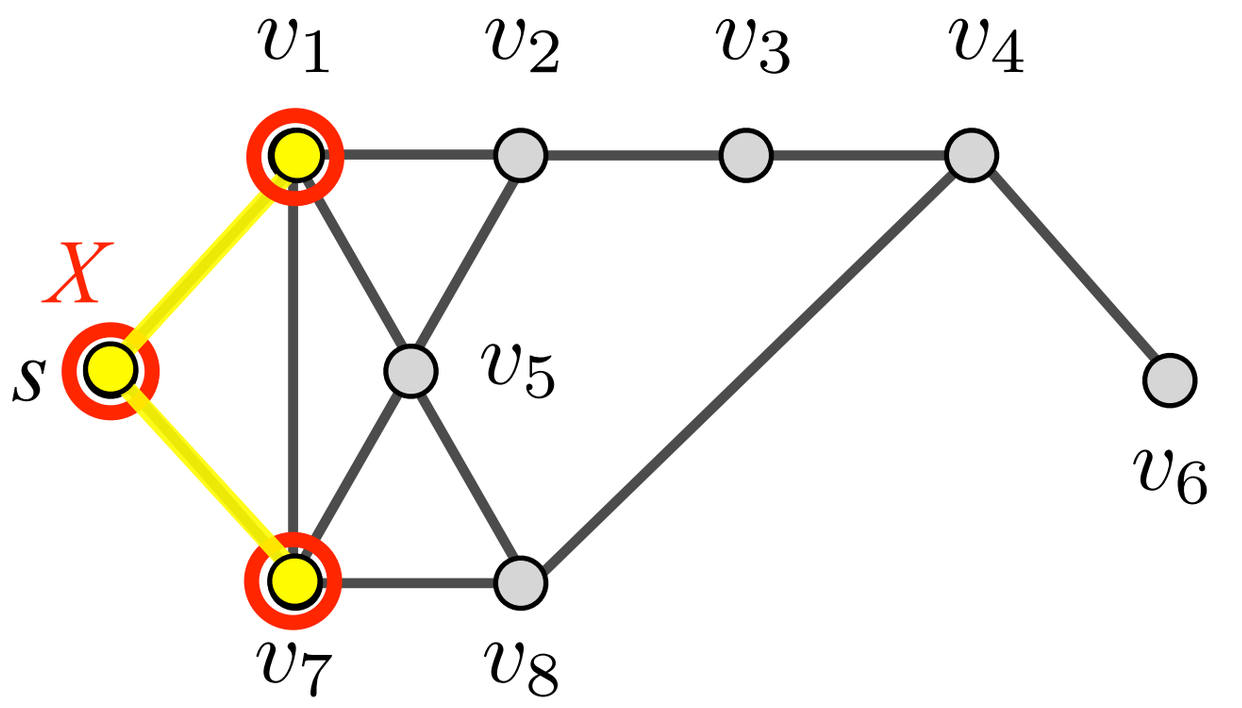
INPUT: Graph G = (V,E), Kraten s

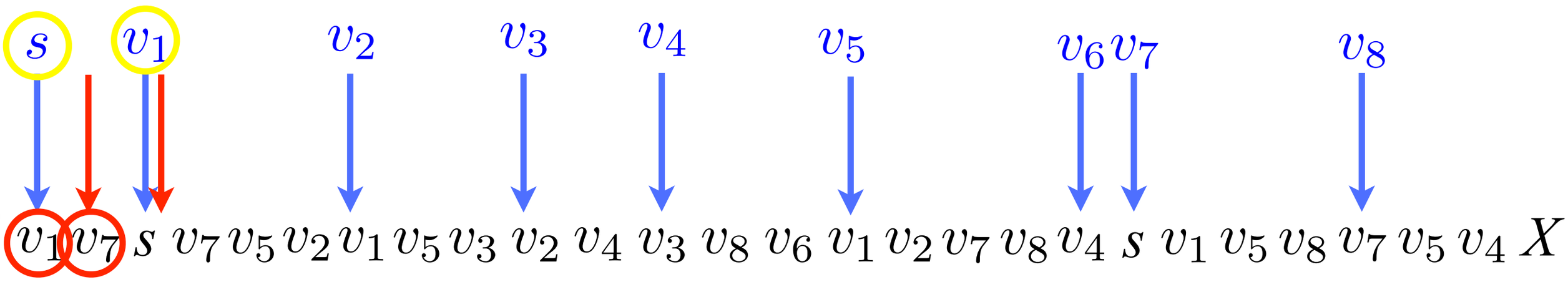
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

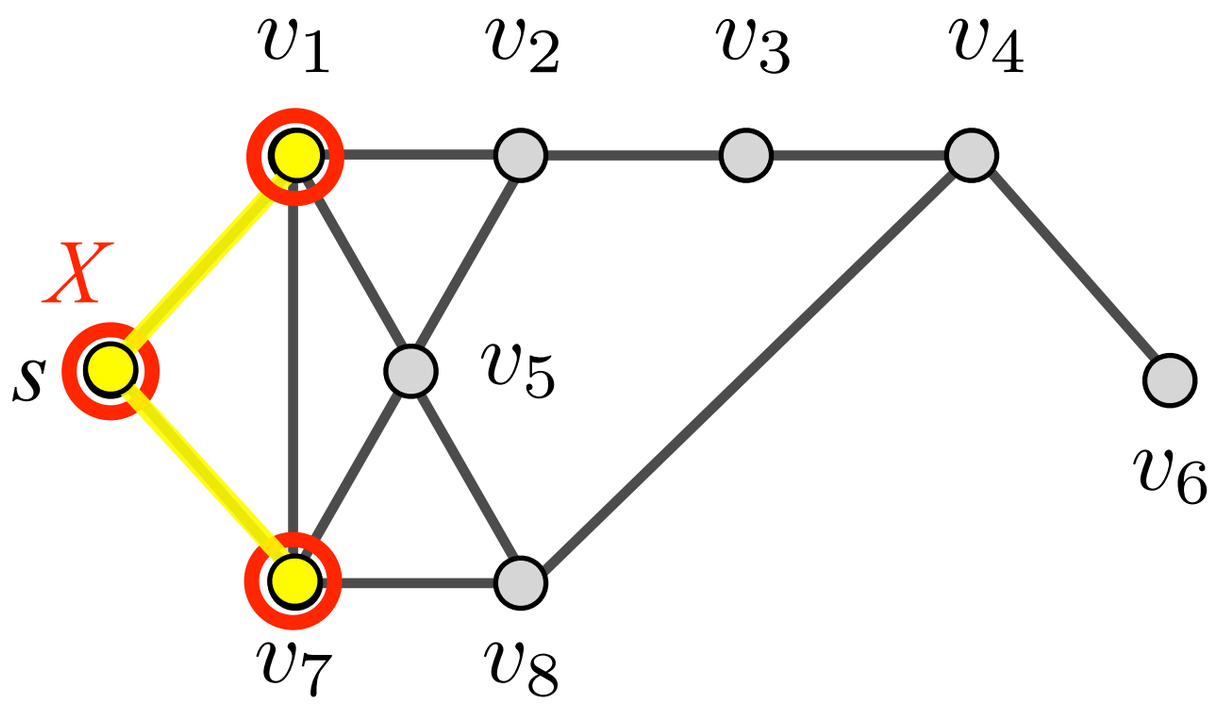
INPUT: Graph G = (V,E), Kraten s

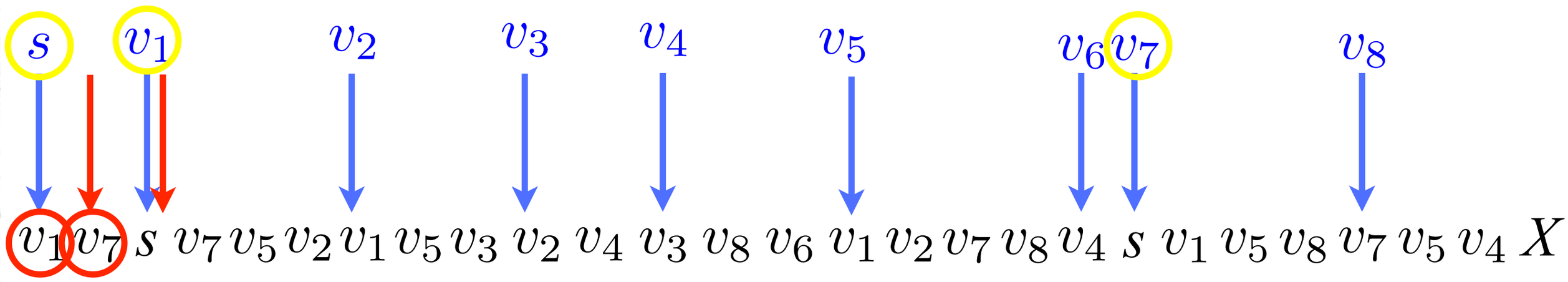
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

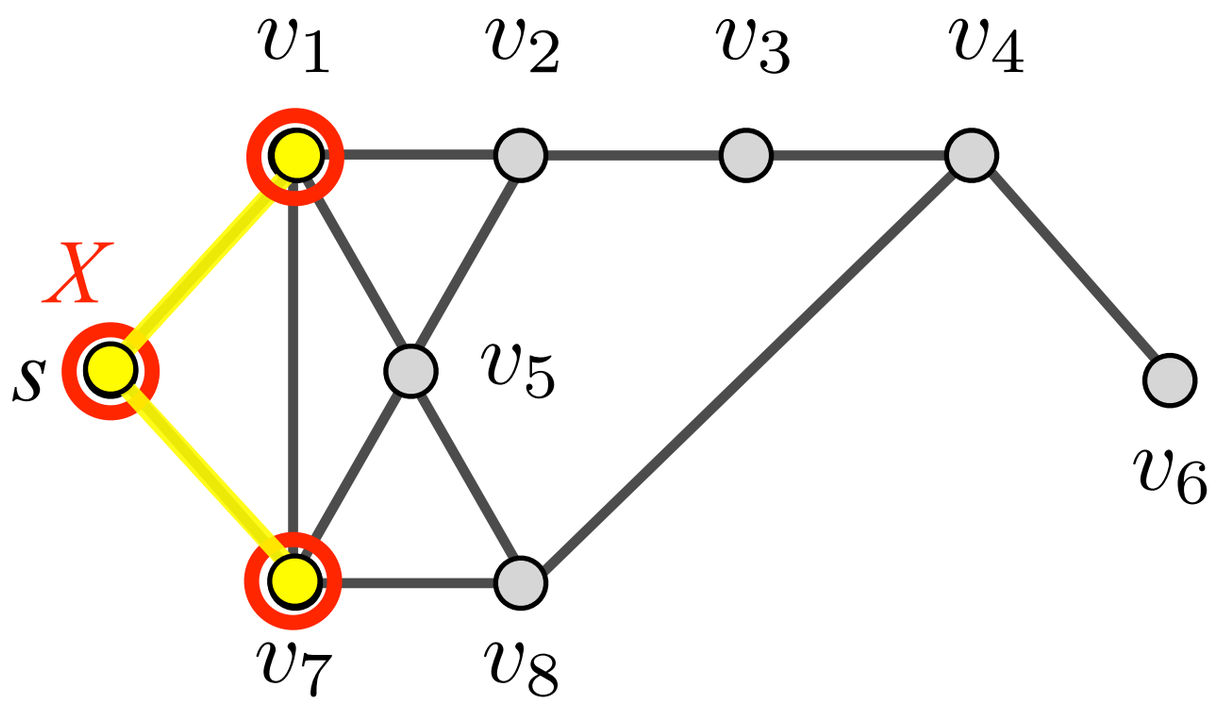
INPUT: Graph G = (V,E), Kraten s

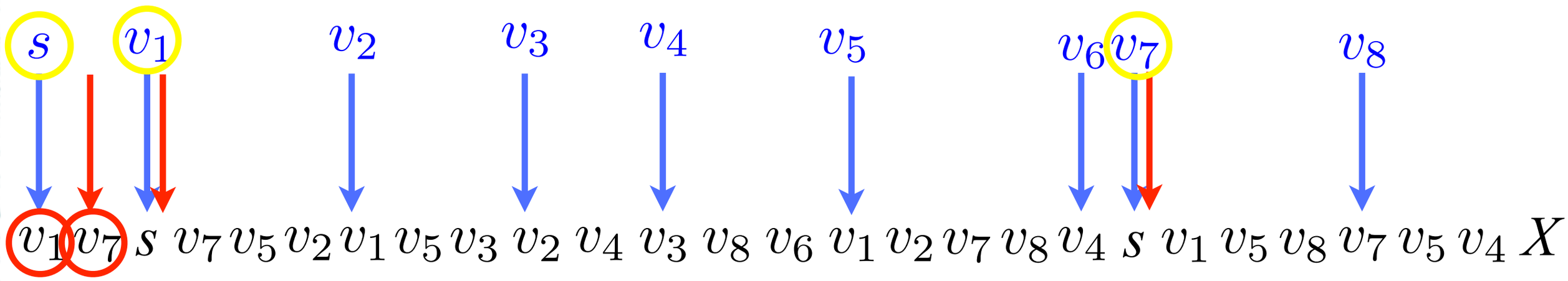
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

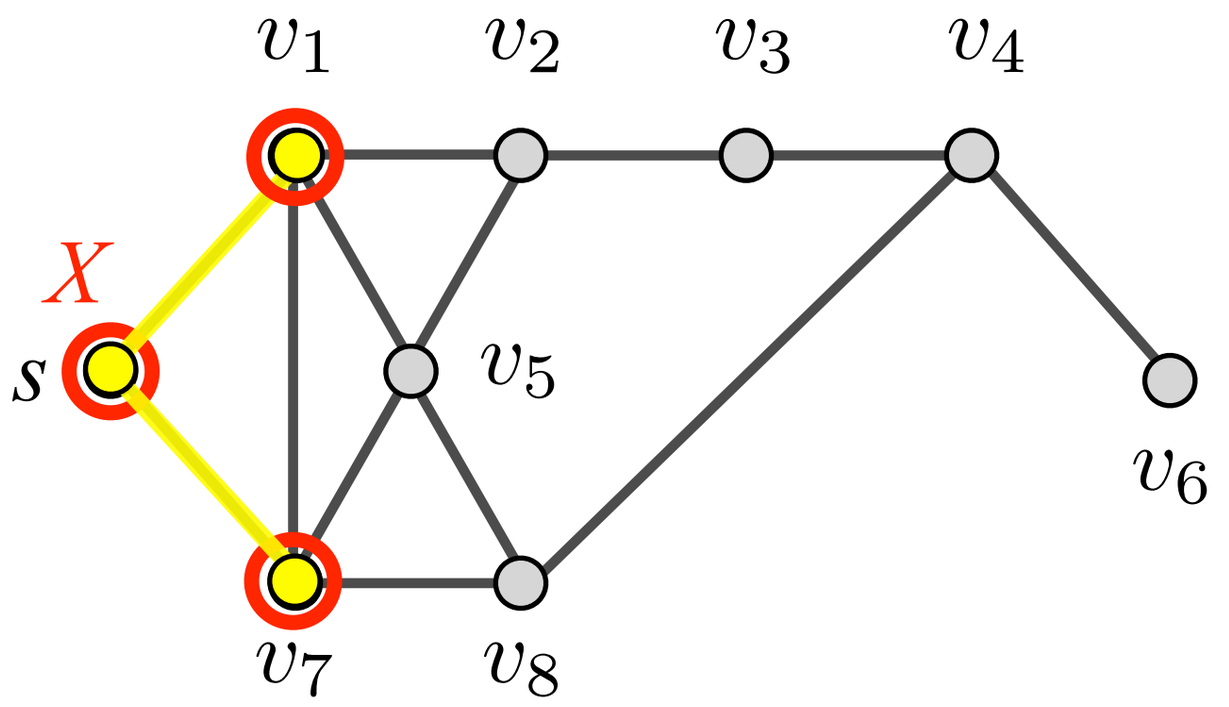
INPUT: Graph G = (V,E), Kraten s

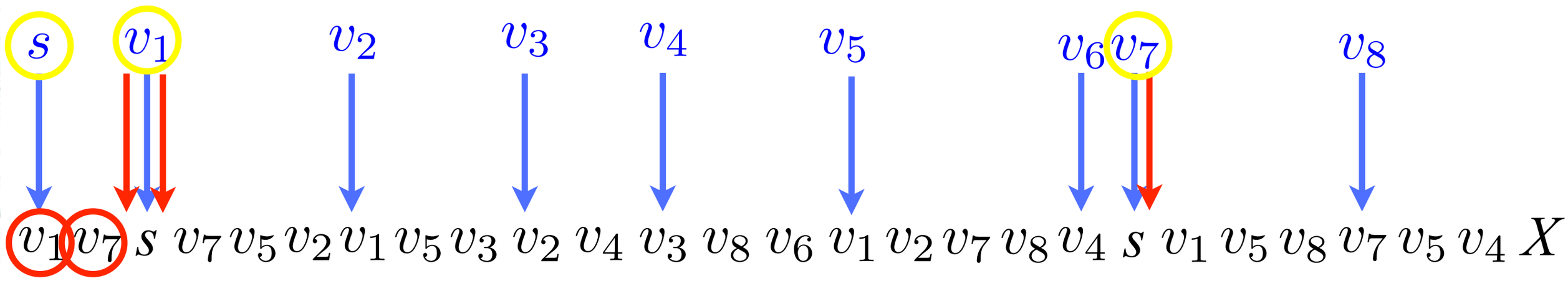
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

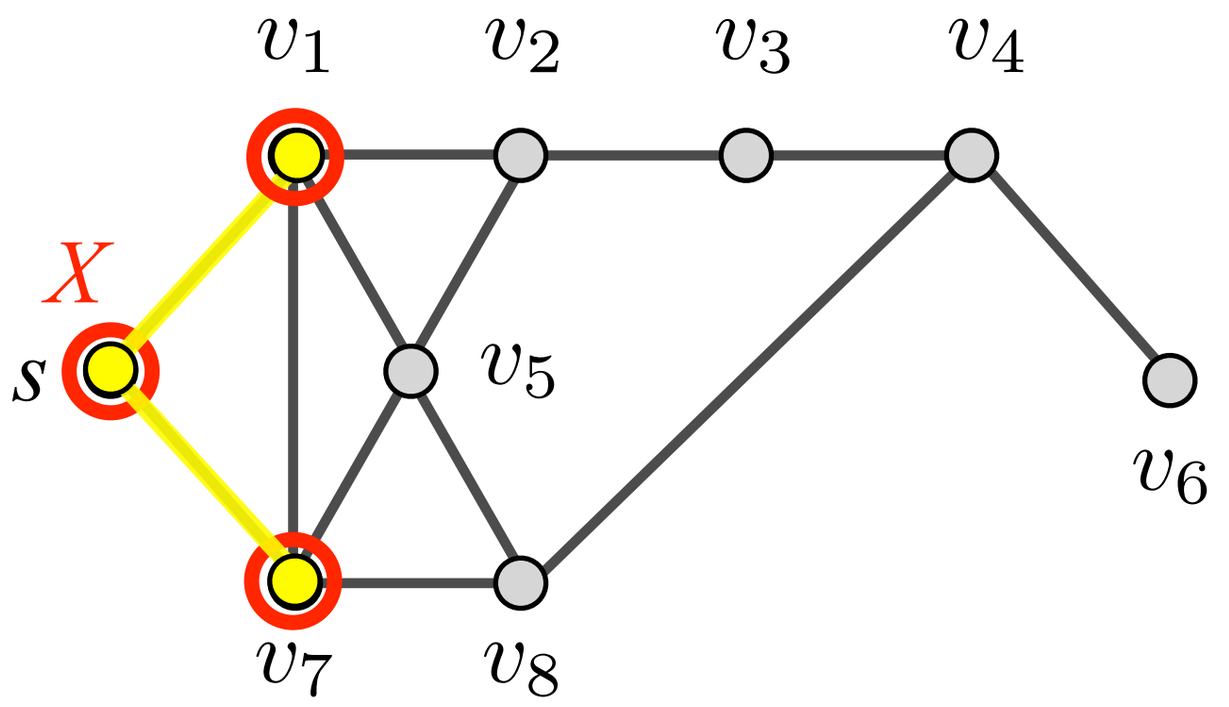
INPUT: Graph G = (V,E), Kraten s

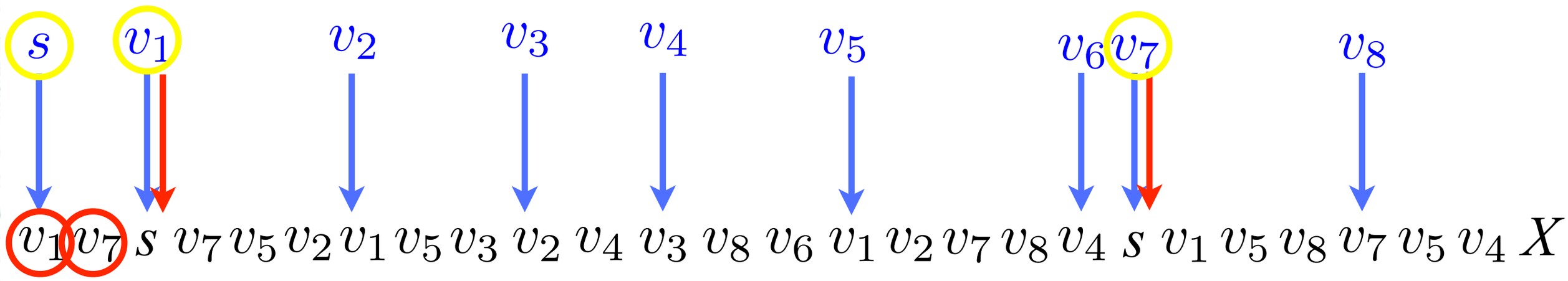
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

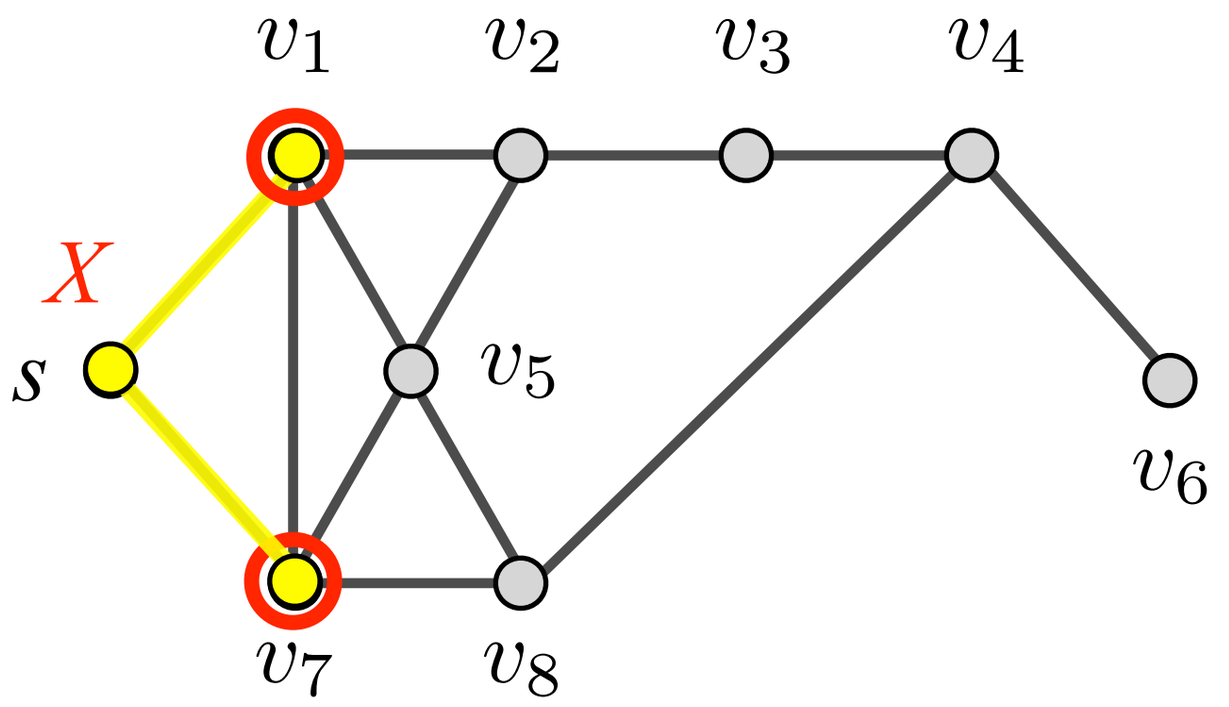
INPUT: Graph G = (V,E), Kraten s

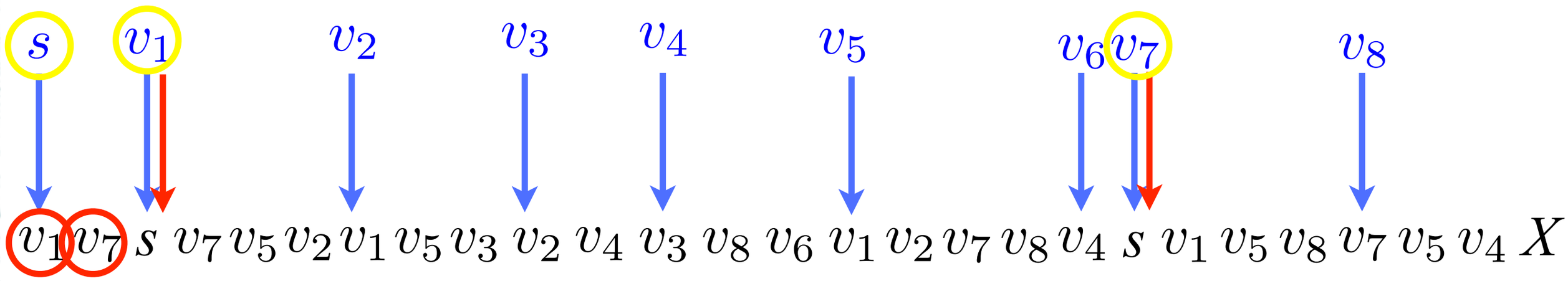
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

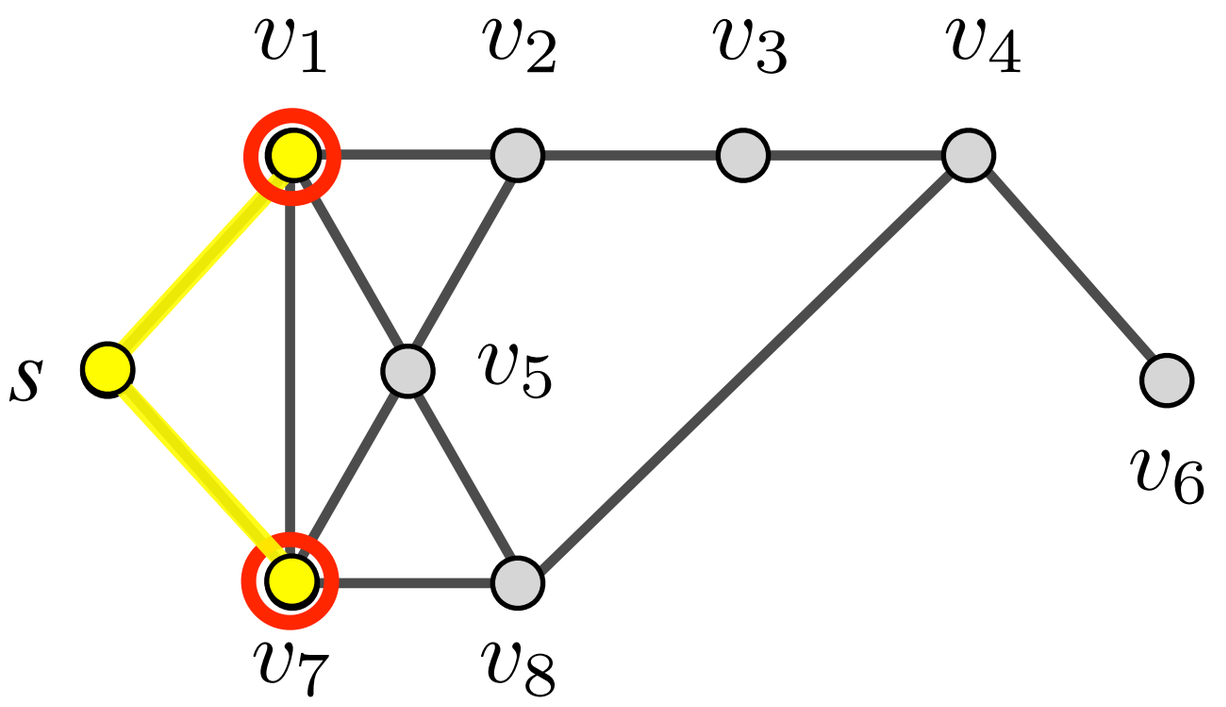
INPUT: Graph G = (V,E), Kraten s

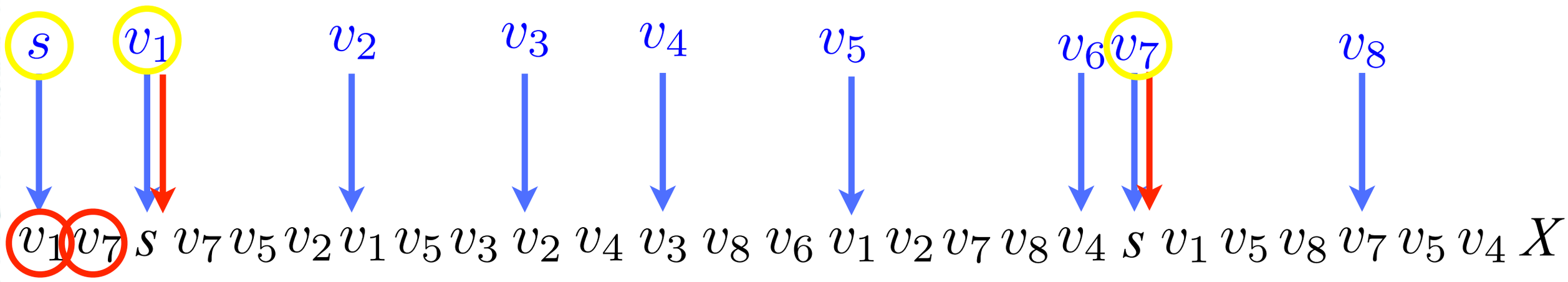
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

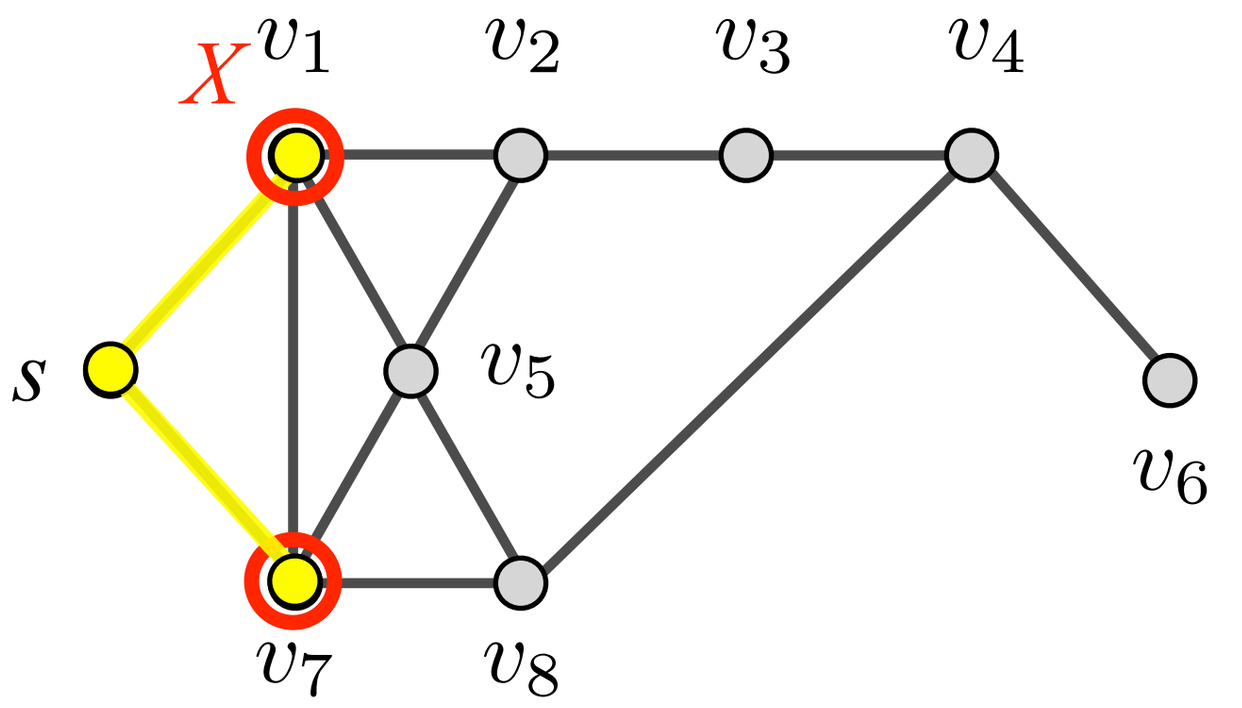
INPUT: Graph G = (V,E), Kraten s

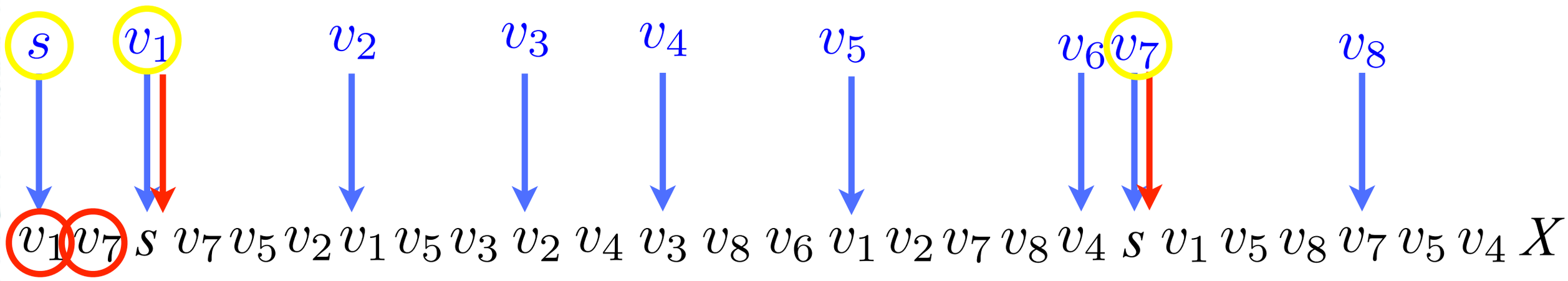
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

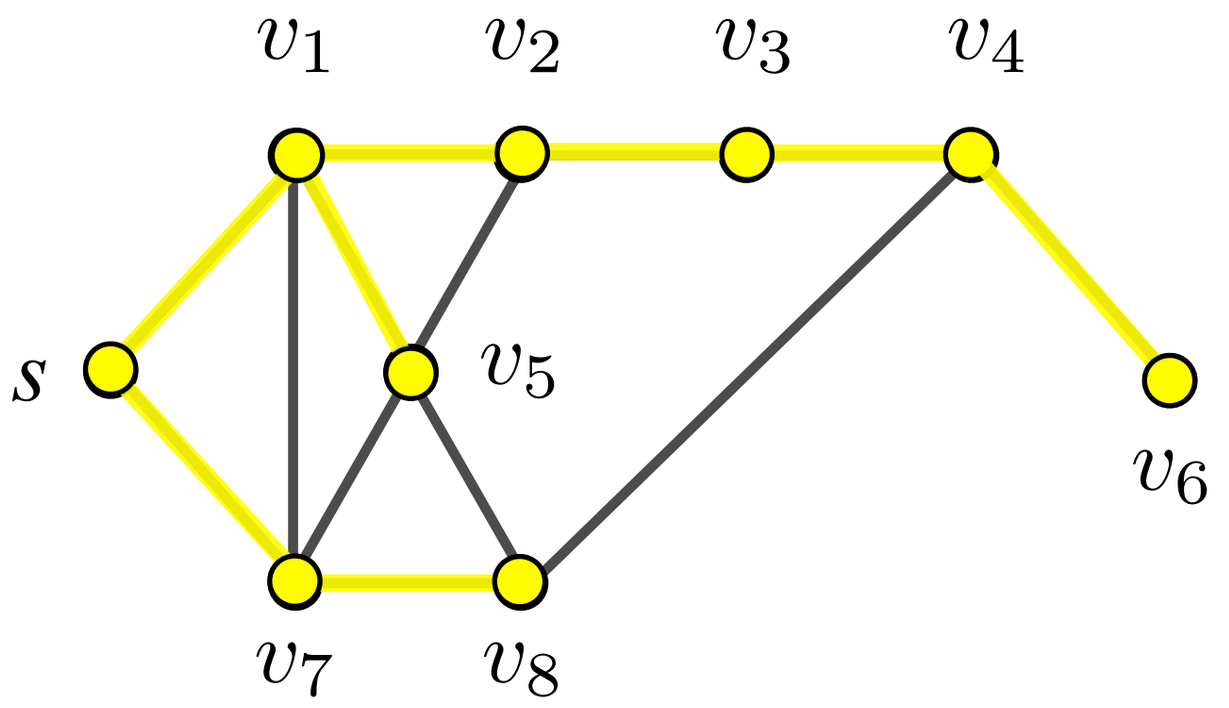
INPUT: Graph G = (V,E), Kraten s

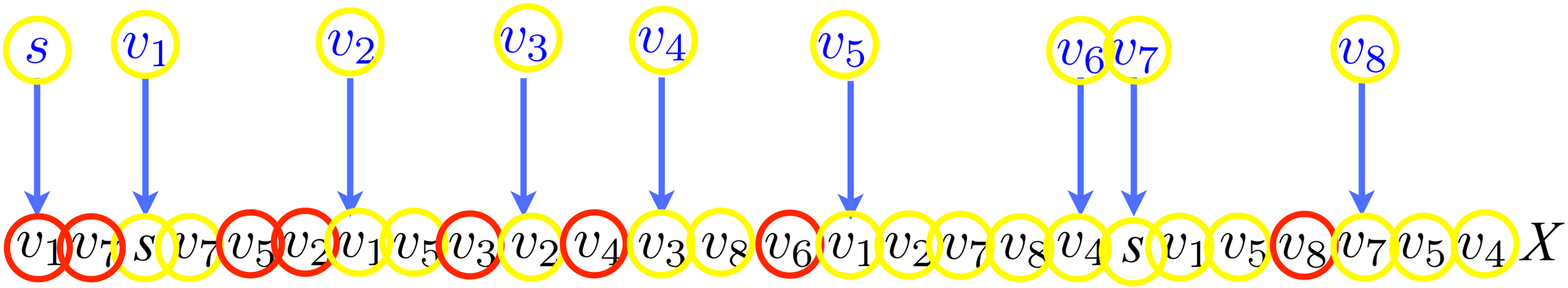
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

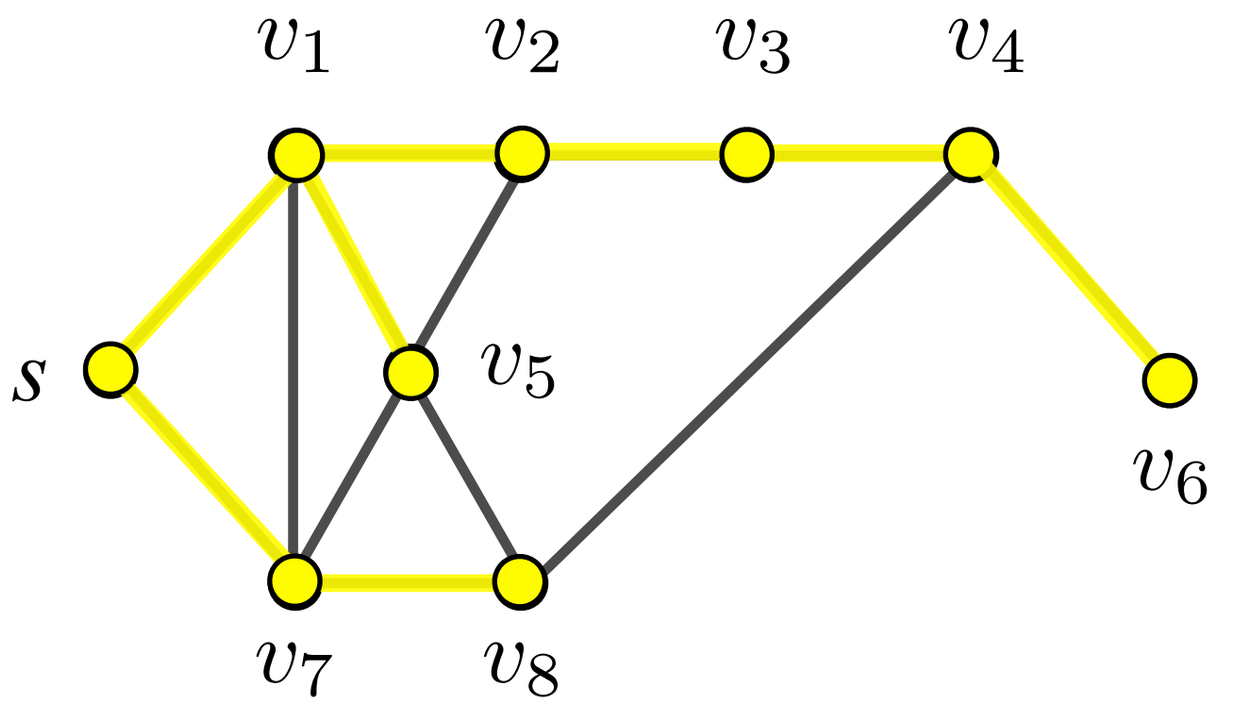
INPUT: Graph G = (V,E), Kraten s

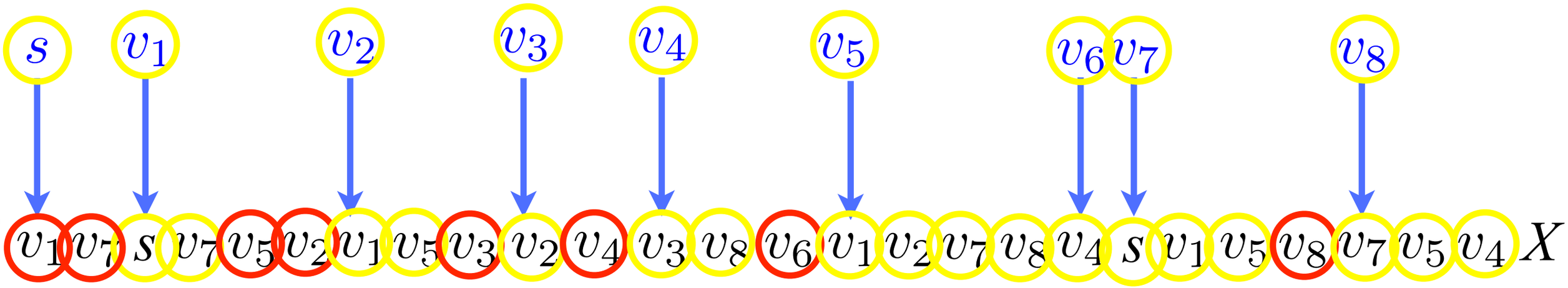
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP





Algorithmus 3.7

INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w ∈ V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w ∈ V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP

Algorithmus 3.7

INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w ∈ V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w ∈ V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP

Algorithmus 3.7

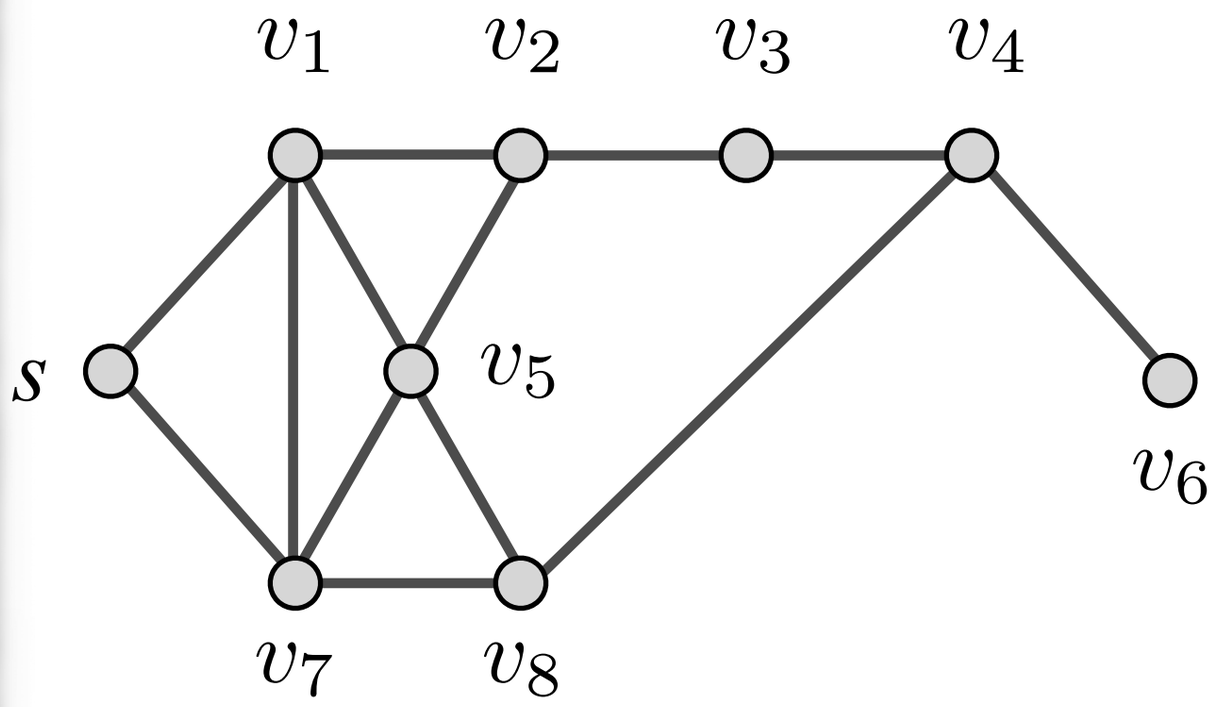
INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP



Algorithmus 3.7

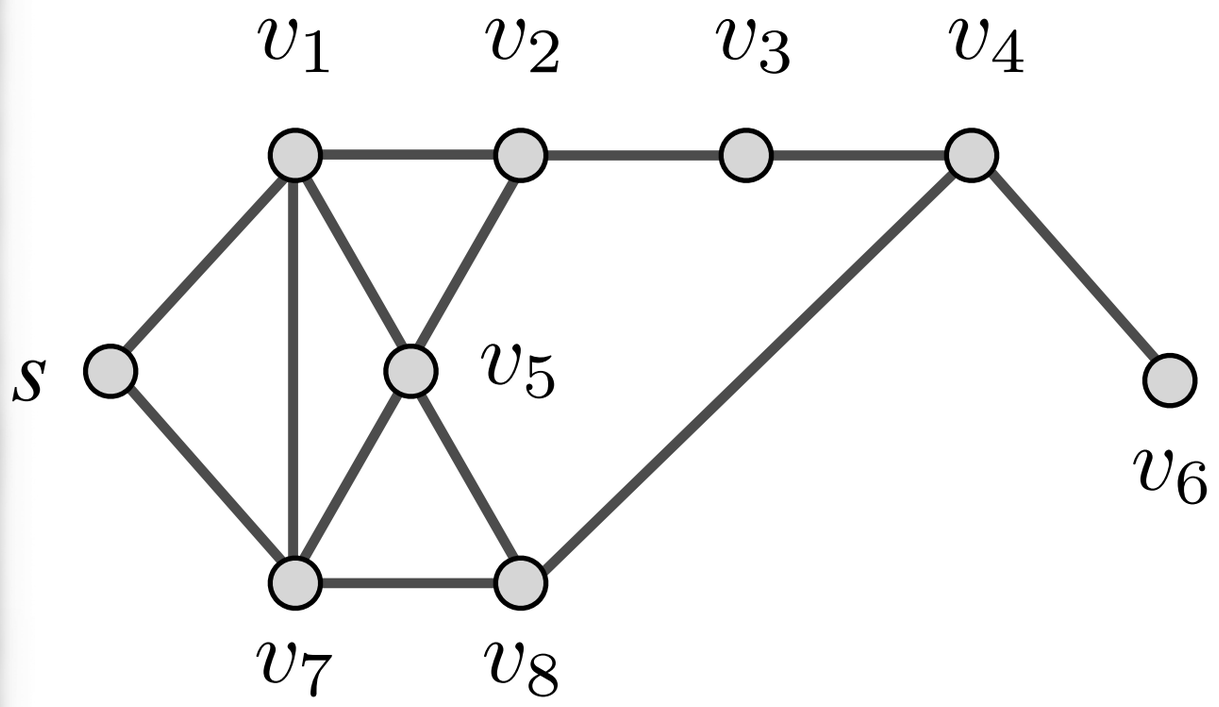
INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP



Algorithmus 3.7

INPUT: Graph G = (V,E), Kraten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

Sei R:={s}, Y:={s}, T:=∅

2. WHILE (R≠∅) DO {

2.1. Wähle v∈ R

2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN

2.2.1. R:=R\{v}

2.3. ELSE {

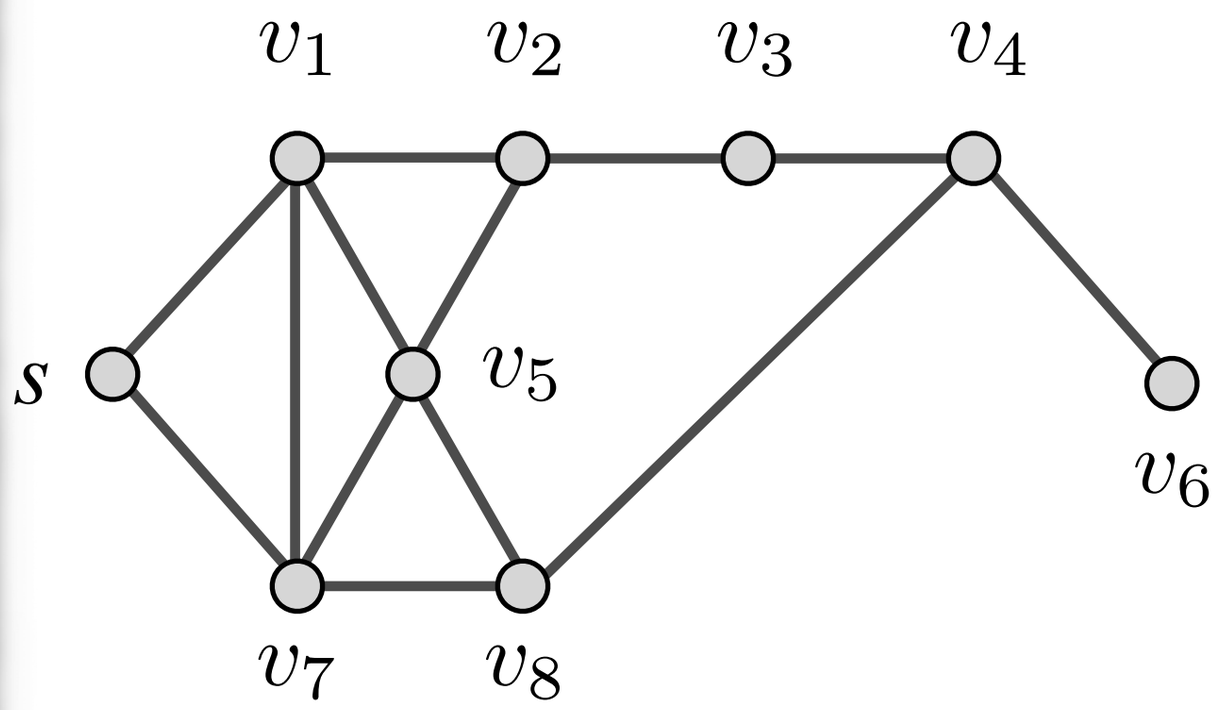
2.3.1.Wähle ein w∈V\Y mit e={v,w}∈ E

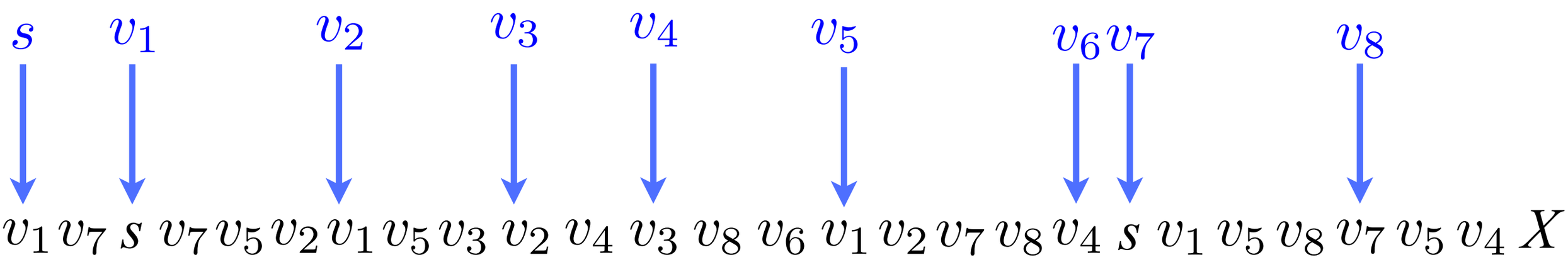
2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}

}

}

3. STOP





Algorithmus 3.7

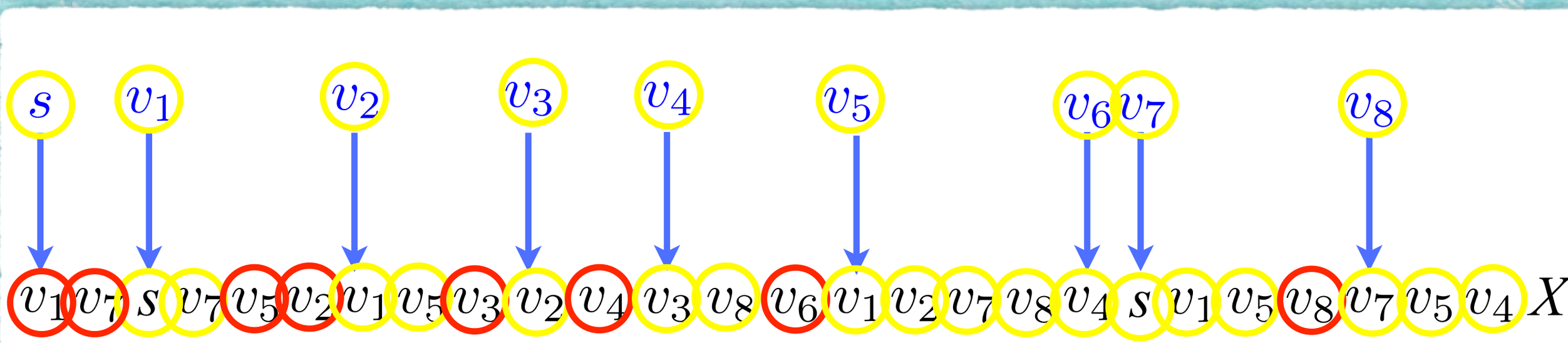
INPUT: Graph G = (V,E), Knoten s

OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist,

Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w ∈ V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w ∈ V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP



Algorithmus 3.7

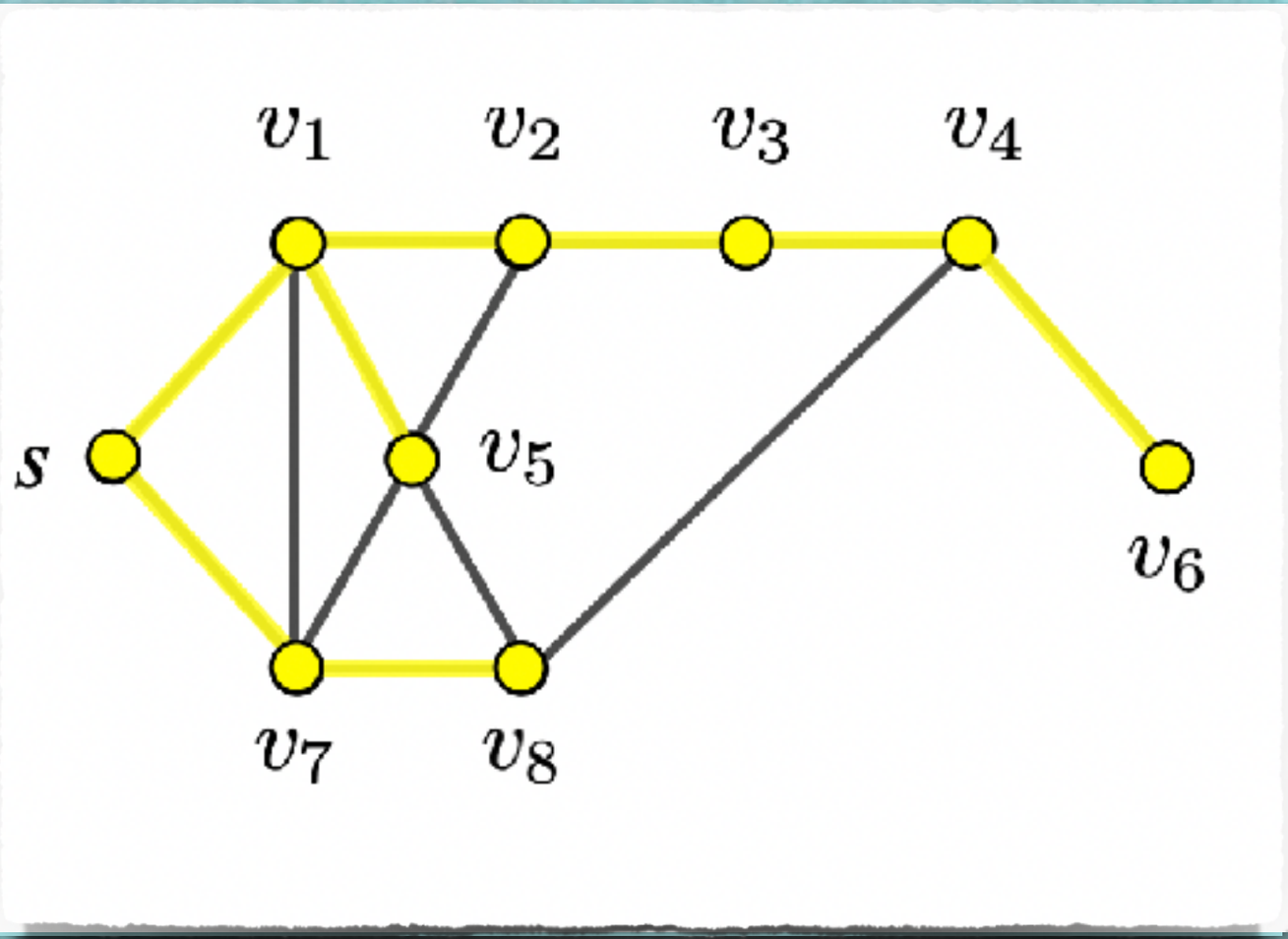
INPUT: Graph G = (V,E), Kraten s

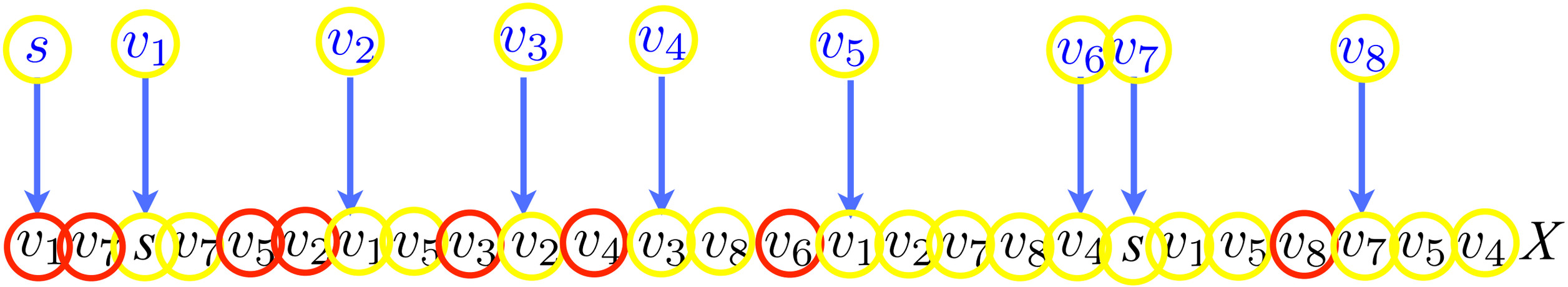
OUTPUT: Knotenmenge Y ⊆ V, die von s aus erreichbar ist;

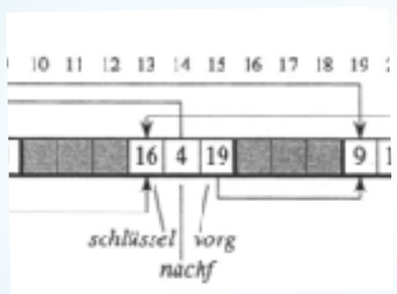
Kantenmenge T ⊆ E, die die Erreichbarkeit sicherstellt

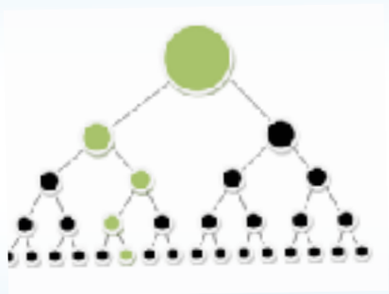
1. Sei R:={s}, Y:={s}, T:=∅
2. WHILE (R≠∅) DO {
   - 2.1. Wähle v ∈ R
   - 2.2. IF (es gibt kein w∈V\Y mit e={v,w}∈E) THEN
     - 2.2.1. R:=R\{v}
   - 2.3. ELSE {
     - 2.3.1.Wähle ein w∈V\Y mit e={v,w} ∈ E
     - 2.3.2. Setze R := R ∪ {w}, Y := Y ∪ {w}, T := T ∪ {e}
     - }

   }
3. STOP










Kapitel 4: Dynamische Datenstrukturen

Algorithmen und Datenstrukturen
WS 2020/21

Prof. Dr. Sándor Fekete

Wie verwalten wir dynamische Mengen von Objekten?

Wie verwalten wir dynamische Mengen von Objekten?

Waschkorb

4.1 Grundoperationen

4.1 Grundoperationen

Aufgabenstellung:

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***
- ***Ausführen von verschiedenen Operationen (s.u.)***

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***
- ***Ausführen von verschiedenen Operationen (s.u.)***

Im Folgenden:

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***
- ***Ausführen von verschiedenen Operationen (s.u.)***

Im Folgenden:

S Menge von Objekten

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***
- ***Ausführen von verschiedenen Operationen (s.u.)***

Im Folgenden:

S Menge von Objekten

k Wert eines Elements ("Schlüssel")

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***
- ***Ausführen von verschiedenen Operationen (s.u.)***

Im Folgenden:

S **Menge von Objekten**

k **Wert eines Elements (“Schlüssel”)**

x **Zeiger auf Element**

4.1 Grundoperationen

Aufgabenstellung:

- ***Verwalten einer Menge S von Objekten***
- ***Ausführen von verschiedenen Operationen (s.u.)***

Im Folgenden:

S Menge von Objekten

k Wert eines Elements ("Schlüssel")

x Zeiger auf Element

NIL spezieller, "leerer" Zeiger

4.1 Grundoperationen

4.1 Grundoperationen

<u>SEARCH(S,k):</u> **“Suche in S nach k”**

4.1 Grundoperationen

<u>SEARCH(S,k):</u> **“Suche in S nach k”**

Durchsuche die Menge S nach einem Element von Wert k.

4.1 Grundoperationen

<u>SEARCH(S,k):</u> **“Suche in S nach k”**

Durchsuche die Menge S nach einem Element von Wert k.

Ausgabe: Zeiger x, falls x existent

4.1 Grundoperationen

<u>SEARCH(S,k):</u> **“Suche in S nach k”**

Durchsuche die Menge S nach einem Element von Wert k.

Ausgabe: Zeiger x, falls x existent
NIL, falls kein Element Wert k hat.

4.1 Grundoperationen

4.1 Grundoperationen

<u>INSERT(S,x):</u> **“Füge x in S ein”**

4.1 Grundoperationen

<u>INSERT(S,x):</u> **“Füge x in S ein”**

Erweitere S um das Element, das unter der Adresse x steht.

4.1 Grundoperationen

DELETE(S,x): **“Entferne x aus S”**

DELETE(S,x): **“Entferne x aus S”**

Lösche das unter der Adresse x stehende Element aus der Menge S.

4.1 Grundoperationen

MINIMUM(S): **“Suche das Minimum in S”**

MINIMUM(S): **"Suche das Minimum in S"**

Finde in S ein Element von kleinstem Wert.

<u>MINIMUM(S):</u> **“Suche das Minimum in S”**

Finde in S ein Element von kleinstem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

MINIMUM(S): **"Suche das Minimum in S"**

Finde in S ein Element von kleinstem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

Ausgabe: Zeiger x auf solch ein Element

4.1 Grundoperationen

MAXIMUM(S): **“Suche das Maximum in S”**

MAXIMUM(S): **“Suche das Maximum in S”**

Finde in S ein Element von größtem Wert.

<u>MAXIMUM(S):</u> **“Suche das Maximum in S”**

Finde in S ein Element von größtem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

MAXIMUM(S): **"Suche das Maximum in S"**

Finde in S ein Element von größtem Wert. (Annahme: Die Werte lassen sich vollständig vergleichen!)

Ausgabe: Zeiger x auf solch ein Element

4.1 Grundoperationen

PREDECESSOR(S,x):
“Finde das nächstkleinere Element”

PREDECESSOR(S,x):
“Finde das nächstkleinere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstkleinerem Wert in S.

<u>PREDECESSOR(S,x):</u>
“Finde das nächstkleinere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstkleinerem Wert in S.

Ausgabe: Zeiger y auf Element

PREDECESSOR(S,x):
“Finde das nächstkleinere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstkleinerem Wert in S.

Ausgabe: Zeiger y auf Element
NIL, falls x Minimum von S angibt

4.1 Grundoperationen

SUCCESSOR(S,x):

“Finde das nächstgrößere Element”

<u>SUCCESSOR(S,x):</u>
“Finde das nächstgrößere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstgrößerem Wert in S.

<u>SUCCESSOR(S,x):</u>
“Finde das nächstgrößere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstgrößerem Wert in S.

Ausgabe: Zeiger y auf Element

<u>SUCCESSOR(S,x):</u>
“Finde das nächstgrößere Element”

Für ein in x stehendes Element in S, bestimme ein Element von nächstgrößerem Wert in S.

Ausgabe: Zeiger y auf Element
NIL, falls x Maximum von S angibt

4.1 Grundoperationen

4.1 Grundoperationen

<u>Wie nimmt man das vor?</u>

4.1 Grundoperationen

Wie nimmt man das vor?

Wie lange dauert das,
in Abhängigkeit von der Größe von S?

4.1 Grundoperationen

<u>Wie nimmt man das vor?</u>

**<u>Wie lange dauert das,
in Abhängigkeit von der Größe von S?</u>**

Unsortierte Unterlagen:

4.1 Grundoperationen

<u>Wie nimmt man das vor?</u>

**<u>Wie lange dauert das,
in Abhängigkeit von der Größe von S?</u>**

Unsortierte Unterlagen:
Immer alles durchgehen, also: **O(n)**

4.1 Grundoperationen

<u>Wie nimmt man das vor?</u>

**<u>Wie lange dauert das,
in Abhängigkeit von der Größe von S?</u>**

Unsortierte Unterlagen:
Immer alles durchgehen, also: **O(n)**

Sortierte Unterlagen: Geht schneller!

4.1 Grundoperationen

4.1 Grundoperationen

Langsam:

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

  Alle Objekte anschauen

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

  Alle Objekte anschauen

Sehr schnell:

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

Alle Objekte anschauen

Sehr schnell:

- ***O(1):*** ***konstante Zeit***

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

  Alle Objekte anschauen

Sehr schnell:

- ***O(1):*** ***konstante Zeit***

  Immer gleich schnell, egal wie groß S ist.

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

  Alle Objekte anschauen

Sehr schnell:

- ***O(1):*** ***konstante Zeit***

  Immer gleich schnell, egal wie groß S ist.

Schnell:

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

Alle Objekte anschauen

Sehr schnell:

- ***O(1):*** ***konstante Zeit***

Immer gleich schnell, egal wie groß S ist.

Schnell:

- ***O(log n): logarithmische Zeit***

4.1 Grundoperationen

Langsam:

- ***O(n):*** ***lineare Zeit***

  Alle Objekte anschauen

Sehr schnell:

- ***O(1):*** ***konstante Zeit***

  Immer gleich schnell, egal wie groß S ist.

Schnell:

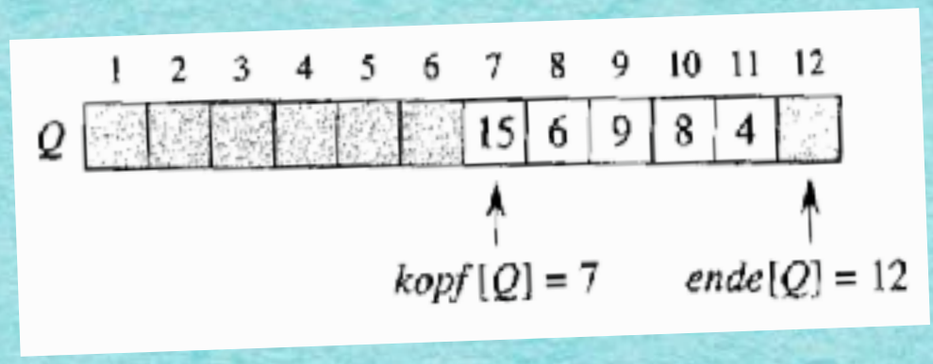
- ***O(log n):*** ***logarithmische Zeit***

  Wiederholtes Halbieren

4.2 Stapel und Warteschlange

4.2 Stapel und Warteschlange

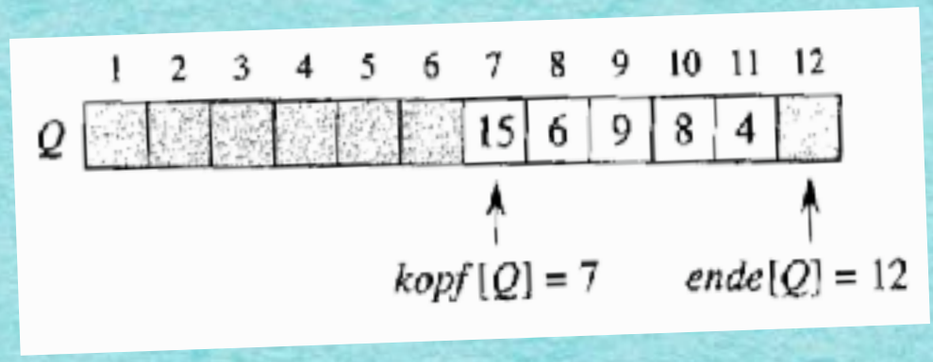
OUT



IN

4.2 Stapel und Warteschlange

OUT

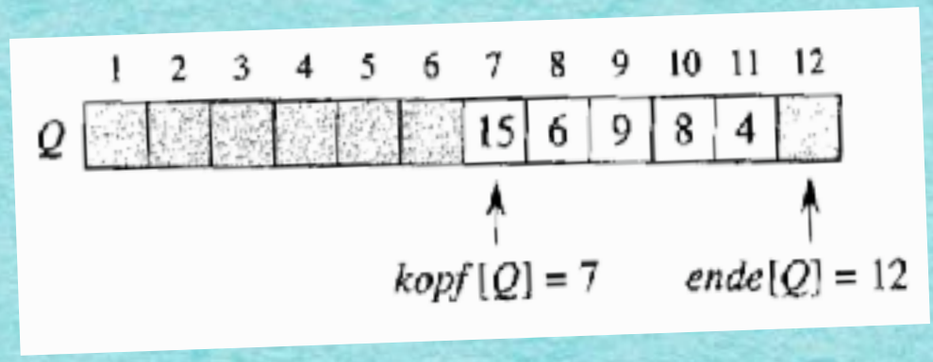


IN

Enqueue: 17, 3, 5

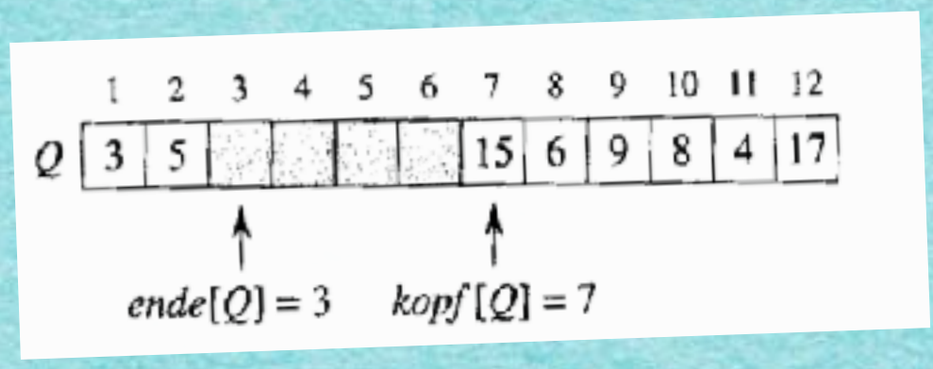
4.2 Stapel und Warteschlange

OUT



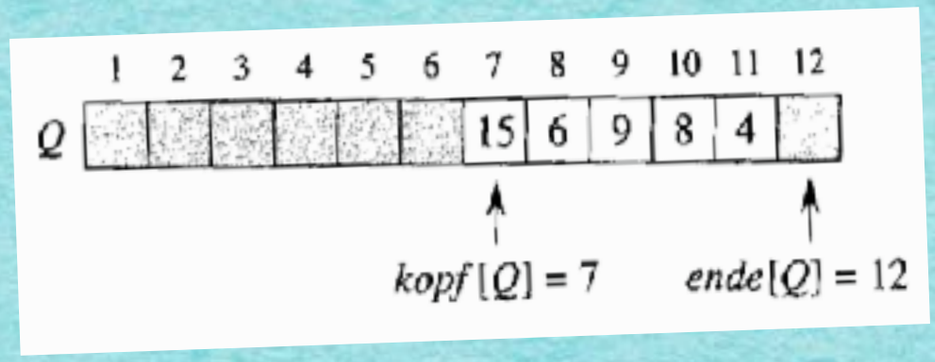
IN

Enqueue: 17, 3, 5



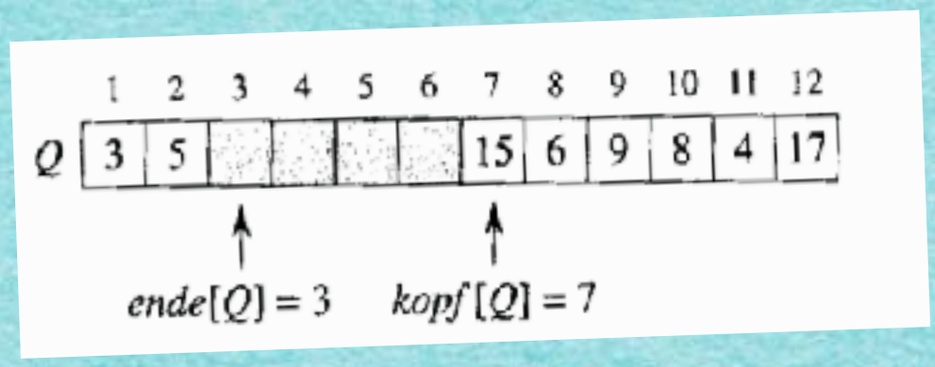
4.2 Stapel und Warteschlange

OUT



IN

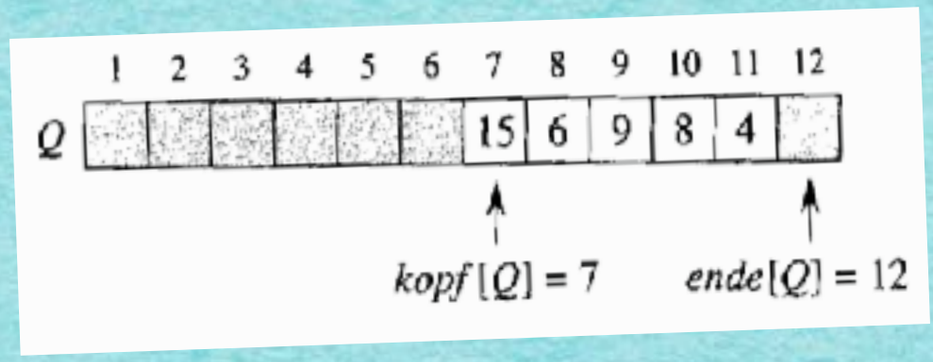
Enqueue: 17, 3, 5



Dequeue:

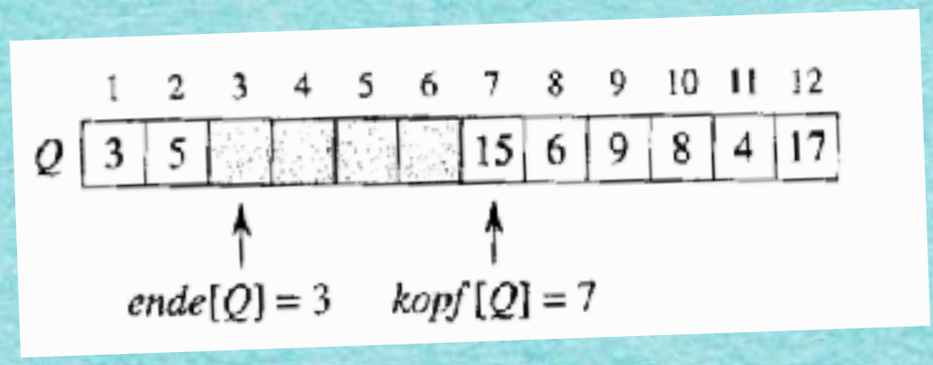
4.2 Stapel und Warteschlange

OUT

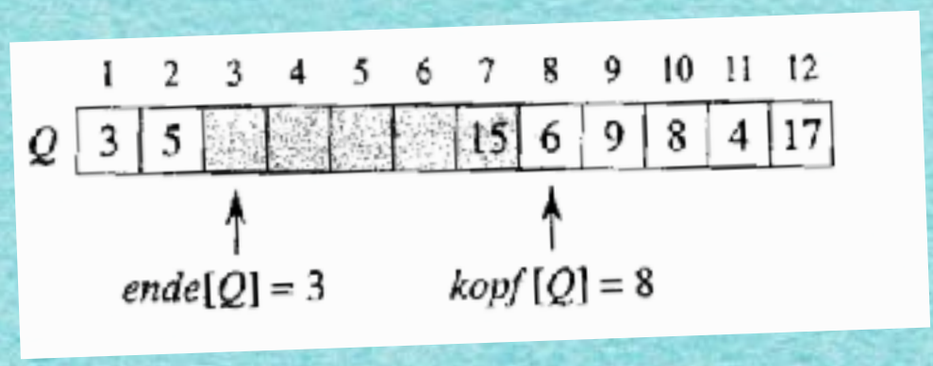


IN

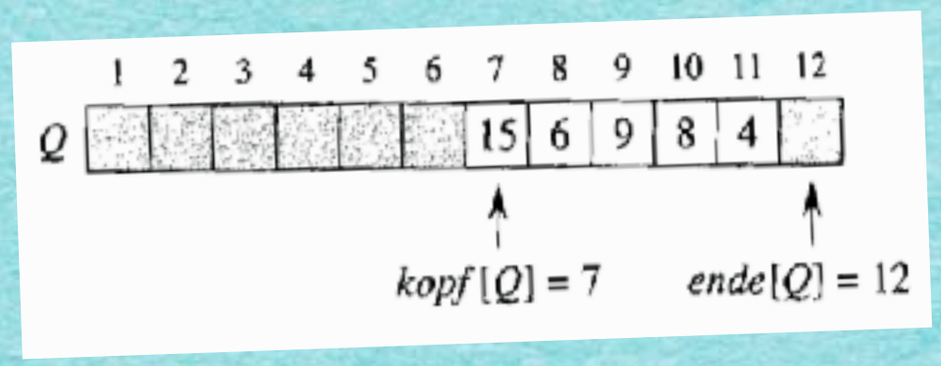
Enqueue: 17, 3, 5

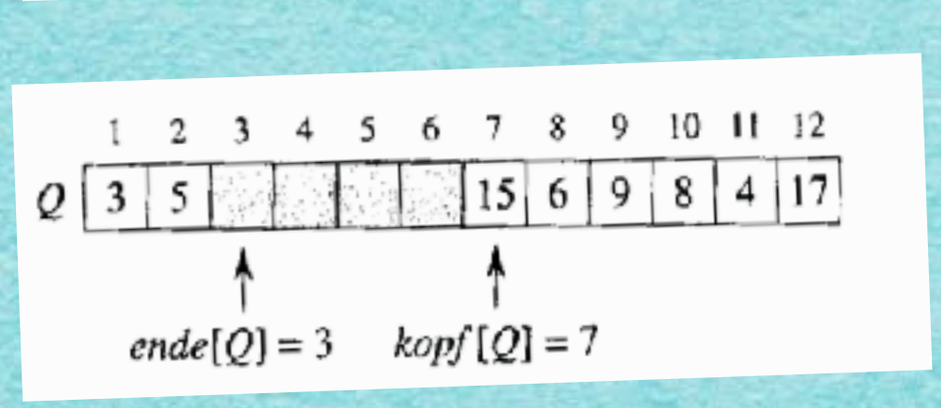


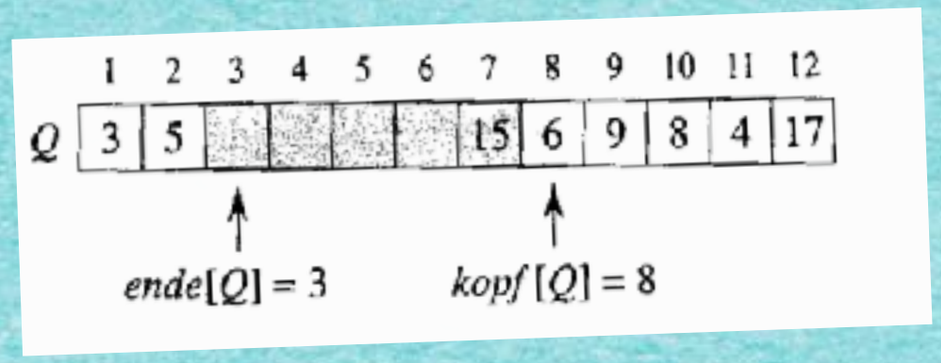
Dequeue:



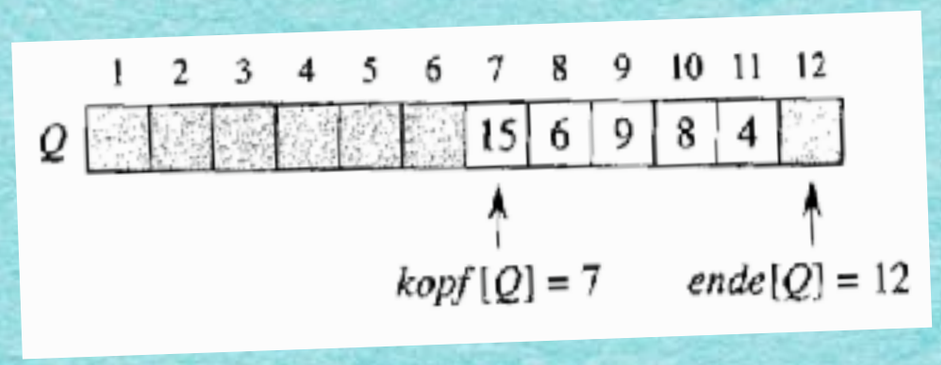
Warteschlange auf Array umgesetzt

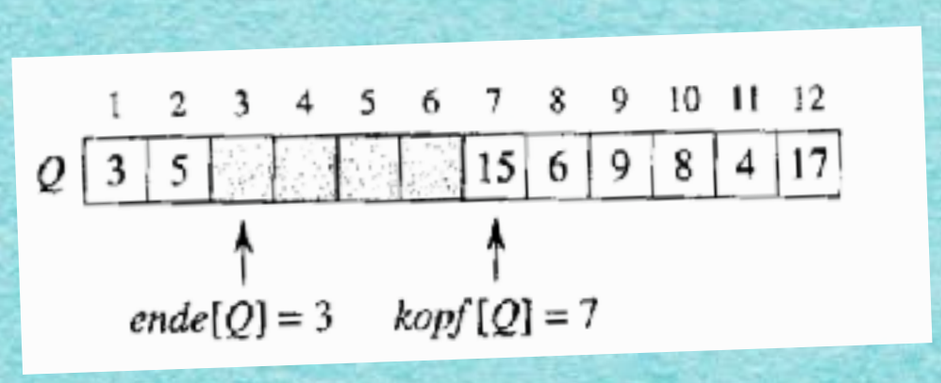


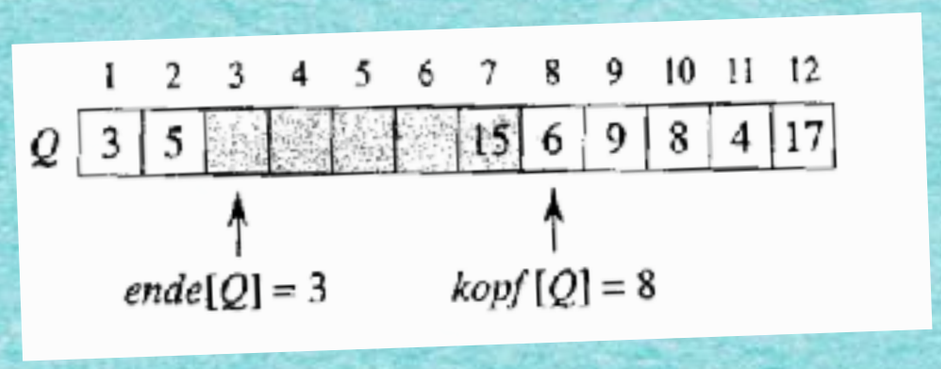




Warteschlange auf Array umgesetzt

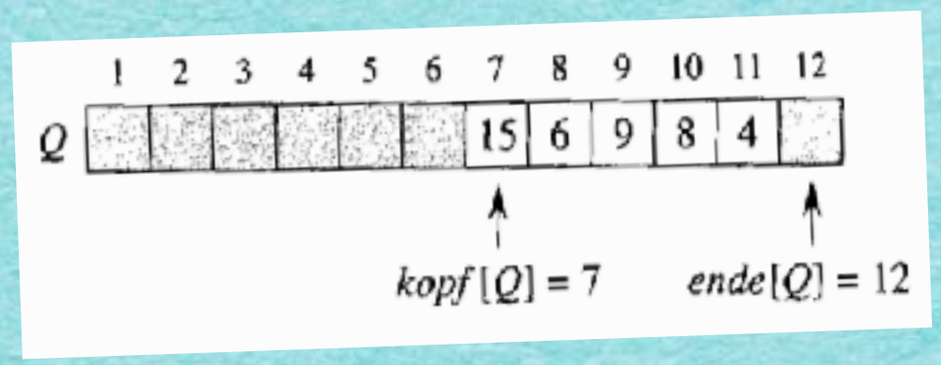


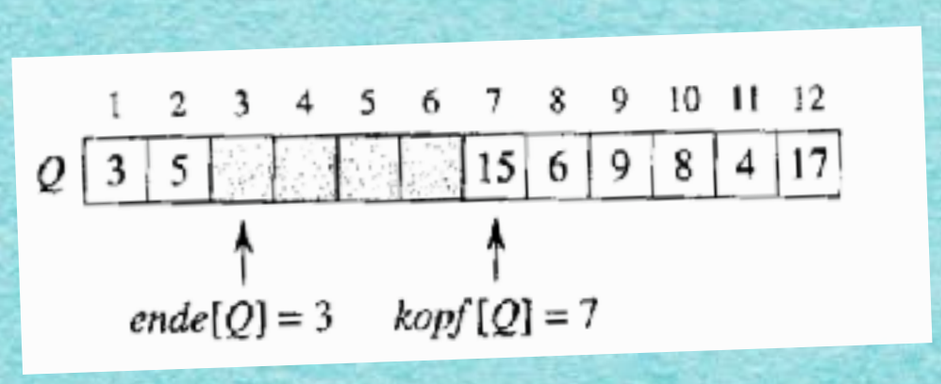


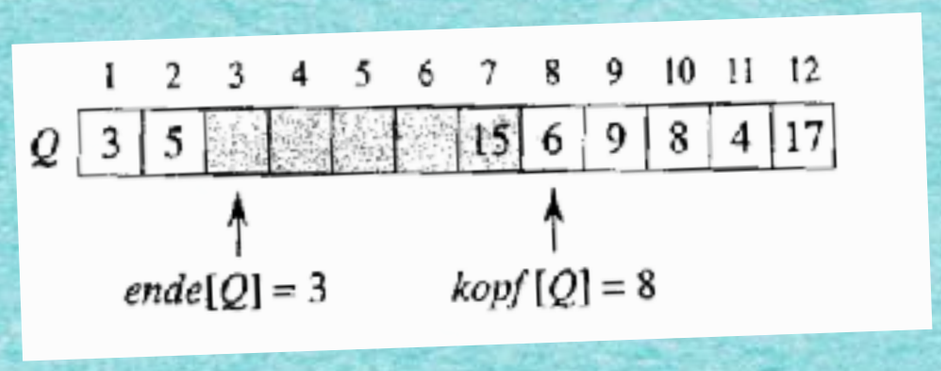



```
ENQUEUE(Q, x)
1  Q[ende[Q]] ← x
2  if ende[Q] = länge[Q]
3     then ende[Q] ← 1
4     else ende[Q] ← ende[Q] + 1
```

Warteschlange auf Array umgesetzt



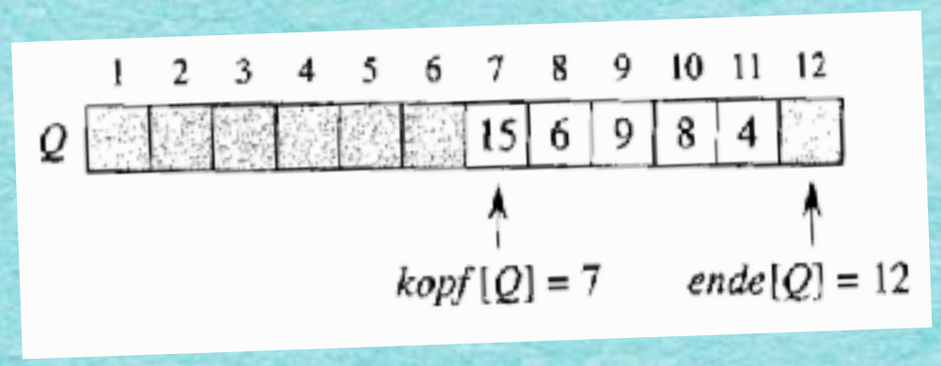


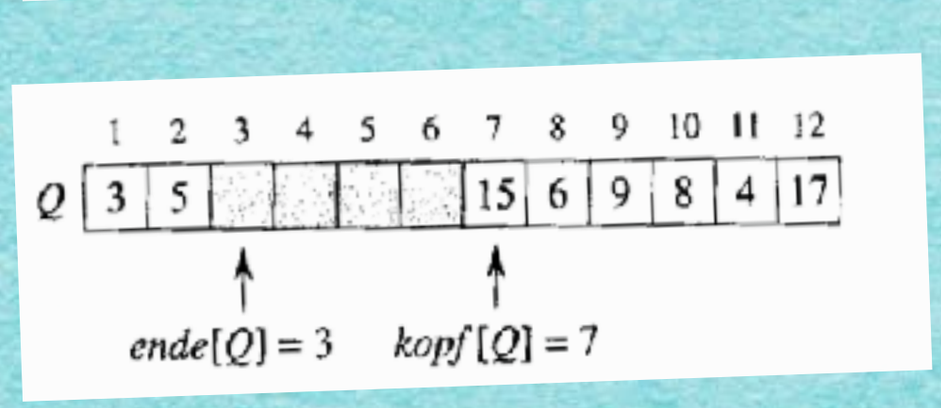


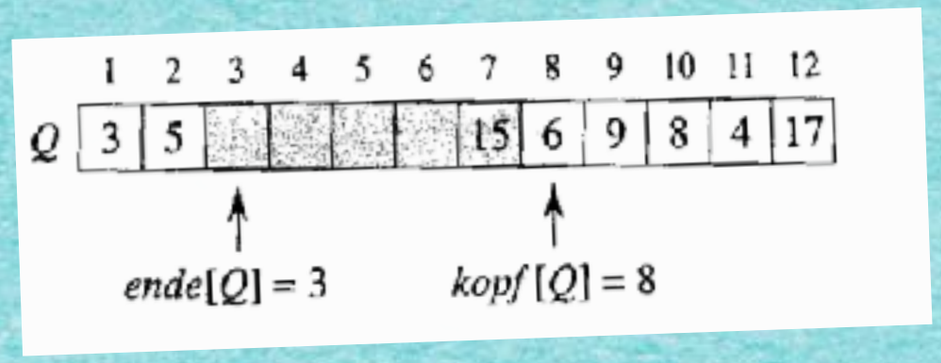

```
ENQUEUE(Q, x)
1  Q[ende[Q]] ← x
2  if ende[Q] = länge[Q]
3     then ende[Q] ← 1
4     else ende[Q] ← ende[Q] + 1
```

```
DEQUEUE(Q)
1  x ← Q[kopf[Q]]
2  if kopf[Q] = länge[Q]
3     then kopf[Q] ← 1
4     else kopf[Q] ← kopf[Q] + 1
5  return x
```

Warteschlange auf Array umgesetzt

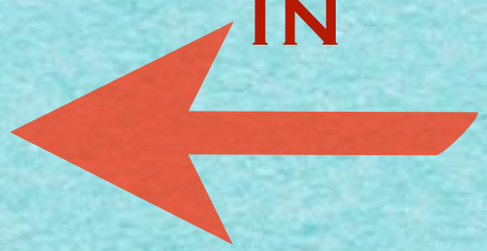




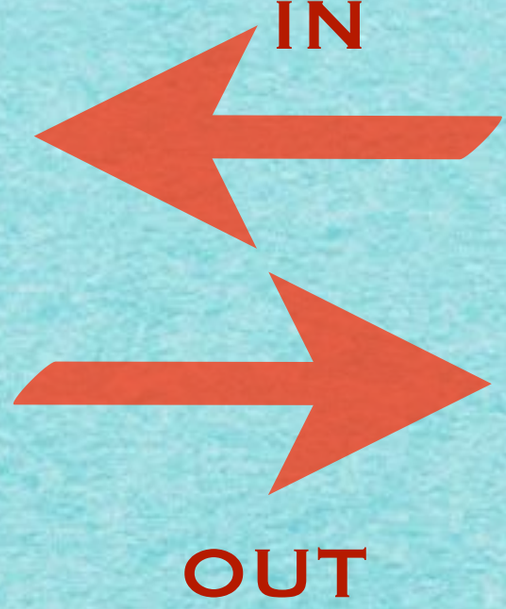


Stack auf Array umgesetzt

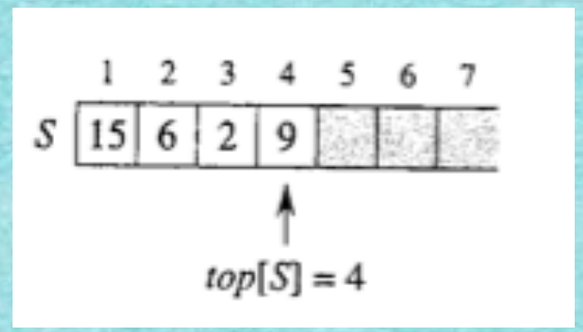
Stack
auf Array umgesetzt

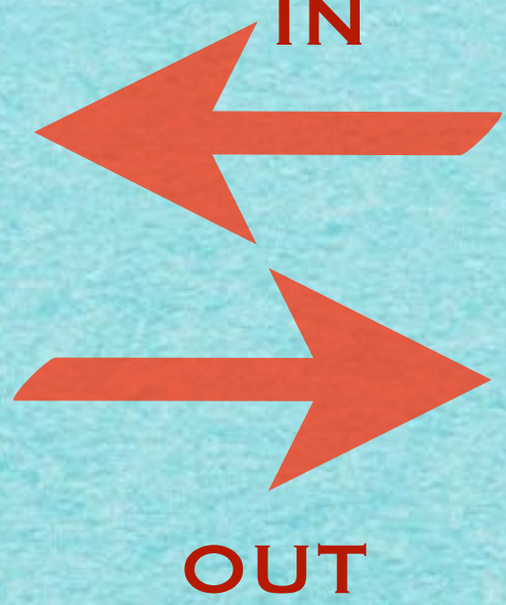


Stack
auf Array umgesetzt

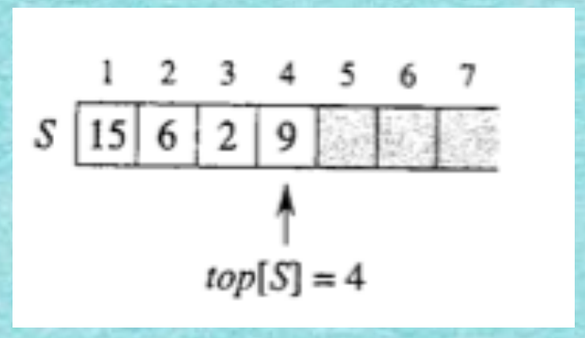


Stack
auf Array umgesetzt

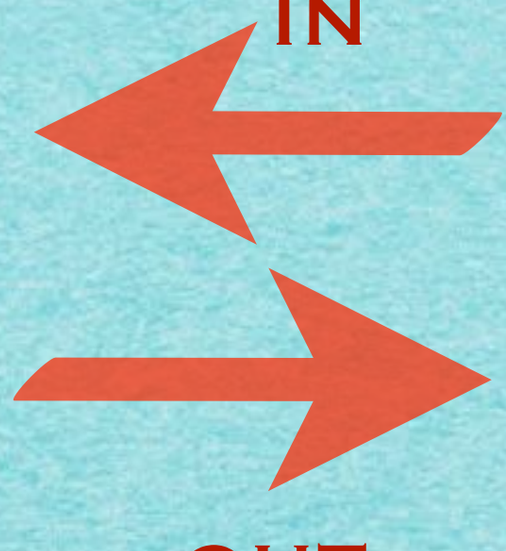




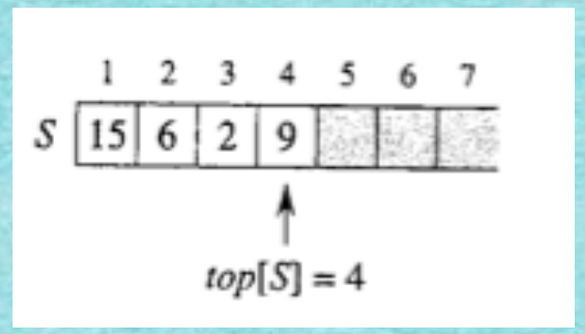
Stack
auf Array umgesetzt

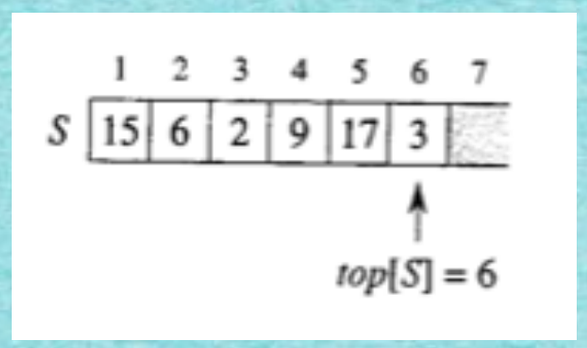


Push: 17, 3

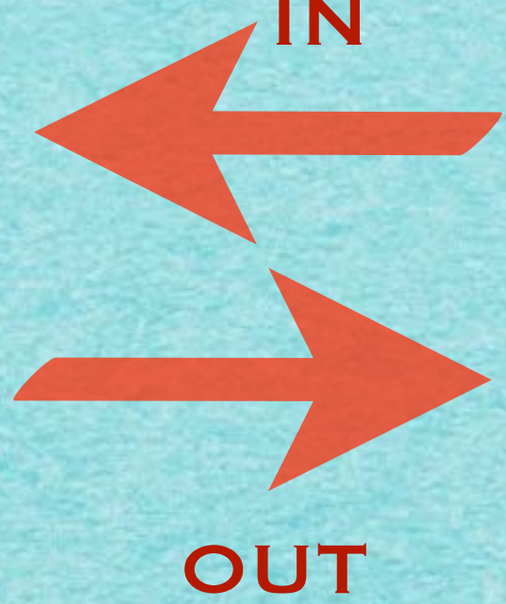


Stack
auf Array umgesetzt

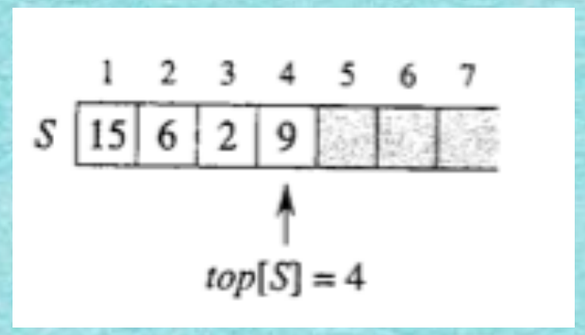


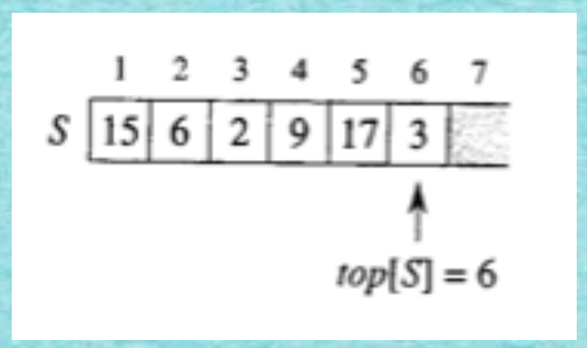


Push: 17, 3



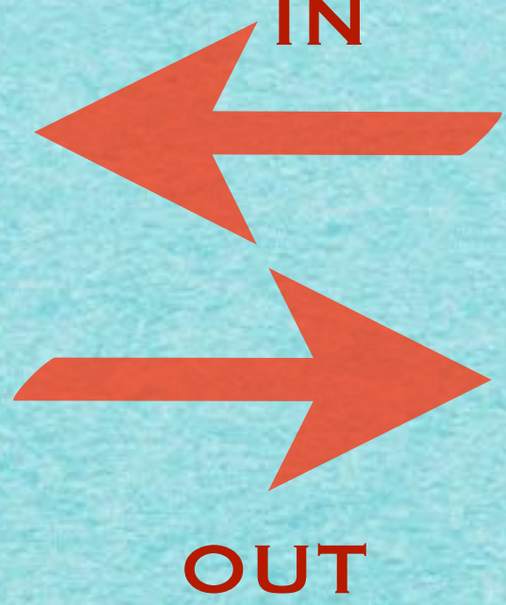
Stack
auf Array umgesetzt



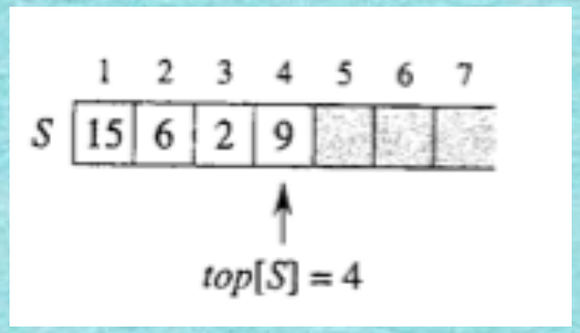


Push: 17, 3

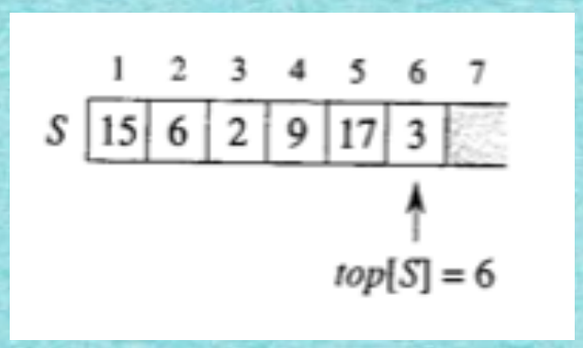
Pop



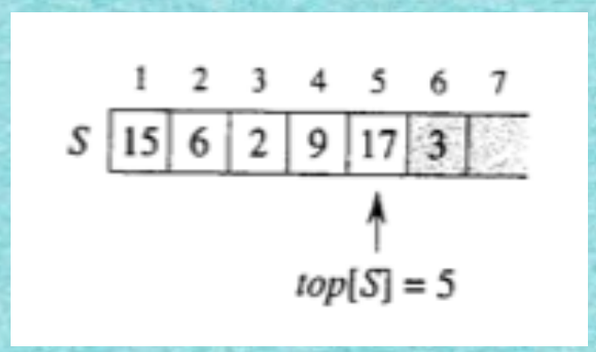
Stack
auf Array umgesetzt

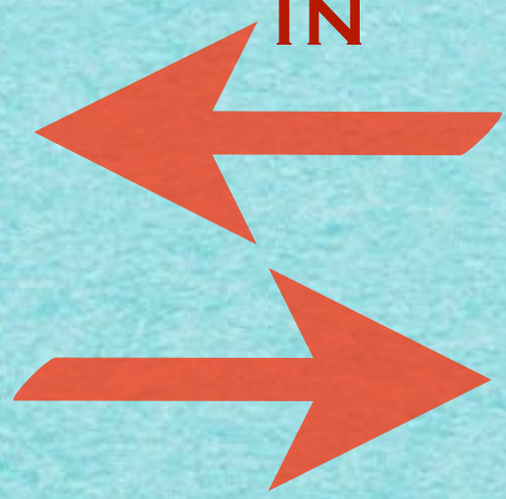


Push: 17, 3

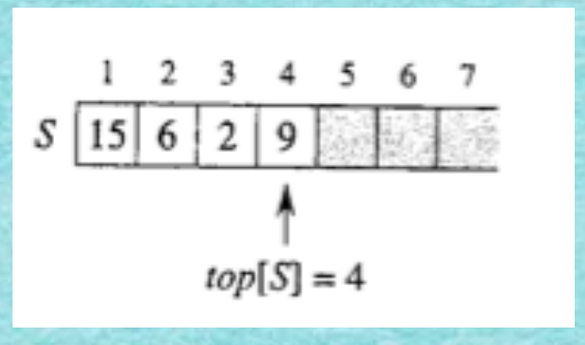


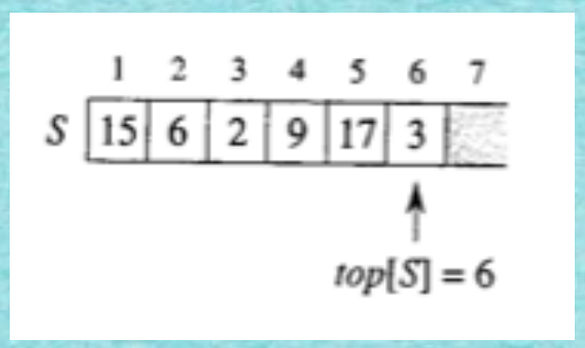
Pop

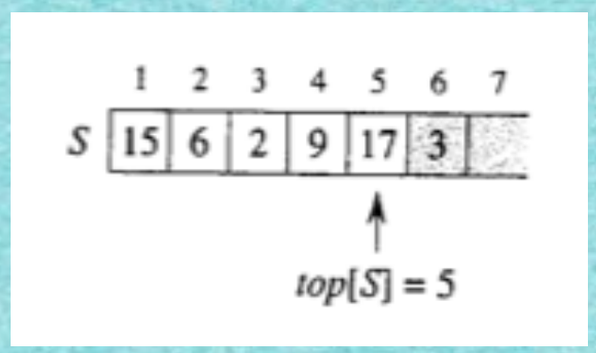


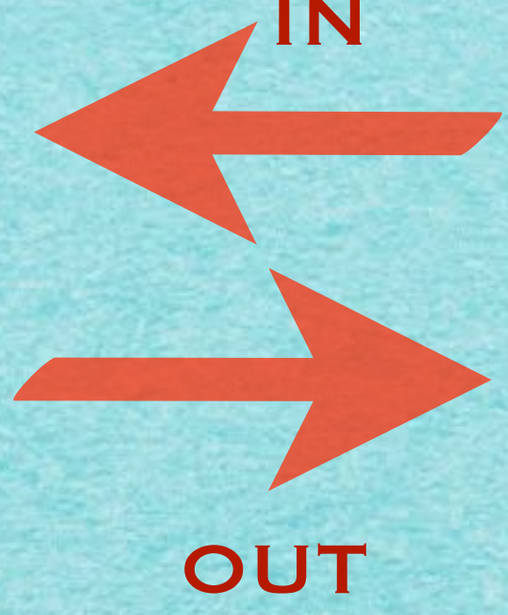


Stack
auf Array umgesetzt

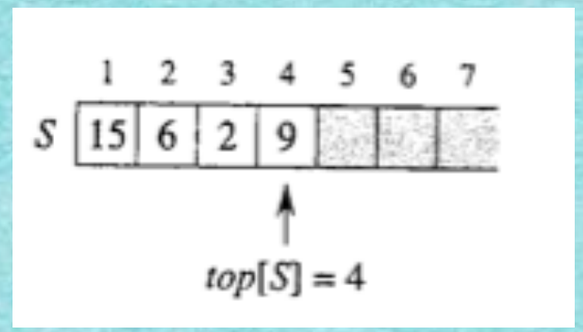




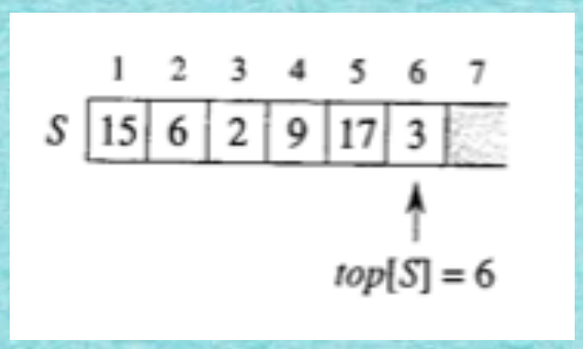


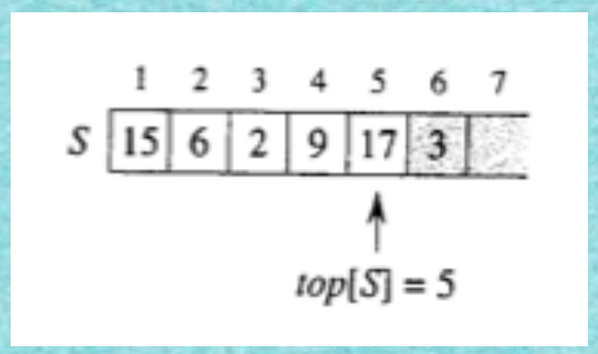


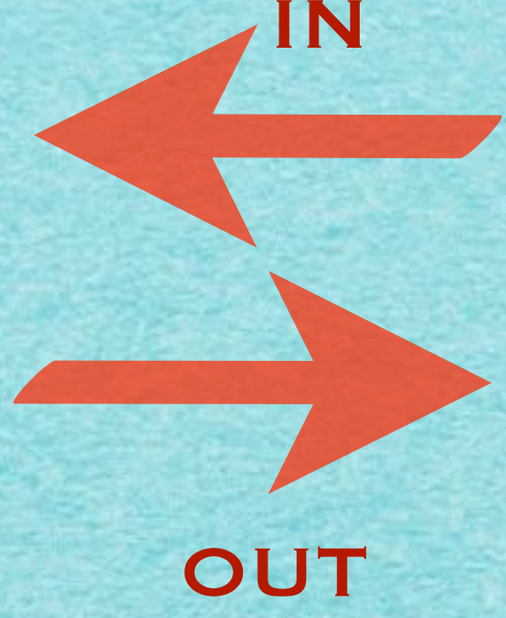
Stack auf Array umgesetzt



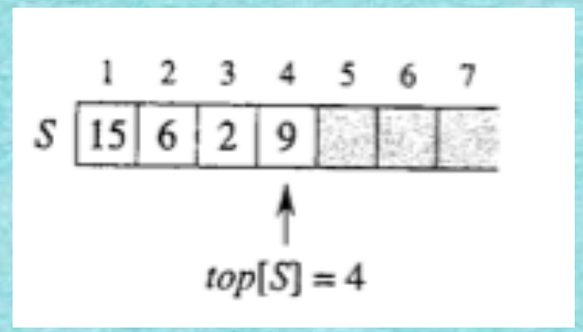

```
STACK-EMPTY(S)
1  if top[S] = 0
2      then return WAHR
3      else  return FALSCH
```



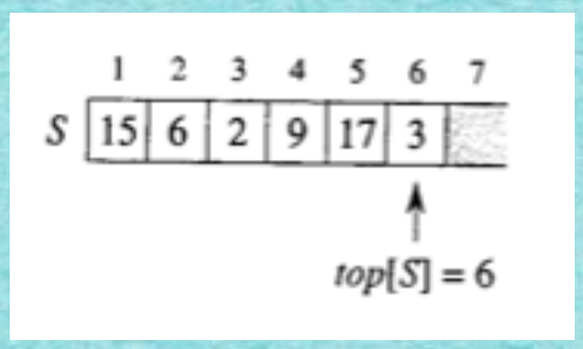




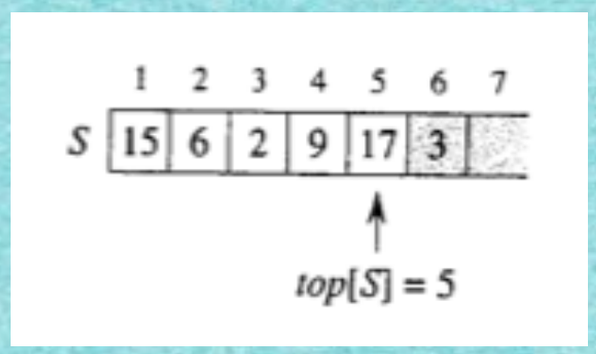
Stack auf Array umgesetzt

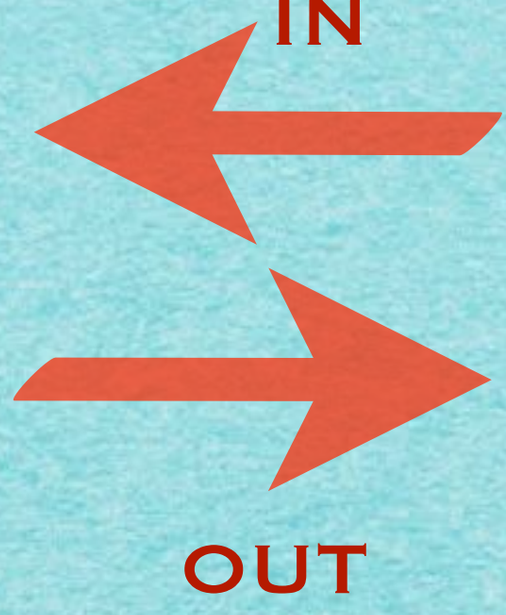



```
STACK-EMPTY(S)
1  if top[S] = 0
2      then return WAHR
3      else  return FALSCH
```

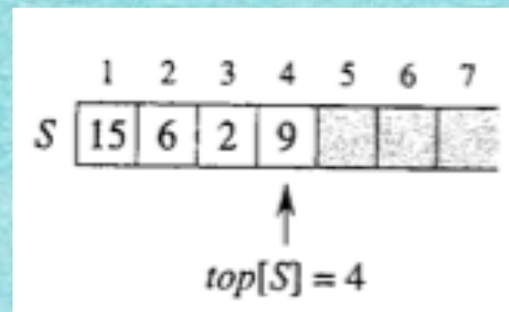



```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```

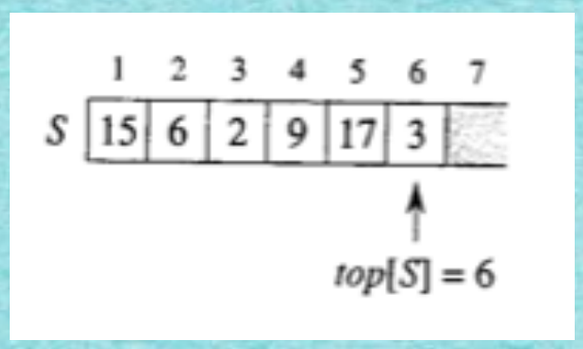




Stack auf Array umgesetzt



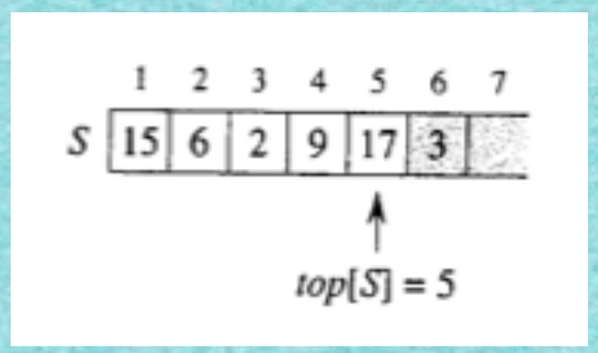

```
STACK-EMPTY(S)
1  if top[S] = 0
2      then return WAHR
3      else  return FALSCH
```




```
PUSH(S, x)
1  top[S] ← top[S] + 1
2  S[top[S]] ← x
```

IN

OUT




```
POP(S)
1  if STACK-EMPTY(S)
2      then error "Unterlauf"
3      else  top[S] ← top[S] − 1
4            return S[top[S] + 1]
```

4.3 Verkettete Listen

4.3 Verkettete Listen

Idee:

4.3 Verkettete Listen

Idee:

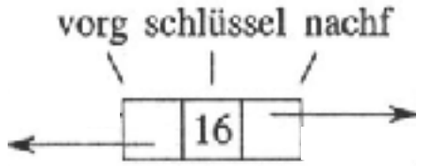
4.3 Verkettete Listen

Idee:

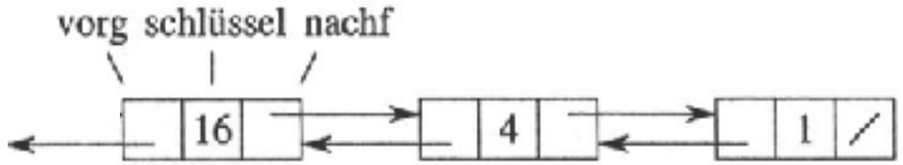
Ordne Objekte nicht explizit in aufeinanderfolgenden Speicherzellen an, sondern gib jeweils Vorgänger und Nachfolger an.

Struktur einer doppelt verketteten Liste

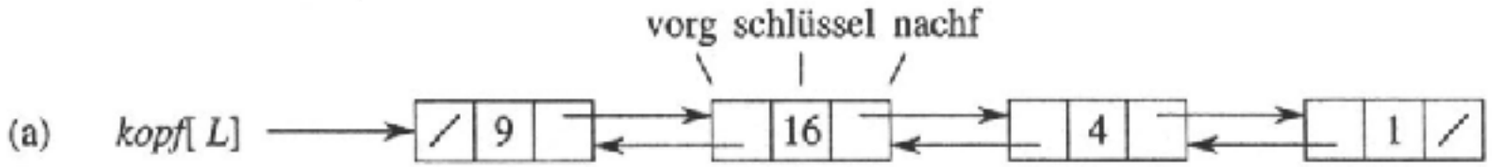
Struktur einer doppelt verketteten Liste



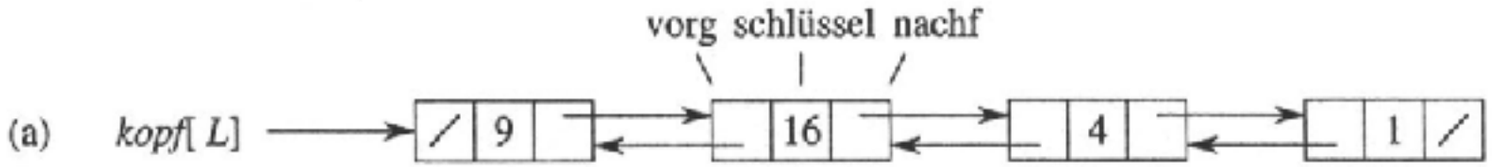
Struktur einer doppelt verketteten Liste



Struktur einer doppelt verketteten Liste

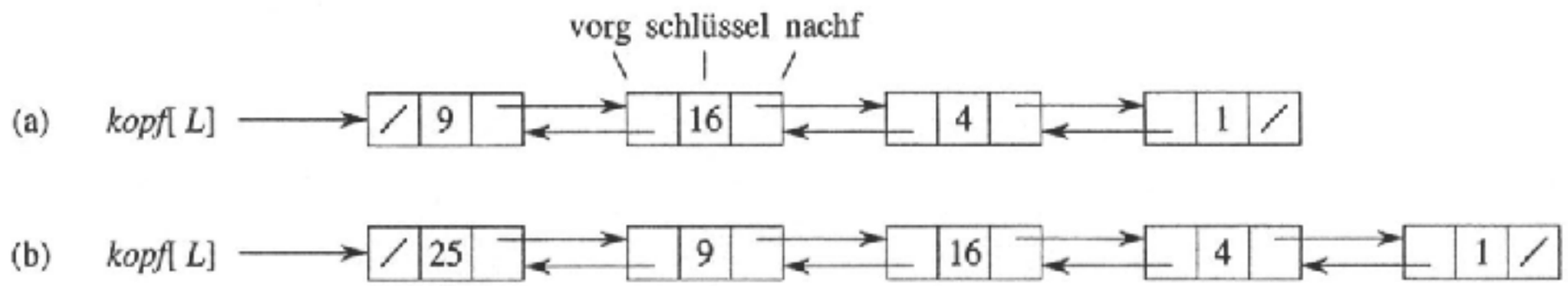


Struktur einer doppelt verketteten Liste



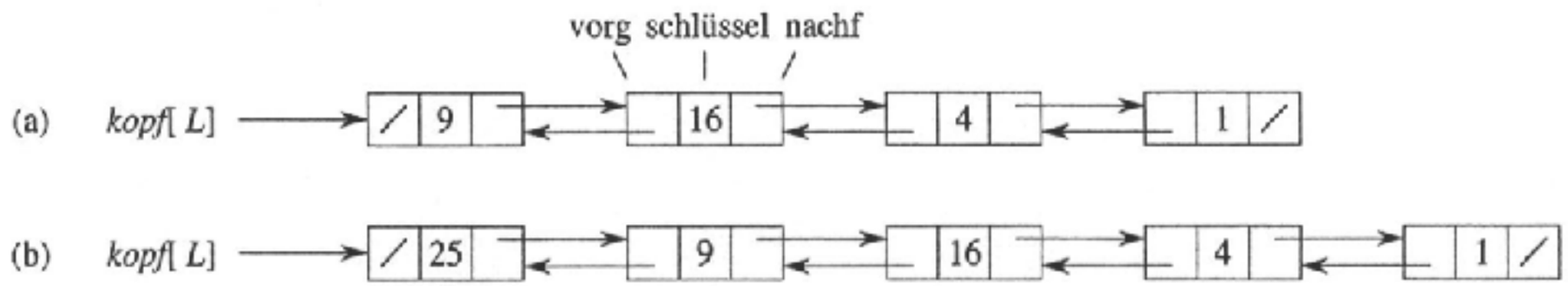
- **Füge vorne das Element mit Schlüssel 25 ein.**

Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

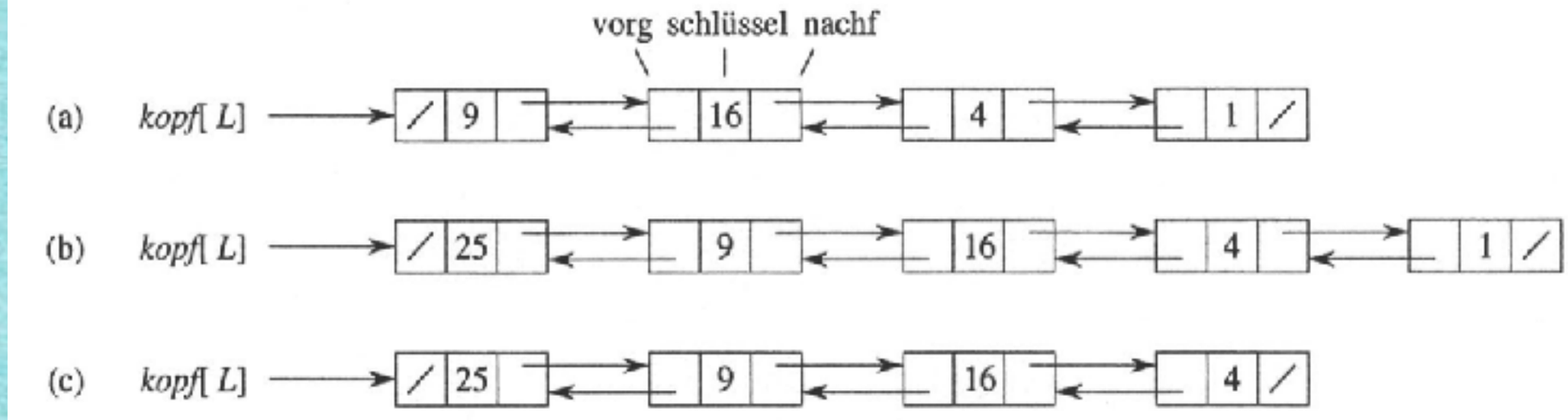
Struktur einer doppelt verketteten Liste



- **Füge vorne das Element mit Schlüssel 25 ein.**

- **Finde ein Element mit Schlüssel 1 und lösche es.**

Struktur einer doppelt verketteten Liste

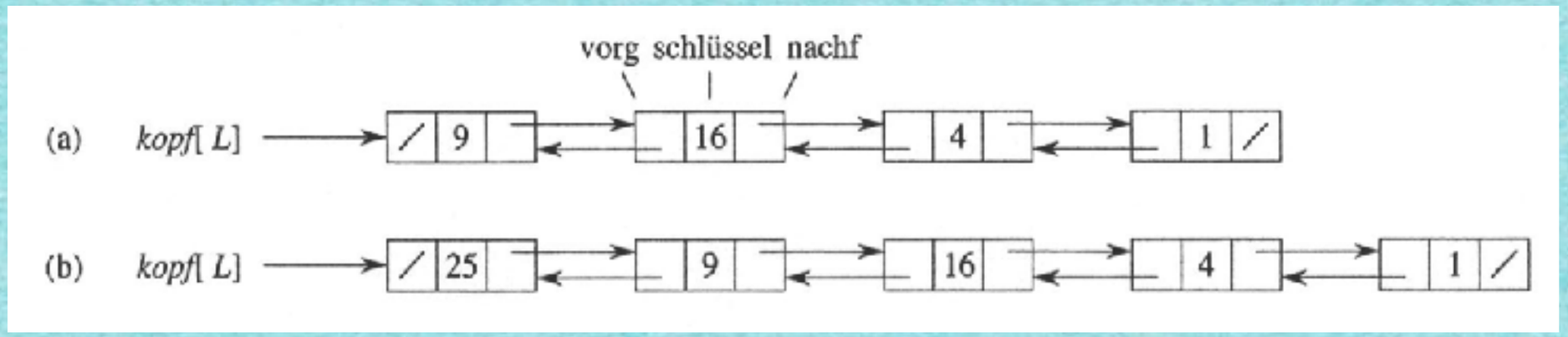


- **Füge vorne das Element mit Schlüssel 25 ein.**

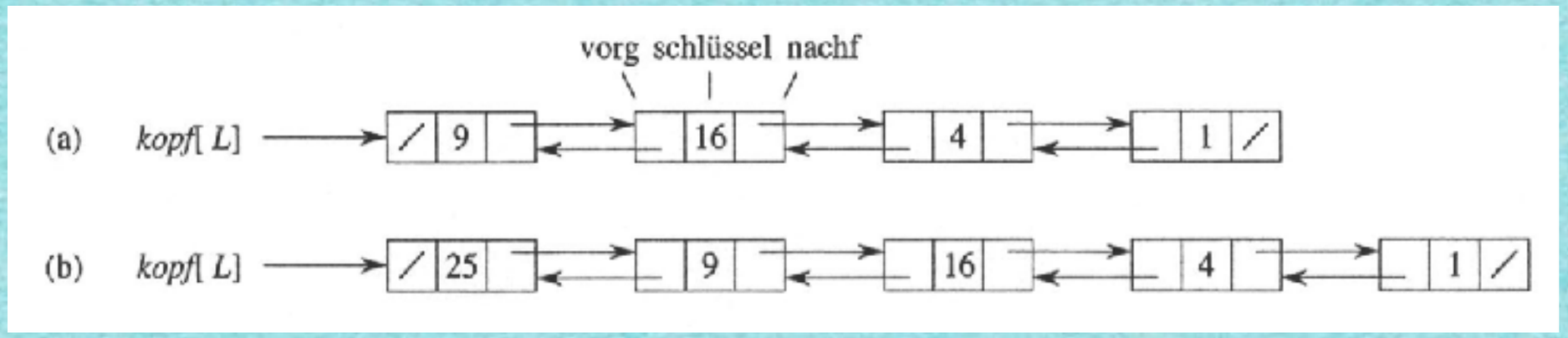
- **Finde ein Element mit Schlüssel 1 und lösche es.**

Einfügen in eine doppelt verkettete Liste

Einfügen in eine doppelt verkettete Liste



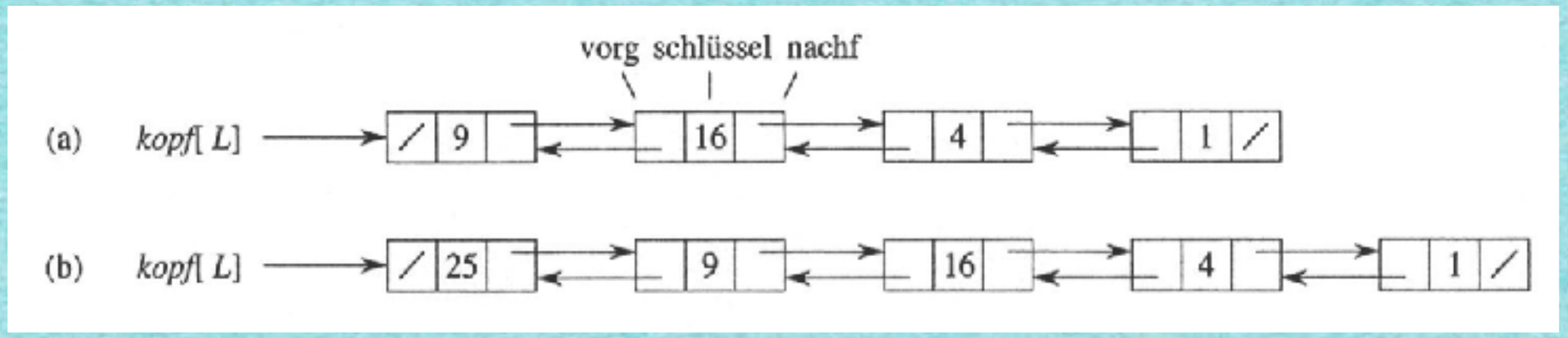
Einfügen in eine doppelt verkettete Liste



LIST-INSERT(L, x)

1 $nachf[x] \leftarrow kopf[L]$
2 **if** $kopf[L] \neq$ NIL
3 **then** $vorg[kopf[L]] \leftarrow x$
4 $kopf[L] \leftarrow x$
5 $vorg[x] \leftarrow$ NIL

Einfügen in eine doppelt verkettete Liste

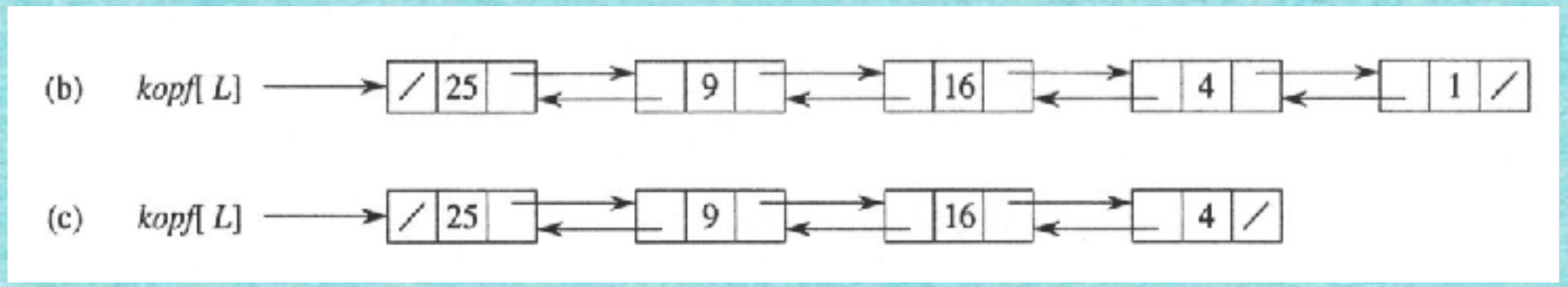



```
LIST-INSERT(L, x)
1  nachf[x] ← kopf[L]
2  if kopf[L] ≠ NIL
3      then vorg[kopf[L]] ← x
4  kopf[L] ← x
5  vorg[x] ← NIL
```

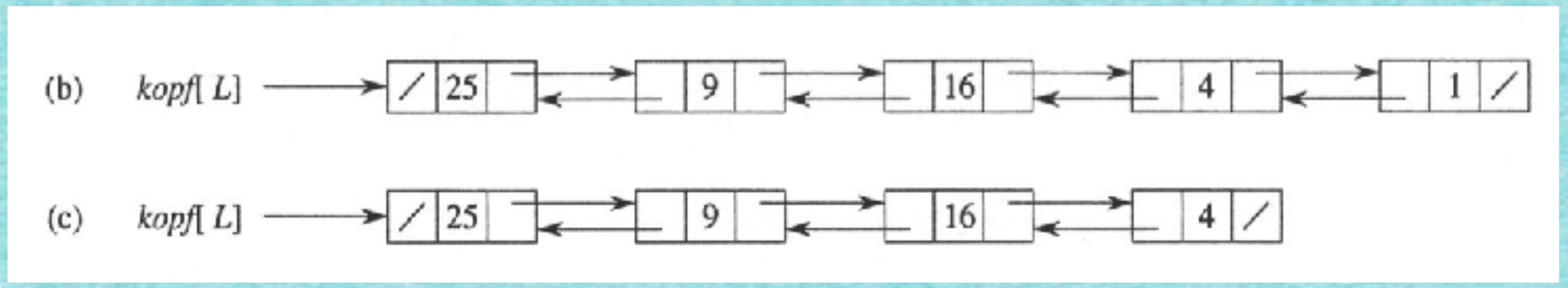
<u>Laufzeit:</u> O(1)

Löschen aus einer doppelt verketteten Liste

Löschen aus einer doppelt verketteten Liste



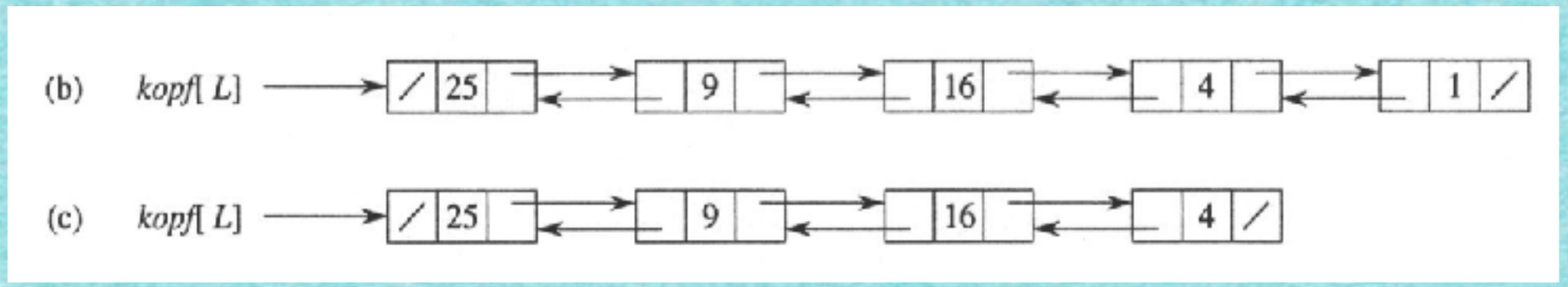
Löschen aus einer doppelt verketteten Liste



LIST-SEARCH(L, k)

1 $x \leftarrow kopf[L]$
2 **while** $x \neq$ NIL und $schlüssel[x] \neq k$
3 **do** $x \leftarrow nachf[x]$
4 **return** x

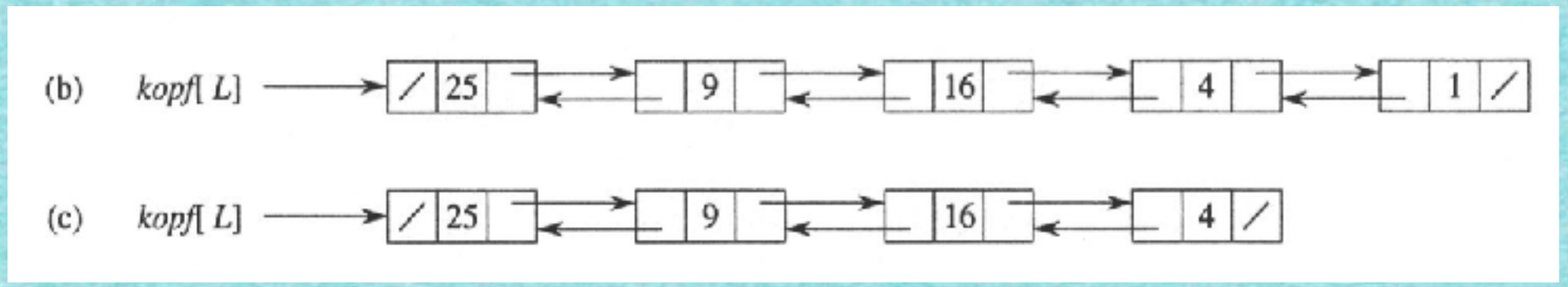
Löschen aus einer doppelt verketteten Liste




```
LIST-SEARCH(L, k)
1  x ← kopf[L]
2  while x ≠ NIL und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

Laufzeit: O(n)

Löschen aus einer doppelt verketteten Liste



<u>Laufzeit:</u> O(n)

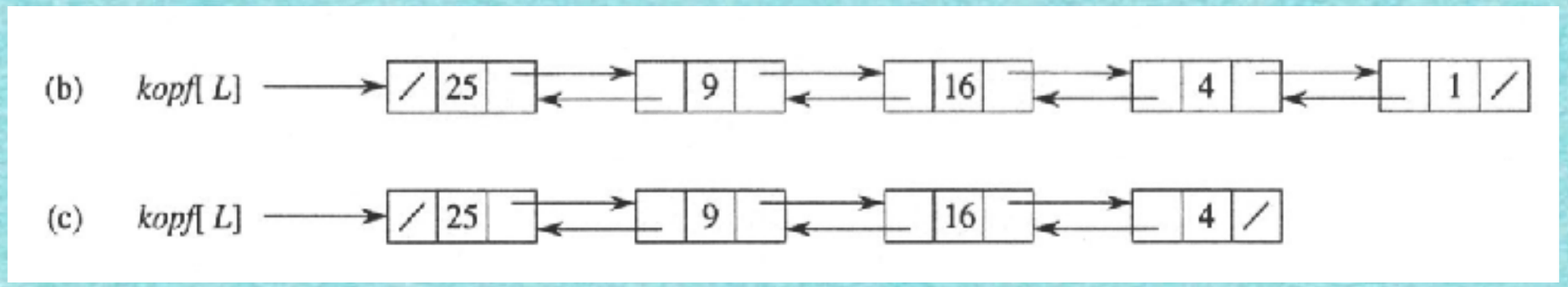
LIST-SEARCH(L, k)

```
1  x ← kopf[L]
2  while x ≠ NIL und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

LIST-DELETE(L, x)

```
1  if vorg[x] ≠ NIL
2     then nachf[vorg[x]] ← nachf[x]
3     else  kopf[L] ← nachf[x]
4  if nachf[x] ≠ NIL
5     then vorg[nachf[x]] ← vorg[x]
```

Löschen aus einer doppelt verketteten Liste



Laufzeit: O(n)

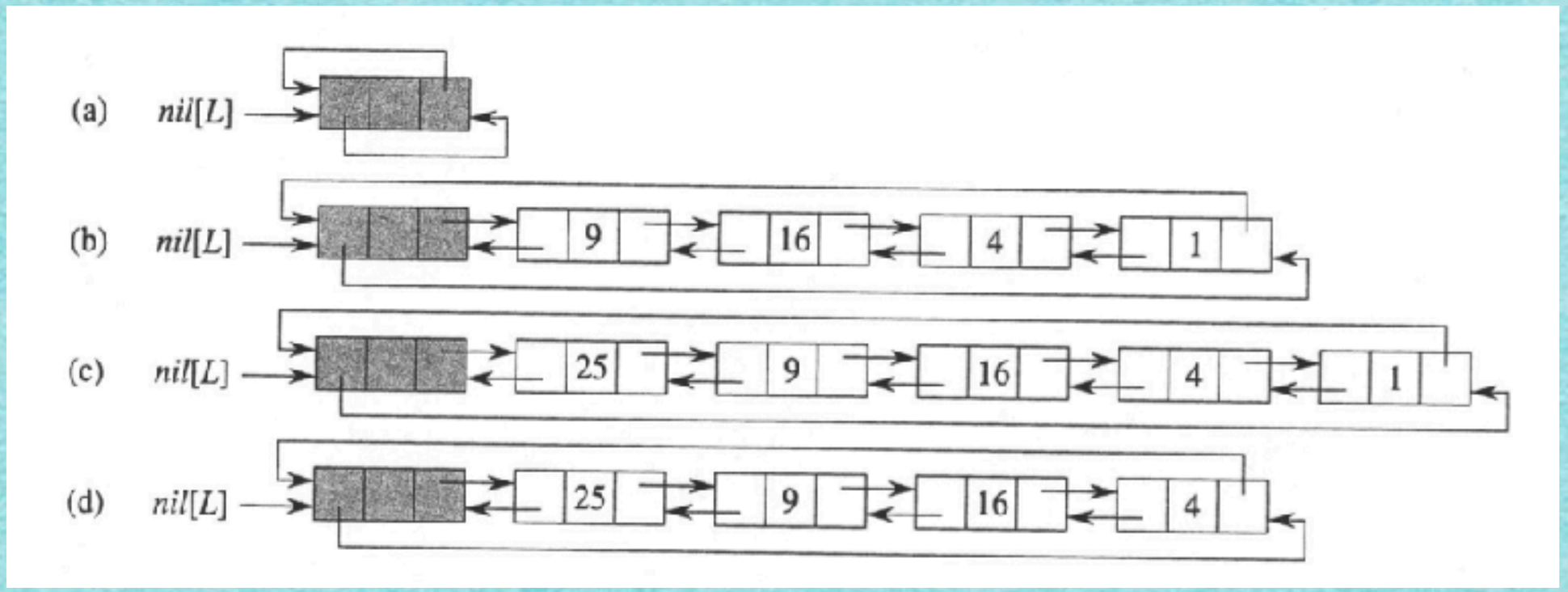
```
LIST-SEARCH(L, k)
1  x ← kopf[L]
2  while x ≠ NIL und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

Laufzeit: O(1)

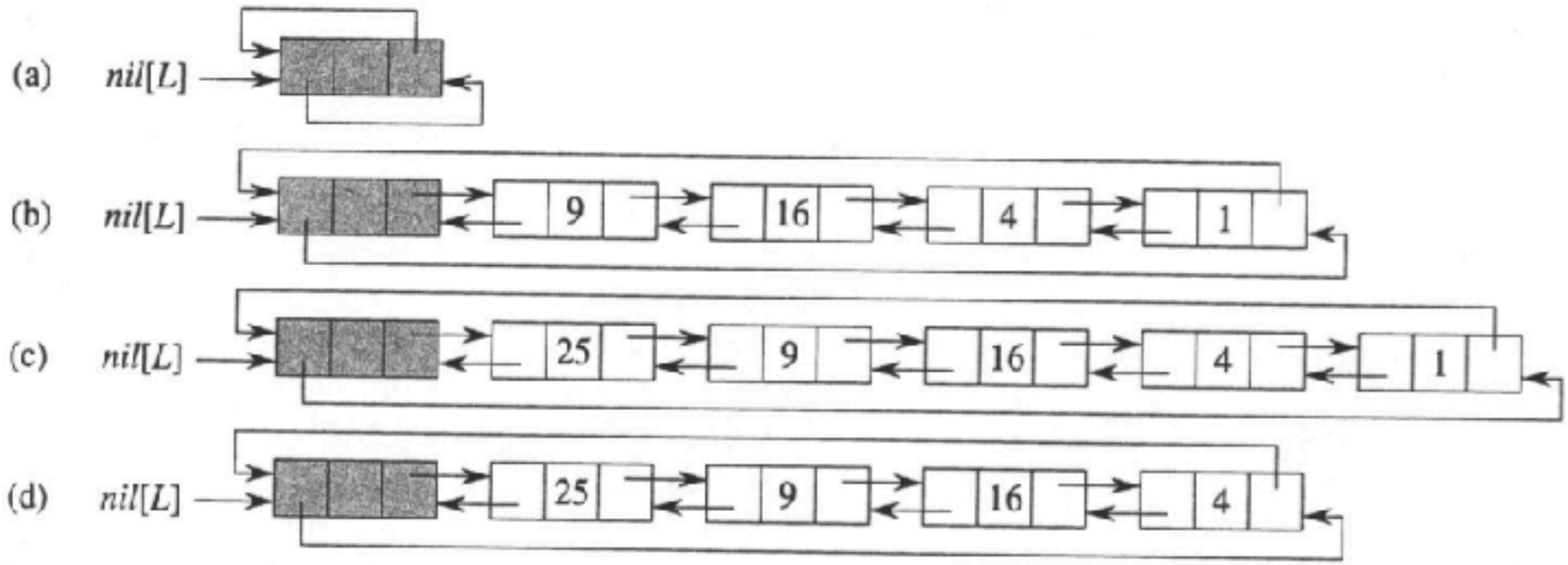
```
LIST-DELETE(L, x)
1  if vorg[x] ≠ NIL
2     then nachf[vorg[x]] ← nachf[x]
3     else kopf[L] ← nachf[x]
4  if nachf[x] ≠ NIL
5     then vorg[nachf[x]] ← vorg[x]
```

Alternative: Zyklische Struktur mit "Wächter" nil[L]

Alternative: Zyklische Struktur mit "Wächter" nil[L]



Alternative: Zyklische Struktur mit "Wächter" nil[L]

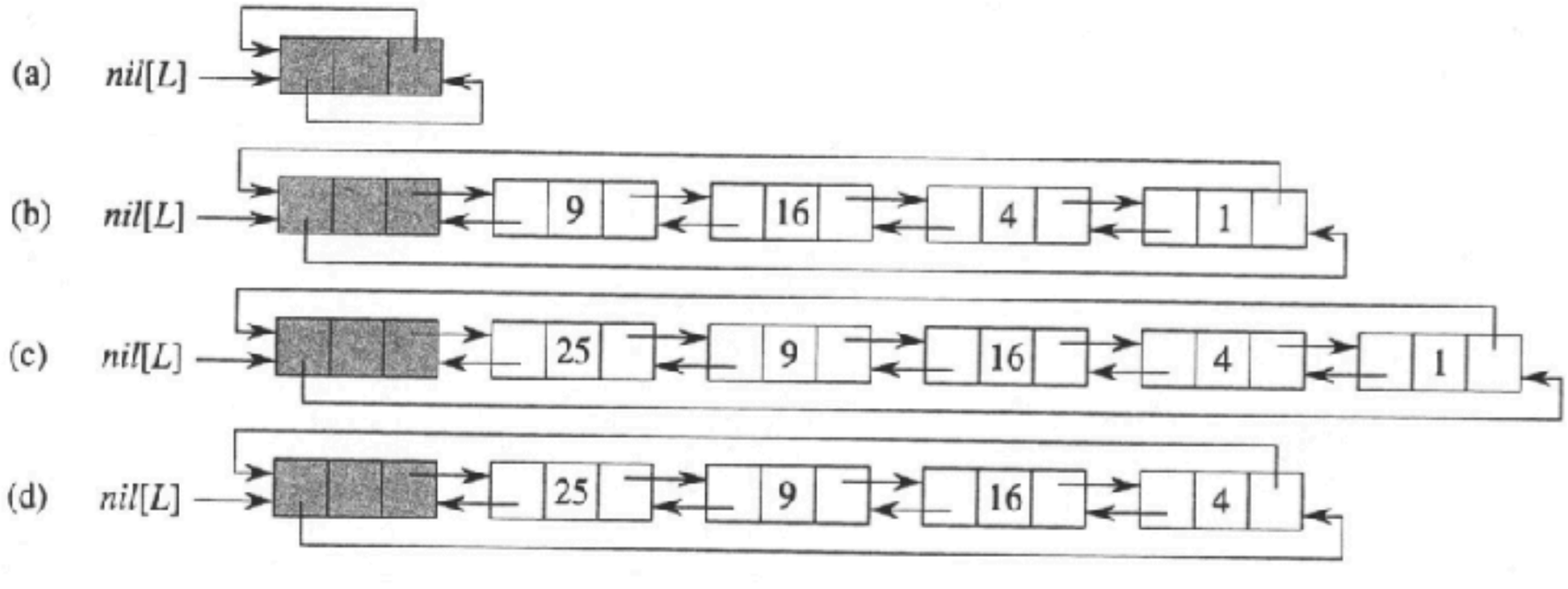


LIST-INSERT′(L, x)

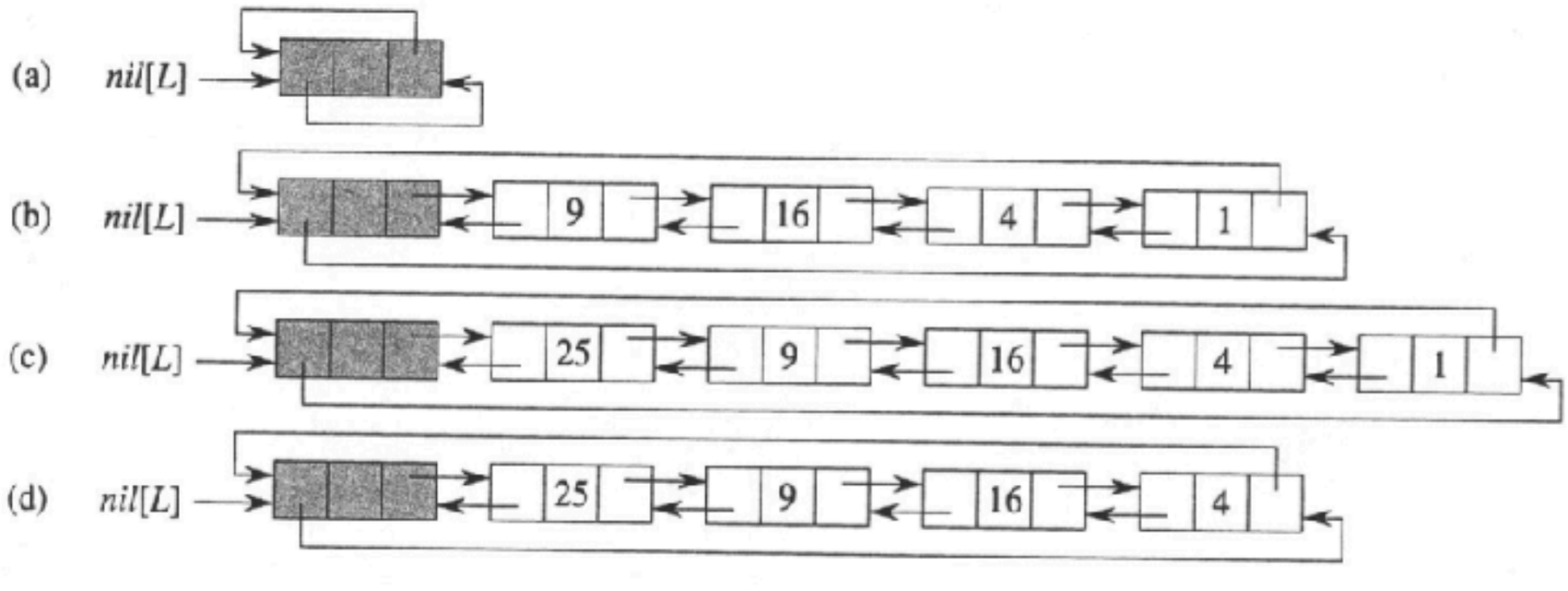
1 $nachf[x] \leftarrow nachf[nil[L]]$
2 $vorg[nachf[nil[L]]] \leftarrow x$
3 $nachf[nil[L]] \leftarrow x$
4 $vorg[x] \leftarrow nil[L]$

Alternative: Zyklische Struktur mit Wächter "nil[LL]"

Alternative: Zyklische Struktur mit Wächter "nil[L]"

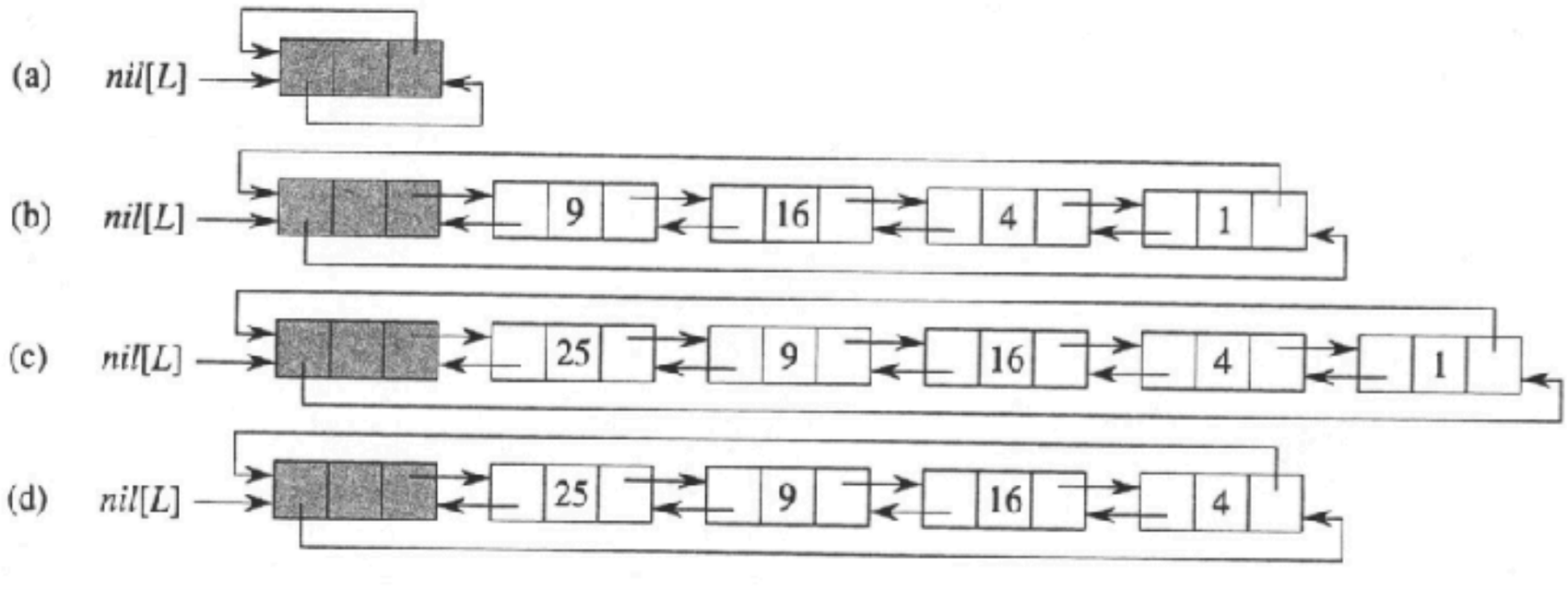


Alternative: Zyklische Struktur mit Wächter "nil[L]"




```
LIST-SEARCH'(L, k)
1  x ← nachf[nil[L]]
2  while x ≠ nil[L] und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

Alternative: Zyklische Struktur mit Wächter "nil[L]"

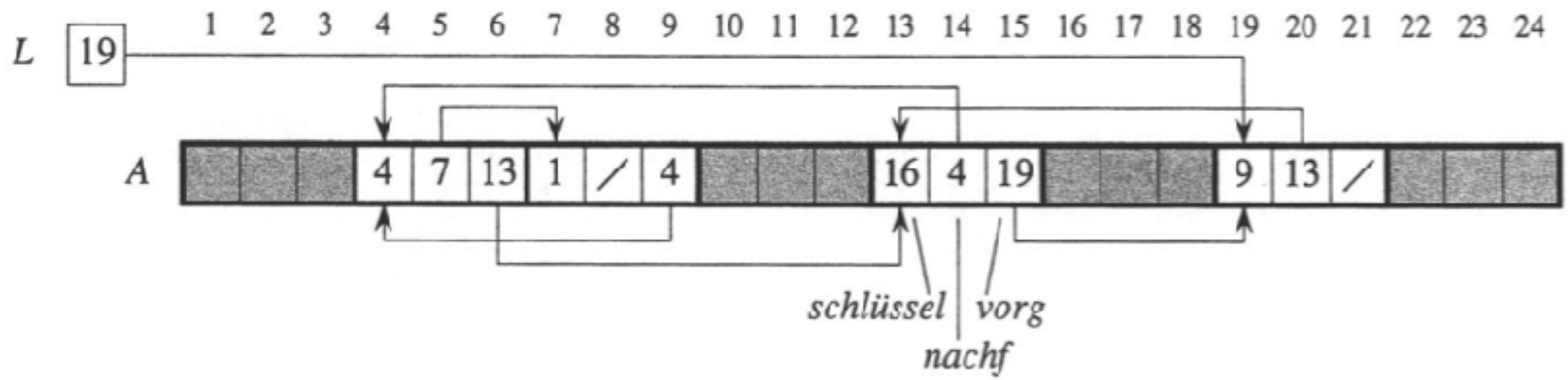



```
LIST-SEARCH'(L, k)
1  x ← nachf[nil[L]]
2  while x ≠ nil[L] und schlüssel[x] ≠ k
3        do x ← nachf[x]
4  return x
```

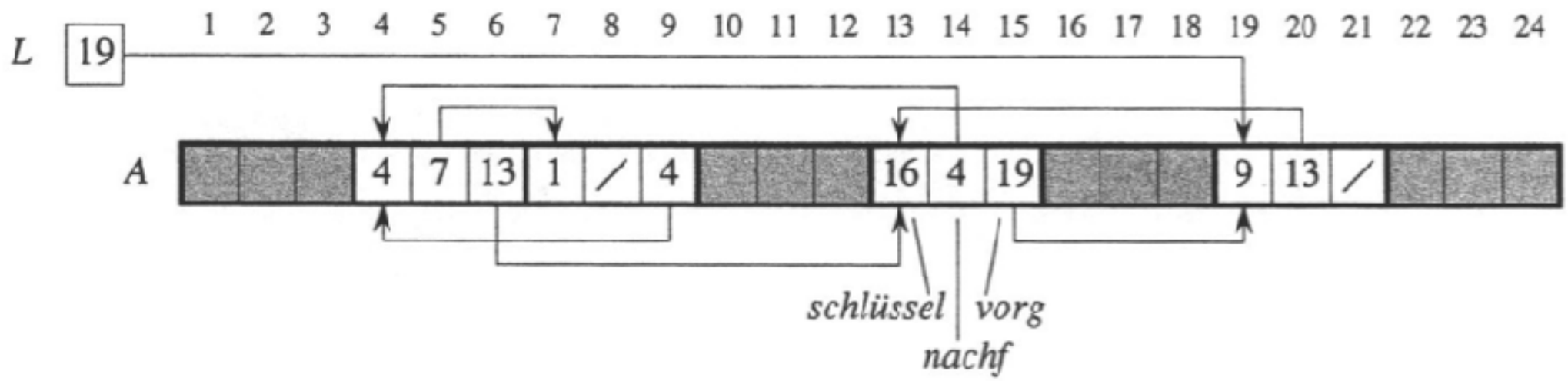
```
LIST-DELETE'(L, x)
1  nachf[vorg[x]] ← nachf[x]
2  vorg[nachf[x]] ← vorg[x]
```

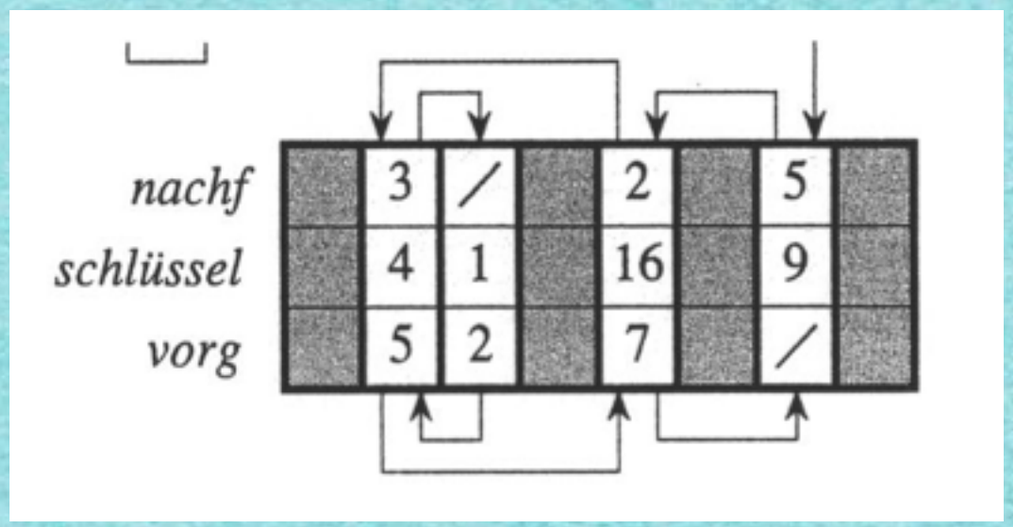
Speicherung kann irgendwo erfolgen!

Speicherung kann irgendwo erfolgen!



Speicherung kann irgendwo erfolgen!





4.4 Binäre Suche

4.4 Binäre Suche

Aufgabenstellung:

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

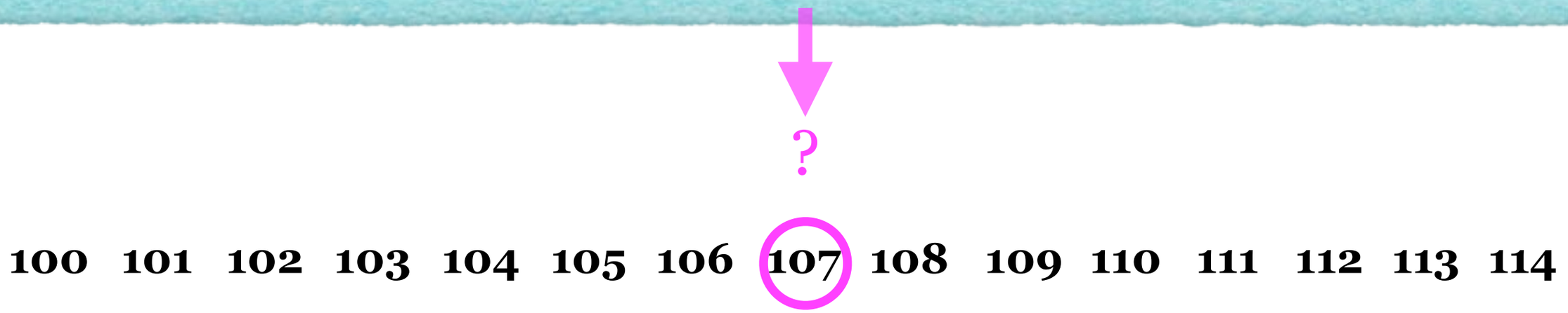
- ***Rate eine Zahl zwischen 100 und 114!***

100 101 102 103 104 105 106 107 108 109 110 111 112 113 114

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

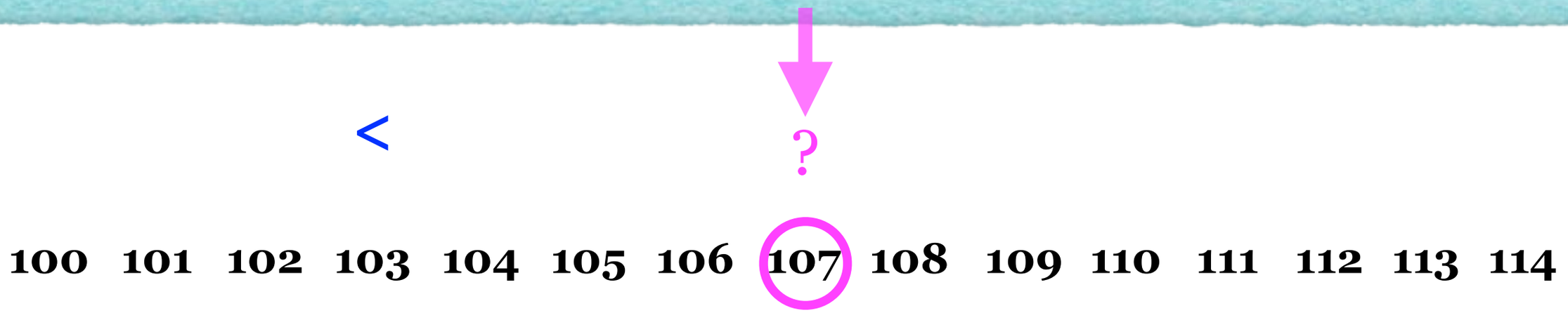
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

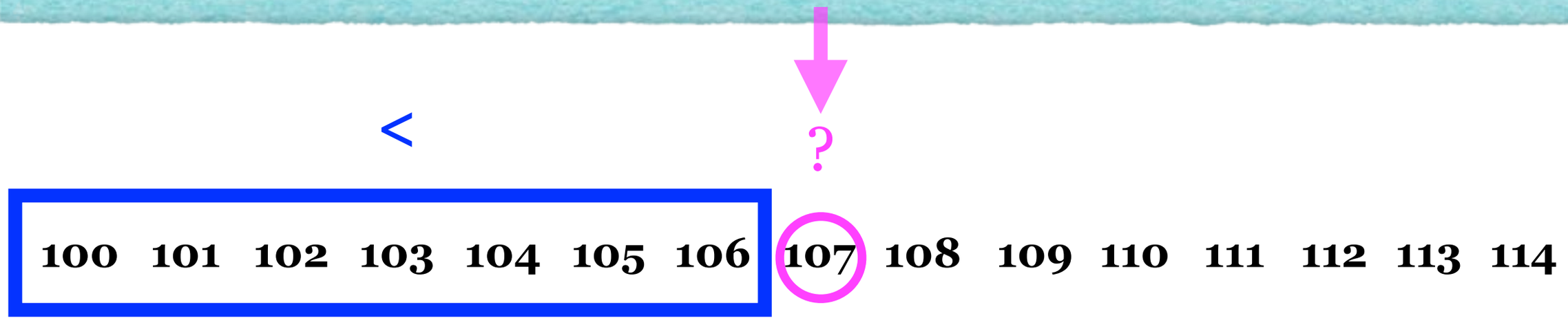
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

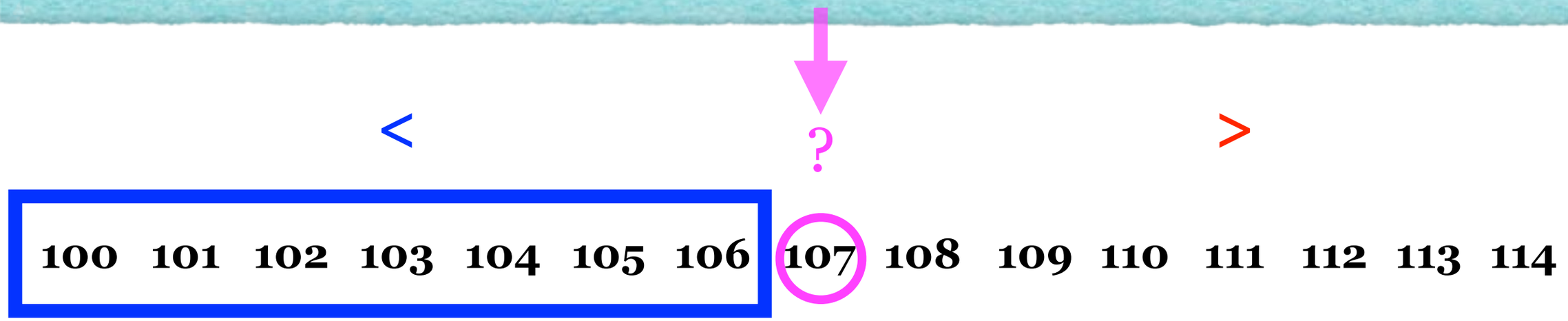
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

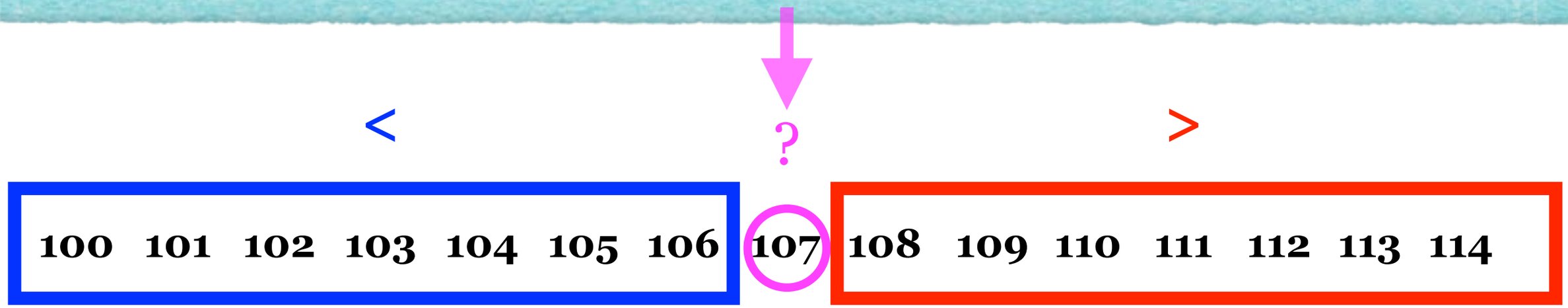
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

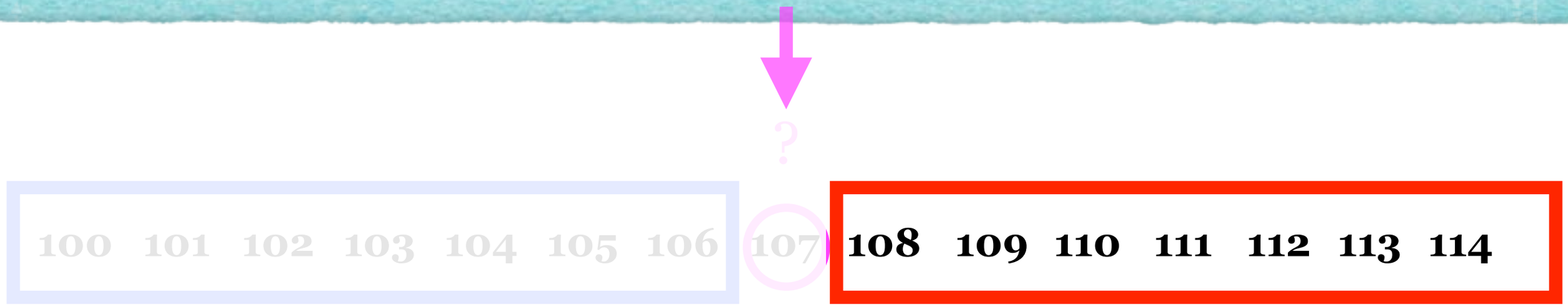
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

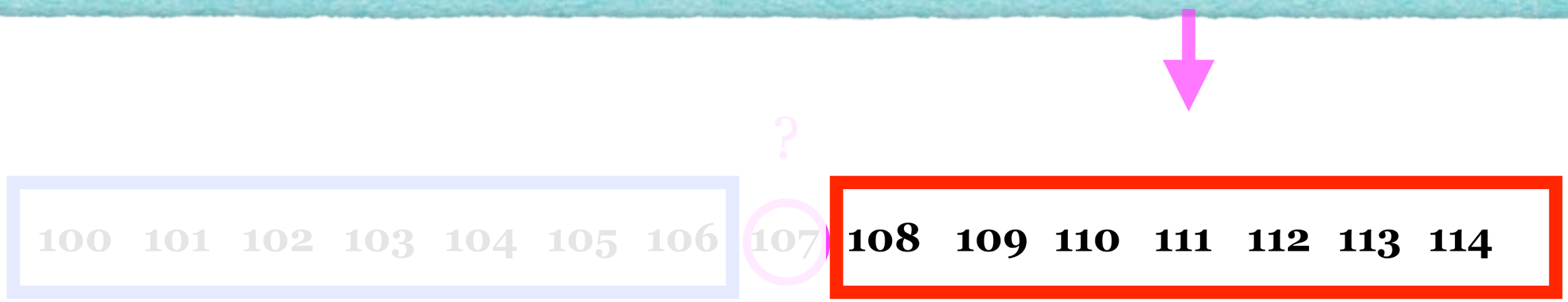
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

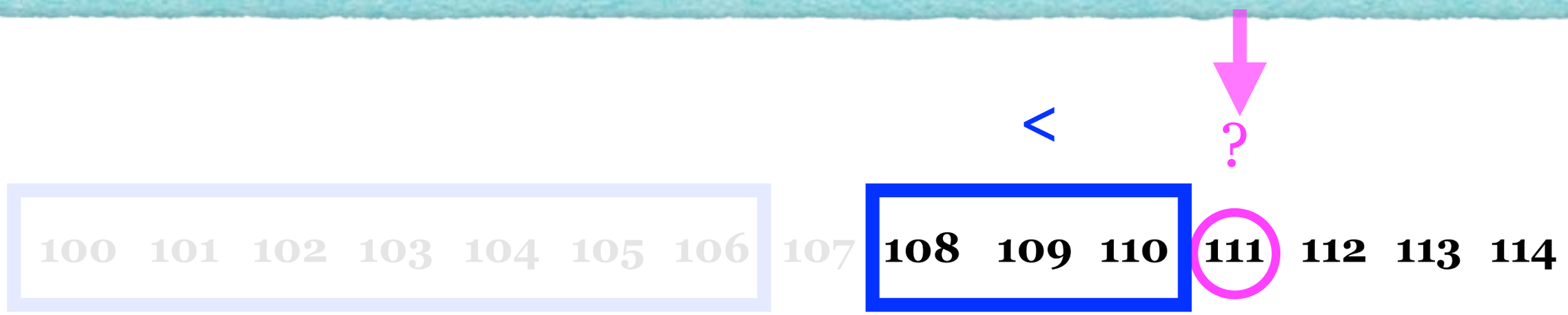
<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

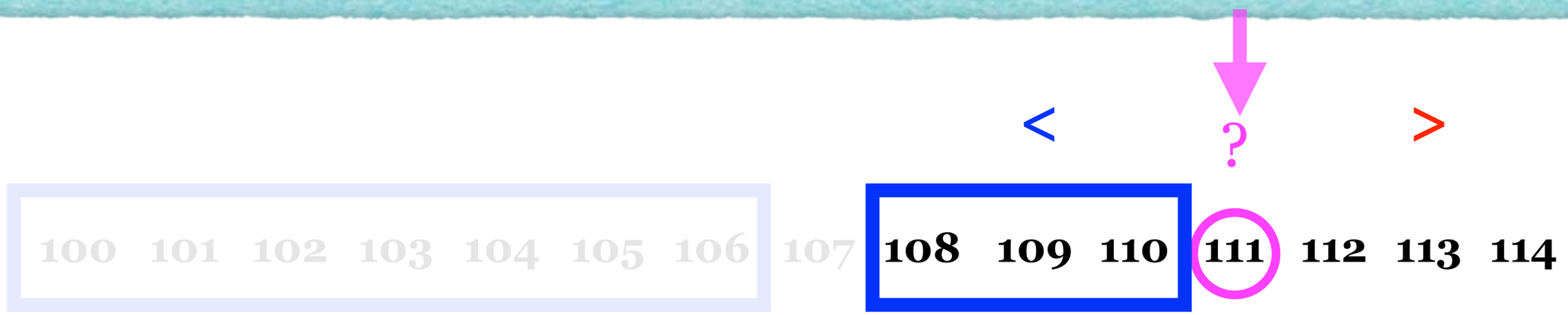
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

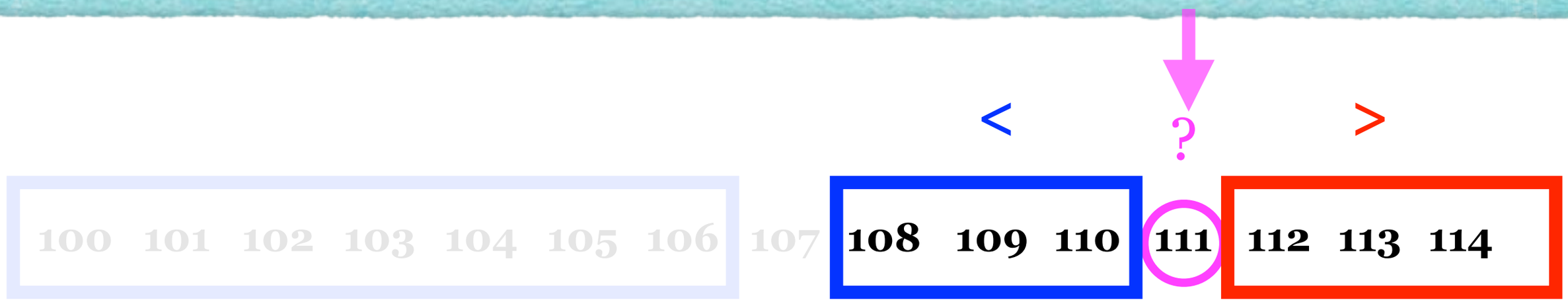
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

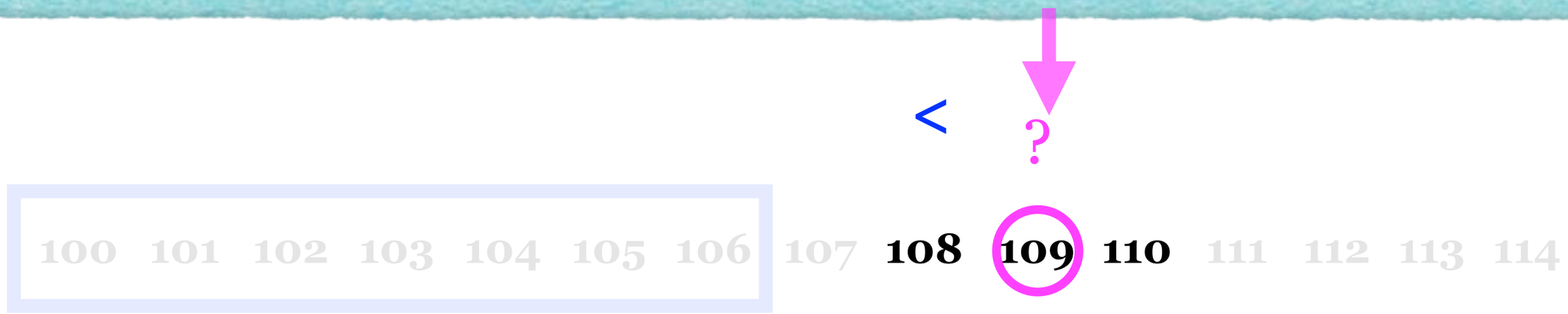
<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

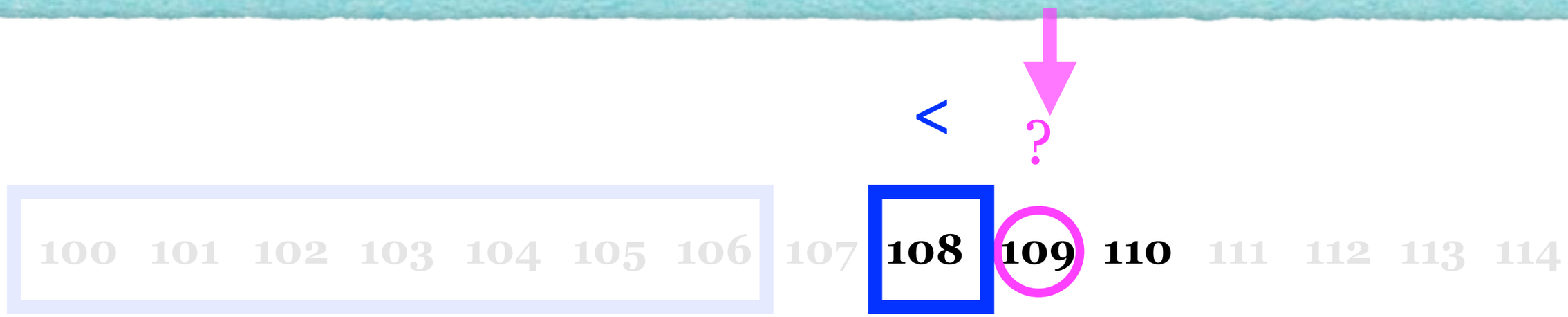
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

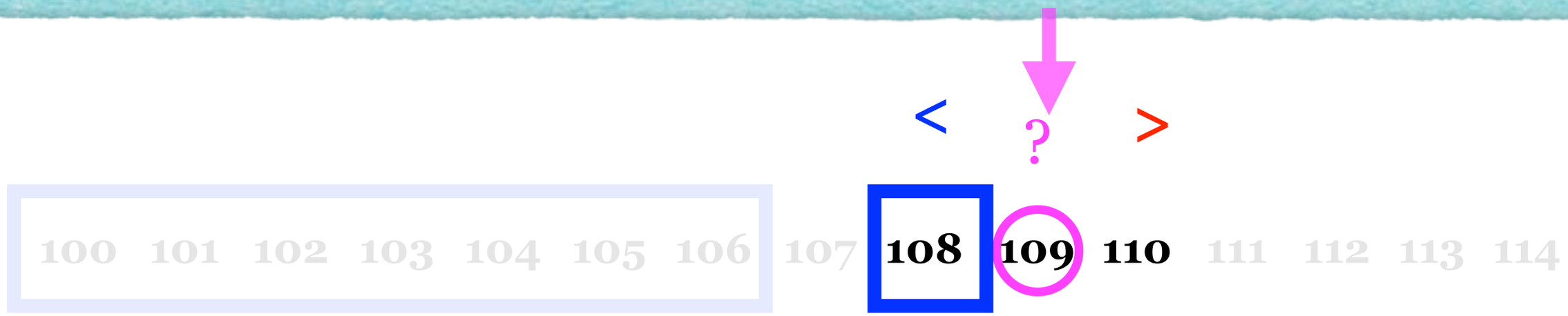
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

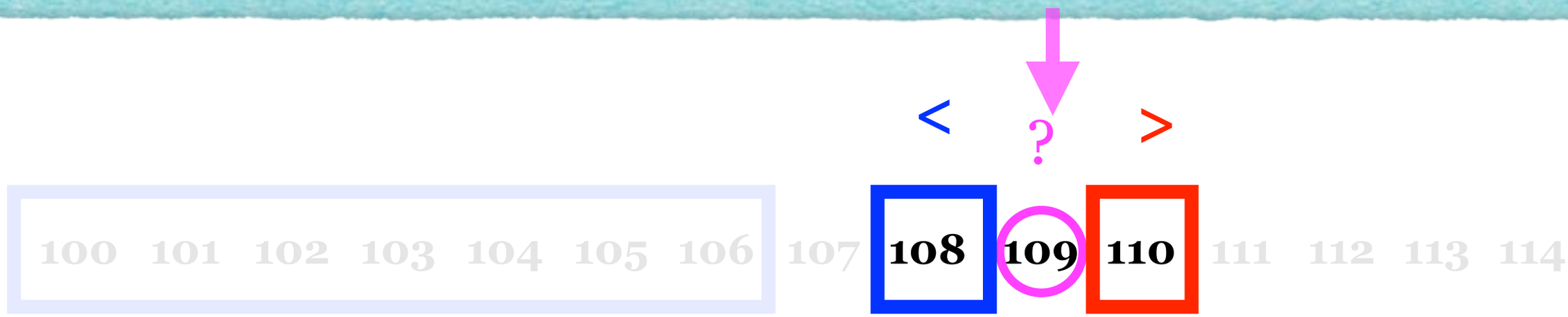
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

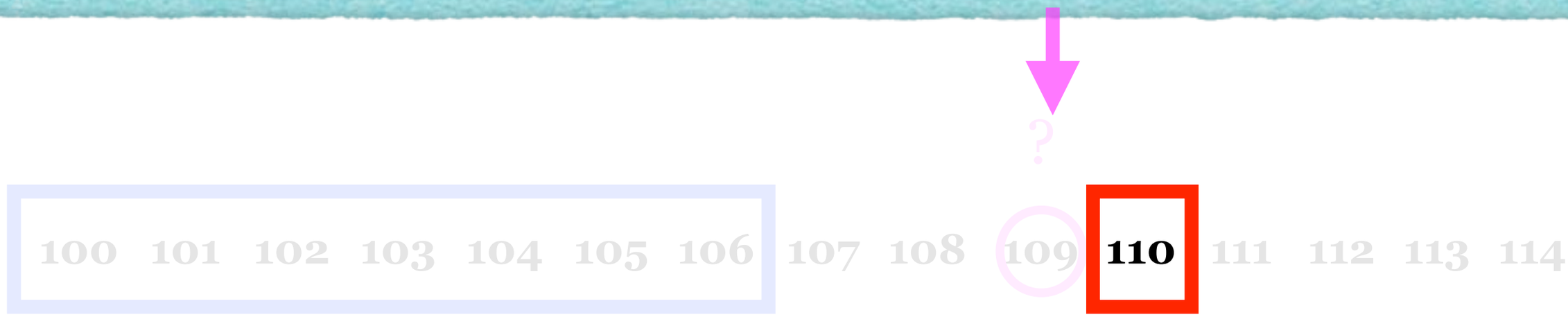
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

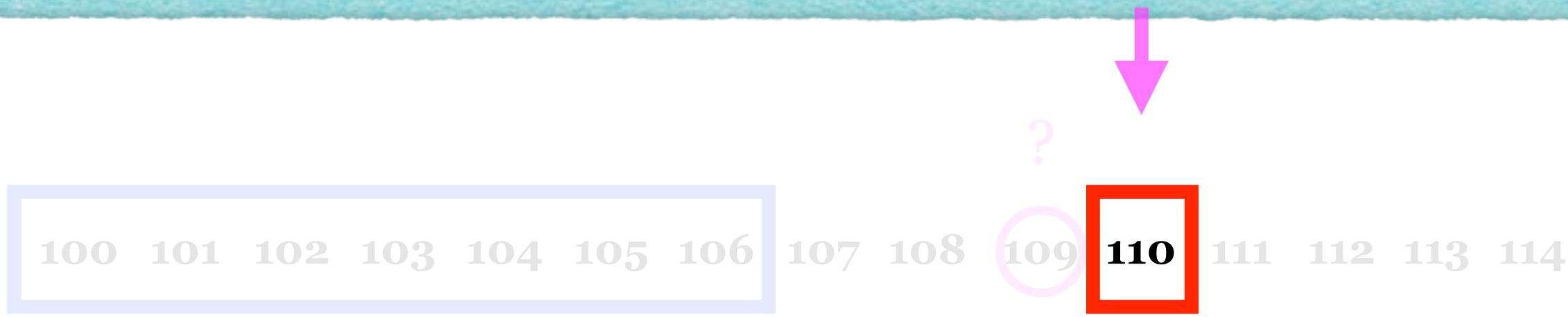
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

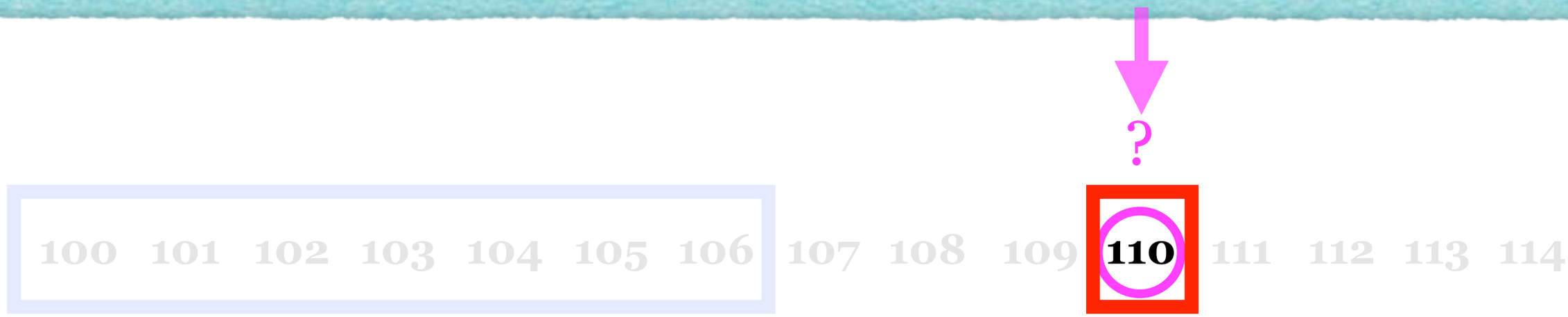
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

4.4 Binäre Suche

Aufgabenstellung:

4.4 Binäre Suche

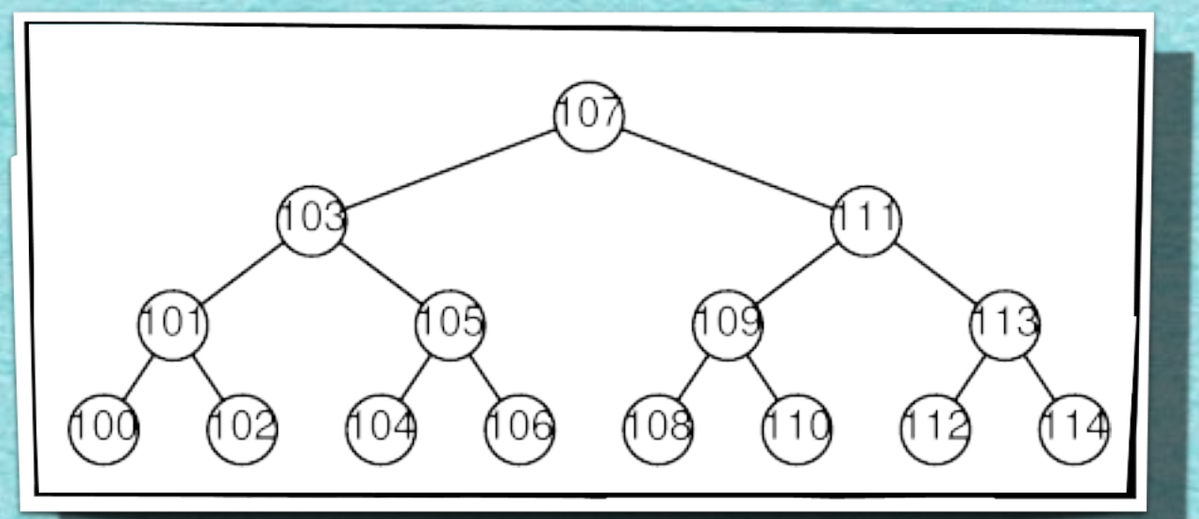
<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

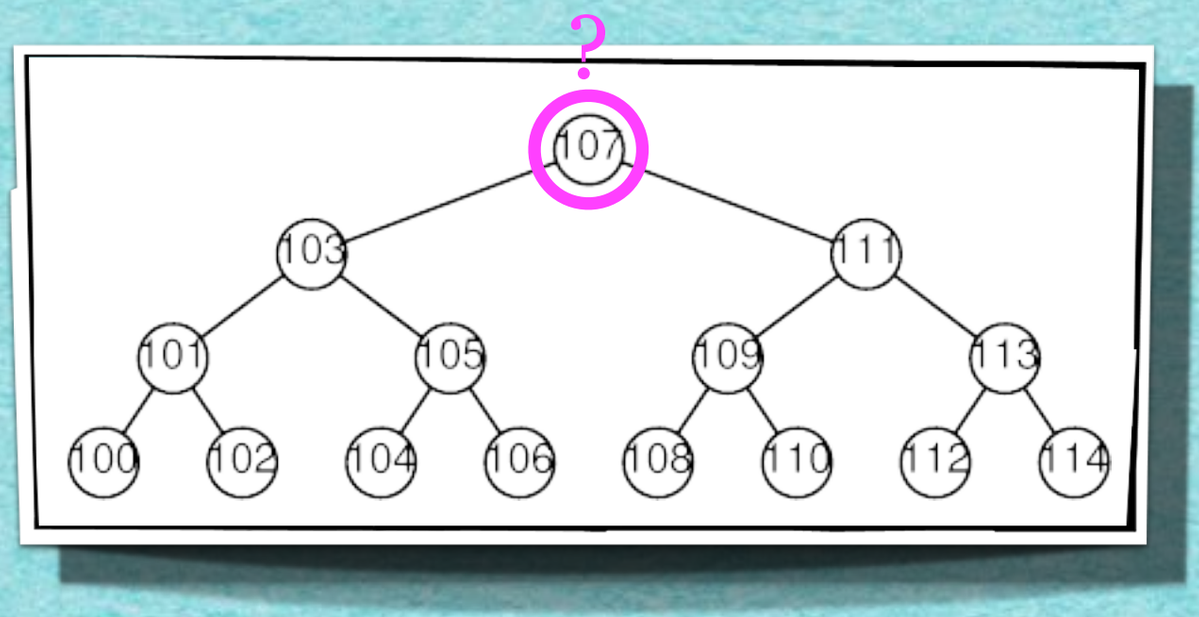
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

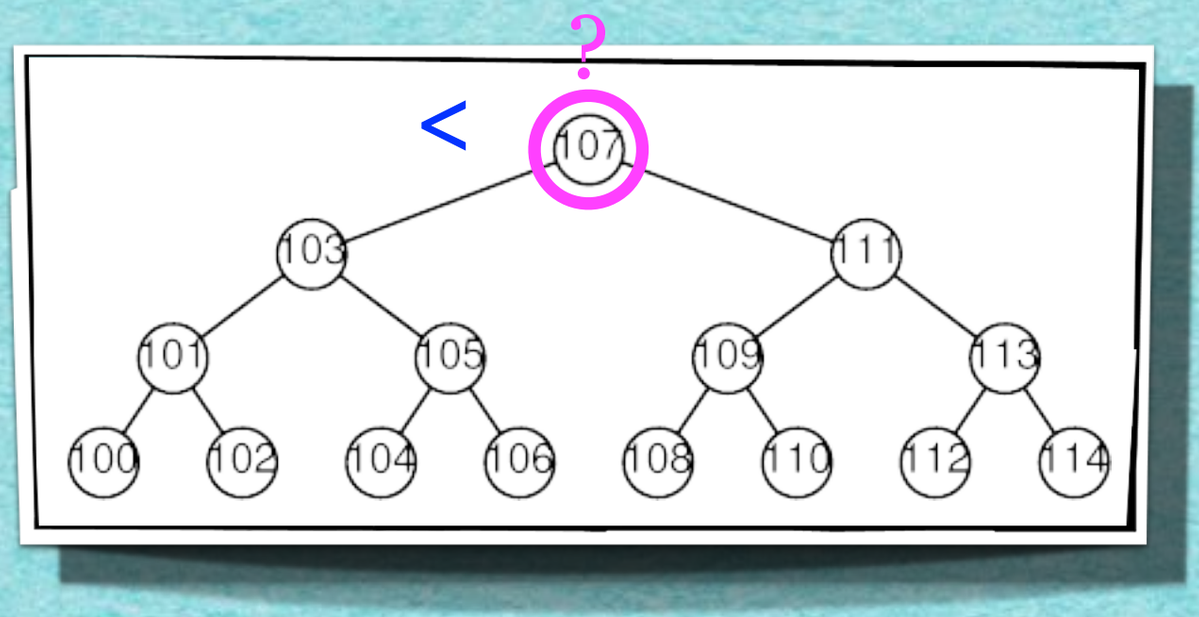
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

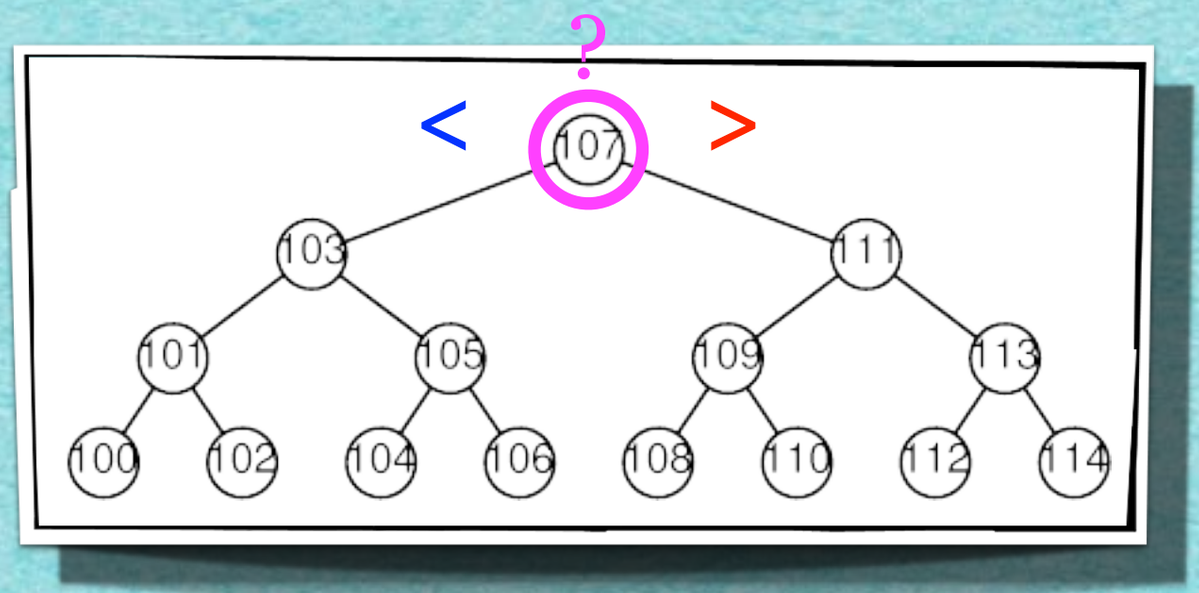
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

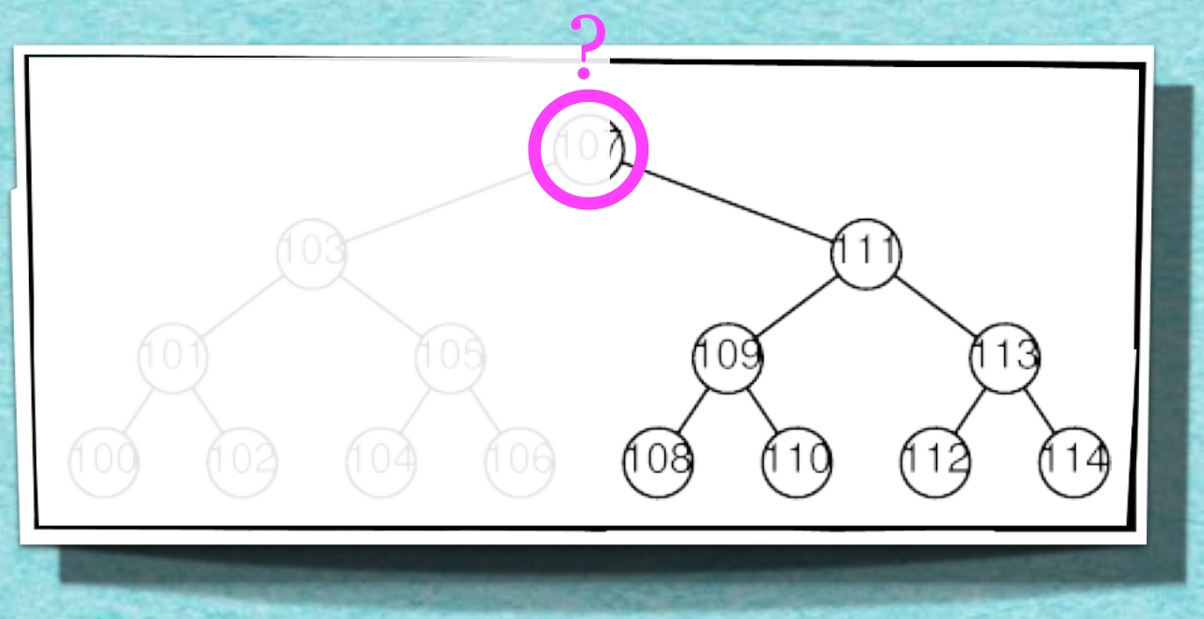
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

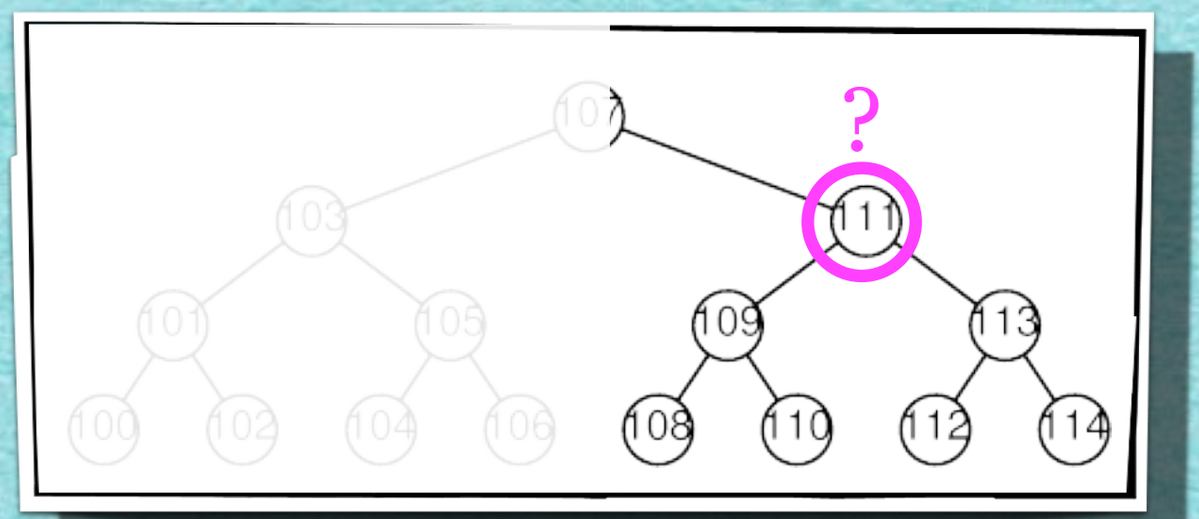
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

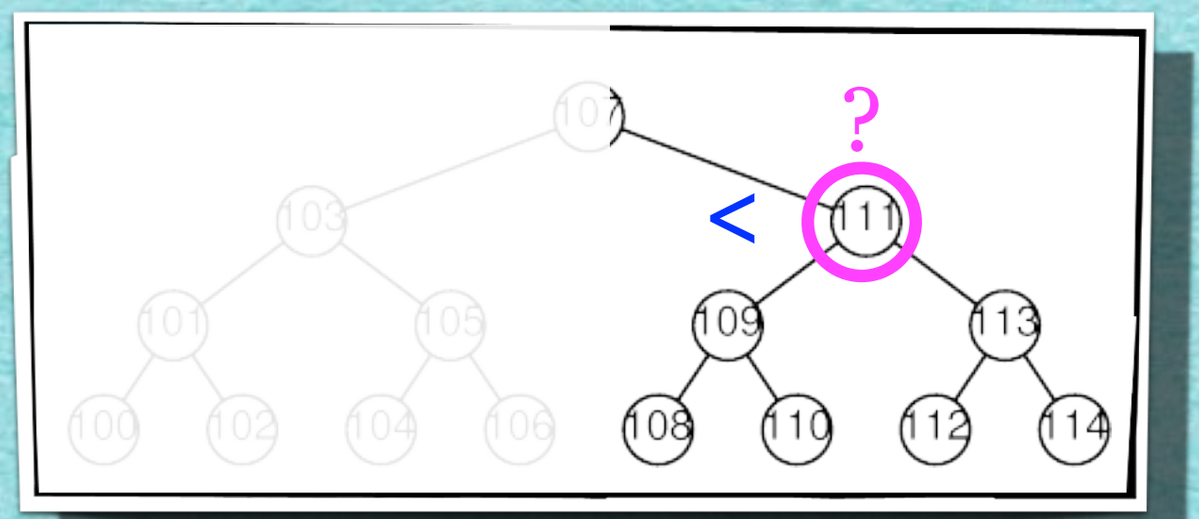
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

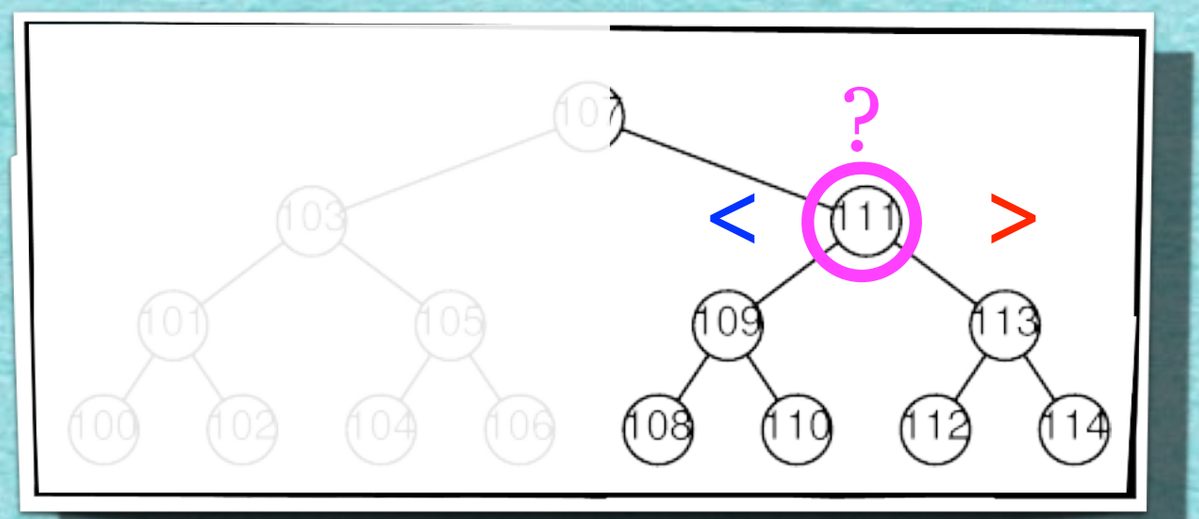
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

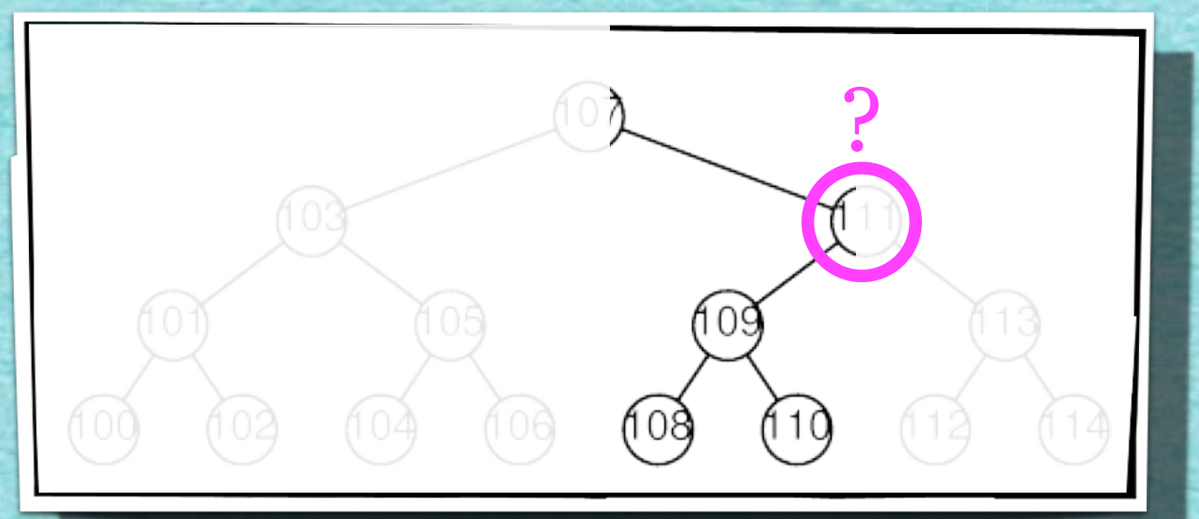
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

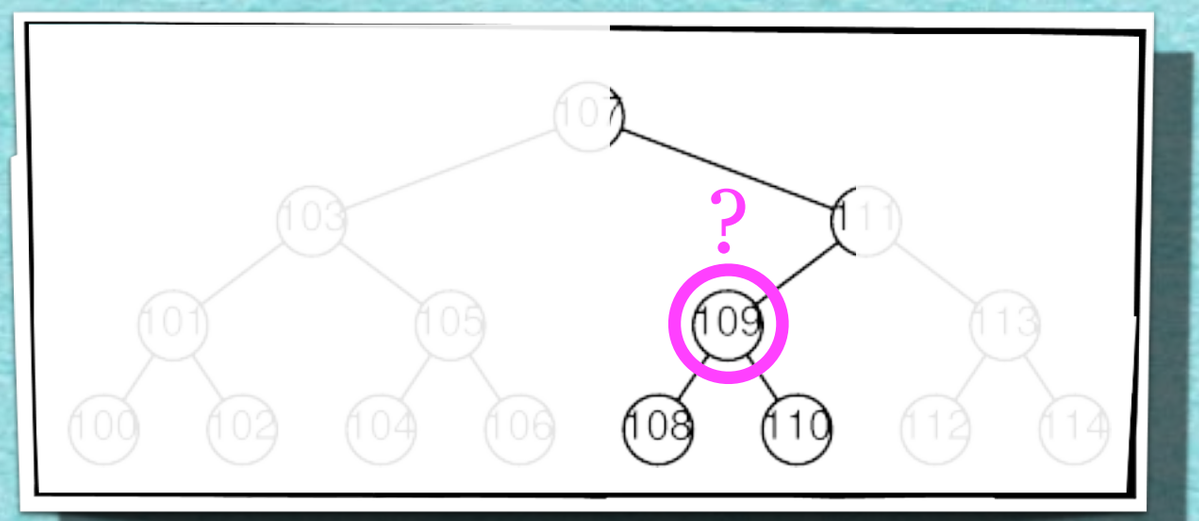
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

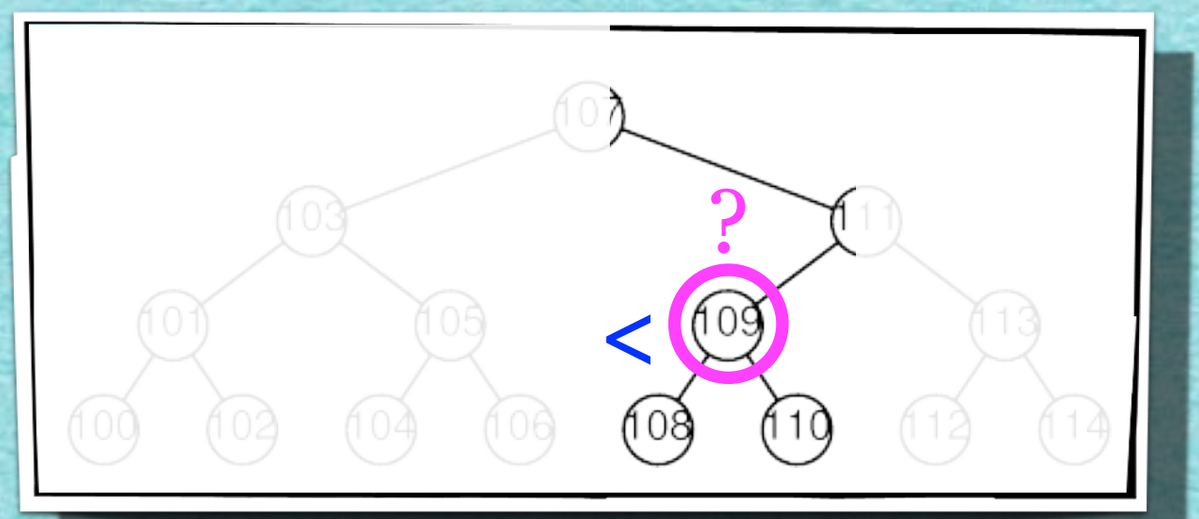
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

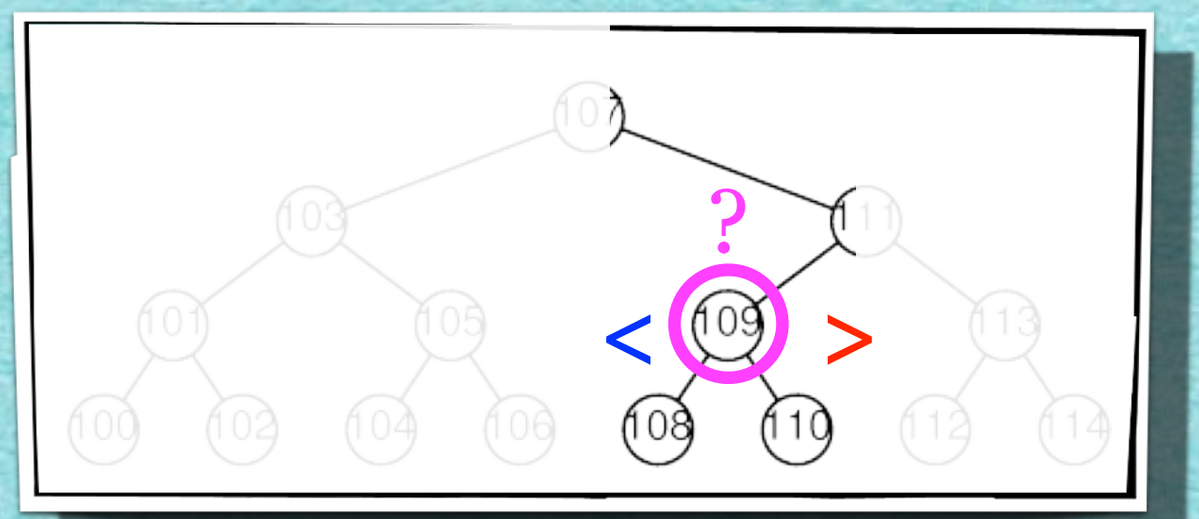
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

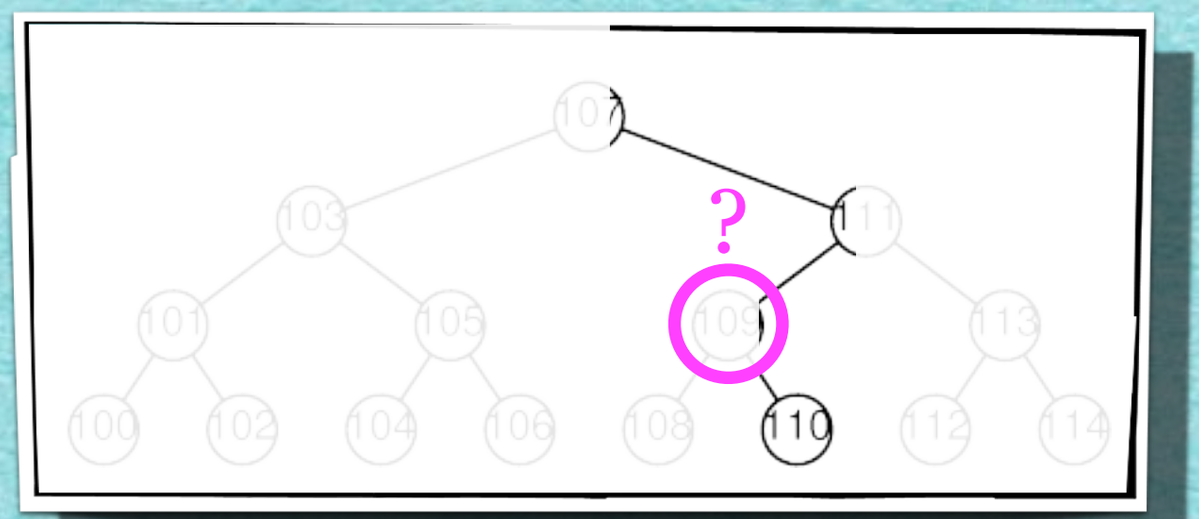
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

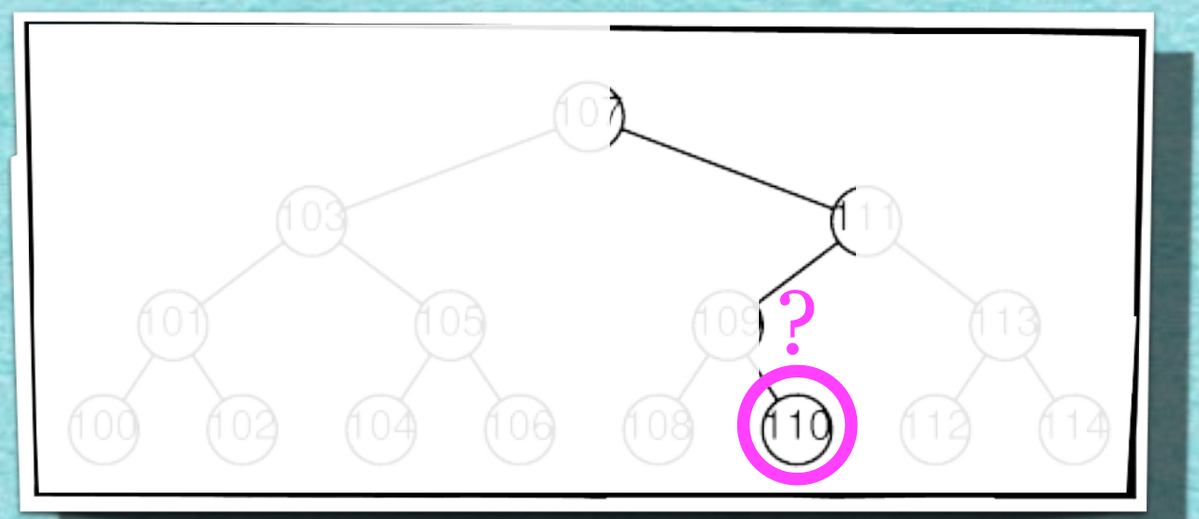
- ***Rate eine Zahl zwischen 100 und 114!***



4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Rate eine Zahl zwischen 100 und 114!***



Algorithmus 4.1

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,

linke Randposition LINKS, rechte Randposition RECHTS,

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {

   1.1. MITTE:= $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {

   1.1. MITTE:= $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$

   1.2. IF (S[MITTE]=WERT) THEN

      1.2.1. RETURN MITTE

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {

    1.1. MITTE:= $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$

    1.2. IF (S[MITTE]=WERT) THEN

        1.2.1. RETURN MITTE

    1.3. ELSEIF (S[MITTE]>WERT) THEN

        1.3.1. RECHTS:=MITTE-1

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {

    1.1. MITTE:= $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$

    1.2. IF (S[MITTE]=WERT) THEN

        1.2.1. RETURN MITTE

    1.3. ELSEIF (S[MITTE]>WERT) THEN

        1.3.1. RECHTS:=MITTE-1

    1.4. ELSEIF

        1.4.1. LINKS:=MITTE+1

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {

    1.1. MITTE:= $\left\lfloor \frac{\text{LINKS} + \text{RECHTS}}{2} \right\rfloor$

    1.2. IF (S[MITTE]=WERT) THEN

        1.2.1. RETURN MITTE

    1.3. ELSEIF (S[MITTE]>WERT) THEN

        1.3.1. RECHTS:=MITTE-1

    1.4. ELSEIF

        1.4.1. LINKS:=MITTE+1

}

Algorithmus 4.1

INPUT: Sortierter Array mit Einträgen S[I], Suchwert WERT,
linke Randposition LINKS, rechte Randposition RECHTS,

OUTPUT: Position von WERT zwischen Arraypositionen LINKS und RECHTS, falls existent

BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1. WHILE (LINKS≤RECHTS) DO {
    1.1. MITTE:= $\left\lfloor \frac{LINKS + RECHTS}{2} \right\rfloor$
    1.2. IF (S[MITTE]=WERT) THEN
        1.2.1. RETURN MITTE
    1.3. ELSEIF (S[MITTE]>WERT) THEN
        1.3.1. RECHTS:=MITTE-1
    1.4. ELSEIF
        1.4.1. LINKS:=MITTE+1
    }
2. RETURN "WERT nicht gefunden!"

4.4 Binäre Suche

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

4.4 Binäre Suche

<u>Aufgabenstellung:</u>

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
      1.1. MITTE:= ⌊(LINKS + RECHTS) / 2⌋
      1.2. IF (S[MITTE]=WERT) THEN
        1.2.1. RETURN MITTE
      1.3. ELSEIF (S[MITTE]>WERT) THEN
        1.3.1. RECHTS:=MITTE-1
      1.4. ELSEIF
        1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

<u>Aufgabenstellung:</u>

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

1 2 5 6 13 17 28 33 42 47 52 64 89 96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

i

S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	LINKS													
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	LINKS												RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
      1.1. MITTE:= ⌊(LINKS + RECHTS) / 2⌋
      1.2. IF (S[MITTE]=WERT) THEN
        1.2.1. RETURN MITTE
      1.3. ELSEIF (S[MITTE]>WERT) THEN
        1.3.1. RECHTS:=MITTE-1
      1.4. ELSEIF
        1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

		LINKS											RECHTS	
i		1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]		1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	LINKS												RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	LINKS						MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.   WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
                 1.2.1. RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
                 1.3.1. RECHTS:=MITTE-1
        1.4.  ELSEIF
                 1.4.1. LINKS:=MITTE+1
     }
2.   RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	LINKS						MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	LINKS						MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

		LINKS					MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

							MITTE	LINKS						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
      1.1. MITTE:= ⌊(LINKS + RECHTS) / 2⌋
      1.2. IF (S[MITTE]=WERT) THEN
        1.2.1. RETURN MITTE
      1.3. ELSEIF (S[MITTE]>WERT) THEN
        1.3.1. RECHTS:=MITTE-1
      1.4. ELSEIF
        1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

	MITTE	LINKS								RECHTS
i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
S[i]	**1**	**2**	**5**	**6**	**13**	**17**	**28**	**33**	**42**	**47**	**52**	**64**	**89**	**96**

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

							MITTE	LINKS						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS			MITTE			RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS			MITTE			RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS			MITTE			RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS		RECHTS	MITTE			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
      1.1. MITTE:= ⌊(LINKS + RECHTS) / 2⌋
      1.2. IF (S[MITTE]=WERT) THEN
        1.2.1. RETURN MITTE
      1.3. ELSEIF (S[MITTE]>WERT) THEN
        1.3.1. RECHTS:=MITTE-1
      1.4. ELSEIF
        1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS		RECHTS	MITTE			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS		RECHTS	MITTE			
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=42

								LINKS	MITTE	RECHTS				
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

RETURN 9

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS													
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

		LINKS												RECHTS
i		1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]		1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS						MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS						MITTE						RECHTS
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS					RECHTS	MITTE						
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS		MITTE			RECHTS							
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1. MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2. IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3. ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4. ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS		MITTE			RECHTS								
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1. MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2. IF (S[MITTE]=WERT) THEN
            1.2.1. RETURN MITTE
        1.3. ELSEIF (S[MITTE]>WERT) THEN
            1.3.1. RECHTS:=MITTE-1
        1.4. ELSEIF
            1.4.1. LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	MITTE	LINKS	RECHTS										
i	3	4	6	1	2	5	7	8	9	10	11	12	13	14
S[i]	5	6	17	1	2	13	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS	MITTE	RECHTS									
i	4	5	6	7	8	9	10	11	12	13	14	
S[i]	6	13	17	28	33	42	47	52	64	89	96	

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

	LINKS	MITTE	RECHTS									
i	4	5	6	7	8	9	10	11	12	13	14	
S[i]	6	13	17	28	33	42	47	52	64	89	96	

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

RECHTS

MITTE

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

LINKS

RECHTS

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

RECHTS

LINKS

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

RECHTES

LINKS

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

```
BINÄRESUCHE(S,WERT,LINKS,RECHTS)

1.  WHILE (LINKS≤RECHTS) DO {
        1.1.  MITTE:= ⌊(LINKS + RECHTS) / 2⌋
        1.2.  IF (S[MITTE]=WERT) THEN
            1.2.1.  RETURN MITTE
        1.3.  ELSEIF (S[MITTE]>WERT) THEN
            1.3.1.  RECHTS:=MITTE-1
        1.4.  ELSEIF
            1.4.1.  LINKS:=MITTE+1
    }
2.  RETURN "WERT nicht gefunden!"
```

Binäre Suche

Aufgabenstellung:

- ***Finde eine gesuchte Zahl in der gegebenen sortierten Menge!***

WERT=7

				RECHTES	LINKS									
i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S[i]	1	2	5	6	13	17	28	33	42	47	52	64	89	96

WERT nicht gefunden!

4.4 Binäre Suche

Satz 4.2

Satz 4.2

Die binäre Suche terminiert in O(log(RECHTS-LINKS)) Schritten

Satz 4.2

Die binäre Suche terminiert in O(log(RECHTS-LINKS)) Schritten (für RECHTS>LINKS).

Satz 4.2

Die binäre Suche terminiert in O(log(RECHTS-LINKS)) Schritten (für RECHTS>LINKS).

Beweis:

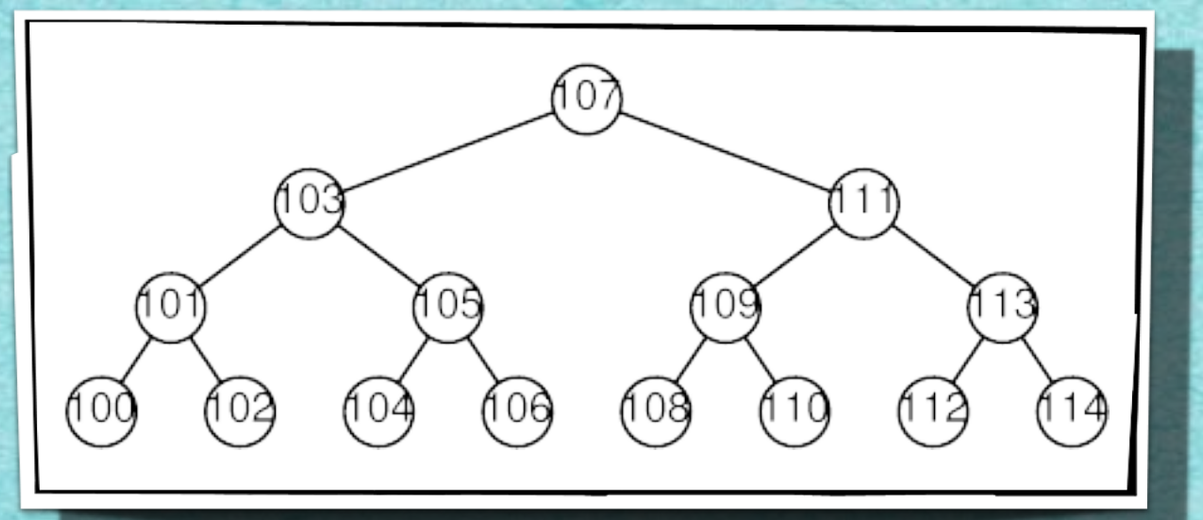
Selbst!

Satz 4.2

Die binäre Suche terminiert in O(log(RECHTS-LINKS)) Schritten (für RECHTS>LINKS).

Beweis:

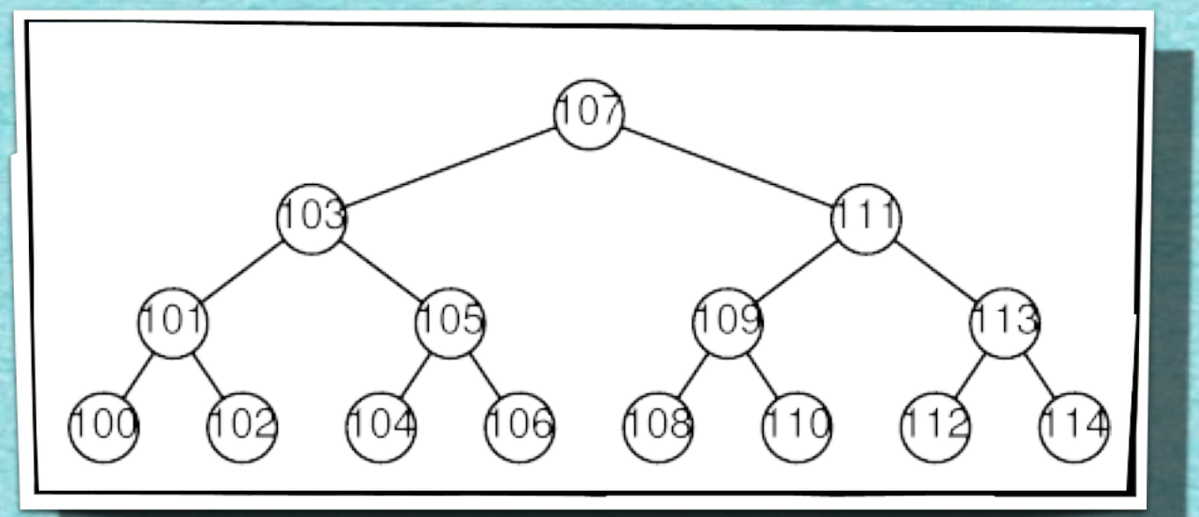
Selbst!



4.5 Binäre Suchbäume

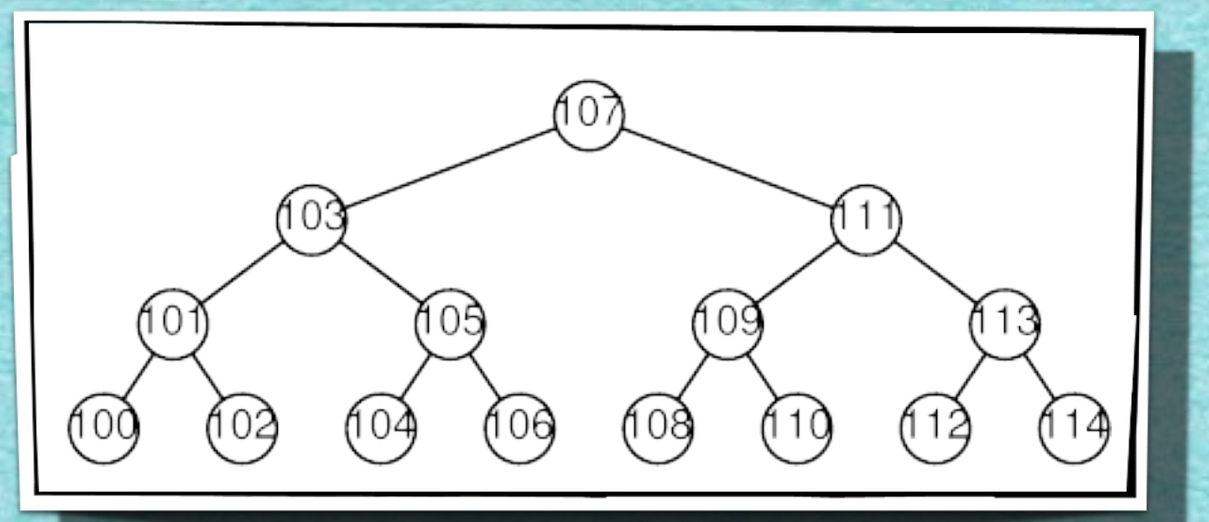
4.5 Binäre Suchbäume

4.5 Binäre Suchbäume



4.5 Binäre Suchbäume

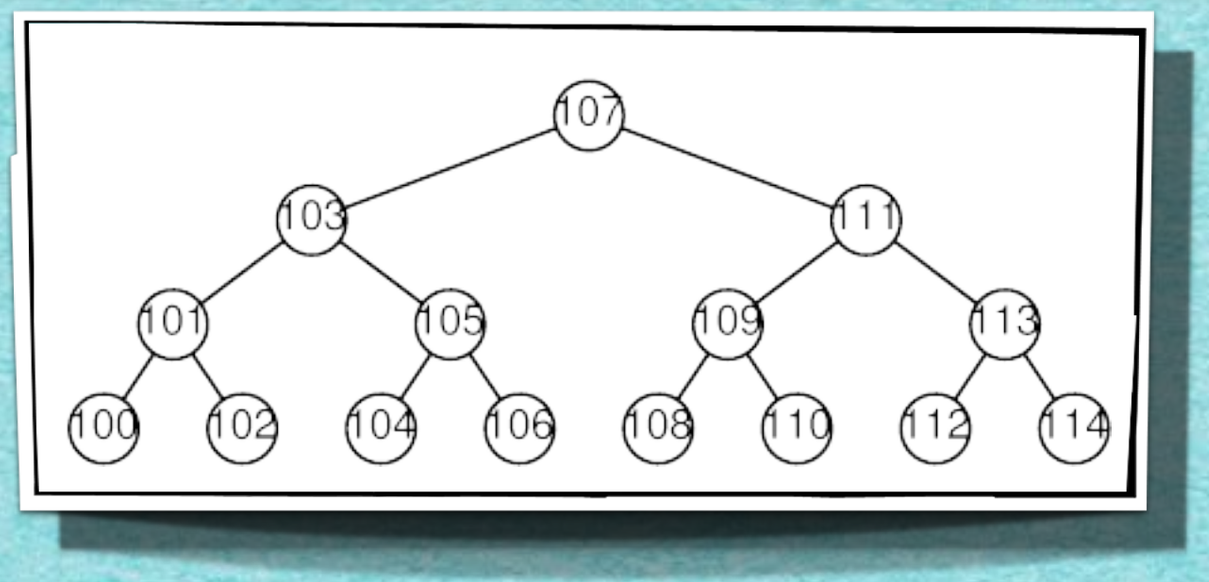
Ideen:



4.5 Binäre Suchbäume

Ideen:

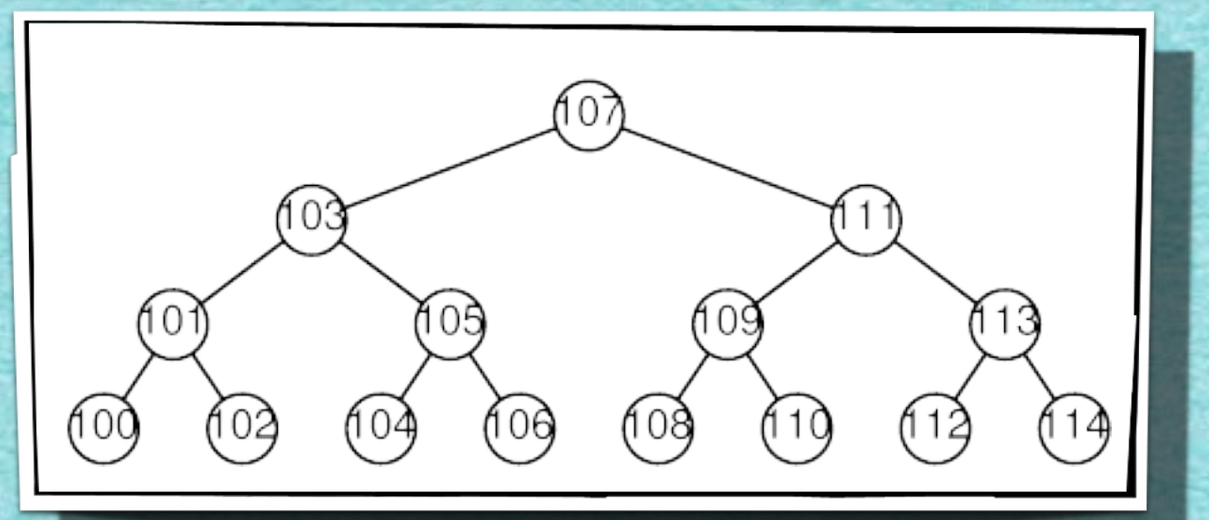
- ***Strukturiere Daten wie im möglichen Ablauf einer binären Suche!***

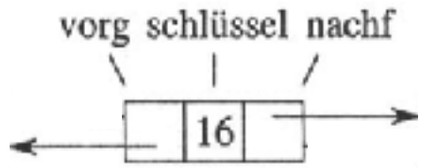


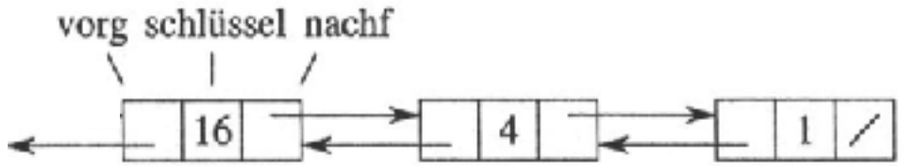
4.5 Binäre Suchbäume

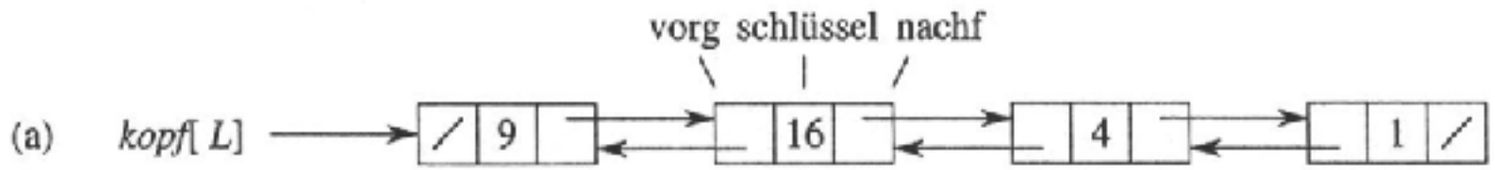
Ideen:

- ***Strukturiere Daten wie im möglichen Ablauf einer binären Suche!***
- ***Erziele logarithmische Zeiten!***

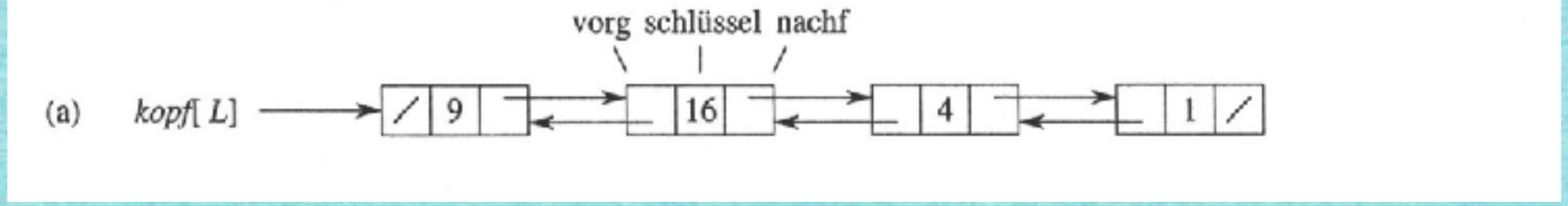



vorg schlüssel nachf
16

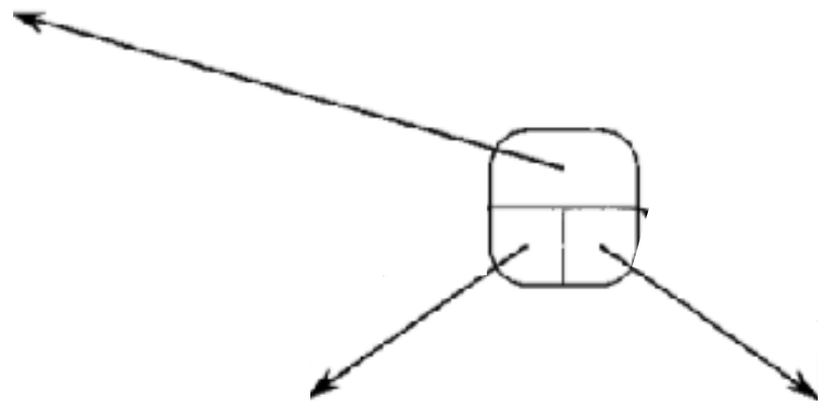

vorg schlüssel nachf
16
4
1


vorg schlüssel nachf
(a)
kopf[L]
9
16
4
1

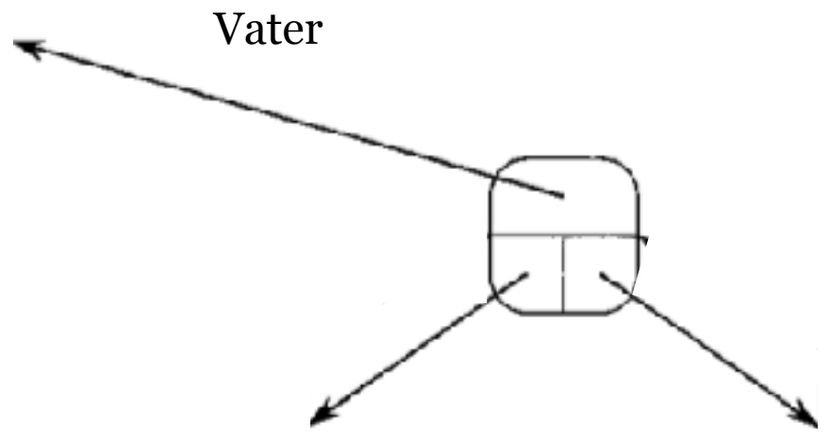
Struktur einer doppelt verketteten Liste



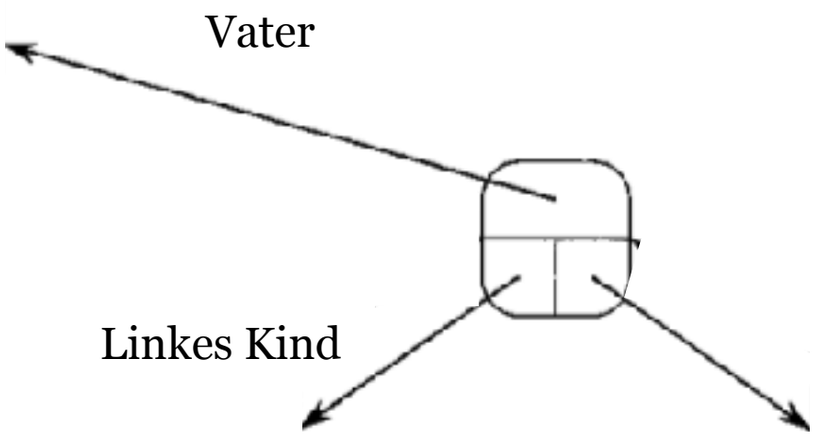
Binärer Suchbaum



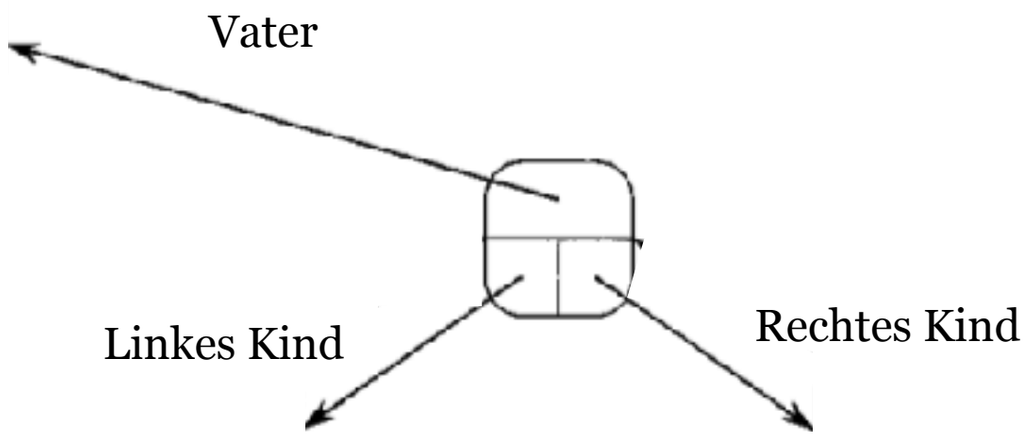
Binärer Suchbaum



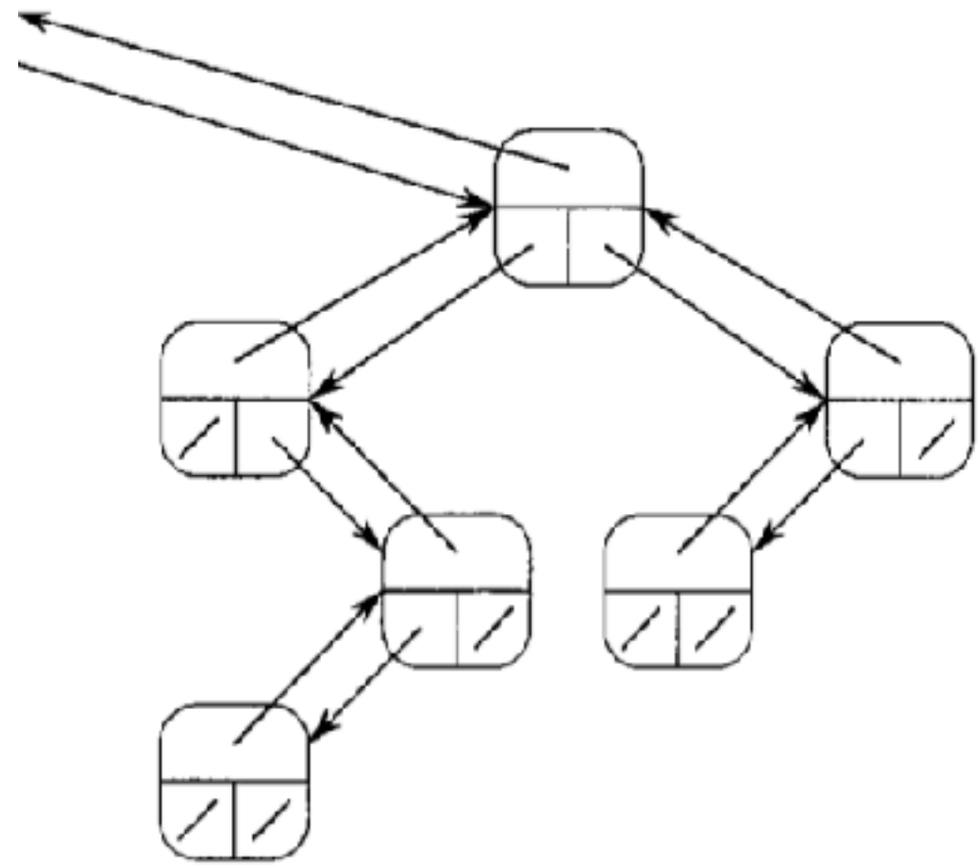
Binärer Suchbaum



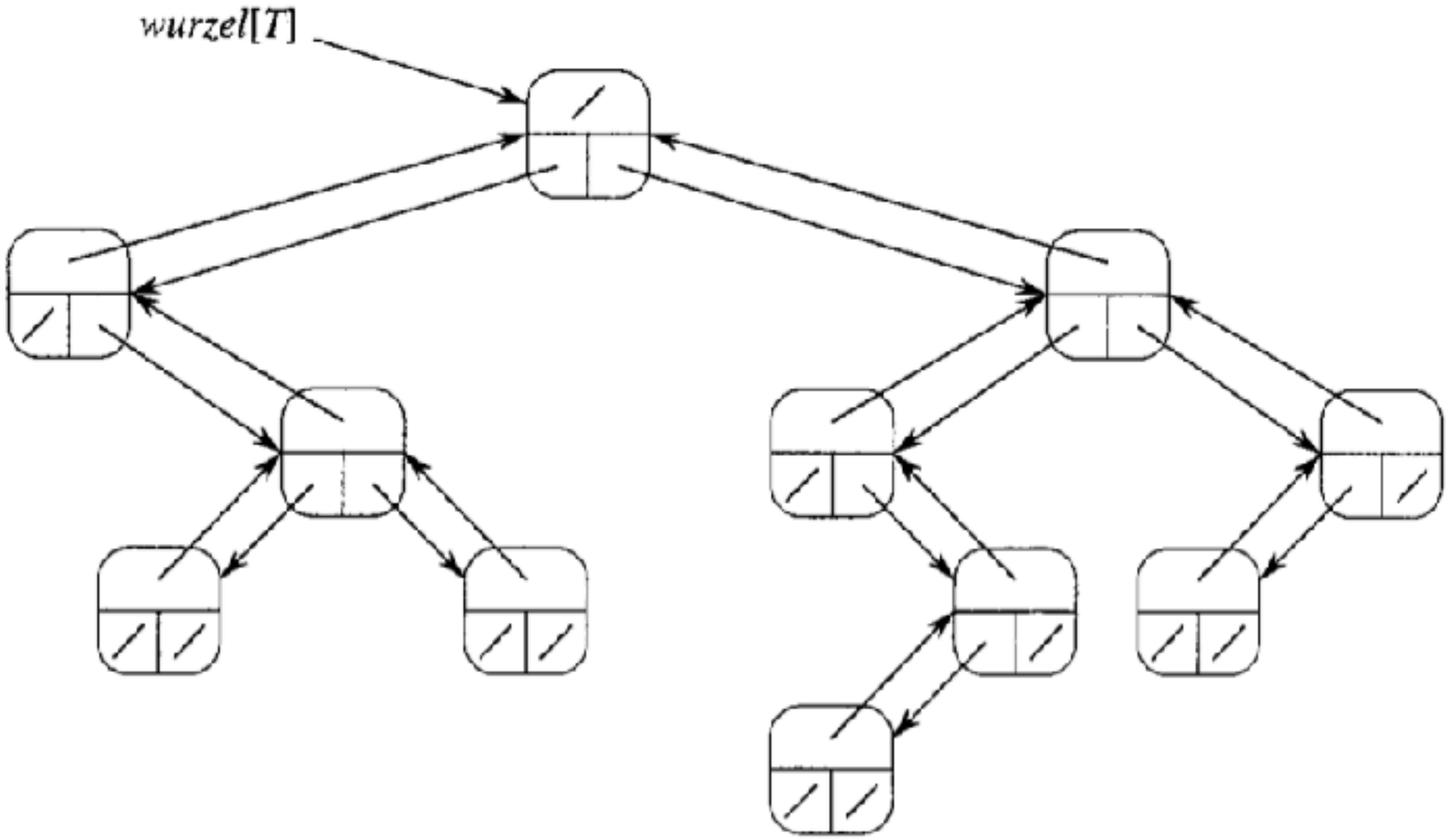
Binärer Suchbaum



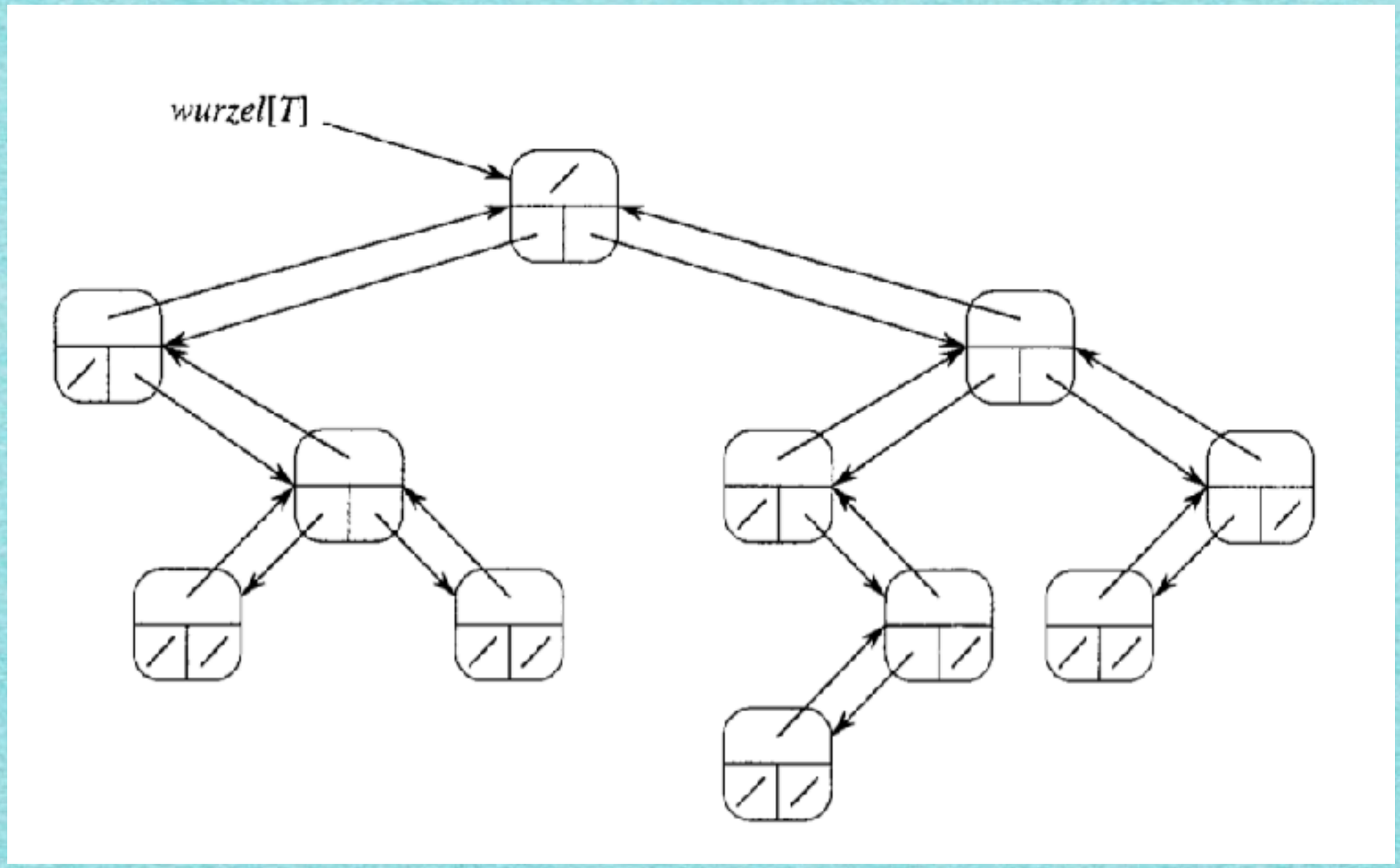
Binärer Suchbaum



Binärer Suchbaum



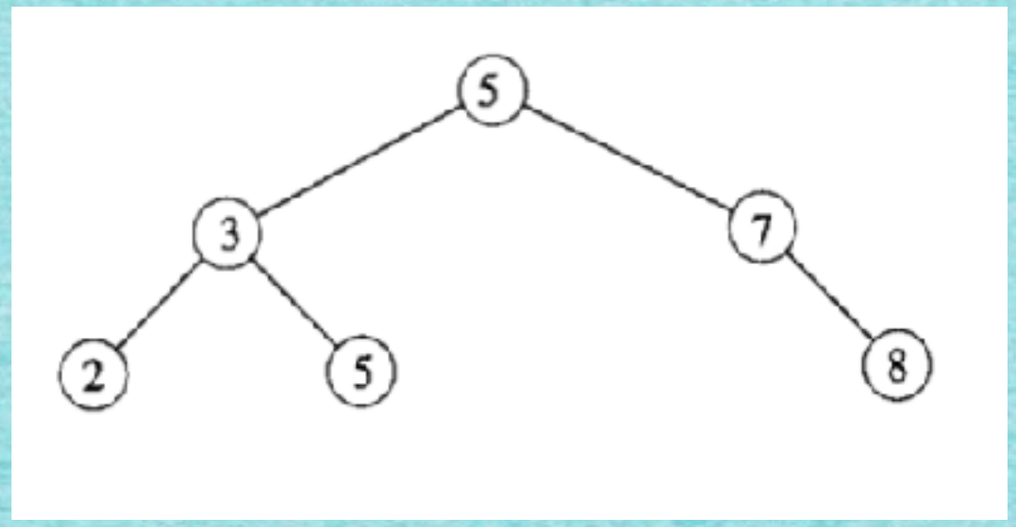
Binärer Suchbaum



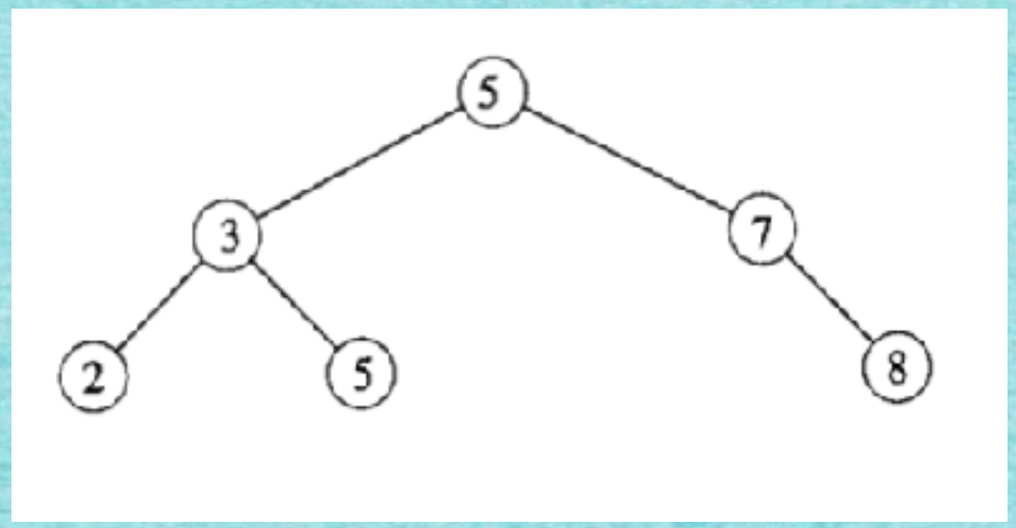
Außerdem wichtig: Struktur der Schlüsselwerte!

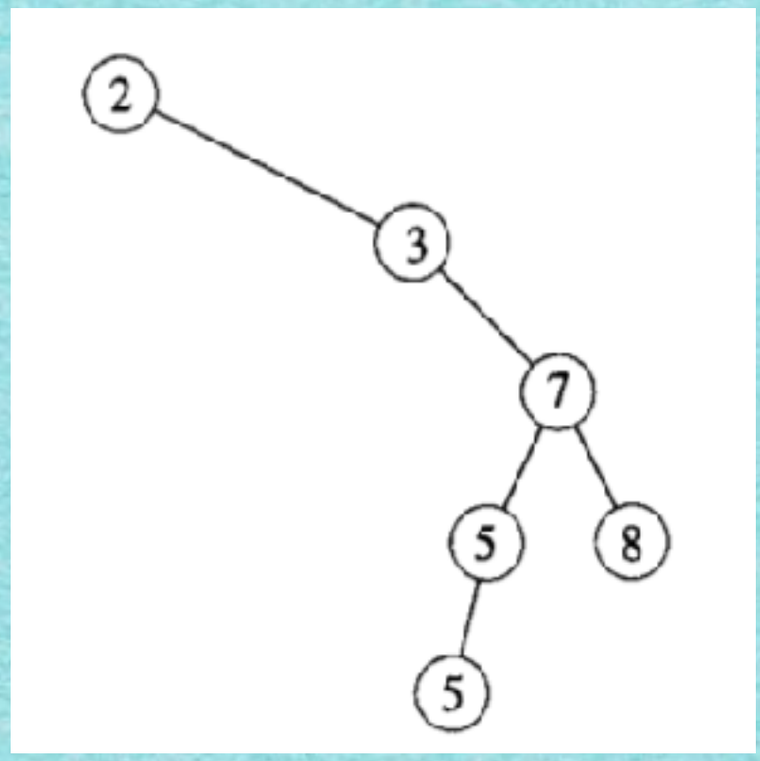
Ordnungsstruktur

Ordnungsstruktur

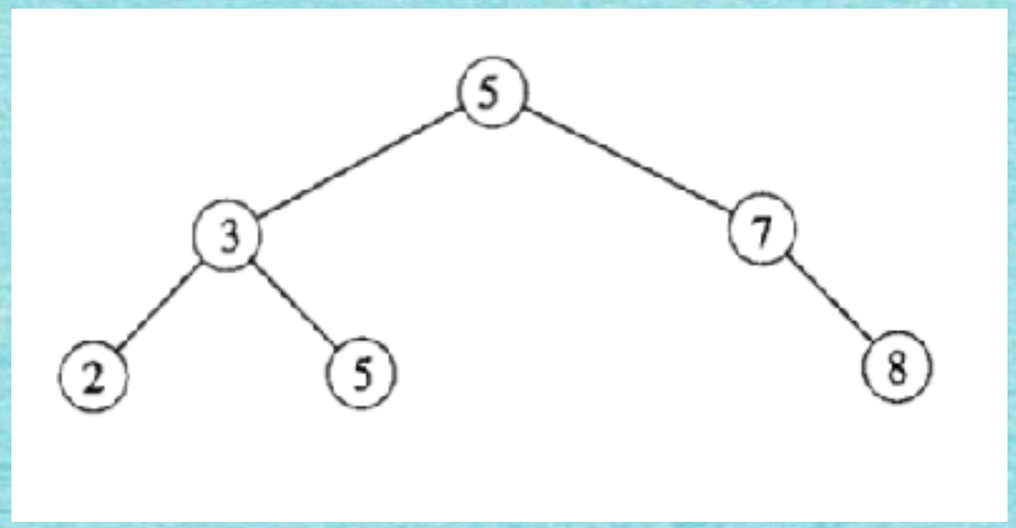


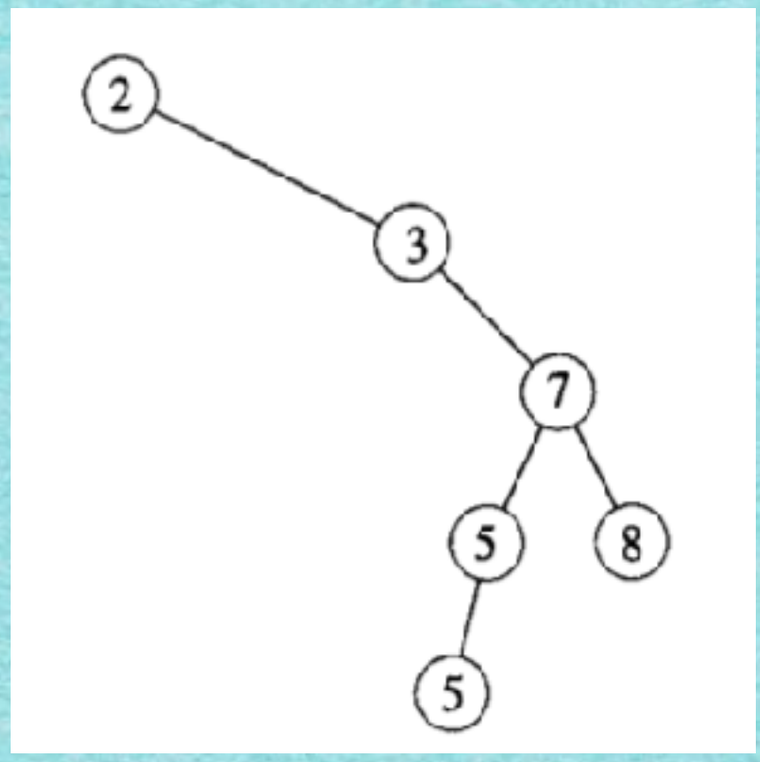
Ordnungsstruktur





Ordnungsstruktur





Linker Teilbaum: Kleinere (bzw. nicht größere) Zahlen
Rechter Teilbaum: Größere Zahlen

Mehr demnächst!

s.fekete@tu-bs.de