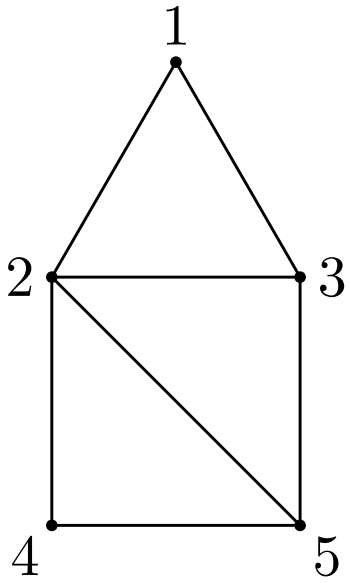
Rekapitulation: Darstellung von Graphen



Adjazenzliste

1	2, 3
2	1, 3, 4, 5
3	1, 2, 5
4	2, 5
5	2, 3, 4

Alle bisherigen Algorithmen:

Adjazenzliste gute Darstellung

Kritische Operation: Alle ausgehenden Kanten besuchen.

All Pairs Shortest Paths

Eingabe: Gerichteter Graph mit Kantengewichten

Ausgabe: Abstände und kürzeste Wege zwischen allen Knotenpaaren

Laufzeit $O((|V|^2 + |V| \cdot |E|) \log |V|)$ mit Dijkstra.

$\longrightarrow$ Wende Dijkstra auf jeden Knoten an.

Algorithmus von Floyd

Algorithmus

procedure *Floyd*1 :
for $i = 1, ..., n$ **do**
  for $j = 1, ..., n$ **do** $d[i, j, 0] := length[i, j]$ **od**
od;
for $k = 1, ..., n$ **do**
  for $i = 1, ..., n$ **do**
    for $j = 1, ..., n$ **do**
      $d[i, j, k] := min\{d[i, j, k-1], d[i, k, k-1] + d[k, j, k-1]\}$
    od
  od
od

Der Abstand von i nach j ist in $d[i, j, n]$ zu finden.

Theorem

Die kürzesten Wege zwischen allen Knotenpaaren eines gerichteten Graphen $G = (V, E)$ können in $O(|V|^3)$ Schritten gefunden werden.

Beweis.

Per Induktion: $d[i, j, k]$ enthält die Länge des kürzesten Pfades von i nach j, wenn nur $1, \ldots, k$ als Zwischenstationen erlaubt sind. Dann enthält $d[i, j, n]$ den wirklichen Abstand. □

Algorithmus von Floyd

Einfachere Version:

Algorithmus

procedure *Floyd* :
for $i = 1, ..., n$ **do**
  for $j = 1, ..., n$ **do** $d[i,j] := length[i,j]$ **od**
od;
for $k = 1, ..., n$ **do**
  for $i = 1, ..., n$ **do**
    for $j = 1, ..., n$ **do**
      $d[i,j] := min\{d[i,j], d[i,k] + d[k,j]\}$
    od
  od
od

Spezialfall: Transitive Hülle – Algorithmus von Warshall

Frage: Zwischen welchen Knotenpaaren gibt es einen Weg?

Algorithmus

```
procedure Warshall :
for i = 1, ..., n do
  for j = 1, ..., n do D[i,j] := A[i,j] od
od;
for k = 1, ..., n do
  for i = 1, ..., n do
    for j = 1, ..., n do
      D[i,j] := D[i,j] || D[i,k] && D[k,j]
    od
  od
od
```

Transitive Hülle

Theorem

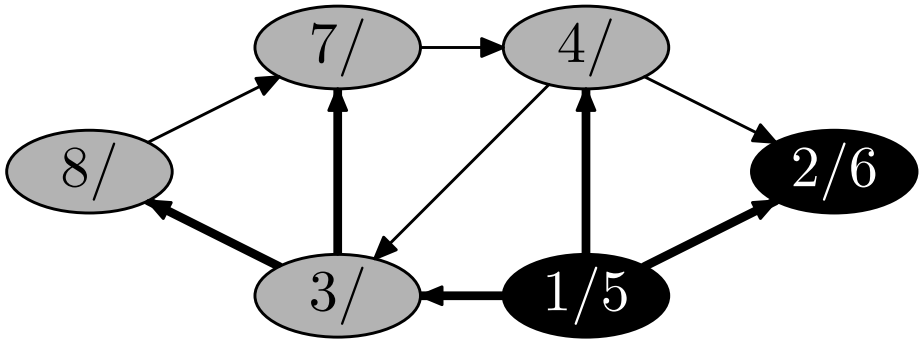
Wir können die transitive Hülle eines gerichteten Graphen $G = (V, E)$, der k starke Zusammenhangskomponenten hat, in $O(|V| + |E'| + k^3)$ Schritten berechnen, wobei E' die Kanten der transitiven Hülle sind.

Beweis.

1. Berechne die starken Zusammenhangskomponenten
2. Schrumpfe jede SCC X zu einem Knoten
3. Berechne die transitive Hülle H dieses Graphen
4. Für jede Kante (X, Y) in H gib alle Kanten (x, y) mit $x \in X, y \in Y$ aus.

□

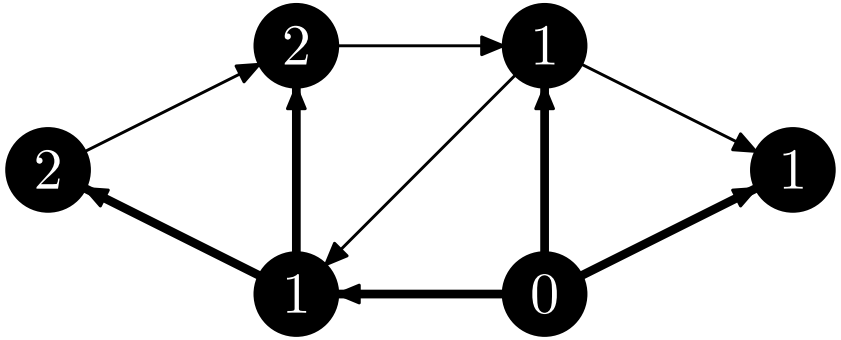
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

Breitensuche



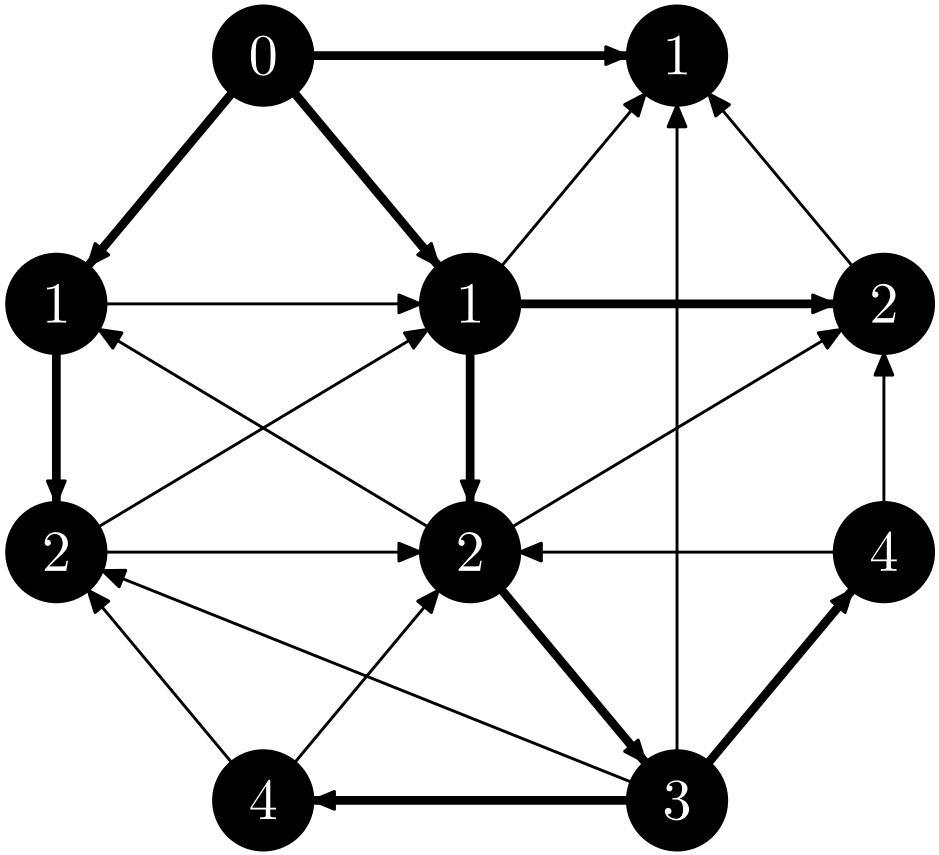
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche – Größeres Beispiel



Java

```
public void BFS(Node s, Map<Node, Node> p) {
   SimpleIterator<Node> it;
   Map<Node, Integer> color = new Hashtable<Node, Integer>();
   Queue<Node> q = new Queue<Node>();
   for(it = nodeiterator(); it.more(); it.step()) {
      it.key().weight = 1.0e10; color.insert(it.key(), 0); } // white
   s.weight = 0.0; q.enqueue(s); color.insert(s, 1);  // gray
   while(!q.isempty()) {
      Node v = q.dequeue();
      Iterator<Node, Edge> adj;
      for(adj = neighbors.find(v).iterator(); adj.more(); adj.step()) {
         Node u = adj.key();
         if(color.find(u) == 0) {
            color.insert(u, 1);  // gray
            u.weight = v.weight + 1; p.insert(u, v); q.enqueue(u);
         }
      }
      color.insert(v, 2);  // black
   }
}
```

Breitensuche

Theorem

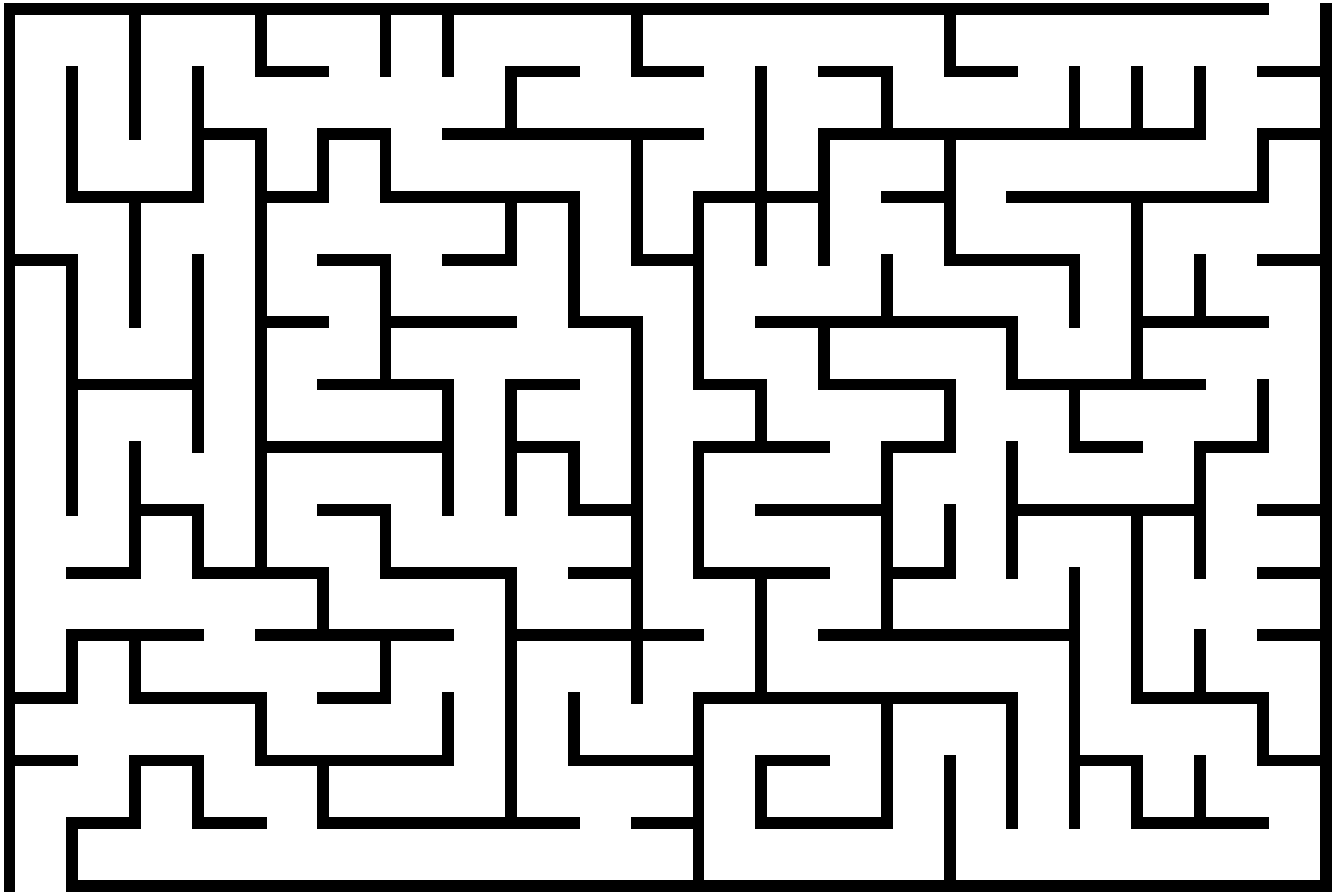
Gegeben sei ein gerichteter oder ungerichteter Graph $G = (V, E)$ und ein Knoten $s \in V$.
Die kürzesten Wege von s zu allen anderen Knoten und die zugehörigen Abstände können in $O(|V| + |E|)$ berechnet werden.

Beweis.

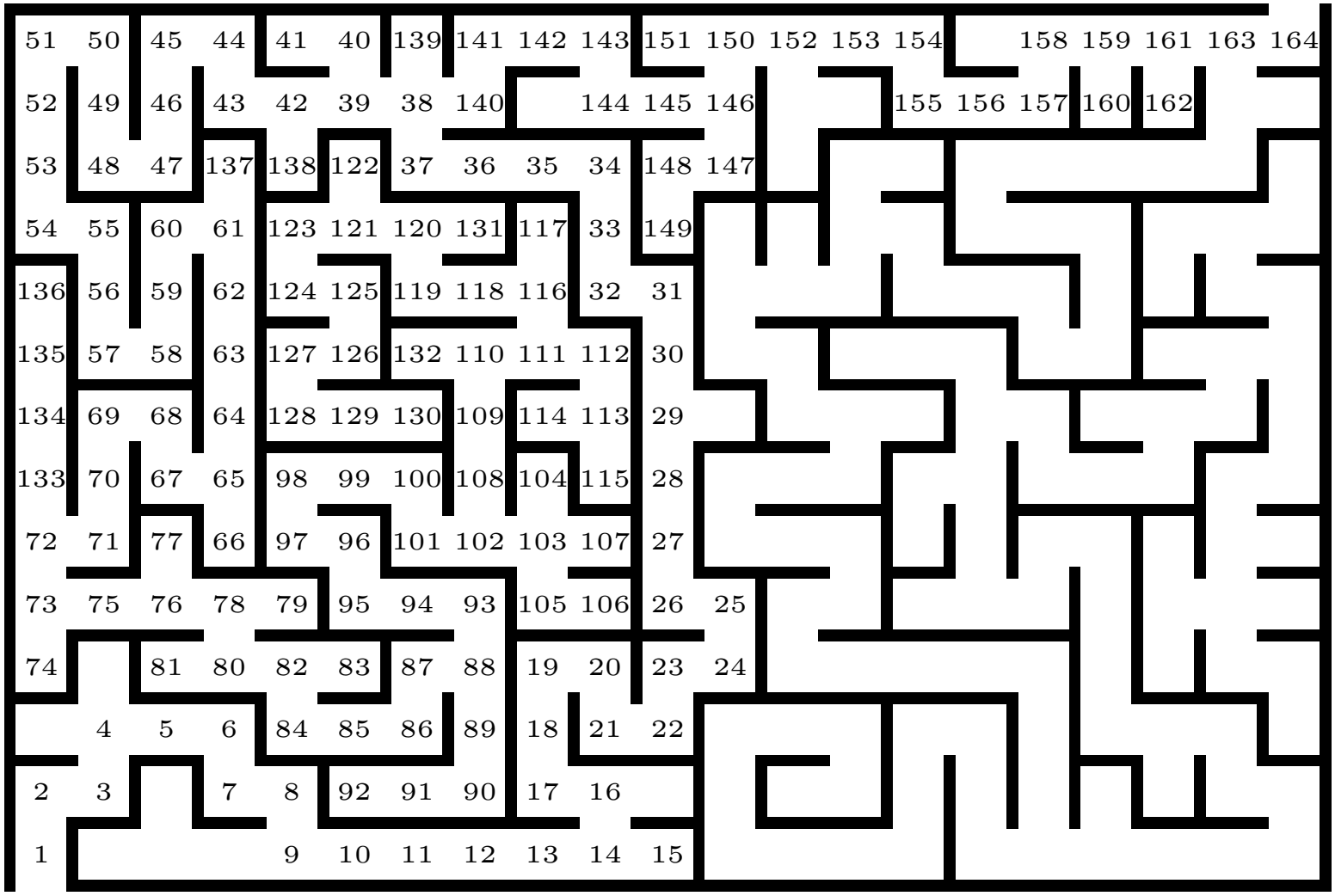
Wir verwenden Breitensuche.
Die Laufzeit ist offensichtlich linear.

Der Korrektheitsbeweis sei hier weggelassen (etwas lang und technisch). □

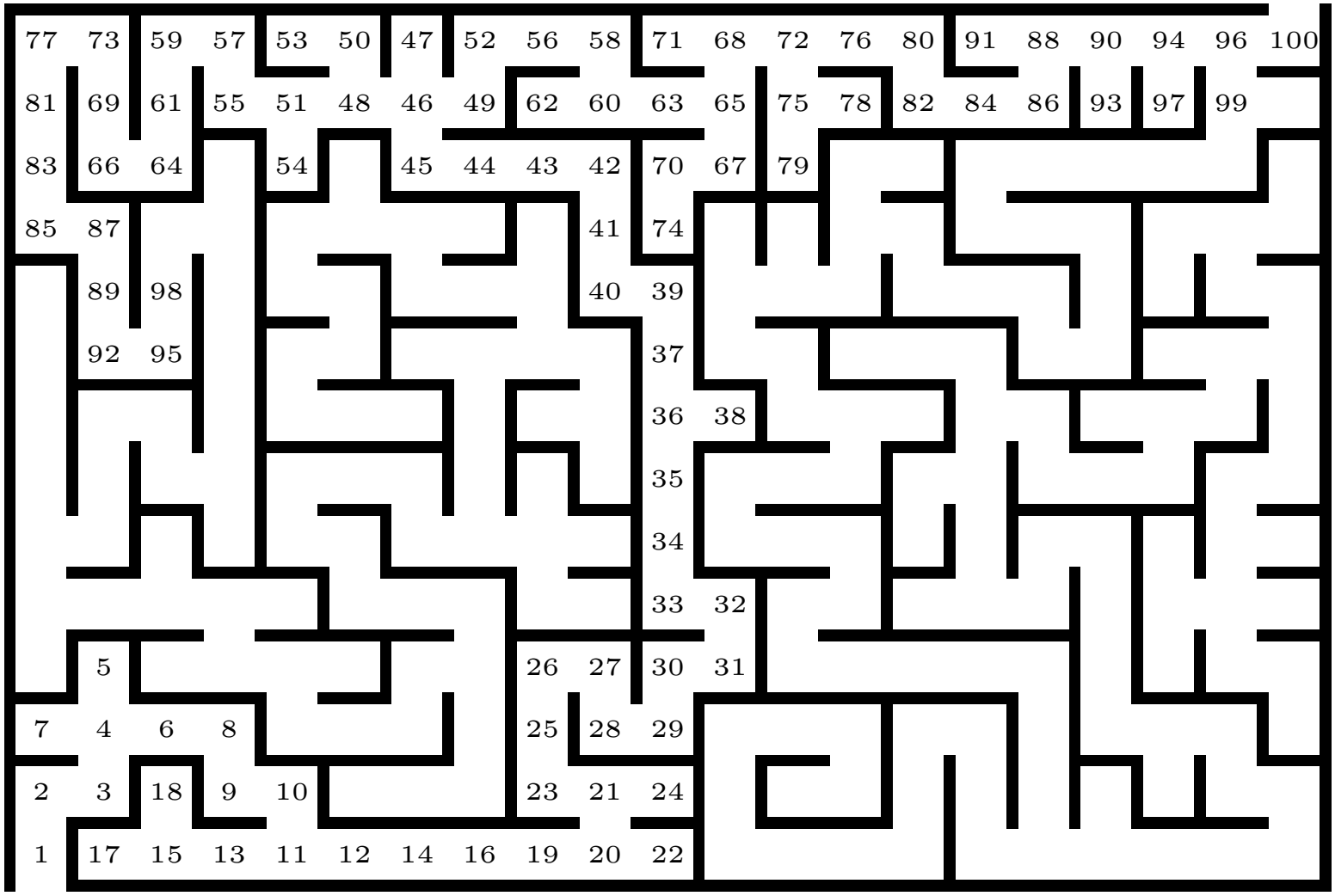
Beispiel



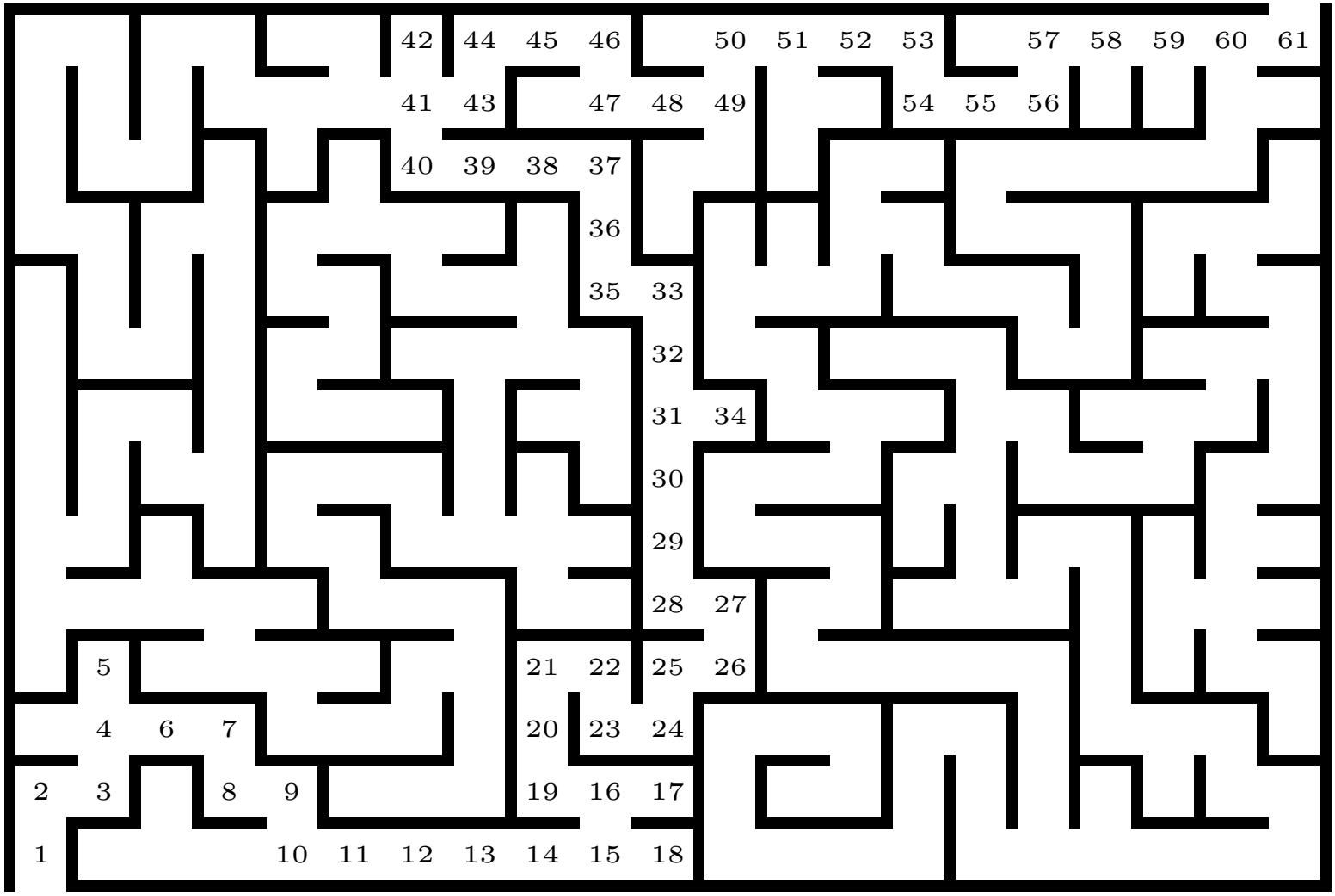
Beispiel: DFS



Beispiel: BFS



Beispiel: LC


42 44 45 46 50 51 52 53 57 58 59 60 61
41 43 47 48 49 54 55 56
40 39 38 37
36
35 33
32
31 34
30
29
28 27
5 21 22 25 26
4 6 7 20 23 24
2 3 8 9 19 16 17
1 10 11 12 13 14 15 18