Tag 23 - A Long Walk

Teil 1

Teil 1 ließ sich mit einem leicht modifizierten Dijkstra-Algorithmus lösen. Dazu habe ich die separate Klasse "Kreuzung" genutzt.

Lösungsvorschlag Klasse Kreuzung

```
import java.util.HashMap;

public class Kreuzung
{
    private HashMap<String,Integer> distanzen;
    private int gesamtDistanz;
    private boolean istBesucht;

    /**
     * Konstruktor für Objekte der Klasse Kreuzung
     */
    public Kreuzung()
    {
        this.gesamtDistanz = 0;
        istBesucht = false;
        distanzen = new HashMap<String,Integer>();
    }

    public HashMap<String,Integer> getDistanzen() {
        return this.distanzen;
    }

    public void addDistanz(String key, int value) {
        distanzen.put(key, value);
    }

    public void setBesucht() {
        this.istBesucht = true;
    }

    public boolean istBesucht() {
        return this.istBesucht;
    }

    public int getGesamtDistanz() {
        return this.gesamtDistanz;
```

```
    }

    public void setGesamtDistanz(int d) {
        this.gesamtDistanz = d;
    }
}
```

Lösungsvorschlag Teil 1

```
private char[][] originalMap, map;
private int breite, hoehe;
private HashMap<String,Kreuzung> kreuzungen;

public long partOne() {
    breite = inputLines.get(0).length();
    hoehe = inputLines.size();
    map = new char[breite][hoehe];
    kreuzungen = new HashMap<String,Kreuzung>();

    // lies map ein
    for (int y = 0; y < hoehe; y++) {
        String line = inputLines.get(y);
        for (int x = 0; x < breite; x++) {
            map[x][y] = line.charAt(x);
        }
    }

    // setzte x bei Kreuzungen ein und erstelle Knoten
    map[1][0] = 'x';
    kreuzungen.put(1 + "," + 0, new Kreuzung());
    map[breite-2][hoehe-1] = 'x';
    kreuzungen.put((breite-2) + "," + (hoehe-1), new Kreuzung());

    for (int y = 1; y < hoehe-1; y++) {
        String line = inputLines.get(y);
        for (int x = 1; x < breite-1; x++) {
            if (map[x][y] != '.') {
                continue;
            }

            int possibleWays = 0;
            if (map[x-1][y] != '#') {
                possibleWays++;
            }
            if (map[x+1][y] != '#') {
                possibleWays++;
            }
            if (map[x][y-1] != '#') {
```

```
                possibleWays++;
            }
            if (map[x][y+1] != '#') {
                possibleWays++;
            }

            if (possibleWays >= 3) {
                map[x][y] = 'x';
                kreuzungen.put(x + "," + y, new Kreuzung());
            }
        }
    }

    originalMap = copyMap(map);

    // laufe von jeder Kreuzung in jede mögliche Richtung -> jede gerichtete
(!) Kante wird erfasst
    for (String k: kreuzungen.keySet()) {
        int kx = Integer.parseInt(k.split(",")[0]);
        int ky = Integer.parseInt(k.split(",")[1]);

        if (kx-1 >= 0 && map[kx-1][ky] != '>' && map[kx-1][ky] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx-1, ky, kx, ky);
        }
        if (kx+1 < breite && map[kx+1][ky] != '<' && map[kx+1][ky] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx+1, ky, kx, ky);
        }
        if (ky-1 >= 0 && map[kx][ky-1] != 'v' && map[kx][ky-1] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx, ky-1, kx, ky);
        }
        if (ky+1 < hoehe && map[kx][ky+1] != '^' && map[kx][ky+1] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx, ky+1, kx, ky);
        }
    }

    // modifizierter Dijkstra
    // setze k anfangs auf den links-oberen Startpunkt
    Kreuzung k = kreuzungen.get(1 + "," + 0);
    Queue<String> queue = new LinkedList<>();

    while(k != null) {

        k.setBesucht();

        // distanzen von k zu dessen nachbarn
        HashMap<String,Integer> d = k.getDistanzen();
        for (String nkk: d.keySet()) { // nachbar-kreuzung-key
```

```
            Kreuzung nk = kreuzungen.get(nkk);
            if (!nk.istBesucht()){
                if (!queue.contains(nkk) && !nkk.equals((breite-2) + "," +
(hoehe-1))) {
                    queue.offer(nkk);
                }

                if (k.getGesamtDistanz() + d.get(nkk) >
nk.getGesamtDistanz()) {
                    nk.setGesamtDistanz(k.getGesamtDistanz() + d.get(nkk));
                }
            }
        }

        k = kreuzungen.get(queue.poll());
    }

    return kreuzungen.get((breite-2) + "," + (hoehe-1)).getGesamtDistanz();
}

private char[][] copyMap(char[][] m) {
    char[][] newMap = new char[breite][hoehe];
    for (int y = 0; y < hoehe; y++) {
        for (int x = 0; x < breite; x++) {
            newMap[x][y] = m[x][y];
        }
    }
    return newMap;
}

private void erfasseKante(int x, int y, int startX, int startY) {
    int steps = 1;
    while (map[x][y] != 'x') {
        map[x][y] = '#';
        // left
        if (map[x-1][y] != '#' && !(x-1 == startX && y == startY)) {
            x--;
        }
        // right
        else if (map[x+1][y] != '#' && !(x+1 == startX && y == startY)) {
            x++;
        }
        // up
        else if (map[x][y-1] != '#' && !(x == startX && y-1 == startY)) {
            y--;
        }
        // down
        else if (map[x][y+1] != '#' && !(x == startX && y+1 == startY)) {
            y++;
```

```
        } else {
            // kein Weg vorhanden
            System.out.println("Kein Weg gefunden");
            return;
        }
        steps++;
    }

    Kreuzung k = kreuzungen.get(startX + "," + startY);
    k.addDistanz(x + "," + y, steps);
}
```

Teil 2

Teil 2 ist ein NP-hartes Problem! Dijkstra genügt hier nicht mehr, da Zyklen im Graphen vorkommen können. Lösen lässt es sich u.a. mit einer Breitensuche. Speichere in einer Queue alle nächsten noch zu probierenden Pfade ab. Speichere pro Pfad (eigene Klasse) den bisherigen gelaufenen Weg, die Gesamtdistanz und den aktuellen Kreuzungsnamen.

Lösungsvorschlag Klasse Pfad

<code java> import java.util.ArrayList;

/ * **Beschreiben Sie hier die Klasse Pfad.** * * **@author (Ihr Name)** * **@version (eine Versionsnummer oder ein Datum)** */ **public class Pfad { private ArrayList<String> pfad; private int distanz; private String kreuzung; /**

- Konstruktor für Objekte der Klasse Pfad
- /

public Pfad(ArrayList<String> pfad, int distanz, String kreuzung)

```
  {
      this.pfad = pfad;
      this.distanz = distanz;
      this.kreuzung = kreuzung;
  }
```

```
  public ArrayList<String> getPfad() {
      return this.pfad;
  }
  public int getDistanz() {
      return this.distanz;
  }
  public String getKreuzung() {
      return this.kreuzung;
  }
```

}

</code

Lösungsvorschlag Teil 2

<code java> public long partTwo() {

```
  breite = inputLines.get(0).length();
  hoehe = inputLines.size();
  map = new char[breite][hoehe];
  kreuzungen = new HashMap<String,Kreuzung>();
```

```
  // lies map ein
  for (int y = 0; y < hoehe; y++) {
      String line = inputLines.get(y);
      for (int x = 0; x < breite; x++) {
          char c = line.charAt(x);
          if (c == '#') {
              map[x][y] = '#';
          } else {
              map[x][y] = '.';
          }
      }
  }
```

```
  // setzte x bei Kreuzungen ein und erstelle Knoten
  map[1][0] = 'x';
  kreuzungen.put(1 + "," + 0, new Kreuzung());
  map[breite-2][hoehe-1] = 'x';
  kreuzungen.put((breite-2) + "," + (hoehe-1), new Kreuzung());
```

```
  for (int y = 1; y < hoehe-1; y++) {
      String line = inputLines.get(y);
      for (int x = 1; x < breite-1; x++) {
          if (map[x][y] != '.') {
              continue;
          }
```

```
          int possibleWays = 0;
          if (map[x-1][y] != '#') {
              possibleWays++;
          }
          if (map[x+1][y] != '#') {
              possibleWays++;
          }
          if (map[x][y-1] != '#') {
              possibleWays++;
          }
          if (map[x][y+1] != '#') {
```

```
                possibleWays++;
            }

            if (possibleWays >= 3) {
                map[x][y] = 'x';
                kreuzungen.put(x + "," + y, new Kreuzung());
            }
        }
    }

    originalMap = copyMap(map);

    // laufe von jeder Kreuzung in jede mögliche Richtung -> jede gerichtete
(!) Kante wird erfasst
    for (String k: kreuzungen.keySet()) {
        int kx = Integer.parseInt(k.split(",")[0]);
        int ky = Integer.parseInt(k.split(",")[1]);

        if (kx-1 >= 0 && map[kx-1][ky] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx-1, ky, kx, ky);
        }
        if (kx+1 < breite && map[kx+1][ky] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx+1, ky, kx, ky);
        }
        if (ky-1 >= 0 && map[kx][ky-1] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx, ky-1, kx, ky);
        }
        if (ky+1 < hoehe && map[kx][ky+1] != '#') {
            map = copyMap(originalMap);
            erfasseKante(kx, ky+1, kx, ky);
        }
    }

    Queue<Pfad> queue = new LinkedList<>();
    Pfad p = new Pfad(new ArrayList(), 0, 1 + "," + 0);
    queue.add(p);
    int maxPath = 0;

    while(!queue.isEmpty()) {
        p = queue.poll();
        if (p.getKreuzung().equals((breite-2) + "," + (hoehe-1))) {
            if (p.getDistanz() > maxPath) {
                maxPath = p.getDistanz();
            }
            continue;
        }
        // distanzen von k zu dessen nachbarn
```

```
      Kreuzung k = kreuzungen.get(p.getKreuzung());
      HashMap<String,Integer> d = k.getDistanzen();
      for (String nkk: d.keySet()) { // nachbar-kreuzung-key
          if (!p.getPfad().contains(nkk)){
              ArrayList<String> pfad = new ArrayList(p.getPfad());
              pfad.add(p.getKreuzung());
              queue.offer(new Pfad(pfad, p.getDistanz() + d.get(nkk), nkk));
          }
      }
```

```
  }
```

```
  return maxPath;
```

} </code