Aufgabe 5: Hüpfburg

Team-ID: 00534

Team: Monads

Bearbeiter/-innen dieser Aufgabe:
Leopold Jofer, Benjamin Hantschel, Robert Zeitter

18. November 2022

Inhaltsverzeichnis

Lösungsidee

Idee 1: Bruteforcing

Durch rohe Gewalt lassen sich alle Möglichkeiten errechnen, die mit den Regeln nach einer bestimmten Schrittzahl möglich sind, auf denen Sasha und Mika landen. Jedoch erfordert dies eine Menge Rechenleistung. Zudem gibt es aus programmiertechnischer Sicht keine endliche Anzahl an Schritten (in Echt natürlich schon), wodurch bei nichtvorhandensein eines erfolgreichen Parcours, der Code unendlich weiter laufen würde. Bei einem programmierten Abbruch kann es jedoch passieren, dass Sprungfolgen nicht beachtet werden.

Idee 2

Wenn ein direkter Weg zwischen Sasha und Mika existiert, dann existiert auf diesem auch ein gemeinsamer Knoten auf dem sie beide landen können. Dieser kann dann verwendet werden, um von beiden Seiten, sowohl Sasha, als auch Mika einen Weg zu diesem zu berechnen.

Umsetzung

Mithilfe des A*-Algorithmus wird effizient der kürzeste Weg zwischen Sasha und Mika gefunden, der mit den gegebenen Knoten und Pfaden möglich ist. Dieser ist heuristisch, also verwendet eine Schätzfunktion, um so zielgerecht zu suchen, statt alle möglichen Kombinationen, die existieren, zu berechnen.

Mithilfe des File Picker werden von einer auswählbaren Textdatei die Pfeile Zeile für Zeile eingelesen, dabei aber die erste Zeile ignoriert. Anschließend werden die Beziehung zwischen der ID der Nodes und der geschriebenen Zahlen gespeichert, um mit den IDs im Algorithmus zu arbeiten und später die Zahlen dieser wiedergeben zu können. Dann wird ermittelt, welche Elemente bereits im Graph vorhanden sind und welche nicht, die im letzteren Fall noch hinzugefügt werden, was mit einem Wertepaar aus zwei Adressen möglich gemacht wurde mithilfe des Datentyps Zahlenpaar. Der A*-Algorithmus berechnet mit diesen Daten dann den kürzesten Weg zwischen Sasha und Mika und gibt diesen graphisch wieder. Das kann mit einem Fehler, dass kein Weg existiert erfolgen, oder aber mit Erfolg, bei dem dann noch die kürzeste existierende Route dargestellt wird

Beispiele

Wie die Terminalausgabe bei vorgegebenem Beispiel huepfburg3 aussieht, ist in Abbildung 1 zu sehen:

```
C:\Users\Benny\Downloads\BWINF-main\BWINF-main\src\Aufgabe 5>cargo run --release
    Finished release [optimized] target(s) in 1.54s
     Running `C:\Users\Benny\Downloads\BWINF-main\BWINF-main\src\target\release\bwinf_aufgabe_5.exe`
> Select one file to run using Huepfburg huepfburg3.txt
Route found: 5 steps
1 → 10 → 4 ← 5 ← 2
```

Abbildung 1: Ausgabe der Aufgabe 5 bei Verwendung der Datei huepfburg3.txt

Zuerst wird angegeben, dass eine Route gefunden wurde. Wenn eine existiert, folgt eine auflistung der zu gehenden Route von Sasha und Mika. Beide Startknoten sind dabei farblich eingefärbt und der Zielknoten ist dabei grün.

Eine kompliziertere Route ist in Abbildung 2 sichtbar, bei der die Datei huepfburg4 verwendet wurde:

```
C:\Users\Benny\Downloads\BWINF-main\BWINF-main\src\Aufgabe 5>cargo run --release
    Finished release [optimized] target(s) in 0.09s
     Running `C:\Users\Benny\Downloads\BWINF-main\BWINF-main\src\target\release\bwinf_aufgabe_5.exe`
> Select one file to run using Huepfburg huepfburg4.txt
Route found: 18 steps
1 → 99 → 98 → 97 → 96 → 86 → 76 → 75 → 74 → 73 ← 63 ← 53 ← 43 ← 33 ← 23 ← 22 ← 12 ← 2
```

Abbildung 2: Ausgabe der Aufgabe 5 bei Verwendung der Datei huepfburg4.txt

Quellcode

```rust
fn main() -> Result<(), Box<dyn Error>> {
  let lines = shared::file_picker::read_lines("Huepfburg")?;
  let (graph, elements) = parse_file(lines);

  let sasha = *elements.get_by_left(&1u8).unwrap();
  let mika = *elements.get_by_left(&2u8).unwrap();

  let path = astar(&graph, sasha, |node| node == mika, |_| 0, |_| 0);
  display(path, elements);

  Ok(())
}
```

```rust
fn parse_file(file: Lines<BufReader<File>>) -> (UnGraph<u8, ()>, BiMap<u8,
NodeIndex>) {
  let mut edges: BiHashMap<u8, NodeIndex> = BiMap::new();
  let mut graph = UnGraph::<u8, ()>::new_undirected();
  let mut file = file.skip(1);

  while let Some(Ok(line)) = file.next() {
    let pair = Pair::from_str(&line).unwrap();

    let indices = pair.map(|n| match edges.get_by_left(&n) {
      Some(edge) => *edge,
      None => {
        let idx = graph.add_node(*n);
        edges.insert(*n, idx);
        idx
      }
    });

    graph.add_edge(indices.0, indices.1, ());
  }

  return (graph, edges);
}
```