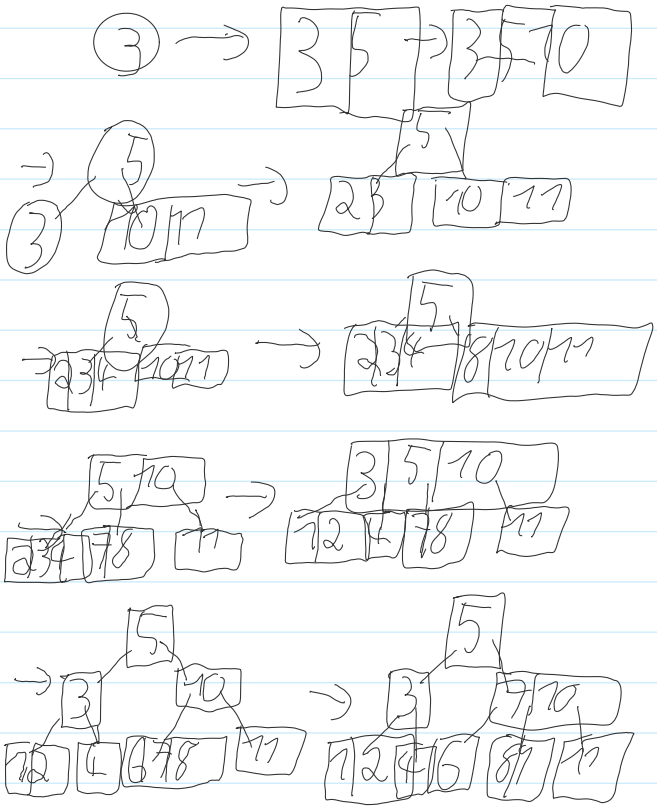
AuD Hausübung 9

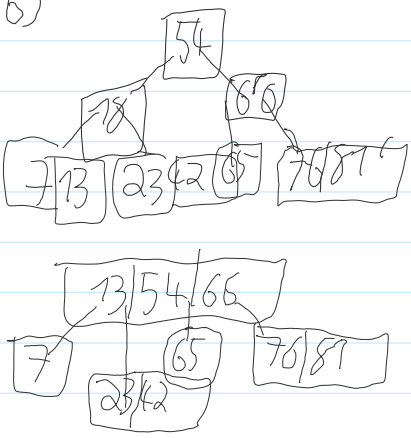
Donnerstag, 27. Juni 2019

Nicolas Petermann, Jonas Franz, Julian Imhof

1+1a)



b)



c) Die Knoten haben mindestens $t-1$ Knoten und höchstens $2t-1$ pro Knoten. Die Invarianten werden durch das Verschmelzen, Rotieren und verschieben eingehalten

d) Man muss durch diese Änderungen die Knoten eher verschmelzen oder verschieben/rotieren. Der B-Baum geht auch mehr in die Breite

H2)

```
/**
 * Erstelle Karthesiches Produkt von u, v, w und finde Teilmenge
 */
public static List<List<Integer>> getM(List<Integer> u, List<Integer> v, List<Integer> w, int q) throws Exception {
    List<List<Integer>> m = new ArrayList();
    for (int u0 = 0; u0 < u.size(); u0++) {
        for (int v0 = 0; v0 < v.size(); v0++) {
            for (int w0 = 0; w0 < w.size(); w0++) {
                List<Integer> coords = new ArrayList();
                coords.add(u.get(u0));
                coords.add(v.get(v0));
                coords.add(w.get(w0));
                m.add(coords);
            }
        }
    }

    return backtrack(m, new ArrayList(), q);
}

/**
 * Finde Teilmenge via backtracking
 */
public static List<List<Integer>> backtrack(List<List<Integer>> m, List<List<Integer>> curr, int q) {
    if (curr.size() == q)return curr;
    else {
        for (int i = 0; i < m.size(); i++) {
            List<Integer> m0 = m.get(i);
            if (curr.stream().filter(x -> x.get(0) == m0.get(0) || x.get(1) == m0.get(1) || x.get(2) == m0.get(2)).count() > 0)continue;
            curr.add(0, m0);
            List<List<Integer>> c0 = backtrack(m, curr, q);
            if (c0.size() == q)return c0;
            else curr.remove(0);
        }
    }

    return curr;
}
```

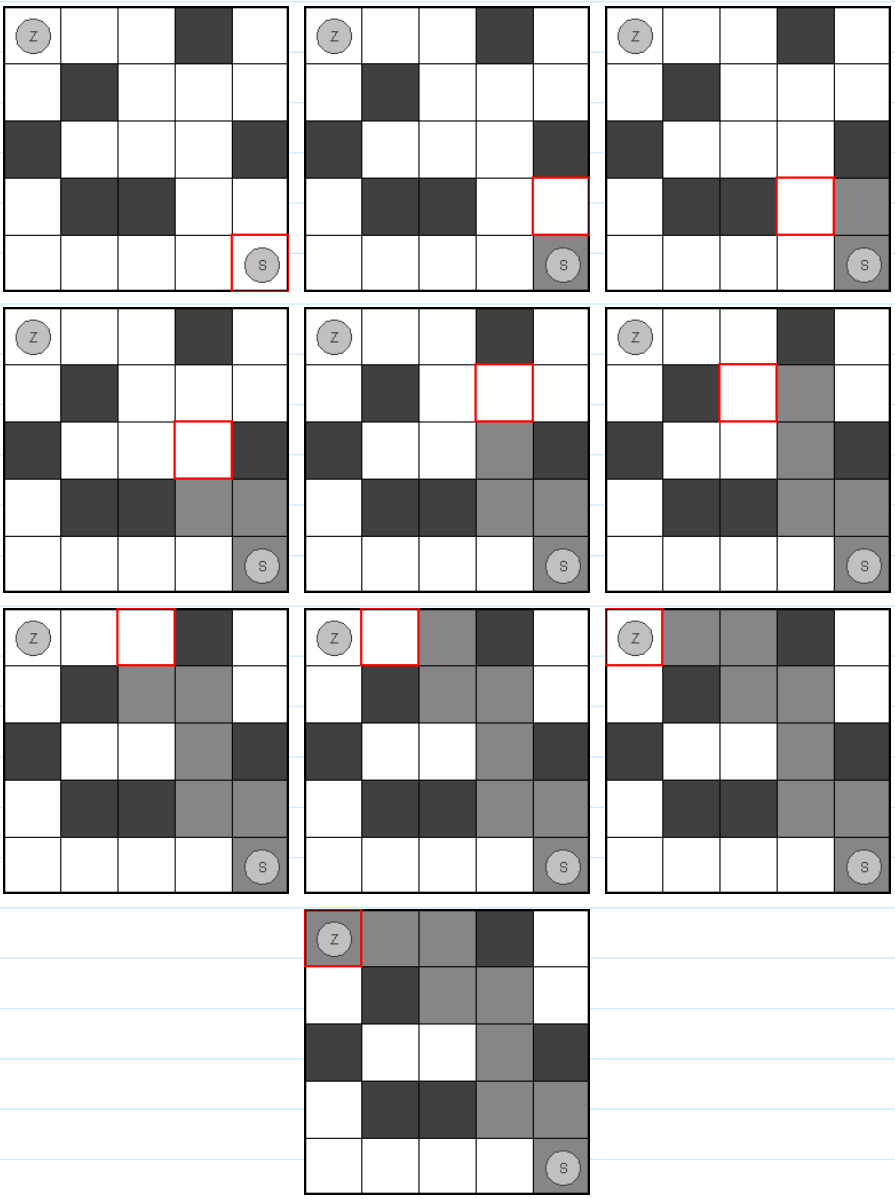
H3)

a)

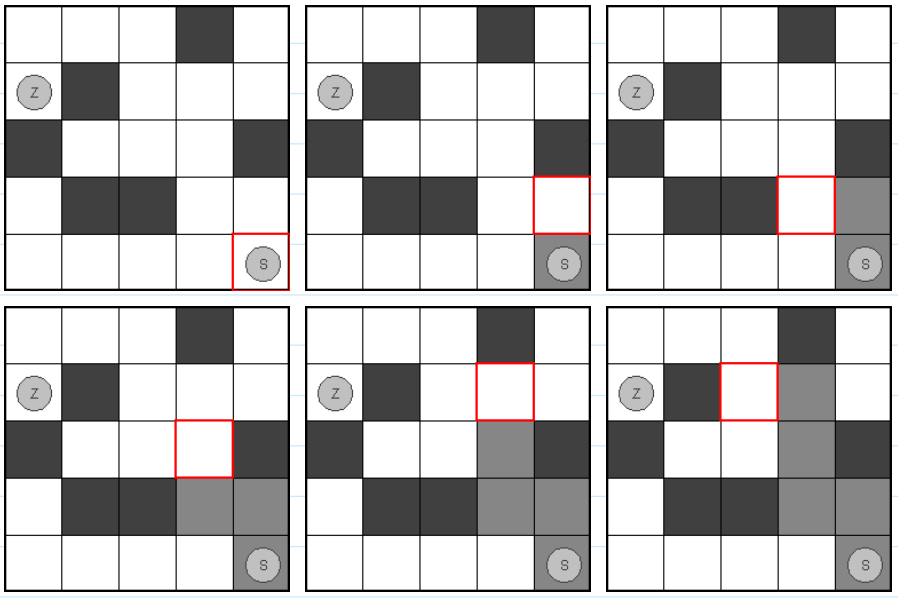
(i)

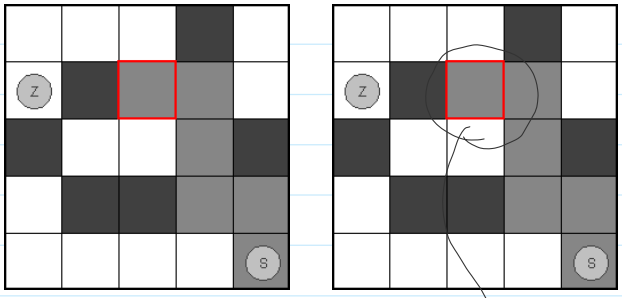
```
/**
 * Calculate Manhattan Distance
 */
public int getDist(Point p0, Point p1) {
    return Math.abs(p0.x - p1.x) + Math.abs(p0.y - p1.y);
}

public void hillClimber() {
    while (this.curr != this.end) {
        this.render();

        grid [curr.x] [curr.y] = 2;

        try {
            Thread.sleep(250);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        double currDistance = this.getDist(curr, end);
        double lowest = currDistance;
        int index = -1;

        if (curr.y > 0 && this.grid [curr.x] [curr.y - 1] == 0 && this.getDist(end, new Point(curr.x, curr.y - 1)) < lowest) {
            lowest = this.getDist(end, new Point(curr.x, curr.y - 1));
            index = 2;
        }
        else if (curr.x > 0 && this.grid [curr.x - 1] [curr.y] == 0 && this.getDist(end, new Point(curr.x - 1, curr.y)) < lowest) {
            lowest = this.getDist(end, new Point(curr.x - 1, curr.y));
            index = 0;
        }
        else if (curr.y < 4 && this.grid [curr.x] [curr.y + 1] == 0 && this.getDist(end, new Point(curr.x, curr.y + 1)) < lowest) {
            lowest = this.getDist(end, new Point(curr.x, curr.y + 1));
            index = 3;
        }
        else if (curr.x < 4 && this.grid [curr.x + 1] [curr.y] == 0 && this.getDist(end, new Point(curr.x + 1, curr.y)) < lowest) {
            lowest = this.getDist(end, new Point(curr.x + 1, curr.y));
            index = 1;
        }

        if (index == 0)curr.x--;
        else if (index == 1)curr.x++;
        else if (index == 2)curr.y--;
        else if (index == 3)curr.y++;
    }
}
```

(ii)



(iii)



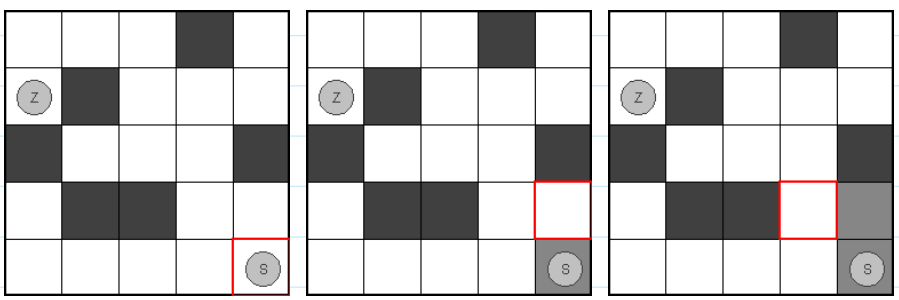


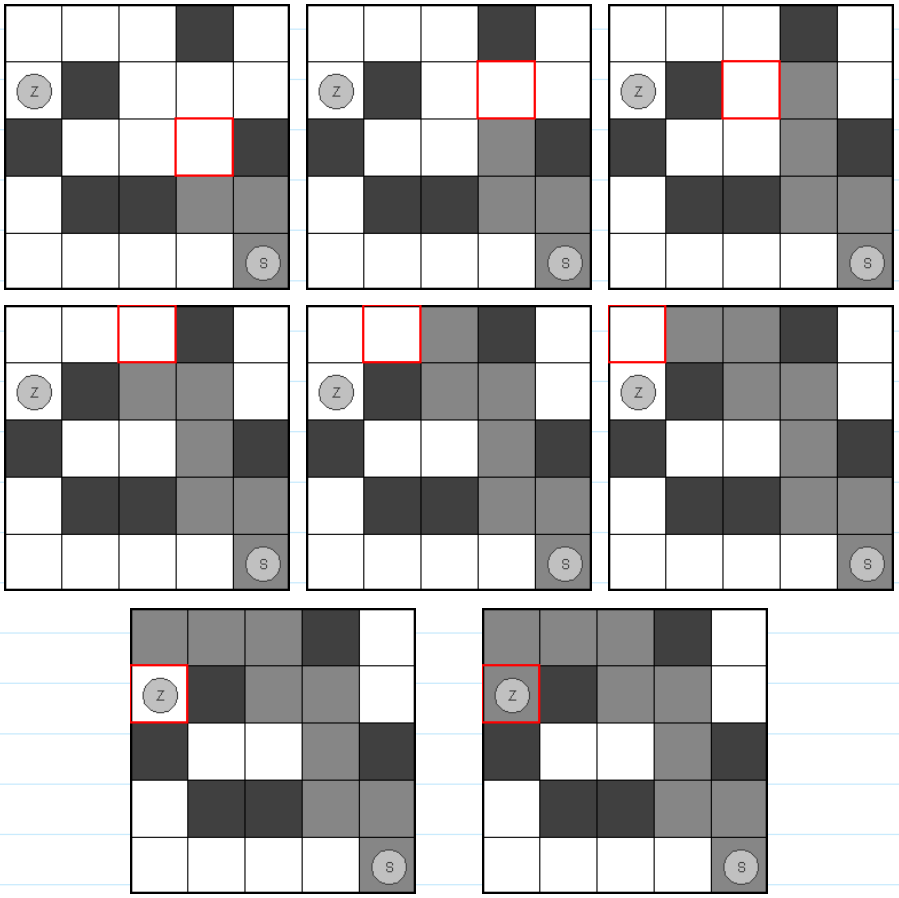
Der Algorithmus erreicht nicht das Ziel, da er seiner Meinung nach schon am nächsten am Ziel ist. Egal in welche Richtung er gehen würde, die Distanz zum Ziel würde sich vergrößern. Man könnte einbauen, dass der Algorithmus wenn möglich standardmäßig einen Schritt nach oben machen würde und dann vergleicht ob es eine bessere Alternative gibt.

```
public void hillClimber() {
    while (this.curr != this.end) {
        this.render();

        grid [curr.x] [curr.y] = 2;

        try {
            Thread.sleep(250);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        double currDistance = this.getDist(curr, end);
        double lowest = currDistance;
        int index = -1;

        if (curr.y > 0 && this.grid [curr.x] [curr.y - 1] == 0) {
            lowest = this.getDist(end, new Point(curr.x, curr.y - 1));
            index = 2;
        }
        if (curr.x > 0 && this.grid [curr.x - 1] [curr.y] == 0 && this.getDist(end, new Point(curr.x - 1, curr.y)) < lowest) {
            lowest = this.getDist(end, new Point(curr.x - 1, curr.y));
            index = 0;
        }
        if (curr.y < 4 && this.grid [curr.x] [curr.y + 1] == 0 && this.getDist(end, new Point(curr.x, curr.y + 1)) < lowest) {
            lowest = this.getDist(end, new Point(curr.x, curr.y + 1));
            index = 3;
        }
        if (curr.x < 4 && this.grid [curr.x + 1] [curr.y] == 0 && this.getDist(end, new Point(curr.x + 1, curr.y)) < lowest) {
            lowest = this.getDist(end, new Point(curr.x + 1, curr.y));
            index = 1;
        }

        if (index == 0)curr.x--;
        else if (index == 1)curr.x++;
        else if (index == 2)curr.y--;
        else if (index == 3)curr.y++;
    }
}
```

Keine Zusätzliche Bedingung

Durch diese Anpassung erreicht der Algorithmus auch sein Ziel.





b)

(i)

```
/**
 * Get function value of f(x,y)
 */
public double f(double x, double y) {
    return Math.pow(2, -Math.pow(x, 2) - Math.pow(y, 2) + 1) +
    Math.pow(3 * 2, -Math.pow(x, 2) - Math.pow(y, 2) + (2 * x)
    * (4 * y) - 5);
}

public double hillClimber2(double x0, double y0) {
    while (true) {
        double x1 = x0, y1 = y0;
        double f0 = this.f(x0, y0);
        double f1 = f0;
        for (double i = x0 - 1; i < x0 + 1; i++) {
            for (double a = y0 - 1; a < y0 + 1; a++) {
                if (this.f(i, a) > f0) {
                    f0 = this.f(i, a);
                    x1 = i;
                    y1 = a;
                }
            }
        }

        if ((x1 == x0 && y1 == y0) || f1 == f0)return f0;

        x0 = x1;
        y0 = y1;
    }
}
```

(ii) Für $S_0 = (-2, -4)$:

1.0721394614761022E63
1.1231009685908055E105
2.5609513428721644E156
1.2711560155617886E217
1.3734453760407511E287
Infinity
Infinity
Infinity

Es gibt kein globales Extremum

Für $S_1 = (4, 4)$

6.482817337415922E70
6.482817337415922E70

Falscher Wert, lokales Extremum