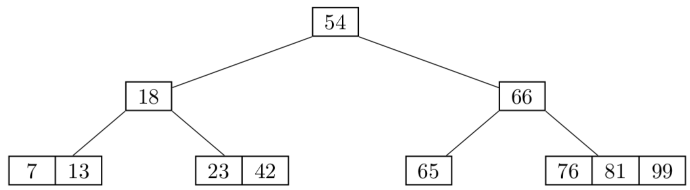
H1 (B-Bäume) **(2+1+1+1 Punkte)**

(a) Fügen Sie der Reihe nach Knoten mit den Schlüsseln $3, 5, 10, 11, 2, 4, 8, 7, 1, 6, 9$ in einen leeren B-Baum mit Grad $t = 2$ ein. Verwenden Sie dabei die Algorithmen aus der Vorlesung. Skizzieren Sie Ihr Zwischenergebnis nach jeder Einfügeoperation.

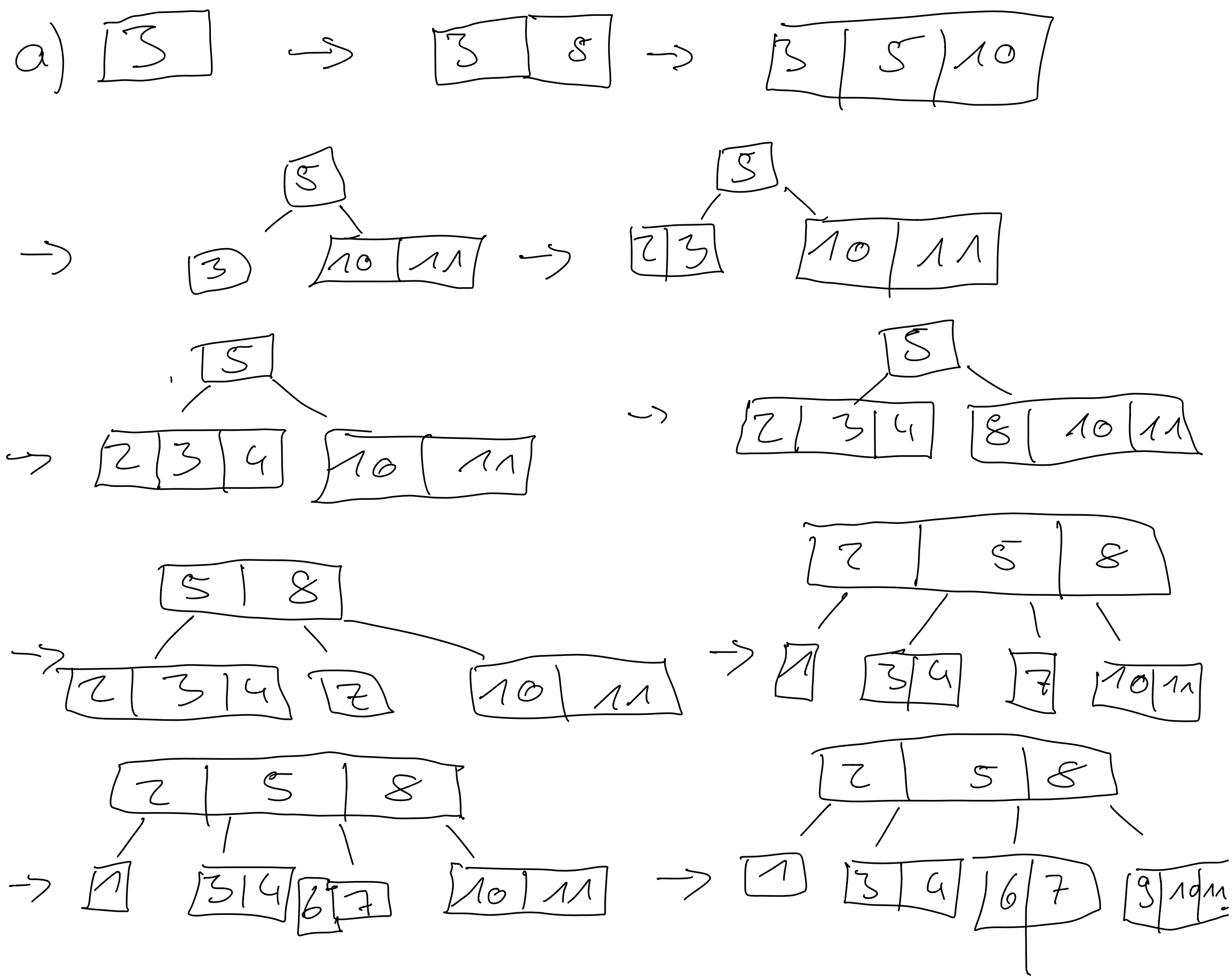
(b) Löschen Sie die Schlüssel 99 und 18 in gegebener Reihenfolge aus dem untenstehenden B-Baum mit Grad $t = 2$, und zeichnen Sie den daraus resultierenden B-Baum nach jeder Löschoperation. Verwenden Sie dabei die Algorithmen aus der Vorlesung.



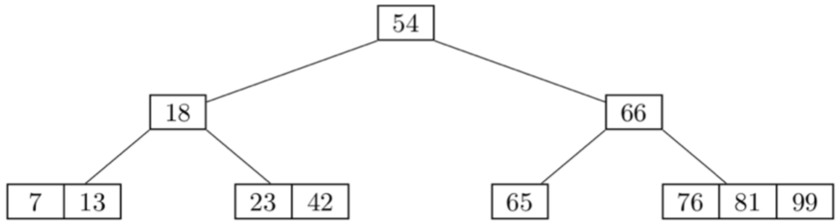
(c) Wie lauten die Invarianten für das Einfügen und Löschen eines Schlüssels in einem B-Baum mit Grad t? Und wie wird sichergestellt, dass die Invarianten eingehalten werden?

(d) Nehmen Sie für einen Moment an, dass die Knoten eines B-Baumes mit Grad t mindestens t Schlüssel statt nur $t - 1$ Werte enthalten müssen. Welche Auswirkungen hat dies auf das Einfügen und Löschen von Schlüsseln in einem B-Baum?

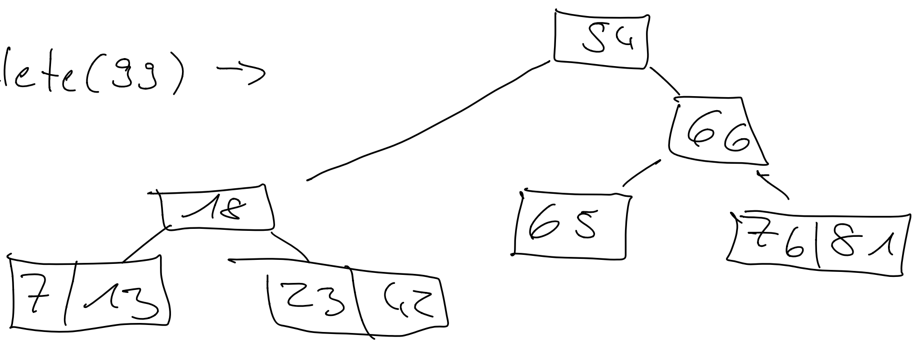
a)



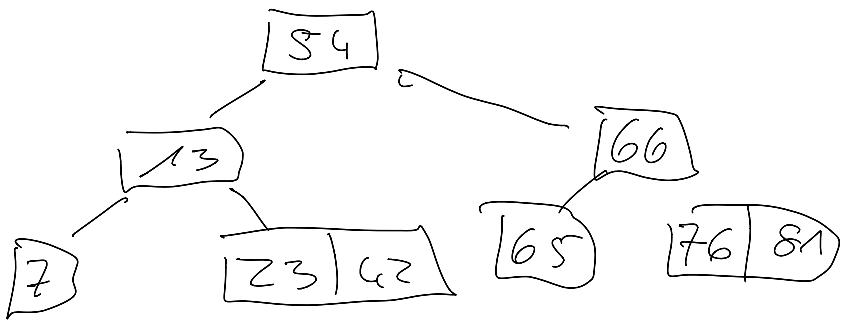
b)



delete(99) →



delete(18) →



c) B-Baum Eigenschaften:

1) zw. t-1 & 2t-1 Werte (Wurzel 1 & 2t-1)
2) Werte aufsteigend in Knoten
3) alle gleiche Höhe Blätter
4) innere Knoten n+1 Kinder: $\forall k_j \in j$ $k_1 \leq key[1]$.

Einfügen:

1) Wenn Blatt < 2t-1 Werte → Einfügen ✓

Sonst

Teile in zwei Blätter mit $t-1$ und

Füge mittlern Wert in Elternknoten.

Warum wird keine Baum Regel verletzt?

→ Da "Einfügen" rekursiv nach oben funktioniert und d. Baum vor dem Einfügen schon alle B-Baum Eigenschaften erfüllt haben muss folgt, dass auch nach dem aufruf v. Insert alle Eigenschaften erfüllt bleiben.

Löschen:

Wenn Blatt $> t-1$ → löschen → keine Regel verletzt

wenn Blatt $= t-1$ & linker o. rechter Geschwister $\geq t$

↳ rotiere

↳ keine Regel verletzt (siehe VL. Beweis)

wenn Blatt $= t-1$ & linker/rechter Geschwister $= t-1$

↳ verschmelze ⇒ keine Regel verletzt, da maximal $2t-2$ Werte und $t-1 < 2t-2 < 2t-1$ ✓

Wenn Blatt $= t-1$ & Kind $> t-1$ → größten Wert v. linken Kind

bzw. Kleinsten
v. Rechten Kind
in Elternknoten
⇒ jeweils mind $t-1$
Werte ✓

Wenn Blatt $= t-1$ & Beide Kinder $= t-1$ → verschmelze Kinder
⇒ $t-1 + t-1 = 2t-2$ ✓

⇒ Alle Regeln eingehalten □

d)

H2.

```
MENGE-UNIQUE (M, q)
    let points be a new List of arrays of int
    IF MENGE-UNIQUE-BACKTRACKING (m, q, points, 0, 0)
        RETURN points
    ELSE
        ERROR

MENGE-UNIQUE-BACKTRACKING (m, q, points, s, a)
    IF points.length > 1
        IF SIND-VERSCHIEDEN (points)
            RETURN TRUE
        ELSE
            RETURN FALSE
    ELSE
        WHILE points.length < q
            IF s < m.length
                points.add (m [s])
            ELSE
                points.remove (points.length - 1)
                a++
                s=a
                if a == M.length
                    RETURN FALSE
                ELSE
                    points.add (M[a])
            IF NOT MENGE-UNIQUE-BACKTRACKING (m, q, points, s+1, a)
                IF points.length > 0
                    points.remove (points.length - 1)
                ELSE
                    RETURN FALSE
            s++
    RETURN TRUE
```

```
SIND-VERSCHIEDEN (points)
    IF points.length < 1
        RETURN FALSE
    FOR i = 0 TO Points[0].length - 1
        FOR j=0 TO points.length - 1
            FOR K=j+1 TO points.length-1
                IF points [j][i] == points [K][i]
                    RETURN FALSE
    RETURN TRUE
```

H3. BERGSTEIGERALGORITHMUS

i)

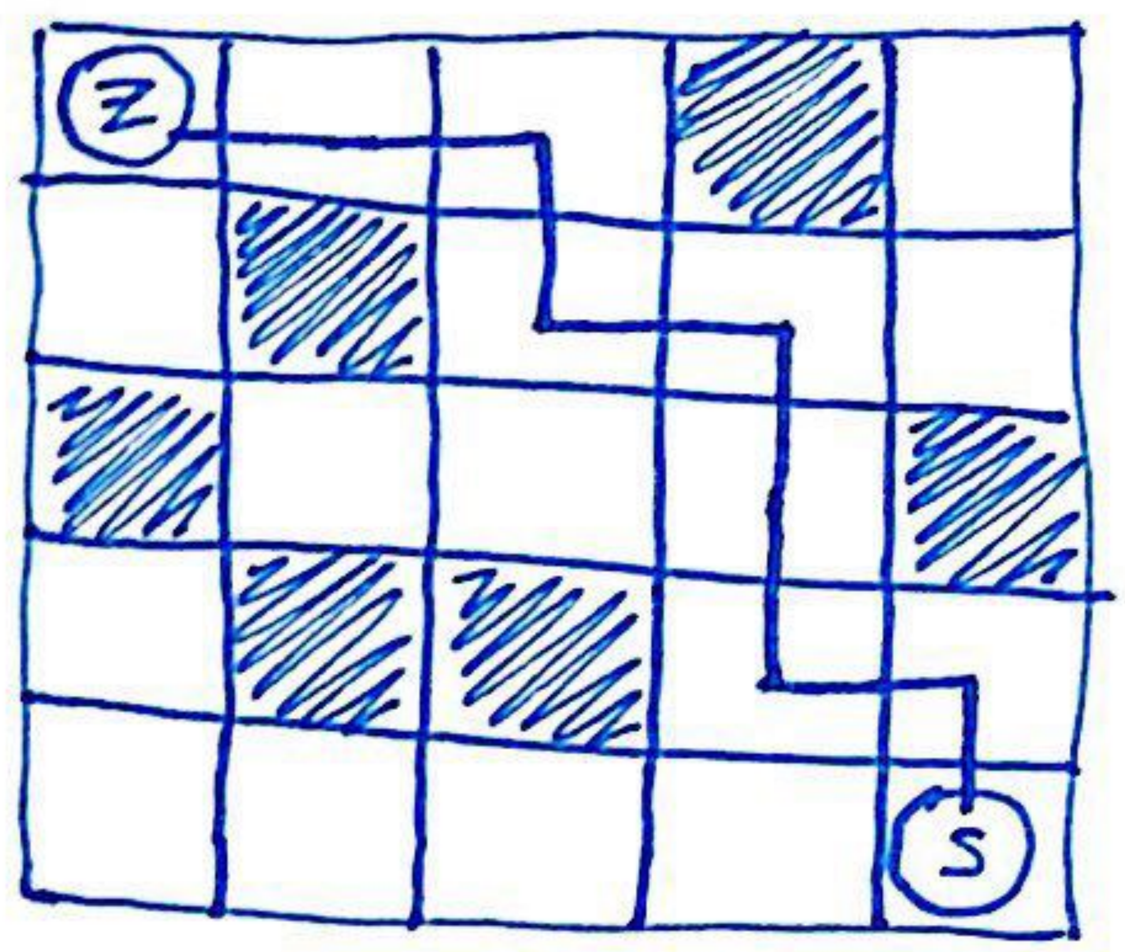
```
BERGSTEIG( panel , x, y , zx , zy)
    ax =x
    ay=y
    dist= 2 · panel[0].length     // Maximaler Abstand.
    adist = f(x,y,zx,zy)
    WHILE adist < dist
        dist=adist
        FOR i = 1 DOWN TO -2 // Mit 2 Sprung; also i= 1 und -1
            IF ((i==1 AND Y<4) OR(i==-1 AND y>0)) AND panel[x][y+i]
                IF f(x, y+i ,zx, zy)< adist
                    ax=x
                    ay= y+i
                    adist= f(ax,ay, zx, zy)
            IF ((i=-1 AND x<4) OR (i==1 AND x>0)) AND panel[x-i][y]
                IF f(x-i, y, zx, zy) < adist
                    ax = x-i
                    ay= y
                    adist= f(ax, ay, zx,zy)
        x=ax
        y=ay
    return [x,y]

f (x1, y1, x2, y2)
    RETURN |x1 - x2| + |y1 - y2|
```

ii) Wir betrachten hier die Positionen von 0 bis 4 und nicht von 1 bis 5 wie auf dem Bild, weil unser Algorithmus so arbeitet. Wir könnten aber die Lösung einfach ändern, um die gewünschten Positionen zwischen 1 und 5 zu bekommen. ~~Es~~ Es ist Analog.

Wir fangen also in [4,0] an (analog zu [5,1])

[4,0] → [4,1] → [3,1] → [3,2] → [3,3] → [2,3] → [2,4] → [1,4] → [0,4] → Ziel wird erreicht!



iii)

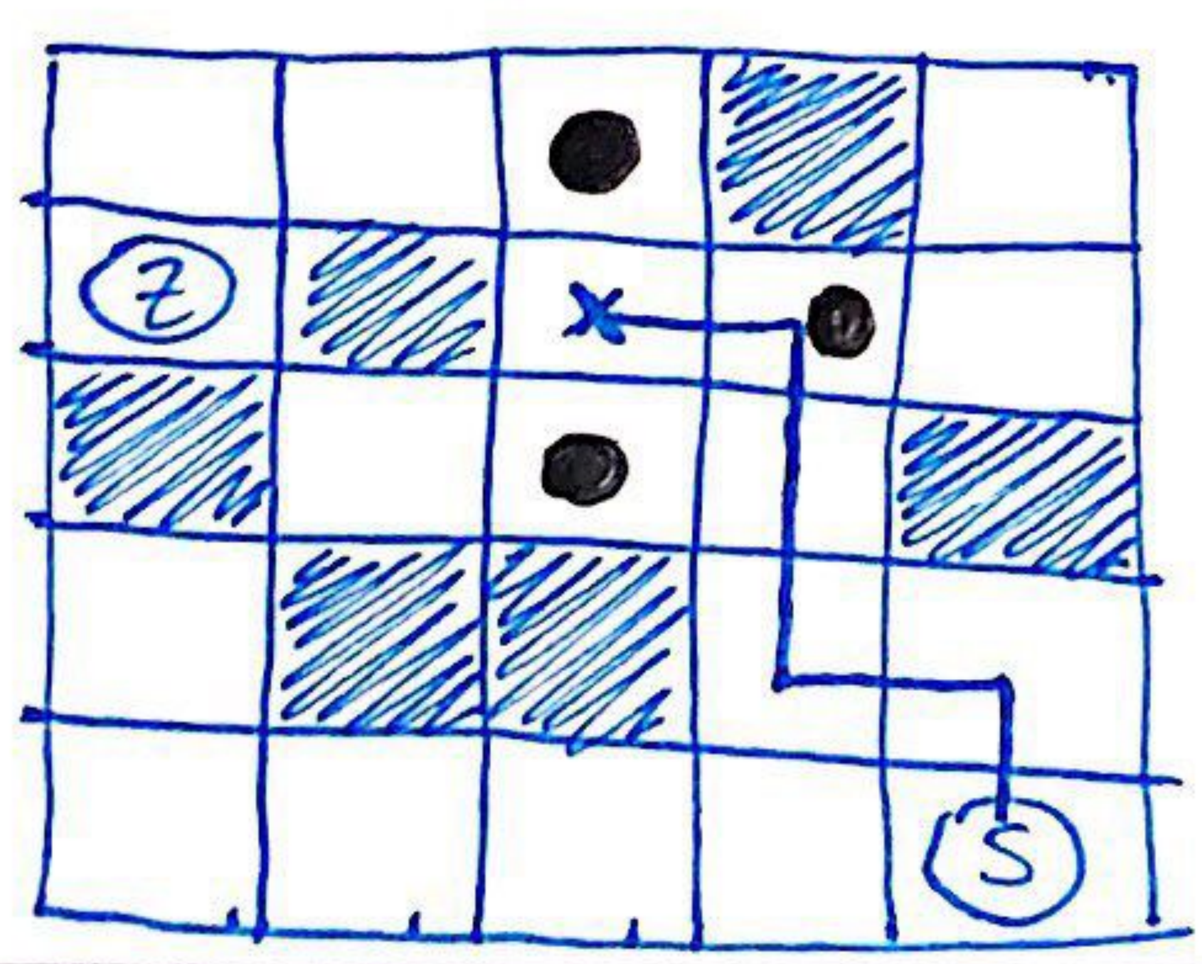
Wir müssen nichts anpassen, eifach nur einen neuen Punkt zx, zy geben, in diesem Fall zx = 0 ~~und~~ zy = 3

panel bleibt unverändert, weil es gleich wie links ist.

Wir erreichen in diesem Fall das Ziel nicht, wir bleiben in [2,3] gesperrt.

Von [2,3] aus gibt es keine Position 1 entfernt, die näher ist zu Z

ich habe die mögliche Positionen schwarz markiert →



ii)

FÜR $S_0 = (-2,-4) \to (-1,-3) \to (0,-2) \to (0,-1) \to \boxed{(0,0)}$

FÜR $S1 = (4,4) \to (3,3) \to (2,2) \to \boxed{(1,2)}$

Warum sind sie verschieden? Weil es zwei Lokale Maxima gibt. Wenn wir die Funktion zeichnen:

Es ist entscheidend, wo wir anfangen, um eines oder anderes Maximum zu finden