AuD, Gü, 06.01.11

I. Stacks und Queues
II. Berechnung einer konvexen Hülle mit Hilfe eines Stacks
III. Verkettete Listen
IV. Binäre Suchbäume
V. AVL-Bäume

⇒ „alte" Klausuraufgaben

I. Stacks und Queues

A) Queues Q

1	2	3	4	5	6	7
			13	7	2	

head[Q]=4 tail[Q]=7

ENQUEUE(Q,8)

1	2	3	4	5	6	7
			13	7	2	8

tail[Q]=1 head[Q]=4

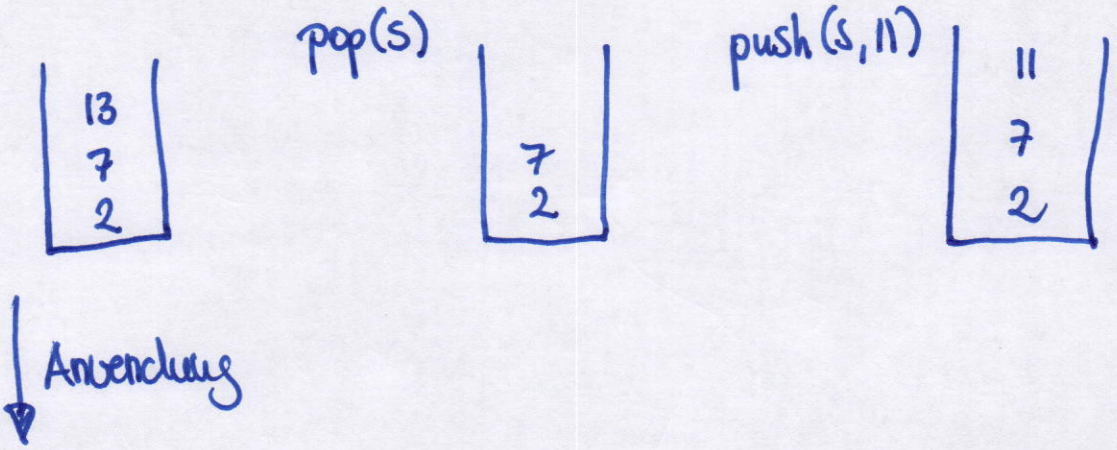
DEQUEUE(Q)

1	2	3	4	5	6	7
				7	2	8

tail[Q]=1 head[Q]=5

→ Aufgabe 5 ...!

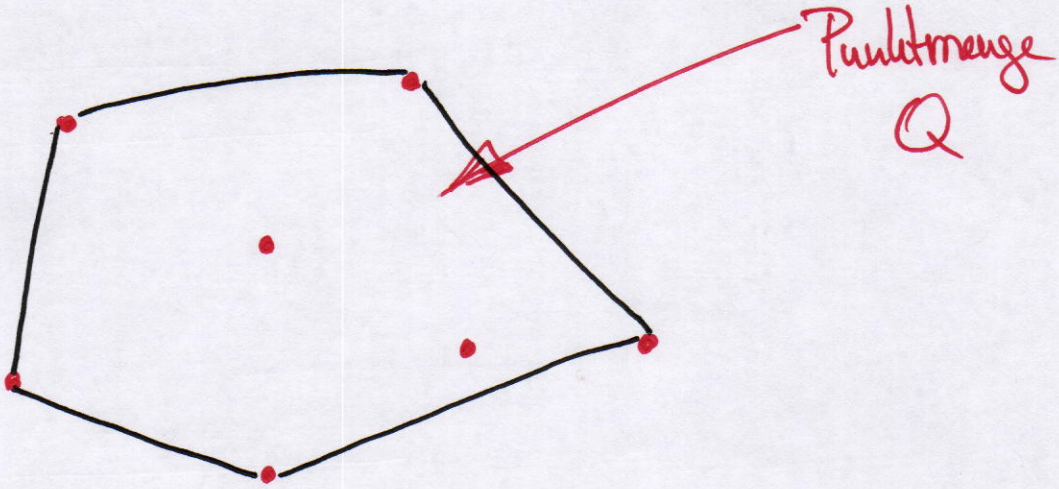
B) Stack S



Anwendung

II. Berechnung einer konvexen Hülle mit Hilfe eines Stacks

↳ Was ist eine konvexe Hülle?



konvexe Hülle von Q: (CH(Q))

kleinste konvexe Polygon P, so dass jeder Punkt aus Q
- auf Ps Rand ODER
- in Ps Innerem

liegt.

Vorstellung: Punkte – Nägel auf Brett

konvexe Hülle – Gummiband um alle Nägel

Wir haben Q gegeben und wollen CH(Q) berechnen.

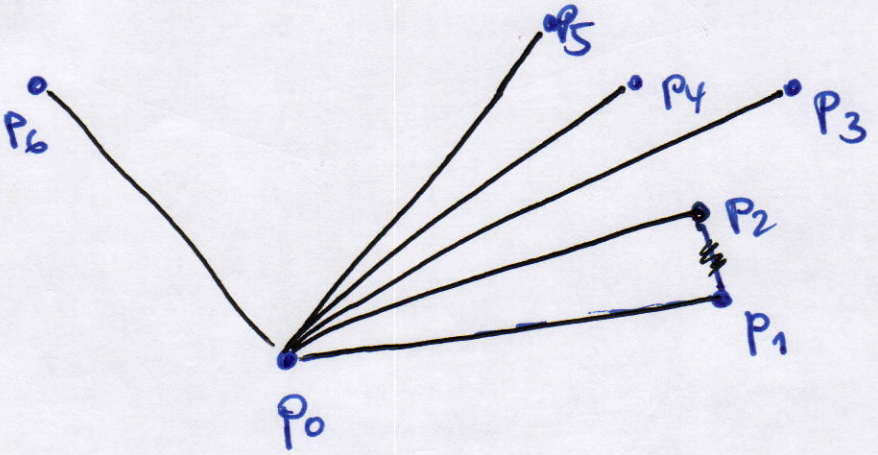
↳ Algorithmus, den wir betrachten, heißt Grahams Scan.

Nutzt: einen „Rotationssweep"

Dazu: verarbeitet ~~Kanten~~ Punkte in Reihenfolge der Polarwinkel, die sie mit Referenz~~knoten~~punkt bilden

↳ hier: Referenzpunkt, der mit kleinster y-Koordinate (Tie-breaking: linkester solcher)

Das heißt??



Dann: Berechnung von CH(Q) indem Kandidatenpunkte als Stack vorgehalten werden.

- jeder ~~Knoten~~ Punkt des Inputs wird einmal auf Stack „gepusht"
- ~~Knoten~~ Punkte, die nicht Knoten von CH(Q) sind werden irgendwann vom Stack „gepopt"

↳ Am Ende des Alg. enthält S genau die Knoten von CH(Q) in Reihenfolge: gegen den UZS auf dem Rand

verwendete Funktionen:

TOP(S): gibt Punkt oben auf S wieder, ohne S zu ändern

NEXT-TO-TOP(S): ——"—— unter obersten —"—, ——"——

$|Q| \geq 3$!

↳ Folien

Without proof:

<u>Theorem:</u> GRAHAM-SCAN ist korrekt.

$O(n \lg n)$, $n = |Q|$

~~Lego geht schneller~~

GRAHAM-SCAN(Q)

1.) Sei p_0 der Knoten in Q mit kleinster y-Koordinate, oder der linkester solcher Punkte (im Fall, dass mehrere solche existieren)

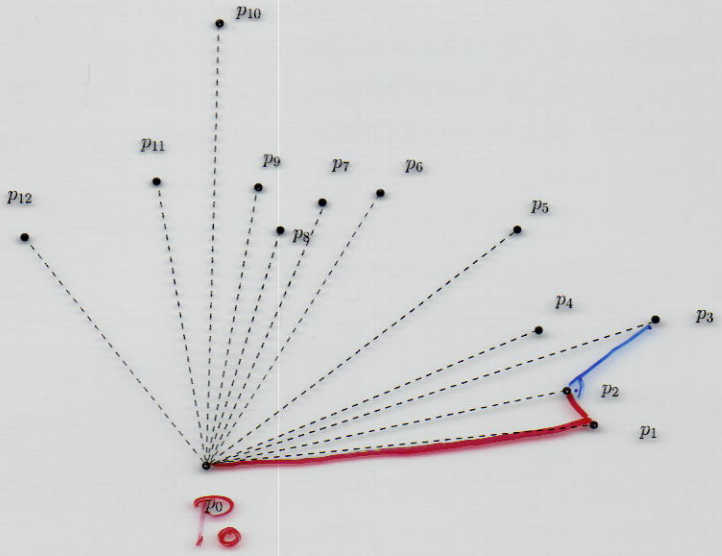
2.) Seien $p_1, p_2, \ldots, p_m$ die verbleibenden Punkte in Q, sortiert nach dem Polarwinkel, gegen den UZS um p_0
haben mehrere Punkte den gleichen Winkel, entferne alle bis auf den mit größtem Abstand zu p_0

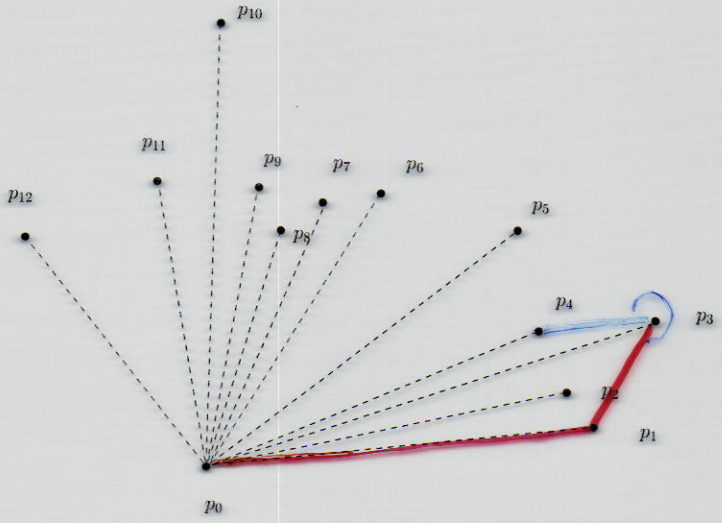
3.) PUSH(p_0, S)

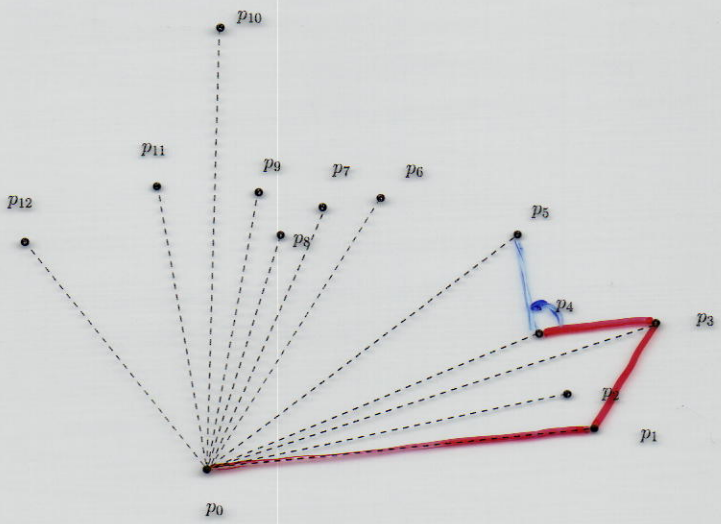
4.) PUSH(p_1, S)

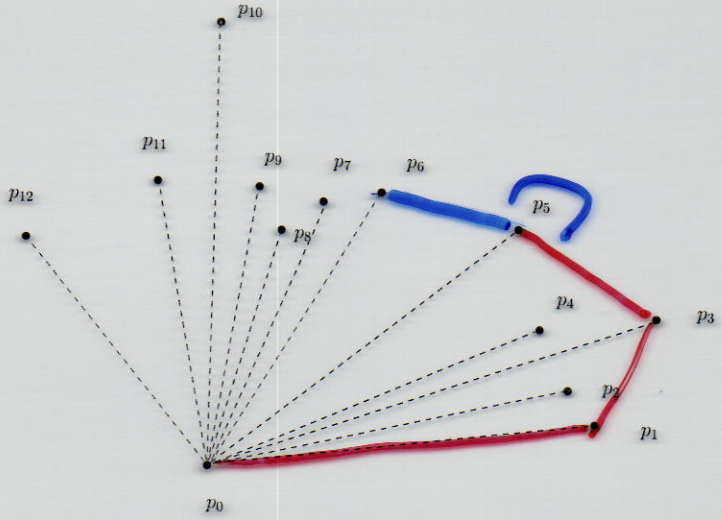
5.) PUSH(p_2, S)

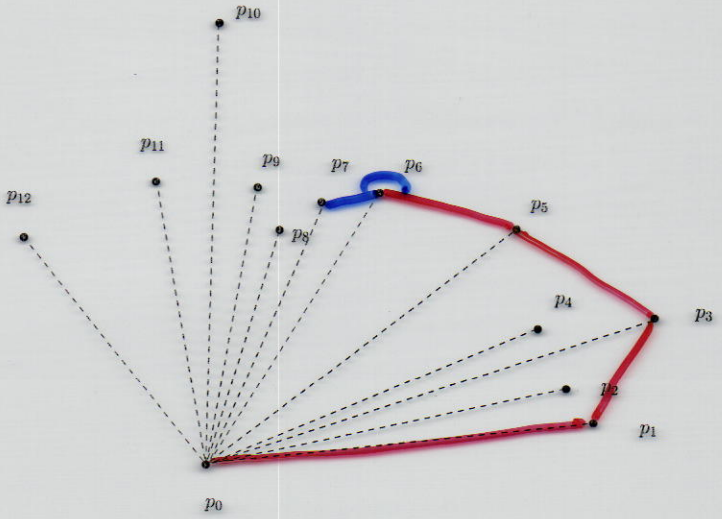
FOR(i=3 **TO** m)
    DO WHILE der Winkel zwischen NEXT-TO-TOP(S), TOP(S) und $\cancel{P_i}$ p_i macht eine Nicht-Links-Drehung
        DO POP(S)
    PUSH(p_i, S)
return S

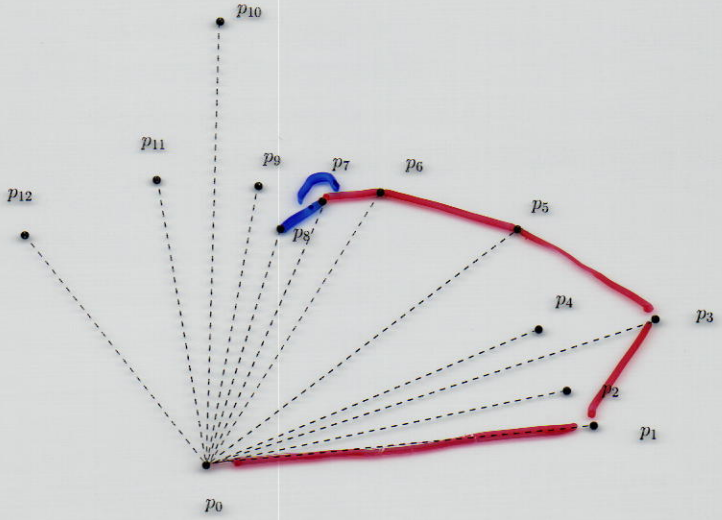

p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_5
p_8
p_4
p_3
p_2
p_1
p_0
P_0

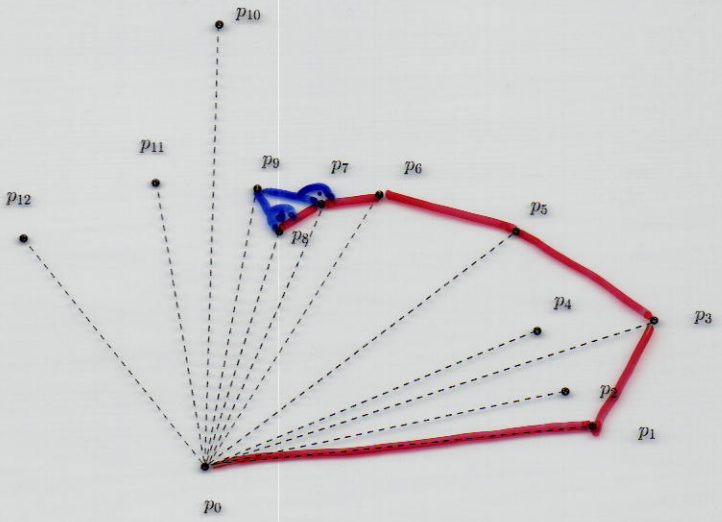

p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_5
p_8
p_4
p_3
p_2
p_1
p_0

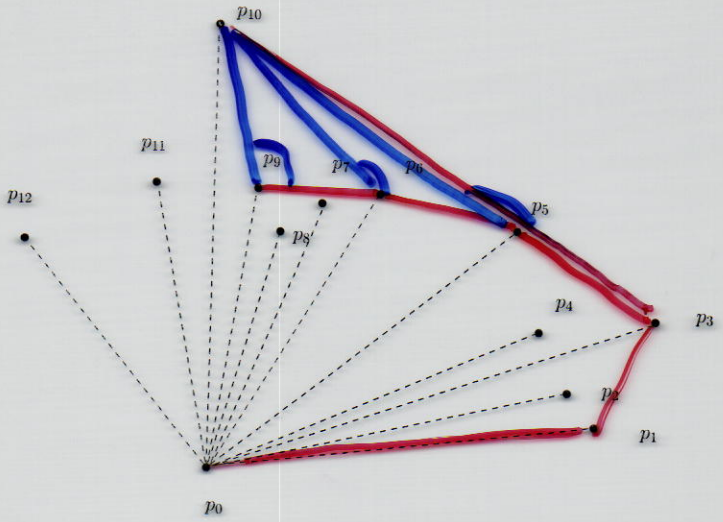

p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_5
p_8
p_4
p_3
p_2
p_1
p_0

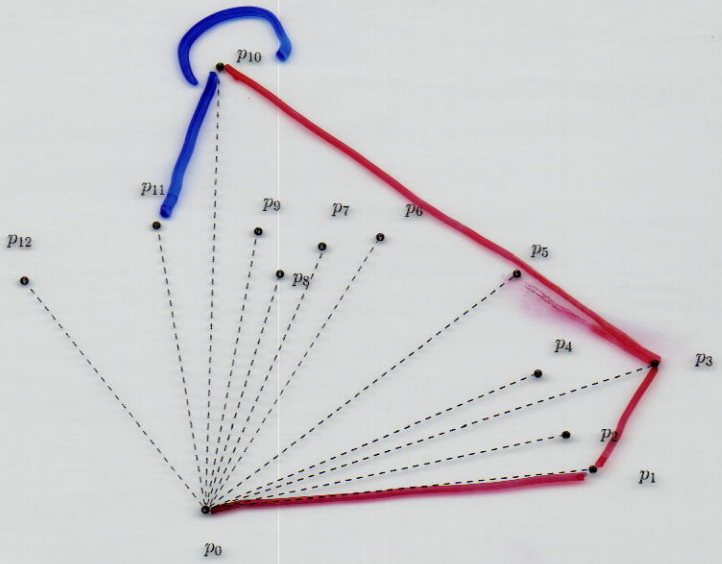

p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_5
p_8
p_4
p_3
p_2
p_1
p_0

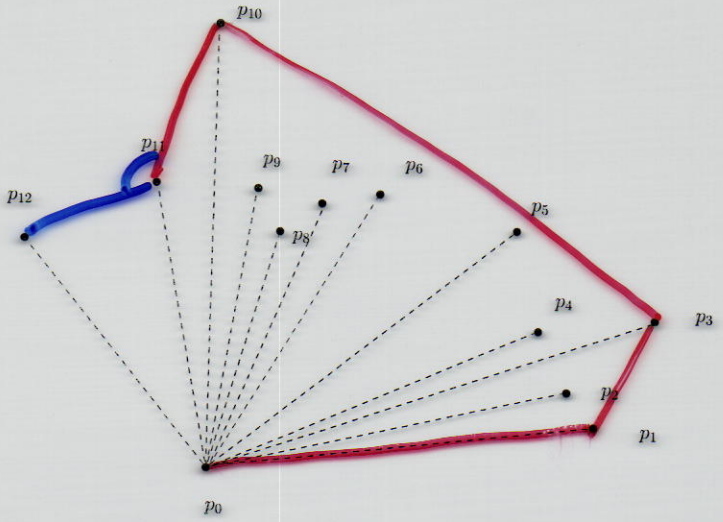

p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_5
p_8
p_4
p_3
p_2
p_1
p_0

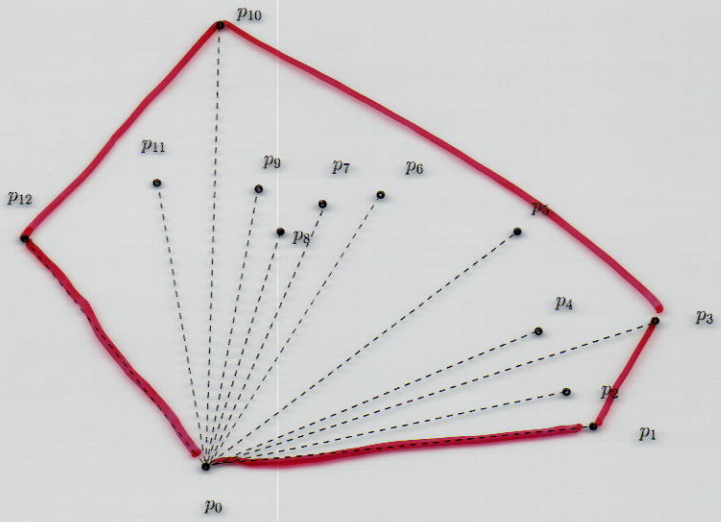

p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_5
p_8
p_4
p_3
p_2
p_1
p_0

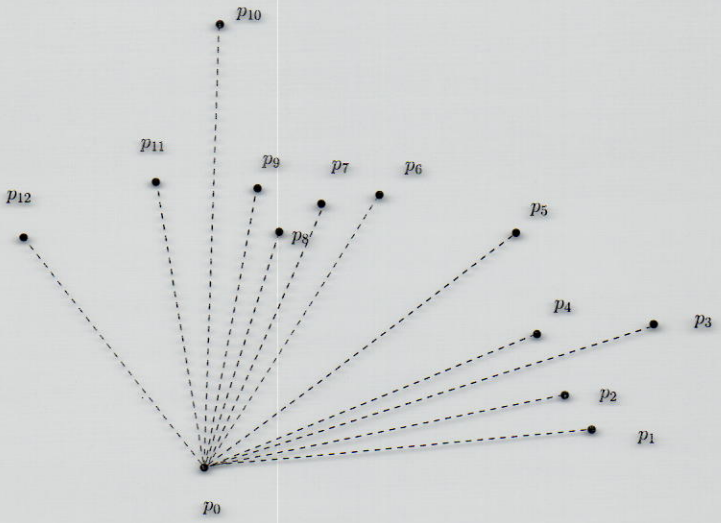

p_{10}
p_{11}
p_{12}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_2
p_1
p_0


p_{10}
p_{11}
p_{12}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_2
p_1
p_0


p_{10}
p_{11}
p_{12}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_2
p_1
p_0

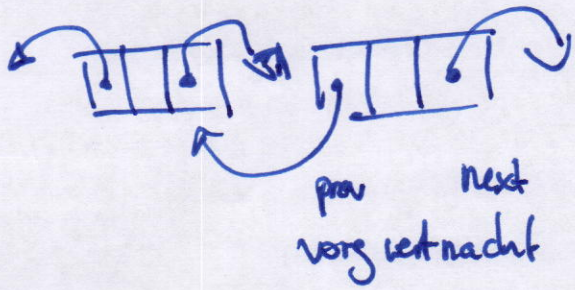

p_{10}
p_{11}
p_{12}
p_9
p_7
p_6
p_8
p_5
p_4
p_3
p_2
p_1
p_0


p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_8
p_5
p_4
p_3
p_2
p_1
p_0


p_{10}
p_{11}
p_9
p_7
p_6
p_{12}
p_5
p_8
p_4
p_3
p_2
p_1
p_0

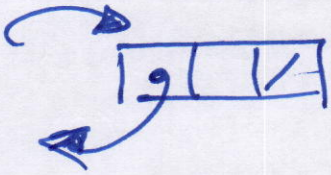
III Verkettete Listen

doppelt verkettet:



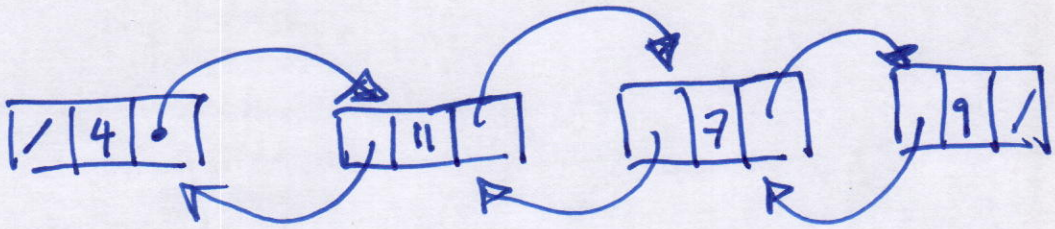
head[L]:

letztes Element:



Beispiel:

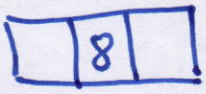
head[L] →



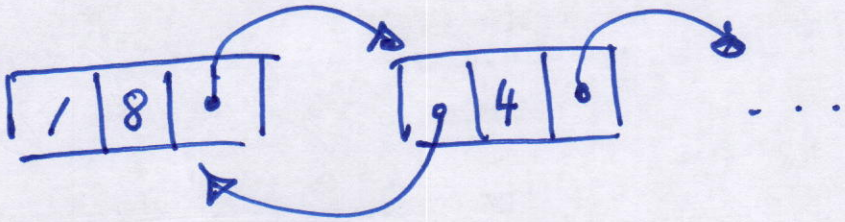
LIST-INSERT(L,8) (8 wird vorne eingefügt)

nachf[x]:= head[L]
vorg[head[L]]:=x
head[L]:=x
vorg[x]:= NIL

x:

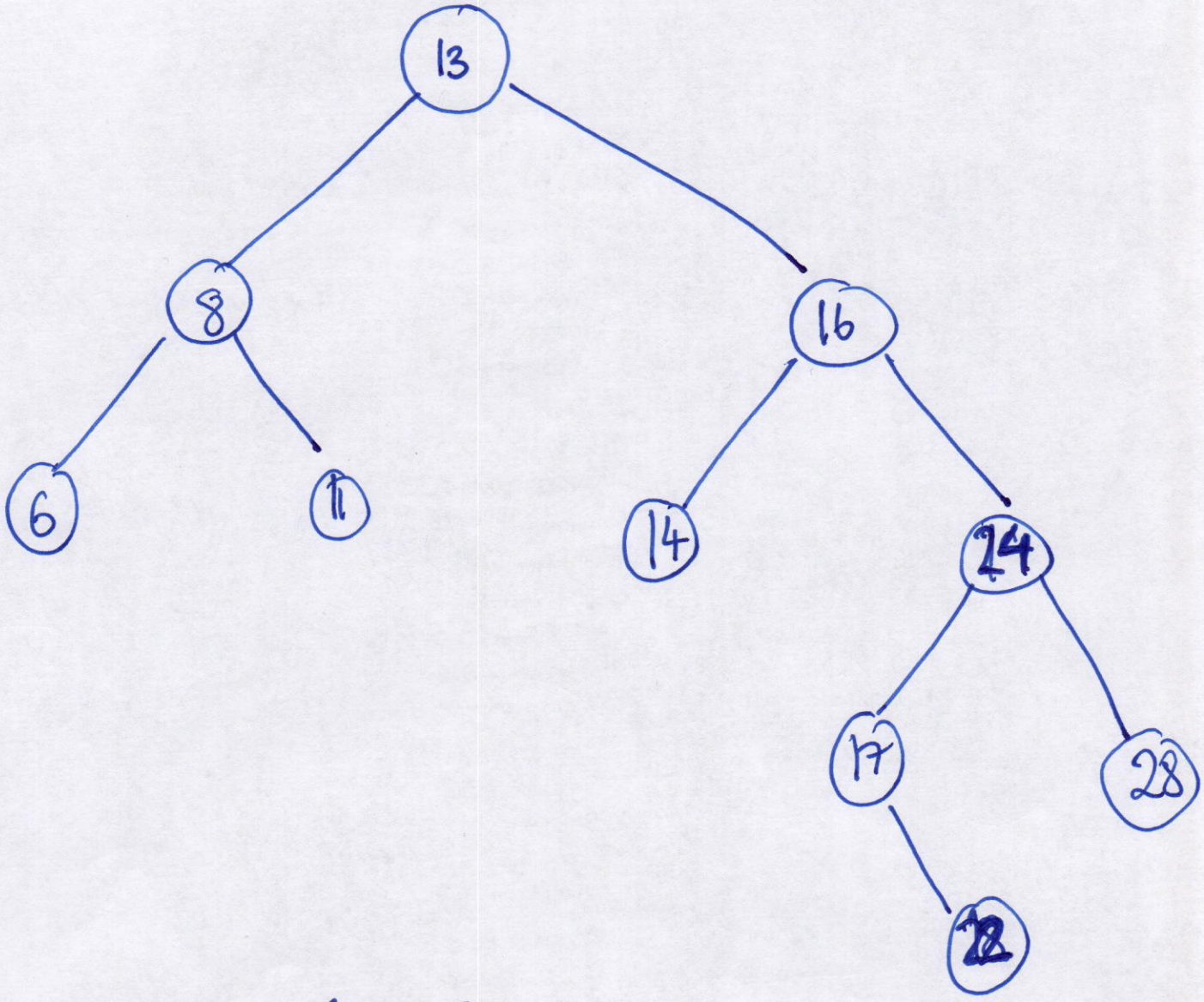


head[L]



Klausuraufgaben: WS08/09

IV. Binäre Suchbäume



Baum-Löschen (T, 22) ✓

Baum-Löschen (T, 16)

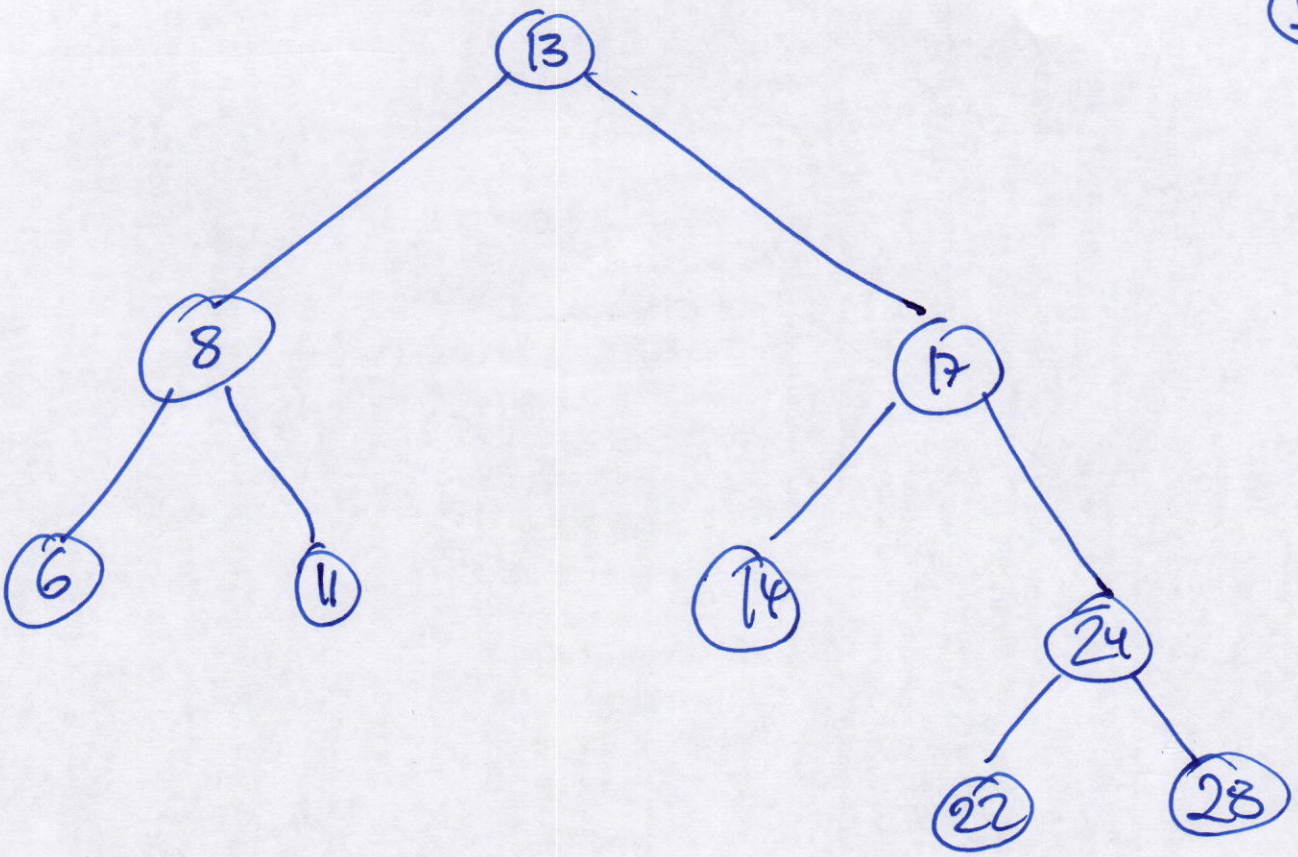
Nachfolger von 16: 17 (Min. im rechten Teilbaum)

17 an die Stelle von 16

22 nicht auf (analog zu Löschen von 17)

↳ ~~Klausuraufgabe~~

~~↳ siehe HA 4~~



↳ Klausuraufgaben

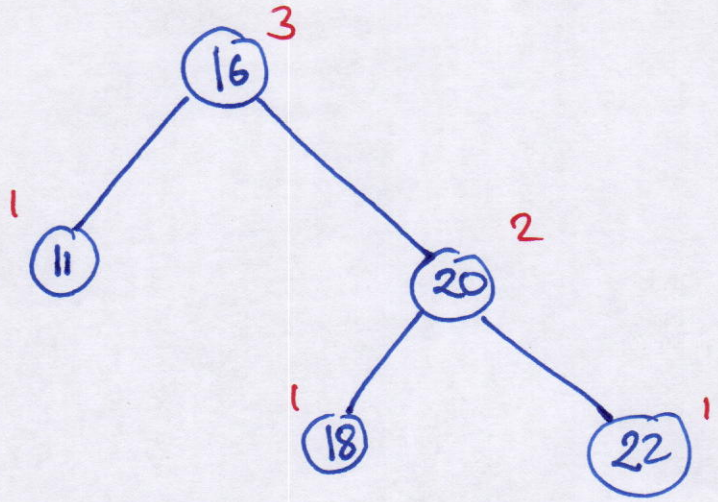
↳ siehe HA 1

(oder SoSe 2010)

AVL-Bäume

↳ Höhe (v) = # Knoten auf längstem Pfad von v zu einem Blatt

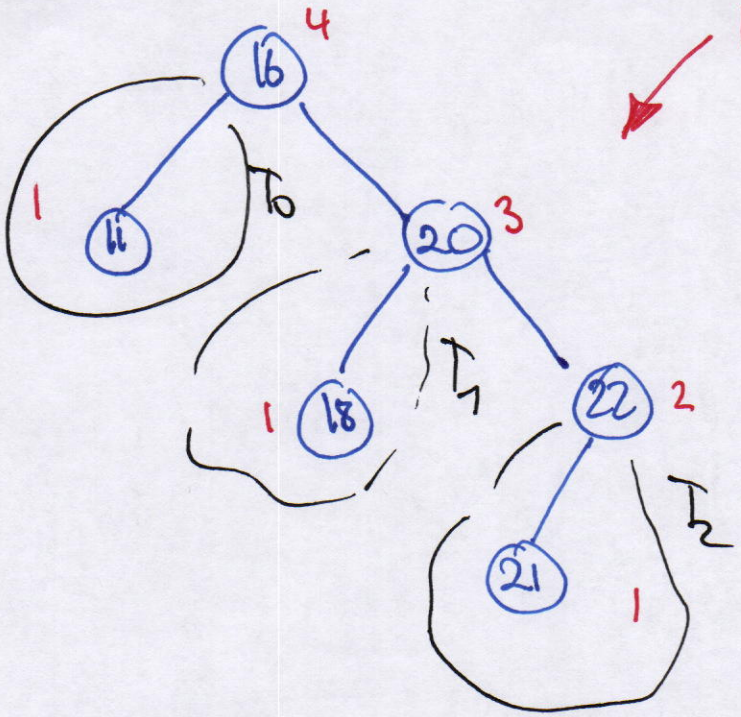
Beispiel:



höhen-balanciert

Ein binärer Suchbaum ist höhenbalanciert, wenn sich für jeden inneren Knoten v, die Höhe der beiden Kinder von v um höchstens 1 unterscheidet.

21 Einfügen:



nicht höhenbalanciert