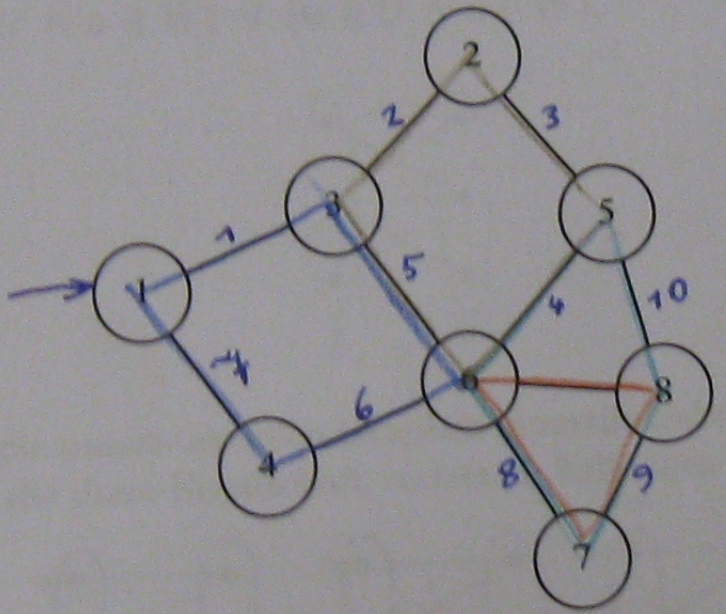
✓ **Aufgabe 27** Führen Sie auf dem untenstehenden Graphen den Tiefensuche Algorithmus zum Finden von Kreisen in einem Graphen aus (Algorithmus 34 Depth-First-Search3, Seite 119). Verwenden Sie den Knoten 1 als Startknoten. Die Nachbarn eines Knotens werden immer in aufsteigend sortierter Reihenfolge betrachtet.

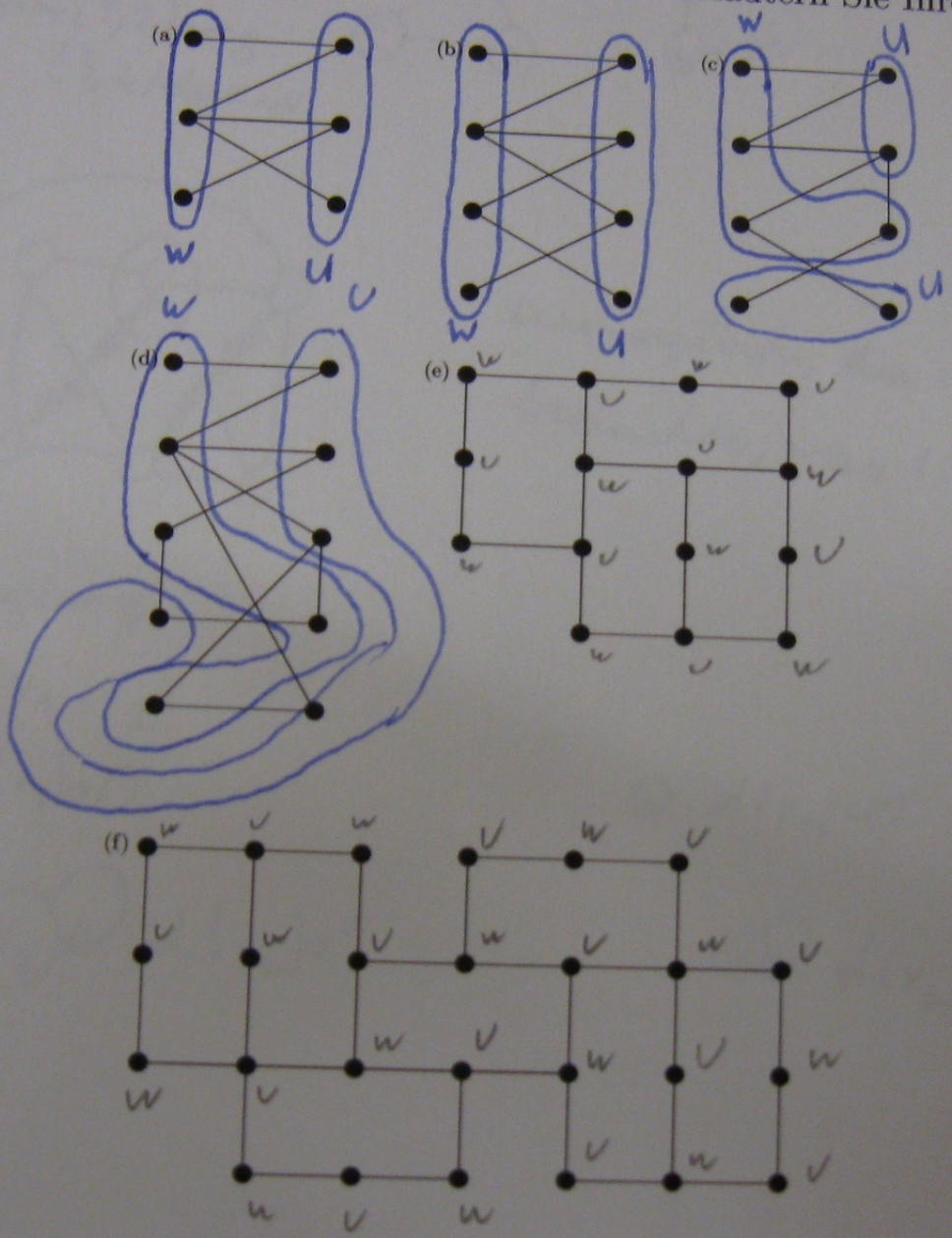
Geben Sie an, in welcher Reihenfolge die Knoten jeweils besucht werden und welche Kreise der Algorithmus findet. Geben Sie einen beliebigen Kreis des Graphen an, der *nicht* vom Algorithmus gefunden wurde.



nicht gefunden:
1-3-2-5-6-4

U W

Untersuchen Sie, ob die untenstehenden Graphen bipartit sind und markieren Sie gegebenenfalls die Kanten, die diese Eigenschaft verletzen. Erläutern Sie Ihre Vorgehensweise!



Aufgabe 30

(a) Gegeben sei der folgende ungerichtete Graph $G = (V, E)$ mit

$$V = \{1, 2, 3, 4, 5, 6\} \text{ und}$$
$$E = \{(1, 2), (1, 3), (2, 3), (4, 5), (5, 6)\}.$$

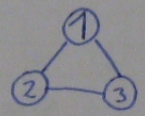
Zeichnen Sie den Graphen. Ist der Graph ein Baum oder ein Wald (Wald: kreisfreier Graph; Baum: zusammenhängender kreisfreier Graph)? Wieviele Kanten müssen Sie mindestens hinzufügen bzw. entfernen um aus dem Graphen einen Baum bzw. einen Wald zu machen?

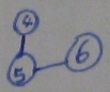
(b) Erstellen Sie einen ungerichteten Graphen mit 7 Knoten, bei dem jeder Knoten den Grad 3 hat, oder beweisen Sie, dass das nicht möglich ist.

(c) Beweisen oder widerlegen Sie: In jedem Graphen mit mindestens 2 Knoten existieren zwei Knoten u, v mit $u \neq v$ und $d(u) = d(v)$.

---

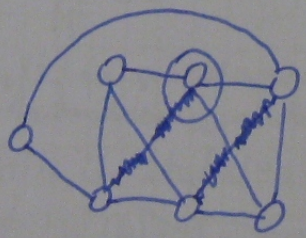
a) für Wald (1,2) weg
für Baum (1,2) weg
(3,4) dazu





weder Wald noch Baum

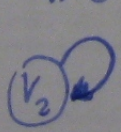
b)



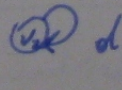
unmöglich, da ich $\frac{3 \cdot 7}{2} = 21/2$ Kanten
bräuchte, was 10,5 wäre

c) Wiederlegung:
1 Knoten v_1, v_2

ungerichtet:  $d(v_1) = 0$

 $d(v_2) = 1$

 $d(v_1) = 1$

 $d(v_2) = 2$

21) a) 7500 Einträge, Bereich $[0, 10000]$

8192: schlecht, weil $8192 = 2^{13}$, was zu vermeiden ist.

7499: weil < 7500

35537: zu groß für nur 7500 Einträge

7507: Wahrscheinlichkeit von Kollisionen ist viel zu groß

$\Rightarrow$ zu oft rehashen

8999: gute Wahl, weil ~20% größer als 7000

b) $h(k,i) = (h_1(k) + i\, h_2(k)) \bmod m$ $\quad m = 8999$

$i \in 0, \ldots, m-1$

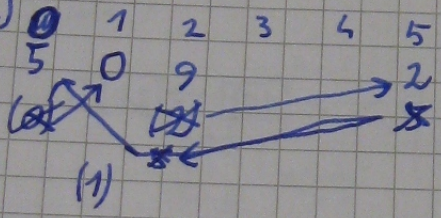
$h_1(k) = k \bmod m$

$h_2(k) = 1 + (k \bmod m')$ $\quad m' = m-1 \vee m-2$

h_2: Für alle Schlüssel k muss $h_2(k)$ relativ prim zu m sein.

$\text{ggT}(h_2(k), m) = 1$

c)



Schlüssel $[0, 10]$

$h_1(k) = k \bmod m$

(9, 2, 5, 0, 1)

$h_2(k) = \begin{cases} 7 - (h_1(k)) & h_1(k) \neq 0 \\ \text{sonst } 1 \end{cases}$

$h_2(k)$ zeigt dann 1 Keys auf 0 zu

platzieren und dann mit Schrittweite 1 weiterzugehen

$h(k,i) = (h_1(k) + i\, h_2(k) \bmod m)$ $\quad i \in 0, \ldots, m-1$

Bsp: $h_1(k) = 0$ schon belegt

$h_2(k) = 6$

$\Rightarrow h(k,1) = ((1+6) \bmod 7) \Rightarrow 0$ auch belegt

$h(k,2) = 6 \Rightarrow$

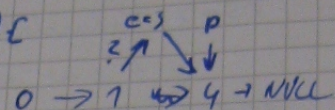
für 7: wäre 0 schon belegt 5 schrittweise um 1 weiter

Lösung: verwendung anderer Funktion:

$h_2(k) = 1 + (k \bmod 8997)$

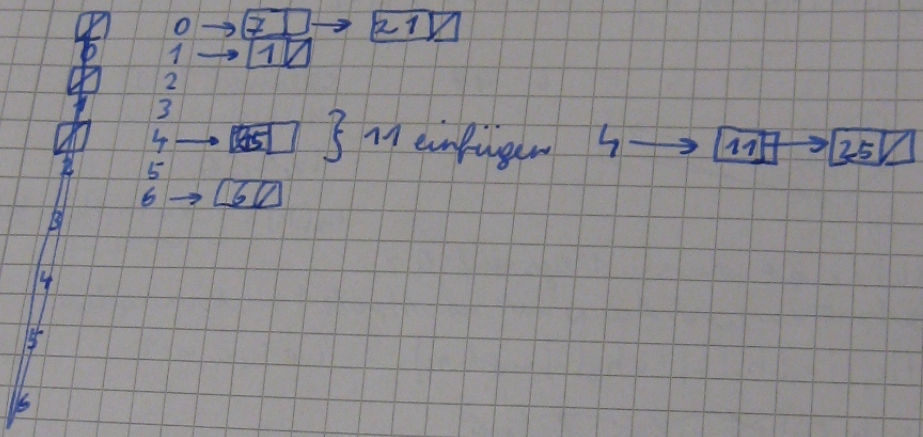
22) a) Einfügen:
Eingabe: Hashtabelle T; Element e
Ausgabe: T mit e eingefügt
Variable: Hashindex pos, Pointer p

```
1: pos = hash(e.key);
   p = T[pos]; p1 = NULL;
2: while(p ≠ NULL ∧ e.key ≥ p.key){
4:   p1 = p++;
5: }
6: e.next = p;
   if(p1 ≠ NULL){
7:   p1.next = e;
   }else{
     T[pos] = e;
   }
```



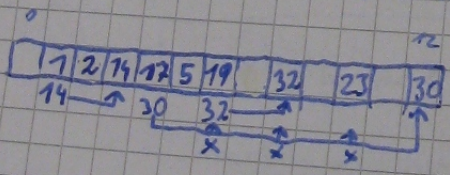
e.next = p

b) ⟨1, 6, 7, 25, 21, 11⟩ $h(k) = k \bmod 7$



23) ⟨5, 1, 19, 23, 14, 17, 32, 30, 2⟩ $m = 13$ $h(k) = k \bmod 13$

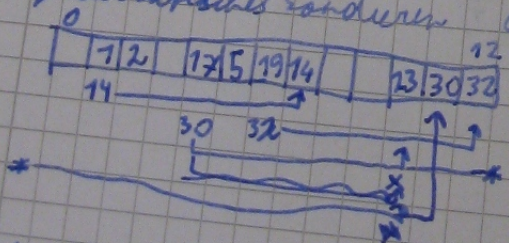
a) lineares Sondieren: Schrittweite 2



$h(k,i) = (h'(k) + i) \bmod m$

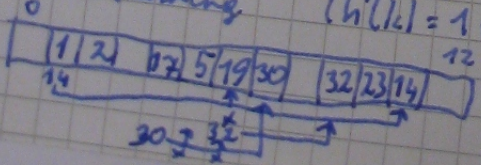
6 zusätzliche Schritte

b) quadratisches Sondieren $c_1 = 2, c_2 = 4$ $h(k,i) = [h(k) + c_1 i + c_2 i^2] \bmod 13$



4 zusätzliche Schritte

c) double Hashing $(h'(k) = 1 + (k \bmod 5))$ $h(k,i) = (h_0(k) + i(h'(k)) \bmod 13$



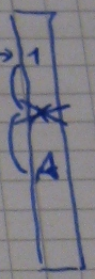
6 zusätzliche Schritte

24) a) Eingabe: feld; Elementanzahl m
Ausgabe: initialisierte Hashtabelle
Variablen: Index j;

1: for(j = 0, ..., m-1){
2: feld[j].zustand = frei;
3: }

b) Eingabe: feld; Elementanzahl m; gesucht gesuchtes Element gesucht
Ausgabe: void()
Variablen: Index i = h[gesucht]

1: while(feld[i].zustand ≠ frei ∧ feld[i].key ≠ gesucht){
2: i += c;
3: }
4: if(feld[i].zustand ≠ frei){
6: feld[i].zustand = wiederfrei;
7: }



25)

$m = 7$

$h_1(k) = 1 \cdot k \bmod 7$

$h_2(k) = 3 \cdot k \bmod 5 + 2$

$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$

0	1	2	3	4	5	6
8	1		3			13

8 ← 17

Einfügen ⟨8, 17⟩
dann: 13 löschen, 15 einfügen

8: $h(k,1) = j_1$
1: $h(k,1) = j_2$

$j_1 = (1+6) \bmod 7 \Rightarrow 0$

37: $j_1 = (3+3) \bmod 7 = 6$

3. $j_2 = (3+6) \bmod 7 = 2$

0	1	2	3	4	5	6
8	1	3	17		15	~~13~~

15

$15 = j_1 = (1+11) \bmod 7 = 5$

$1 = j_2$

26) linear

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Schlüssel	~~13~~	27	29				6	20	21	22	10	37	~~12~~
Status	W	B	W	F	F	R	W	B	B	B	B	B	B

9 → 34

9 → 12 wird gelöscht ⇒ wieder frei

34 wird auf Index 12 geschrieben