Matrix Multiplikation

Aufwand

↳ Standard-Algorithmus: $O(n^3)$

↳ Naives DaC: $T(n) = 8 \cdot T(\frac{n}{2}) + \underbrace{4 \cdot (\frac{n}{2})^2}_{\Theta(n^2)}$

gleicher Aufwand

Master Theorem:

$a = 8 = 2^3, b = 2, k = 2$

$\Rightarrow 8 > 2^4 = 4$ (1. Fall)

$T(n) = O(n^{\log_2(8)}) = O(n^3)$

↳ Trick von Strassen:

$T(n) = 7 \cdot T(\frac{n}{2}) + \underbrace{18 \cdot \frac{n^2}{4}}_{\Theta(n^2)}$

Untere Schranke: $\Omega(n^2)$

Master Theorem

$a = 7, b = 2, k = 2$

$7 > 2^2 = 4$ (1. Fall)

$T(n) = O(n^{\log_2(7)}) = O(n^{2.8})$

Maximale Teilsuche

↳ Brute Force:

Aufwand entspricht Anzahl, wie oft Zeile 7 ausgeführt wird

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} 1$$

äußere Schleife · mittlere Schleife · innere Schleife · Aufwand für Zeile 7

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$= \frac{n^3+3n^2+2n}{6}$$

$$= O(n^3)$$

D&C:

$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

Master Theorem

$a = 2, b = 2, k = 1$

$2 = 2^1$ (2. Fall)

$T(n) = O(n \cdot \log_2(n))$ → deutlicher Gewinn

Huffman-Codierung

Für das Decodieren wird die Präfix-Eigenschaft

↳ Bsp:

a	01
b	010

Nachricht

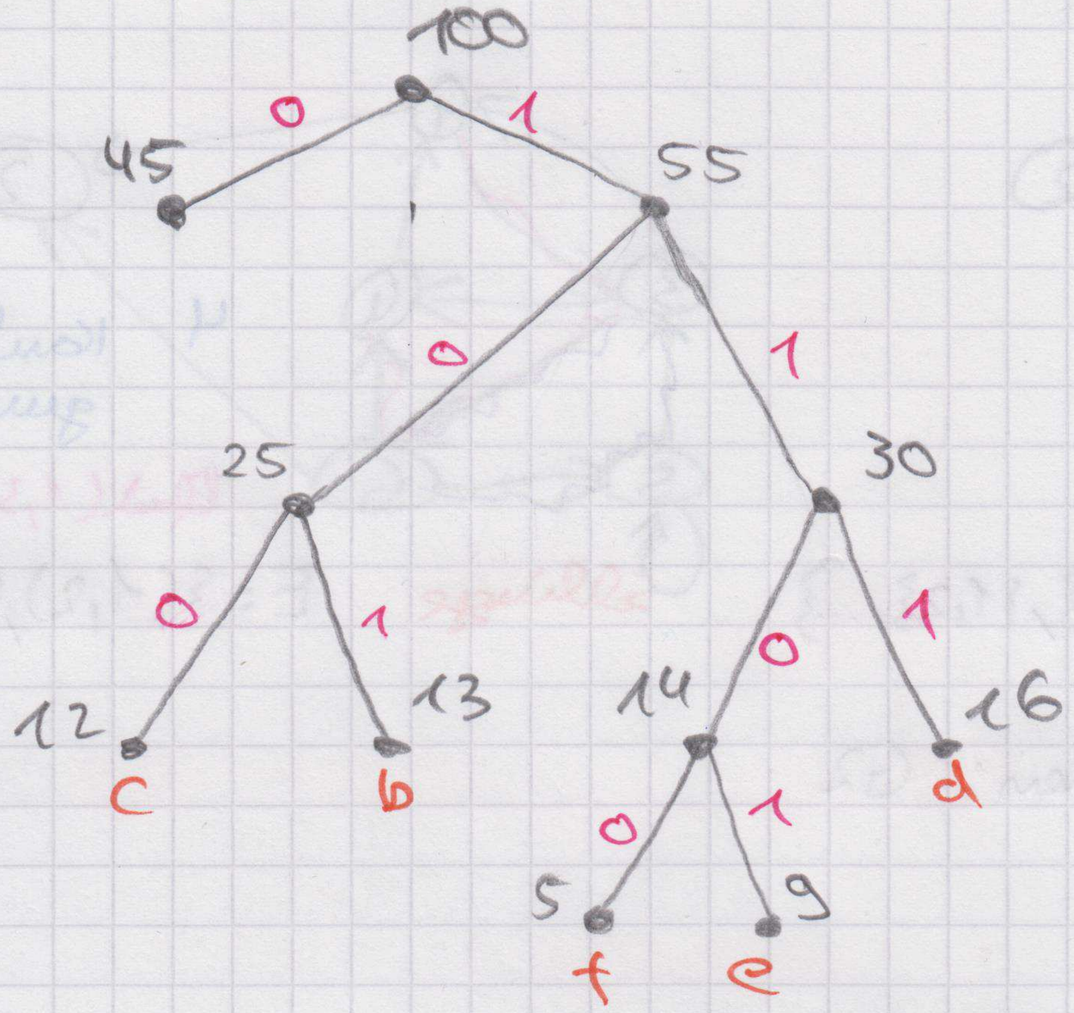
aba : 01001001

a b a

Mehrere Möglichkeiten beim Codieren

Bsp. Code-Baum

für Alphabet aus Skript.



z.B. Code Wörter:

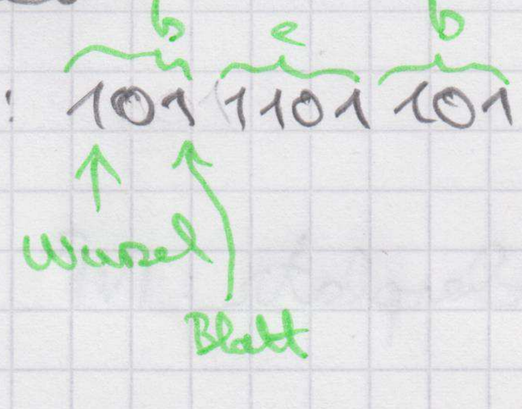
b : 101

e : 1101

Präfix-Eigenschaft war erfüllt, da die inneren Knoten nicht als Code Wörter verwendet werden.

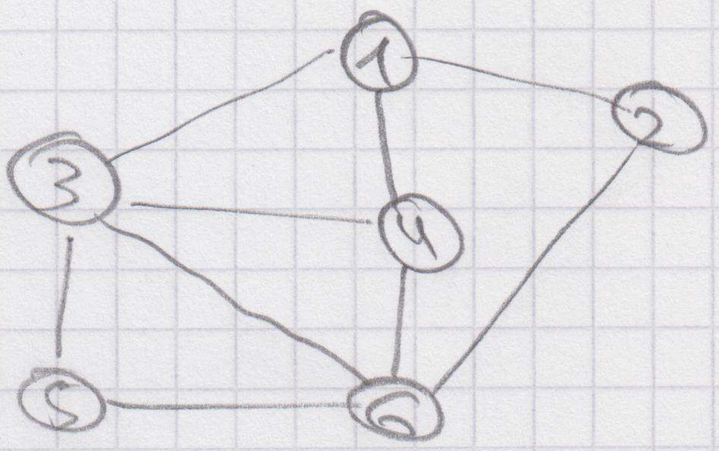
Decodierung auch über Baum:

Nachricht



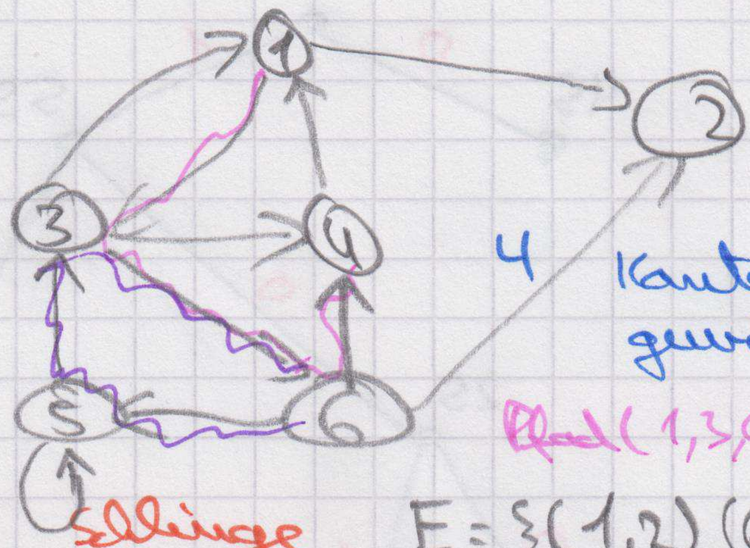
Graphen

a) ungerichtet



$V = \{1, \dots 6\}$, $E = \{\{1,3\}, \{1,2\}, \dots\}$

b) gerichtet / gewichtet



Zyklus (6, 5, 3, 6)

Pfad (1, 3, 6, 4) hat Länge 3

Schlinge

$E = \{(1,2), (6,4), (4,6), \dots\}$

Speicherung von Graphen: G2

a) Kantenliste

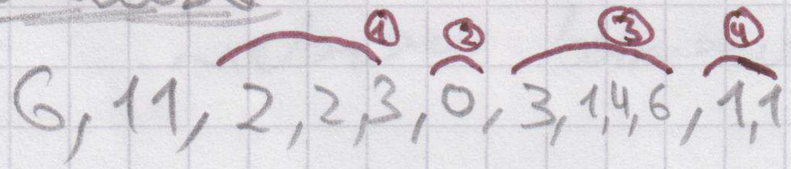
6, 11, $\overline{1,2}$, $\overline{1,3}$, $\overline{3,1}$, $\overline{3,4}$, $\overline{3,6}$ ...

|V| Anz. Knoten

|E| Anz. Kanten

↳ Speicherplatz: $2 + 2 \cdot |E|$

b) Knotenliste



↳ Speicherplatz: $2 + |E| + |V|$

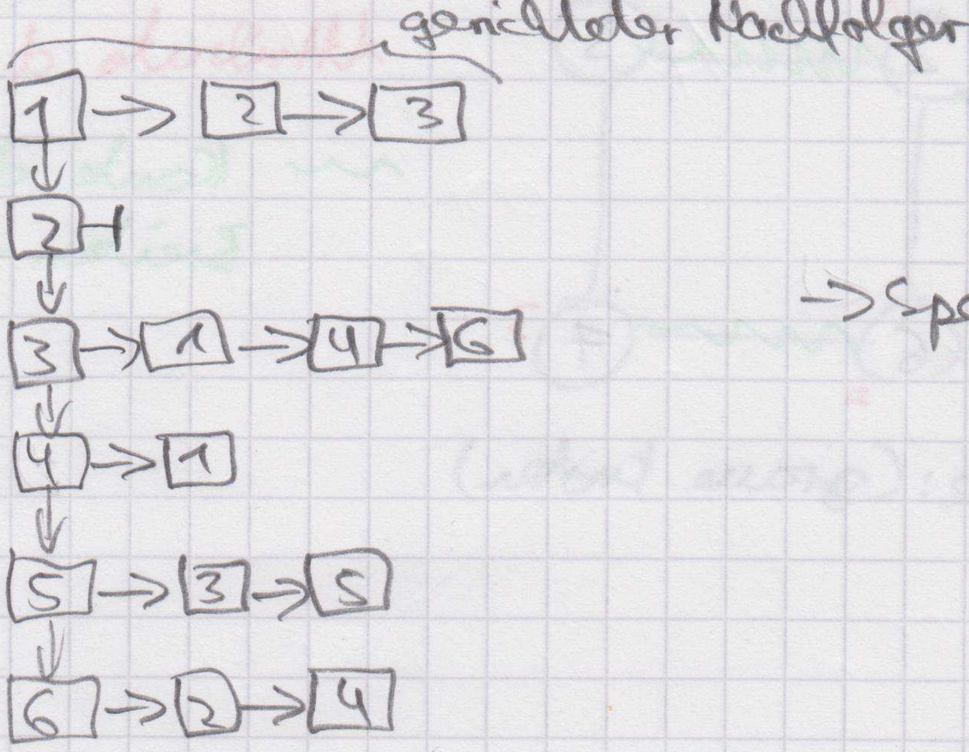
c) Adjazenzmatrix

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	1	0	0	1	0	1
4						
5						
6						

→ Speicherplatz: $|V|^2$

Vorteil: liefert direkt Ergebnis, ob Kante zwischen i nach j verläuft

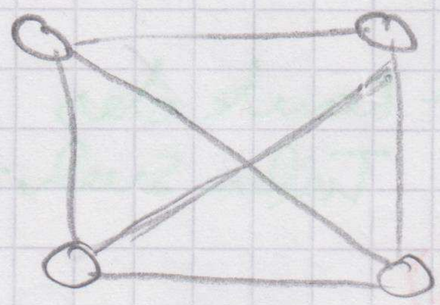
d) Adjazenz-Liste



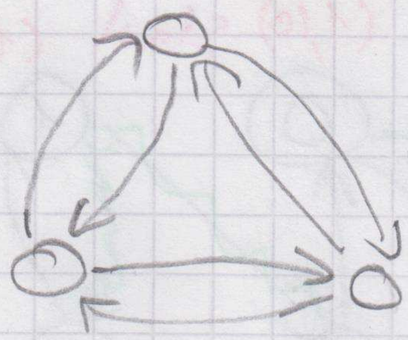
→ Speicherplatz: $O(|V| + |E|)$

schlichter Graph: vollständig

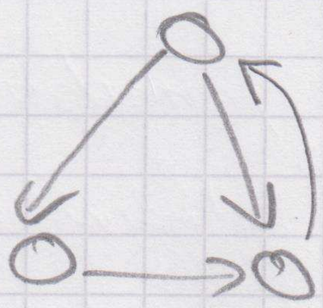
ungerichtet



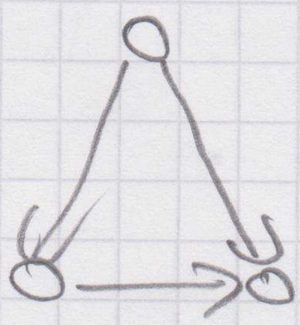
gerichtet



stark / schwach zusammenhängend



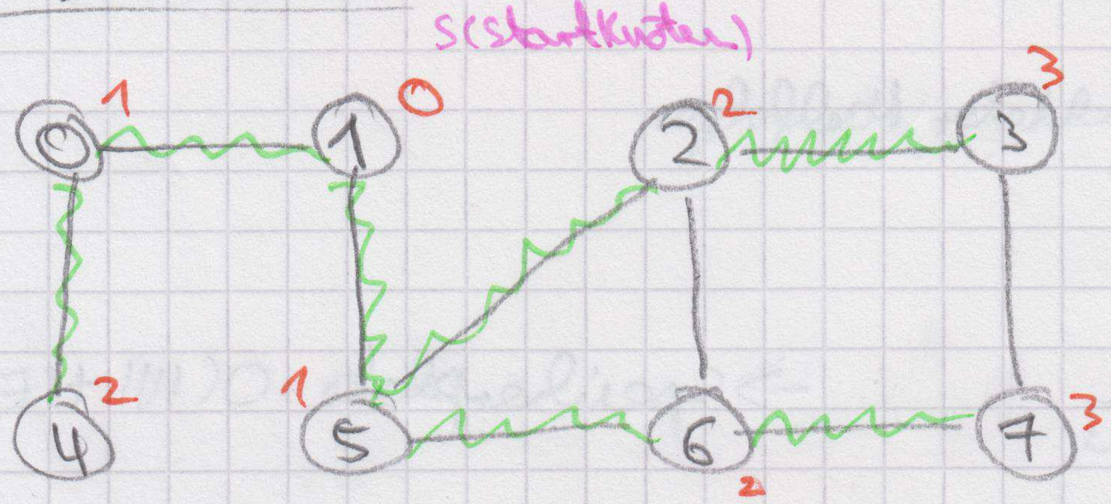
stark (von jedem Punkt zu jedem)



zugehöriger ungerichteter Graph

zusammenhängend!

Bsp: Breitensuche



Attribute d

~ Kante des Breiten-Such-Baumes

→ Inhalt der Schlange: (graue Knoten)

d = 0: ~~1~~

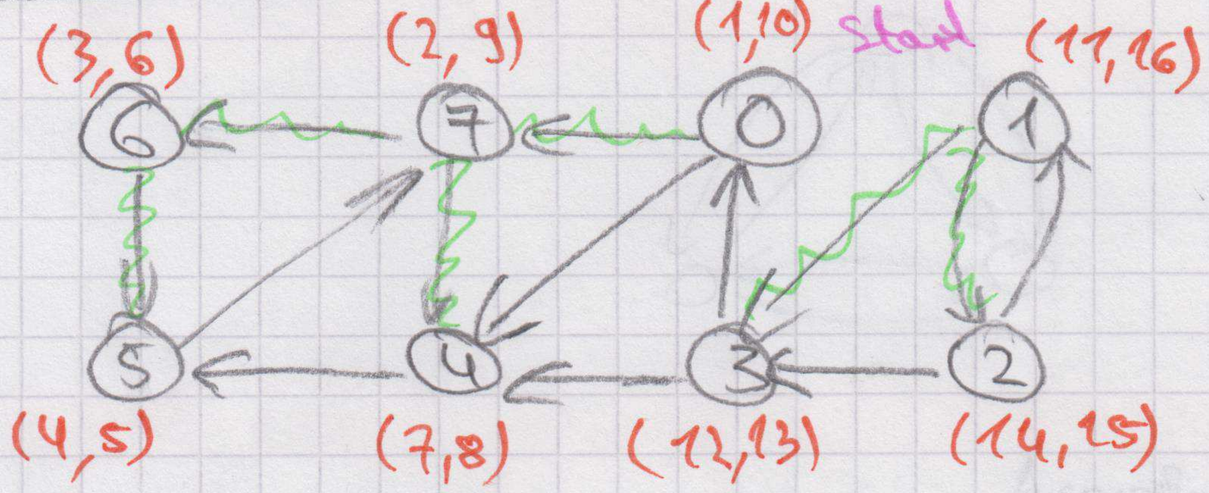
d = 1: ~~0~~ ~~5~~

d = 2: ~~4~~ ~~2~~ ~~6~~

d = 3: ~~3~~ ~~7~~

d = 4:

Bsp: Tiefensuche



~ Kante des Tiefen Such-Baumes

(d,f)

d: Entdeckungszeit

f: Endzeit