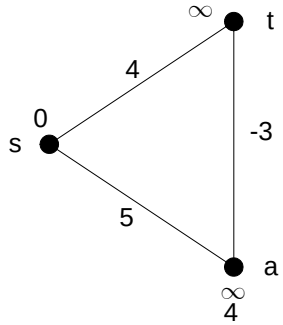
ALP III **Stefan Lenz**
Übung 11 **Roussi Alexiev**

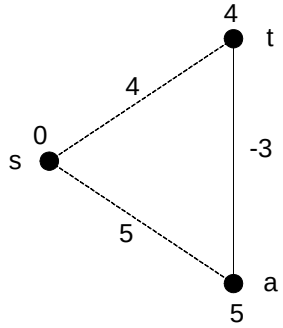
Aufgabe 64

Konstruieren Sie einen Graphen, der auch Kanten mit negativer Länge enthält, und bei dem der Algorithmus von Dijkstra die kürzesten Wege nicht richtig bestimmt. Der Graph soll keine Kreise negativer Länge enthalten.

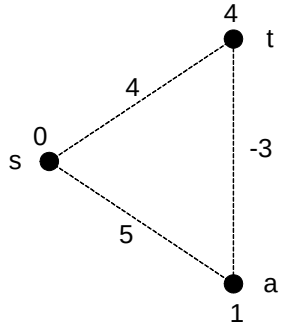
Gegeben Graph G;
s ist der Startknoten;
setze s als Bezugsknoten;
setze dis[s]=0;
setze dis[Rest]= ∞ ;



markiere (s, t);
setze dis[t]=4;
markiere (s, a);
setze dis[a]=5;
setze t als Bezugsknoten, da dis[t]< dis[a];



markiere (t, a);
setze dis[t]=1;
Algorithmus wird beendet;



Wie man leicht sieht, hat der Algorithmus die Distanz zwischen s und t fälschlicherweise mit 4 statt mit 2 berechnet.

Aufgabe 68

Berechnen Sie die Verschiebefunktion der Morse-Folge
01101001100101101001011001101001100101100110100101101001100 10110
Verschiebefunktion:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
h[i]	**0**	**1**	**1**	**1**	**2**	**3**	**2**	**2**	**3**	**4**	**5**	**2**	**3**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**2**

i	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
h[i]	**3**	**4**	**5**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**2**

i	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
h[i]	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**

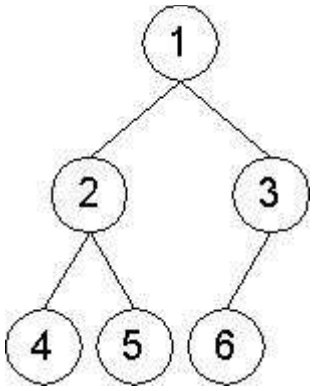
i	63	64
h[i]	**15**	**16**

D-Halden

Es sollen sog. „d-Halden“ (eine Variante bei der jeder innere Knoten bis zu d>=2 Kinder hat) betrachtet werden.

Eine „d-Halde“ ist ein fast vollständiger (*Blätter alle auf dem selben Level und von links nach rechts aufgefüllt*) d-närer Baum deren Knoten die Haldeneigenschaft erfüllen :
Der Wert eines Knotens ist immer geringer als der Wert seiner Kinder.

Halden können effektiv in Arrays gespeichert werden, da sich Kinder- und Elternknoten leicht aus dem Index eines Knotens errechnen lassen.
(Im folgenden sind die Knoten jeweils mit ihrem Index bezeichnet.)

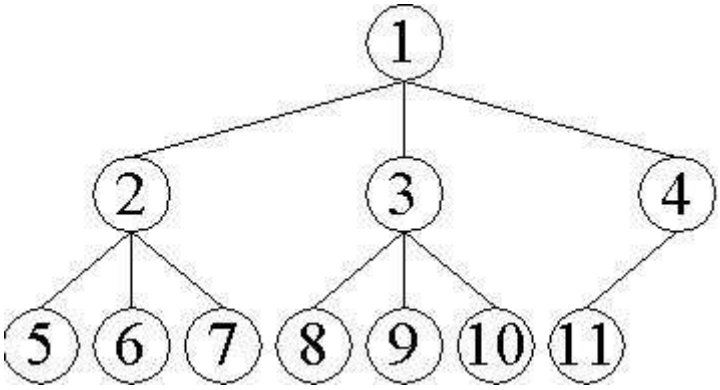


Für d=2 erhält man die üblichen aus der Vorlesung bekannten Halden :
Das erste Kind eines Knotens mit Index i erhält man (sofern vorhanden) mit der Formel :
child(i)[1] = $2*i$ = $2*(i-1)+2$

Das letzte (also zweite) Kind entsprechend mit :
child(i)[2]= $2*i+1$ = $2*(i-1)+3$

Den Elternknoten bestimmt man für alle Knoten mit Index i>1 (i=1--> Wurzel) durch :
elter (i)= $\lfloor \frac{i}{2} \rfloor$

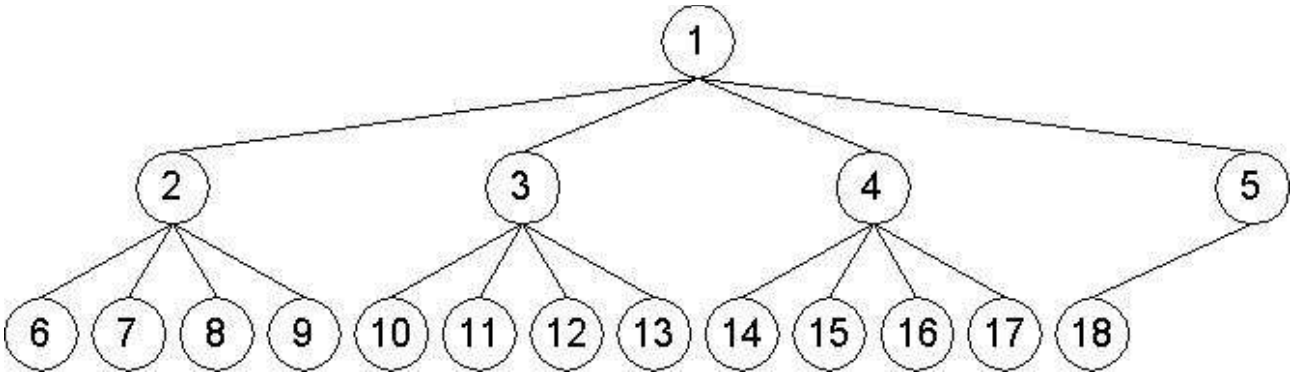
Für d=3 erhält man eine Halde der folgenden Struktur :



child(i)[1]= $3*i-1 \;=\; 3*(i-1)+2$
child(i)[3]= $3*i-1+2 \;=\; 3*(i-1)+4$

elter (i)= $\lfloor \frac{i+1}{3} \rfloor$

Und bei d=4 ?



child(i)[1]= $4*i-2 \;=\; 4*(i-1)+2$
child(i)[4]= $4*i-2+3 \;=\; 4*(i-1)+5$

elter (i)= $\lfloor \frac{i+2}{4} \rfloor$

Der allgemeine Fall :

child(i)[1] = $d*(i-1)+2$
child(i)[d] = $d*(i-1)+2+(d-1)$

elter (i)= $\lfloor \frac{i+d-2}{d} \rfloor$

Was bedeutet das für die Laufzeit der Funktionen zugross und zuklein ?

zuklein wird aufgerufen, wenn der Wert eines Knotens potentiell kleiner ist als der seines Elternknotens.
Im schlimmsten Fall fängt man bei einem Blatt an und hört bei der Wurzel auf.
Also ist die Laufzeit = $O(\text{Höhe der Halde}) \;=\; O(\log_d(n))$

zugross wird aufgerufen , um sicherzustellen dass ein Knoten nicht grösser ist als seine Kinder.
Im schlimmsten Fall fängt man bei der Wurzel an und hört erst bei einem Blatt auf.
Also ergibt sich für die Anzahl der Rekursionen : $O(\text{Höhe der Halde}) \;=\; O(\log_d(n))$

Bei jedem Aufruf muss das kleinste von bis zu d Kindern festgestellt werden
Da die Geschwisterknoten in einer Halde nicht sortiert sind, bleibt nur die sequentielle Suche und die frisst $O(d)$ Zeit.
Insgesamt ergibt sich somit für zugross eine laufzeit von $O(d*\log_d(n))$

Wie wirkt sich das auf einfügen, minEntferne und verkleinereSchlüssel aus ?

- einfügen :
  Hier wird das neue Element als letztes Blatt hinzugefügt und anschliessend zuklein aufgefrufen ==> $O(\log_d(n))$

- minEntferne :
  Die Wurzel wird entfernt, durch das letzte Blatt ersetzt und anschliessend zugross auf der neuen Wurzel aufgerufen ==> $O(d * \log_d(n))$

- verkleinereSchlüssel :
  Der Schlüssel eines beliebigen (referenzierbaren) Knotens wird verkleinert und anschliessend zuklein aufgerufen ==> $O(\log_d(n))$

Konsequenzen für den Dijkstra-Algorithmus

Betrachte den Dijkstra-Algorithmus unter Verwendung einer auf d-Halden basierenden PW-Schlange :
V = Menge der Knoten $|V| = n$
E = Menge der Kanten $|E| = m$

Die PW-Schlange wird zur Verwaltung der Menge G der gesehenen Knoten verwendet.

- Jeder Knoten wird einmal in die Menge G übernommen
  ==> n-mal einfügen
- n-mal muss der kleinste Schüssel aus G bestimmt und entfernt werden
  ==> n-mal minEntferne
- maximal m-mal (nämlich für jede Kante einmal) kann es passieren, dass ein kürzerer Weg zu einem bereits gesehenen Knoten gefunden wird
  ==> max. m-mal verkleinereSchlüssel

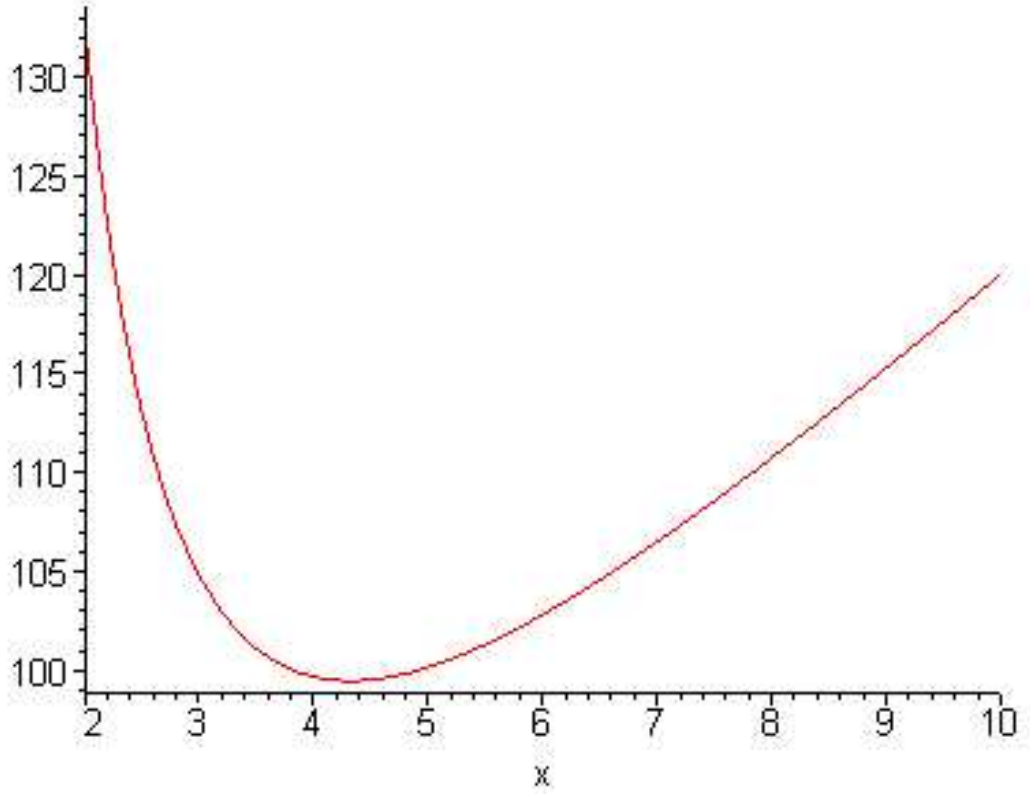
Also ergibt sich für den Dijkstra-Algorithmus eine asymptotische Laufzeit von
$O((m+n) * \log_d(n) + n * d * \log_d(n))$

Bestimmung der optimalen asymptotischen Laufzeit
Die erste Idee war schnell mal $f(d) = (m+n) * \log_d(n) + n * d * \log_d(n)$ nach d abzuleiten, die Nullstellen zu suchen und ein Minimum anzugeben.
(Diese Idee werde ich aber erstmal bis Mafi 6 zurückstellen ...)

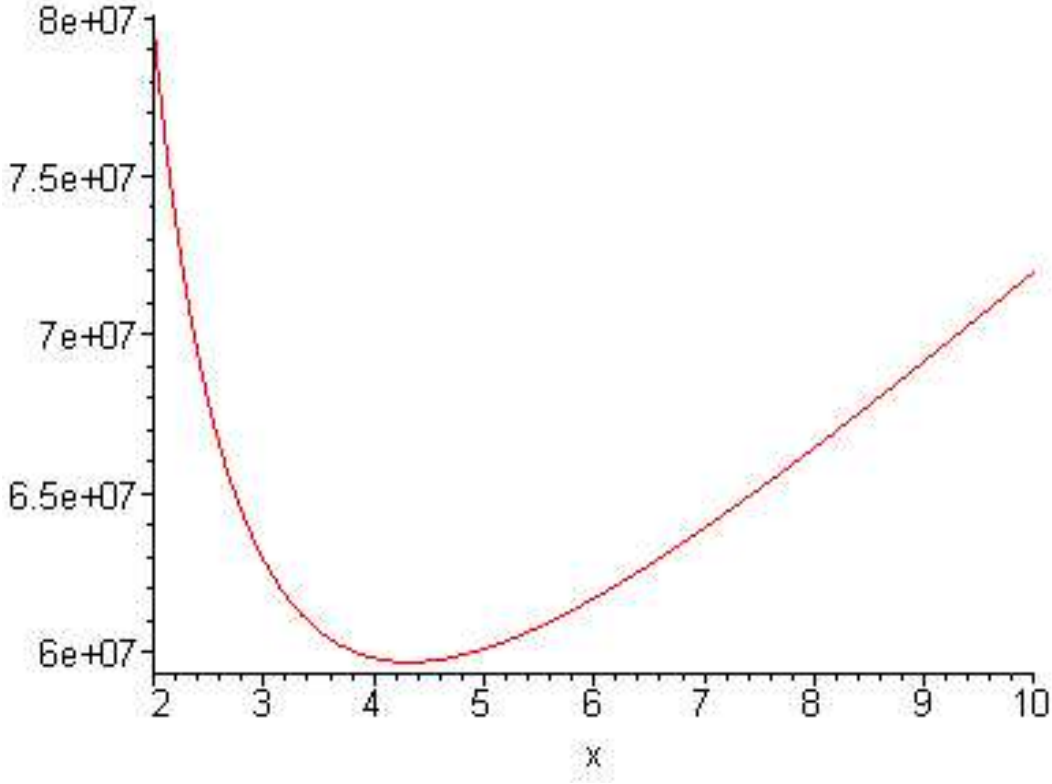
Alternativ kann man sich ja mal die Funktion für verschiedene n und m ausplotten lassen. Interessant sind hier natürlich v.a. Die Extremfälle $m = \Theta(n)$ bzw $m = \Theta(n^2)$

$m = \Theta(n)$

n=10



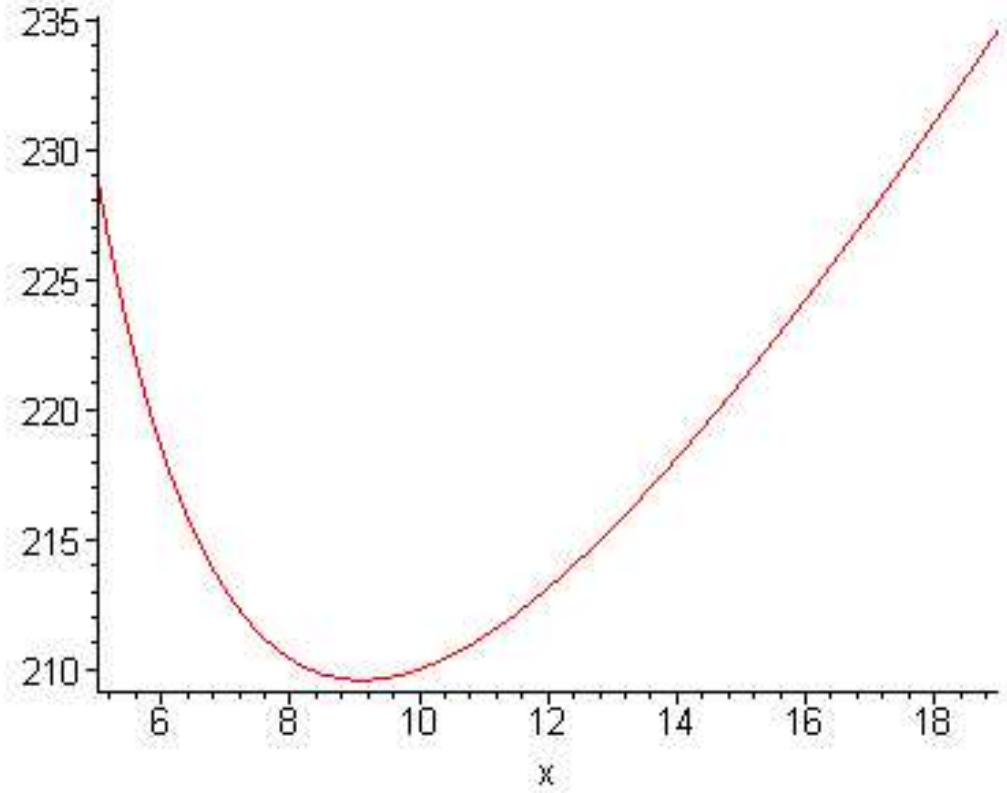
n=1.000.000



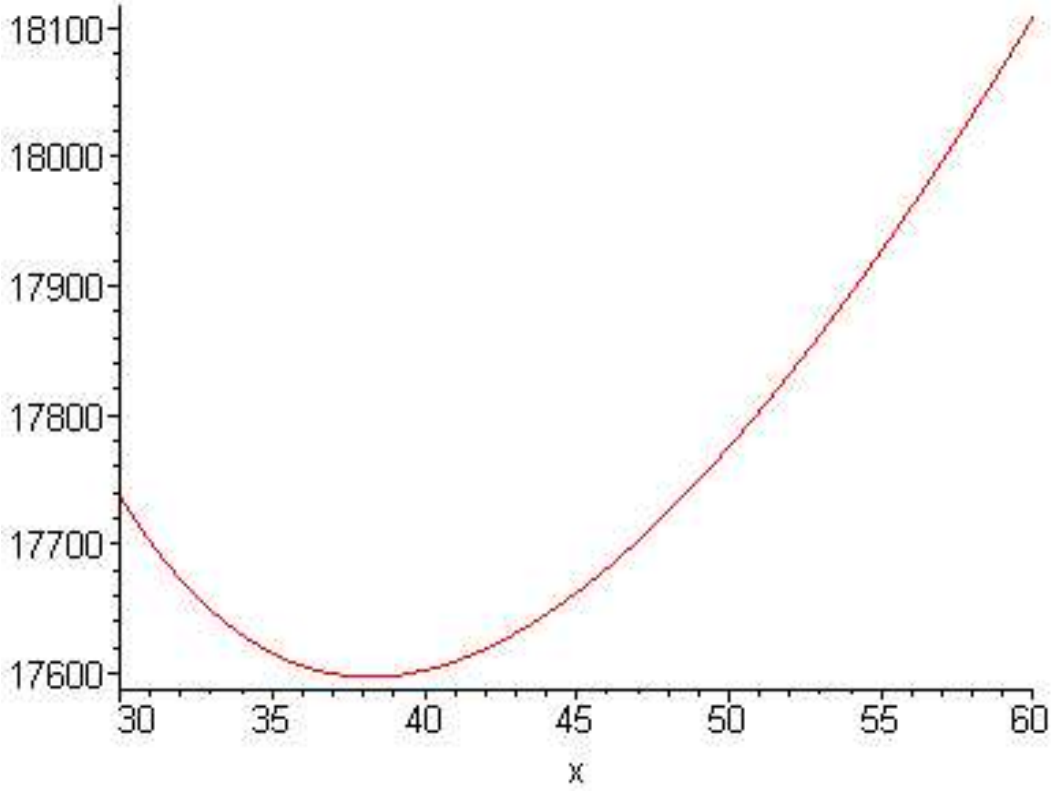
Interessanterweise ist die Grösse von n in diesem Fall scheinbar unrelevant .
Das Minimum liegt immer zwischen 4 und 5, daher sollte man d=4 wählen, wenn man weiss, dass der zu untersuchende Graph eher wenig Kanten enthält.

$m = \Theta(n^2)$

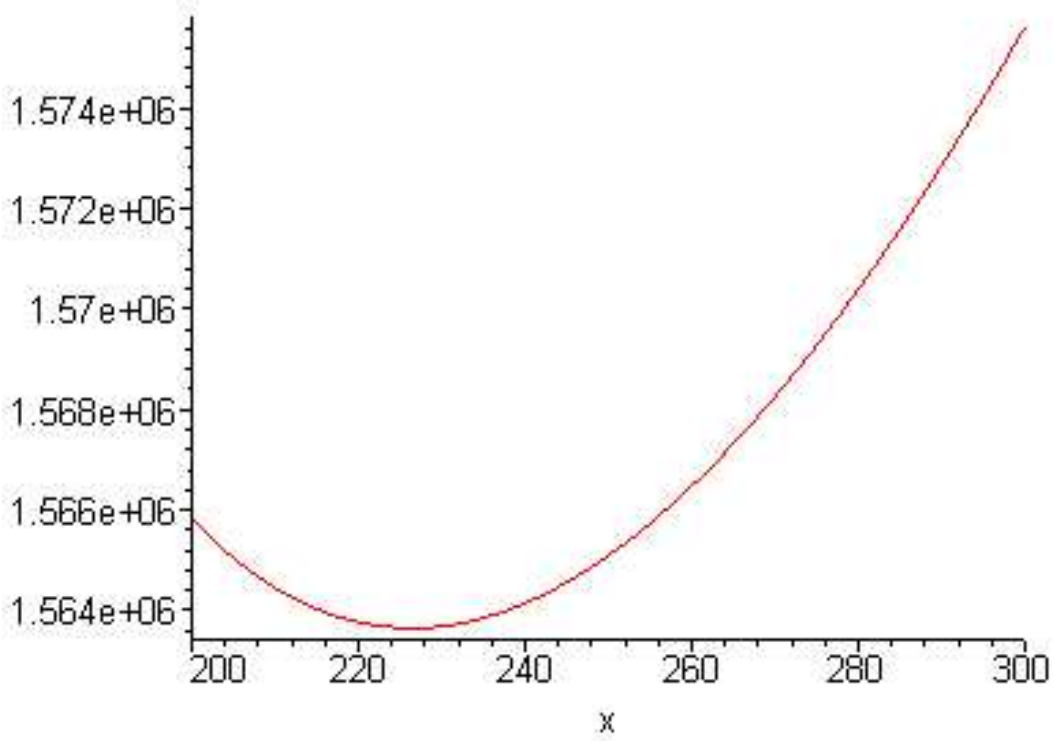
n=10



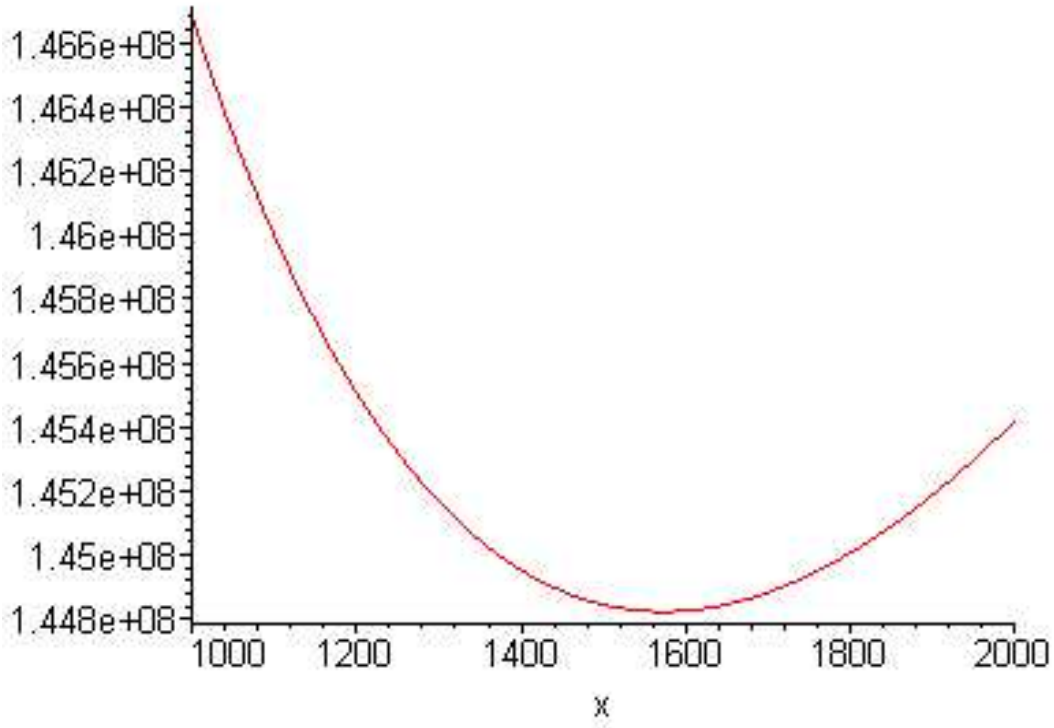
n=100



n=1000



n=10.000



Hierbei fällt auf, dass der optimale Grad d der Halde mit steigender Knotenzahl n, und dazu implizit exponential steigender Kantenzahl m immer grösser wird.
Wenn vollständiger der zu untersuchende Graph ist, desto mehr lohnt sich daher ein genauerer Blick auf den verwendeten Haldengrad.