Datenbankanwendung

Wintersemester 2014/15

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Hash Join

ID	name
10	Jim
13	Joe
14	Sue
15	Pete
21	Dave
23	Anne

Angestellte

$\bowtie$

number	ID
100	23
110	10
120	15
130	23
140	23
150	13
160	15
170	21

Telefon

Wende Hashfunktion an auf Join-Attribut(e)
$\rightarrow$ Partitioniert Tupel in Buckets

Hash Join

ID	name
15	Pete
21	Dave

Angestellte$_0$

$\bowtie$

number	ID
120	15
160	15
170	21

Telefon$_0$

$=$

ID	name	number
15	Pete	120
15	Pete	160
21	Dave	170

Ergebnis$_0$

ID	name
10	Jim
13	Joe

Angestellte$_1$

$\bowtie$

number	ID
110	10
150	13

Telefon$_1$

$=$

ID	name	number
10	Jim	110
13	Joe	150

Ergebnis$_1$

ID	name
14	Sue
23	Anne

Angestellte$_2$

$\bowtie$

number	ID
100	23
130	23
140	23

Telefon$_2$

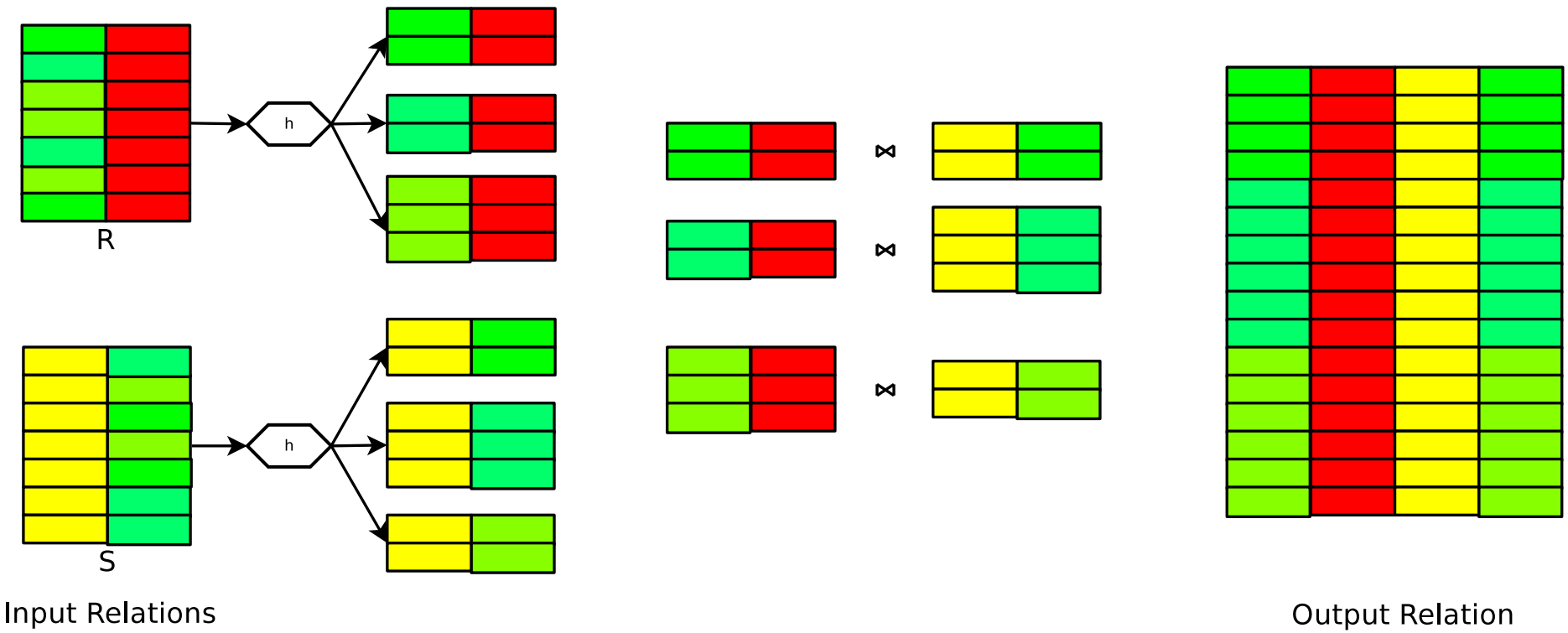
$=$

ID	name	number
23	Anne	100
23	Anne	130
23	Anne	140

Ergebnis$_2$

Hash Join

- Hashing jeder Relation basierend auf Join Attribut(en)
- Jeder Bucket muss klein genug für Hauptspeicher sein
- Tupel in korrespondierenden Buckets der beiden Relationen werden dann verbunden



Input Relations

Output Relation

Hash Join: Pseudocode

Aka. Grace Hash Join (in Literatur)

Partitioniere R in k Partitionen R_i

for each tuple $r \in R$ **do**
    lese r und füge es Puffer-Seite (Block) $h(r)$ hinzu

Partitioniere S in k Partitionen S_i

for each tuple $s \in S$ **do**
    lese s und füge es Puffer-Seite (Block) $h(s)$ hinzu

Hash Join: Pseudocode (2)

Sondierung (engl. Probing) Phase

for $l = 1, \ldots, k$ **do** $\{$

//Erzeuge im Hauptspeicher Hash-Tabelle für R_l mit Hash-Funktion h_2
for each tuple $r \in$ partition R_l **do**
lese r und füge es in Hash-Tabelle anhand $h_2(r)$ ein

//Lese S_l und teste nach passenden Tupeln aus R_l
for each tuple $s \in$ partition S_l **do** $\{$
lese s und teste Hash-Tabelle unter Verwendung von $h_2(s)$
für passende Tupel $r \in R$: $Res := Res \cup (r \circ s)$ $\}$
leere Hash-Tabelle und betrachte nächste Partition.
$\}$

Kosten und Anwendbarkeit der verschiedenen Join-Strategien

Nested Loop Join

- Kann für alle Arten von Joins benutzt werden
- Kann aber sehr teuer sein

Merge Join

- Eingaben müssen bereits sortiert vorliegen
- Oder für den Join extra sortiert werden
- Kann ggf. Indexe ausnutzen

Hash join

- Gute Hashfunktionen sind elementar
- Performanz am besten, wenn kleinere Relation direkt in Hauptspeicher passt

Was ist mit Joins über mehrere Attribute? Was mit Joins, die nicht auf Gleichheit (=) testen? Welche Implementierungen sind anwendbar?

Kostenschätzung

(c) xkcd.com

Selektivitätsschätzung für kostenbasierte Optimierung

Selektivitätsschätzung: Idee

- Gegeben: Anfrage und Relationen
- Wie viele Tupel sind als Ergebnis zu ewarten?
- Wie viele Tupel fallen als Zwischenergebnisse an?

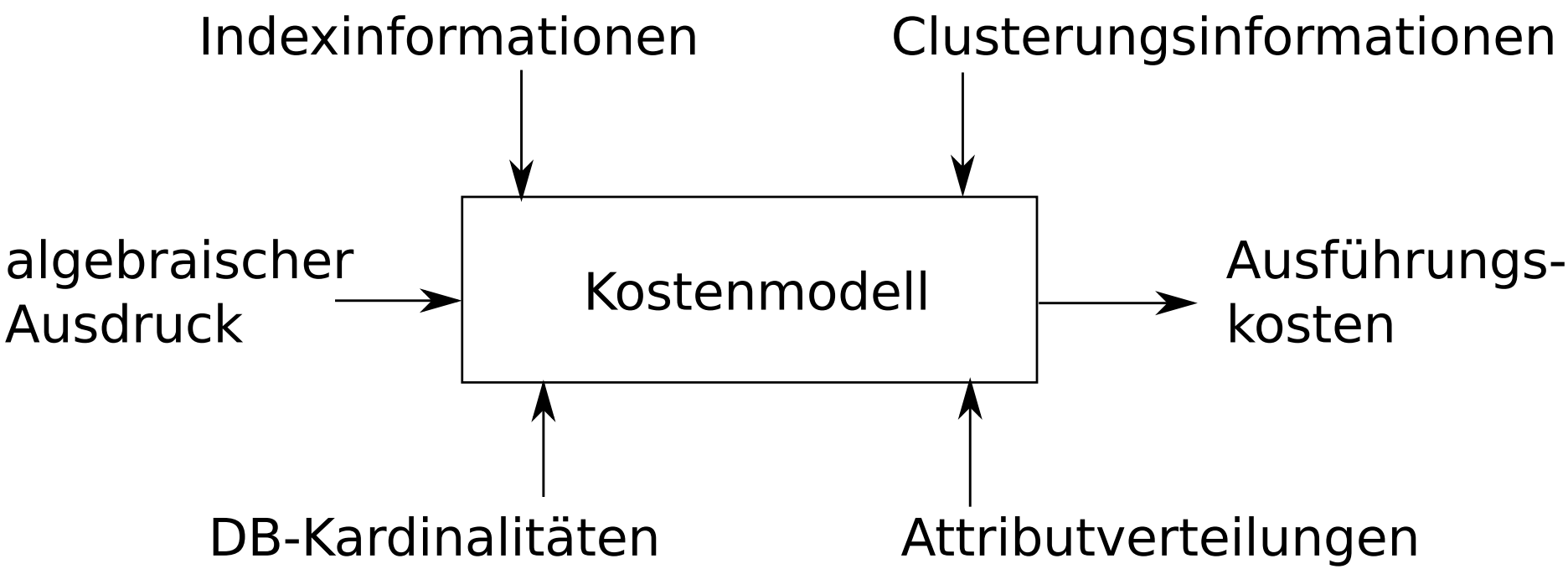
Selektivitätsschätzung: Werkzeuge

- Kenntnisse/Statistiken über zugrunde liegende Daten
- Generische Annahmen über Selektivität benutzter Prädikate
- Schätzung der Selektivität von o.g. Operatoren (z.B., Join)

Kostenbasierte Optimierung

- Generiere **alle** denkbaren Anfrageauswertungspläne:
  = Enumeration/Aufzählung möglicher Pläne
- Schätze deren Kosten ab:
  - Kostenmodell
  - Statistiken
  - Histogramme
  - Kalibrierung gemäß verwendetem Rechner
  - Abhängig vom verfügbaren Speicher
  - Aufwands-Kostenmodell
    - Durchsatz-maximierend
    - versus Antwortzeit-minimierend
- Behalte den Plan mit den geringsten geschätzten Kosten

Kostenmodelle



Grundlagen

- $T(R)$ ist die Anzahl der Tupel in Relation R
- $V(R, A)$ ist die Anzahl der verschiedenen Attributsausprägungen für Attribut A.
- Dementsprechend: $V(R, [A_1, A_2, ..., A_n])$

Aufgabe der Kostenschätzung

- Gegeben eine Anfrage, wie groß ist das Ergebnis und wie viele Tupel fallen als Zwischenergebnisse an?
- z.B. wie viele Tupel sind in $\sigma_{A=13434}(R)$

Schätzungen für Selektion

Gegeben eine Selektion $S = \sigma_{A=c}(R)$.
Wie viele Tupel sind in S?
Schätzung:

$$T(S) = \frac{T(R)}{V(R, A)}$$

Gilt falls die Werte für A zufällig aus allen möglichen Werten gezogen wurden.

Ungleichheit

Was ist bei $S = \sigma_{A<c}(R)$?
Im Allgemeinen, ohne weitere Annahmen: $T(S) = T(R)/2$

Aber, Intuition: man wählt mit solchen Bedingungen meist weniger Tupel aus.
Besser: $T(S) = T(R)/3$

Schätzung für "not-equals"

Gegeben eine Selektion $S = \sigma_{A \neq c}(R)$.
Wie viele Tupel sind in S?

Einfache Schätzung

- "Mehr oder weniger" alle Tupel erfüllen die Bedingung (naja, bis auf ein paar, aber egal)
- Also: $T(S) = T(R)$

Leicht verbessert

- Die Tupel mit $A = c$ erfüllen die Bedingung nicht.
- Also: $T(S) = T(R)\frac{V(R,A) - 1}{V(R,A)}$

Was passiert mit Kaskaden von Selektionen?
Selektivität ist Produkt der einzelnen Selektivitäten!

Selektion mit ODER-Bedingungen

Gegeben:

$$S = \sigma_{C_1 \vee C_2}(R)$$

Annahme die Bedingungen werden nie gemeinsam erfüllt

- Also: **entweder** gilt C_1 **oder** C_2
- Schätzung: Summe der beiden einzelnen Selektivitäten.
- Beobachtung: Überschätzt oft. Was kann dann passieren?

Besser: Annahme die Bedingungen sind unabhängig

- Annahme: m_1 Tupel erfüllen C_1 und m_2 Tupel erfüllen C_2
- Dann $T(S) = T(R) * (1 - (1 - \frac{m_1}{T(R)})(1 - \frac{m_2}{T(R)}))$

Selektion mit ODER-Bedingungen: Erläuterung

Wieso $T(S) = T(R) * (1 - (1 - \frac{m_1}{T(R)})(1 - \frac{m_2}{T(R)}))$?

$1 - \frac{m_1}{T(R)}$ ist der Anteil der Tupel die C_1 **nicht** erfüllen.

analog für C_2. Dann ist

$(1 - \frac{m_1}{T(R)})(1 - \frac{m_2}{T(R)})$ ist der Anteil der Tupel die C_1 **und** C_2 **nicht erfüllen**.

$(1 - ***)$ der Anteil der Tupel die C_1 **oder** C_2 erfüllen.

Klar: Multipliziert mit $T(R)$ ergibt $T(S)$

Andere Schätzer

Projektion

- Ändert die Kardinalität nicht
- Sehr wohl aber die Größe der Tupel!

Kartesisches Produkt

- Einfach: Produkt der beteiligten Kardinalitäten

Jetzt wird es **unklar was zu tun ist**:

Union

- Obere Schranke: Summe der beiden Kardinalitäten
- Untere Schranke: Größere der beiden Kardinalitäten
- Empfehlung aus Literatur: Irgendwas dazwischen, z.B. größere plus die halbe kleinere Kardinalität

Andere Schätzer

Schnitt

- Obere Schranke: Kleinste der beiden Kardinalitäten
- Untere Schranke: 0
- Empfehlung aus Literatur: Durschnitt dieser beiden Werte.

Differenz $R - S$

- Obere Schranke: $T(R)$
- Untere Schranke: $T(R) - T(S)$
- Empfehlung aus Literatur: $T(R) - \frac{T(S)}{2}$

Schätzung für Joins: Natürlicher Join

Zwei Relationen: $R = (X, Y)$ und $S = (Y, Z)$
Annahme: Y ist ein einfaches Attribut, keine Menge von Attributen. X und Z dürfen Mengen sein.

Was können wir dann sagen? Leider nur sehr wenig, da z.B. folgende Fälle auftreten können:

- Die beiden Relationen haben disjunkte Mengen für Y-Werte. Also ist der Join leer, d.h. $T(R \bowtie S) = 0$.
- Y ist Schlüssel von S und Fremdschlüssel in R. Dann findet jedes Tupel in R einen Joinpartner in S, also $T(R \bowtie S) = T(R)$
- Fast alle Tupel aus R und S haben den gleichen Wert für Y, also $T(R \bowtie S) = T(R) * T(S)$

Schätzung für Joins: Natürlicher Join: Annahmen

Häufig auftretende Fälle. Weitere Annahmen:

- Es gibt Y-Werte $y_1, y_2, y_3, ...$
- Relationen benutzen diese Werte in dieser Reihenfolge.
- Dann: Falls $V(R, Y) \leq V(S, Y)$ so gilt auch, dass jeder Y-Wert in R auch ein Y-Wert in S ist.

Erhaltung der Wertemengen

- Falls A kein Join-Attribut ist, gilt

$$V(R \bowtie S, A) = V(R, A)$$

- D.h. Attribut A verliert durch den Join keine möglichen Werte.

Schätzung für Joins: Natürlicher Join

Gesucht: Größe des Joins $R(X,Y) \bowtie S(Y,Z)$

- Gegeben, zwei Tupel: $r \in R$ und $s \in S$
- Was ist die Wahrscheinlichkeit, dass $r.Y = s.Y$?
- Annahme: $V(R,Y) \geq V(S,Y)$, also gibt es den Y-Wert von s in R.
- Also: Wahrscheinlichkeit, dass $r.Y = s.Y$ ist $1/V(R,Y)$
- Umgekehrt: falls $V(R,Y) < V(S,Y)$ analog.
- **Insgesamt:** Übereinstimmung in Y mit Wahrscheinlichkeit $1/max(V(R,Y),V(S,Y))$

$$T(R \bowtie S) = \frac{T(R) * T(S)}{max(V(R,Y),V(S,Y))}$$

"Essentially, all models are wrong, but some are useful."
(George E. P. Box)

Wie können Größen abgeschätzt werden?

Wichtig: Statistiken über Werte von Attributen, Größen von Relationen
Aber wie kann man diese berechnen?

Durch Scannen der gesamten Mengen

- Kann hin und wieder berechnet werden, naürlich besser nicht zur Anfragezeit.
- Moderne DMBS haben bestimmte Befehle dafür.

Ausprägungen für Attribut A

- Falls $V(R, A)$ nicht zu groß ist: speichere einen Zähler für jeden Wert von A.
- Sonst: Gruppierung. Evtl: Exaktes Speichern für eine Teilmenge der Werte (z.B. der häufigsten).

$\Rightarrow$ Histogramme oder parametrisierte Verteilungen!

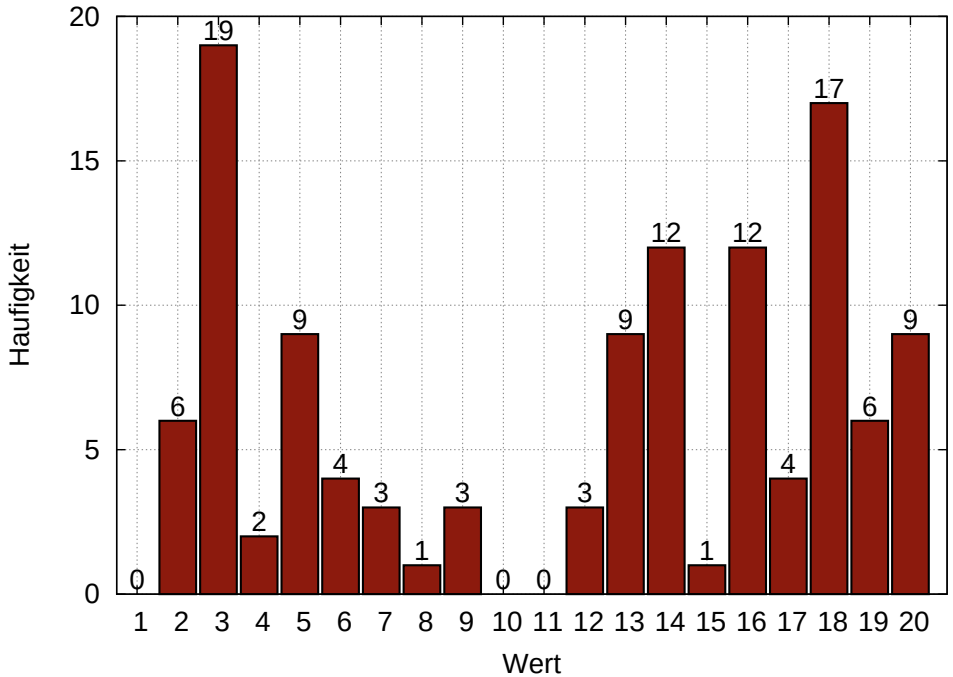
Beispieldaten

{2, 5, 3, 20, 18, 7, 16, 18, 18, 2, 2, 17, 14, 3, 20, 3, 16, 6, 7, 3, 16, 16, 15, 5, 20, 13, 16, 20, 12, 14, 13, 3, 14, 18, 14, 14, 16, 18, 19, 3, 5, 2, 5, 14, 20, 17, 3, 17, 16, 3, 2, 19, 3, 9, 13, 4, 3, 16, 14, 13, 13, 16, 20, 14, 4, 2, 3, 18, 7, 3, 5, 3, 6, 9, 18, 3, 16, 18, 20, 18, 5, 18, 5, 18, 13, 14, 19, 13, 14, 3, 14, 18, 14, 18, 18, 16, 19, 5, 3, 17, 18, 3, 19, 3, 20, 9, 16, 12, 20, 8, 12, 13, 13, 19, 18, 6, 3, 5, 18, 6}

Verteilung der Daten

Wert $\rightarrow$ Häufigkeit.

Dargestellt im "Histogramm-Stil":



Wir sehen: Es liegen 20 Werte im Wertebereich. Es gibt 120 Datenpunkte.

Alternativ: nicht die absolute Häufigkeit sondern normalisiert in [0,1], also "Wahrscheinlichkeit" bei zufälligem Zugriff auf Daten einen bestimmten Datenpunkt zu treffen.

Zusammenfassen/Beschreiben großer Datenmengen

Beschreibung durch parametrisierte Verteilung (Funktion)

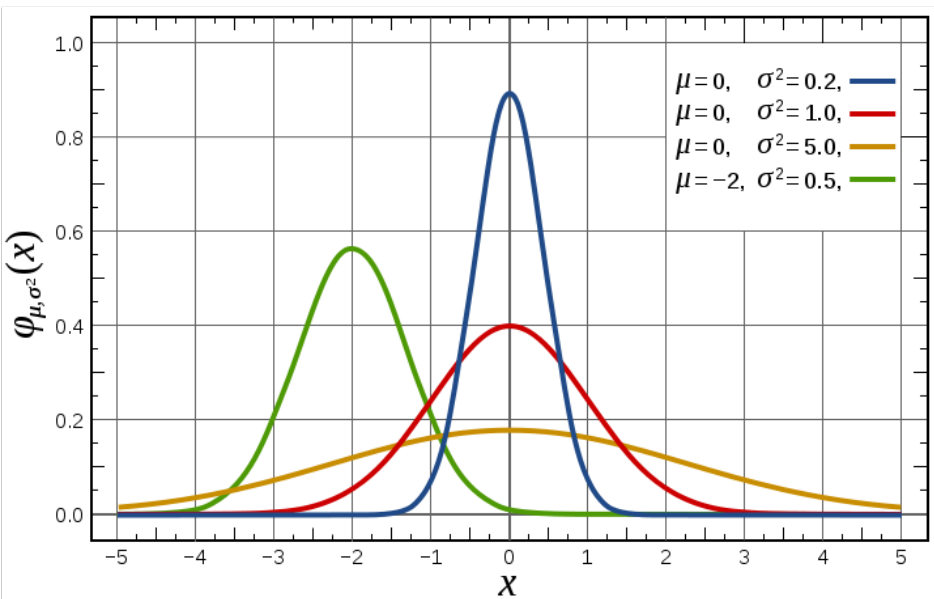
- Z.B. die Daten folgen einer Normalverteilung mit Erwartungswert μ und Varianz σ^2.

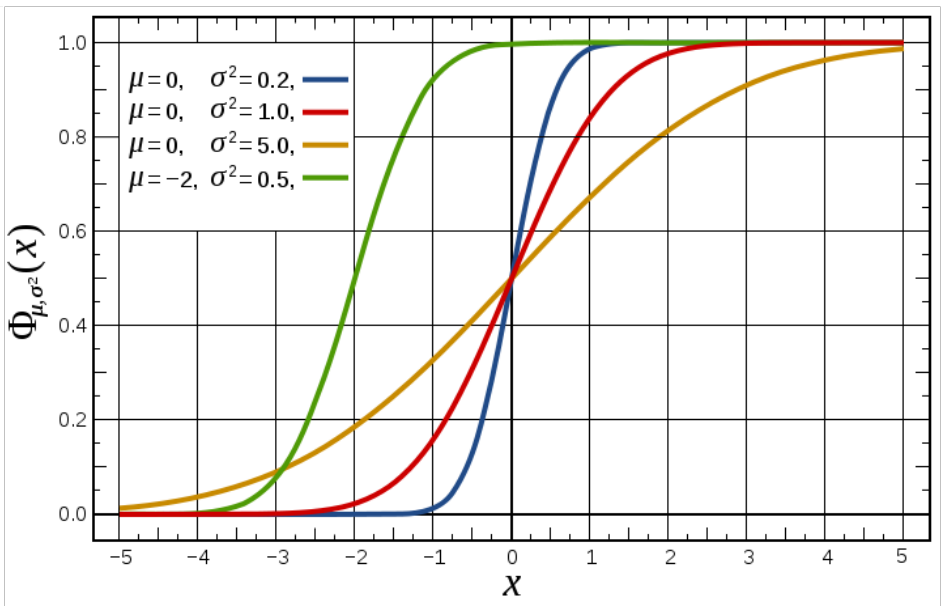
Zusammenfassen von Werten in Zellen (aka. Eimer oder Buckets)

- Wie viele Werte fallen in $[0, 5[$, wie viele Werte fallen in $[5, 10[$, etc.

Parametrisierte Verteilungen

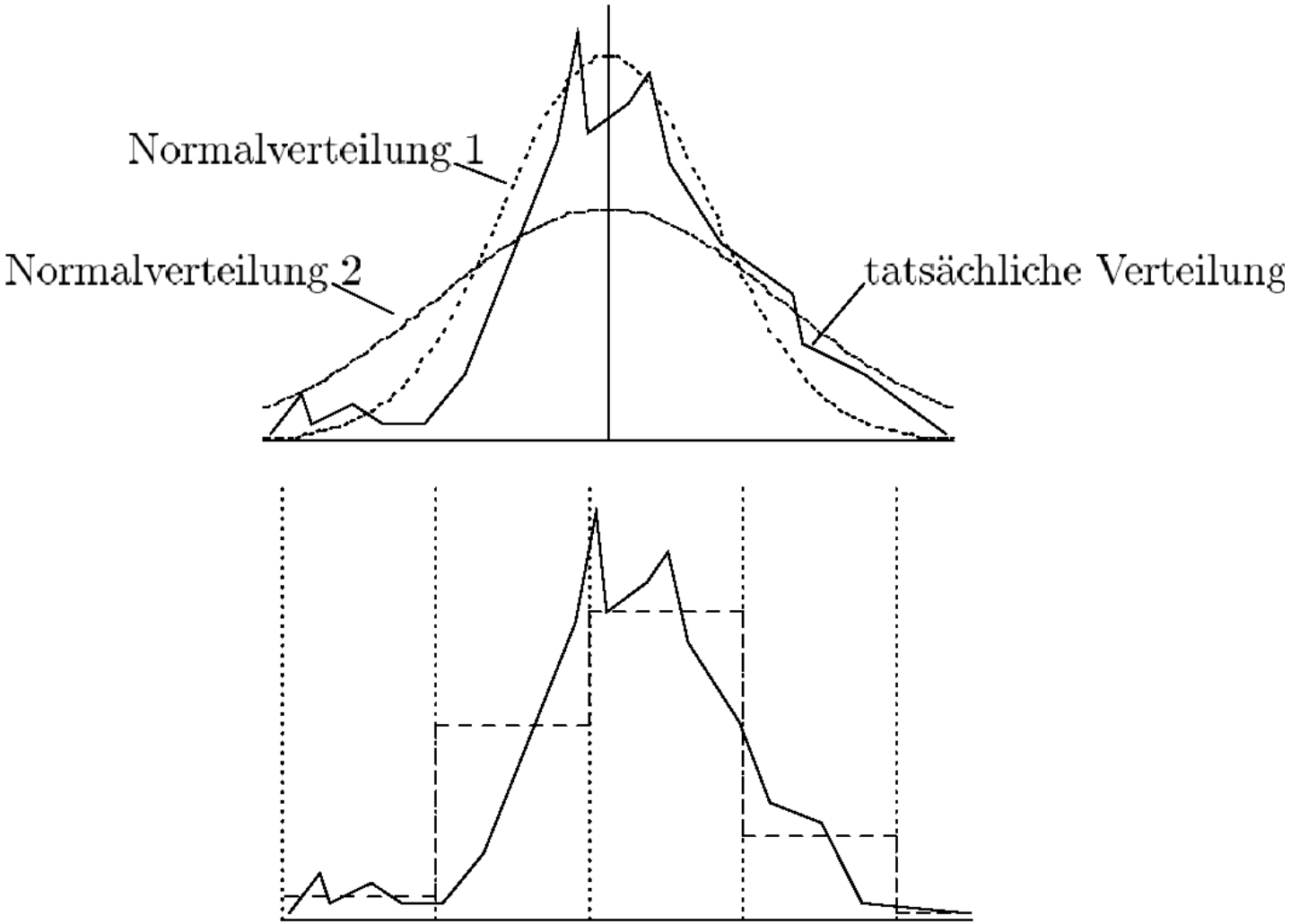
Dichtefunktion und Verteilungsfunktion der Normalverteilung:





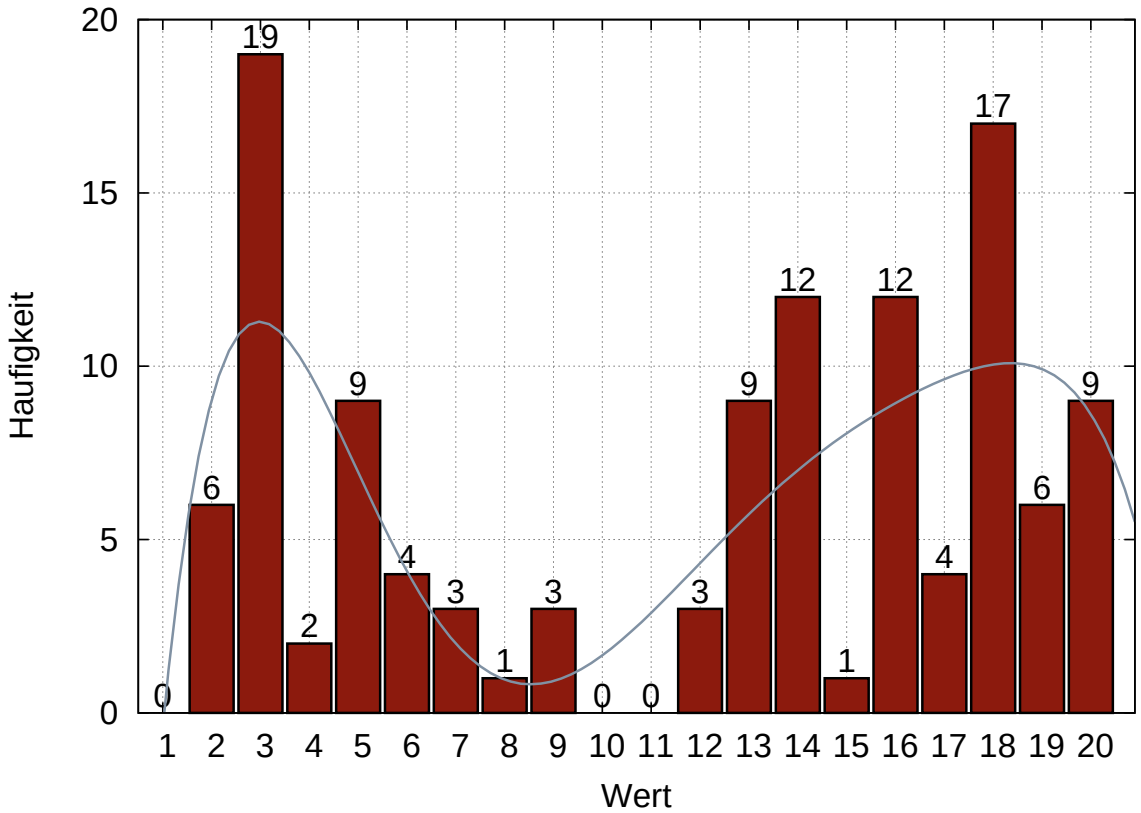
Abbildungen aus Wikipedia

Parametrisierte Verteilungen und Histogramme



Oft ist es schwierig eine tatsächliche Verteilung durch eine parametrisierte Verteilung auszudrücken. Histogramme sind flexibler.

Parametrisierte Verteilungen



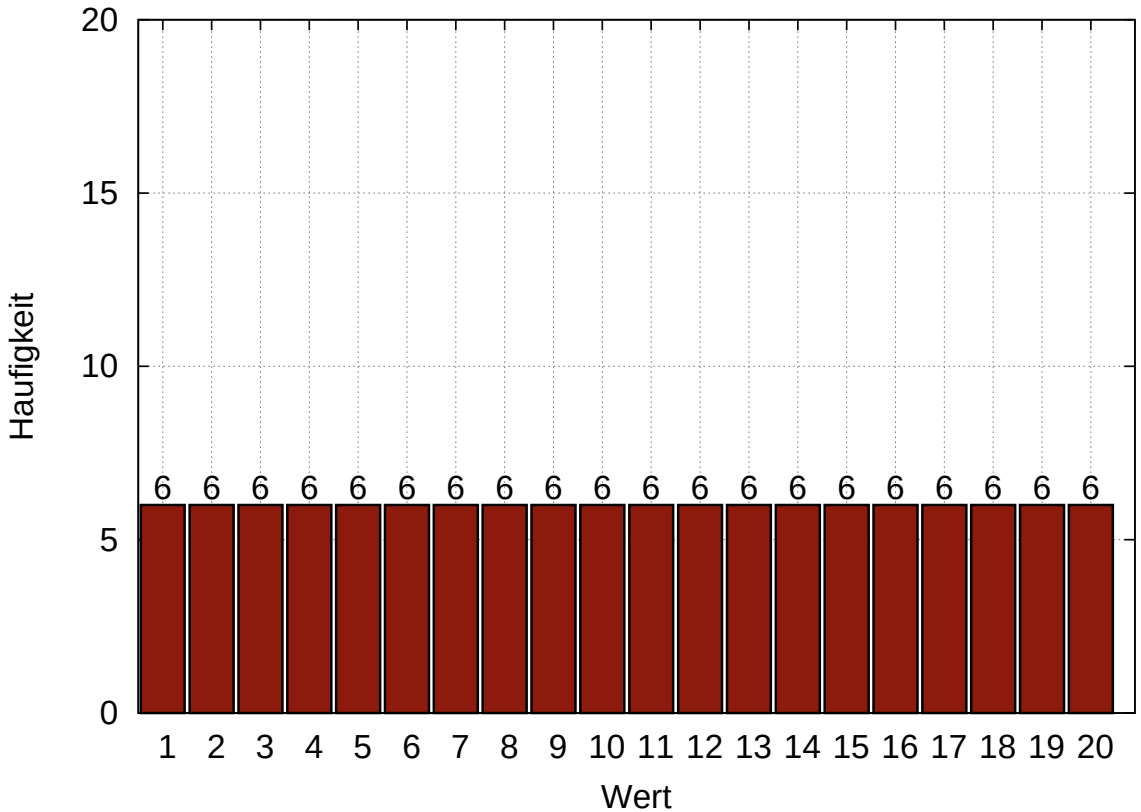
Hier, fit durch Polynom 6. Grades (mit Hilfe des Tools xmgrace):

$$f(x) := -21.884 + 29.533 * x - 9.2268 * x^2 + 1.2529 * x^3 - 0.085019 * x^4$$

$$+0.0028667 * x^5 - 3.8485 * 10^{-5} * x^6$$

Annahme Gleichverteilung

Es liegen 20 Werte im Wertebereich. Es gibt 120 Datenpunkte. Jeder Wert kommt, unter Annahme einer Gleichverteilung, also 6 Mal vor.



Oft nicht sehr realistische Darstellung (aber äußerst kompakt).

Histogramme

Histogram teilt den Wertebereich in Zellen oder Intervalle (Englisch: buckets oder cells). Für jede dieser Zellen wird die Anzahl der Elemente gespeichert, die in die Zelle fallen.

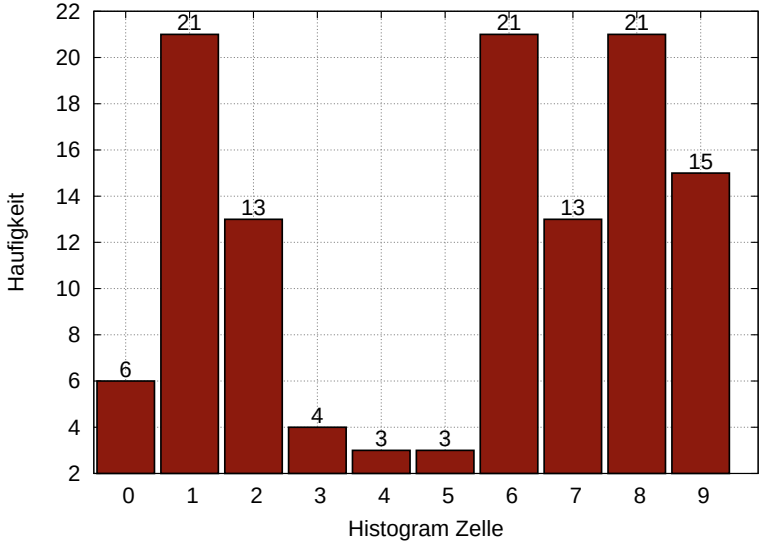
Equi-Width-Histogramme:

- Zellen habe immer die gleiche Breite im Wertebereich

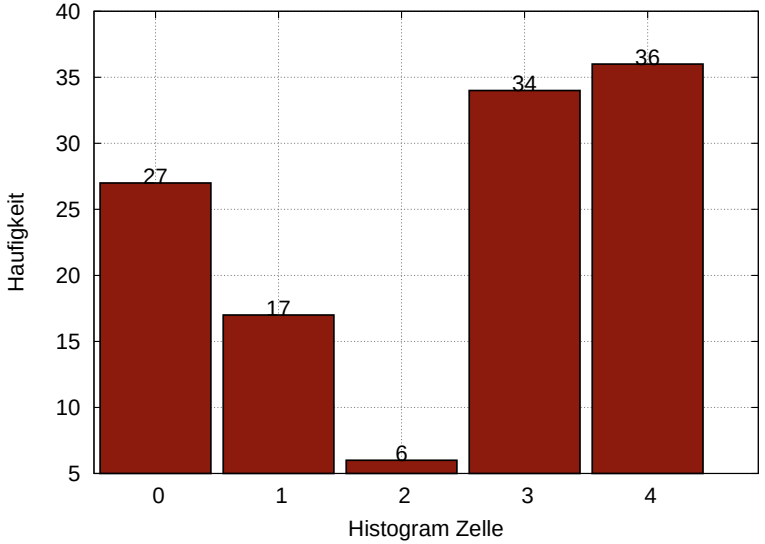
Equi-Depth (oder equi-height genannt) Histogramme:

- Zellen haben die gleiche "Höhe"
- Um dies zu erreichen: Breite der Zellen wird angepasst

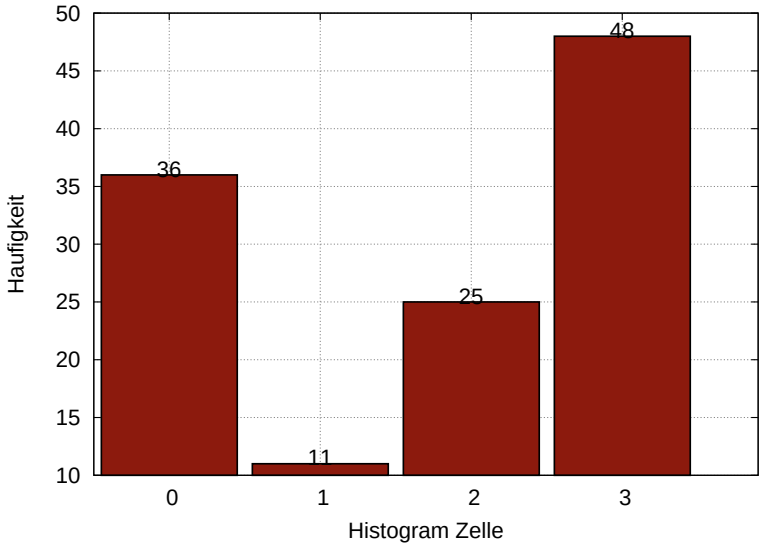
Equi-Width-Histogramme (Beispiele an o.g. Daten)



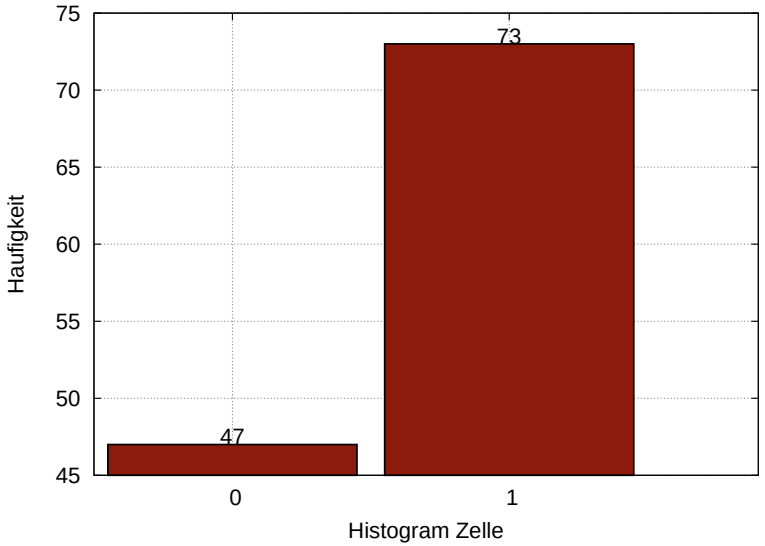
(a) Width = 2



(b) Width = 4

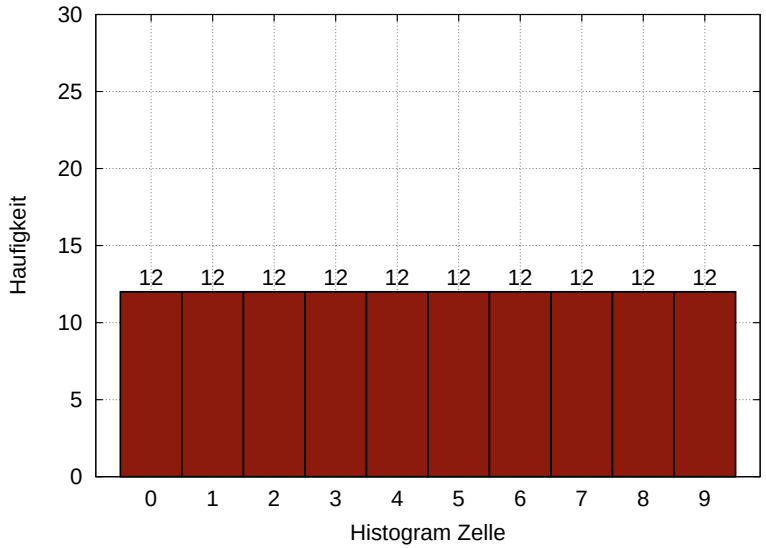


(a) Width = 5

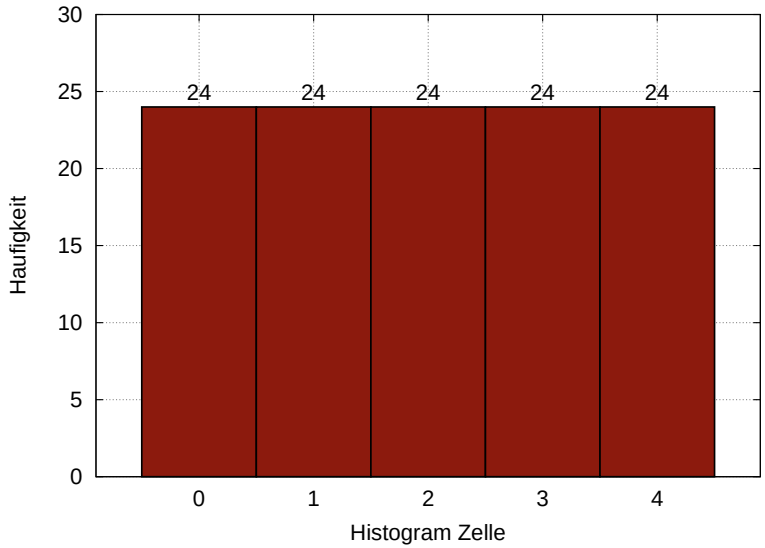


(b) Width = 10

Equi-Depth-Histogramme (Beispiele an o.g. Daten)



(a) Depth = 10%



(b) Depth = 20%

Grenzen der Zellen (je rechter Punkt, d.h. Ende)

- (a) 3,3,6,12,14,16,17,18,19
- (b) 3,12,15,18,20

Schätzungen mit Histogrammen

Punktanfragen

- Wie viele Tupel haben den Wert $A = 10$?
- Nachschauen: Zelle in die der Wert 10 fällt.
- Annahme hier: Gleichverteilung innerhalb der Zelle.
- Resultat: Anzahl der Tupel geteilt durch Breite der Zelle.
- Bzw. wenn bereits "normalisiert" dann nur Wert der Zelle.

Bereichsanfragen

- Wie viele Tupel haben einen Wert $A > 10$?
- Nachschauen: In welche Zelle fällt der Wert 10? (Achtung insbesondere bei Equi-Depth Histogrammen)
- Resultat: Summe der Größen der darüberliegenden Zellen und anteilsmäßig diese Zelle (wie oben).
- Oder: Histogramm für kumulative Verteilung betrachten

Formale Definition

Gegeben eine Menge von von n Datenpunkten s_i mit einer Zugeordneten Frequenz (Häufigkeit) $f(s_i)$. Wir schreiben auch f_i für $f(s_i)$.
Diese s_i seien bereits geordnet (obdA):

$$s_1 < s_2 < s_3 < \ldots s_n$$

Der Vektor der Häufigkeiten

$$F = [f(s_1), f(s_2), \ldots, f(s_n)]$$

Histogramm

- Ein Histogramm partitioniert diesen Vektor der Häufigkeiten in B Buckets (Zellen oder Intervalle) I_i. Dabei ist $B << n$.
- Jedes Intervall I_i wird durch einen Wert h_i (z.B. Durchschnitt) repräsentiert.

Fehler

Das heisst, für ein Intervall $I_i = [b_i, e_i]$, wobei b_i der Beginn und e_i das Ende des Intervalls markieren, wird die Häufigkeit der Werte $s_{b_i}, s_{b_i+1}, s_{b_i+2}, \ldots s_{e_i}$, die in dem Intervall enthalten sind, durch h_i approximiert.

Beispiel:

- Wie häufig kommt der Wert s_{b_i+2} vor?
- Natürlich genau $f(s_{b_i+2}) = f_{b_i+2}$ mal.
- Aber durch Histogramm eben approximiert durch h_i

Fehler

- Der Fehler, der dadurch für Wert s_{b_i+2} entsteht ist also $h_i - f_{b_i+2}$
- Normalerweise wird $|h_i - f_{b_i+2}|$ oder $(h_i - f_{b_i+2})^2$ benutzt

Sum Squared Error (SSE)

Es ist üblich h_i als Durchschnitt über die in I_i enthaltenen Daten (Häufigkeiten) zu berechnen, also

$$h_i = AVG([b_i, e_i]) = \frac{\sum_{b_i \leq k \leq e_i} F[k]}{e_i - b_i + 1}$$

Fehlermaß

Sum Squared Error (SSE)

$$SSE([a, b]) = \sum_{k=a}^{b} (F[k] - AVG([a, b]))^2$$

- Klar: Je kleiner dieser Fehler, desto besser die Approximation der (aller) Daten durch das Histogramm

V-Optimale Histogramme

Algorithmus zum Berechnen des bzgl. SSE optimalen Histogramms (mit Kosten SSE^*) unter Verwendung von B Intervallen (Buckets).

$$SSE^*(i,k) = min_{1 \leq j \leq i}\{SSE^*(j,k-1) + SSE([j+1,i])\}$$

Ausnutzen, dass

$$SSE([i,j]) \geq SSE([i,k]) + SSE([k+1,j])$$

Literatur: Optimal Histograms with Quality Guarantees. H.V. Jagadish et al., VLDB Conference, 1998.

V-Optimale Histogramme: Algorithmus

$SSE^*(i,k)$ beschreibt optimalen Fehler für Histogram über ersten i Werten und k Zellen.

Dynamische Programmierung über

$$SSE^*(i,k) = min_{1 \leq j \leq i}\{SSE^*(j,k-1) + SSE([j+1,i])\}$$

for $k := 1$ **to** B
  for $i := 1$ **to** n
    for $j := 1$ **to** i
      if $besterror[j][k-1] + SSE(j+1,i) < besterror[i][k]$
        update $besterror[i][k]$
      ...
    end
  end
end

Initialisierung von $besterror[][]$ und Randfälle nicht im Code gezeigt.

Implementierungs Trick für SSE

Lemma 1 im original Papier von Jagadish et al.

$$SSE([i,j]) = \sum_{i \leq k \leq j} F[k]^2 - (j-i+1) * AVG([i,j])^2$$

mit

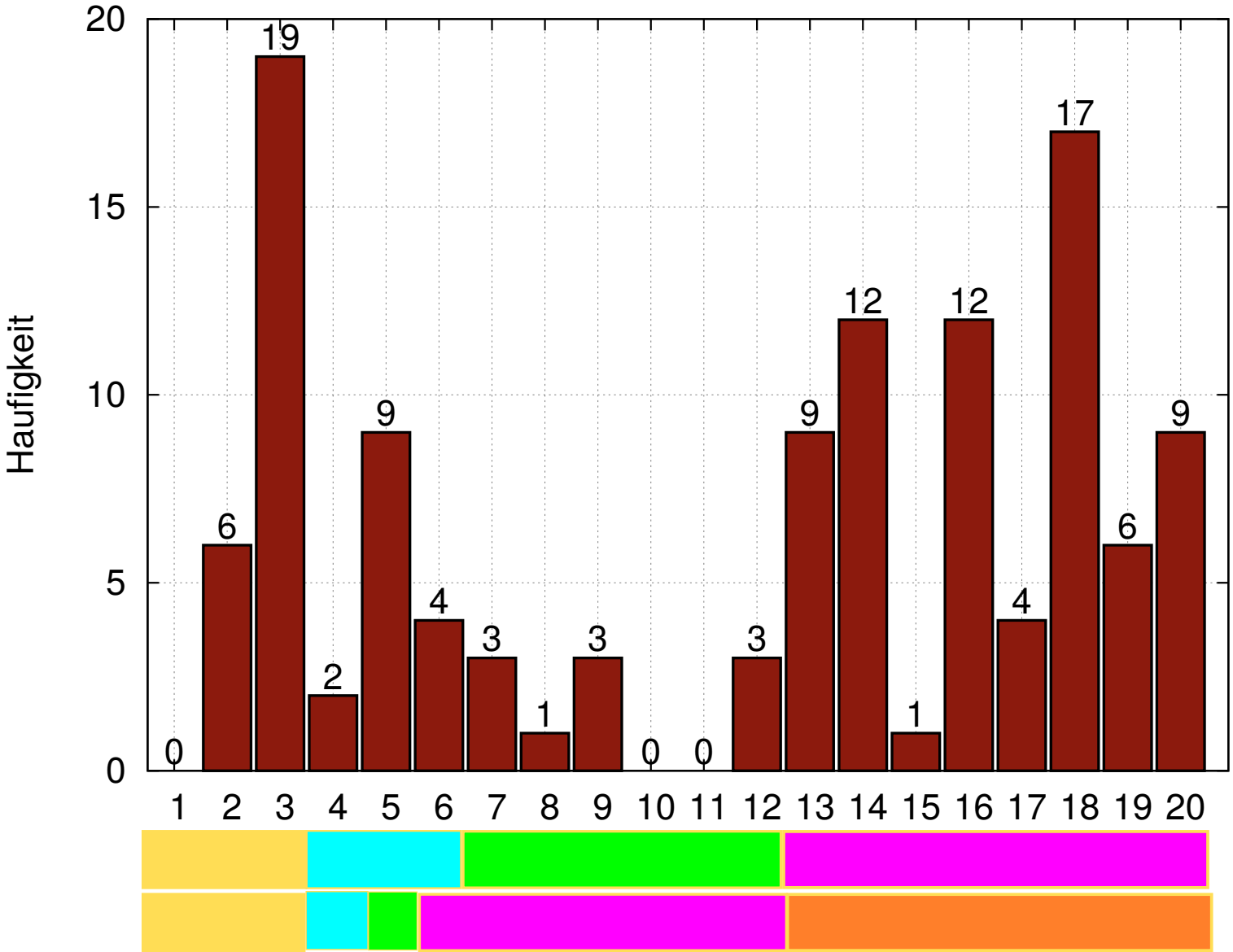
$$\sum_{i \leq k \leq j} F[k]^2 = PP[k] - PP[i-1]$$

und

$$AVG([i,j]) = \frac{P[j] - P[i-1]}{(j-i+1)}$$

wobei $P[i] = \sum_{1 \leq k \leq i} F[k]$ und $PP[i] = \sum_{1 \leq k \leq i} F[k]^2$ (welche vorberechnet werden).

Beispiel: Für o.g. Daten und B = 4 bzw. 5 Zellen



Deskriptive Statistik

- Minimaler und maximaler Wert eines Attributs
- Durchschnittlicher Wert und Median
- Weitere Aussagen über Verteilungen
- Beispielwerte

Anwendungen

- Produkt-Mananger: "In 99,9% aller Fälle liegt die Antwortzeit unseres Systems XYZ unter 10ms."
- Professor: "75% aller Studenten, die die Klausur mitgeschrieben haben, haben mindestens 80 Punkte erreicht."

Quantile einer Verteilung

Definition

Gegeben eine Zufallsvariable X mit Verteilungsfunktion F. Für ein $p \in [0, 1]$, definieren wir $F^{-1}(p)$ als kleinsten Wert x, so dass $F(x) \geq p$.

Dieser Wert $F^{-1}(x)$ wird das p Quantil von X genannt. Die Funktion F^{-1} wird Quantilfunktion genannt.

- Das p Quantil wird auch $100p$ Perzentil genannt.
- Das $1/2$ Quantil, bzw. das 50. Perzentil einer Verteilung wird auch Median genannt.
- Das $1/4$ Quantil, bzw. das 25. Perzentil, wird auch **erstes Quartil** genannt,
- das $3/4$ Quantil, bzw. das 75. Perzentil, wird auch **drittes Quartil** genannt.

Probabilistic Counting

$V(R, A)$ ist die Anzahl der verschiedenen Attributsausprägungen für Attribut A.
Wie kann diese Größe berechnet werden?

- Klar, via Duplikat-Eliminierung durch Sortieren oder durch Hashing
- Oder durch probabilistische Methoden (Schätzer)

Flajolet Martin (FM) Sketch (aka. Hash Sketch)

Vorgeschlagen von Flajolet und Martin in 1985[1]

- Erzeuge einen leeren Bitvektor B der Länge $m = log(N)$
- Scan über Eingabedaten: Dabei wird für jedes Objekt eine Position im Bitvektor berechnet und auf "1" gesetzt:
  - Hashing eines Objekts i in eine m-bit Zahl $h(i)$
  - Berechne Position k des am wenigsten signifikanten "1" Bits von h(i)
  - Setze bit $B[k]$ auf "1"

Beispiel

Eingabe: 17, 5, 19, 211, 17, 5, 31
Annahme h(17)=010100, dann ist das am wenigsten signif. 1 Bit = 3
Annahme h(5)=000101, dann ist das am wenigsten signif. 1 Bit = 1

[1] Philippe Flajolet, G. Nigel Martin: Probabilistic Counting Algorithms for Data Base Applications. J. Comput. Syst. Sci. 31(2): 182-209 (1985)

Schätzer

- Am Ende sieht B dann z.B. so aus: $B = 111010$
- Betrachte die Position t des am weitesten links stehenden "0" Bits, hier im Beispiel $t = 4$.
- Dann ergibt sich die Schäzung für die tatsächliche Anzahl n als

$$\hat{n} = 2^t / 0.7735$$

in unserem Beispiel mit $t = 4$: $\hat{n} = 2^4 / 0.7735 \approx 20.685$

Verbesserung der Schätzung

Verwendung mehrerer Bitvektoren B (entsprechend mit unterschiedlichen Hashfunktionen h) und Berechnung eines Durchnittlichen Werts von t.

Idee/Intuition

- $B[0]$ wird ungefähr $n/2$ mal gesetzt
- $B[1]$ wird ungefähr $n/4$ mal gesetzt

...

Also:

- $B[i] = 0$ falls $i >> log_2(n)$
- $B[i] = 1$ falls $i << log_2(n)$
- "Mischung" aus 1s und 0s um $i \approx log_2(n)$ herum

Tuning von Datenbanken

- Statistiken (Histogramme, etc.) müssen explizit angelegt werden
- Andernfalls liefern die Kostenmodelle falsche Werte
- Oracle:
  - `analyze table Professoren compute statistics for table;`
  - Man kann sich auch auf approximative Statistiken verlassen
  - Anstatt compute verwendet man estimate
- DB2:
  - `runstats on table ....`
- Postgres:
  - `analyze`
  - `http://www.postgresql.org/docs/9.0/static/catalog-pg-statistic.html`

Statistiken in Postgresql

Tabelle analysieren mit:

```
analyze lineitem;
```

Daten werden in einer internen Tabelle (pg_statistic) abgelegt.

```
select s.*
from pg_statistic s, pg_stat_all_tables t
where s.starelid = t.relid and t.relname = 'lineitem'
```

Beispiel: Auszug aus pg_statistic (in Postgresql)

Für Tabelle `lineitem` des TPC-H Datasets (`http://www.tpc.org/tpch/`)

	starelid oid	staattnum smallint	stainherit boolean	stanullfrac real	stawidth integer	stadistinct real	stakind1 smallint	stakind2 smallint
1	16408	4	f	0	4	7	1	3
2	16408	15	f	0	11	7	1	3
3	16408	1	f	0	4	370025	1	2
4	16408	2	f	0	4	192835	1	2
5	16408	3	f	0	4	9969	1	2
6	16408	5	f	0	5	50	1	3
7	16408	6	f	0	8	0.121147	1	2
8	16408	7	f	0	4	11	1	3
9	16408	8	f	0	4	9	1	3
10	16408	9	f	0	2	3	1	3
11	16408	10	f	0	2	2	1	3
12	16408	11	f	0	4	2504	1	2
13	16408	12	f	0	4	2459	1	2
14	16408	13	f	0	4	2514	1	2
15	16408	14	f	0	26	4	1	3
16	16408	16	f	0	27	0.140107	1	2

Beispiel: Auszug aus pg_statistic (in Postgresql) (2)

Für Tabelle `lineitem` des TPC-H Datasets (`http://www.tpc.org/tpch/`)

stanumbers1 real[]	stanumbers2 real[]	stanumbers3 real[]	stanumbers4 real[]	stanumbers5 real[]	stavalues1 anyarray	stavalues2 anyarray
{0.247433,	{0.173839}				{1,2,3,4,5,6,7}	
{0.1444,0.	{0.156636}				{"REG AIR ","FOB "	
{0.000133		{0.999999}			{5411684,182887,371522,710	{227,652
{0.000133		{-0.004888			{97390,118951,130319,2308,	{16,1895
{0.0004,0.		{-0.016965			{8451,1263,9259,3401,3511,	{1,108,2
{0.022033	{0.022108				{40.00,38.00,19.00,32.00,2	
{0.0001,0.		{0.003937			{27763.92,37923.76,39240.0	{938.03,
{0.0941667	{0.090250				{0.10,0.06,0.00,0.07,0.04,	
{0.113333,	{0.108163}				{0.04,0.03,0.07,0.02,0.06,	
{0.506933,	{0.380856}				{N,R,A}	
{0.501033,	{0.498091}				{O,F}	
{0.0008,0.		{-0.00247			{1994-11-09,1996-07-18,199	{1992-01
{0.000933		{-0.00256			{1993-07-31,1993-12-10,199	{1992-01
{0.000866		{-0.00257			{1998-01-27,1993-06-02,199	{1992-01
{0.255,0.2	{0.242461}				{"TAKE BACK RETURN	
{0.000266		{0.009277			{" carefully"," furiously	{" about