3. Einführung in mehrdimensionale, insbesondere räumliche Zugriffsstrukturen

Ausgangspunkt: eine Datenbank, die eine Sammlung von *sehr vielen* mehrdimensionalen Objekten ist,

- z.B. von *points*, *lines*, *regions*-Objekten, möglichst mit ihren kleinsten umgebenden Rechtecken (*bounding boxes*),
- oder auf feinerer Ebene z.B. von unterliegenden Realm-Punkten und -Segmenten.

Hauptanforderungen:

→ Suchen nach mehreren Kriterien,
z.B. nach den zwei (x, y)-Dimensionen

→ Indexierung von Punkten *und*
von Objekten mit räumlicher Ausdehnung

Klassifikation von Anfragen

Definitionen:

- Ein Datenraum D ist eine Sammlung von N Sätzen des Typs R = (A_1,... A_n), wobei jeder Satz ein Punktobjekt durch ein geordnetes n-Tupel t = (a_1,a_2,... a_n) von Werten darstellt. Die Attribute A_1,..., A_k ($k \leq n$) mögen den Schlüssel bilden.
- Eine Anfrage Q spezifiziert einige Bedingungen, die von den Schlüsselwerten der Sätze in der Treffermenge erfüllt sein müssen.
- Schnittbildende Anfragen (*intersection queries*): gesuchte Objekte überlappen mit dem Anfragebereich
- Enthaltenseins- oder Umschließungsanfragen (*containment queries*): gesuchte Objekte sind ganz im Anfragebereich enthalten oder enthalten diesen vollständig

[0]Die folgenden Folien basieren überwiegend auf Begleitmaterialien von Härder/Rahm zu Vorlesungen über ihr Buch "Datenbanksysteme – Konzepte und Techniken der Implementierung", erschienen in der 2. Auflage beim Springer-Verlag 2001.

Klassifikation der schnittbildenden Anfragen

1. ***Exakte Anfrage*** *(exact match query)*:
   spezifiziert für jeden Schlüssel einen Wert

   $$Q = (A_1 = a_1) \wedge (A_2 = a_2) \wedge \ ... \wedge (A_k = a_k)$$

2. ***Partielle Anfrage*** *(partial match query):*
   spezifiziert s < k Schlüsselwerte

   $$Q = (A_{i_1} = a_{i_1}) \wedge \ (A_{i_2} = a_{i_2}) \wedge \ \ ... \wedge \ (A_{i_s} = a_{i_s})$$
   mit $1 \leq s < k$ und $1 \leq i_1 < i_2 < ... < i_s \leq k$

3. ***Bereichsanfrage*** *(range query)*:
   spezifiziert einen Bereich $r_i = [l_i , u_i]$ für jeden Schlüssel A_i

   $$Q \ = (A_1 \in r_1) \wedge \ ... \wedge (A_k \in r_k)$$
   $$\equiv (A_1 \geq l_1) \wedge \ (A_1 \leq u_1) \wedge \ ... \wedge (A_k \geq l_k) \wedge \ (A_k \leq u_k)$$

4. ***Partielle Bereichsanfrage*** *(partial range query):*
   spezifiziert für s < k Schlüssel einen Bereich r_{ij}

   $$Q = (A_{i1} \in r_{i1}) \ \wedge \ ... \wedge (A_{is} \in r_{is})$$

Klassifikation von Anfragen (Forts.)

➥ bei den schnittbildenden Anfragen lassen sich alle 4 Fragetypen als allg. Bereichsanfrage (3.) ausdrücken:

- genauer Bereich $[l_i = a_i = u_i]$ für Gleichheit $A_i=a_i$
- unendlicher Bereich $[-\infty ,\infty]$ für ausgelassene Schlüssel A_i

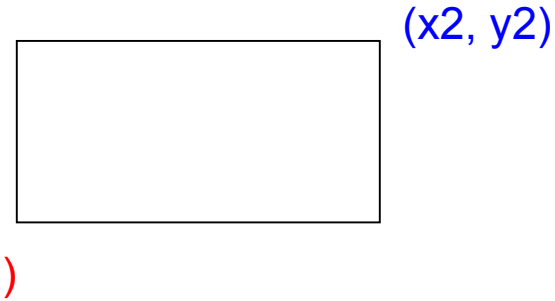
Klassifikation der Enthaltenseins-Anfragen

- ***Punktanfrage*** (*point query*): Gegeben ist ein Punkt im Datenraum D.
  Finde alle Objekte, die ihn enthalten.

- ***Gebietsanfrage*** (*region [window] query):* Geg. ist ein Anfragegebiet [-fenster].
  Finde alle Objekte, die es enthalten / darin enthalten sind .

Beispiele: Suche nach Rechtecken

Tabelle RECTANGLES (x1, y1, x2, y2)



a) Bestimme alle Rechtecke, die den Punkt (2,5) enthalten:

```
SELECT    x1, y1, x2, y2
FROM      RECTANGLES
WHERE     x1 <= 2 AND  x2 >= 2 AND  y1 <= 5  AND  y2 >= 5
```

b) Bestimme die Rechtecke, deren Eckpunkt links unten in (0,0,10,10) liegt:

```
SELECT    x1, y1, x2, y2
FROM      RECTANGLES
WHERE     x1  >= 0 AND x1 <= 10 AND y1 >= 10  AND y1 <= 10
```

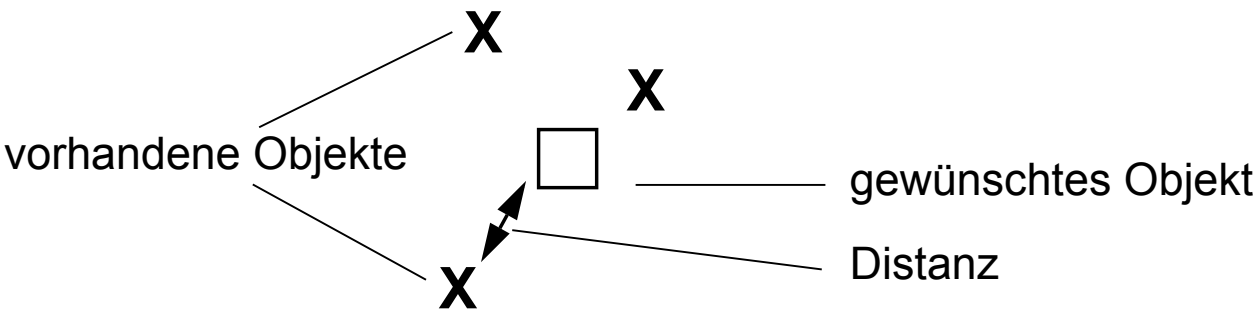
➥ das sind harmlose Fragen, die sogar im Relationenmodell mühelos beantwortet werden können

Klassifikation von Anfragen (Forts.)

Nächster-Nachbar-Anfragen (*best match query*, *nearest neighbor query*):

- gewünschtes Objekt nicht vorhanden

  ➥ Frage nach möglichst ähnlichen Objekten



- "best" wird bestimmt über eine Distanzfunktion.

- Beispiele:
  - Punkt liegt in der Nähe eines Bezugspunktes.
  - Objekt erfüllt nur 8 von 10 geforderten Eigenschaften.

- **Bestimmung des nächsten Nachbarn:**

  d = Distanzfunktion geeignet zu definieren !

  B = Sammlung von Punkten im k-dim. Raum

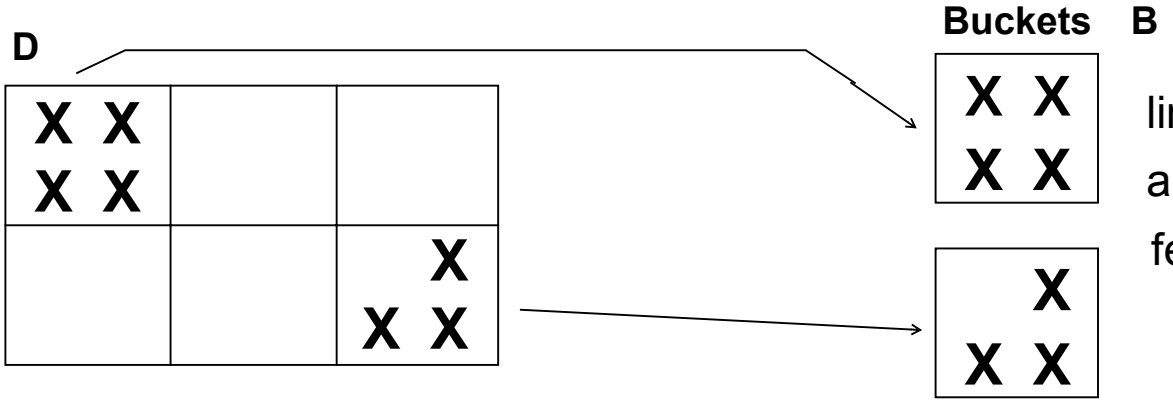
  Gesucht: nächster Nachbar von p (in B)

  Der nächste Nachbar ist q, wenn

$$(\forall r \in B)\ [d(r,p) \geq d(q,p)]$$

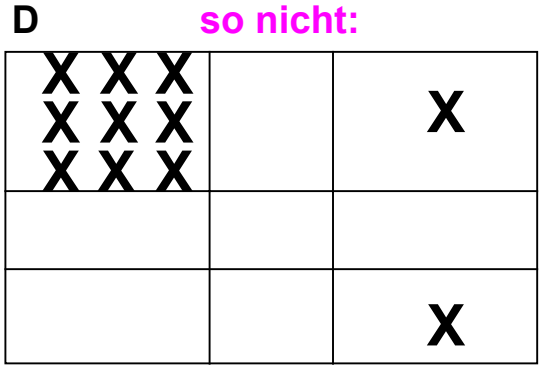
Grundprobleme/-anforderungen an Speicherung

1. Erhaltung der topologischen Struktur (Clusterbildung)



linearer Speicher
aufgeteilt in Buckets
fester Größe

2. Anpassung an stark variierende Dichte der Objekte



Starke Änderung der räumlichen Belegung über die Zeit

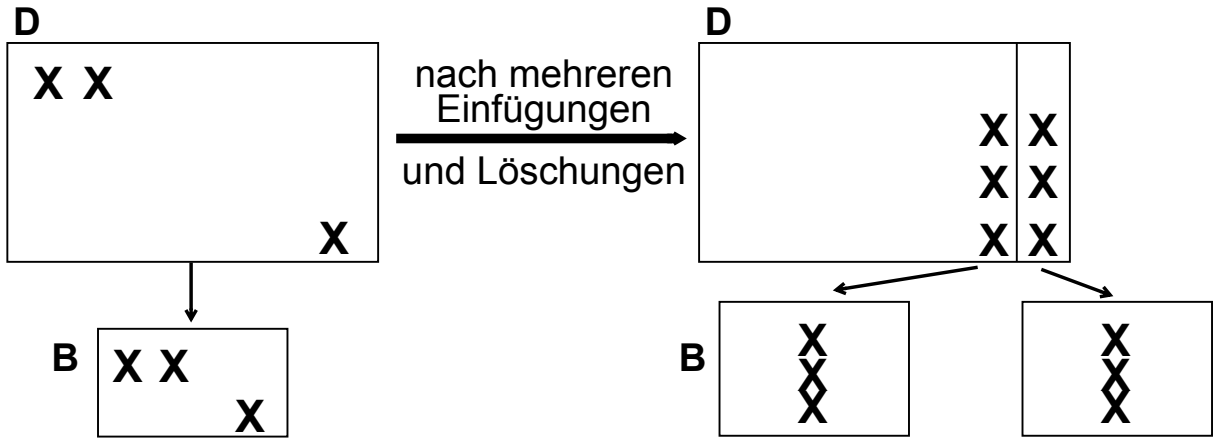
↳ keine regelmäßige Aufteilung von D

↳ aber: gleiche Bucketgrößen

3. Verschiedene Objektrepräsentationen

- Punktobjekte
- Objekte mit Ausdehnung

4. Reorganisation bei dynamisch veränderlichem Datenbestand



⇒ **balancierte Zugriffsstrukturen:**

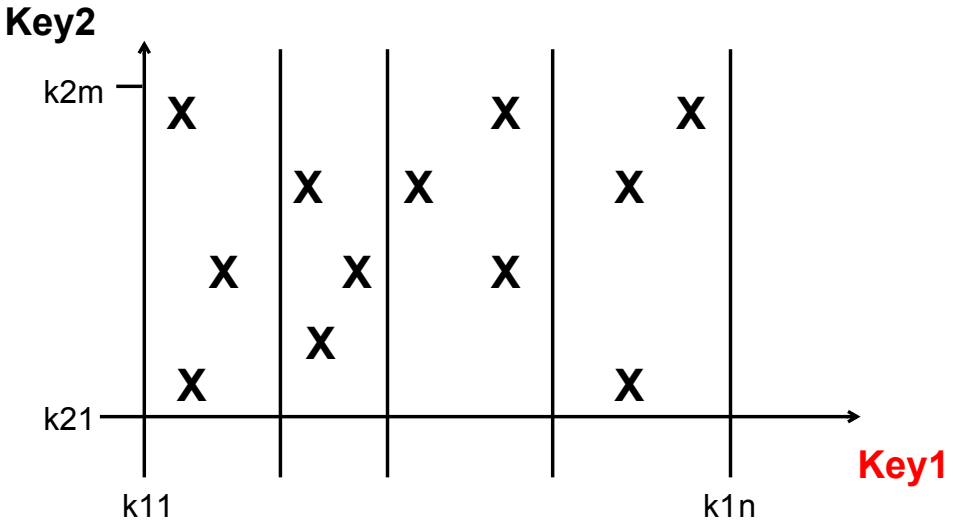
- beliebige Belegungen und Einfüge-/Löschreihenfolgen
- Garantie eines gleichförmigen Zugriffs ➥ 2 oder 3 Externspeicherzugriffe

Mehrattributzugriff über eindimensionale Zugriffspfade

- **bisher:**

  Indexierung (Invertierung) einer Dimension, z. B. B*-Baum

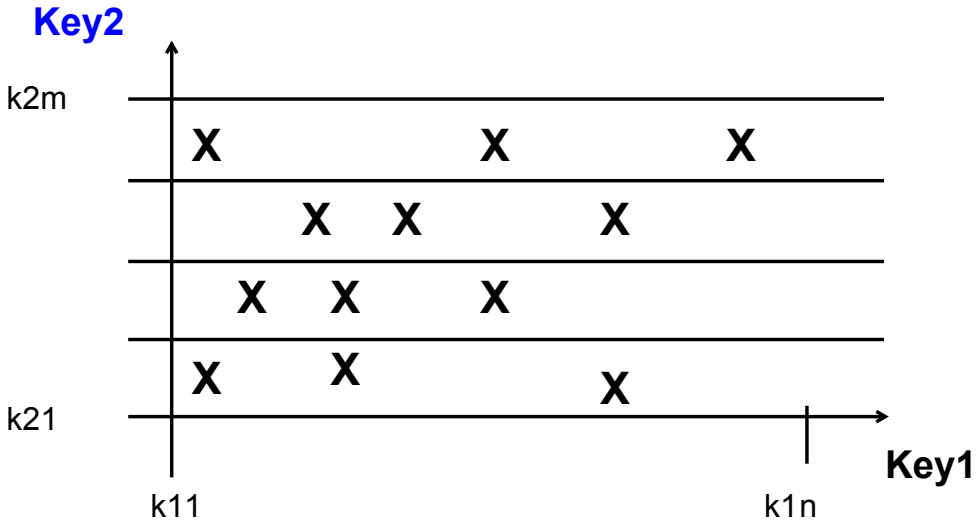
- **Zerlegungsprinzip des Schlüsselraumes beim B*-Baum (2-dim.):**



B*-Baum (Key1)

➥ Partitionierung des Raumes nach Werten von Key1

- **zusätzlicher B*-Baum möglich:**



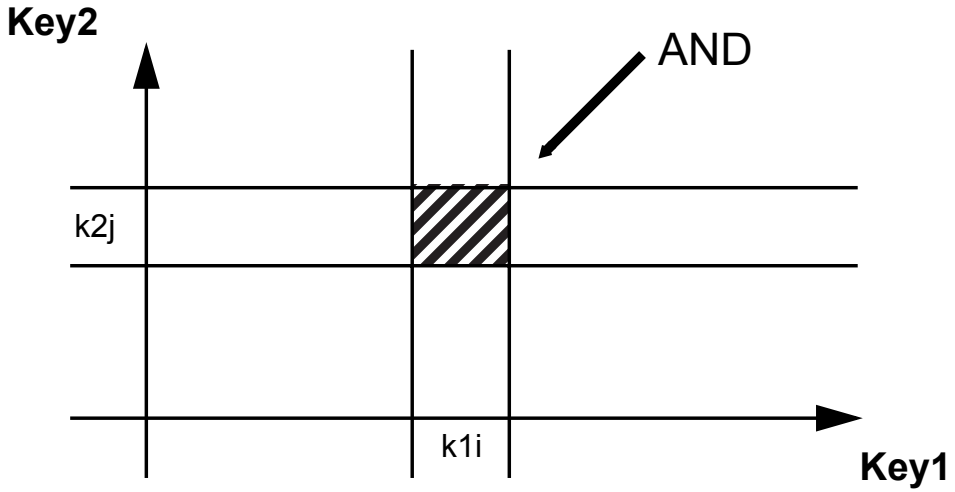
B*-Baum (Key2)

➥ Partitionierung des Raumes nach Werten von Key2

Mehrattributzugriff über eindimensionale Zugriffspfade (Forts.)

- **Zugriff nach** (Key1 = k1i) { OR / ... / AND } (Key2 = k2j) **Separate Schlüssel**
  - Zeigerliste für k1i : aus B*-Baum (Key1)
  - Zeigerliste für k2j : aus B*-Baum (Key2)

➥ **Mischen von Zeiger-(TID-)Listen + Zugriff auf Ergebnistupel**



➥ große Zeigerlisten und Zwischenergebnisse !

- **Simulation des mehrdimensionalen Zugriffs mit einem B*-Baum ?**

Idee: **Konkatenierte Schlüssel**:
Konkatenierte Werte:

Key1	**Key2**
k11	k21
k11	k22
⋮	
k11	k2m
k12	k21
k12	k22
⋮	
k12	k2m
k13	k21
⋮	
k1n	k2m

Unterstützung von Suchoperationen ?

- (Key1 = k1i) AND (Key2 = k2j) ? ja
- Key2 = k2j ? nein
- Key1 = k1i ? ja
- OR-Verknüpfung ? nein

aber:

keine Erhaltung von Nachbarschaften, also immer noch schlechte Unterstützung von Bereichsanfragen

Mehrdimensionale Verfahren zur Organisation der Datensätze

Quad-Tree (Quadranten-Baum)

- **Speicherungsstruktur für 2-dimensionalen Mehrattributzugriff:**

  Ziel: Berücksichtigung von Nachbarschaftsbeziehungen

  Zerlegungsprinzip des Datenraumes D:

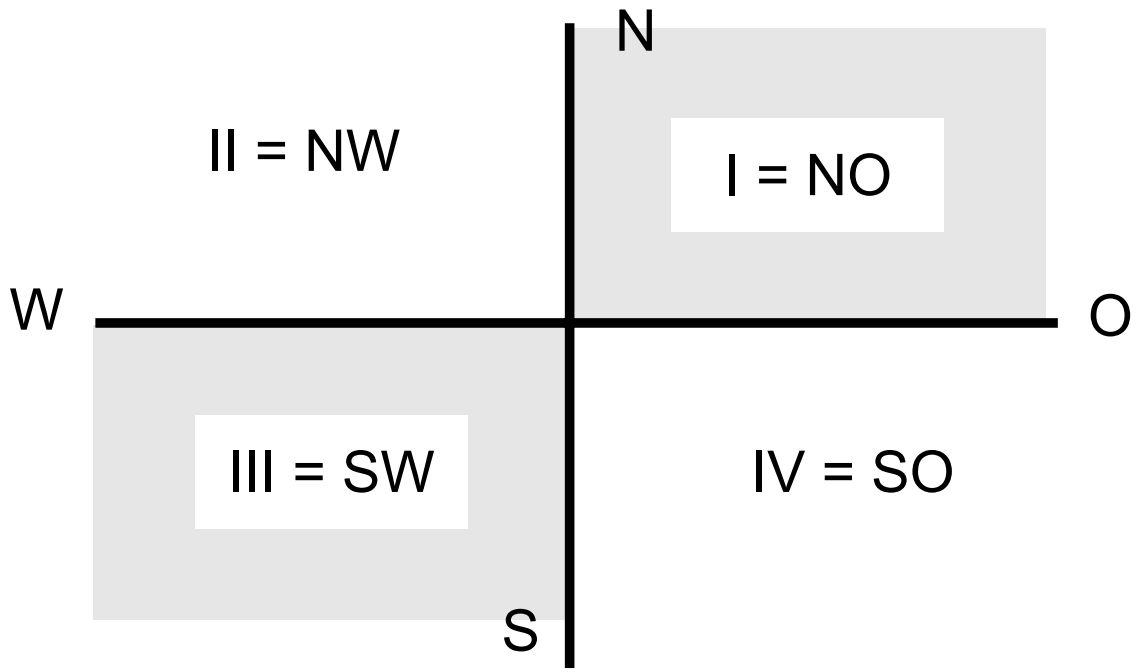
  rekursive Partitionierung durch Quadranten

- **Realisierung als Generalisierung des Binärbaumes:**

  - jeder Knoten enthält einen Satz
  - Grad eines Knotens: max. 4
  - Wurzel teilt 2-dim. Raum in 4 Quadranten auf
  - rekursive Aufteilung jedes Quadranten durch Wurzel eines Unterbaumes
  - i-ter Unterbaum eines Knotens enthält Punkte im i-ten Quadranten

- **Bsp.: geographische Daten: x, y - Koordinaten**

  Aufteilung:



- **Verallgemeinerung:**

  k-dimensionaler Schlüssel $\Rightarrow$ Grad jedes Knotens: 2^k

  k=2: [point] quad-tree k=3: oct-tree k=4: hex-tree

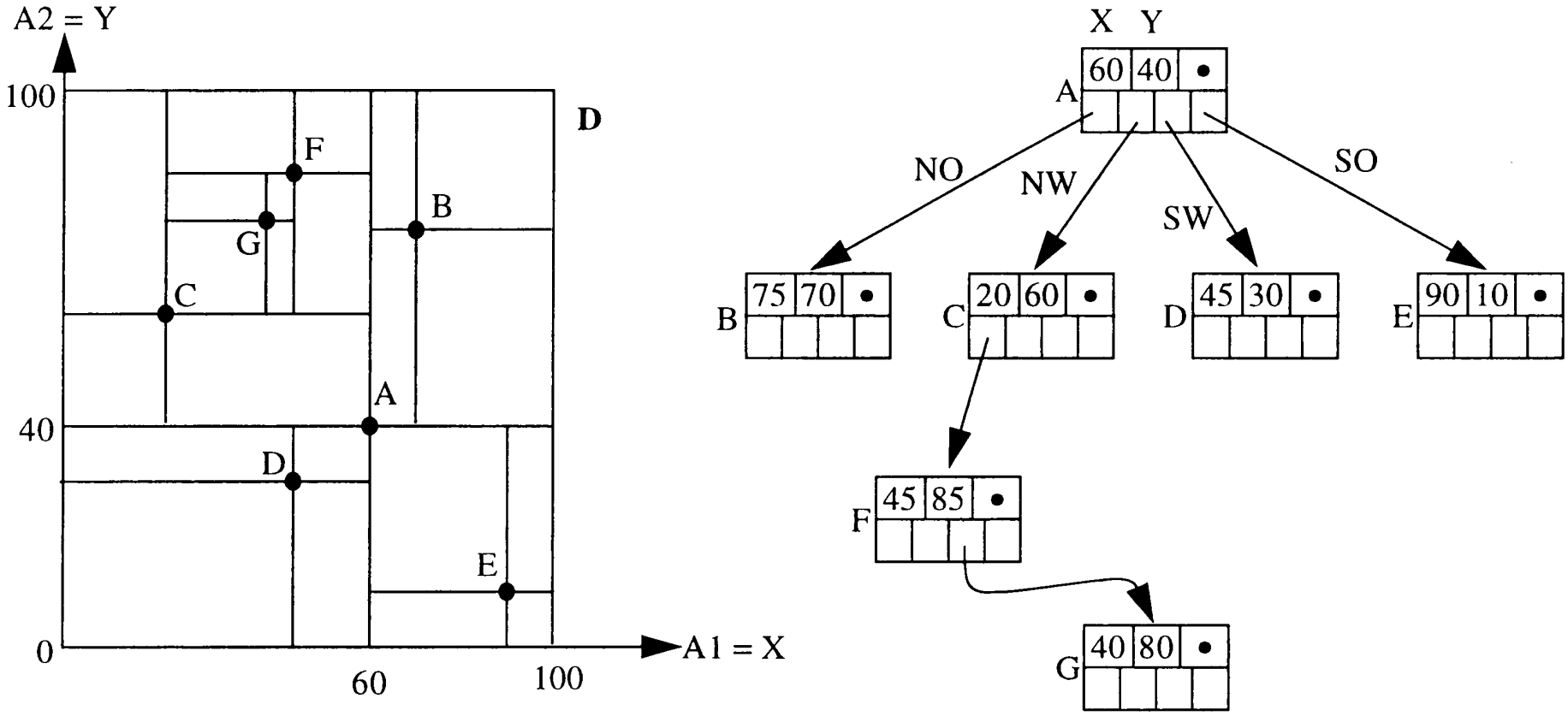


Abb. 9.5: Zerlegungsprinzip für D und Struktur des zugehörigen Quadranten-Baums

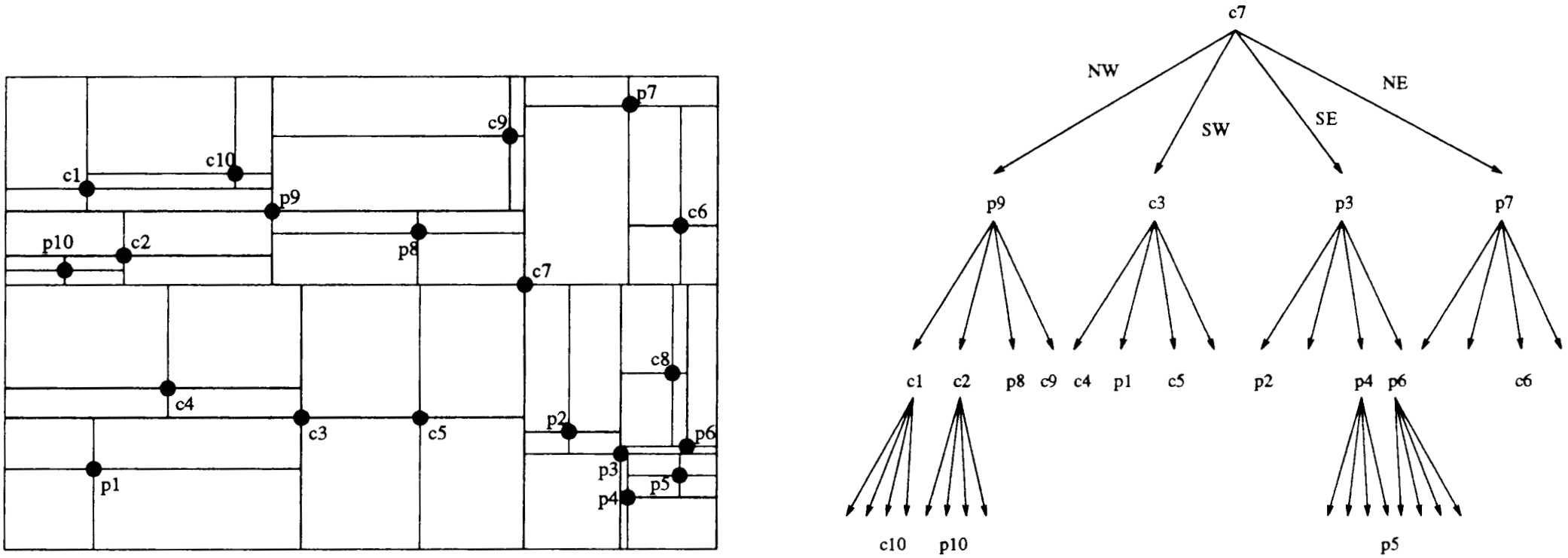
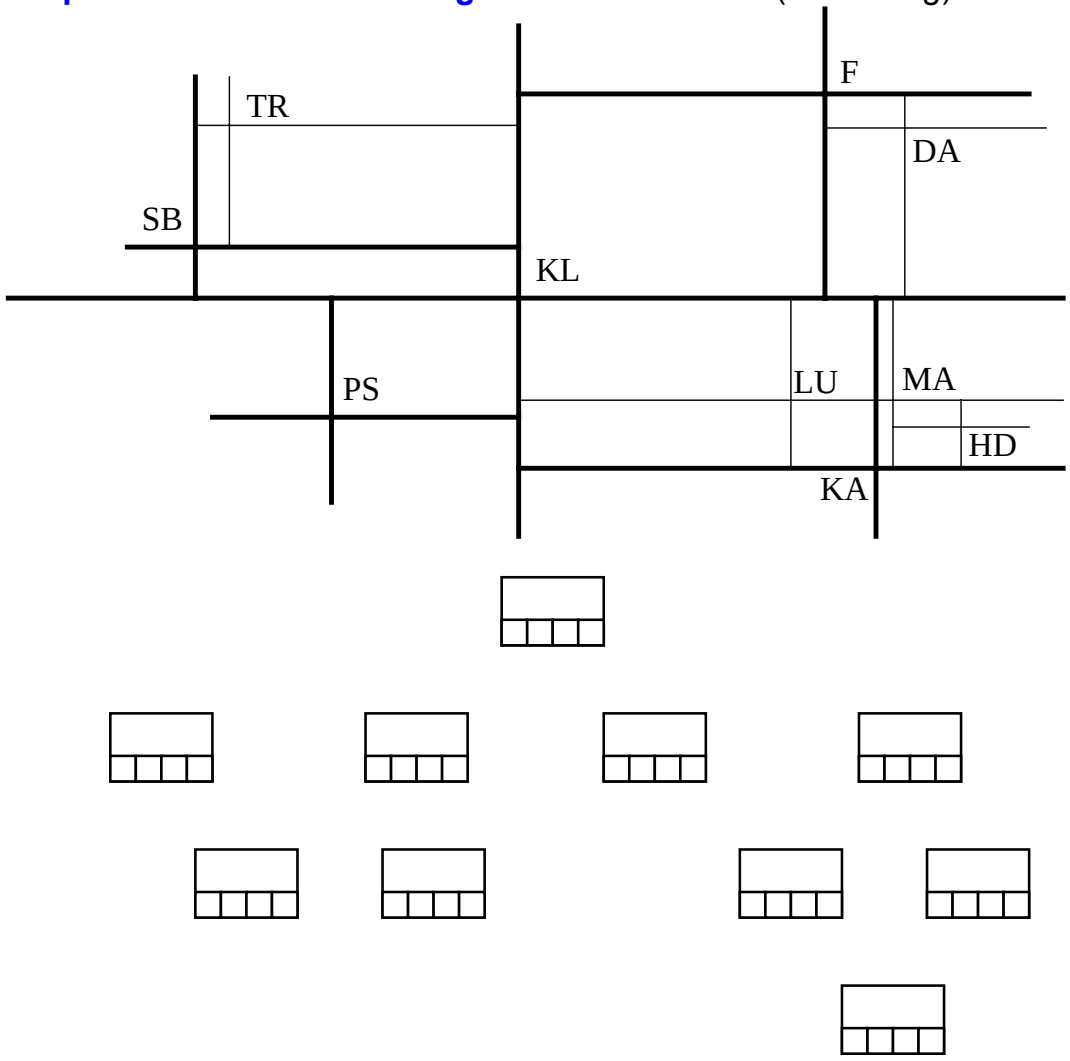


Figure 14. Point quadtree.

Quad-Tree (Forts.)

- **Beispiel: räumliche Aufteilung --> Baumstruktur** (zur Übung)



- **Eigenschaften:** Suchen nur bei exakter Anfrage einfach, sonst komplex
  - Baumstruktur abhängig von Einfügereihenfolge (unbalanciert)
  - Löschen von Zwischenknoten schwierig (Neueinfügen der Unterbäume)
  - keine Erhaltung der Topologie - nur interne Datenstruktur

Mehrdimensionale binäre Suchbäume (k-d-Bäume)

- **andere Erweiterung des Binärbaumes**
  - Berücksichtigung von k Schlüsseln bzw. Dimensionen: k-d-Baum
  - alle Datensätze werden mit Hilfe der Baumstruktur organisiert: knotenorientiert (homogen)
  - Wartungsoperationen wie beim binären Suchbaum, aber
  - **auf jeder Ebene erfolgt Schlüsselvergleich für einen der k Schlüssel**

- **Diskriminator legt den Schlüssel auf jeder Ebene fest**
  - zyklische Variation des Diskriminators d:
    für alle Knoten der Baumebene i gilt: $d = (i \bmod k) + 1$
  - der linke (rechte) Nachfolger zu einem Knoten enthält alle Sätze mit kleineren (größeren) Werten für das Diskriminatorattribut

    $\forall\, Q \in \text{LEFT}(P): \quad A_d(Q) \leq A_d(P)$

    $\forall\, R \in \text{RIGHT}(P): \quad A_d(R) > A_d(P)$

- **Eigenschaften:**
  - Baumstruktur abhängig von Einfügereihenfolge (unbalancierter Baum)
  - Eingrenzung des Suchraumes für Partial-Match- und Bereichsanfragen komplex
  - Löschen ist sehr schwierig
  - nur interne Datenstruktur

- **Weitere Variante des k-d-Baumes**
  - blattorientiert (heterogen, Speicherung der Sätze in Buckets = Blätter)

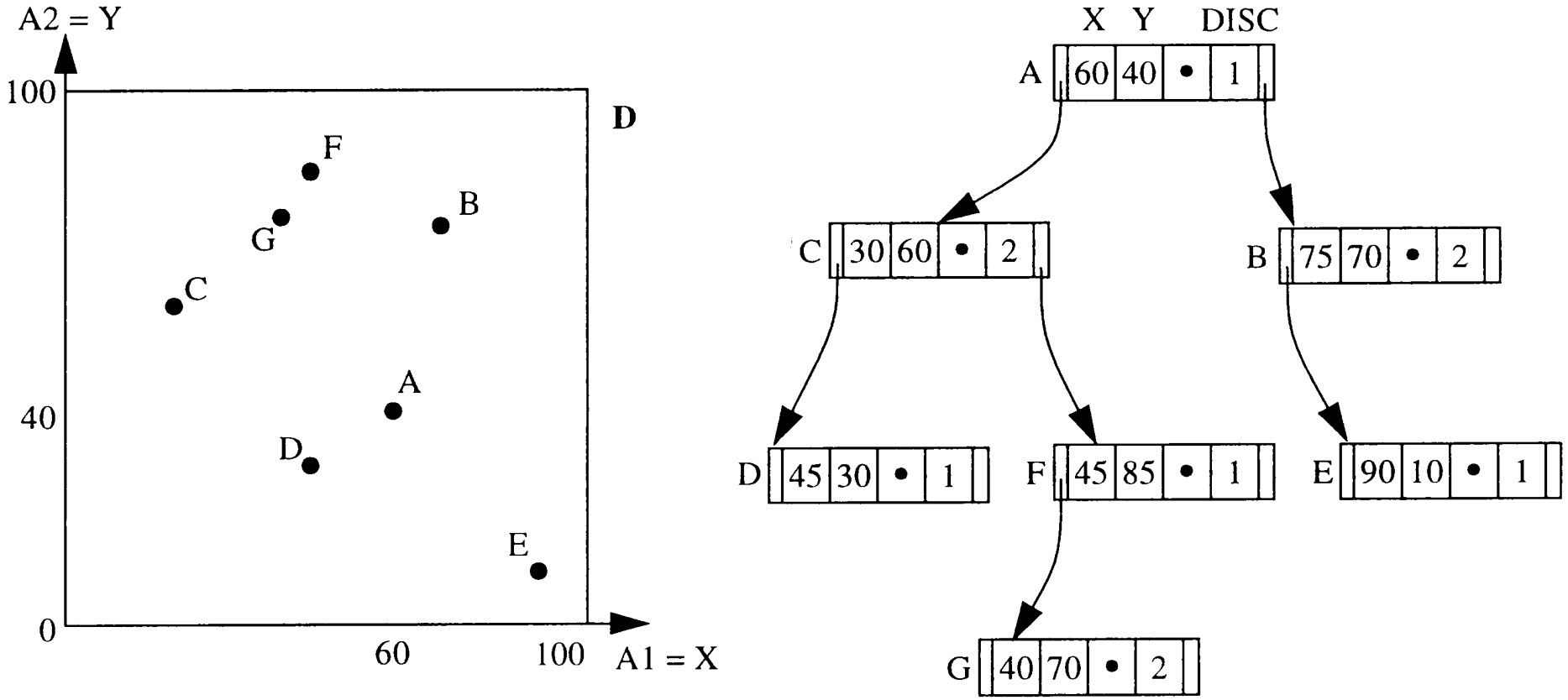


Abb. 9.7: Organisation der Datensätze beim homogenen k-d-Baum

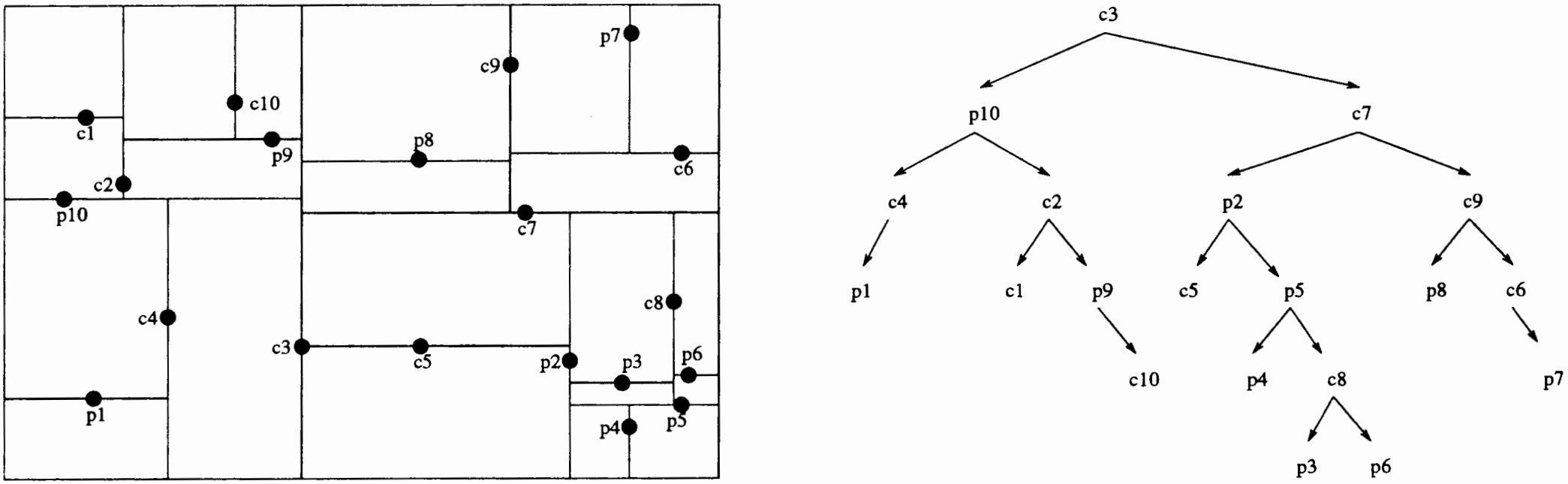


Figure 10. k-d-tree.

Beispiel: Tabelle --> 3-d-Baum (Übung)

Alter	Gehalt	Ort
35	17K	KL
28	40K	F
29	15K	DA
25	45K	KL
40	12K	SB
29	16K	DA
30	17K	F
42	100K	F
29	14K	DA