UMIT
the health & life sciences university

Vorlesung 7:
C++-Programmierung mit Standardklassen

- Referenzen, Funktionen, Parameterübergabe
- Vektor-Container
- Übungsbeispiel: k-Means-Algorithmus

UMIT
the health & life sciences university

Call by Reference

(Wiederholung)

Alternativ zur Übergabe von Zeigern als Parameter kann in C++ ein *Call by Reference* erzielt werden, indem ein Funktionsargument durch Voranstellen von & als *Referenz* gekennzeichnet wird *(Beispiel aus Block 6)*:

```
void orthodrome (double phi_dep, double lam_dep,
                 double phi_arr, double lam_arr,
                 double &ll, double &azi_dep, double &azi_arr)
{ ... }
```

und in `main()`:

```
orthodrome (phi1, lam1, phi2, lam2, pathangle, azi1, azi2);
```

Call by Reference

(Wiederholung)

- Auch bei der Übergabe als Referenzparameter wird nicht der *Wert* der Parametervariablen, sondern ihre *Speicheradresse* übergeben.
- Analog zur Übergabe mittels Zeiger ist es auch hier nicht möglich, Literale (wie `3`) oder zusammengesetzte Ausdrücke (wie `a+b`) als Referenzparameter zu übergeben.
- **Vorteil gegenüber Zeigern:** Der Adressoperator beim Aufruf der Funktion entfällt. Damit muss der/die Programmierer/in beim Aufruf der Funktion sich in den meisten Fällen keine Gedanken darüber machen, ob die einzelnen Parameter als Werte oder Referenzen übergeben werden!
- Im Unterschied zu Zeigern müssen Referenzen aber stets auf gültige Variablen verweisen, es gibt keine „Null-Referenz“!
- Referenzparameter können auch *read-only* übergeben werden:

```
void print (const string &s);
```

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode verstehen

→`0701-vectors.cpp`

- Legen Sie ein neues **C++**-Konsolenprojekt mit dem Quelltext aus der (in Moodle bereit gestellten) Datei `0701-vectors.cpp` an
- Bringen Sie dieses Programm zum Laufen
- Erarbeiten Sie sich anhand dieses Programms die darin vorkommenden neuen C++-Sprachelemente
- Achten Sie auch auf die Include-Dateien

Vektor-Container

- Vektor-Container sind von der Standardbibliothek bereitgestellte Klassen für Felder veränderlicher Größe.

  Sie eignen sich damit als komfortable Alternative zu herkömmlichen Feldern und auch zu dynamisch allozierten Feldern.

- Zur Benutzung muss der Header `vector` eingebunden werden.

- Vektor-Container können mit einem frei wählbaren Datentyp der Elemente eingerichtet werden, dieser Datentyp wird als ein Parameter des Containertyps in spitzen Klammern übergeben:

```
vector <int> data;    // deklariert Vektor von Ganzzahlen
vector <string> text; // deklariert Vektor von Zeichenketten
vector <vector <unsigned char>> image; // zweidim. Pixelfeld
```

Anmerkung zum letzten Beispiel: Wenn der C++11-Standard im Compiler nicht aktiviert ist, erzeugen die beiden schließenden spitzen Klammern für die beiden ineinander verschachtelten Vektor-Container im letzten Beispiel einen Compilerfehler. Vor C++11 musste an dieser Stelle ein Leerzeichen eingefügt werden.

UMIT
the health & life sciences university

Vektor-Container

Methoden für Vektor-Container (Auswahl)

- *Vektor*`[`*Index*`]` Indexzugriff ohne Bereichsprüfung,
- *Vektor*`.at (`*Index*`)` Indexzugriff mit Bereichsprüfung
- *Vektor*`.size()` aktuelle Größe des Vektors
- *Vektor*`.push_back (`*Element*`)` Anfügen eines Elements am Ende
- *Vektor*`.pop_back()` Löschen des letzten Elements
- *Vektor*`.clear()` Löscht alle Elemente
- *Vektor*`.empty()` gibt `true` zurück, wenn der Vektor keine Elemente hat, sonst `false`

UMIT
the health & life sciences university

Vektor-Container

Methoden für Vektor-Container (Fortsetzung)

- *Vektor*`.insert (`*Vektor*`.begin()+i,` *Element*`)` Einfügen des Elements an der Stelle mit Index `i`
- *Vektor*`.insert (`*Vektor*`.begin()+i, k,` *Element*`)` Einfügen von `k` Kopien des Elements an der Stelle mit Index `i`
- *Vektor*`.insert (`*Vektor*`.end()-i,` *Element*`)` Einfügen des Elements `i` Stellen vor Ende
- *Vektor*`.erase (`*Vektor*`.begin()+i)` Löschen des Elements an Stelle `i`
- *Vektor*`.erase (`*Vektor*`.begin()+i,` *Vektor*`.begin()+j)` Löschen von Index `i` (einschließlich) bis `j` (ausschließlich)

Dabei sind *Vektor*`.begin()` und *Vektor*`.end()` sogenannte *Iteratoren*. Das sind mit der Vektorklasse verbundene zeigerartige Hilfsdatentypen. Weitere Details sind Literatur oder Online-Referenzen zu entnehmen.

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode verstehen

→`0702-images-vec`

- Legen Sie ein neues **C++**-Konsolenprojekt mit den Quelltext- und Headerdateien aus dem (in Moodle bereit gestellten) Verzeichnis `0702-images-vec` an
- Bringen Sie dieses Programm zum Laufen
- Die Funktionalität des Programms entspricht der des früheren Beispiels `0504-images-struct`, jedoch wurde jetzt statt eines selbst definierten Verbunddatentyps für Bilder eine Darstellung mittels `vector`-Containern der C++-Standardbibliothek verwendet
- Vollziehen Sie den Aufbau und die Funktionsweise des Programms nach und erarbeiten Sie sich anhand dieses Programms die darin vorkommenden weiteren C++-Sprachelemente
- Achten Sie auch wieder auf die Include-Dateien

Dieses Projekt benötigt C++11, daher nötigenfalls die entsprechende Compileroption aktivieren!

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode schreiben

→`0703-bblsort-s` (nach der Vorlesung)

Erstellen Sie basierend auf den Bubblesort-Implementationen in C aus früheren Blöcken, zB `0403-bblsort.c`, ein Programm, das

- mehrere Zeichenketten (zB Namen) als Usereingabe abfragt (Beenden zB mittels Leerzeile),
- diese mit Bubblesort sortiert und
- in der sortierten Reihenfolge ausgibt.

Die Zeichenketten können in einem `vector<string>` gespeichert werden.

k-Means-Algorithmus (Lloyd-Verfahren)

Der *k-Means-Algorithmus* ist ein einfaches Verfahren zur Clusteranalyse (Algorithmus: nächste Folie). Ziel des Verfahrens ist es, eine Menge gegebener Datenpunkte im $\mathbb{R}^m$ (hier: in der Ebene $\mathbb{R}^2$) derart in Gruppen (Cluster – „Klumpen“) zu unterteilen, dass die Summe der Abstände der Datenpunkte zu den Mitten (Schwerpunkten) der jeweiligen Gruppen so klein wie möglich wird.

k-Means-Algorithmus (Lloyd-Verfahren)

(nach https://de.wikipedia.org/wiki/K-Means-Algorithmus)

Gegeben: Datenpunkte $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1} \in \mathbb{R}^m$; Anzahl k der zu bildenden Cluster

Gesucht: Zuordnung der Datenpunkte zu k Clustern; Mittelpunkte $\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{k-1} \in \mathbb{R}^m$ der Cluster

1. *Initialisierung:* Initialisiere die Clusterzentren $\boldsymbol{m}_0, \ldots, \boldsymbol{m}_{k-1} \in \mathbb{R}^m$ mit k zufällig gewählten Datenpunkten $\boldsymbol{x}_i$

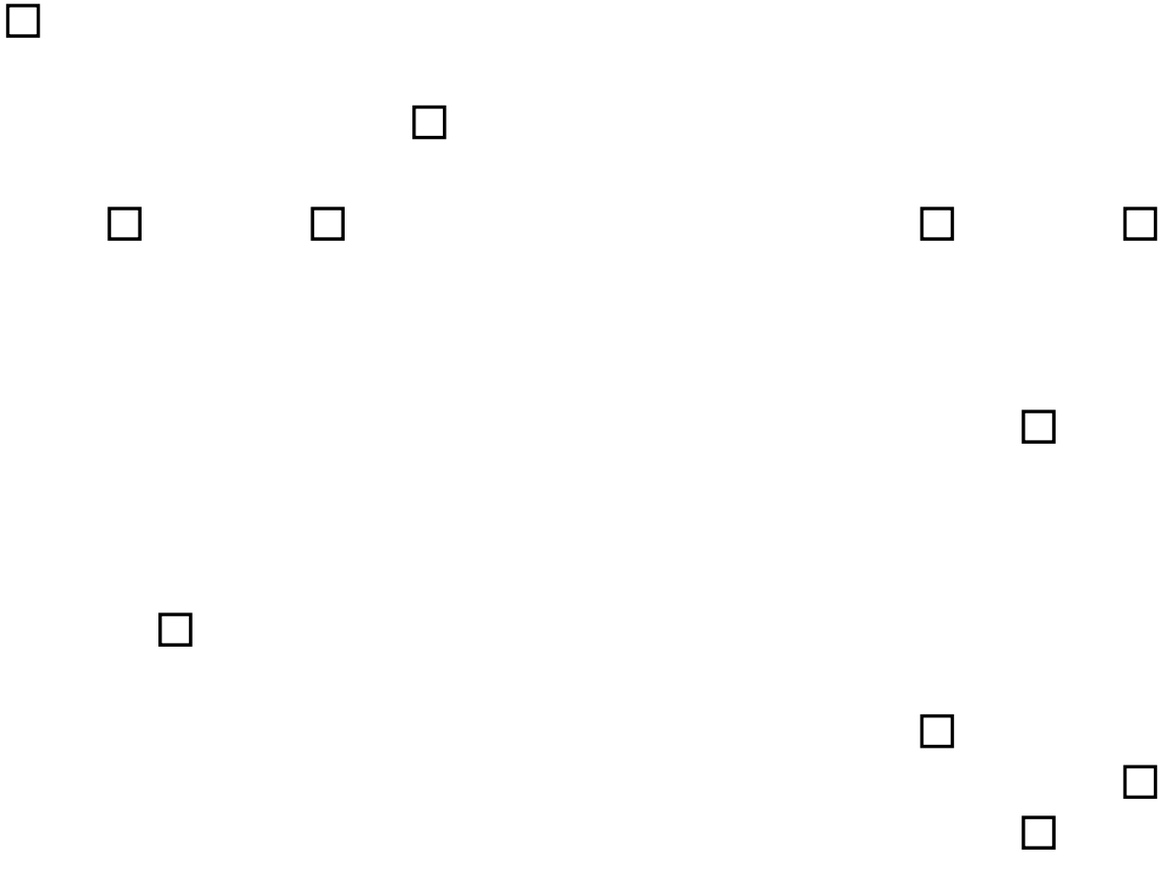
2. *Zuordnung der Datenpunkte:* Ordne jeden Datenpunkt dem nächstliegenden Clusterzentrum zu

3. *Aktualisieren der Clusterzentren:* Berechne jedes $\boldsymbol{m}_j$ neu als Mittelwert der diesem Clusterzentrum zugeordneten Datenpunkte

4. *Abbruchbedingung (ab 2. Durchlauf):* Wenn in Schritt 2 die Zuordnung aller Datenpunkte gleich wie im vorangegangenen Durchlauf geblieben ist, Ende; sonst wiederhole ab Schritt 2.

UMIT
the health & life sciences university

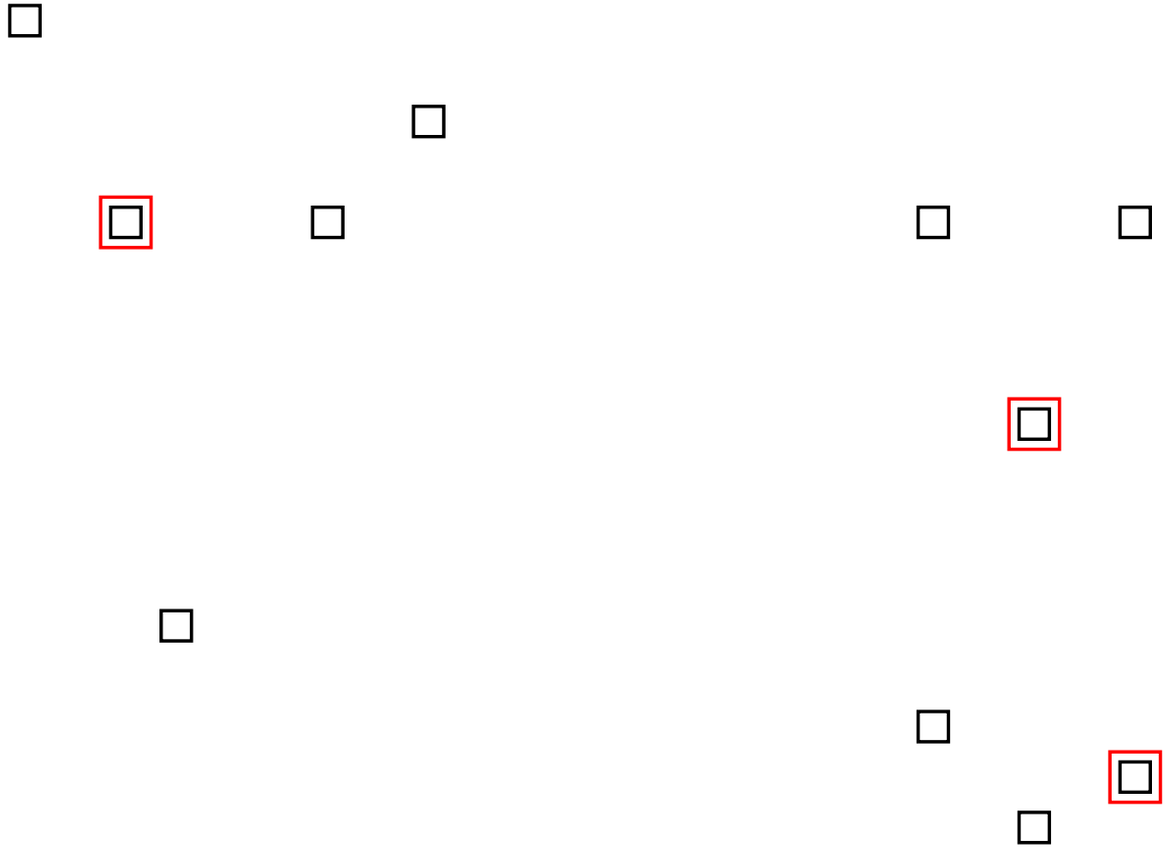
k-Means-Algorithmus (Lloyd-Verfahren), Beispiele



Datenpunkte

UMIT
the health & life sciences university

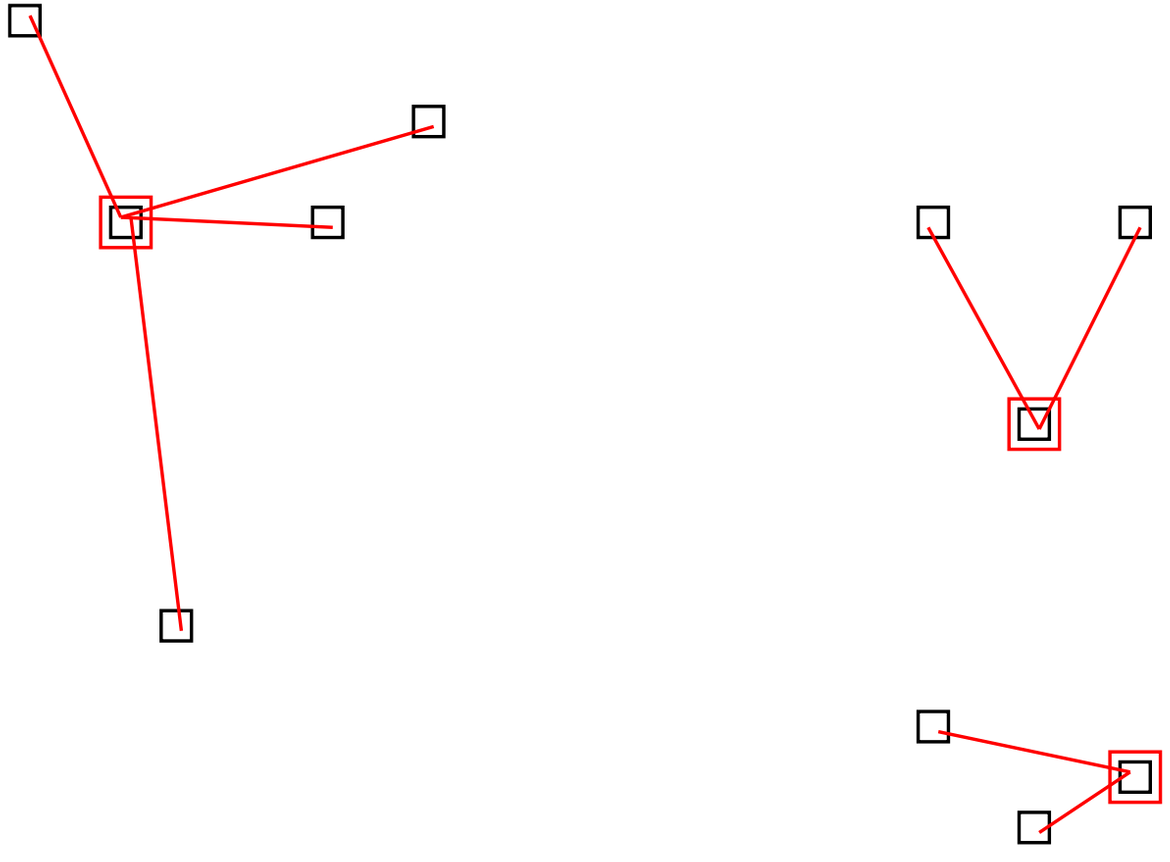
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 1



Initialisierung ($k = 3$)

UMIT
the health & life sciences university

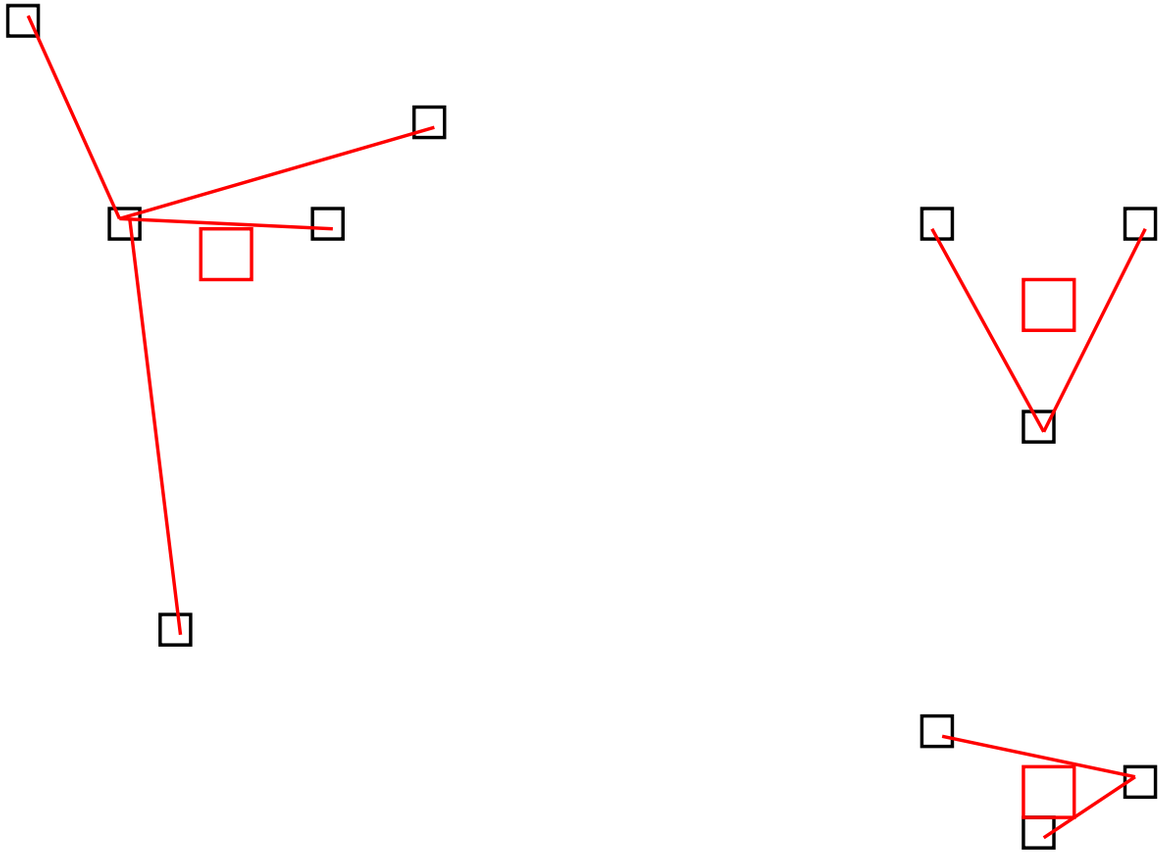
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 1



Datenpunkte den Clusterzentren zugeordnet

UMIT
the health & life sciences university

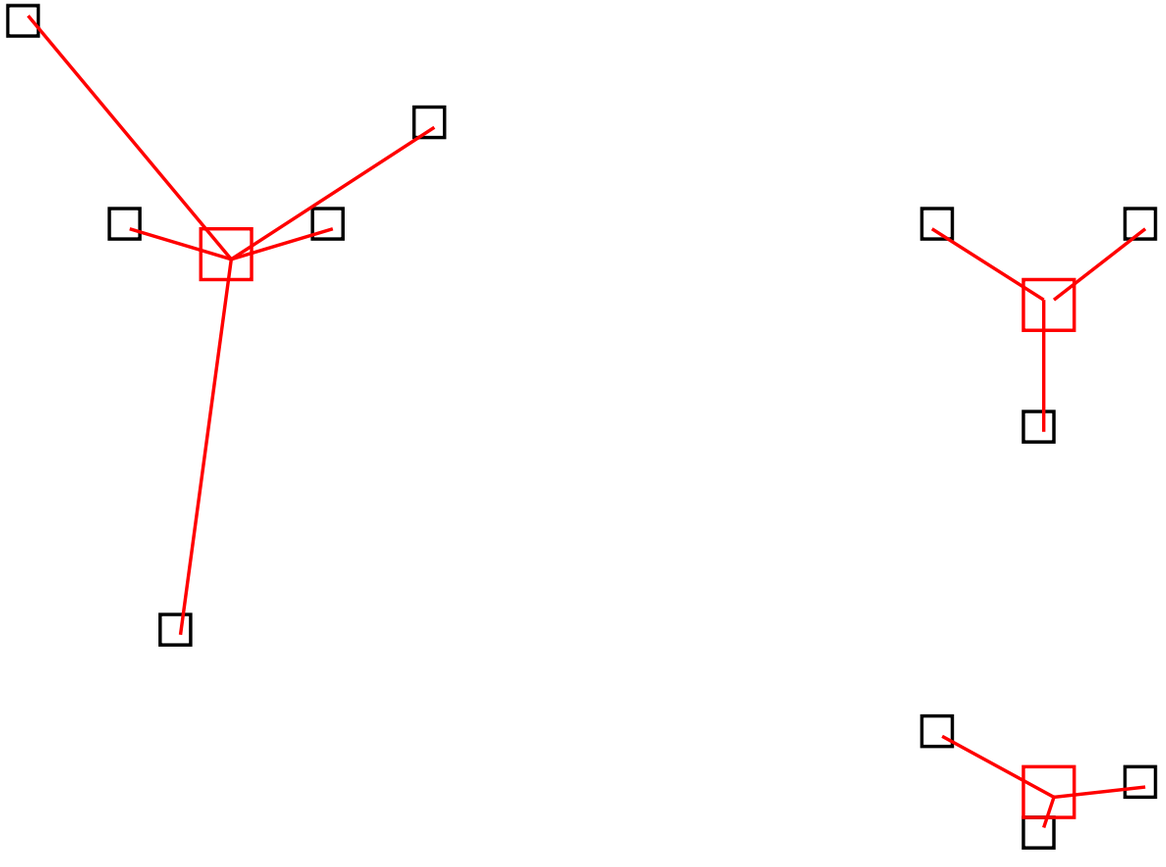
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 1



Clusterzentren neu berechnet

UMIT
the health & life sciences university

k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 1



Datenpunkte den Clusterzentren erneut zugeordnet
Zuordnung unverändert → Endergebnis

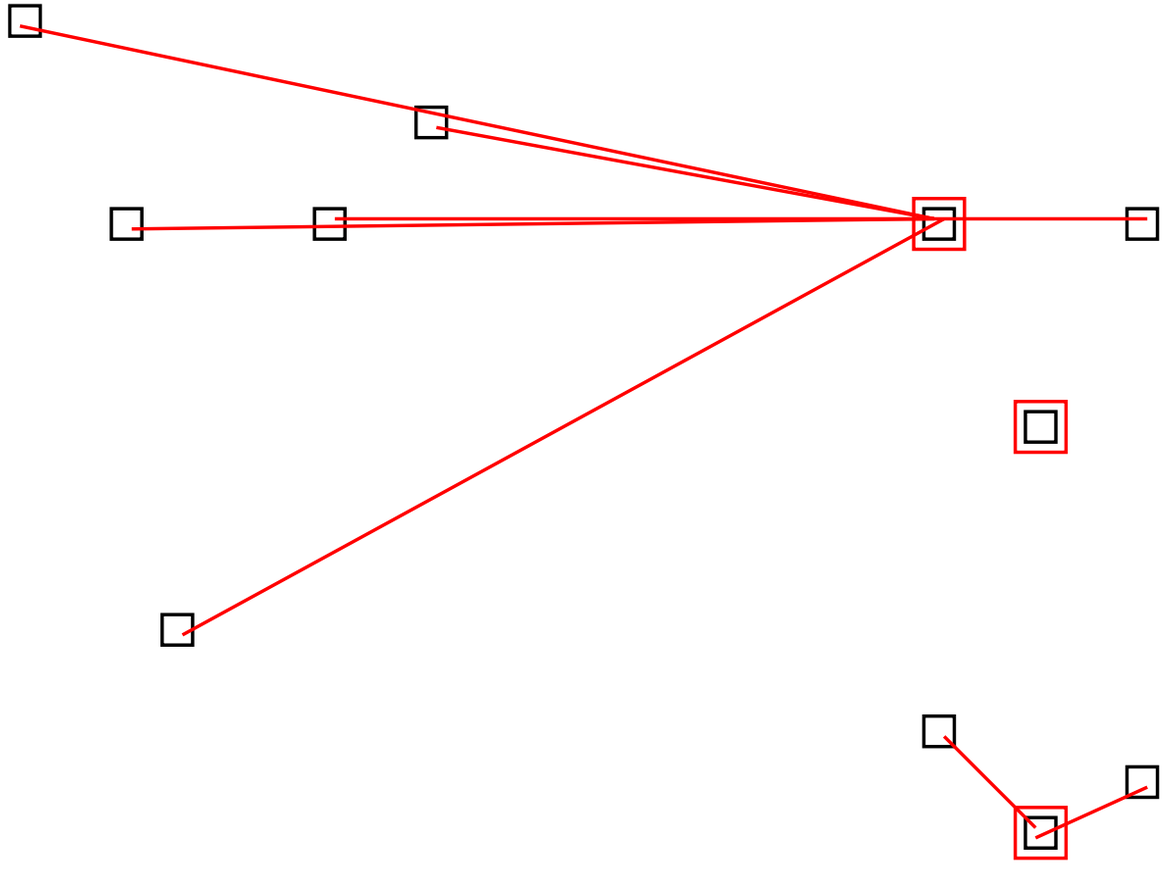
UMIT
the health & life sciences university

k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 2

Initialisierung ($k = 3$)

UMIT
the health & life sciences university

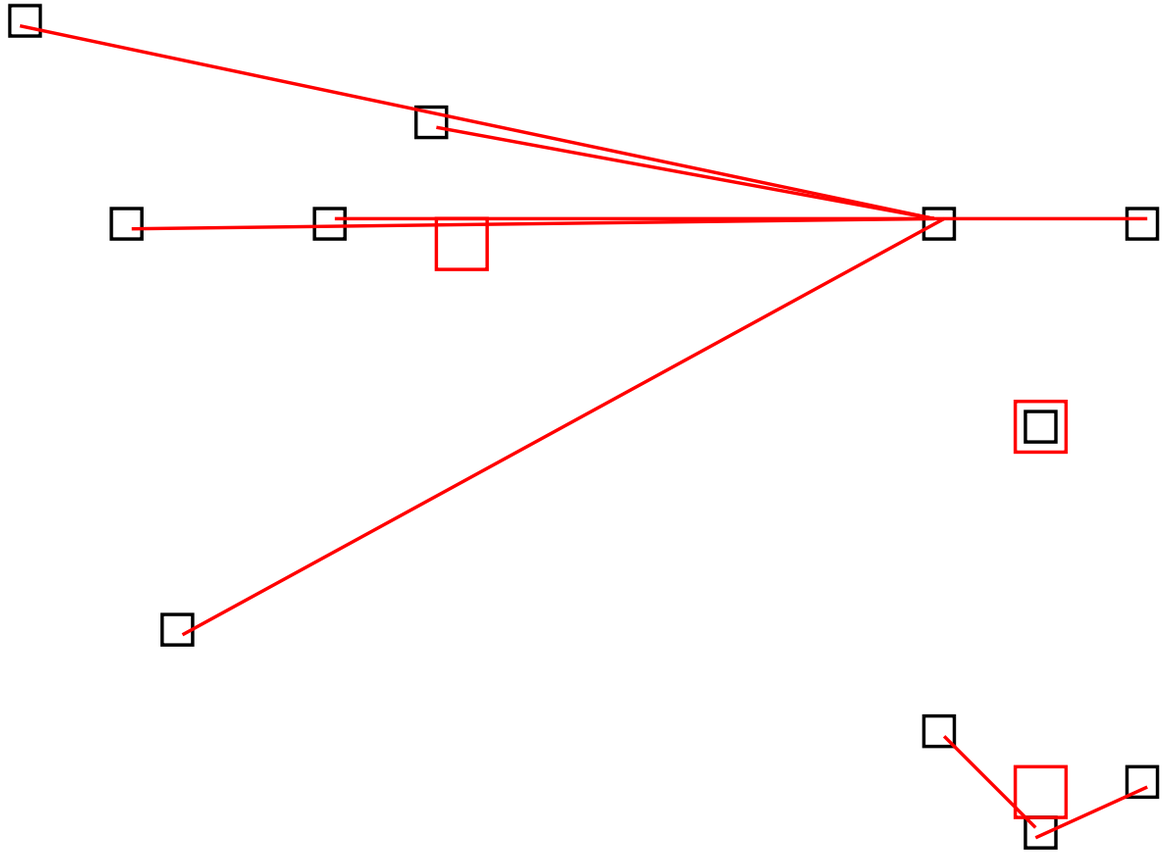
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 2



Datenpunkte den Clusterzentren zugeordnet

UMIT
the health & life sciences university

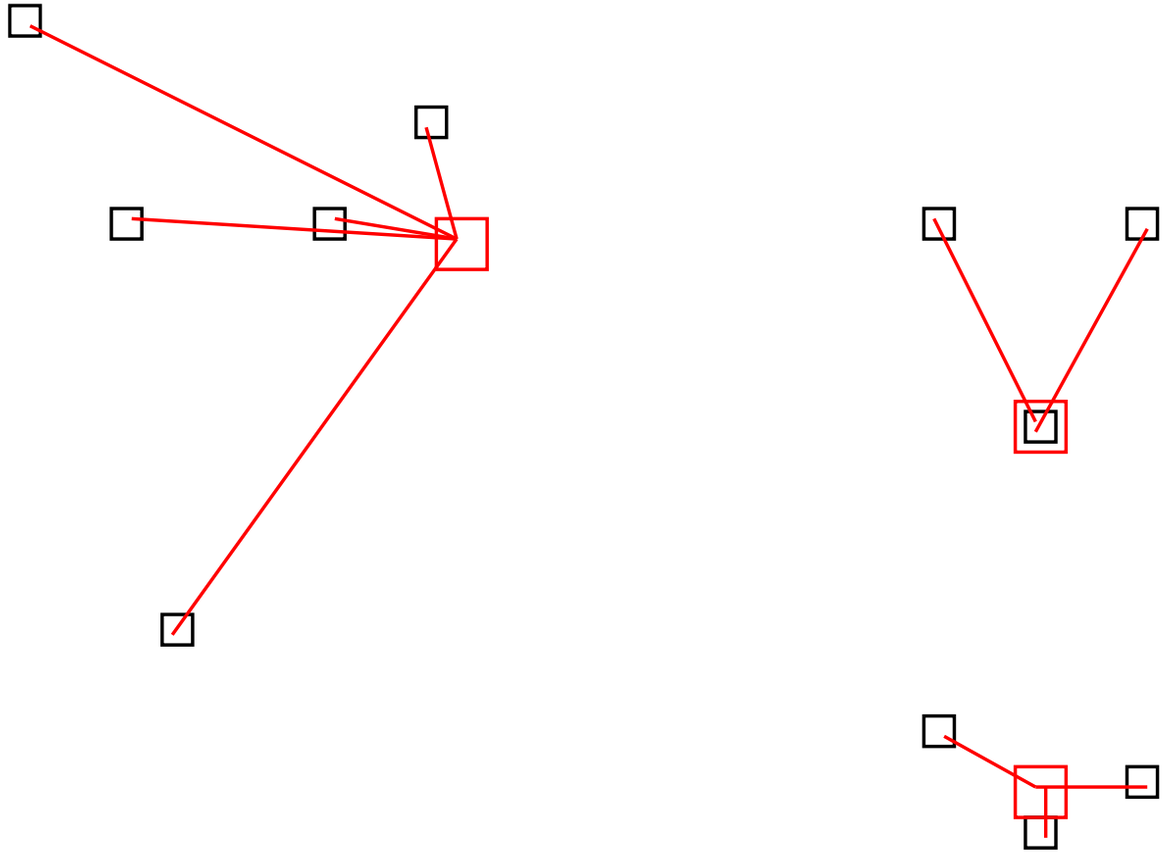
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 2



Clusterzentren neu berechnet

UMIT
the health & life sciences university

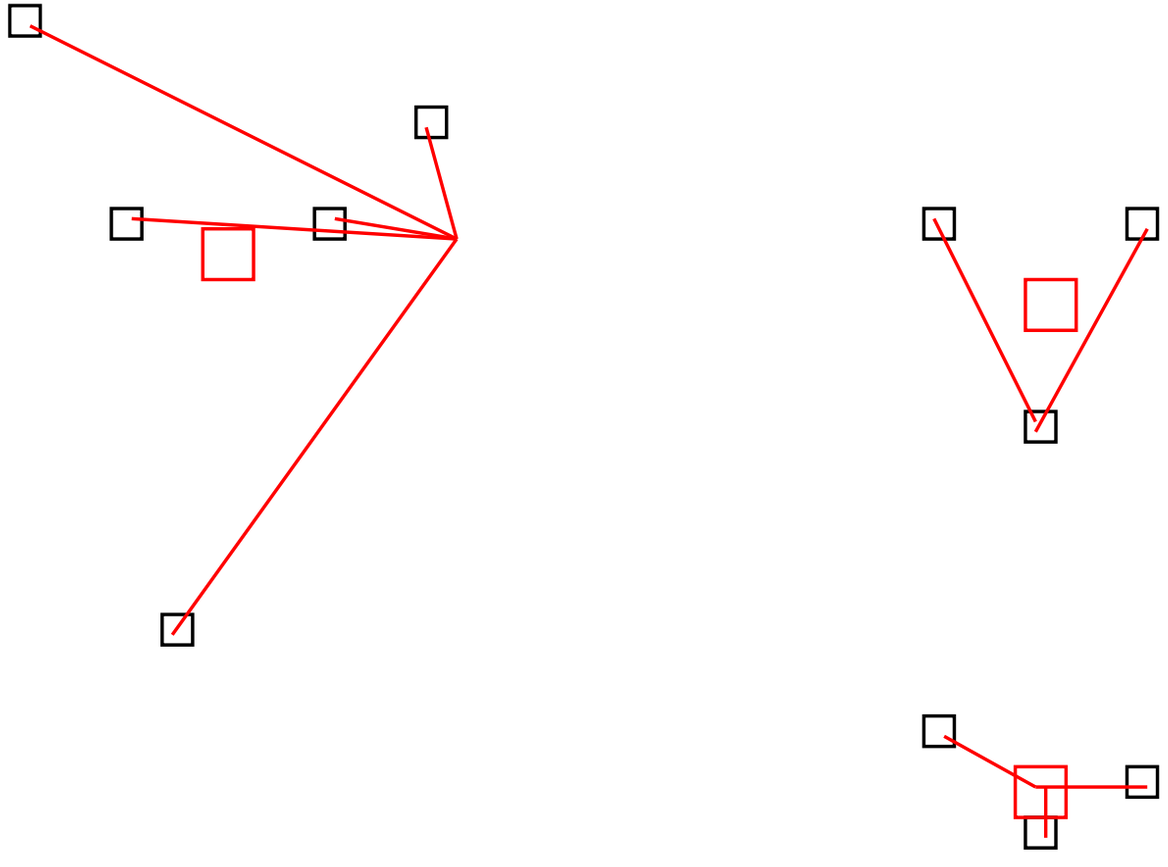
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 2



Datenpunkte den Clusterzentren erneut zugeordnet

UMIT
the health & life sciences university

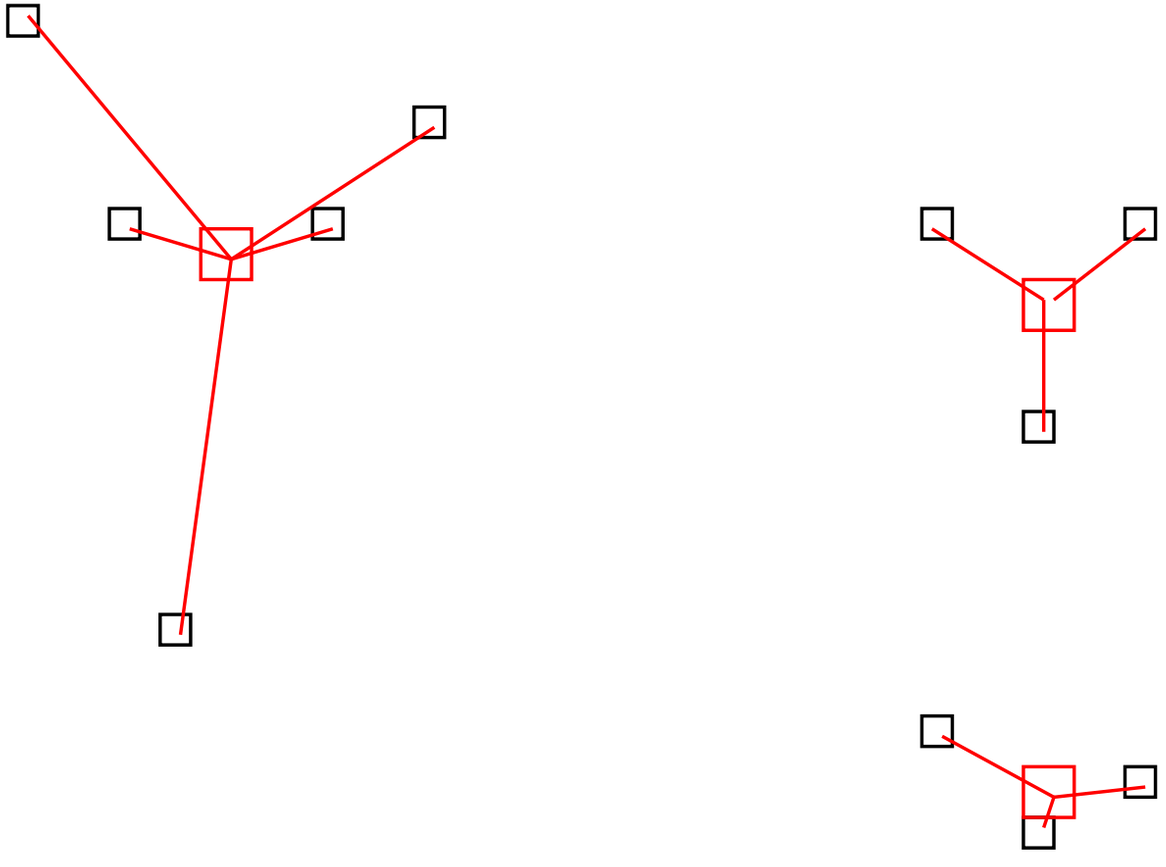
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 2



Clusterzentren neu berechnet

UMIT
the health & life sciences university

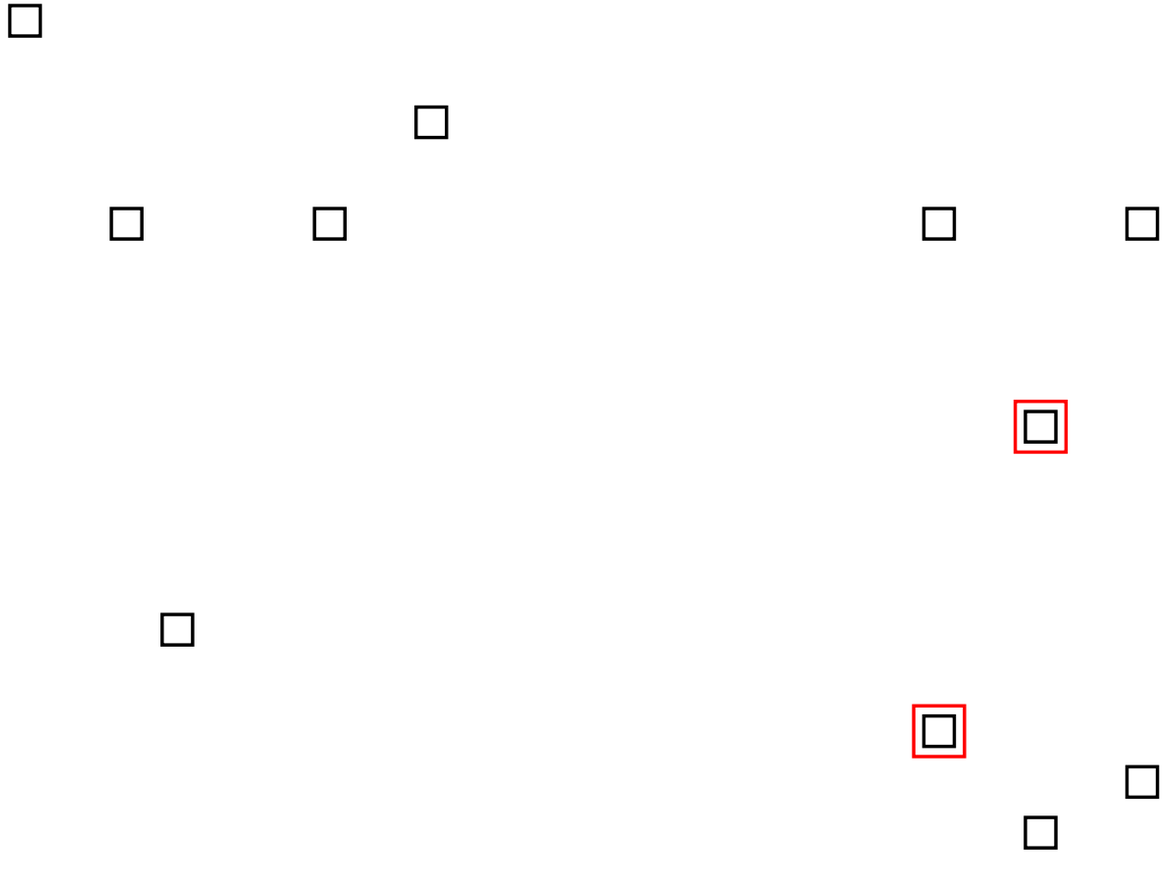
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 2



Datenpunkte den Clusterzentren erneut zugeordnet
Zuordnung unverändert → Endergebnis

UMIT
the health & life sciences university

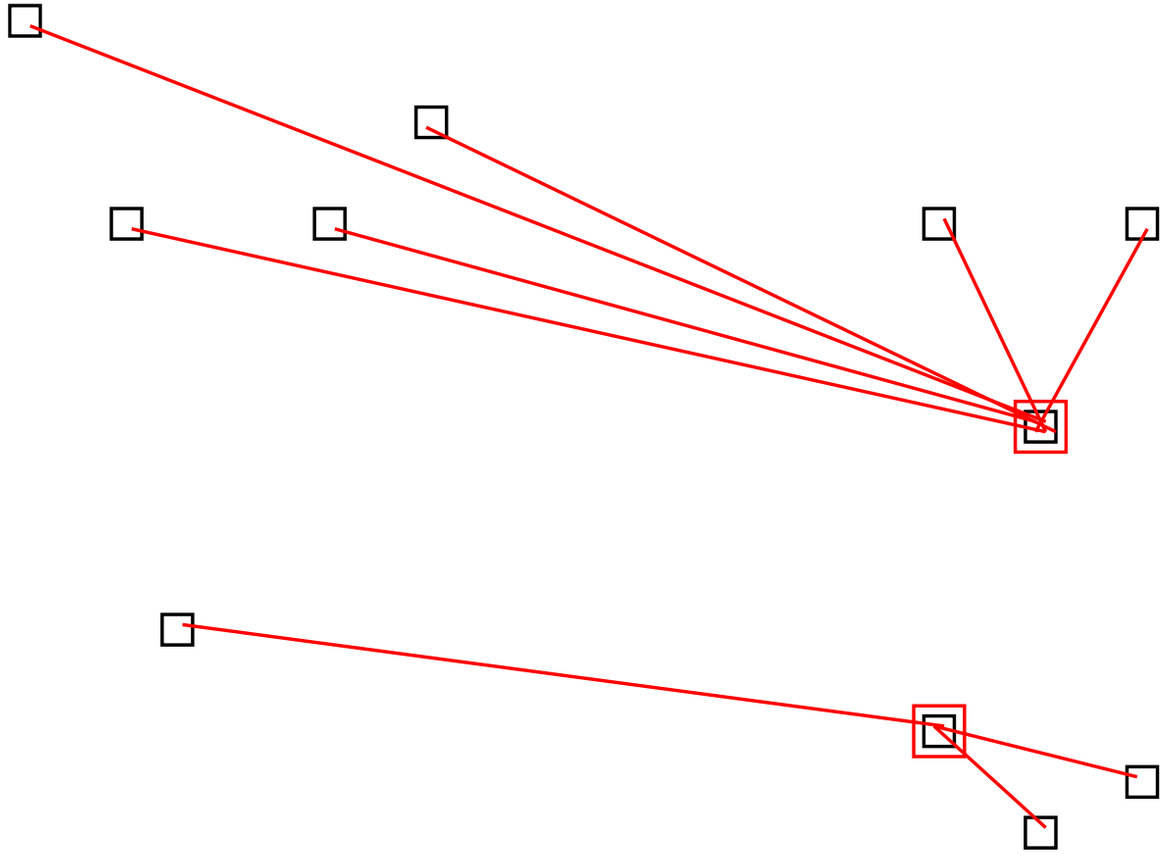
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3



Initialisierung ($k = 2$)

UMIT
the health & life sciences university

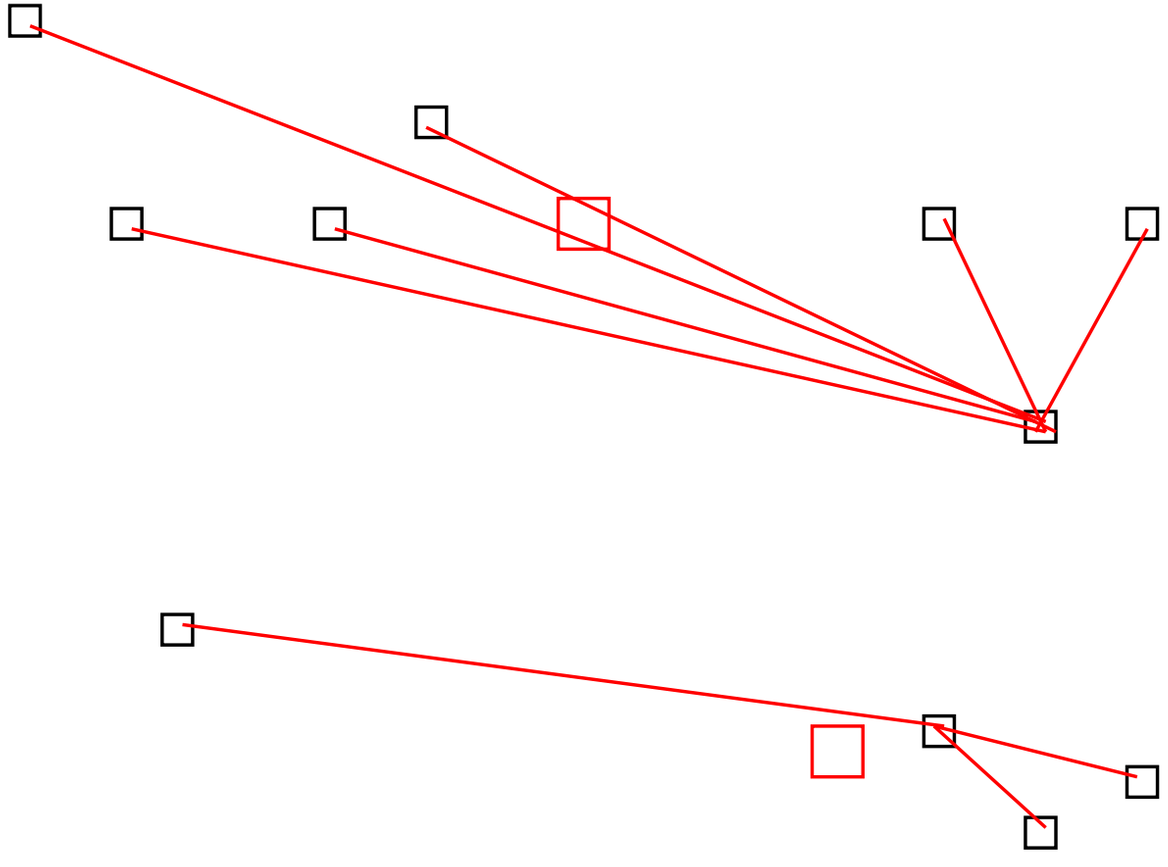
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3



Datenpunkte den Clusterzentren zugeordnet

UMIT
the health & life sciences university

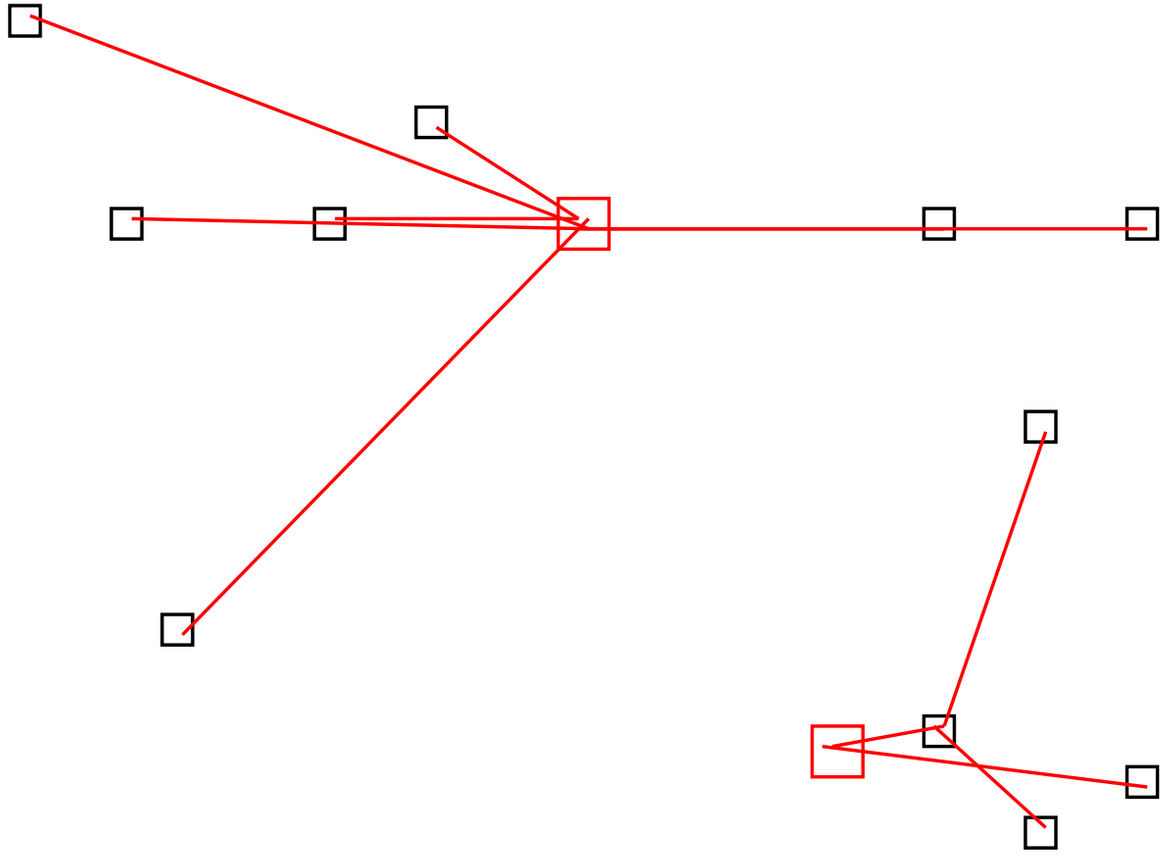
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3



Clusterzentren neu berechnet

UMIT
the health & life sciences university

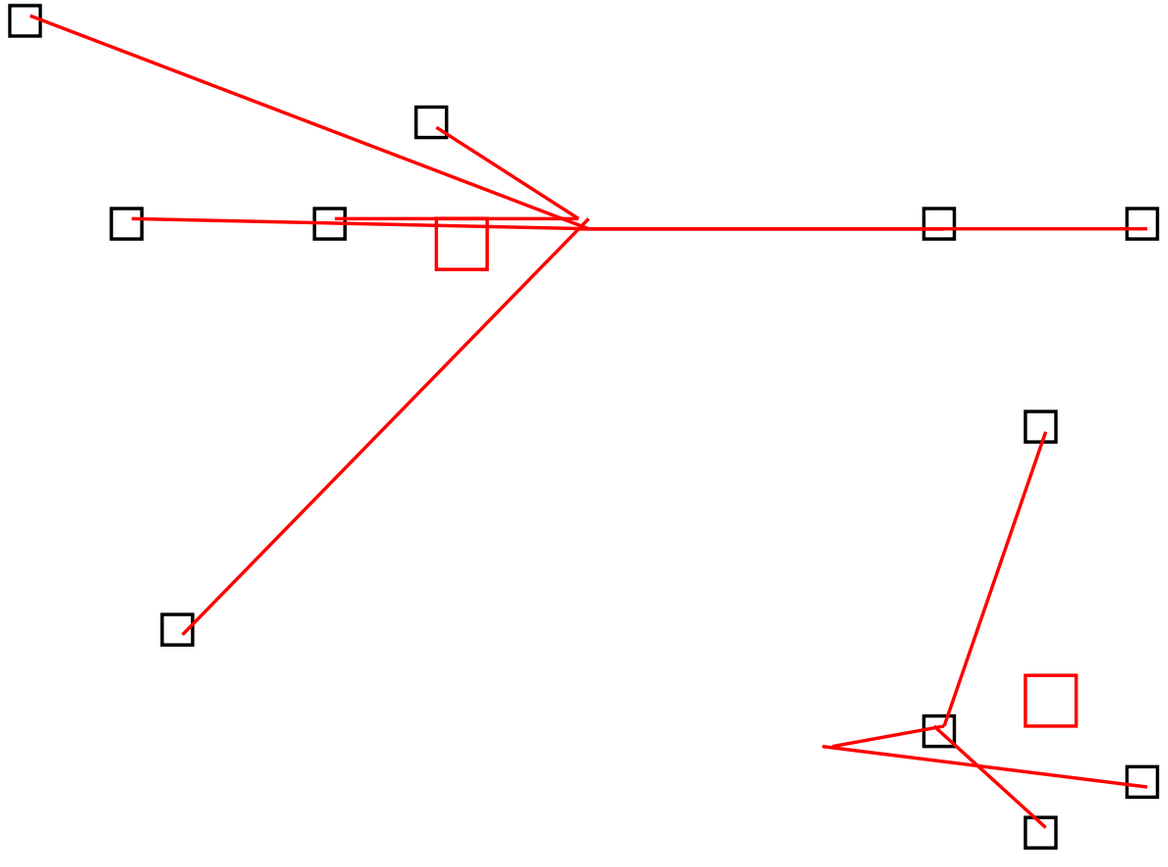
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3



Datenpunkte den Clusterzentren erneut zugeordnet

UMIT
the health & life sciences university

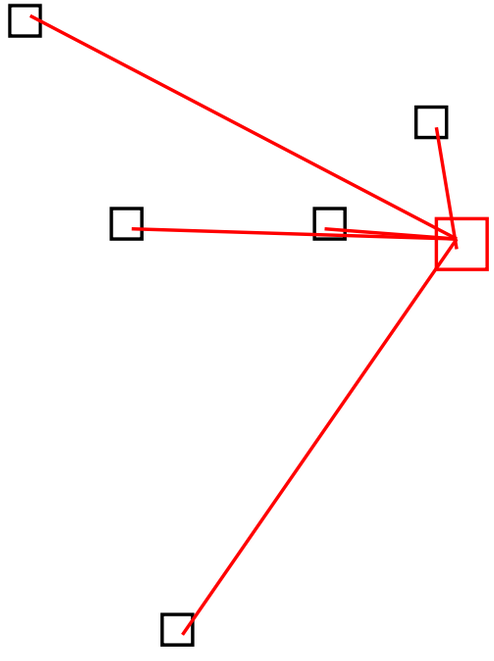
k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3

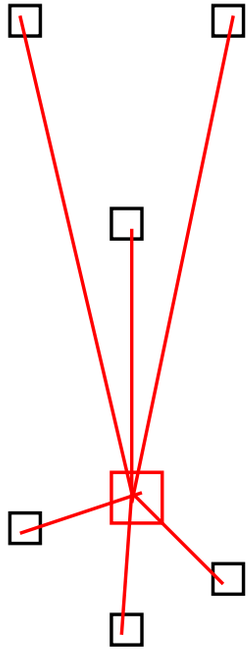


Clusterzentren neu berechnet

UMIT
the health & life sciences university

k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3

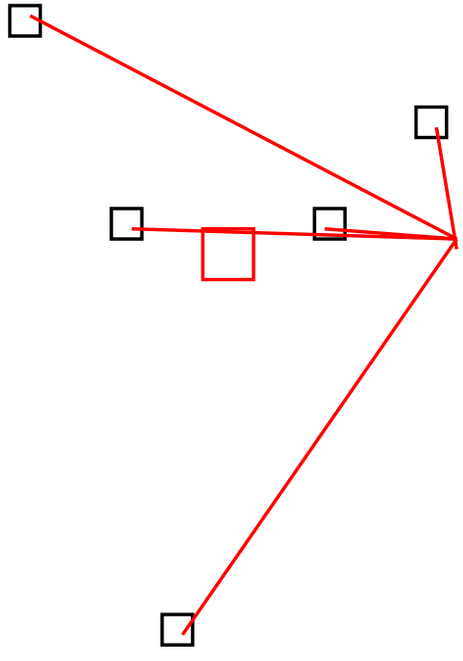


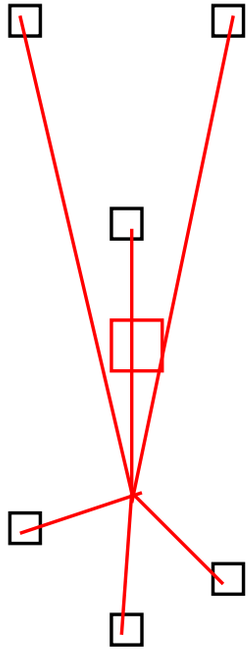


Datenpunkte den Clusterzentren erneut zugeordnet

UMIT
the health & life sciences university

k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3

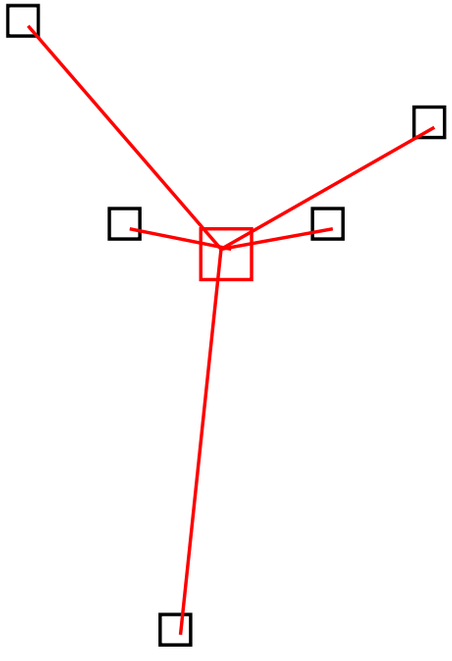


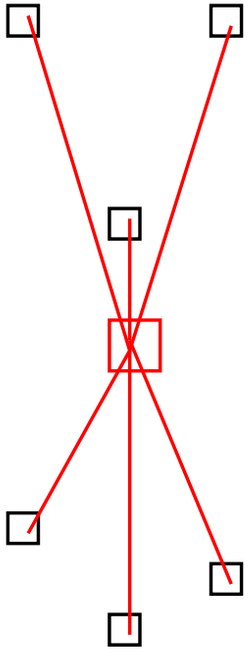


Clusterzentren neu berechnet

UMIT
the health & life sciences university

k-Means-Algorithmus (Lloyd-Verfahren), Beispiel 3





Datenpunkte den Clusterzentren erneut zugeordnet
Zuordnung unverändert → Endergebnis

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode verstehen

→`0704-kmeans-exercise`

- Legen Sie ein neues **C++**-Konsolenprojekt mit den Quelltext- und Headerdateien aus dem (in Moodle bereit gestellten) Verzeichnis `0704-kmeans-exercise` an
- Bringen Sie dieses Programm zum Laufen
- In der gegebenen Version liest das Programm eine Liste von Datenpunkten (2D-Punkte mit kartesischen Koordinaten) aus der Datei `pointlist.txt` ein und gibt diese wieder aus.
- Vollziehen Sie den Aufbau und die Funktionsweise des Programms nach und erarbeiten Sie sich anhand dieses Programms die darin vorkommenden weiteren C++-Sprachelemente

Dieses Projekt benötigt C++11, daher nötigenfalls die entsprechende Compileroption aktivieren!

UMIT
the health & life sciences university

Selbstständiges Üben: Quellcode schreiben

→0705-kmeans (nach der Vorlesung)

In der Datei 0704-kmeans-exercise.cpp ist bereits der Programmrahmen für die Implementation des k-Means-Algorithmus vorbereitet.

- Entfernen Sie die Kommentarzeichen /*. . . */ in der main-Funktion
- Ergänzen Sie den fehlenden Code in der Hilfsfunktion distance() (Berechnung des euklidischen Abstandes zweier Punkte).
- Bereits implementiert ist die Funktion initkmeans() (zufällige Auswahl von k verschiedenen Datenpunkten als Initialisierung für die Clusterzentren)
- Ergänzen Sie den fehlenden Code in der Funktion kmeans() (Iteration aus abwechselnder (Neu-) Zuordnung der Datenpunkte und Neuberechnung der Clusterzentren, mit Abbruchbedingung)
- Testen Sie Ihr Programm mit der bereit gestellten Datei pointlist.txt und $k = 3$ (mehrere Programmläufe!)
- Testen Sie mit anderen Werten für k

UMIT
the health & life sciences university

Inhalte PAD 1

- Strukturierte Programmierung in C
  - Elementare Datentypen, Variablen
  - Funktionen, Kontrollstrukturen
  - Zeiger und Felder
  - Zeichenketten
  - Verbunddatentypen
  - Ein- und Ausgabe, Dateiarbeit
  - Projektstrukturierung
- Strukturierte Programmierung in C++ mit Standardklassen (erste Schritte)
  - Ein- und Ausgabe, Dateiarbeit
  - Zeichenketten
  - Vektor-Container

UMIT
the health & life sciences university

Ausblick auf PAD 2

- Fortsetzung: Strukturierte Programmierung in C++ mit Standardklassen
- Objektorientierte Programmierung in C++ mit selbst erzeugten Klassen
- Standarddatenstrukturen
- Standardalgorithmen