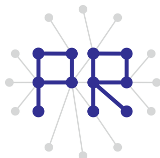
Einführung in die Informatik I

Kapitel II.1: Suchen

Prof. Dr. Marcin Grzegorzek[1]

Research Group for Pattern Recognition
www.pr.informatik.uni-siegen.de

Institute for Vision and Graphics
University of Siegen, Germany



[1] Die im Rahmen dieser Lehrveranstaltung verwendeten Lernmaterialien wurden uns zum Großteil von Herrn Prof. Dr. Wolfgang Wiechert und Herrn Prof. Dr. Roland Reichardt zur Verfügung gestellt.

Inhaltsverzeichnis

I. MATLAB-Einführung

1. Voraussetzungen und Konventionen
2. Variablen und arithmetische Ausdrücke
3. Automatisierungen von Berechnungen
4. Logische Ausdrücke
5. Verzweigungen
6. Schleifen
7. Fehlersuche in Programmen
8. Funktionen
9. Arbeitsweise von Funktionen
10. Vektoren
11. Matrizen

Inhaltsverzeichnis II

II. Algorithmen

- 1. Suchen
- 2. Spezielle Suchalgorithmen
- 3. Sortieren
- 4. Rekursion und Quicksort

Algorithmen

- Der Name stammt aus der mittelaterlichen Übersetzung *Algoritmi de numero Indorum,* eines Werkes des arabischen Mathematikers al-Khwarizmi
- Definition: Systematische Prozedur, die in einer endlichen Zahl von Schritten die Antwort auf eine Frage oder die Lösung eines Problems produziert.
- Algorithmen bestehen aus einzelnen Befehlen.
- Entscheidend ist, dass alle Befehle unmissverständlich sind, d.h. sie haben eine klar festgelegte Bedeutung.
- Das hängt vom Betrachter ab:
  - Kochrezepte sind klar verständlich für Köche
  - Pseudocode ist klar verständlich für Menschen
  - MATLAB-Programme sind eindeutig für MATLAB

Suchen

- Eine grundlegende Operation, die Bestandteil vieler Berechnungsaufgaben ist.
- Das Widerauffinden eines bestimmten Elements oder bestimmter Informationsteile aus einer großen Menge früher gespeicherter Informationen.
- Ziel: Alle Datensätze finden, deren Schlüssel mit einem bestimmten Suchschlüssel übereinstimmen.
- Fertige Programme sind im MATLAB-System bereits für das Suchen (`find`, `min`, `max`) und Sortieren (`sort`) vorhanden.

Beispiel: Karteikasten

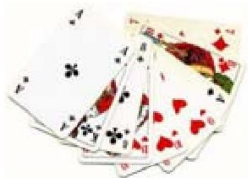
- Die Namen in der Personaldatei sind **alphabetisch sortiert**.
- Jeder Eintrag ist ein Datensatz.
- Der Name oder die Personalnummer ist der **Schlüssel.**
- Der gesuchte Name oder die gesuchte Personalnummer ist der **Suchschlüssel**.

Abstraktion:

- Die Suche nach dem **Namen** entspricht der Suche in einem **sortierten Vektor**.
- Die Suche nach der **Personalnummer** entspricht der Suche in einem **unsortierten Vektor** – weil nach Name sortiert.

Beispiel: Spielkarten

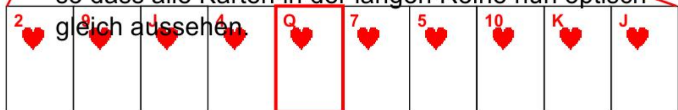
- In einem Stapel von Spielkarten wird nach einer bestimmten Karte gesucht.
- Lösung im **Alltag**: Spielkarten mit der Bildseite nach oben als ungeordneten Haufen ausbreiten und mit dem Auge nach der gewünschten Karte suchen.



- Überlegung: Kann diese Vorgehensweise auch mit den in einer **Programmiersprache** zur Verfügung stehenden Möglichkeiten **umgesetzt** werden? - Das ungeordnete Ausbreiten im Speicher eines Rechners kann nicht realisiert werden.
- Die **nicht realisierbaren Operationen** werden durch andere **ersetzt** bzw. **systematisiert** oder aus einfacheren Operationen zusammengesetzt.

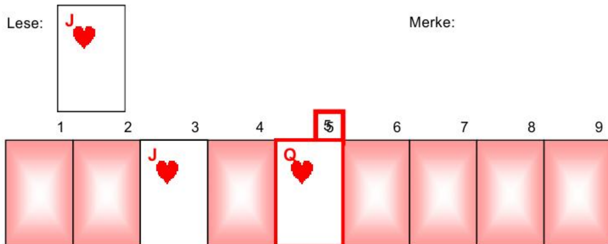
Beispiel: Spielkarten

- Spielkarten werden mit der Bildseite nach oben in einer langen Reihe „ordentlich“ ausgelegt, da vektorielle Daten im Speicher eines Rechners bereits so … sind.
- … die Realisierbarkeit geprüft werden.
- …itliche Überblicken der Karten mit dem Auge … der gewünschten Karte ist nicht … da dies nur mit visueller Mustererkennung …gt werden kann.
- Die Karten werden mit der Bildseite nach unten gedreht, so dass alle Karten in der langen Reihe nun optisch gleich aussehen.

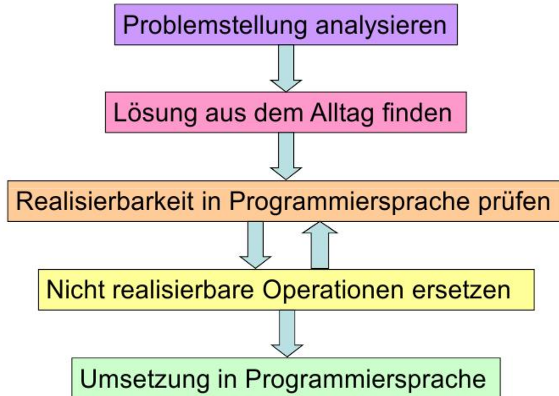


Beispiel: Spielkarten

- Es muss natürlich gestattet bleiben, einzelne Karten umzudrehen, den Kartenwert auszulesen und dann die Karte wieder umzudrehen.
- Außerdem darf man sich einzelne Werte von umgedrehten Karten und auch die Positionen dieser Karten merken, jedoch zu jedem Zeitpunkt der Programmausführung nur eine endliche Zahl solcher Werte.



Heuristik zur Algorithmenentwicklung

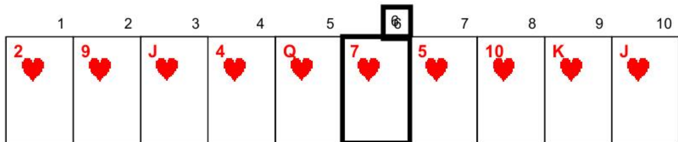


Beispiel: Spielkarten

- Als Beispiel dient folgendes Kartenspiel mit 10 Karten.
- Gesucht wird die „7“, damit ist der Suchschlüssel gegeben.
- Das Kartenspiel wird als Vektor angesehen.
- Der Algorithmus soll den Suchschlüssel im Vektor finden.
- Ggf. Soll auch ermittelt werden, ob der gesuchte Schlüssel überhaupt im Vektor vorkommt.

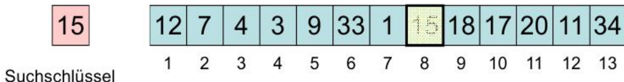
Suchschlüssel

Merke:



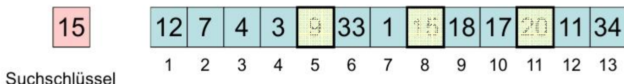
Vektor mit Zahlen

- Als Beispiel wird folgender Vektor mit 13 Zahlen verwendet.
- Der Suchschlüssel ist gegeben.
- Der Algorithmus soll den Suchschlüssel im Vektor finden.
- Ggfs. Soll auch ermittelt werden, ob der gesuchte Schlüssel überhaupt im Vektor vorkommt.



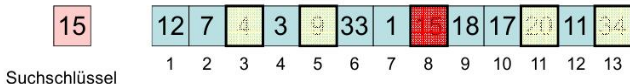
Vergleichsoperationen

- Bei der Suche muss der Suchschlüssel mit dem Inhalt eines Elements des Vektors verglichen werden.
- Zugelassene Vergleichsoperation
  - < kleiner (9 < 15) → Vektor(5)<Suchschlüssel
  - > größer (20 > 15) → Vektor(11)>Suchschlüssel
  - == gleich (15 == 15, Ende der Suche)



Zufälliges Suchen

- Um die Lösung zu finden, rät der Algorithmus eine Zahl zwischen 1 und der Länge des Vektors
- Der Inhalt des zufällig gefundenen Elements wird mit dem Suchschlüssel verglichen.
- Dieses Verfahren wird so lange wiederholt, bis das Element gefunden wurde.
- Beispiel Karteikasten: Irgendeine Karte ziehen, wenn diese falsch war Karte wieder rein und wieder von Vorne.



Beispiel: Zufallssuche

- Der Funktion wird das zu durchsuchende Feld und der Suchschlüssel übergeben. Es wird die Position des Suchschlüssels im Feld zurückgegeben.

Eingabe: $feld(1..n)$ - Daten

key - Suchschlüssel

$p \leftarrow Zufallszahl(1..n)$

while $feld(p) <> key$

$p \leftarrow Zufallszahl(1..n)$

end while ;

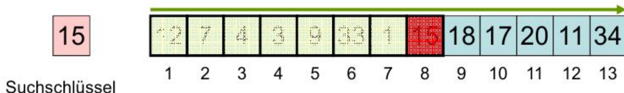
```
function Position=Suche_Zufall(x,Key)
Position=ZufallPosi(length(x));
while x(Position)~=Key
    Position=ZufallPosi(length(x));
End

function Posi=ZufallPosi(n)
      Posi=ceil(n*rand);
```

- Das zufällige Suchen hat den Nachteil, dass die Zufallsfunktion u.U. mehrmals die selbe Zahl liefert.
- Falls der Suchschlüssel im Feld nicht vorkommt, dann bricht der Algorithmus nicht ab (Endlosschleife).

Sequentielle Suche

- Die Elemente werden vom Anfang beginnend nacheinander geprüft.
- Die Schleife wird beendet, sobald das Element gefunden ist.
- Die Funktion liefert bei erfolgloser Suche die Position 0 zurück.
- Andere Namen für dieses Verfahren:
  - Erschöpfendes Durchsuchen
  - Brute Force



Beispiel: Sequentielle Suche

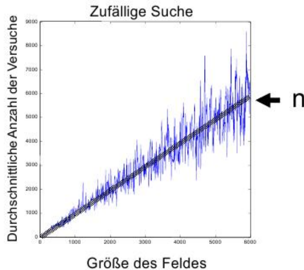
- Mit Hilfe der **break**-Anweisung kann man eine Schleife vorzeitig beenden:

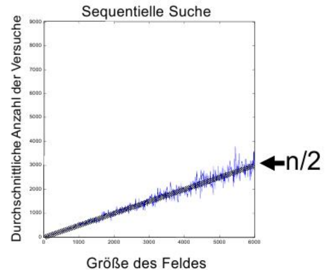
```
function Position=Suche_Seq(x,Key);
Position=0; % Default Rückgabewert
for i=1:length(x)
    if x(i)==Key    % Gefunden...
        Position=i;    % Position merken
        break          % Schleife verlassen
    end
end
```

- Die **break**-Anweisung bezieht sich immer nur auf die innerste **for**- oder **while**-Schleife, die den Befehl umgibt. Sie springt hinter das Ende dieser Schleife.

Analyse des Laufzeitverhaltens

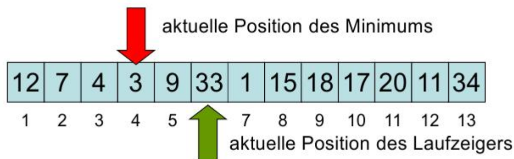
- Je 30 mal eine Zufallszahl in einem Intervall 1:n finden (n=1,2,3,…).
- Die Mittelwert der 30 Versuche wird gebildet





Minimumssuche

- Gesucht wir die Position des Minimums in einer unsortierten Liste.
- Im Gegensatz zum einfachen Suchen
  - ergibt sich erst während der Suche, welcher Eintrag überhaupt gesucht ist
  - kann das Minimum immer gefunden werden
- Für die Minimumssuche werden alle Werte der Reihe nach besucht. Der bisher kleinste Wert wird markiert.



Implementierung der Minimumssuche

```
function pos = minPos(x)
% Position des Minimums finden
n=length(x);        % Anzahl der Werte
pos=1;              % Bisher kleinstes Element
for i = 2:n         % Alle anderen Werte besuchen
   if x(i)<x(pos)   % Neues Minimum gefunden
     pos = i ;
end
```