# 1 Background

Necessary background for topic, in order to understand the problem, motivation and contribution. Approximately 10 pages.

## 1.1 JSON

## 1.2 D3

D3 - kurz für Data-Driven Documents - ist eine Javascript Bibliothek zur Visualisierung von Daten auf Internetseiten. Sie ist ein maßgebliches Hilfsmittel im Zuge dieser Arbeit. Das Konzept von D3 ist es, Datensätze mit Elementen eines HTML-Dokuments zu verknüpfen und dadurch großen Einfluss auf die Darstellung dieser Daten zu haben. Dabei werden neben Hilfsfunktionen zum Strukturieren der Datenanzeige auch diverse Möglichkeiten zur Erstellung und Manipulierung von Datenstrukturen angeboten. Da D3 sehr umfangreich ist und die Meisten Anwendungen nicht alle Funktionen benötigen, ist es in Module unterteilt, die einzeln eingebunden werden können. Die für diese Arbeit besonders relevanten Module sind *Selections*, *Hierarchies*, *Shapes* und *Transitions*.

### 1.2.1 Selections

*Selections* sind einer der wichtigsten Bestandteile von D3. Sie dienen dazu HTML-Elemente zu gruppieren, manipulieren und mit Daten zu verknüpfen. Man erstellt sie mit den Befehlen $d3.select(selector)$ oder $d3.selectAll(selector)$, wobei *select* das erste auf den Übergebenen *selector* passende und *selectAll* alle passenden Elemente in eine *Selection* zusammenführt. Neben den ausgewählten Elementen enthält diese dann auch eine Reihe von Hilfsfunktionen, wie zum Beispiel $selection.data()$. [todo: Erklärung von selections und speziell der Funktionen enter(), exit() und data()]

### 1.2.2 Hierarchies

[todo: Erklärung von heirarchies und wie man damit bäume speichern kann]

### 1.2.3 Shapes

### 1.2.4 Transitions

## 1.3 SVG

[todo: erläuterung von svg elementen, welche elemente im inneren benutzt werden können und ihre bedeutung]

# 2
# Contribution

Most important chapter of the thesis. Describes what the author contributes as research. Discusses intuition, motivation, describes and reasons about necessity of proposed elements. Defines theses based on reasonable assumptions. Discusses relevant aspects of contribution. Approximately 30 to 40 pages. Can be split into multiple chapters.

## 2.1 Baumlayout

### 2.1.1 Linear vs. Radial

Aufgrund des begrenzten Platzes, der auf mobilen Geräten typischerweise zur Verfügung steht, war früh in der Entwicklung zu entscheiden, wie Baumstrukturen möglichst platzsparend anzuordnen sind, ohne Übersichtlichkeit einzubüßen. Betrachtet wurden dabei speziell die allgemein übliche Darstellung (linear) gegenüber einer kreisförmigen (radialen) Anordnung.
Die Abbildungen 2.1 und 2.2 zeigen einen Vergleich zwischen einem Baum in linearer (Abb. 2.1) und in radialer Anordnung (Abb. 2.2). In beiden Fällen ist derselbe Baum der Höhe 4 auf gleicher Fläche abgebildet. Er besteht aus 105 Knoten, wobei alle inneren Knoten jeweils vier Nachfolger haben. Der Wurzelknoten befindet sich beim linearen Layout am oberen Rand. Die übrigen Knoten sind in drei horizontalen Linien darunter angeordnet, wobei die Kinder der Wurzel auf der obersten Linie liegen, deren Kinder auf der mittleren und deren Kinder auf der untersten Linie. Beim radialen Layout ist die Wurzel in der Mitte abgebildet und alle anderen Knoten in drei konzentrischen Kreisen darum herum. Knoten der Tiefe 1 liegen auf dem innersten Kreis, der Tiefe 2 auf dem mittleren Kreis und die Blätter auf dem äußeren Kreis. Den Mittelpunkt der Kreise bildet die Wurzel. Man sieht deutlich, dass im Fall der vertikalen Anordnung (Abb. 2.1) nicht ausreichend Platz für alle Blätter zur Verfügung steht, wodurch es zu starken Überschneidungen kommt. In der radialen Anordnung (Abb. 2.2) können hingegen alle Knoten überschneidungsfrei dargestellt werden.
Um den Grund dafür zu veranschaulichen vergleicht man den Platz, der bei den beiden Herangehensweisen auf einer einzelnen Stufe des Baumes zur Verfügung steht. Unter der Annahme,

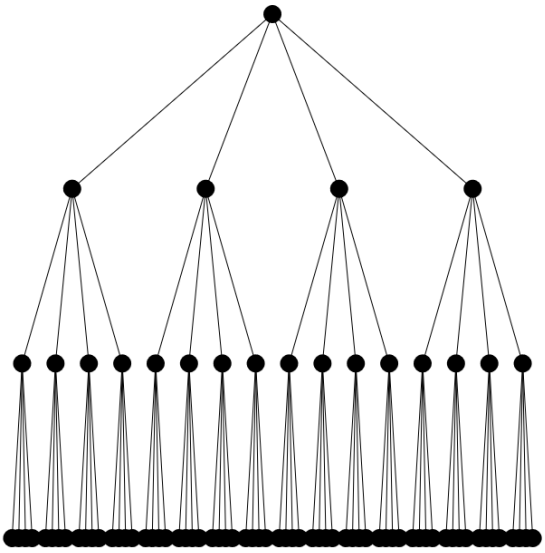

Abbildung 2.1: Lineare Baumdarstellung

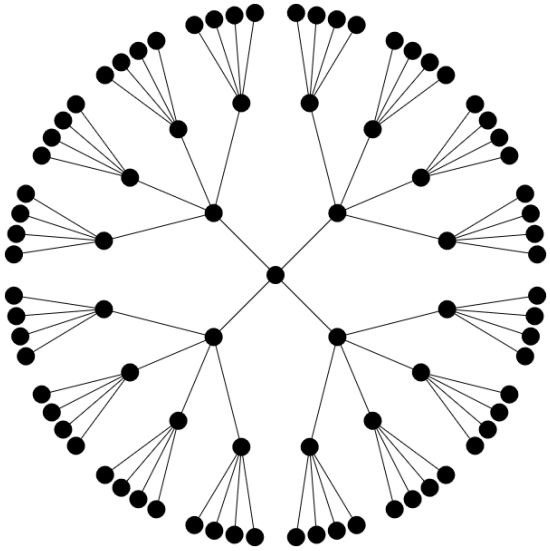

Abbildung 2.2: Radiale Baumdarstellung

dass für die gesamte Darstellung ein Quadrat mit Seitenlänge $a$ zur Verfügung steht und alle Knoten einen Durchmesser von 1 haben ergibt sich für den Platz $l_{linear}$ bzw. $l_{radial}$, der auf einer Stufe des Baumes verfügbar ist

$$\begin{aligned} l_{linear} &= a \\ l_{radial} &= 2 \cdot \pi \cdot r \qquad mit \quad 0 \leq r \leq \frac{a}{2}. \end{aligned}$$

Dabei bezeichnet $r$ den Abstand einer Stufe zum Wurzelknoten. Setzt man $l_{radial}$ und $l_{linear}$ gleich und stellt nach $r$ um, so ergibt sich

$$\begin{aligned} & l_{linear} = l_{radial} \\ \Rightarrow \quad & a = 2 \cdot \pi \cdot r \\ \Rightarrow \quad & \frac{a}{2 \cdot \pi} = r \\ \Rightarrow \quad & r \approx \frac{a}{6}. \end{aligned}$$

Da $l_{linear}$ konstant ist kann man daraus ableiten

$$l_{radial} > l_{linear} \qquad \Leftrightarrow \qquad r > \frac{a}{6}.$$

Das bedeutet, dass ab einem Abstand zur Wurzel von mehr als $\frac{a}{6}$ im radialen Layout mehr Platz für jede Stufe verfügbar ist. Außerdem gilt: Je größer der Abstand $r$, desto mehr Platzersparnis. Das ist vor allem deshalb von Interesse, weil Bäume in der Regel die Eigenschaft haben, dass mit wachsender Baumtiefe auch die Anzahl der Knoten in der jeweiligen Tiefe wächst und diese tiefer liegenden Knoten in der kreisförmigen Darstellung weiter von der Wurzel entfernt sind. Das radiale Layout bietet somit genau dann mehr Platz, wenn üblicherweise mehr Platz benötigt wird. Aufgrund dieser Erkenntnis und der Tatsache, dass beide Darstellungsformen gute Übersichtlichkeit aufweisen, fiel die Wahl letztendlich auf das radiale Layout.

### 2.1.2 Polarkoordinaten

D3 bietet zur Berechnung einer sinnvollen Knotenanordnung von Bäumen die Funktion $d3.tree()$ an, welche unter Verwendung des Reingold-Tilford Algorithmus [todo: quelle] allen Knoten eines Baumes x- und y-Koordinaten jeweils im Bereich von 0 bis 1 zuordnet. Es liegt dann in der Hand des Programmierers, diese sinnvoll zu interpretieren. Um die Knoten, wie in Abbildung 2.2 gezeigt in konzentrischen Kreisen anzuordnen, eignen sich Polarkoordinaten ausgezeichnet. Einem Vorschlag aus der Dokumentation von D3 folgend [todo: quelle] wird die y-Koordinate als Radius $\rho$, die x-Koordinate als Polarwinkel $\varphi$ in Radiant interpretiert. [todo: bild]

$$\rho = y$$
$$\varphi = 2 \cdot \pi \cdot x$$

Zur Darstellung auf dem Bildschirm müssen $\rho$ und $\varphi$ anschließend in kartesische Koordinaten $x_{screen}$ und $y_{screen}$ umgerechnet werden. Die allgemeinen Umrechnungsformeln ergeben sich als

$$x_{screen} = \rho \cdot \cos(\varphi)$$
$$y_{screen} = \rho \cdot \sin(\varphi).$$

Bei dieser Umrechnung wird allerdings noch nicht berücksichtigt, dass die gegebenen Polarkoordinaten auf generischen Koordinaten im Bereich $[0,1]$ basieren. Zur korrekten Positionierung müssen noch eine Skalierung auf die verfügbare Breite (width) $w$ und Höhe (height) $h$, sowie eine Verschiebung in die Mitte der Anzeige vorgenommen werden. Es entstehen die endgültigen Formeln:

$$x_{screen} = \rho \cdot \cos(\varphi) \cdot w + \frac{w}{2} \tag{2.1}$$
$$y_{screen} = \rho \cdot \sin(\varphi) \cdot h + \frac{h}{2}. \tag{2.2}$$

Da die theoretische Seite des grundlegenden Layouts damit geklärt ist, bleibt noch zu erörtern, wie Knoten in der Praxis ihren Weg auf den Bildschirm finden. Auch hier kommt wieder D3 zum Einsatz.

## 2.2 Anzeigen der Knoten mit D3

Um mit D3 auf den Bildschirm zu zeichnen gibt es verschiedene Möglichkeiten. In der Regel werden die HTML-Elemente *canvas* oder *svg* als Zeichenfläche verwendet. Beide haben Vor- und Nachteile, die vor einer Entscheidung abzuwägen sind.

### 2.2.1 Canvas versus SVG

Der größte Unterschied zwischen *canvas* und *svg* Elementen besteht darin, dass ein *svg* jedes Objekt, dass es anzeigen soll einzeln als HTML-Element hinterlegt während ein *canvas* nur das insgesamt entstehende Bild speichert. Für den Browser bedeutet das, dass bei einem *canvas* nur die Darstellung eines einzelnen Bildes zu berechnen ist, während bei einem *svg* alle dargestellten Objekte getrennt behandelt werden. Das führt dazu, dass mit wachsender Zahl anzuzeigender Elemente die Berechnungsdauer des *svg* stärker ansteigt, als die des *canvas*.

Das einzelne Speichern der Anzeigebausteine bringt aber auch Vorteile mit sich. Es ist einfach ein einzelnes Objekt auf dem Bildschirm zu verändern, indem man dessen *svg*-Attribute anpasst, weil der Browser sich dann selbst um die Aktualisierung der Anzeige kümmert. D3 bietet mit seinen *transitions* eine simple Möglichkeit beliebige Animationen auf einem *svg* Element durchzuführen, was auf einem *canvas* umständlicher mit D3 *interpolators* und einer Funktion zu lösen ist, die für jedes Einzelbild der Animation das Bild neu zeichnet. Allgemein scheint die Nutzung eines *svg*-Elements zur Darstellung von Daten mit D3 die am häufigsten gewählte Herangehensweise, was zu einer großen Menge Beispiele führt, die zur Inspiration genutzt werden können. Auch in diesem Fall fällt die Wahl darauf D3 in Verbindung mit *svg* einzusetzen. Diese Entscheidung wird vor allem durch die in Kapitel 2.3 beschriebene Reduzierung der angezeigten Knoten untermauert, denn der große Vorteil schnellerer Berechnung auf *canvas*-Elementen fällt damit kaum noch ins Gewicht.

### 2.2.2 Datenstruktur

Zur Speicherung der Baumdaten werden die Knoten in eine D3 *hierarchy* Umgewandelt. Es ist zu beachten, dass eine *hierarchy* nur durch ein einzelnes Objekt repräsentiert wird, welches die Wurzel (*root*) des Baumes darstellt. Um eine Array aller Knoten zu erhalten kann $root.descendants()$ aufgerufen werden. Jeder einzelne Knoten ist dabei ein Javascript-Objekt, das wichtige Informationen, wie id, Name und ausführliche Beschreibung enthält. Wendet man $d3.tree()$ auf die *hierarchy* an, so kommen zusätzlich noch x- und y-Koordinaten dazu. Unter Einsatz dieser Daten kann der Baum auf ein *svg*-Element gezeichnet werden.

### 2.2.3 Anzeige auf SVG

Mit Hilfe von $d3.select()$ und $selection.append()$ wählt man zuerst das HTML-Element aus, welches die Anzeige beinhalten soll, und hängt ihm ein *svg* an. Der Rückgabewert von *append* ist dabei eine *selection*, die nur das *svg*-Element enthält. Diese nutzt man beim weiteren Vorgehen immer dann, wenn eine Änderung am *svg* vorgenommen werden soll.
Dinge auf einem *svg* anzuzeigen bedeutet, ihm Unterelemente hinzuzufügen. Es gilt also diese aus den gegebenen Knotendaten zu generieren. Es ergibt Sinn, dem *svg* mit *append* zunächst zwei Gruppen (*svg*-Element *g*) für Knoten und Kanten des Baumes einzufügen. Diese werden im Weiteren als Knotengruppe und Kantengruppe bezeichnet. [todo: d3 Vorgehensweise beim Hinzufügen von Elementen im Background erklären (select()/selectAll() -> data() -> enter() -> append()) und das hier dann umschreiben?] Abbildung 2.3 zeigt das gewünschte Aussehen für Knoten und Kanten. Zu sehen sind zwei Knoten als Kreise mit Beschriftung A und B. Die Kante zwischen den Beiden Knoten ist dargestellt als Linie, die beide Kreise miteinander verbindet. Die Unterschiedlichen Farben der Knoten zeigen an, dass sie verschiedene Knotentypen haben. Im medizinischen Kontext könnte es sich zum Beispiel um ein Symptom A und eine Diagnose B handeln.
Mit *selectAll* werden alle *g* Elemente innerhalb der Knotengruppe ausgewählt (zu Beginn gibt es keine) und die entstehende *selection* wird durch $selection.data()$ mit Daten verknüpft. Als Datensatz dient dabei die von $root.descendants()$ zurückgegebene Liste aller Knoten. $selection.enter()$ liefert dann die Knoten, welche noch kein zugehöriges *g*-Element in der Knotengruppe haben und es wird für jeden ein solches erstellt mit einem *circle*- und *text*-Element im Inneren. Die Vorgehensweise beim einfügen der Kanten verläuft analog, mit dem Unter-

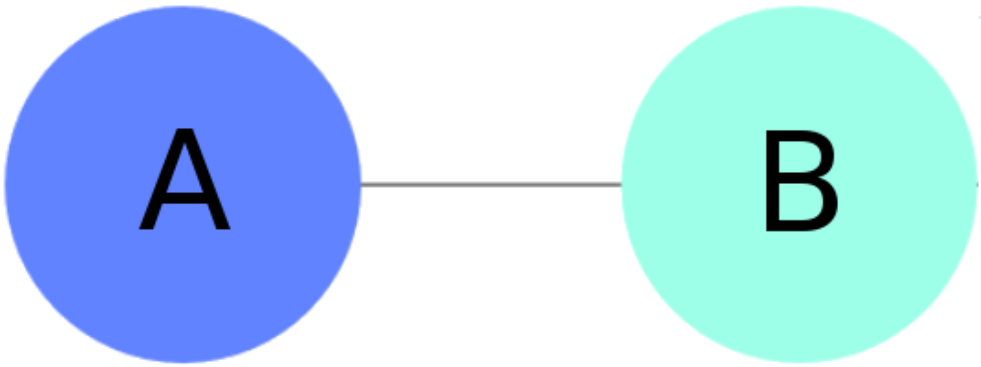


Abbildung 2.3: Konzept für Knoten und Kanten

schied, dass als Datensatz die Rückgabe von $root.links()$ genutzt wird, welche eine Liste aller Kanten des Baumes ist. Anstelle von Gruppenelementen mit *circle* und *text* als Inhalt fügt man der Kantengruppe *path*-Elemente ein, die sich mit $d3.line()$ generieren lassen.
Mobile Geräte haben keine einheitlichen Bildschirmgrößen. Daher variiert Der Radius der Kreise, die die Knoten repräsentieren. Die Größe der Aufschrift soll so berechnet werden, dass sie so groß wie möglich ist, aber nicht den umgebenden Kreis überschreitet.

### 2.2.4 Berechnen der Schriftgröße

[todo: aufteilung auf zwei zeilen implementieren] Angaben von Schriftgrößen beziehen sich auf die Höhe der Buchstaben [todo: quelle]. Das bedeutet nicht notwendigerweise, dass eine Schriftgröße von 10px zu Buchstaben führt, die exakt Zehn Pixel hoch sind, sondern dass die Höhe der Schriftzeichen proportional zur angegebenen Schriftgröße ist. Das bedeutet auch, dass die Breite eines Buchstabens dazu proportional ist, da er beim Vergrößern nicht verzerrt werden soll, wodurch das Verhältnis von Höhe zu Breite immer gleich bleibt. Diese Tatsache lässt sich auf ganze Texte übertragen...

## 2.3 Reduzieren der angezeigten Knoten

Nachdem mit dem radialen Layout (Kapitel 2.1.1) eine erste Maßnahme zum Einsparen von Platz ergriffen wurde, stellt sich als nächstes die Frage, wie die Übersichtlichkeit weiter verbessert werden kann. Da Knoten auf dem Bildschirm Informationen in Form von Text beinhalten sollen, ist es abzusehen, dass jeder einzelne Knoten mehr Platz einnehmen wird, als z.B. in Abbildung 2.2 gezeigt ist. Es bietet sich an, immer nur aktuell relevante Knoten ein- und irrelevante auszublenden. Das erfordert je nach Eingabe dynamische Änderungen an der Anzeige der Baumstruktur (Kapitel 2.4). Zunächst muss jedoch entschieden werden, welche Knoten aktuell relevant oder irrelevant sind. In Abbildung 2.4 ist beispielhaft ein Entscheidungsbaum zu sehen, der in stark vereinfachter Form die Entscheidungsfindung zur Frage, ob ein Regenschirm nötig ist, zeigt. Innere Knoten sind als Ellipsen dargestellt und Blätter, welche endgültige Er-

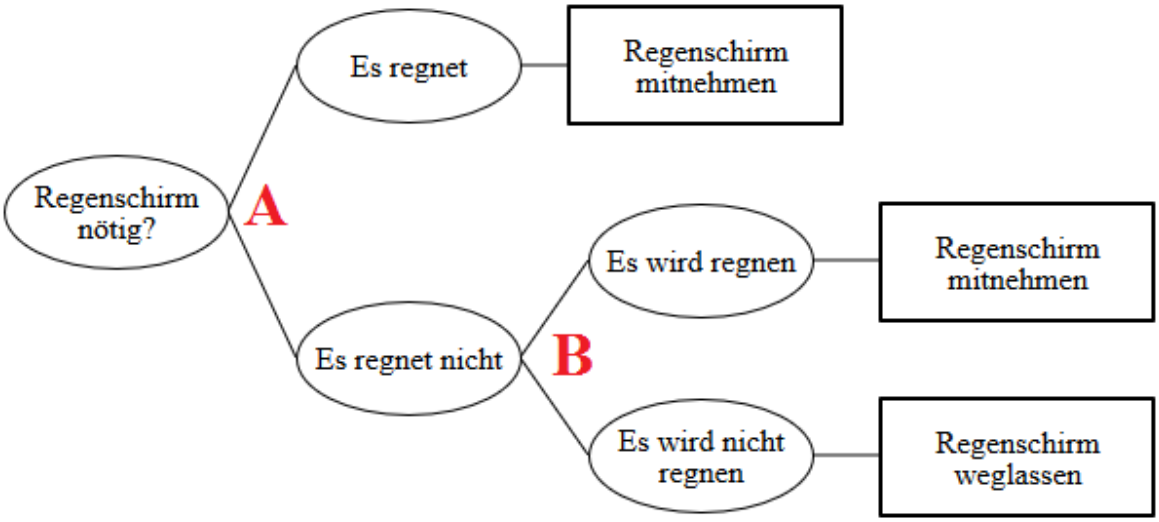


Abbildung 2.4: Ein simpler Entscheidungsbaum

gebnisse repräsentieren, sind rechteckig. Die Beschriftung der Knoten zeigt an, welche Aussage oder Frage sie symbolisieren. Die beiden Stellen, an denen Entscheidungen getroffen werden müssen, sind mit *A* und *B* markiert. In Situation *A* gilt es zu entscheiden, ob es regnet oder nicht. Um diese Entscheidung treffen zu können, ist nicht relevant, ob es zu einem späteren Zeitpunkt regnen wird oder nicht und welche Ergebnisse sich daraus ableiten lassen. Bei *B* wiederum sind vorherige Entscheidungsmöglichkeiten nicht von Interesse, ebenso wie die Folgen davon, ob es regnen wird oder nicht. Man kann sagen, dass bei jeder Verzweigung nur die zur Verfügung stehenden Alternativen relevant sind und angezeigt werden müssen. Da es jedoch nicht nur darum geht, so viel Platz wie möglich zu sparen, sondern auch darum, eine übersichtliche und intuitiv Verständliche Darstellung zu finden ist es hilfreich, außerdem noch den Knoten, von dem die Verzweigung ausgeht zu zeigen. Bei *A* also den Knoten „Regenschirm nötig“, bei *B* „Es regnet nicht“. Auf diese Art ist es leichter die Orientierung zu behalten, selbst wenn ein Großteil des Baumes nicht sichtbar ist.

## 2.4 Navigieren durch den Baum

Durch das Ausblenden vieler der Knoten muss nun eine Möglichkeit geschaffen werden, von einer Entscheidung zur nächsten zu navigieren, wobei immer wieder irrelevante Knoten aus- und relevant gewordene eingeblendet werden. Das wird möglich gemacht, indem die Baumanzeige auf Berührung hin dynamisch aktualisiert wird. Es gilt dabei, dass der Knoten, von dem die aktuelle Verzweigung ausgeht, in der Mitte des Bildschirms zu sehen ist und die nachfolgenden Entscheidungsmöglichkeiten im Kreis darum herum angeordnet werden. Abbildung 2.5 zeigt die Reaktion des Baumes auf verschiedene Nutzereingaben. Auf der linken Seite ist der Ausgangszustand zu sehen und rechts der Nachfolgezustand. Links ist der mittlere Knoten mit „Regenschirm nötig?“ beschriftet und dessen Nachfolger mit „Es regnet nicht“ sowie „es regnet“. Rechts sieht man „Es regnet nicht“ in der Mitte, „Es wird nicht regnen“ und „Es wird

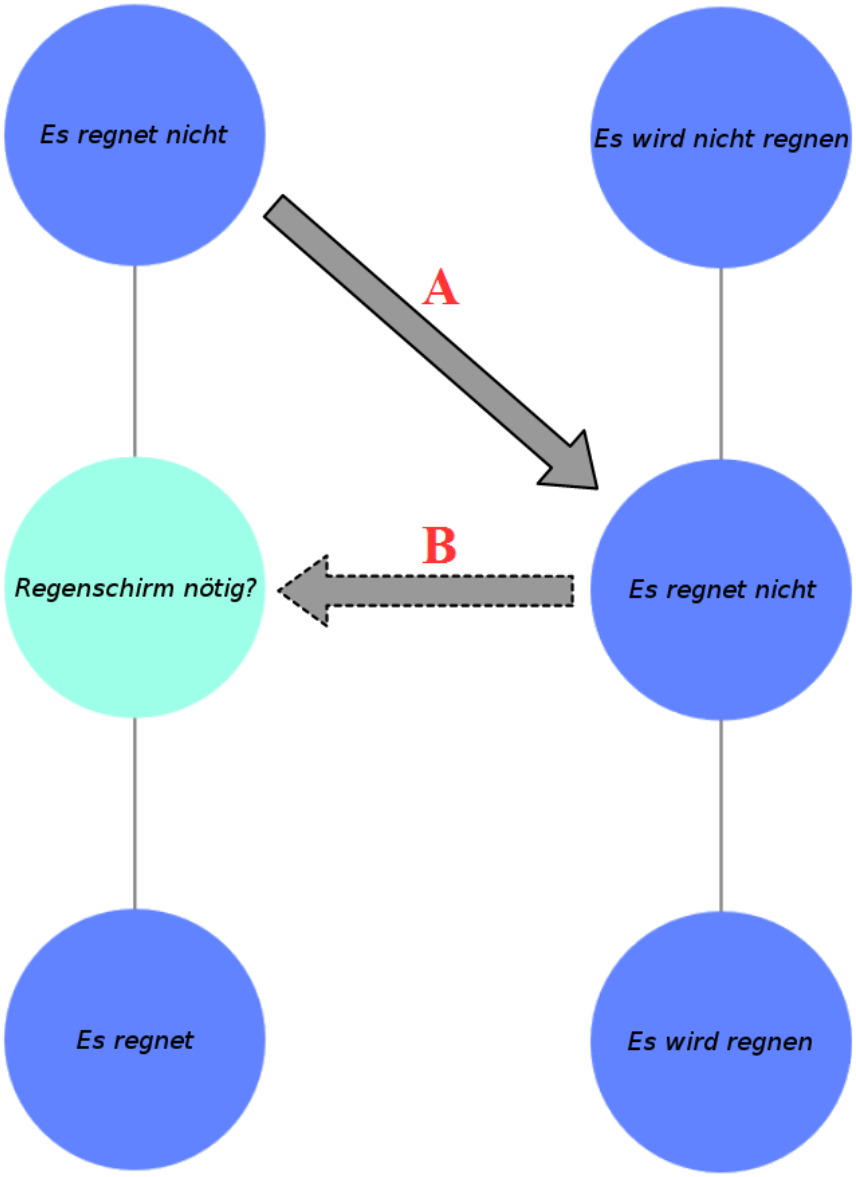


Abbildung 2.5: Beispiel zur Baumnavigation

regnen“ als Folgeknoten. Die mit *A* und *B* beschrifteten Pfeile symbolisieren Reaktionen auf Eingaben, die im Folgenden genauer beschrieben werden.
Durch Antippen eines der äußeren Knoten wird dieser in die Mitte verschoben und es zeigen sich dessen Nachfolger. Tippt man zum Beispiel in Abbildung 2.5 links „Es regnet nicht“ an, dann wird dieser, wie Pfeil *A* zeigt, zum mittleren Knoten, die anderen verschwinden und es erscheinen die neuen Nachfolgeknoten. Möchte man einen Schritt zurück machen, kann der Mittelknoten berührt werden und man wird wieder zur links gezeigten Situation geleitet (illustriert durch Pfeil *B*). Es soll dadurch das Gefühl entstehen, dass man sich durch die Baumstruktur bewegt, wobei besonders die gewählten Animationen (Kapitel 2.5) von großer Wichtigkeit sind. Zuerst ist aber zu klären, wie die interne Datenstruktur verwendet werden kann, um dynamische Aktualisierungen zu ermöglichen.

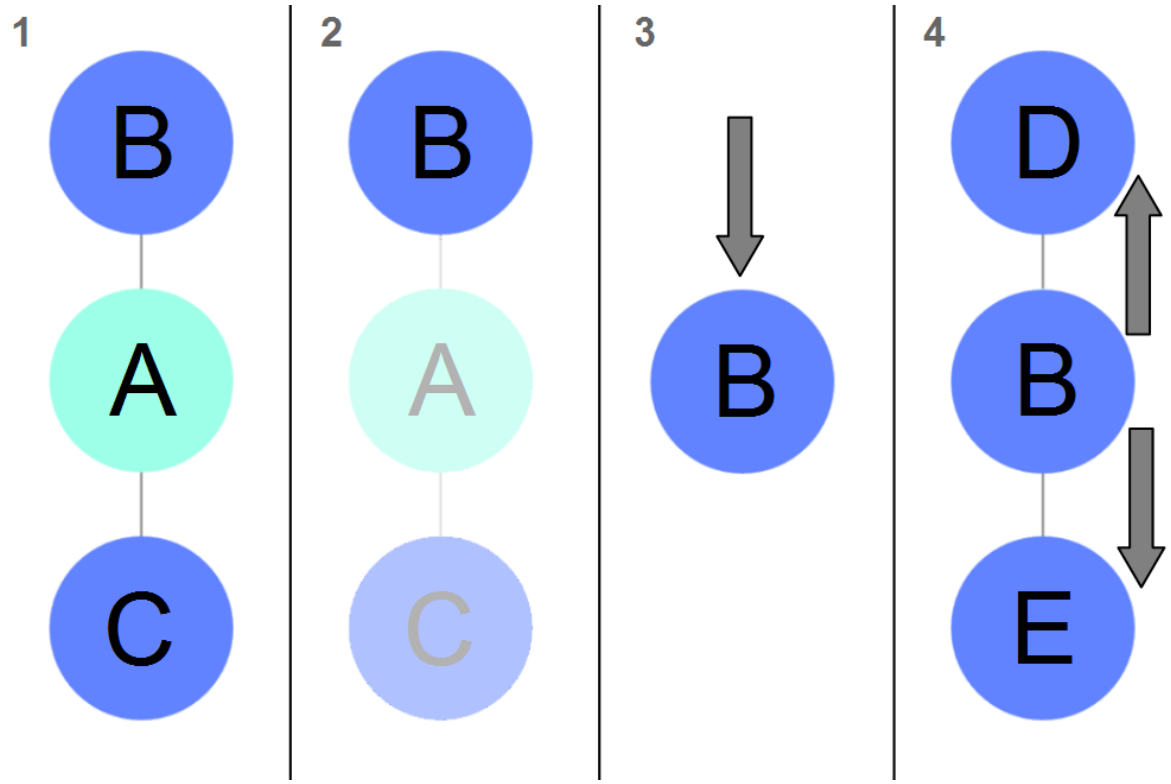


Abbildung 2.6: Animation beim Berühren eines Knotens

### 2.4.1 Dynamische Aktualisierung

## 2.5 Einsatz von Animationen

Häufig kommen bei der hier beschriebenen interaktiven Visualisierung Animationen zum Einsatz und spielen dabei für diverse Aspekte eine zentrale Rolle. [todo: verweise auf material design guidelines]

### 2.5.1 Verbesserung von Orientierung und Übersichtlichkeit

Kapitel 2.3 beschreibt, wie zum Einsparen von Platz Knoten weggelassen werden können. Ein daraus folgender Nachteil besteht im Orientierungsverlust. Obwohl die Menge an Informationen auf dem Bildschirm sinkt, was die Übersichtlichkeit verbessert, wird es schwerer zu wissen, wo man sich im Baum gerade befindet. Durch passende Animationen beim in Kapitel 2.4 beschriebenen dynamischen Aktualisieren des Baumes kann dem Abhilfe geschaffen werden. Es soll dabei durch die Bewegungen auf dem Bildschirm deutlich werden, welche Veränderung eine Eingabe bewirkt. Die gewählte Animation beim Navigieren von einem Knoten zum nächsten ist in Abbildung 2.6 zu sehen. Von links nach rechts mit 1 bis 4 nummeriert werden die vier wichtigen Zeitpunkte im Bewegungsablauf gezeigt. Situation 1 zeigt den Ausgangszustand: Ein Knoten *A* mit dessen Nachfolgerknoten *B* und *C*. Die anderen drei Stadien zeigen was geschieht, wenn man *B* antippt. Zunächst werden bei 2 die Knoten *A* und *C* durchsichtiger, bis sie nicht mehr zu sehen sind. Anschließend bewegt sich *B* in die Mitte der Anzeige (zu sehen bei 3) und zuletzt bewegen sich zu Zeitpunkt 4 von *B* aus die Knoten *D* und *E* nach außen.

Das Ausblenden von *A* und *C* verdeutlicht, dass beide nicht mehr relevant sind, geschieht aber zur Vermeidung von Verwirrung nicht von einem Moment auf den anderen. Die Bewegung von *B* in die Mitte lässt leicht nachverfolgen, dass *B* zum neuen Verzweigungspunkt wird. Wichtig ist dabei, dass *B* zu keinem Zeitpunkt vom Bildschirm verschwindet oder umherspringt, wodurch klar ist, dass es sich noch immer um denselben Knoten handelt. Zuletzt verdeutlicht die von *B* ausgehende Bewegung von *D* und *E* nach außen, dass diese die Nachfolger von *B* sind. Der allgemeine Schwerpunkt liegt darauf, es durch kontinuierliche Bewegungen leichter zu machen, dem Geschehen zu folgen, was zu einem intuitiven Verständnis der Vorgänge auf dem Bildschirm führt.

### 2.5.2 Interaktivität