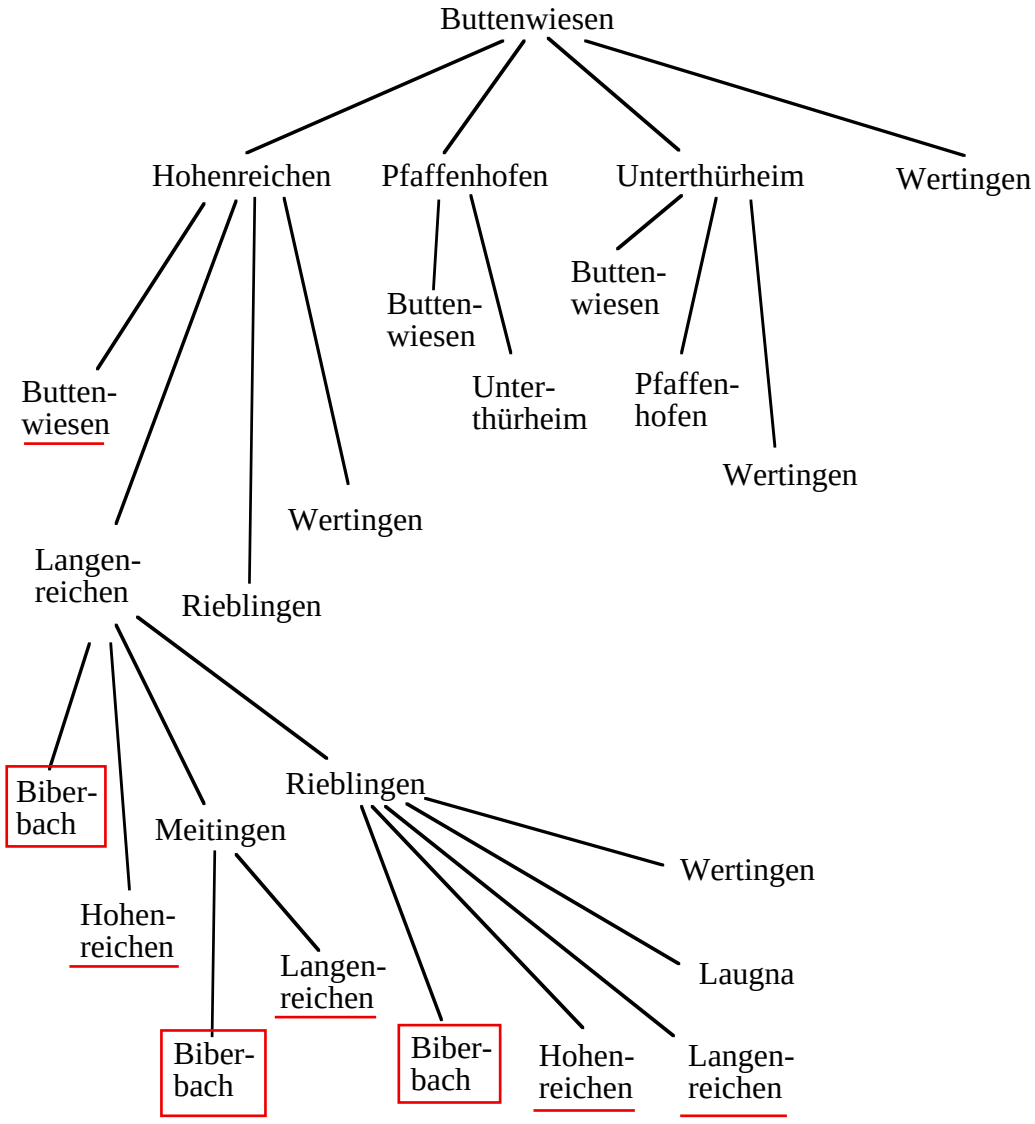
I.2 Backtracking - Verfahren

Nun ergibt sich das Problem des kürzesten Weges. Wie findet man z.B. den kürzesten Weg von Buttenwiesen nach Biberbach. Eine erste Möglichkeit besteht den Weg durch Probieren zu finden. Man geht von Buttenwiesen aus und probiert sämtliche Wege aus, die nach Biberbach führen und nimmt dann den kürzesten. Das Vorgehen lässt sich in einem Baum darstellen. Die Reihenfolge der zu testenden Wege ergibt sich aus der Ordnung in der Adjazenzmatrix:



Der Baum zeigt nur einen Teil des Suchbaums. Nach den roten Strichen wird der Baum nicht fortgesetzt.

Ein Ast des Suchbaums hat keine weitere Verzweigung, wenn:
1. Das Ziel erreicht ist
2. Der Ausgangspunkt erreicht wurde
3. Ein Ort erreicht wurde, an dem man bereits einmal war.

Dieser Suchbaum wird nun durchsucht.

Tiefensuche - Backtracking:
Bei der Tiefensuche (depth-first) wird ausgehend von der Wurzel von einer Verzweigung zur anderen durchlaufen, bis man am Ende angelangt ist. Dann geht man um eine Verzweigung zurück und durchsucht den nächsten Weg bis zum Ende. Da man bei der Tiefensuche immer wieder einen Schritt zurückgeht, spricht man auch vom Backtracking - Verfahren.

Verfahren zur Bestimmung des kürzesten Weges
Den kürzesten Weg findet man, indem man den Suchbaum in Tiefensuche durchläuft. Jedesmal, wenn man das gewünschte Ziel erreicht, wird der Weg und die Fahrstrecke notiert. Danach wählt man von allen diesen Wegen, den mit der kürzesten Wegstrecke aus.

Das obige Verfahren muss nun etwas präzisert werden. Um zu entscheiden ob man bereits in einem Ort war oder nicht, muss der aktuelle Weg abgespeichert werden. Außerdem muss ein gefundener Weg mit dem bisher kürzesten gefundenen Weg verglichen werden. Dazu benötigt man einige neue Variablen:

Benötigte Variablen:

```
aktueller_Weg : tWeg;        (* Ortskennungen des zurückgelegten
                              Weges *)
aktuelle_Anzahl : INTEGER;   (* Anzahl, der als Weg gespeicherten
                              Orte *)
aktuelle_Laenge : INTEGER;   (* zurückgelegte Wegstrecke des
aktuellen Weges *)
minimaler_Weg : tWeg;        (* gefundener Weg mit minimaler
Wegstrecke *)
minimale_Anzahl : INTEGER;   (* Anzahl, der im minimalen Weg
gespeicherten Orte *)
minimale_Laenge : INTEGER;   (* Wegstrecke des minimalen Weges *)
```

Die eigentliche Wegesuche erfolgt in der Prozedur Suche_Weg. Die Prozedur Kurzer_Weg dient vor allem zur Eingabe von Start und Zielort, der Intitialisierung der Variablen, sowie der Ausgabe des Ergebnisses.

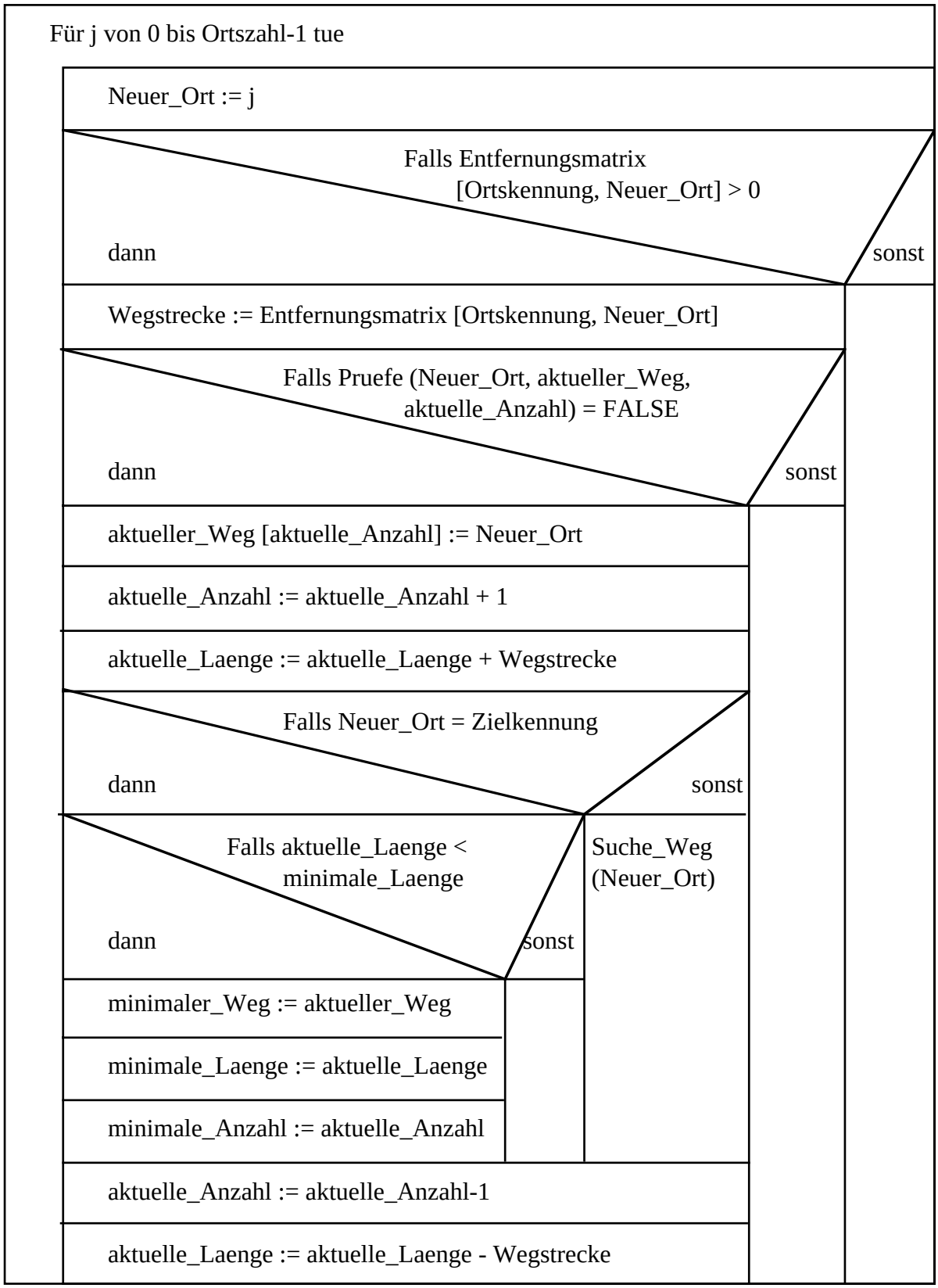
PROCEDURE Kurzer_Weg

Eingabe (Startort)
Eingabe (Zielort)
Startkennung := Kennung (Startort)
Zielkennung := Kennung (Zielort)
aktuelle_Laenge := 0
minimale_Laenge := 10000 (* möglichst große Zahl *)
aktueller_Weg [0] := Startkennung
aktuelle_Anzahl := 1
Suche_Weg (Startkennung)
Ausgabe (minimale_Laenge)
Ausgabe (minimaler_Weg)

Die eigentliche Suche des Weges erfolgt dann in der Prozedur Suche_Weg

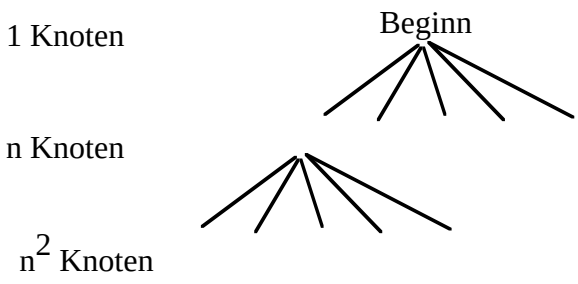
Beschreibung der Wegesuche nach dem Backtracking - Verfahren
Vom Ausgangsort aus überprüft man die möglichen Wege zum nächsten Ort. Wurde dieser Ort auf dem bisherigen Ort noch nicht erreicht, so wird er in den aktuellen Weg eingetragen und akt_Anzahl um eins erhöht. Außerdem wird die aktuelle Länge um die Wegstrecke erhöht. Ist der Neue_Ort der Zielort, so wird die aktuelle Länge mit der minimalen Länge verglichen. Ist die aktuelle Länge kleiner so wird der aktuelle Weg als minimaler Weg und die aktuelle Länge als minimale Länge gespeichert. Ist der Neue_Ort nicht der Zielort, so wird Suche_weg mit dem neuen Ort aufgerufen. Zur weiteren Suchen muss am Schluss der neue Ort wieder aus dem Weg entfernt werden und die aktuelle Länge und die aktuelle Anzahl zurückgesetzt werden.

PROCEDURE Suche_Weg (Ortskennung)



Bewertung des Backtracking-Verfahrens:
Um nun die Güte des Algorithmus zu bewerten, versucht man den Zeitbedarf für die Suche abzuschätzen. Hierzu überlegt man sich wieviele Knoten der Suchbaum enthält. Denn diese Knoten müssen alle betrachtet werden. Hierbei geht man meistens vom schlimmsten Fall aus (Worst - Case).

Betrachten wir nun den Suchbaum. Von jedem Ort gehen mehrere Verzweigungen aus. Maximal kann zu jedem Ort eine Verzweigung führen. Das heisst: Bei n Orten gibt es maximal n - 1 Verzweigungen. Um die Rechnung zu vereinfachen, ersetzen wir n-1 durch n. Da ein Weg schlimmsten Falls durch alle Orte geht, ist die Höhe des Baumes höchstens n. Damit enthält der Suchbaum maximal n^{n-1} Wege mit n Knoten. Insgesamt müssen damit $n^{n-1} \cdot n = n^n$ Knoten betrachtet werden



n	n^n
1	1
2	4
3	27
4	256
5	3125
6	46656
7	823543
8	16777216
9	387420489
10	10000000000
…	
20	$1{,}048576 \cdot 10^{26}$
…	
100	$1 \cdot 10^{200}$
…	
1000	$1 \cdot 10^{3000}$

Betrachtet man das Verhalten der Zahl n^n, so stellt man fest, dass diese Zahl sehr schnell ansteigt. Auch bei schnellen Rechnern wird es ab einer gewissen Ortszahl problematisch.
Verwenden wir nun einen Rechner mit einer Taktfrequenz von 400 MHz. Pro Takt kann ein Rechner eine Anweisung durchführen. 400 MHz Taktfrequenz bedeutet also $400 \cdot 10^6$ Anweisungen pro Sekunde. Beinhaltet unser Routefinder nun zum Beispiel 1000 Orte (dürfte bei europaweiten Routefindern sicher überschritten werden), so beträgt die benötigte Suchzeit:

$$\frac{1 \cdot 10^{3000}}{400 \cdot 10^6}\text{s} = 2{,}5 \cdot 10^{2991}\text{s} = 4.16667 \cdot 10^{2989}\,\text{min}$$

$$= 6{,}9444 \cdot 10^{2987}\text{h} = 2{,}89352 \cdot 10^{2986}\text{d} = 7{,}92745 \cdot 10^{2983}\text{a}$$

Das Universums ist ungefähr 10^{10} Jahre alt.
Dies zeigt, dass dieses Verfahren auch mit wesentlich schnelleren Rechnern nur für kleine Ortszahlen verwendet werden kann.

In Wirklichkeit sind die Bäume natürlich nie mit so vielen Verzweigungen versehen und auch die Wege sind im allgemeinen kürzer. Trotzdem zeigt die obere Abschätzung den rapiden Anstieg der Komplexität. Für große Ortszahlen ist das Verfahren völlig ungeeignet. Auch mit Rechnern künftiger Generationen wird das Problem nur unwesentlich verändert.
Daher ist es wichtig, das Verfahren zu verbessern.
Wie lässt sich die Komplexität des Algorithmus verkleinern. ?

Verbesserung des Backtracking - Algorithmus:

Bisher endet der Suchbaum, wenn der neue Ort bereits besucht wurde oder der Zielort ist. Nachdem der kürzeste oder schnellste Weg gesucht wird, kann man den Suchbaum auch beenden, wenn die aktuelle Länge größer als die aktuelle Länge ist. Damit werden aus dem Suchbaum zusätzlich Knoten entfernt und der Suchaufwand wird geringer.

Diese Verfahren bringt jedoch nur geringe Verbeserungen. Gravierende Änderungen bringt erst ein völlig anderer Algorithmus.