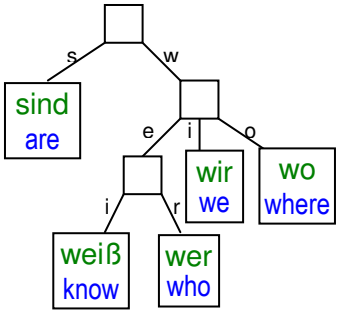
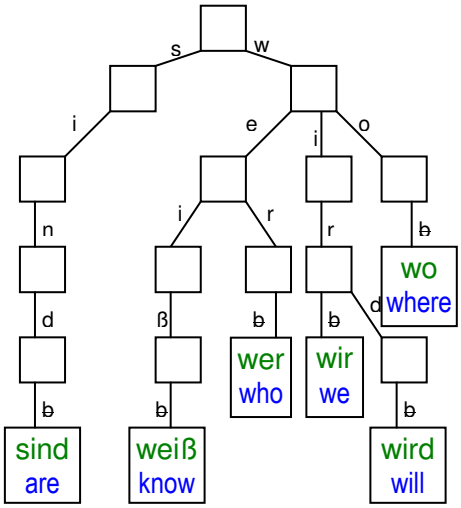
Algorithmen und Datenstrukturen

2. Praktikumsaufgabe: Trie — Abgabetermin: 12.1.15

Ein alphabetischer Suchbaum (trie, sprich: trai) ist ein Vielweg-Baum zur Speicherung von Assoziationen Key → Value, wobei Key ein Wort (string) ist. Er kann z.B. als Wörterbuch verwendet werden. Die Key-Value-Paare werden nur in den Blättern gespeichert (hohler Baum). Die inneren Knoten des Baumes dienen nur als Wegweiser für die Suche. Das Suchwort wird von links nach rechts durchlaufen. Bei jedem Zeichen wird verzweigt (neuer Knoten):
Besteht z.B. das Wörterbuch nur aus den Wortpaaren (wer,who), (weiß,know), (wo,where), (wir,we), (sind,are), so kann folgender Suchbaum verwendet werden:



Ein Key kann auch Anfangsstück (Präfix) eines anderen sein, z.B. wenn wir das Wortpaar (wird,will) hinzunehmen. Deshalb ist obige Darstellung noch nicht brauchbar. Wir schließen daher jedes Wort durch ein Trennzeichen ƀ ab, das die Verzweigung zu einem Blatt markiert:



Die Blätter haben also einen anderen Typ als die inneren Knoten: Sie sind – wie die inneren Knoten – eine Spezialisierung (Erweiterung) einer abstrakten Knotenklasse. Während die inneren Knoten markierte Kindzeiger haben, enthalten die Blätter einen Record (im Bild ein Suchwort = Schlüssel, dem ein Wert `T` zugeordnet ist, z.B. das zugehörige englische Wort. Überlegen Sie sich, ob Sie den Schlüssel überhaupt abspeichern müssen.)

Entwickeln Sie eine C++-Implementierung des Tries mit der Schnittstelle eines Suchbaums, wie er in der Vorlesung behandelt wurde. Die Daten sind Paare, bestehend aus einem String als Schlüssel und beliebigen Nutzdaten `T` (vergl. `map` in der STL):

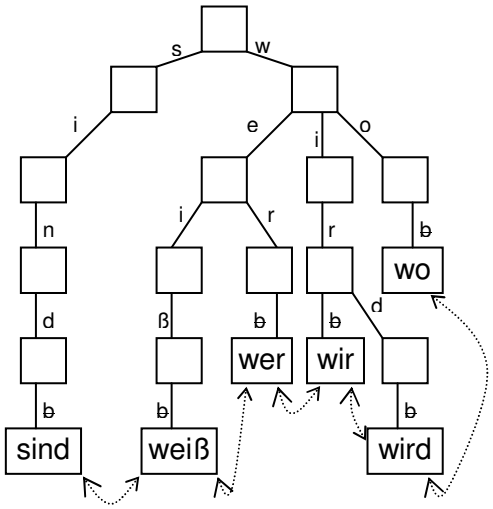
```
template <class T, class E=char>
class Trie {
public:
   typedef basic_string<E> key_type;           // string=basic_string<char>
   typedef pair<const key_type, T> value_type;
   typedef T mapped_type;
   typedef ... iterator;
   typedef std::reverse_iterator<iterator> reverse_iterator;
   bool empty()const;
   iterator insert(const value_type &);
   void erase(const key_type& value);
```

```
    void clear();                                                  // erase all
    iterator lower_bound(const key_type& testElement);             // first element >= testElement
    iterator upper_bound(const key_type& testElement);             // first element >  testElement
    iterator find(const key_type& testElement);                    // first element == testElement
    iterator begin();                                              // returns end() if not found
    iterator end();
    reverse_iterator rbegin();
    reverse_iterator rend();
};
```

Definieren Sie die Knotenklasse lokal zum Trie (innere Klasse).
Verwenden Sie für die Sohnzeiger eines Knotens eine geordnete lineare Liste (Implementierung aus der Vorlesung oder STL-Klasse). Dadurch sind die Blätter von links nach rechts aufsteigend sortiert.

Natürlich gehört zu der Implementierung auch ein Hauptprogramm, das einen Trie anlegt und **alle** Trie-Operationen testet. Es soll ein Menü anbieten, mit dem man den Trie verändern kann und die Ergebnisse ausgeben kann. Sehen Sie auch eine Ausgabe des gesamten Tries in einer primitiven Darstellung vor.

Es gibt nur einen sinnvollen Iterator. Er durchläuft die Blätter in der vorgegebenen Ordnung. Eine ineffiziente Implementierung verwendet Vaterzeiger in jedem Knoten (vergleiche Vorlesung und STL). Da keine Ausgleichoperationen benötigt werden, wird der Vaterzeiger ausschließlich für den Iterator benötigt. Eine elegantere Implementierung verwendet "Fädelung". Hierbei werden die unbenutzten Zeiger in den Blättern verwendet, um auf die Vorgänger und Nachfolger zu zeigen.



Die Konstruktion der Fädelungszeiger ist komplizierter. Beim Einfügen (Suchen = Abstieg im Baum) merkt man sich den letzten Knoten, in dem noch eine weitere Verzweigung nach rechts möglich war. Vom nächsten (rechten) Sohn dieses Verzweigungspunktes sucht man den linkesten Enkel (`slideLeft`). Dieser ist der Nachfolger. Analog ist der rechteste Enkel des linken Sohnes der Vorgänger.

Gruppen mit zwei StudentInnen können sich auf die Implementierung eines Vorwärtsiterators beschränken und diesen mittels Vaterzeigern und verallgemeinertem Inorder-Durchlauf realisieren (siehe binäre Suchbäume der Vorlesung).

Gruppen mit drei StudentInnen müssen entweder

- gefädelte Tries mit Vorwärts- und Rückwärtsiteratoren (ohne Vaterzeiger) implementieren *oder*
- statt Iteratoren einen *Patricia-Trie*[1] implementieren:
  Wie obiges Beispiel zeigt, haben viele innere Knoten nur eine Tochter (zumindest bei natürlichsprachigen Schlüsseln). Dadurch kommen weit mehr innere Knoten (ohne Nutzdaten) vor als Blätter. *Patricia-Bäume* vermeiden diese Speicherplatzverschwendung. In ihnen werden die Knoten ohne Verzweigung eingespart und dafür die eingesparte Pfadlänge (= Anzahl der Knoten zwischen zwei Verweigungsknoten) im Vaterknoten vermerkt. Außerdem werden die Daten (Schlüssel und Nutzdaten) in allen Knoten gespeichert. Alternativ speichert man in den inneren Knoten nur den (übersprungenen) gemeinsamen Präfix und die Daten in den Blättern.

[1] H. Reß, G. Viebeck: Datenstrukturen und Algorithmen, S. 347 ff,
G. Saake, K.-U. Sattler: Algorithmen und Datenstrukturen, S. 381 f