Abgabe ESA 4 Ivo Valls

Zeitkomplexität der Methoden `Insert(Knoten<T> knoten)`, `Search(Knoten<T> knoten)`, `Size()`, und `MinElement()`

Insert

Da ein Knoten immer als Blatt eingefügt wird, muss im Worstcase bis zur Höhe h des Baumes gegangen werden. Im Bestcase ist der Baum leer, damit ergibt sich für Insert eine eine Komplexität von O(log n) im durchschnitt O(n) im Worstcase (Der baum ist eigentlich eine Verketteteliste) und O(1) im Bestcase.

Search

Da bekannt ist in welche Richtung größere/kleinere Werte liegen müssen Skaliert die Zeit für das Suchen mit der Höhe also O(h). Im worstcase ist der Baum zu einer Verlinkten liste entartet also wieder O(n).

Size

Durch geeignete Implementierung kann die Komplexität auf O(1) vermindert werden, indem man sämtliche manipulationen am Baum nur über eine Schnittstelle zulässt und nicht Knoten frei aneinander hängen kann wie hier. Im vorliegenden fall müssen sämtliche Knoten traversiert werden, es ergibt sich Somit eine komplexität von O(n)

MinElement

Hier gilt analog zu Search wieder das selbe, durch die Ordnung im Baum ergibt sich wieder eine Zeitkomplexität von O(log n) bzw O(n) im Worstcase

Könnte eine solche Datenstruktur die Methoden frequency und mostFrequent der

Programmieraufgabe „Vorbereitung Suchmaschine“ effizienter machen? Erklären Sie Ihre Antwort.

Frequency

Ja, wenn man den Key wert des Trees etwas komplexer gestaltet und doppelte einträge vermeidet. Der Baum könnte die Wörter als Key verwenden und eine weitere Property enthalten welche die Anzahl (Count) enthält. So müsste beim auslesen der Frequenz lediglich die Frequenz ausgelesen werden und die Suche ginge schneller werden. Damit hätte man ein Dictionary<Wort,Anzahl> welches man als Lookup benutzen könnte.

mostFrequent

Ja. Wenn man das für Frequency generierte Dictionary einmalig sortiert hat man auch hier immer O(1) zugriff auf mostFrequent