Oblig 4 233 Cecilie Lyshoel

A)

Definisjoner:
-Hashtable: Er et array der hash-nøkler og hash-verdier er lagret

-Hashfunction: Gjør search keys til integer hash-kode som refrerer til annen verdi slik at det blir en kobling.

-HashCode: Er en integer-verdi som fungerer som indeks til referanseverdi i en hash function.

-Collosion: Kollisjon oppstår når man legger en søkenøkkel inn i lokasjon i en hash table som allerede er i bruk. Hver enkelt søkenøkkel må ha sin egen lokasjon i hash table-en for at

-Bucket: i hastable blir nøkel og verdi-par lagret i bøtter og hashtable bruker nøkler til å komme frem til bøtten. Bøttene skal være unike.

Fordel med å benytte hashtable for å implementere dictionary fremfor en kjedet implementasjon:

Det er mye lettere å slå opp i en dictionary ved hjelp av hashtable fordi man slipper å sortere først, man ganske enkelet slår opp i dictionaryen basert på nøkkel. Rekkefølgen skal ikke ha noe å si i hashtable, men i en lenket implementasjon er dette viktig for at man effektivt skal kunne slå opp i dictionaryen. Hashtable er også en fordel fordi man har håndtering av kollisjoner og sørger for at ikke flere nøkler mapper til samme verdi.

Hvordan kan kollisjoner håndteres (gitt at hashtable er mindre enn antall element vi skal lagre):

Man kan bruke seperate chaning for å håndtere kollisjoner ved at man endrer strukturen på hele hashtabellen. Man vil da få en liste med bøtter. Bøtten vil også inneholde alle oppføringer som mapper til samme element i listen som der man refererer til. Man kan igjen linke bøttene som noder. Har man en kollisjon der man prøver å sette inn ulike verdier i etter samme indeks, så kan men heller la disse verdiene linke til hverandre i samme bøtte innenfor samme lokasjon i listen. Hashtabellen blir på denne måten ikke full, men kan bli treg og lite effektiv (selv om man sorterer etter beste evne)

Hvordan kan kollisjoner håndteres (gitt at hashtable er stor nok for antall element vi skal lagre):

Man kan bruke open addressing der man i utgangspunktet ikke har dårlig plass i hashbabellen, men siden alle elementer blir lagret i hashtabellen i seg selv må antall nøkler være mindre enn eller det samme som størrelsen av hashtabellen. Da finner man en ledig plass i hashtabellen til det man prøver å sette inn i hashtabellen. Når man har funnet en ledig plass, bruker man denne i stedet for å referer til ny oppføring. Etter hvert som hashtabellen begynner å bli full vil det ta lenger til å finne en ledig plass i tabellen. Man kan bruke litt ulike teknikker når man benytter seg av en open addressing fremgangsmåte. Linær sondering, Kvadratisk sondering og dobbel hashing.

B)

Definisjon:

-Binært tre:
Et tre består av noder der man har foreldrenoder og barnnoder. I et binært tre så kan hver foreldrenode maksimalt ha to barnnoder, høyre og venstre. Har en node ingen barn klasses den bladnode. Rotnoden er den øverste noden som gener seg ut og har ingen foreldrenode over seg. Barnnodene til rotnoden kan deles inn i høyre og venstre og om disse igjen har egen barnnoder så vil dette utgjøre egne subtrær.

-Binært søketre:
Et binært søketre er et binært tre der man sammanligner verdien av nodene. Har foreldrenoden større verdi en barnnode, så er barnnode til venstre. Er foreldrenodes verdi mindre enn barnnode, så er barnnode til høyre for foreldrenoden. Verdi kan være tall der for eksempel 1 er mindre enn 3, eller alfabetisk der for eksempel C har lavere verdi enn V.

Preorden	R S U X T V W Y
Inorden	U X S R V T Y W
Postorden	X U S V Y W T R

Det binære søketret som foreleser har hentet fra wikipedia:

-Innsetting av 15:
Større enn 8, derfor høyre
Større enn 10, derfor høyre
Større en 14, derfor settes 15 inn i høyre subnode til 14 (siden det ikke er noen subnode her fra før)

-Innsetting av 2
Mindre enn 8, derfor venstre
Mindre enn 3 derfor venstre
Større enn 1, derfor settes inn i høyre subnode til 1 (siden det ikke er noen subnode her fra før)

-Innsetting av 5:
Mindre enn 8, derfor venstre
Større enn 3, derfor høyre
Mindre enn 6, derfor venstre
Større enn 4, settes derfor inn i høyre subnode til 4 (siden det ikke er noen subnode her fra før)

-Sletting av 13:
Her er det ingen subnoder, kan derfor fjernes uten videre

-Sletting av 6:
Node 6 har to subnoder.
Skal man slette node 6, må subnodene innsettes på nytt. Node 6 har to barn, og etter å ha listet påå

tabellen etter inorden så er det den noden som kommer etter 6 være den verdien som flyttes opp og eratter verdien i node 6 og noden den opprinnelig var i skal slettes. I dette tilfellet er det 7 som skal erstatte 6 og noden hvor 7 var i det opprinnelige treet vil bli fjernet. Inorden for dette binære søketreet er [1, 3, 4, 6, 7, 8, 10, 13, 14].

Kjøretidsanalyse av innsetting og sletting i algoritmen for lenket implementasjon av binært søketre:

searchNode(): O(n) worst case.
deleteNode(): O(n) worst case
insertNode(): O(n) worst case
getSuccessor(): O(n)
getRootNode(): O(1)

C)

Kjøretidsanalyse av haugsorteringsalgoritme:

Heap sort : O(n log n)