OBLIG 4 – Hallvard Moan Kristiansen

OPPGAVE A.

- Spredetabell (hashtable):
  Spedetabell brukes for å beregne en indeks i et array hvor elementet vil bli satt in og søkt. Denne spredetabellen har et stort tildelt lagringsminne, og den har en hash-funksjon som er valgt til å alltid returnerer verdier som ligger et sted innenfor det tilgjengelige minnet. For å lagre en verdi, er nøkkelen «hashed», og verdien er plassert på stedet tildelt av «hashen». For å se en verdi oppgitt av nøkkelen, trengs det kun å «hashe» nøkkelen for å få tilbake plasseringen av den tilsvarende verdien. Ved bruk av en god hash-funksjon får man god hastighet. Den gjennomsnittlige tiden det tar å søke etter et element i en spredetabell er O(1).
  Source: (https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/)

- Spredefunksjon (hashfunction):
  En spredefunksjon er den funksjonen som kartlegger verdier av variabel størrelse til verdier med fast størrelse, kjent som hashverdiene. Dette kan lettere forklares ved et eksempel, så om man antar at hver bokstav i alfabetet har en tilordnet verdi; (f.eks. 'A' = 1, 'b' = 2, 'c' = 3 etc.) vil man da kunne skrive en enkel hash-funksjon for teksten som bare legger til alle verdiene for hver karakter. Så teksten "abc" ville ha en hashverdi på 1 + 2 + 3 = 6, og "adc" ville være 1 + 4 + 3 = 8.

- Spredekode (hashcode):
  Spredekode returnerer en integer verdi, generert av en hashing-algoritme. De objektene som er like, vil returnere samme hash-kode. Man kan si at Spredekode er en "hjelper" -funksjon som tillater samlinger som HashMap å gjette hvor de lagret en gjenstand. Den tar altså et objekt og sender ut en numerisk verdi.

- Kollisjon (collision)
  Ulike «inputs» kan føre til at vi får samme «output» hash-verdi. Dette hender når inntastingsstørrelsen er større enn størrelsen på hashverdien. Om vi bruke eksempelet fra spredefunksjonen over, kan man bevise kollisjon ved å vise til at alle bokstavene har en verdi, og om man da skriver flere bokstaver sammen abc (1+2+3=6) og aaac (1+1+1+3=6) vil man få tilbake samme verdi som er 6, selv om vi bruker ulike bokstaver og rekkefølge. Dette er kjent som kollisjon og det er ønsket at hashing algoritmer blir utformet for å minimere antall kollisjoner på tvers av disse «outputsene».

- Bøtte (bucket)
  Et Hashtable internt inneholder buckets hvor der den lagrer nøkkler / verdi parene. Hashtable bruker nøkkelens hashcode for å bestemme hvilken nøkkel og verdi par som skal kartlegge.

  Buckets er placeholders for keyvalue par eller objekter. Størrelsen av samlingen bestemmer antall bøtter som blir oprettet. Antallet bøtter blir nesten doblet dersom den nåværende størrelsen av bøtten når 0.75% av lasten. Buckets blir lagret i heapen.

Blir mange teorispørsmål, men teori er også viktig.
Du får inntil 15 poeng for begrepsforklaringene (3 for hvert). 5 for å forklare fordelen med spredetabell, og 10 for kollisjonsforklaring

Hvorfor er det en fordel å benytte en spredetabell (hashtable) for å implementere en ordbok (dictionary) fremfor en kjedet struktur?
Det som er fordel ved å benytte en spredetabell fremfor, en kjedet struktur er at man slipper å iterere seg frem for å finne en verdi. Mens med spredetabell kan man hente objektet ut direkte ved å peke på verdien i «memory», noe som sparer mye tid. Det positive med kjedet struktur er om du har mange verdier så vil den gå gjennom det automatisk, mens med spredetabell slipper man holde styr på verdiene. Spredetabell (o(n)), kjedet struktur (o(n^2).

Hvis spredetabellen er mindre enn antall element vi skal lagre, får vi kollisjoner etter hvert. Forklar hvordan kollisjoner kan håndteres. Forklar også hvordan kollisjoner kan håndteres hvis spredetabellen har plass til alle elementene vi skal lagre.

Dersom det oppstår at spredetabellen er full og man opplever kollisjon, kan man utvide spredetabellen for å få sprede-funksjonen til å re-distribuere spredekoder til elementene. Dette gjør det mulig for at de videre-distribuerer det og får det lagret på en ny plass.

Dette er også noe en kan gjøre selv om tabellen ikke er full, men det vil være u optimalt, ettersom spredefunksjonen kan peke til samme bøtte over flere spredekoder. Man kan løse dette ved å re-hashe.

DELOPPGAVE B.

Et binært tre er en ikke-lineær datastruktur hvor hver knute kan ha maksimalt 2 barn noder.

Binært søketre er et binært tre der en node har en verdi som er større enn alle verdier i den venstre deltre og mindre enn alle verdier i høyre undertreet.

Rekkefølger:
InOrden: U – X – S – R – V – T – Y – W.
PreOrden: R – S – U – X – T – V – W – Y.
Postorden: X – U – S – Y – V – W – T – R.

Hvilken rekkefølge blir nodene skrevet ut ved preorden, inorden og postorden gjennomgang.
(6 poeng) Gitt følgende binære søketre (hentet fra wikipedia):

Man setter inn 15, 15 > 8 høyre, 15 > 10 høyre, 15 > 14 høyre, 15
Man setter inn 2 2 < 8 venstre, 2 < 3 venstre, 2 > 1 høyre, 2
Man setter inn 5 5 < 8 venstre, 5 > 3 høyre, 5 < 6 venstre, 5 > 4 høyre, 5
Man sletter 13 13 > 8 høyre, 13 > 10 høyre, 13 < 14 venstre, slett 13
sletter 6 6 < 8 venstre, 6 > 3 høyre, 6. 4 vil ta plassen til 6 når man slettet 6.

Kjøretidsanalyse: //heapsorten er O(n) når man kjører en kjøretidsanalyse, ettersom den kun kaller på seg selv. Altså den er rekusiv.