Håkon Lervåg(599025), Vetle Færø(600872) og Fredrik Birkeland(591345)

Oblig5 – DAT102

Oppgåve 1

a) Et balansert BS-Tre vil seie at alle nivå, unntatt det nedste er fulle. Det vil sei at alle noder, unntatt det foreldra til det nedste nivået, har to barn.

b) Dette BS-treet er ikkje balansert sida me har ein mykje lengre sti i venstredel, og alle nivå unntatt det nedste ikkje er fulle. Etter ein høgrerotasjon om rett node, får me 8 som rot og 17 som høgre underbarn, og 3 som venstre underbarn til 8. Dei to barna til 17 blir 11 og 30, mens barnet til 3 blir 1.

c) Dette BS-treet er ikkje balansert fordi alle nodne, unntatt det siste, ikkje er fulle, og fordi høgreside er lengre enn venstre. Etter ein venstrerotasjon er 30 den nye rota. Venstrebarn til rota er 17, venstrebarn til 17 er 8 og venstrebarn til 8 er 3. Det er ingen høgrebarn på venstreside. Høgrebarn til rota er 40, høgrebarn til 40 er 45. det er ingen venstrebarn på høgreside.

d) Det er viktig at eit BS-tre er balansert fordi det har veldig mykje på tidsbruken å gjere. Dersom me har eit ubalansert tre, vil me eventuelt måtte søke gjennom mange fleire element for å finne riktig element, enn visst me har eit balansert tre.

Oppgåve 2

a) Ved å utføre ein breidde-først gjennomgang av grafen med c som startnode, får me rekkefølga og køen c, e, f, b, d, a

b) Ved å utføre ei djupne-først gjennomgang av grafen med c som startnode, får me rekkefølga og stabelen c, e, d, a, b, f

c) Mengde E (edges) = {{a, d}, {d, b}, {d, e}, {b, e}, {b, f}, {e, f}, {e, c}, {c, f}}
Mengde V (vertex) = {a, b, c, d, e, f}

Nabomatrise

	a	**b**	**c**	**d**	**e**	**f**
a	0	0	0	1	0	0
b	0	0	0	1	1	1
c	0	0	0	0	1	1
d	1	1	0	0	1	0
e	0	1	1	1	0	1
f	0	1	1	0	1	0

Naboliste

Node	Naboar
a	d
b	d, e, f
c	e, f
d	a, b, e
e	b, c, d, f
f	b, c, e

Oppgåve 3

a)

Hashing er omgjøring av einkvar data (String, int, long...) til ein ny representativ verdi, ved hjelp av hash-funksjoner, og lagrer desse i såkalte «hash-tables». Data-en vår kan så raskt hentes tilbake ved å bruke hashes, vår representative verdi, til å begrense søkeområdet vårt i hash-tabellane. Hash-funksjoner er altså einkvar funksjon som kan kartlegge data opp mot ein representativ verdi; ein hash.

Ein hash-collision er når to ulike hash-inputs peker til og produserer same output. Dette forekommer ettersom hash-funksjoner har uendelig stor input-lengde, men ei set output-lengde. Output-lengden varierer frå kva type hash-funksjon man bruker. Ein sekvens av kollisjoner kalles «clustering». I «Linear probing» har man to typer clusters, primary og secondary clustering. Primary clustering er når to inputs genererer samme output og det former seg «grupper» i hash-tabellen. Desse gruppene forekommer ettersom at visst det ikkje er ledig i den første og tiltenkte plassen i tabellen, går man videre til neste plass i tabellen og sjekker om denne er ledig osv. Dette kan gje oss problemer ettersom det ved lengre clusterings kan begynne å ta tid å finne ledig plass. Secondary clustering det samme som primary clustering, men dersom to inputs ikkje har samme første posisjon i tabellen (skal initially bli plassert på samme plass), så vil dei ikkje havne i samme kollisjonskolonne. Dette gjer også at secondary clustering ikkje er like ille som primary clustering.

Kriteriene for ein god hash-funksjon er bestemt av fire hovudkriterier, funksjonen er 100% bestemt av dataen som skal bli hashet, funksjonen bruker all input-dataen, funksjonen genererer veldig ulike verdier for all data og sist men ikkje minst at funksjonen distribuerer dataene heilt likt utover heile setet med mulige hash-verdier.

b)

	Linear Probing	**Kjedalister**
0	VV50000	VV50000
1	EL65431	ZX87181
2	ZX87181	
3		
4	TA14374	EL32944
5	UV14544	
6	EL32944	
7	EL47007	EL47007
8		
9		

c) Java-kode

d) Dei to strengene gir ulike hashcodes fordi me ikkje override'er hashcode metoden. Dersom me ikkje gjer dette ender me opp med øydelagde objekt ettersom hashcode vil ta i bruk dei samme feltene som sin equals metode. Dersom me berre overskriver equals vil me ende opp med at nokre objekt behandles likt andre objekter, der hashcode derimot standard

behandler alle objekt ulikt. Dette kan me derimot fikse ved å generere ein hashcode metode og overskrive denne.

e) Java-kode
Me fekk visualisert korleis Hashset er veldig mykje raskere enn binærsøk. Der Hashset brukte 0,01s på å søke gjennom 10.000 tall, brukte binærsøk ca 3,5s.