Porteføljeopgave 1

Algoritmer og datastrukturer

2024

Rikke Aa. Boysen

29. November

1 Opgave 1

Opgaven går ud på at skrive en algoritme, der tager en string som input og returnerer det oftest forekommende ord. algorritmen fungerer ved først at fjerne kommaer og punktummer ved at erstatte dem med blanke felter. Herefter konvertere den store bogstaver til små ved hjælp af deres ACII værdi.

Her efter gennemløbes strengen og når den når til et blank charchter tolkes det som et ord der bliver indsat i et map. Dette holder styr på hvor mange gange det enkelte ord optræder. Det ord der optræder flest gange vil herefter blive retuneret.

Koden kan ses nedenunder.

```
    string MostCommon(string s)
{
    //Convert the whole string to lowercase and remove "," & "."

    for (int i = 0; i < s.length(); i++)
    {
        if (s[i] == ',' || s[i] == '.')
        {
            s[i] = ' '; // Replace punctuation with space
        }
        else if (s[i] >= 'A' && s[i] <= 'Z')
        {
            s[i] = s[i] + ('a' - 'A'); // converts uppercase ASCII values to their corresponding l
        }
    }

    // Define variabels used
    map<string, int> result;
    string common;

   string word = "";
    for (int i = 0; i < s.length(); ++i)
    {
        if (s[i] != ' ') //While the charchter is not a space the chachter is input into the "word
        {
            word += s[i]; // Build the current word
        }
```

```
        else if (!word.empty()) // When the word is done add it to the result map
        {
            result[word]++;
            if (common.empty() || result[word] > result[common]) {
                common = word;
            }
            word = ""; // Reset the word
        }
    }

    // Return the most common word
    return common;
}
```

1.1 Store O for metoden

For at finde Store-O tidskompleksiteten for fuktionen startes der ved første for-løkke. Denne løber hele strengen igennem, hvilket vil sige fra 0 til N og vil derfor have en store O på $O(N)$.

Kigger man på næste forløkke er det samme gældende. Denne løber også fra 0 til N og vil have en store O på $O(N)$.

Da begge løkker har samme tidskompleksitet vil derfor være $O(N)$.

2 Opgave 2

3 Opgave 3

Denne opgave tager udgangspunkt i et binært søgetræ som ikke er et AVL-træ

3.1 In order traversering

Først kigges der på hvordan noderne vil blive besøgt i en in order traversering. Dette er em metode der gå udpå at man først besøger venstre side, herefter roden og til sidst højre side af træet. Man starter ved at besøge den mest venstre liggende node, som i dette tilfælde er 1. Her efter går man videre til 2 og derefter ned af den højre side af den vesntre gren. Når dette er helt besøgt går man til roden. Når dette er besøgt forsætter man på samme måde, men den højre gren. Man besøger først den mest venstre liggende og arbejder sig fra venstre mod højre. De noder man ville besøge villde derfor bilve:

1 - 2 - 3 - 9 - 11 - 17 - 25 - 57 - 90 - 13

3.2 Level order traversering

Level order traversering er en metode hvor man besøger noderne fra venstre mod højre, men startende fra toppen. Det vil sige at den første node besøgt ville blive 11, hvor efter man går et nivuea ned og starter fra venstre. Her villde det så blive node 2 der blev besøgt først og derefter node 13. Den række følge noderne ville blive besøgt i med level order traversering er:

11 - 2 - 13 - 1 - 9 - 57 - 3 - 25 -90 - 17

3.3 Internal path length

Den "internal path length" regnes som dybden af alle noderne i træet, hvor man ikke tæller roden med, da den har dybde 0.

$2 * 1(2, 13) + 3 * 2(1, 9, 57) + 3 * 3(3, 25, 90) + 1 * 4(17) = 21$

Den " internal path length" er 21.

3.4 Omrranger til AVL træ

For at træet kan omrangeres til et AVL træ, så skal alle subtræerne opfylde at højdeforskellen mellem alle noderne i et subtræ er mindre eller ligmed 1. For at opnå dette kan man benytte sig af rotationer.

Den første rotation, man ville kunne lave er en højre rotation af node 57, så denne bytter plads med node 25. Dette ville nu gøre at det højre sub træ har en højde på 2

Man kan derefter lave en venstre rotation af node 13, så denne bytter plads med node 25. Dette ville give en højde i det venstre subtræ på 1 og en højde i det højre subtræ på 2. Da dette giver en forskel på $2 - 1 = 1$, ligesom i det venstre subtræ opfylder det højre subtræ nu kravene for at være et AVL træ. Træet vil komme til at se ud som følgende i fig. 1

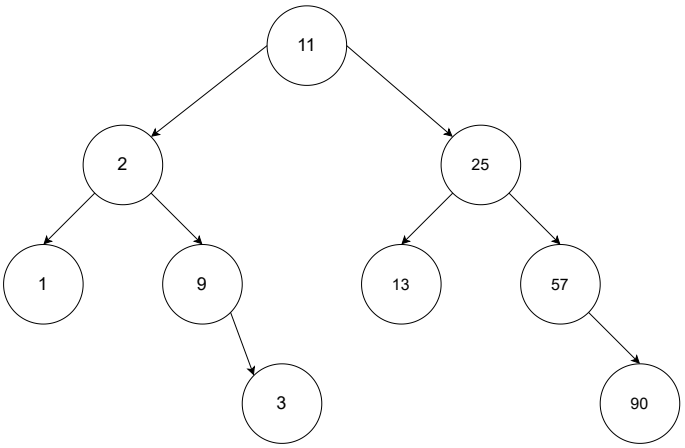

Figure 1: Omrangering til AVL-træ

3.5 Kunne træet have været et AVL-træ før den seneste operation?

Første udgangspunkt tages i indsættelsen af node 3. Hvis man går en operation tilbage og fjerner 3, vil højden forskellen i den venstre side vil være 0. Dette løser dog ikke problemet med ubalencen i ved node 13 i den højre gren, hvor højdeforskellen stadig er 2 og træet derfor ikke er et AVL træ.

Herefter kigges der på operationen delete af node 12. Dette ville give en højde forskel i den højre gren på 2 og træet vil derfor stadig ikke være et AVL træ.

En anden opereration indsættelsen af node 17. Hvis man undlader dette ville højden i det venstre subtræ være 0 og i det højre subtræ være 2, hvilket ville give en højde forskel på 2 og stadig ikke være et AVL træ.

For at træet skulle have været et AVL træ skulle man muligehvis undlade 2 operationer fx Delete af node 12 og indesættelsen af node 17, da dette ville give en højde forskel i den højre gren på 1.

4 Opgave 4

4.1 Post order traversering

I post order travesering besøger man noderne fra venstre mod højre dvs. først besøges den venstre gren og derefter den højre hvor man til sidst besøger noden.

Rækkefølge noderne ville blive besøgt i er:

1 - 8 - 15 - 5 - 12 - 10 - 22 - 20 - 28 - 38 - 45 - 50 - 48 - 30 - 40 - 36 - 25

4.2 Pre order traversering

I pre order traversering besøges først roden, derefter subtræet længst til venstre og til sidst det længst mod højre.

Rækkefølge noderne ville blive besøgt i er:

25 - 20 - 10 - 5 - 1 - 8 - 12 - 15 - 22 - 36 - 30 - 28 - 40 - 38 - 48 - 45 - 50

4.3 Internal path length

Den "internal path length" regnes som dybden af alle noderne i træet, hvor man ikke tæller roden med, da den har dybde 0.

$2 * 1(20, 36) + 4 * 2(10, 22, 30, 40) + 5 * 3(5, 12, 28, 38, 48) + 5 * 4(1, 8, 15, 45, 50) = 45$

Den " internal path length" er 45.

4.4 Er det et AVL-træ?

Som nævnt tidligere så får at være et AVL-træ så skal alle subtræerne opfylde at højdeforskellen mellem alle noderne i et subtræ er mindre eller ligmed 1. Højderne i træet er fundet som det kan ses på fig. 2.

Først startes der med det venstre subtræ. Her er højden forskellen begrenet til at være $(1 + 2 + 1) - 3 => 4 - 3 = 1$ hvilket opfylder kravet til AVL-træ. Herefter beregnes højdeforskellen fra det højre subtræ til at være $3 - (2 + 1) => 3 - 3 = 0$, hvilket også opfylder kravet for et AVL-træ. Det kravet er opfyldt i begge subtræer er træet et AVL-træ.

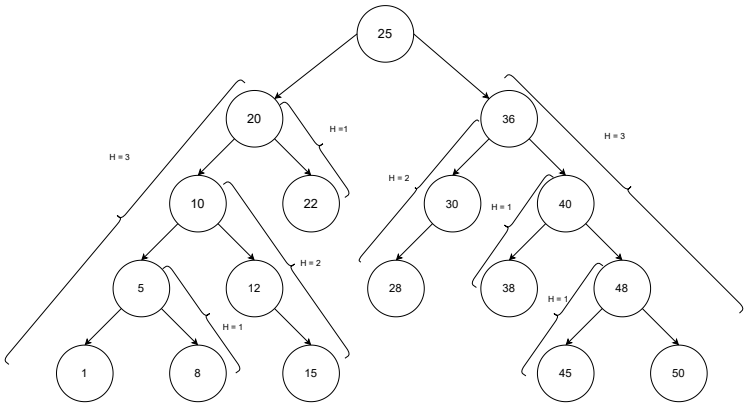

Figure 2: Udregning af højder på træ

5 Opgave 5

For at finde et minimum spanning tree gøres der brug af Kruskal. Først laves en tabel over alle kanter, deres vægt og hvilken kant de fører til. Denne kan ses i table 1.

Vægt	Udgangspunkt	Destination
1	0	1
1	0	4
1	3	6
1	3	7
2	0	5
2	9	10
3	5	8
3	5	2
3	7	11
4	6	7
4	8	9
5	5	9
5	10	11
6	0	2
7	4	8
8	5	10
8	6	11
8	1	2
14	2	3
20	5	6

Table 1: Soteret liste af kanter.

Herefter vælges en kant udfra den vægts. Der tjekkes om den danner en lukket sløjfe med andre noder, hvis dette ikke er tilfældet tilføjes den til rækken. Første kant der tages fat i er 0, denne har en vægt på 1 til kant 1. Her efter kigges der på næste med en vægt på 1, detter er i fra kant 0 til 4. Dette fortsættes med de næste i rækken og de tilføjes til rækken så længe de ikke danner en lukket sløjfe. Et tilfælde hvor sådan en sløjfe optsår kunne være ved 0 - 1 - 2 - 0.

Minimum spanning tree for grafen ville se ud som følgende.

0	1	3	7	6	4	5	10	2	8	9	11

Table 2: Minimum spanning tree

Den totale vægt findes ved at lægge vægten fra alle noderne sammen.

$1 + 1 + 1 + 1 + 2 + 2 + 3 + 3 + 3 + 4 + 5 = 26$

Træets totale vægt er 26

6 Opgave 6

Dijkstra's Algorithm benyttes ved at starte i den valgte startnode, hvorefter man finder den korteste afstand til de næste noder. Afstanden og den tidligere besøgte node indsættes i tabellen. Derefter fortæstter man med den node med lavest afstand. Hvis afstanden til en tidligere besøgt node viser sig at være mindre opdateres afstanden samt den tidligere besøgte node. Dette gøre indtil alle muligheder for hver node er blevet besøgt.

Slutkonfigurationen for grafen kan ses i table 3

v	known	d_v	p_v
A	True	0	0
B	True	5	A
C	True	3	A
D	True	9	E
E	True	7	G
F	True	8	E
G	True	6	B

Table 3: Slutkonfiguration for Dijkstras algoritme.