# Skriftlig eksamen i ADA5

# Algoritmer og datastrukturer

# 2. januar, 2024

Der er i alt 5 opgaver til denne eksamen.

Der afleveres ét og kun ét pdf-dokument. Kode, som du ønsker testet, **skal** være editérbar og derfor *ikke* i form af screenshots.

I programmeringsopgaverne kan du vælge frit mellem Java og C++.

Du må gerne skrive dine forklaringer på dansk.

**Opgave 1 (15 %)**

Hvad er Store O (Big-Oh) tidskompleksiteten for nedenstående metode? Begrund dit svar.

```
public static String myM(int N)
{
    int x = 0, y = 0;
    for (int j = 0; j < N; j++)
    {
        for (int i = N; i > 0; i=i/3)
        {
            for (int k = N; k > 0; k=k/2)
            {
                x++;
            }
        }
    }
    for (float v = 0; v < N; v+=Math.sqrt(0.001)) //C++: #include <math.h>
        y++;
    return x+" "+y;
}
```

En brugbar strategi for at løse opgaven, udover at analysere koden, vil være at lave nogle eksperimenter med koden, hvor du tæller antal af operationer, dvs. ser på returværdien for forskellige værdier af parameteren (N) for at skitsere et estimat af Store O tidskompleksiteten.

**Opgave 2 (15 %)**

Skriv en *rekursiv* metode/algoritme med følgende signatur:

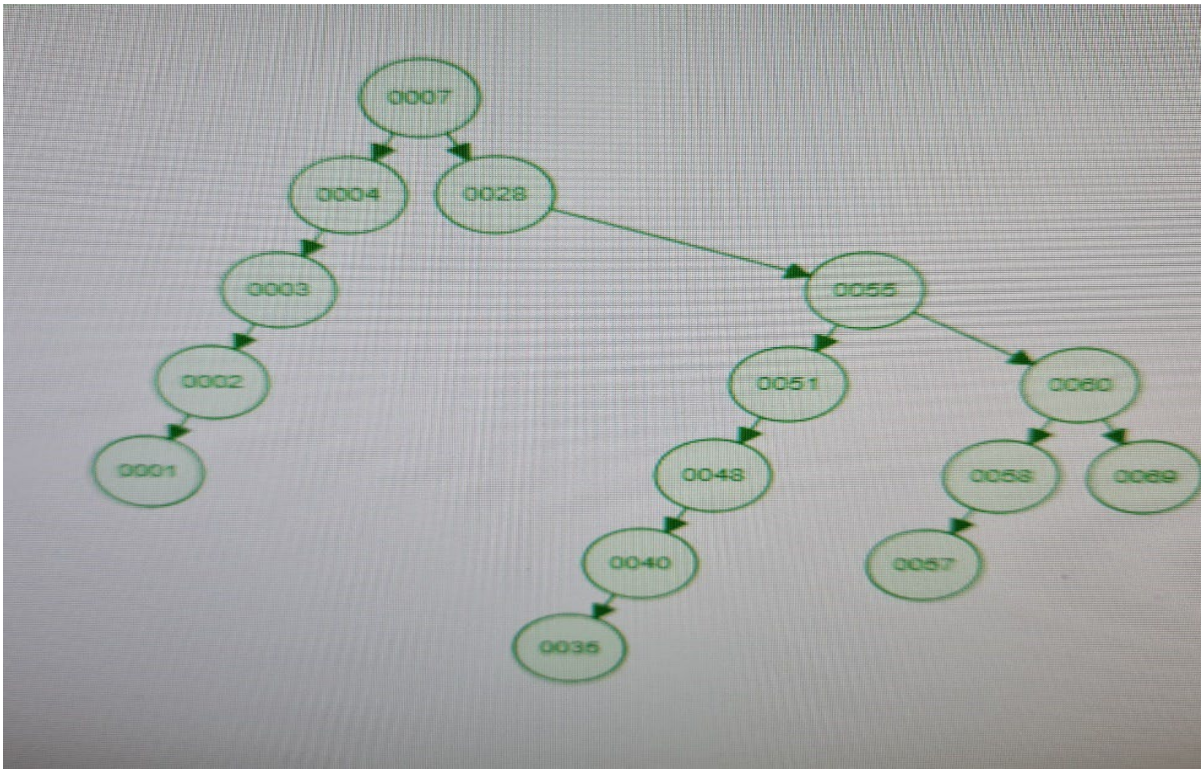
```
int sumDeleligMedTreOgOtte(int N)
```

Metoden returnerer summen af de tal, som er mindre end eller lig med N og er delelige med 3 eller 8. Er et tal deleligt med både tre og otte, tælles det kun med én gang.

Kaldt med parameteren 26 returneres 132 (3+6+8+9+12+15+16+18+21+24).

**Opgave 3 (40 %)**

Nedenstående figur forestiller et binært søgetræ.




Vi vil definere et begreb, som vi kalder en *gren,* og som karakteriseres på følgende måde. Det er en samling af tre noder, hvorom det gælder:

- Node X har ét barn.
- Node X har ingen søskende.
- Node X's barn har ingen søskende.
- Node X's barn har ét barn, som er et blad.

I ovenstående træ opfylder noderne 3 og 48 definitionen af X, og træet indeholder altså to grene.

Programmeringsopgaven (25 %) går ud på at skrive en metode til implementering i dit træ (det behøver ikke at være et binært søgetræ, men skal være et binært træ), som kan returnere antallet af grene i træet. I vores eksempel er det korrekte svar som nævnt to. Det antages, at roden *ikke* kan være en del af en gren.

Tip: Det kan anbefales at skrive en hjælpemetode med følgende signatur:

```
BinaryNode getOnlyChild(BinaryNode node)
```

Metoden returnerer parameterens eventuelle enebarn. Hvis der er to eller ingen børn, returneres `null`.

Husk kun at aflevere den kode du har tilføjet til dit træ og ikke også resten af træet.

Supplerende opgave 1 (10 %): hvad karakteriserer træet i figuren i øvrigt – hvad er træet, og hvad er det ikke? Derudover ønskes de sædvanlige numeriske oplysninger, og desuden: hvad er træets optimale højde, og hvordan kan den udtrykkes matematisk?

Supplerende opgave 2 (5 %): hvordan ville du gribe opgaven an, hvis du skulle transformere det binære søgetræ i figuren til en prioritetskø? Hvilke trin ville det kræve, og hvad vil tidskompleksiteten (Store O) for din løsning være?

**Opgave 4 (15 %)**

Nedenstående figur viser en tom hopscotch hash tabel:

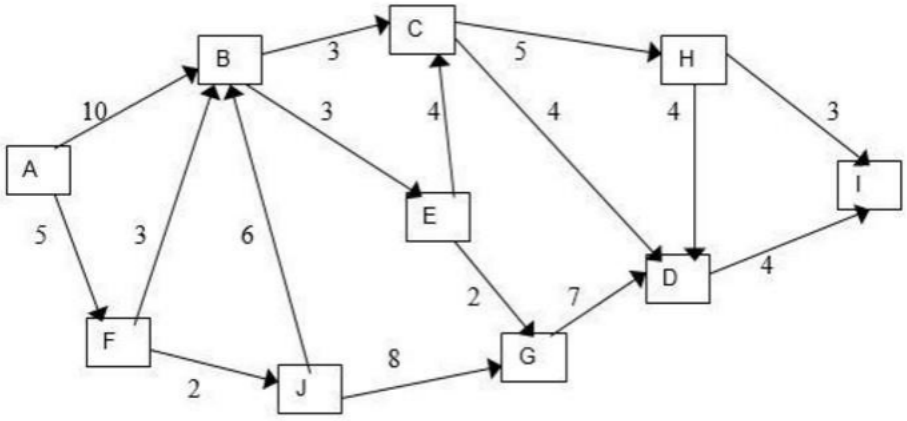
| Indeks | Værdi | Hop |
|---|---|---|
| 44 | | 0000 |
| 45 | | 0000 |
| 46 | | 0000 |
| 47 | | 0000 |
| 48 | | 0000 |
| 49 | | 0000 |
| 50 | | 0000 |
| 51 | | 0000 |
| 52 | | 0000 |
| 53 | | 0000 |
| 54 | | 0000 |
| 55 | | 0000 |
| 56 | | 0000 |
| 57 | | 0000 |

Opgaven går ud på at indsætte følgende elementer i tabellen og opdatere 'hoppen' (the hop) i overensstemmelse hermed:

| Værdi | Hasher til indeks |
|---|---|
| A: | 47 |
| B: | 51 |
| C: | 55 |
| D: | 51 |
| E: | 46 |
| F: | 51 |
| G: | 49 |
| H: | 53 |
| I: | 50 |
| J: | 51 |

Beskriv det problem, der opstår, hvis næste indsættelse i tabellen hasher til indeks 50.

**Opgave 5 (15 %)**




Nedenstående tabel viser startkonfigurationen for en traversering af ovenstående graf ved anvendelse af Dijkstras algoritme. Vis slutkonfigurationen for traversering af grafen startende i vertex A (du behøver ikke at medtage mellemkonfigurationerne i din besvarelse).

| **v** | **known** | **$d_v$** | **$p_v$** |
|---|---|---|---|
| A | false | 0 | 0 |
| B | false | ∞ | 0 |
| C | false | ∞ | 0 |
| D | false | ∞ | 0 |
| E | false | ∞ | 0 |
| F | false | ∞ | 0 |
| G | false | ∞ | 0 |
| H | false | ∞ | 0 |
| I | false | ∞ | 0 |
| J | false | ∞ | 0 |

Hvordan ville et minimum spanning tree for grafen se ud, hvis den var undirected? Angiv hvilken algoritme, der er anvendt og træets vægt.

Du behøver ikke at tegne grafen; men du skal, under alle omstændigheder, angive, i hvilken rækkefølge kanterne (edges) tilføjes. En god løsning kunne derfor være noget lignende nedenstående (som vedrører en anden graf end ovenstående):

| Nr. | Fra | Til | Vægt |
|---|---|---|---|
| 1 | C | F | 2 |
| 2 | H | J | 3 |
| 3 | B | D | 3 |