Skriftlig eksamen i ADA5

Algoritmer og datastrukturer

3. januar, 2025

Der er i alt 7 opgaver til denne eksamen.

Der afleveres ét og kun ét pdf-dokument og ikke noget andet. 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 selv om, du vil skrve dine forklaringer på dansk eller på engelsk.

Opgave 1 (10 %)

Skriv en rekursiv algoritme som kan tælle antallet af forekomster af et bestemt bogstav i en String.

Eksempel: kaldt med ”banana” og ’a’ returneres værdien 3.

Mulig signatur: `int countLettersInWord(String word, char letter, int index);`

Opgave 2 (15 %)

Betragt følgende algoritme/metode, hvori der indgår to for-loops:

```
public static long bigOh(double N)
{
    long x = 0; long y = 0;
    for (int i = 0; i < N; i++) //for-loop nummer 1
    {
         for (int j = 0; j < Math.pow(Math.log(N),2); j++)
         // log er ln (den naturlige logaritme)
         {
             for (int k = 0; k <= Math.sqrt(N); k++)
             {
                 x++;
             }
         }
         i += i;
    }

    for (long k = 0; k < N*Math.sqrt(N); k++) //for-loop nummer 2
          y++;

    System.out.println(x+" "+y);
    return x+y;
 }
```

Angiv algoritmens tidskompleksitet i Store-O notationen.
Hvis der indtræffer et skifte i det største antal operationer i henholdsvis for-loop nummer 1 og for-loop nummer 2 (i runde tal: værdierne af x og y), så angiv den værdi af N (cirka), hvor skiftet indtræffer.

I C++ kan man bruge metoderne `sqrt`, `log` og `pow` til kvadratrod, naturlig logaritme og potensopløftning. De findes i biblioteket <`cmath`>.

Opgave 3 (25 %)

Denne opgave går ud på at tilføje en funktionalitet til den udleverede kode *CriticalPath/KritiskVej.*

Funktionaliteten skal kunne finde den aktivitet, som har det største slæk (slack) og tillige angive varigheden af slækket.

Slæk kan defineres som det antal tidsenheder, der er til rådighed til at udføre aktiviteten minus aktivitetens eget tidsforbrug. I det udleverede data (se nedenfor) har aktivitet A 3 tidsenheders slæk – forskellen mellem event 1's længste aktivitet (B), som er 6 og A's tidsforbrug, som er 3.

```
1;A;3
1;B;6
1;C;4
2;D;5
3;E;4
3;F;1
4;G;2
4;H;7
4;I;1
5;J;4
```

Den korrekte aktivitet og slækkets varighed skal udskrives, og det rigtige svar er aktivitet **I** med et slæk på **6** tidsenheder.

Opgaven kan løses *uden* at ændre i de eksisterende statements i den udleverede kode (kun tilføjelser), og det anbefales.

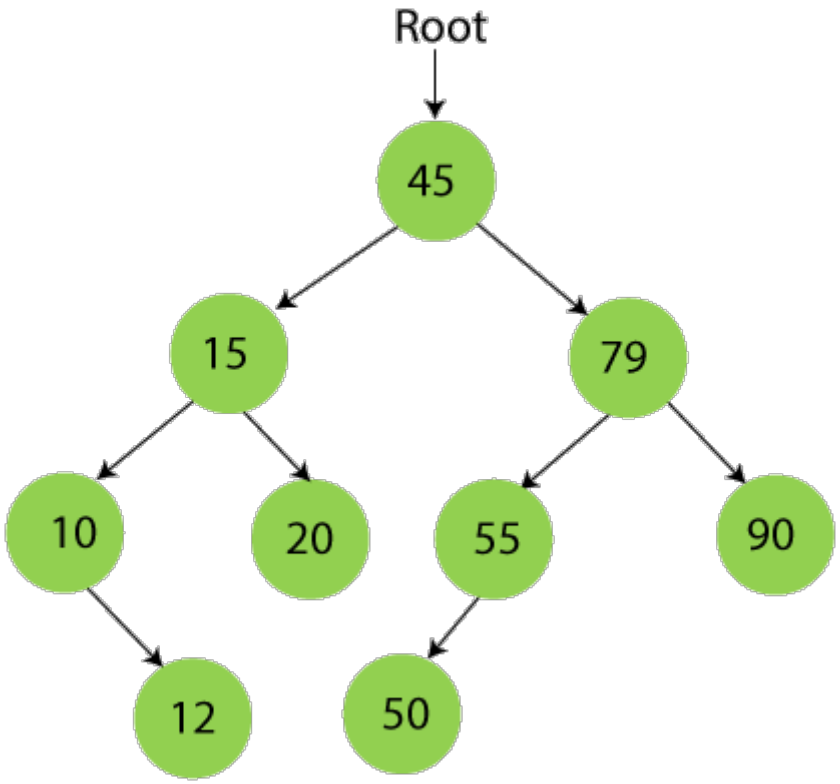
Opgave 4 (20 %)

Skriv en metode, som kan returnere ruten fra roden til en bestemt node i dit binære søgetræ (BinarySearchTree).

Forslag til signatur (C++):

```
string BinarySearchTree::findRoute(BinaryNode* root, int value)
```

en fuldstændig løsning a lá Weiss vil inkludere en *public* wrapper-metode med kun den eftersøgte værdi som parameter. Ovenstående metode vil så være *private*.



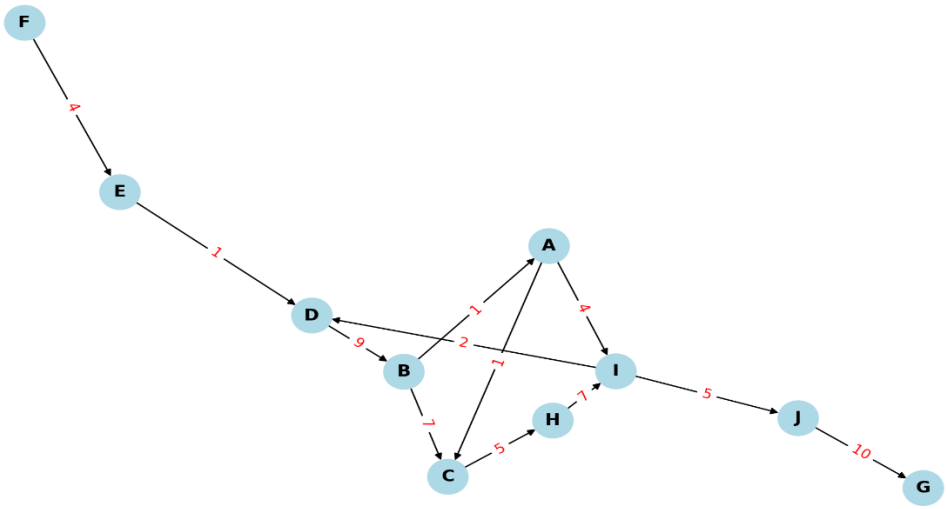
Kaldt med værdien 10 vil ovenstående træ returnere ”45 15 10” og kaldt med værdien 50 skal der returneres ”45 79 55 50”. Hvis den eftersøgte værdi ikke findes i træet, returneres en passende fejlmeddelelse.

Husk kun at aflevere de tilføjede metoder og *ikke* dit komplette træ.

Opgave 5 (12 %)

Med nedenstående graf skal du løse to opgaver:

1. Traverser grafen med Dijkstras algoritme under anvendelse af den angivne startkonfiguration.
2. Etabler et *minimum spanning tree* for den 'undirectede' version af grafen ved anvendelse af Kruskals algoritme. Besvarelsen skal bestå af en liste af edges i den rækkefølge, algoritmen vil tilføje dem. Angiv tillige træets samlede vægt.



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

Opgave 6 (13 %)

En hashtabel har plads til 13 elementer.

De første mange indsættelser hasher til indeks 3. Hvordan ser tabellen ud, når rehashing er uundgåelig under anvendelse af *quadratic probing*?

Brugte indeks kan markeres med Y1, Y2, Y3 etc.

0	
1	
2	
3	X
4	
5	
6	
7	
8	
9	
10	
11	
12	

Opgaven kan med fordel løses ved at skrive et lille program, som simulerer quadratic probing på en tabel med 13 elementer.

Opgave 7 (5 %)

Hvorfor kan nedenstående tabel *ikke* repræsentere en prioritetskø?

I øverste række er prioriteterne angivet; i nederste række står de tilsvarende tabelindeks.

	7	19	11	22	28	13	26	34	26	42	27	21	14	81	18	66	38	69	27
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	15	17	18	19