Øvelse 1

Skriv en algoritme, der tager en string som input og returnerer det oftest forekommende ord. Stringen er almindelig tekst (store og små bogstaver), og ord er adskilte af blanke, kommaer, punktummer eller kombinationer af disse (se eksempel nedenfor – du kan antage, at der altid er en blank imellem to ord, og at den sidste karakter altid er et punktum). Ord antages at være ens, selv om nogle forekomster begynder med lille bogstav og andre med stort bogstav.

Du kan antage som precondition for algoritmen, at input parameteren kun indeholder små bogstaverer [a-z] store bogstaver [A-Z], blanke, kommaer (,) og punktummer (.).

Eksempel:

{The cattle were running back and forth, but there was no wolf to be seen, heard, or smelled, so the shepherd decided to take a little nap in a bed of grass and early summer flowers. Soon he was awakened by a sound he had never heard before.}

Det korrekte svar I dette eksempel er ordet **a**, som forekommer tre gange.

Algoritmen skal være optimeret hvad angår køretid, og der skal angives et Store-O estimat for din løsning. Svaret skal begrundes.

Svar

Den store O estimat er $O(N^2)$, fordi der er en løkke, inde i en anden løkke. Det betyder den vokser med en kvadratisk vækst.

```
internal class Øvelse1
{
   private static char[] delimiterChars = {' ', ',', '.', ':', '\t'};

   private static string textToAnalyse = "The cattle were running back and forth, " +
                                         "but there was no wolf to be seen, heard, or smelled, so the shepherd
decided to take a " +
                                         "little nap in a bed of grass and early summer flowers. Soon he was
awakened by a " +
                                         "sound he had never heard before.";

   static void Main(string[] args)
   {
      Console.WriteLine("Most frequent word is: " + MostFrequentlyOccurringWord(textToAnalyse));
    }

   private static string MostFrequentlyOccurringWord(string phrase)
   {
      string[] words = phrase.Split(delimiterChars);
      List<Word> wordsObjects = new List<Word>();
```

```
        foreach (var word in words)
        {
            if(word == "") continue;
            Word currentWord = new Word
            {
                word = word,
                amount = 0
            };

            Console.WriteLine("Added new word: " + currentWord.word);
            foreach (var wordToCompare in words)
            {
                if (wordToCompare == currentWord.word)
                {
                    currentWord.amount++;
                }
            }
            wordsObjects.Add(currentWord);
        }

        string mostFrequentWord = "";
        int highestAmount = 0;
        foreach (var word in wordsObjects)
        {
            if (word.amount > highestAmount)
            {
                mostFrequentWord = word.word;
                highestAmount = word.amount;
            }
        }

        return mostFrequentWord;
    }

    [Serializable]
    public class Word
    {
        public string word;
        public int amount;
    }


}
```

```
"C:\Program Files\JetBrains\JetBrains Rider 2023.2.2\plugins\dpa\DotFiles\JetBrains.DPA.Runner.exe" --handle=1936 --backend-pid=36112 --etw-collect-flags=3 --detach-event-name=dpa.detach.1936 "C:/Users/olegs/Documents/SDU - Copy/SDU/Semester
Data struktur og Algoritmer/Portefølje Opgave 2 E2025/Porteføljeopgave 2 E2025/Porteføljeopgave 2 E2025/bin/Debug/Porteføljeopgave_2_E2025.exe"
Added new word: The
Added new word: cattle
Added new word: were
Added new word: running
Added new word: back
Added new word: and
Added new word: forth
Added new word: but
Added new word: there
Added new word: was
Added new word: no
Added new word: wolf
Added new word: to
Added new word: be
Added new word: seen
Added new word: heard
Added new word: or
Added new word: smelled
Added new word: so
Added new word: the
Added new word: shepherd
Added new word: decided
Added new word: to
Added new word: take
Added new word: a
Added new word: little
Added new word: nap
Added new word: in
Added new word: a
Added new word: bed
Added new word: of
Added new word: grass
Added new word: and
Added new word: early
Added new word: summer
Added new word: flowers
```

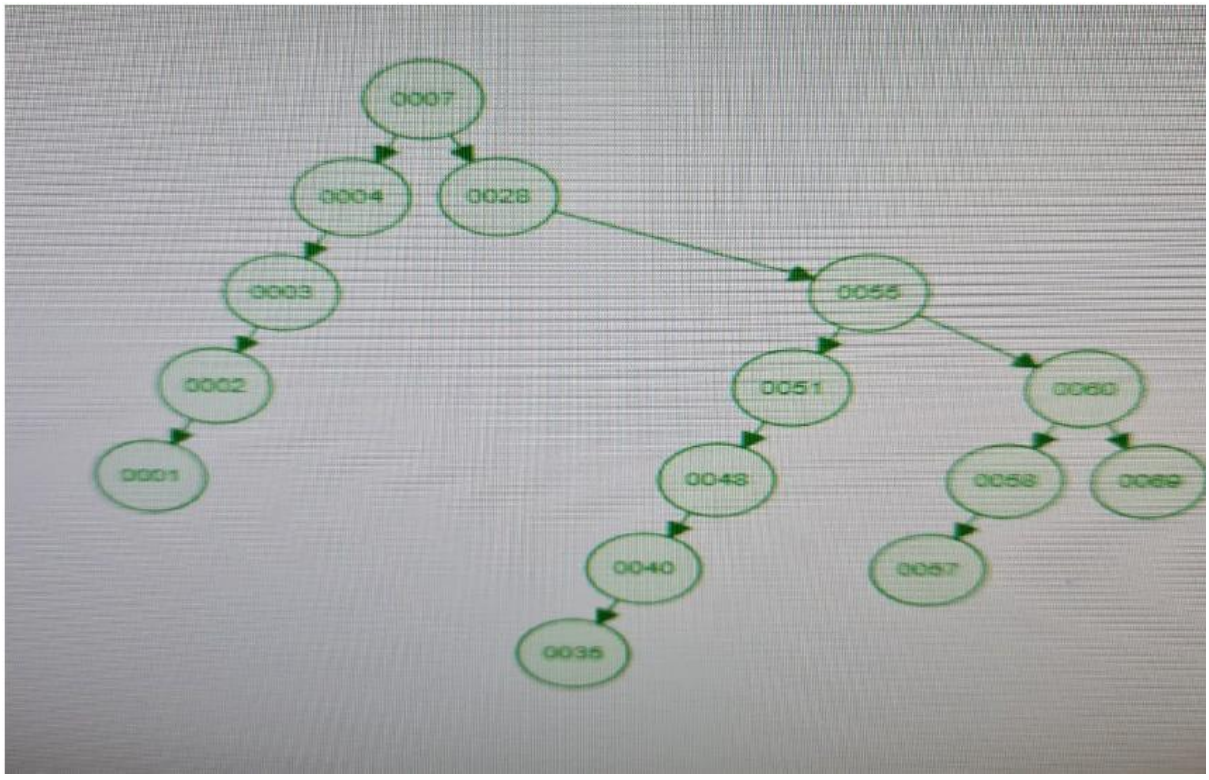


```
Added new word: Soon
Added new word: he
Added new word: was
Added new word: awakened
Added new word: by
Added new word: a
Added new word: sound
Added new word: he
Added new word: had
Added new word: never
Added new word: heard
Added new word: before
Most frequent word is: a

Process finished with exit code 0.
```


Øvelse 2

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 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.

Svar 1

```
public static List<Node> branch = new List<Node>();

static void Main(string[] args)
{
   Node root = new Node(7);
   Node rootLeft = new Node(4);
   Node rootRight = new Node(28);

   Node rootLeftOne = new Node(3);
   Node rootLeftTwo = new Node(2);
   Node rootLeftFour = new Node(1);

   Node rootRightOne = new Node(55);

   Node rightNodeLeft = new Node(51);
   Node rightNodeLeftOne = new Node(48);
   Node rightNodeLeftTwo = new Node(40);
   Node rightNodeLeftThree = new Node(35);

   Node rightNodeRight = new Node(60);
   Node rightNodeRightLeft = new Node(56);
   Node rightNodeRightLeftOne = new Node(57);

   Node rightNodeRightRight = new Node(69);
```

```
    root.left = rootLeft;
    root.right = rootRight;

    rootLeft.left = rootLeftOne;
    rootLeftOne.left = rootLeftTwo;
    rootLeftTwo.left = rootLeftFour;

    rootRight.left = rootRightOne;

    rootRightOne.left = rightNodeLeft;
    rightNodeLeft.left = rightNodeLeftOne;
    rightNodeLeftOne.left = rightNodeLeftTwo;
    rightNodeLeftTwo.left = rightNodeLeftThree;

    rootRightOne.right = rightNodeRight;

    rightNodeRight.left = rightNodeRightLeft;
    rightNodeRightLeft.left = rightNodeRightLeftOne;

    rightNodeRight.right = rightNodeRightRight;

    root.GetOnlyChild(root);

    Console.WriteLine(branch.Count);

}

public class Node {
    public int data;
    public Node left, right;

    public Node(int d) {
        this.data = d;
        left = null;
        right = null;
    }

    public void GetOnlyChild(Node node, Node parent = null)
    {
        if (node == null) return;

        bool oneChild = (node.left != null) ^ (node.right != null);

        bool noSiblings =
            parent == null ||
            (parent.left == node && parent.right == null) ||
            (parent.right == node && parent.left == null);
```

```
    if (oneChild && noSiblings)
    {
        Node child = node.left ?? node.right;
        bool childOneChild = (child.left != null) ^ (child.right != null);

        Node grandChild = child.left ?? child.right;
        bool grandChildIsLeaf =
            grandChild != null &&
            grandChild.left == null &&
            grandChild.right == null;

        if (childOneChild && grandChildIsLeaf)
            Øvelse2.branch.Add(node);
    }

    GetOnlyChild(node.left, node);
    GetOnlyChild(node.right, node);
}
```

```
5/Data struktur og Algoritmer/Portefølje Opgave 2 E2025/Porteføljeopgave 2 E2025/Porteføljeopgave 2 E2025/bin/Debug/Porteføljeopgave_2_E2025.exe"
2

Process finished with exit code 0.
```

Supplerende opgave 1: 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?

Svar 2

Træet er et binært søgetre, da alle venstre deltræer indeholder mindre noder end roden, og alle højre deltræer større.

Træet er ikke komplet, da nogle niveauer ikke er fuldt udfyldt, og det er hellere ikke perfekt, da bladene ikke alle ligger på samme dybde.

Træet er ikke balanceret, højresiden er betydeligt dybere end venstresiden

Træets optimale højde er $h = 007 \rightarrow 028 \rightarrow 055 \rightarrow 048 \rightarrow 040 \rightarrow 036 = 6$.

Supplerende opgave 2: 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?

Svar 3

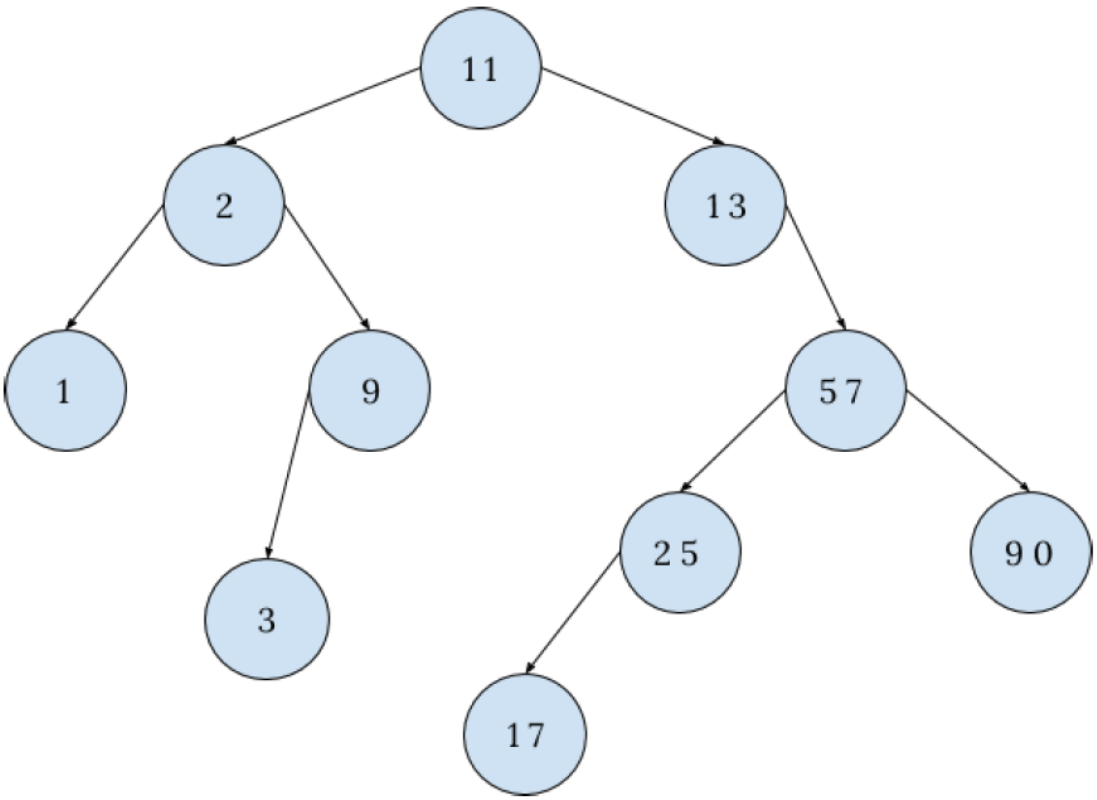
For at transformere binære søgetre til prioritets kø, gøres følgende:

- Gå igennem hele binære søgetre og indsamle alle værdier i et array
  - $O(n)$
- Byg en heap ud fra arrayet vha. Floyds buttom-up build heap algoritme
  - $O(n)$
- Brug arrayed som prioritetskø, hvor insert og deletMin kører $O(\log n)$

Samlet tidskompleksitet: $O(n)$

Øvelse 3

Nedenfor ses et binært søgetræ, som ikke er et AVL-træ:



List rækkefølgen I hvilken noderne vil blive besøgt I en ”in order” og i en ”level order” traversering.

Svar

In order:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 9 \rightarrow 11 \rightarrow 13 \rightarrow 17 \rightarrow 25 \rightarrow 57 \rightarrow 19$$

Level order:

$$11 \rightarrow 2 \rightarrow 13 \rightarrow 1 \rightarrow 9 \rightarrow 57 \rightarrow 3 \rightarrow 25 \rightarrow 90 \rightarrow 17$$

Hvad er træets internal path length?

$$n_{11} = 0$$

$$n_2 = 1$$

$$n_9 = 2$$

$$n_{13} = 1$$

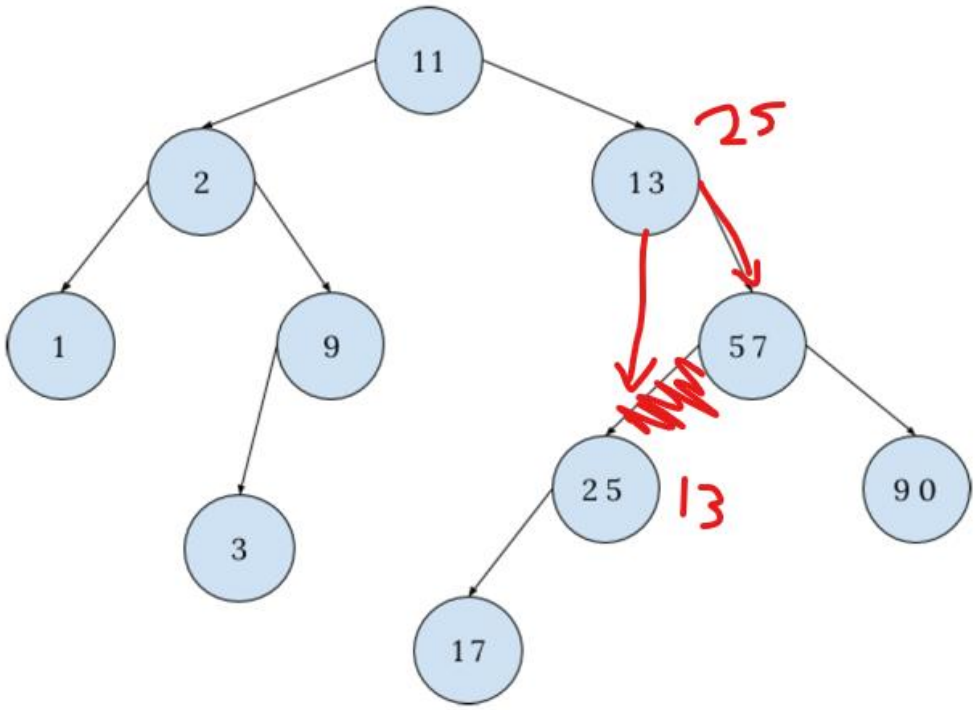
$$n_{57} = 2$$

$$n_{25} = 3$$

$$0 + 1 + 2 + 1 + 2 + 3 = 9$$

Ubalancen i træet er ved node 13, hvor højden af det højre subtræ er 3, og det venstre subtræ er 0. Hvordan kan man omarrangere noderne i det højre subtræ, så hele træet bliver et AVL-træ?

Svar



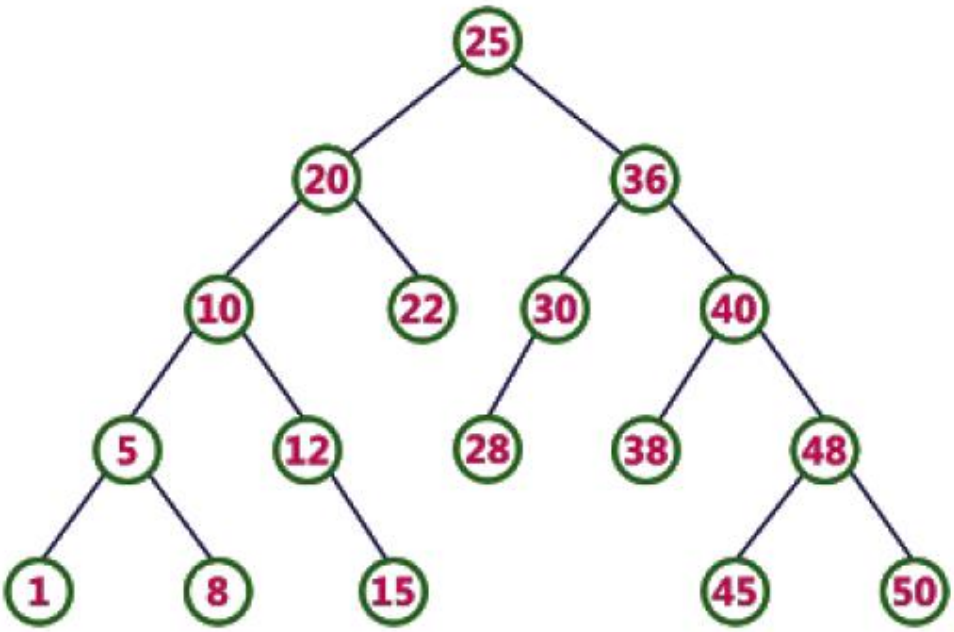
Kunne træet have været et AVL-træ før den seneste operation (insert eller delete, men ikke rotation)? Eksempler på seneste operation kunne være indsættelse af node 3 eller sletning af node 12 (venstre barn af node 13). Begrund dit svar.

Svar

Nej træet ville ikke kunne være et AVL-træ før de seneste operationer. Hvis node 3 blev fjernet nu, så vil der være endnu større ubalance mellem den venstre side og højre side. Hvis vi tilføjer 12, så betyder det at træet igen vil have større ubalance. Derfor er svaret Nej

Øvelse 4

Nedenfor ses et binært søgetræ:



List rækkefølgen i hvilken noderne besøges i en post order og i en pre order traversering.

Svar

Pre order:

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

Post order:

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

Hvad er træets internal path length?

Svar

$$0 + 1 + 1 + 2 + 2 + 2 + 3 + 3 + 3 = 17$$

Er det et AVL-træ? Hvorfor eller hvorfor ikke?

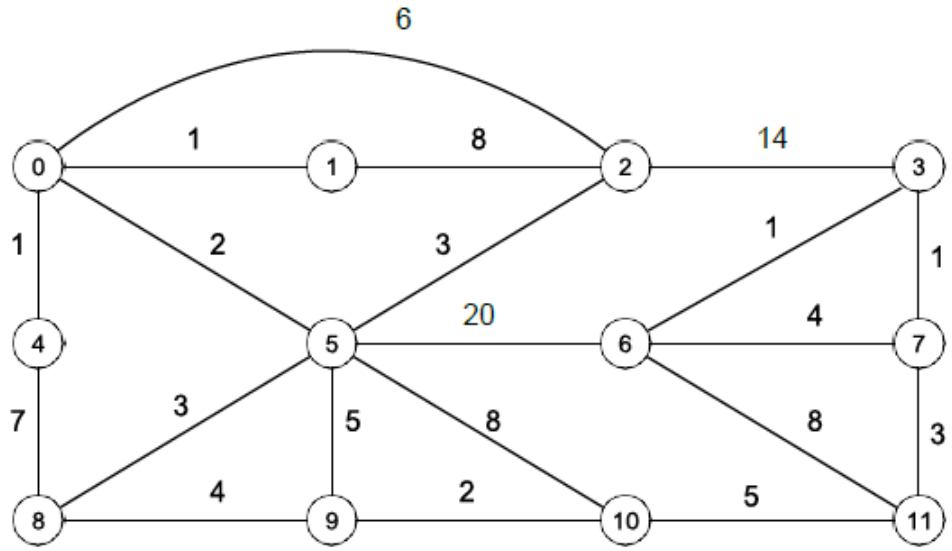
Svar

Fordi venstre deltræ højde er den samme som højre deltræ højde. $h_v - h_h = \leq 1$

$$8 - 8 = 0 \leq 1$$

Træet er altså balanceret

Øvelse 5



Find et *minimum spanning tree* for ovenstående graf. Svaret skal være en liste af edges/kanter, der viser i hvilken rækkefølge algoritmen vil etablere forbindelser mellem noderne. Angiv også træets totale vægt samt hvilken algoritme, du har brugt.

Svar

Adjacency Matrix

	0	1	2	3	4	5	6	7	8	9	10	11
0	∞	1	6	∞	1	2	∞	∞	∞	∞	∞	∞
1	1	∞	8	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	6	8	∞	14	∞	3	∞	∞	∞	∞	∞	∞
3	∞	∞	14	∞	∞	∞	1	1	∞	∞	∞	∞
4	1	∞	∞	∞	∞	∞	∞	∞	7	∞	∞	∞
5	2	∞	3	∞	∞	∞	20	∞	3	5	8	∞
6	∞	∞	∞	1	∞	20	∞	4	∞	∞	∞	8
7	∞	∞	∞	1	∞	∞	4	∞	∞	∞	∞	3
8	∞	∞	∞	∞	7	3	∞	∞	∞	4	∞	∞
9	∞	∞	∞	∞	∞	5	∞	∞	4	∞	2	∞
10	∞	∞	∞	∞	∞	8	∞	∞	∞	2	∞	5
11	∞	∞	∞	∞	∞	∞	8	3	∞	∞	5	∞

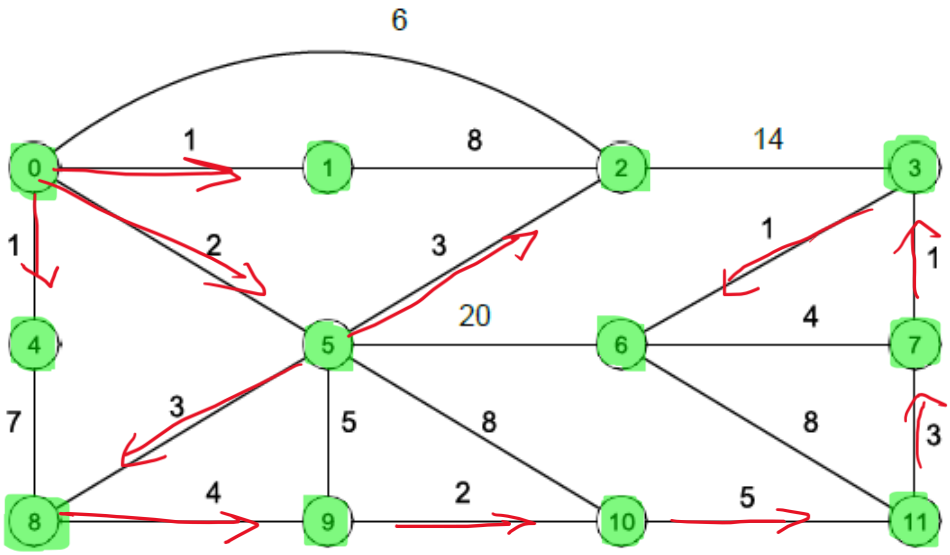
Jeg vil nu anvende c til at løse opgaven. Først vælger jeg startsnoden, i dette tilfælde $T = \{0\}$

Node 0, har fire kanter hvis man kigger på matrixen. Man skal vælge den kant som har den laveste vægt, og ikke indgår i den nuværende liste T.

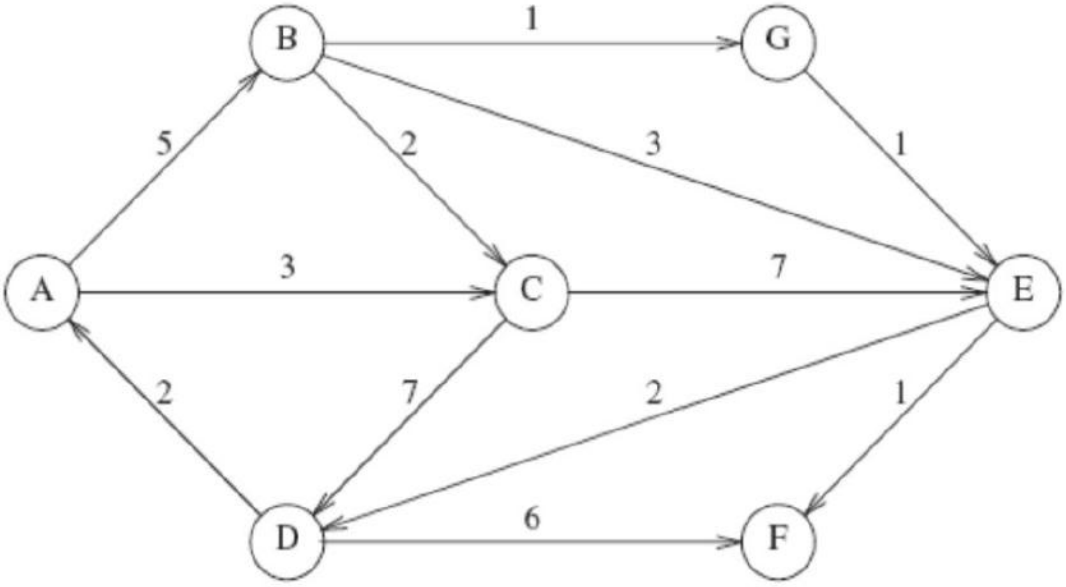
Den endelige liste vil se sådan ud, hvis vi fortsætter med algoritmen:

$$T = \{0,1,4,5,8,2,10,11,7,3,6\}$$

$$\{0,1;1\}\{0,4;1\}\{0,5;2\}\{5,8;3\}\{5,2;3\}\{8,9;4\}\{9,10;2\}\{10,11;5\}\{11,7;3\}\{7,3;1\}\{3,6;1\} = 26$$



Øvelse 6



Nedenstående tabel viser startkonfigurationen for en traversering af ovenstående graf med anvendelse af Dijkstras algoritme. Vis slutkonfigurationen for en traversering startende i node/vertex *A*. Du behøver ikke at vise mellemkonfigurationerne.

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

Svar

Trin 1:

v	known	d_v	p_v
A	F	0	-
B	F	∞	-
C	F	∞	-
D	F	∞	-
E	F	∞	-
F	F	∞	-
G	F	∞	-

Trin 2:

v	known	d_v	p_v
A	T	0	-
B	F	5	A
C	F	3	A
D	F	∞	-
E	F	∞	-
F	F	∞	-
G	F	∞	-

Trin 3:

v	known	d_v	p_v
A	T	0	-
B	F	5	A
C	T	3	A
D	F	10	C
E	F	10	C
F	F	∞	-
G	F	∞	-

Trin 4:

v	known	d_v	p_v
A	T	0	-
B	T	5	A
C	T	3	A
D	F	10	C
E	F	8	b
F	F	∞	-
G	F	6	B

Trin 5:

v	known	d_v	p_v
A	T	0	-
B	T	5	A
C	T	3	A
D	F	10	C
E	F	7	G
F	F	∞	-
G	T	6	B

Trin 6:

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

Færdig:

Trin 6:

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