Guldsmed Miniprojekt

Beskrivelse af problemet

Givet en kæde på en tilfældig længde, kan man opdele den i forskellige elementer som giver forskellige priser. Disse er 1 for 1, 13 elementer for 140 og 30 for 300. Når der skæres en kæde fra, så fjernes det element der skæres over også, og bliver solgt for 1. Så skal vi finde ud af, hvad den maksimale indtjening kan være, på den given længde.

Design

For at hjælpe guldsmeden med at finde den bedste pris, skal vi have en funktion som tager et tal ind, en længde og en liste af værdier af de forskellige ting som guldsmeden kan lave. Som output gives der et tal ud, den bedste pris. Det her problem kan løses både med en rekursiv eller dynamisk funktion.

Rekussiv

Først skal der være nogle slut scenarier. Her køres der igennem de forskellige måde hvor vi allerede ved hvad der oplagt er det bedste. Så det tjekkes til at starte med. Hvis kæden er mindre end 0, så sættes prisen til at være et minus tal, for at sikre at det altid bliver det dårligste. Derefter beregnes de bedste priser hvis der klippes på alle muligheder. Dette gøres ved at kalde funktionen igen, hvor man klipper af til en halskæde og til et armbånd, samt kigger på hvis man bare sælger alle som enkelte. Hvorefter der tjekkes hvilke af disse situationer der er bedst, og returnere de bedste.

Pseudokode

Input:
En længde givet i et heltal: length
En liste af værdier: Values
Output:
Et heltal der giver den bedst mulige pris
Funktion:

```
If length < 0
        return -2000
end

If length == 0
        return 0
end

If length == 1 then
        return Values[0]
end

If length == 13
        return Values[1]
end

If length == 30
        return Values[2]
end

arm = Funktion(length-14) + Values[1] + Values[0]
hals = Funktion(length-31) + Values[2] + Values[0]
en = 1*length

if hals >= arm and hals >= en
        return hals
else if arm >= en and arm >= hals
        return arm
else
        return en
end
```

Dynamisk

Først sikrer vi at inputtet er et tal over 0, da det ikke giver mening hvis lænden er minder i 0. Herefter laver vi en liste til at holde styr på de udregnet værdiger. Så køre vi et for loop hvor vi tjekker alle længder fra 1 til input. I for loopet sætter vi plads af den længde vi tjekker i listen til pladsen før pule en, derefter tjekker vi om den længde vi tjekker er længer lig med 13, lig med 30, længer end 13 og længer end 30.
Hvis længden er lig med 13 sætter vi plads af den længde vi tjekker i listen lig med 140.
Hvis længden er lig med 30 sætter vi plads af den længde vi tjekker i listen lig med 300.
Hvis længden er længer end 13, så tjekker vi om det der står på pladsen, er mindre end det der står på pladsen puls det der står på pladsen 14 før puls 1, hvis det er større sætter vi plads af den længde vi tjekker i listen lig med der står på pladsen puls det der står på pladsen 14 før puls 1.
Hvis længden er længer end 30, så tjekker vi om det der står på pladsen, er mindre end det der står på pladsen puls det der står på pladsen 31 før puls 1, hvis det er større sætter vi

plads af den længde vi tjekker i listen lig med der står på pladsen puls det der står på pladsen 31 før puls 1.

Pseudokode

Input:
En længde givet i et heltal: length
En liste af værdier: Values
Output:
Et heltal der giver den bedst mulige pris
Funktion:

```
If length < 0 then
        return 0
end

 memory = list[length +1]

for i = 1, i <= lenght, i +1
        memory[i] = memory[i - 1] +1

        if i == 13
                memory[i] = memory[i - 13] + Values[1]
        end

        if i == 30
                memory[i] = memory[i - 30] + Values[2]
        end

        if i > 13
                if memory[i] < memory[i - 14] +Values[1] +Values[0]
                        memory[i] = memory[i - 14] +Values[1] +Values[0]
                end
        end

        if i > 30
                if memory[i] < memory[i - 31] +Values[2] + Values[0]
                        memory[i] = memory[i - 31] +Values[2] + Values[0]
                end
        end
end
return memory[lenght-1]
```

Implementering

Funktionen har fået sin egen fil med en klasse, for at separere de forskellige løsninger, Dette gør at det er nemmer for gruppemedlemmerne at arbejde samtidige med kode.
Den regressive kode er implementeret i Rekrusiv.cs (se bilag 2) og den dynamiske kode er implementeret i Dynamisk.cs (Se bilag 3).

Bonus

Tidsmåling

For at måle tidsforskellen på de to funktioner, bruges der en funktion som tager en integer som input, hvor den så opretter et ur og starter det. Så stoppes dette ur når der er regnet den bedste pris ud. Dette bliver gjort for begge. Så derefter udskrives resultaterne i konsollen.

Bilag

Bilag 1: Program.cs

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6.
7. namespace GuldSmed
8. {
9.     class Program
10.      {
11.          static string PrittyFormatResult(int number, long ms)
12.          {
13.              return String.Format("Bedste Pris Dynamisk:
   {0}\nUdregnet på {1} Ms\n", number, ms);
14.          }
15.
16.          static void TimeAndRunGetPriceRight(int input)
17.          {
18.              var watchDynamisk =
   System.Diagnostics.Stopwatch.StartNew();
19.              int dynamiskInt = Dynamisk.GetPriceRight(input,
   values);
20.              watchDynamisk.Stop();
21.              Console.WriteLine(PrittyFormatResult(dynamiskInt,
   watchDynamisk.ElapsedMilliseconds));
22.              var watchRekusiv =
   System.Diagnostics.Stopwatch.StartNew();
23.              int rekusivInt = Rekrusiv.GetPriceRight(input, values);
24.              watchRekusiv.Stop();
```

```
25.              Console.WriteLine(PrittyFormatResult(rekusivInt,
   watchRekusiv.ElapsedMilliseconds));
26.          }
27.
28.          static Dictionary<int, int> values = new Dictionary<int,
   int>
29.          {
30.              {1,  1},
31.              {13, 140},
32.              {30, 300}
33.          };
34.
35.          static void Main(string[] args)
36.          {
37.              string input = Console.ReadLine();
38.              while (input != "end")
39.              {
40.                  if (int.TryParse(input, out int i))
41.                  {
42.                      TimeAndRunGetPriceRight(i);
43.                  }
44.                  input = Console.ReadLine();
45.              }
46.              Console.ReadKey();
47.          }
48.      }
49.  }
```

Bilag 2: Rekrusiv.cs

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6.
7. namespace GuldSmed
8. {
9.     class Rekrusiv
10.      {
11.          public static int GetPriceRight(int length, Dictionary<int,
   int> vals)
12.          {
13.              if (length < 0)
14.              {
15.                  return int.MinValue;
16.              }
17.
18.              if (length == 0)
19.              {
20.                  return 0;
21.              }
22.
23.              foreach (int l in vals.Keys)
24.              {
25.                  if (length == l)
26.                  {
27.                      return vals[l];
28.                  }
```

```
29.             }
30.
31.             int arm = GetPriceRight(length -
   14,vals)+vals[13]+vals[1];
32.             int hals = GetPriceRight(length - 31, vals) + vals[30]
   + vals[1];
33.             int one = vals[1] * length;
34.
35.             if (hals >= arm && hals >= one)
36.             {
37.                 return hals;
38.             }
39.             else if (arm >= one && arm >= hals)
40.             {
41.                 return arm;
42.             }
43.             else
44.             {
45.                 return one;
46.             }
47.         }
48.     }
49. }
```

Bilag 3: Dynamisk.cs

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6.
7. namespace GuldSmed
8. {
9.     class Dynamisk
10.     {
11.         public static int GetPriceRight(int length, Dictionary<int,
   int> vals)
12.         {
13.             if (length <= 0)
14.             {
15.                 return 0;
16.             }
17.
18.             int[] memory = new int[length + 1];
19.
20.             for (int i = 1; i <= length; i++)
21.             {
22.                 memory[i] = memory[i-1] + vals[1];
23.
24.                 if (i == 13)
25.                 {
26.                     memory[i] = memory[i - 13] + vals[13];
27.                 }
28.
29.
30.                 if (i > 13)
31.                 {
```

```
32.                         if (memory[i] < memory[i - 14] + vals[13] +
   vals[1])
33.                         {
34.                             memory[i] = memory[i - 14] + vals[13] +
   vals[1];
35.                         }
36.                     }
37.
38.                     if (i == 30)
39.                     {
40.                         memory[i] = memory[i - 30] + vals[30];
41.                     }
42.
43.
44.                     if (i > 30)
45.                     {
46.                         if (memory[i] < memory[i - 31] + vals[30] +
   vals[1])
47.                         {
48.                             memory[i] = memory[i - 31] + vals[30] +
   vals[1];
49.                         }
50.
51.                     }
52.
53.                 }
54.
55.                 return memory[length];
56.
57.             }
58.
59.         }
60.     }
```