A-LABB
DD1320 Tillämpad datalogi
A1 Paper

Marwin Haddad

marwinh@kth.se
19980520-1051

December 2021

Uppgiftsbeskrivning

Syftet med uppgiften är att skriva ett program som löser problemet **A1 Paper** och analysera programmet med avseende på datastrukturer, algoritm och tidskomplexiteten. Problemet är formulerat enligt följande:

Att tejpa ihop två A2-ark ger ett A1, två A3-ark ett A2 och så vidare. Går det, givet antalet pappersark i olika storlekar i A-serien, bestämma längden tejp som krävs för att skapa ett A1-ark. Antag att längden tejp som behövs till att sätta ihop två ark är lika med deras långsida. Ett A2-papper har långsidan $2^{-3/4}$m och kortsidan $2^{-5/4}$m och varje papperstorlek i följd har samma form men hälften av arean av den föregående[1].

Länk till kattisinlämning `https://kth.kattis.com/submissions/8132392`

Datastrukturer

Den datastruktur som används i koden är en *array*. En array består av en samling element av datatyper vilka kan identifieras givet deras position, eller *index*, i array-datastrukturen. I den här uppgiften används en en-dimensionell array, med andra ord en *list* i python.

Array-strukteren i koden är listan *nlist* innehållande antalet papper per pappersstorlek större än pappersstorleken An till A2 givet $2 \leq n \leq 30$. Papperstorlekarna i listan är sorterade efter största storlek där antalet ark av storlek A2 ligger först och antalet ark av storlek An sist.

För att skapa listan nlist används något som kallas *list comprehension* vilket är ett smidigt sätt att skapa en lista ifall det som ska ligga i listan redan ligger i en annan. I detta fall fås inmatningen redan sorterad och i listform men där elementen är av *string*-datatypen. För att konvertera elementen till *integer*-datatypen konverteras elementen i nämnd list comprehension utan att behöva använda *append*-metoden tillhörande list-strukturen.

Algoritm

Indatan till algoritmen är den minsta tillgängliga pappersstorleken n och en sträng med heltal separerade med mellanslag där varje heltal representerar antal pappersark per pappersstorlek från storleken A2 till och med An. Algoritmen ger utdatan den minsta längden tejp som behövs till att tejpa ihop pappersark till ett ark av storleken A1. Algoritmen lyder enligt följande:

1. Sätt nuvarande lång- och kortsida till lång- och kortsidan av ett A2-papper.
2. Sätt antalet papper av nästa storlek som **behövs** till 2. I början behövs 2st A2 ark för att få ett A1.
3. Sätt begynnelselängden av den tejp som behövs till längden av ett A2-ark.
4. För varje antal ark per storlek:
   (a) Subtrahera antalet ark av **nuvarande** storlek från antalet ark som **behövs**.
   (b) Om antalet ark som **behövs** är noll eller mindre
      i. avsluta algoritmen och returnera längden tejp.
   (c) Definiera om nuvarande lång- och kortsida efter nästa storleks dimensioner.
   (d) Multiplicera nuvarande långsidan med antalet ark som **behövs** och addera produkten till tejplängden.
   (e) Dubblera antalet ark som **behövs**.
5. Upprepa från 4. tills att vi itererat genom alla antal ark av olika storlekar.
6. Återstår det ark som **behövs** efter att ha itererat över alla pappersark drar vi slutsatsen att det är omöjligt att bilda ett A1-papper givet tillgängliga papper.

I följande två exempel används ovan algoritm för att beräkna sammanlagda längden tejp för de två testfallen givna i kattis[1]. Vi ansätter minsta tillgängliga pappersstorlek till n och sekvensen av antal papper per storlek till $\mathbf{S}$. Det sökta arket är av storlek $A1$.

Exempel 1: $n = 4$, $\mathbf{S} = \{1, 0, 5\}$

1. $A2$ har långsidan $2^{-3/4}$m och kortsidan $2^{-5/4}$m.
2. Vi behöver två ark av storleken $A2$ för att få ett $A1$.
3. Nuvarande längden tejp är $2^{-3/4}$m.
4. För varje antal ark i $\mathbf{S}$:
   (a) $2 - 1 = 1$; vi behöver ett till $A2$.
   (b) $1 > 0$; vi fortsätter till (c).
   (c) $A3$ har långsidan $2^{-5/4}$m och kortsidan $\frac{2^{-3/4}}{2}$m.
   (d) $2^{-3/4} + 1 \cdot 2^{-5/4}$
   (e) Antalet $A3$ vi behöver är $2 \cdot 1 = 2$
5. Eftersom det återstår papper i $\mathbf{S}$ upprepar vi steg 4.
   (a) $2 - 0 = 2$; vi behöver två till $A3$.
   (b) $2 > 0$; vi fortsätter till (c).
   (c) $A4$ har långsidan $\frac{2^{-3/4}}{2}$m och kortsidan $\frac{2^{-5/4}}{2}$m.
   (d) $2^{-3/4} + 1 \cdot 2^{-5/4} + 2 \cdot \frac{2^{-3/4}}{2}$
   (e) Antalet $A4$ som behövs är $2 \cdot 2 = 4$.

   Eftersom det återstår papper i $\mathbf{S}$ upprepar vi steg 4.
   (a) $4 - 5 = -1$
   (b) Antalet ark som behövs är färre än noll
      i. vi avslutar algoritmen och returnerar tejplängden $2^{-3/4} + 1 \cdot 2^{-5/4} + 2 \cdot \frac{2^{-3/4}}{2} \approx 1.609$m.

Exempel 2: $n = 3$, $\mathbf{S} = \{0, 3\}$

1. Stegen 1 - 4 är desamma för detta exempel.
4. För varje antal ark i $\mathbf{S}$:
   (a) $2 - 0 = 2$; vi behöver två till $A2$.
   (b) $2 > 0$; vi fortsätter till (c).
   (c) $A3$ har långsidan $2^{-5/4}$m och kortsidan $\frac{2^{-3/4}}{2}$m.
   (d) $2^{-3/4} + 2 \cdot 2^{-5/4}$
   (e) Antalet $A3$ vi behöver är $2 \cdot 2 = 4$
5. Eftersom det återstår papper i $\mathbf{S}$ upprepar vi steg 4.
   (a) $4 - 3 = 1 \ldots$
   $\vdots$
6. Eftersom vi itererat genom hela $\mathbf{S}$ och fortfarande behöver papper av nästa storlek drar vi slutsatsen att det är omöjligt att bilda ett ark av storleken A1 givet $\mathbf{S}$.

Tidskomplexitet

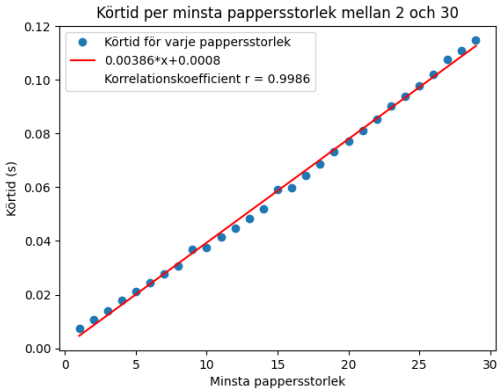
I analysen av algoritmens tidskomplexitet kommer de mest tänkbara tidskrävande utfallen att analyseras.

Genom en överblick av koden går det att komma fram till antagandet att algoritmen har linjär tidskomplexitet. Detta innebär att algoritmens tidfunktion beror linjärt på antalet termer, det vill säga att en förändring av antalet termer algoritmen tar alltid leder till en proportionell ändring i algoritmens körtid. Eftersom varje for-slinga i sig i värsta fall itererar över varenda element i listan nlist beror algoritmens körtid av antalet termer[2].

För att verifiera antagandet modifierades huvudprogrammet med koden i **Listing 2** i **Appendix 1**. Filen *two_to_thirty* är en textfil där varenda rad utgör en minsta papperstorlek $2 \leq n \leq 30$ och där alla termer i vardera rad utgör antalet papper per papperstorlek större än n. För att simulera den mest tidskrävande körtiden undersöks de fall där vi endast har 2^{n-1} stycken av den minsta storleken. Detta tydliggörs av **Listing 3** i **Appendix 1**.

n	2	5	10	20	30
tid(s)	0.0075515	0.0177652	0.036822	0.0730783	0.114705

Figur 1:



Figur 2:

Resultatet av undersökningen ges av ovan figurer. I **Figur 1** visas körtiden för de mest tidskrävande utfallen för papperstorlekarna A_2, A_5, A_{10}, A_{20} och A_{30}. **Figur 2** är ett diagram där punkterna utgör körtiden för $2 \leq n \leq 30$. Utefter hur punkterna är fördelade ansattes en linjär regressionslinje med hjälp av minstakvadratmetoden. I och med erhållen korrelationskoefficient fås determinationskoefficienten, det vill säga korrelationskoefficienten i kvadrat, till $r^2 = 0.9972 \approx 99.7\%$. Korrelationskoefficienten ger ett mått på hur nära de observerade mätvärdena ligger den ansatta regressionslinjen och determinationskoefficienten anger hur stor andel av variationen i körtiden som ges av variation i antalet element, under förutsättningen att sambandet mellan körtiden och antalet termer i inmatningen faktiskt är linjärt[3]. Givet ovan Korrelations- och determinationskoefficient dras slutsatsen att antagandet om att algoritmen är $\mathcal{O}(n)$ stämmer.

Appendix 1: Programkod

```
def calc_tape(nlist):
    # definiera dimensioner av A2 ark
    long_side = 2**(-3/4)
    short_side = 2**(-5/4)

    # summa av alla tejplängder, vi börjar med längden av en A2 pga det är den
    # minsta tejplängden man kan behöva
    sum_tape = long_side

    # eftersom varje storlek är dubbelt så stor som nästa så börjar vi med att
    # anta att vi behöver 2st A2
    sheets_needed = 2

    # current_size motsvarar antal papper av varje storlek i listan nlist
    for current_size in nlist:

        # skillnaden mellan antalet papper vi behöver och antalet vi har ger oss
        # antalet vi behöver av nästa storlek
        sheets_needed -= current_size

        # om vi inte behöver några ark av nästa storlek vet vi att vi är klara
        if sheets_needed <= 0:
            return sum_tape

        # nästa långsida är nuvarande kortsida // nästa kortsida är hälften av
        # nuvarande långsida
        long_side, short_side = short_side, long_side/2

        # adderar till tejplängden antalet ark vi behöver av nuvarande längd multiplicerat
        # med  nästa storleks längd
        sum_tape += sheets_needed * long_side

        # antalet ark vi behöver av nästa storlek är dubbelt så många
        sheets_needed *= 2
    # returnerar 'impossible' ifall "sheets_needed" aldrig blir noll
    return 'impossible'

def main():
    n = int(input())
    nlist = [int(x) for x in input().split()][:n-1]
    # funktion som beräknar och returnerar längden av tejpen
    print(calc_tape(nlist))

if __name__ == '__main__':
    main()
```

Listing 1: Hela koden

```
def main(nrow):
    nlist = [int(n) for n in nrow.split()]
    return calc_tape(nlist)


if __name__ == '__main__':
    tidlista = []
    with open('two_to_thirty') as f:
        for i, row in enumerate(f.readlines()):
            tidlista.append(timeit.timeit(stmt=lambda: main(row), number=10000))

    x = np.array(range(1, len(tidlista)+1))
    k, m = np.polyfit(x, tidlista, 1)[0], np.polyfit(x, tidlista, 1)[1]
    print()
    print(tabulate([['n', '2', '5', '10', '20', '30'],
                    ['tid(s)', tidlista[0], tidlista[3], tidlista[8], tidlista[18], tidlista[28]]],
                   headers='firstrow'))
    r = np.corrcoef(x, tidlista)
    plt.plot(x, tidlista, 'o', label='Körtid för varje pappersstorlek')
    plt.plot(x, k*x+m, 'r', label=f'{np.round(k, 5)}*x+{np.round(m, 5)}')
    plt.plot([], [], ' ', label=f'Korrelationskoefficient r = {round(r[0][1], 4)}')
    plt.legend()
    plt.title('Körtid per minsta pappersstorlek mellan 2 och 30')
    plt.xlabel('Minsta pappersstorlek')
    plt.ylabel('Körtid (s)')
    plt.show()
```

Listing 2: Tillagd kod som verifierar tidskomplexitet - resterande kod är densamma.

```
2
0 4
0 0 8
0 0 0 16
0 0 0 0 32
0 0 0 0 0 64
0 0 0 0 0 0 128
0 0 0 0 0 0 0 256
0 0 0 0 0 0 0 0 512
0 0 0 0 0 0 0 0 0 1024
          :
          :
          :
          :
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 67108864
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 134217728
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 268435456
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 536870912
```

Listing 3: Textfilen *two_to_thirty*.

Appendix 2: Testdata

```
indata:
    Ange minsta storlek: 2

    Ange antal per storlek från A2 till A2:
    2

utdata:
    0.5946035575013605
```

Listing 4: Test där vi endast har två stycken $A2$. Ger kortast möjliga tejplängd i meter.

```
indata:
    Ange minsta storlek: 30

    Ange antal per storlek från A2 till A30:
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 536870912

utdata:
    33259.78073974868
```

Listing 5: test där vi har 2^{29} stycken $A30$. Ger längsta möjliga tejplängd i meter.

```
indata:
    Ange minsta storlek: 30

    Ange antal per storlek från A2 till A30:
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 536870911

utdata:
    impossible
```

Listing 6: Test där vi har $2^{29} - 1$ stycken $A30$. Undersöker programmets känslighet.

Referenser

[1] Ulf Lundström, KTH Challenge 2015, hämtad 07-12-21, https://kth.kattis.com/problems/a1paper

[2] Linda Kann, *Föreläsning 3: Komplexitetsanalys, sökning, rekursion* i DD3120 Tillämpad datalogi, hämtad 07-12-21, https://canvas.kth.se/courses/26987/pages/forelasning-3-komplexitetsanalys-sokning-rekursion

[3] Gunnarson R. Korrelation och regression [Dept of Prim Health Care Göteborg University - Research methodology web site]. Uppdaterad 05-01-02. Hämtad 08-12-21. https://infovoice.se/fou/bok/statmet/10000053.shtml