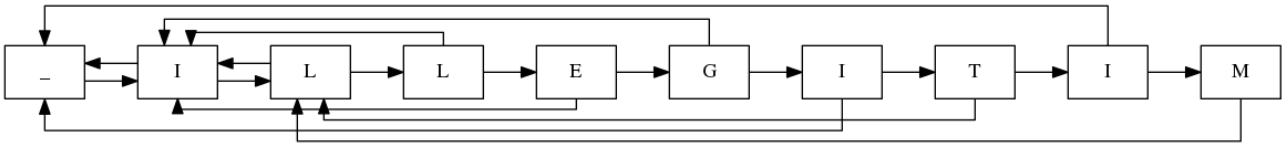
Tildatenta 2019-10-21 lösningsförslag

Uppgift 1 KMP



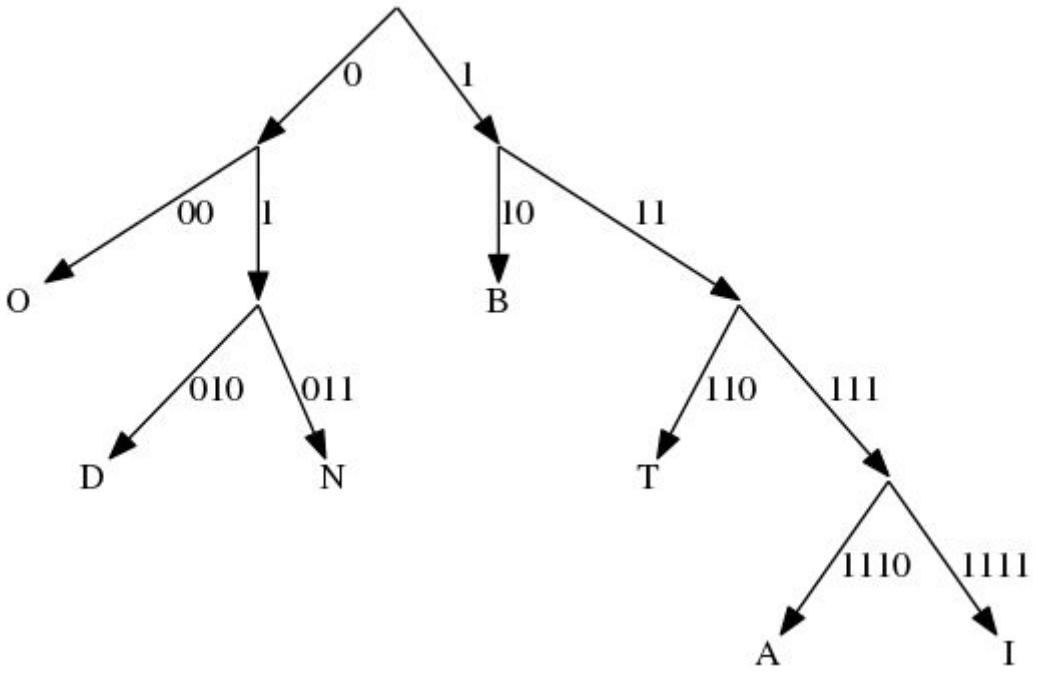
next-vektor:
`[0, 1, 1, 1, 1, 0, 2, 0, 2]`

Uppgift 2 Tidskomplexitet

	Algoritm	**Tidskomplexitet**	**Värsta fallet**
1	b, Linjärsökning bland n element	O(n)	O(n)
2	d, Sökning i hashtabell med n element	O(1)	O(n)
3	c, Binärsökning med n element	O(log n)	O(n)
4	a, Urvalssortering av n element	O(n²)	O(n²)

Uppgift 3 Komprimering

Bobo har rätt.



Uppgift 4 Kryptering

a.) One-time pad. Caesarchiffer är alltför lätt att knäcka - det är bara att prova igenom alla skiften (byta A mot B etc, byta A mot C etc). One-time pad använder slumpmässiga bitar - mycket säkrare.

b.) Alice måste först se till att hon och Bob har kopior av samma uppsättning slumpmässiga binära tal = pad1. Hon kodar sitt meddelande binärt och använder bitvis XOR med det slumpade binära talet för att kryptera. Bob kan dekryptera meddelandet genom att göra XOR med samma pad1.

c.) Så här går det till:

Alice krypterar

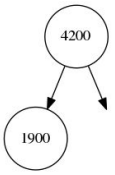
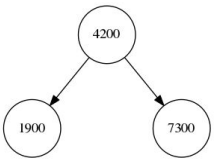
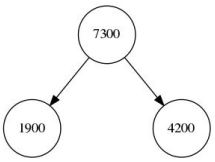
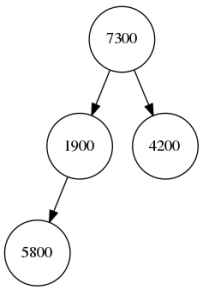
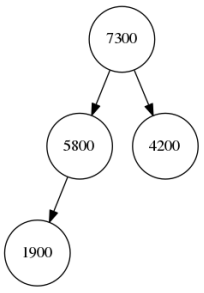
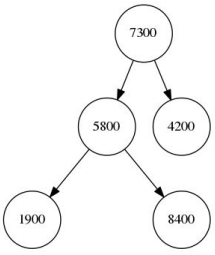
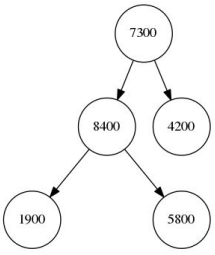
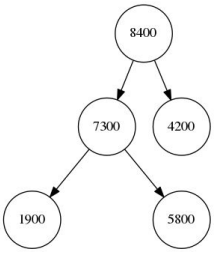
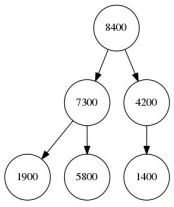
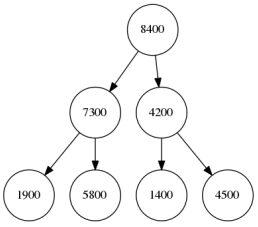
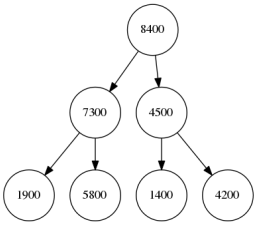
meddelande	100000110010111010100100000110001001001011000111
pad	**001001111111010101000111111101001011000011000011**
krypterat	101001001101101111100011111011000010001000000100

Bob dekrypterar:

krypterat	101001001101101111100011111011000010001000000100
pad	**001001111111010101000111111101001011000011000011**
meddelande	100000110010111010100100000110001001001011000111

Uppgift 5 Heap

Svar e) [8400, 7300, 4500, 1900, 5800, 1400, 4200]

4200			
1900			
7300			
5800			
8400			
1400			
4500			

Uppgift 6 Reguljärt uttryck

```
[a-zA-Z0-9.]+@[a-zA-Z]+\.dnt
```

Uppgift C7

Datastruktur	**a) Hur?**	**b) Lämplig?**
binärt sökträd	Varje illegalt primtal sorteras in i binära sökträdet, mindre tal till vänster och större till höger. Operation för sökning finns.	Snabb sökning O(logn) Primtalen lagras i klartext, därför olämplig.
bloomfilter	Varje illegalt primtal hashas in (i en boolesk hahstabell) med flera hashfunktioner. Vid kontroll: Om alla hashfunktioner ger platser där det står True i tabellen är det ett illegalt primtal. OBS! Vi kan råka få True även för primtal som inte är illegala, men det är ingen katastrof (vore värre om det var tvärtom).	Snabb sökning O(1) Primtalen lagras inte alls. Lämplig.
hashtabell	Varje illegalt primtal hashas in i en hashtabell. I noderna lagras själva primtalet (behövs eftersom det kan bli krockar).	Snabb sökning O(1) Primtalen lagras i klartext, därför olämplig. Det går förstås att lagra primtalen saltade och hashade (samma idé som för lösenord) viket ger hemlig lagring.
prioritetskö/heap	Varje illegalt primtal läggs in med själva primtalet som prioritet. Barnen till en nod är alltid större (alternativt mindre) än föräldern, så om vi plockar ut talen blir dom sorterade. Sökning finns inte med bland prioritetsköns operationer.	Måste plocka ut talen ur *prioritetskön* för att kontrollera om primtalet finns med, O(nlogn). Alternativt linjärsöka i *heapvektorn* O(n) (abstraktionsbrott). Primtalen lagras i klartext. Olämplig.

Uppgift A8

a) Algoritm:

indata: x (mutbudget)

n mutkolvar (en vektor med de n tjänstemännen som ska mutas)

utdata: Ja om det gick att muta alla n med totalt x kronor, Nej annars

1. Upprepa punkt 2 - 4 nedan för varje person p i listan **mutkolvar**:
2. Hitta ett intervall [**minmuta**, **maxmuta**] mellan vilka mutgränsen ligger.
   a) Börja med **muta** = **x**.
   b) Om muta accepteras (dvs **p.acceptera(muta)**):
      Halvera **muta** tills den inte längre accepteras
   c) Intervallet blir då mellan **minmuta = muta** och **maxmuta = muta*2**
3. Nu gör vi binärsökning mellan minmuta och maxmuta för att hitta den exakta mutgränsen.
   a) Sätt **muta = mittpunkten** i intervallet
   b) Om muta accepteras:
      sätt **maxmuta = muta**
     annars:
      sätt **minmuta = muta + 1**
   c) Är intervallets längd är större än 1? Upprepa a) och b)
   d) ...annars vet vi att maxmuta är p:s mutgräns
4. Minska budgeterade värdet **x = x - mutgräns**
   Om x < 0:
     så har vi överskridit budget, svaret är **Nej**.
   Annars
     fortsätt från punkt 1 med nästa person
5. Har vi kommit igenom alla personer så är svaret **Ja**.

b) Demonstration:

Budget x = 100

Mutgränserna kommer visa sig vara [75, 20] (men det vet vi inte från början).

För person 1:

Hitta mutgränsen

100 accepteras, men inte 50

=> Mutgränsen ligger mellan 50 och 100

Binärsökning maxmuta = 75

Binärsökning minmuta = 63

Binärsökning minmuta = 70

Binärsökning minmuta = 73

Binärsökning minmuta = 75

Mutar person 1 med 75 kronor.

För person 2:

Hitta mutgränsen

100, 50, 25 accepteras, men inte 12

=> Mutgränsen ligger mellan 12 och 25

Binärsökning minmuta = 19

Binärsökning maxmuta = 21

Binärsökning maxmuta = 20

Binärsökning minmuta = 20

Mutar person 2 med 20 kronor.

Pengarna räckte, 5 kronor kvar. Svaret är JA

c) Tidskomplexitet:

Binärsökning bland x tal => log(x)
Totalt n varv => en faktor n
Svar: O(n*log(x))

Ingående variabler: n (antal tjänstemän), x(mutornas storlek)