Kortfattad lösningsskiss tentamen 160322

1

```
K A L L E K A L L A S K A R L
0 1 1 1 1 0 1 1 1 5 1 0 1 3 1
```

Bilden med pilar är inte med i denna lösningsskiss

2

b) är rätt svar

3)

Kön före vändning: Kajsa <- Kicki <- Pippi <- Titti

Kön efter vändning: Titti <- Pippi <- Kicki <- Kajsa

Algoritmen är att plocka ut alla element ur kön och stoppa dem i stacken. Därefter poppa alla element i stacken **och lägga tillbaka dem** i kön.

Kön
Kajsa <- Kicki <- Pippi <- Titti

Stack (tom)

Kön
Kicki <- Pippi <- Titti

Stack

Kajsa

Kön

Pippi <- Titti

Stack

Kicki
Kajsa

….

Kön (tom)

Stack

Titti
Pippi
Kicki
Kajsa

Kön

Titti

Stack
Pippi
Kicki
Kajsa

Kön

Titti <- Pippi

Stack
Kicki
Kajsa

…

4

Det finns flera godkända svar.

F 11
P 10
I 001
U 010
’ ’ 011
A 0000
R 0001

Det finns flera lösningar beroende hur man vänder på trädet, en alternativ lösning i trädform

```
[IARPU F:100] - [U F:55] - [F:30]
                         \ [U :25] - [ :15]
                                   \ [U:10]
              \ [IARP:45] - [P:25]
                          \ [IAR:20] - [AR:10] - [R:5]
                                                \ [A:5]
                                     \ [I:10]
```

5

a) Ja, det är O(N) I båda fallen

b) Ja, i en sorterad vektor kan man använda binärsökning som är O(N). Det går att implementera mer tidskrävande sökningar t.ex. O(N^3) men det gör man i allmänhet inte.

c) Ja, sökning i en sorterad vektor och i sett balanserat binärt sökträd är båda O(N)

d) Ja, efterföljande element måste flyttas in.

Det är godkänt med tre av fyra rätt på den här uppgiften.

6

a) Hammingavståndet är 3

b) De tolkas som F T F F T T F T (F – False, T – True)

Det är godkänt med ett slarvfel på uppgiften.

7)

	1)	2)	3)	4)	5)
a)	G	U	G	U	G
b)	G	G	U	G	U
c)	G	G	G	U	U

Det finns flera omdömen man kan göra om syntaxerna. Ingen av syntaxerna är jättebra. Rekursionen ligger fel, de borde ligga kring namn om man ser på exemplen. Sista syntaxen tillåter namn som ”Pigg Knatte Joakim von”. I en av syntaxerna är rekursionen hårdkodad.

8)

Vid linjär probning sparas både key och value efter varandra. Om nyckel man söker efter inte finns på hashindex så söker man linjärt efterföljande element tills man träffar på en tom plats. Då antas elementet inte finnas i hashvektorn.

a) När man tar bort ett element skapas ett hål. Sökning kommer att stanna här och direkt efterföljande element inte genomsökas (de försvinner)

b) Man måste lägga dit ett element som markerar att det varit något här

c) Det är samma problem

d) Krocklistor har inte samma problem. Borttagning är enkelt att göra. Det blir inget hål i listan

9)

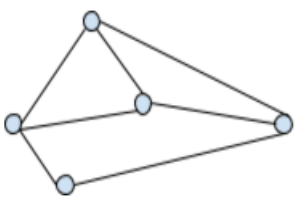
Det finns flera sätt att läsa in data på. Man behöver en datastruktur för grafen och man behöver veta hur många attraktioner det finns. En datastruktur som fungerar är t.ex. en dictionary med attraktionerna som nycklar och en lista med andra attraktioner som value.

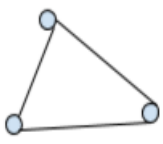
Datastrukturen ska beskrivas med exempeldata.

Ingång	Hoppet	Åksjukan	Godishjulet
Hoppet	Ingång	Åksjukan	Mardrömstunneln
Åksjukan	Ingång	Hoppet	Mardrömstunneln
Mardrömstunneln	Hoppet	Åksjukan	Godishjulet
Godishjulet	Ingång	Mardrömstunneln	
Musfällan	Bergbanan	Vattenfallet	
Bergbanan	Musfällan	Vattenfallet	
Vattenfallet	Musfällan	Bergbanan	

Algoritmen blir en fullständig grafgenomgång antingen med bredden först eller djupet först. Förutom de vanliga stegen där redan-tidigare-besökta noder beaktas så måste man räkna eller markera de attraktioner man behandlat. Efter grafgenomgången jämför man med förväntat antal attraktioner för att avgöra om man når alla.

Uppgiften ger höjd för att läsa in alla attraktioner i en grafstruktur. Men, man måste på något sätt kunna jämföra med förväntat antal attraktioner. Attraktionerna kan ligga i två disjunkta grafer varav bara den ena gås igenom av algoritmen.





10

```
def insert(i, v):
    if i < len(v):
        x = Nod(v[i])
        x.left  = insert(2*i, v)
        x.right = insert(2*i + 1, v)
        return x
    else:
        return None
```

Avbrottsvillkoret är när vi försöker indexera utanför vektorn.

Vid ett givet ett index i vektorn.

- Skapa en nod med elementet på detta index.
- Anropa rekursivt med index 2*i och skapa ditt vänsterbarn
- Sätt ditt högerbarn till det rekursiva anropet med index 2*i+1
- Returnera noden