KTH-CSC, Alexander Baltatzis

DD1320/1321, Tilda, lösningsskiss 18 mars 2014

1. *Var är bussen*

E Boyer-Moore kollar för vart tredje tecken i texten om det ingår i ”178” och hamnar så småningom på första 7. Tecknet närmast efter är dock ingen 8, så man hoppar vidare till 1. Sen stämmer 7 och 8 så 178 har hittats.

2. *Resa snabbt med Metron*

E Först skapas directory med stationens namn som nyckel och en lista på grannstationernas namn som värde. Man går till exempel igenom varje linje, skapar vid behov ny directorypost och fyller på grannlistan.

Sedan skapas ett problemträd bredden-först. Noderna innehåller stationsnamn och faderspekare. Startstationen är stamfar och läggs i en kö. Upprepade gånger tas en station ur kön och dess barn (directoryvärdet) läggs in sist i kön med faderspekare. När destinationen dyker upp följer man faderspekarna och får då vägen med så få byten som möjligt.

3. *Tidsordning i binärt sökträd*

E Algoritmen heter *inorder* och anropas `inorder(root)`

```
def inorder(p):
    if p==None: return
    inorder(p.left)
    print(p.value,end=" ")
    inorder(p.right)
```

Tiderna antas vara insorterade i tidsordning, inte i bokstavsordning, alltså så att 6:15 kommer före 13:20.

4. *Skriv ut hållplatstabell med binärt sökträd*

C Med en global variabel `hglobal=` kan *inorder* klara uppgiften.

```
def inorder(p):
    global hglobal
    if p==None: return
    inorder(p.left)
    h,m=p.value.split(":")
    if h==hglobal: print(m,end=" ")
    else:
        print(h,"   ",m,end=" ")
        hglobal=h
    inorder(p.right)
```

Den som lider av globalallergi kan i stället ge `hh` som inparameter till inorder och låta inorder returnera det nya värdet på `hh`. Anropet blir `inorder(root,hh)` och koden så här.

```
def inorder(p,hh):
    if p==None: return
    hh=inorder(p.left,hh)
    h,m=p.value.split(":")
    if h==hh: print(m,end=" ")
    else: print(h,"   ",m,end=" ")
    hh=inorder(p.right,h)
    return hh
```

5. *Nästa avgångstid i binärt sökträd*

A En sökning efter tiden `t` kommer antingen att hitta tiden i en nod eller att hamna i ett löv som har lite fel tid. Om `p.value` $\geq$ `t` har vi hittat nästa avgång och returnerar `p.value`. Annars är vi i ett löv och vill returnera den närmast större tiden. Den måste vi ha passerat tidigare i sökningen, den senaste gången vi gick åt vänster. Därför måste vi spara tiden varje gång vi går åt vänster, så här.

```
# next departure at time t
p=root
while p!=None:
    time=p.value
    if t<time:
        p=p.left
        lefttime=time
    elif t>time: p=p.right
    else: break
if time>=t: print(time)
else: print(lefttime)
```

6. *Stoppa in i heap*

E Alternativ b) är rätt.

7. *Heap eller inte heap?*

E Alternativ d) är rätt.

8. *Resonera kring syntax*

C Testfall som är komplicerade men följer syntaxen och testfall som är enkla men inte följer syntaxen. Andra bra testfall är tomma meddelanden och långlånga meddelanden eftersom sådana ofta skapas av misstag.

9. *Bygg Huffmanträd*

E Huffmanträdet ger koderna

```
A 0
N 10
S 11
```

10. *Avkoda Huffmanträd*

A Problem finns när man ska välja om ett delträd ska sättas till vänster eller till höger. Med följande regel blir trädet unikt bestämt.

När två delträd ska hopfogas jämförs bokstäverna som ingår i träden och det träd sätts till vänster som har den tidigaste bokstaven i alfabetet.

I exemplet sätts först N-trädet till vänster om S-trädet och sedan A-trädet till vänster om NS-trädet. Meddelandet kan sedan avkodas till ANANAS.

11. *Datastruktur för resdata*

C SL-kortets ID hashas till ett index i en lagom stor hashtabell. Om olika kort hashas till samma index finns krocklista av poster. Varje post innehåller ID och en pekare till en lista med tid+plats.

12. *Sortera undanflykter*

C Eftersom listan är ordnad så när som på ett par poster i slutet ska bubbelsorteringen ske från slutet. Då blir komplexiteten högst $2n$ och i genomsnitt bara n.