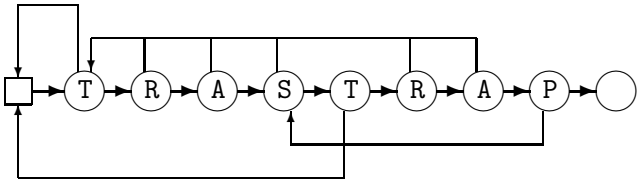
KTH, NADA, Linda Kann

2D1320, Tilda, lösningsskiss 19 oktober 2004

1. *Fågelläte*

(4p)



i	next[i]
1	0
2	1
3	1
4	1
5	0
6	1
7	1
8	4

(2p) Automaten går igenom följande tillstånd för inmatningen TANTRASTRAPATS:
1 2 1 0 1 0 1 2 3 4 5 6 7 8

2. *Fågeltanke*

(5p) Om trädet endast består av roten och ett löv blir resultatet 0 eftersom vi inte har något villkor som definierar ett löv. Korrekt tanke skulle vara:

Rekursion: Antal löv är 1 om roten är ett löv, annars antal löv i vänster delträd + antal löv i höger delträd.
Basfall: Antal löv i ett tomt träd är 0
(Man kan också lägga till lövvillkoret i basfallet.)

3. *Fågelväg*

(10p) Bästaförstsökning: Läs in all info om startplatser och landningsplatser samt etappavstånd från filen och lagra i en datastruktur som är sökbar på startplatser, t ex ett sökträd. Objekten i problemträdet har följande data: landningsplats, tillryggalagt avstånd och pappapekare. Gå igenom problemträdet på följande sätt: starta med stamfar (avstånd noll och pappapekare null) i Arktis och skapa söner genom att gå igenom alla tänkbara landningsplatser och plussa på avståndet. Sönerna läggs i en min-prioritetskö med avståndet som nyckel. När alla söner till stamfadern skapats plockar man nästa ur prioritetskön och skapar söner till den. Om Antarktis pluppar ut ur prioritetskön har vi fått bästa lösningen och kan skriva ut färdvägen.

För att slippa dumsöner kan man använda en hashvektor med landningsplatsen som nyckel, där man håller reda på vilket det hittills kortaste avståndet till varje plats är. Klasser för sökträd, prioritetskö och hashtabell behövs. Metoder är bland annat `readFile()`, `makeSons()` och `writeChain()` som anropas nar man hittat till Antarktis.

4. *Fågelsökning*

(4p) Sökmetod A, som tar lika lång tid oberoende av antal uppslagsord är hashning, som är O(1). Sökmetod B tar dubbelt så lång tid när antalet uppslagsord ökar kvadratiskt ($100^2 = 10000$) och är alltså O(logn), vilket innebär binärsökning (i sorterad array eller binärt sökträd).

5. *Fågelperspektiv*

(2p) **a.** Ja, ett reguljärt uttryck kan användas för att ange de villkor som gäller för söksträngen, t ex börjar med p eller P osv.

(2p) **b.** En del av testningen är att köra programmet med många uppsättningar av korrekta indata, men även körning med felaktiga indata, hög belastning, återstart efter krasch mm ingår.

(2p) **c.** Nej, paritetsbitar lägger till redundans (istället för att ta bort som vid komprimering), för att man ska ha möjlighet att upptäcka om fel har uppstått vid dataöverföring.

(2p) **d.** Om man vill sortera många små arrayer var för sig är insättningssortering snabbast i praktiken. (Men om man sedan vill samsortera till en stor array är merge-delen av mersort bäst.)

(2p) **e.** Om man skriver ut trädet i inorder så skrivs först vänster delträd, sen roten och sist höger delträd. Om vänster delträd är tomt så kommer roten ut först.

6. *Fågelkomprimering*

(4p) Bokstäverna får följande koder:
K:11
U:10
E:011
I:010
L:001
C:000
Meddelandet är KUCKELIKU

(4p) **7.** *Fågelskri*

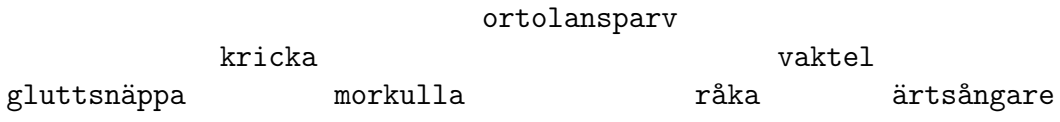
<skri> ::= <start> <mitt> <slut>
<start> ::= HY | HY <start>
<mitt> ::= HIH
<slut> ::= ε | Y | Y <slut>

(3p)

Gör en metod för varje vänsterled ovan. Tjuvtitta på nästa tecken för att avgöra vilken metod som ska anropas. I start-metoden får man läsa in H och kika på nästa tecken för att avgöra om man ska anropa start rekursivt eller anropa mitt-metoden.

8. *Fågelträd*

(2p) Ett balanserat träd:



(2p) Ett balanserat binärträd med tusen fåglar skulle få cirka 10 nivåer ($2^{10} = 1024$).