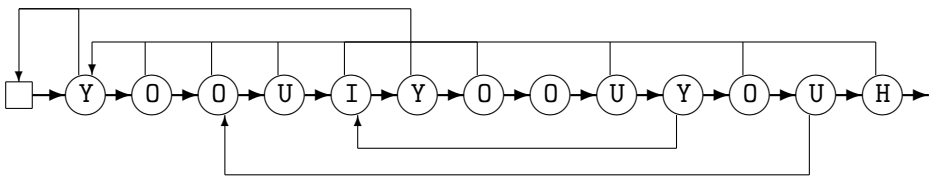
KTH-CSC, Alexander Baltatzis

DD1320, Tilda, lösningsskiss 11 mars 2013

1. *Ultimat automat*

(8p)



i	next[i]
1	0
2	1
3	1
4	1
5	1
6	0
7	1
8	1
9	1
10	5
11	1
12	3
13	1

Det är lite lurigt att den ena av de två mer "avancerade" bakåtpilarna är en begynnelsebokstav

2. *Ukrainsk paranoia*

(10p) Sverige skapar krypteringsnycklar och publicerar sin publika nyckel. Ukraina skapar signeringsnycklar och publicerar sin publika nyckel. Meddelandet krypteras först med Sveriges publika nyckel. Sedan krypteras/signeras meddelandet med Ukrainas privata nyckel för att autentisera att det är just Ukraina som skickat det. Sverige dekrypterar med Ukrainas publika nyckel och sedan med Sveriges privata nyckel. På detta sätt garanteras att meddelandet kom från rätt avsändare (autentisering) och att ingen kunnat tjuvläsa (kryptering) det.

Med "utan att gå in för mycket på matematiska detaljer" menas att man inte behöver skriva om detaljerna i RSA. Det räcker att skriva att den matematiska funktionen är svår att reversera.

3. *Utlandets prioritering*

(5p) **a)**

```
[8,30,33,49,37,57,36,82,91,50]
[30,37,33,49,50,57,36,82,91]
[33,37,36,49,50,57,91,82]
[36,37,57,49,50,82,91]
[37,49,57,91,50,82]
[49,50,57,91,82]
```

(5p) **b)**

```
[32,40,41,75,103]
[32,40,41,75,103,44]
[12,40,32,75,103,44,41]
[12,18,32,40,103,44,41,75]
[12,18,32,40,103,44,41,75,59]
[12,18,32,40,49,44,41,75,59,103]
```

4. *Vann rätt låt?*

(8p) a) Alla kodexemplen fungerar. De tre sista räknar dock fel om inte alla rader är lika långa och kraschar om första raden är tom. `foo4` kraschar dessutom om det är färre än 10 rader. `foo1` kan räkna jättefel om det smugit sig in någon annan siffra i matrisen.

(4p) c) Alla kodexemplen kollar i värsta fallet alla $m \cdot n$ matriselement, där m är antal rader och n antal kolumner. Därmed är beräkningskomplexiteten $\mathcal{O}(m \cdot n)$ för alla algoritmerna. Jämförelser som görs för varje rad ger ett extra $\mathcal{O}(m)$ som inte höjer detta. Det är viktigt att definiera vad man avser med N och M. Svar som $\mathcal{O}(N)$

(6p) kan vara rätt om man definierar N som antal element i matrisen.

b) Det finns flera omständigheter man kan ta hänsyn till t.ex. om indatat består nästan enbart av antingen nollor eller ettor. Om indatat är jämnt fördelat. Om addition är dyrare eller billigare än jämförelse på den platform man kör. Slutligen finns omständigheten att man inte vet hur indata ser ut och utifrån den synvinkeln är det bara dumt med den här typen av optimeringsförsök.

`foo1` är snabbast om andelen ettor är liten eftersom det då blir få additioner och den inte gör någon jämförelse. Om jämförelse `if x==0` är snabbare än addition (`ones += x`) så är `foo2` snabbast när andelen ettor är medelstor. `foo3` är snabbast om andelen ettor är stor, eftersom den bryter när raden får för många nollor. `foo4` kan bli snabbast om andelen ettor är stor i början av matrisen, eftersom den är ett mellanting mellan andra och tredje algoritmen.

(8p) **5.** *Utred bloomfilter*

"Utred huruvida ..." betyder att man ska `utreda om` man kan använda bloomfilter för att jämföra två matriser. Svaret på det är att det är en dålig idé. Det är bättre att jämföra två givna matriser elementvis.

Bloomfilter som abstrakt datatyp har i princip två funktioner `put(word)` och `exists(word)` och det finns inget direkt gränssnitt för att jämföra två olika matriser. Att indata (matrisen) består av nollor och ettor ska inte blandas ihop med bloomfiltrets interna data.

Ett bloomfilter som skapats med hjälp av en matris kan användas för att kolla om andra matriser överensstämmer med originalet. Ett praktiskt fall skulle kunna vara att matriselementen är pixlar och att man vill veta om bilden finns i ett stort bildarkiv. Bloomfiltret funkar ju så att man beräknar hashvärden enligt en uppsättning formler (summan av diagonalelementen skulle kunna vara en sådan formel) och kollar att filtret har ettor för dessa index. Bildarkivsexemplet att jämföra en matris mot flera tidigare matriser med Bloom-filter är en bra idé.

Om man tar matriselementen åtta och åtta, rad för rad, kan man se matrisen som en följd av ASCII-tecken, alltså en text, och bloomfilter används ju på detta sätt av i stavningsprogram. Så visst kan man använda bloomfilter för matriser.

Man skulle kunna svara att det går att använda bloom-filter genom att anropa 14 hashfunktioner element för element men observera att detta är minst 14 gånger sämre än elementvis jämförelse. Detta bör nämnas i den utredning som efterfrågades.

6. *Uppfinn sorteringsalgoritm*

(14p) En modifierad räknesortering är den bästa metoden. Summera raden och sätt in ettor på så många platser i början, resten nollor. Komplexiteten blir $\mathcal{O}(m \cdot n)$ men antalet jämförelser blir noll.

Quicksortpartitionering (damernaförstsortering) är näst bäst, där vänstra pekaren får stanna på nolla, högra stannar på etta och platsbyte sker. Samma komplexitet men antalet jämförelser blir nu en miljon.

7. *Utmärkta algoritmer och datastrukturer*

(20p) Här finns många tänkbara algoritmer. Den bästa är att leta reda på första radens sista etta med binär sökning. Det kräver 10 jämförelser. Sök nedåt i kolumnen till nästa etta, sedan linjär (eller binärsökning) åt höger till sista ettan i den raden, nedåt till nästa etta, åt höger osv. Total krävs 999 jämförelser neråt och högst 999 (troligare 500) åt höger. För att minnas vilka maxraderna är kan man lägga radnumren i en lista som skrivs över varje gång ett nytt max hittas.. Komplexiteten är $\mathcal{O}(m + n)$ och i exemplet behövs 14 jämförelser. Glöm inte definiera vad m och n står för.

Indata är matrisen. Utdata är en tupel av ett tal och en lista, maxantalet ettor och maxradernas nummer.

Det finns flera andra lösningar, man kan göra samma sak från andra hållet och bearbeta nollor. Man kan lägga antal ja-röster och radnummer i en prioritetskö prioriterat på antal ja-röster och plocka ut de bästa raderna efter man gått igenom matrisen och innan man returnerar.

8. *Uppdrag: skriv syntax*

För att få full poäng måste man ha med några exempel på godkända och icke-godkända tal. Hädanefter ska jag komma ihåg att skriva det uttryckligen på tentan, troligtvis kommer jag be om intressanta exempel vilket ofta är de vid randen. Nedan är en variant
(4p) på korrekt lösning, notera att rekursiviteten är i andra ledet.

```
<bintal> ::= <bit> | 1 <bitar>
<bitar>  ::= <bit> | <bit> <bitar>
<bit  >  ::= 0 | 1

godkända tal 0, 1, 100
icke godkända -1, 2, 01
```

Hantera inledande nollor är inte jätteviktigt.

(8p) Syntax för binära tal delbara med åtta (vilket är tal som slutar med tre nollor):

```
<bin8tal> ::= 0 | 1000 | 1 <bitar> 000
<bitar>  ::= <bit> | <bit> <bitar>
<bit  >  ::= 0 | 1

godkända tal 0, 1000, 11000, 100000
icke godkända 10001, 2, 100
```