KTH, NADA, Henrik Eriksson

2D1320, Tilda, Tentamenslösning 17 april 1999

1. *DNA–automat*

I det här fallet är Boyer-Moores metod värdelös. Alla bokstäver i alfabetet ACGT ingår ju i sökordet.

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

2. *Binärträdsintervall*

```
PROCEDURE Intervall(firstword,lastword:TEXT;p:Pek)=
VAR startsbefore,endsafter: BOOLEAN;
BEGIN
  IF p=NIL THEN RETURN END;
  startsbefore:=(Text.Compare(firstword,p.word)=-1);
  endsafter:=(Text.Compare(p.word,lastword)=-1);
  IF startsbefore THEN Intervall(firstword,lastword,p.left) END;
  IF startsbefore AND endsafter THEN IO.Put(p.word&"\n") END;
  IF endsafter THEN Intervall(firstword,lastword,p.right) END;
END Intervall;
```

3. *Genmanipulation*

Man bör tolka formuleringen så att det gällde att skriva ut bokstäverna i den transponerade ordningen. Med två köobjekt gör man så här (om vänstraste bokstaven ligger först i kön):

```
FOR i:=1 TO a DO IO.Put(queue1.Get()) END;
FOR i:=1 TO b DO queue2.Put(queue1.Get()) END;
FOR i:=1 TO c DO IO.Put(queue1.Get()) END;
FOR i:=1 TO b DO IO.Put(queue2.Get()) END;
FOR i:=1 TO d DO IO.Put(queue1.Get()) END;
```

Med två stackobjekt gör man så här (om vänstraste bokstaven ligger överst):

```
FOR i:=1 TO a DO IO.Put(stack1.Pop()) END;
FOR i:=1 TO b DO stack2.Push(stack1.Pop()) END;
FOR i:=1 TO c DO IO.Put(stack1.Pop()) END;
FOR i:=1 TO b DO stack1.Push(stack2.Pop()) END;
FOR i:=1 TO b+d DO IO.Put(stack1.Pop()) END;
```

4. *DNA-hashning*

Det tar för lång tid att räkna ut ett hashvärde som beror på varenda bokstav. Ta till exempel var hundrade bokstav och använd i en slinga formeln `h:=4*h+tkn` där `tkn` räknas som 0,1,2,3 för A,C,G,T. Index blir `h MOD size`, där lämpligen `size=150001`.

5. *Kortaste transpositionsvägen*

Låt musens sifferföljd bli stamfar och skapa hela trädet breddenförst med hjälp av en kö. I kön läggs poster med sifferföljd och faderspekare, och söner skapas genom att man gör alla transpositioner av fadern. För att undvika dumsöner, dvs dubbletter, har man exempelvis ett binärträd för använda sifferföljder. När följden 1234... dyker upp har man kortaste transpositionsvägen mellan mus och människa.

6. *Gensyntax*

```
<gene>   ::= <start> <triples> <stop>
<triples>::= <triple> | <triple> <triples>
<triple> ::= <letter> <letter> <letter>
<letter> ::= A | C | G | T
<start>  ::= CTT
<stop>   ::= TAG | GAG
```

7. *Gensortering*

a) En bra quicksort har komplexitet $\mathcal{O}(N \log N)$ men här är vektorn nästan sorterad från början, och då är det lätt hänt att komplexiteten blir $\mathcal{O}(N^2)$.

b) Insättning av M nya gener i N gamla har tydligen komplexitet $\mathcal{O}(MN)$.

c) Sortera dom nya generna med quicksort, samsortera sedan (merge). Komplexiteten blir då $\mathcal{O}(M \log M) + \mathcal{O}(N)$.

8. *Abstrakta gener*

En abstrakt `Gene.T` kan vara en objekttyp med följande gränssnitt.

```
INTERFACE Gene
TYPE T<:AbstractGene;
     AbstractGene = OBJECT
     METHODS
       getLength():INTEGER;
       setLetter(i:INTEGER;tkn:CHAR);
       getLetter(i:INTEGER):CHAR;
       getText():TEXT;
     END;
PROCEDURE New(text:TEXT):T; (* Skapar en gen från en text *)
END Gene.
```

Abstrakta gener är bättre eftersom man kan ändra representation utan att ändra programmen som använder abstrakta gener.