**Datastrukturer & Algoritmer, Datalogi C**
Henning Christiansen 18/11-2003

**Opgaver til stoffet om hash-tabeller, som gennemgået ved forelæsningen 18/11-2003**

Opgave 2 er en afleveringsopgave, afleveringsfrist 5/12, men som vi forventes at begynde på ved øvelserne.

**Opgave 1**
Bogens opgave 20.5 a, b og 20.6 side 710.

**Opgave 2**
Skitser en algoritme til brug i en metode for sletning i en hashtabel, som benytter »linear probing«. Idéen er at man rent faktisk skal sætte den relevante celle i tabellen »null«, så cellen bliver frigivet. (Altså ikke noget som ligner vor lærebogs »snyde-sletning« vha. en boolean som angiver om et objekt i tabellen anset som slettet).

**Opgave 3 (Afleveringsopgave)**

I får leveret en fil med Shakespeares tragedie Hamlet i en rå tekstform, og den skal I bruge i forbindelse med en opgave om hashtabeller. Teksten indeholder godt 4500 forskellige ord optalt når man ikke skelner mellem store-små bogstaver og betragter ethver ikke-bogstav som skilletegn mellem ord.

Det sidste ikke er helt korrekt, da f.eks. ordet »usurp'ts« fejlagtigt læses som to ord »usurp« og »ts«, men vi ser bort fra den slags detaljer her i opgaven.

På kursets hjemmeside finder du to filer »hamlet« med hele teksten og »shorthamlet« som blot indeholder nogle få ord; den sidste er god til indledende tests.

For at forenkle indlæsningen kan du benytte en klasse EasyFile, som er tilgængelig på kursets hjemmeside, og som vi har benyttet til en tidligere opgave (se nærmere beskrivelse som del af opgaveformuleringen 19. september 2003). Brugen af den er også illustreret tilstrækkeligt nedenfor.

**Spørgsmål 2.1**

Du skal skrive et program som benytter hashing til at registrere, hvilke ord, som forekommer i Shakespeares tragedie Hamlet, jvf. principperne ovenfor. Du må ikke bruge nogen prædefineret klasse til hashing, det er meningen du skal skrive det hele selv.

Der skal bruges det princip til organisering af tabellen som kaldes »linear probing«, og den anbefales på det kraftigste, at man dropper en hver idé om en generisk version af hashing, holder sig til en fast tabelstørrelse og undlader re-hashing og ikke tager det så tungt, hvis tabellen løber fuld. Til at repræsentere ord benyttes tekststrenge (String-

objekter) og som udgangspunkt for en hashfunktion benyttes den til strenge hørende hashCode()-metode (og husk at tage modulo tabelstørrelsen). Benyt to arrays, ét som er den egentlige hashtabel (med referencer til strenge) som vi kalder keyTable, og et andet som indholder tællere for antal gange en streng er mødt; det kalder vi countTable. Så hvis f.eks. ordet »denmark« hasher til værdien 4866 (når der et taget modulo tabelstørrelse!) og er registreret 29 gange, så vil keyTable[4866] holde en reference til strengen »denmark« og countTable[4866] tallet 29.

Du kan tage udgangspunkt i denne skitse af et program; de mange konstanter er til brug i det følgende spørgsmål:

```
public  class HashTest {

  public static int primeNumber0 = 13;
  public static int primeNumber1 = 509;
  public static int primeNumber2 = 1021;
  public static int primeNumber3 = 2039;
  public static int primeNumber4 = 4093;
  public static int primeNumber5 = 8191;
  public static int primeNumber6 = 16381;
  public static int primeNumber7 = 32783;
  public static int primeNumber8 = 65537;


  public static int nonPrimeNumber0 = 16;
  public static int nonPrimeNumber1 = 512;
  public static int nonPrimeNumber2 = 1024;
  public static int nonPrimeNumber3 = 2048;
  public static int nonPrimeNumber4 = 4096;
  public static int nonPrimeNumber5 = 8192;
  public static int nonPrimeNumber6 = 16384;
  public static int nonPrimeNumber7 = 32786;
  public static int nonPrimeNumber8 = 65536;

  public static class HashTable {
     private String [] keyTable;
     private int [] countTable;
     private int size;


     public HashTable(int size)
     {....       }

     public void count(String x)
     { // Benyt hashing med linear probing til at finde rette index i
       // keyTable og countTable; tæl keyTable[index] op;}
```

```
    public void printTable()
       { .....}

  }

public static void main(String [] args) {

 HashTable ht = new HashTable(primeNumber0);

 EasyFile F = new EasyFile("shorthamlet");
 F.openEasyFileForRead();
 while( ! F.endOfEasyFile())
    {String s = (F.readStringEasy()).toLowerCase();
     ht.count(s);};
 ht.printTable();  }
}
```

Beskriv din løsningsmetode, aftest din løsning og vedlæg testudskrifter (genereret med en passende illustrativ printTable-metode).

Vink: Den enkleste løsning synes at være at benytte hashing-princippet direkte i metoden count i stedet for at skrive specialiserede metoder til at finde en given streng o.lign. Det kan oplyses at din lærers testløsning fylder 90 linjer incl. alt!

**Spørgsmål 2.2**

Nu skal vi undersøge hvor godt hashtabellen fungerer med forskellige tabelstørrelser, og her skal du køre med den store fil »hamlet«. Da vi ved, at der er godt 4500 forskelige ord, er det altså konstanterne primeNumber4–8 (primtal) og NonPrimeNumber4–8 (2-talspotenser) som er interessante at afprøve.

Som mål for hvor godt tabellen fungerer definerer vi for et givet ord X placeret i tabellen, dets *displacement* til at være afstanden mellem hvor hashfunktionen rammer, og hvor X faktisk ligger i tabellen. Hvis f.eks. et ord ligger der hvor hashFunktionen rammer, vil dets displacement være 0. Vær opmærksom på at tælle denne afstand rigtigt, når linear probing passerer tabelstørrelsen og tæller vide fra plads 0 i arrayet. Så en mere præcis definition af displacement er altså antallet af gange, at linear probing tæller én op (modulo ...) i håb om at finde den rigtige plads.

For hver at de ti nævnte konstanter skal du registrere ved kørsel på »hamlet«:

- den maksimale displacement, som forekommer
- summen af alle displacements som er registreret under søgninger i tabellen under kørslen.

Giv din egen vurdering af resultaterne.

(3)