Några svar till
TDDC91 Datastrukturer och algoritmer

2015-10-26

Följande är lösningsskisser och svar till uppgifterna på tentan. Lösningarna som ges här ska bara ses som vägledning och är oftast inte tillräckliga som svar på tentan.

2. 1. (A) N.
   2. (B) $\log N$.
   3. (C) $N \log N$.
   4. (E) RN.
   5. (G) $(N+R)\log N$. För ett givet värde på i exekveras den inre loopen ungefär $(N+R)/i$ gånger. Totalt får vi alltså $(N+R)(\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\ldots\frac{1}{N})$ vilket asymptotiskt sett är samma som $(N+R)\ln N$.

3. Den här lösningen söker efter nyckeln x utan att nödvändigtvis hitta rotationsindexet r. Vi etablerar invarianten att om x finns i arrayen b så finns x i $b[lo \ldots hi]$. Initialt sätter vi $lo = 0$ och $hi = N - 1$. Sätt nu $mid = (lo + hi)/2$. Det finns sex fall:

   - om $hi < lo$ returnera false
   - annars om $b[mid] = x$ returnera true
   - annars om $b[lo] \leq x < b[mid]$ gör en vanlig binärsökning efter x i $b[lo \ldots mid - 1]$.
   - annars om $b[mid] < x \leq b[hi]$ gör en vanlig binärsökning efter x i $b[mid + 1 \ldots hi]$.
   - annars om $b[mid] < b[hi]$ sätt $hi = mid - 1$
   - annars sätt $lo = mid + 1$

   Antalet jämförelser blir ungefär $2 \log N$ i värsta fallet.

   En annan lösningsvariant är att använda en modifierad binärsökning för att hitta rotationsindexet och sedan en vanlig binärsökning i relevant delarray.

   Observera att det inte går att lösa problemet i logaritmisk tid om nycklarna inte måste vara unika. Vi kan, till exempel, betrakta en array b med bara nollor förutom en enda etta någonstans i arrayens inre. Notera att b är en rotation av en sorterad array, men att det inte finns något effektivt sätt att söka efter nyckeln 1.

4. Lösningen är att använda ett så kallat 4-vägs *trie* (men utökat så att noderna också lagrar antalet strängar som satts in i trie:t som börjar med motsvarande prefix.). Förutom heltalsvariabeln för räknaren behöver vi fyra nodreferenser (antingen en array eller fyra separata variabler).

   Instansvariablerna blir då:

```
private Node root;

private static class Node {
   private int count;
   private Node a, c, t, g;
}
```

   Då kan vi sätta in fragment i strukturen på följande sätt:

```
public void add(String fragment) {
   root = add(root, fragment, 0);
}

private Node add(Node x, String fragment, int i) {
   if (i == fragment.length()) return x;
   if (x == null) x = new Node();
   x.count++;
   char c = fragment.charAt(i);
   if      (c == 'A') x.a = add(x.a, fragment, i+1);
   else if (c == 'C') x.c = add(x.c, fragment, i+1);
   else if (c == 'G') x.g = add(x.g, fragment, i+1);
   else if (c == 'T') x.t = add(x.t, fragment, i+1);
   return x;
}
```

   Att räkna antalet fragment som börjar med ett givet prefix får följande implementation:

```
public int prefixCount(String prefix) {
   Node x = root;
   for (int i = 0; i < prefix.length() && x != null; i++) {
      char c = prefix.charAt(i);
      if      (c == 'A') x = x.a;
      else if (c == 'C') x = x.c;
      else if (c == 'G') x = x.g;
      else if (c == 'T') x = x.t;
   }
   if (x == null) return 0;
   else return x.count;
}
```

   Operationen `prefixCount` tar tid proportionell mot W i värsta fallet.