# Några svar till tenta i TDDC91 Datastrukturer och algoritmer

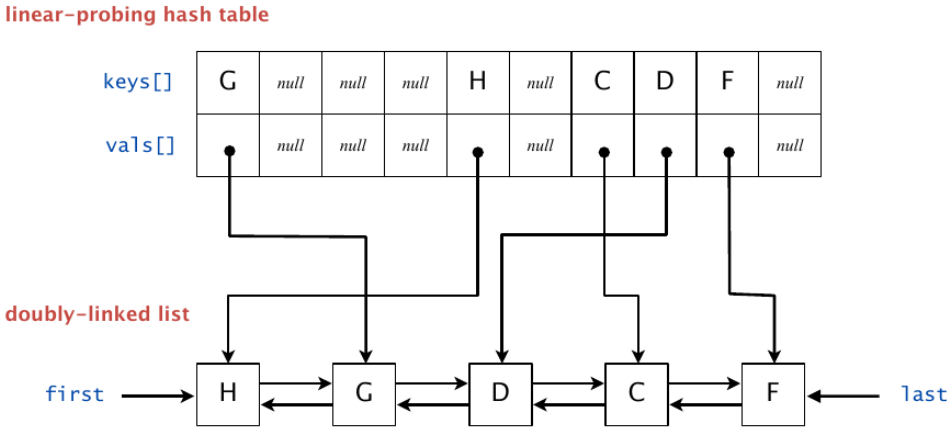
2016-01-05

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

2. Vi använder två datastrukturer :

   - En dubbellänkad lista innehållande de $N$ nycklarna i cachen med den mest nyligen cachade nyckeln först och den minst nyligen cachade nyckeln i slutet.
   - En symboltabell innehållande de $N$ nycklarna i cachen där värdet är en pekare till noden i den dubbellänkade listan innehållande den nyckeln.

   För att möta prestandakraven använder vi en hashtabell med linjär sondering och kapacitet $2N$. (En hashtabell med separat länkning går också bra att använda.)




   `inCache(Key key)`:

   - Använd symboltabellen för att avgöra om nyckeln finns i cachen.

   `cache(Key key)`:

   - Om nyckeln finns i cachen: Flytta motsvarande listnod från dess nuvarande position till början av listan.
   - Om nyckeln inte finns i cachen:
     - Ta bort den sista noden i den länkade listan (och ta bort motsvarande nyckel i symboltabellen) och
     - lägg till en ny nod längst fram i den länkade listan med den givna nyckeln (och sätt in en motsvarande post i symboltabellen).

3. (a) Algoritm 1: $T(N) = 2T(N/2) + c$
   Algoritm 2: $T(N) = T(N/2) + c$
   Algoritm 3: $T(N) = 2T(N/2) + cN$

(b) Algoritm 1: $\mathcal{O}(N)$
Algoritm 2: $\mathcal{O}(\log N)$
Algoritm 3: $\mathcal{O}(N \log N)$

4. Idén är att betrakta de $N$ nycklarna i stigande ordning, så att dublettnycklar finns bredvid varandra. Man kan tänka sig att algoritmen blir lik merge-steget i mergesort om vi skulle foga samman fler än två arrayer.

   - Iterera först över nycklarna i respektive array för att se om det finns någon dublett i någon av dem.
   - Initialisera ett AVL-träd med $k$ nyckel-värdepar där nyckeln är den första (minsta) nyckeln i den $i$:te sorterade arrayen och värdet är arrayens index $i$.
   - Upprepa till AVL-trädet är tomt eller en dublett upptäcks:
     - Ta bort noden med minst nyckel från AVL-trädet och låt $i$ vara index för arrayen den minsta nyckeln kom från.
     - Om nästa kvarvarande nyckel i array $i$ inte redan finns i AVL-trädet, sätt in nyckeln och associera den med värdet $i$.
     - Annars, stanna (dublett upptäckt).

   Mängden extra minne som används är proportionell mot $k$ eftersom AVL-trädet har som mest $k$ poster och vi behöver hålla reda på ett index i var och en av de $k$ sorterade arrayerna. Den totala exekveringstiden är proportionell mot $N \log k$. Flaskhalsen är insättning och borttagning i AVL-trädet. Kostnaden per sådan operation är $\log k$ eftersom AVL-trädet innehåller som mest $k$ nycklar.