Kortfattade lösningsförslag för tentamen i Datastrukturer (DAT036) från 2013-04-05

Nils Anders Danielsson

1. Notera att samma element sätts in i trädet varje gång. Tidskomplexiteten blir $\Theta(n^2)$.

2. Grafrepresentation: Noder är numrerade från 0 till $n-1$, där n är antalet noder. Grafstrukturen representeras av grannlistor: en array med storlek n, där position i innehåller en länkad lista med nod is direkta efterföljare.

   Vi kan implementera algoritmen i uppgiftsspecifikationen på följande sätt (pseudokod):

```
List<Integer> topological-sort(Graph g) {
  // Stacken.
  List<Integer> stack = new List<Integer>;

  // Array som visar om en viss nod besökts.
  Boolean visited[] = new Boolean[g.number-of-nodes];

  // Initialisering av arrayen.
  for (Integer i = 0; i < g.number-of-nodes; i++)
    visited[i] = false;

  // En lokal procedur med tillgång till variablerna
  // stack och visited.
  void dfs(Integer i) {
    visited[i] = true;

    for all immediate successors j of i in g {
      if (not visited[j])
        dfs(j);
    }

    // Postordnings-push.
    stack.push(i);
  }
```

```
  // Djupet först-sökning.
  for (Integer i = 0; i < g.number-of-nodes; i++) {
    if (not visited[i])
      dfs(i);
  }

  // Den topologiskt sorterade listan.
  return stack;
}
```

3. *Enkel lösning.* Sortera arrayen, summera de k största elementen. Värstafallstidskomplexitet med t ex merge sort eller heap sort: $O(n \log n) + O(k) = O(n \log n)$. (Notera att N inte används i svaret. Om man använder radixsortering e d så kan det vara lämpligt att nämna hur N påverkar tidskomplexiteten.)

   En annan lösning. Använd en prioritetskö (en binär minheap) för att beräkna de k största elementen: gå igenom arrayen och sätt in varje element i kön, och ta bort det minsta elementet så fort kön innehåller $k + 1$ element. Extrahera därefter alla tal i kön (med delete-min), och summera dem. Värstafallstidskomplexitet: $O(n \log k) + O(k \log k) = O(n \log k)$.

4. Använd till exempel två hashtabeller, en för udda tal och en för jämna tal.

5. Betrakta följande sekvens bestående av n operationer (för $n \geq 2$):

   - Först $2^{\lfloor \log_2 n \rfloor - 1}$ insättningar.
   - Därefter omväxlande insättningar och borttagningar. Notera att var och en av de här operationerna leder till en fördubbling eller halvering av arrayens längd.

   Låt oss analysera tidskomplexiteten för en godtycklig sådan sekvens. De första insättningarna har tidskomplexiteten $\Theta(2^{\lfloor \log_2 n \rfloor - 1}) = \Theta(n)$. Varje efterföljande steg har tidskomplexiteten $\Theta(n)$. Antalet efterföljande steg är

   $$n - 2^{\lfloor \log_2 n \rfloor - 1} \geq n - 2^{\log_2 n - 1} = n - \frac{n}{2} = \frac{n}{2}.$$

   Den totala tidskomplexiteten blir därför $\Omega(n^2)$.

6. Följande implementation är linjär eftersom konstant arbete utförs för varje nod:

```
public void reverse() {
    ListNode prev = null;
    ListNode cur  = head;
    ListNode next;

    while (cur != null) {
        next     = cur.next;
        cur.next = prev;
        prev     = cur;
        cur      = next;
    }

    head = prev;
}
```