Föreläsning
Datastrukturer (DAT036)

Nils Anders Danielsson

2012-10-31

Repetition

- Tidskomplexitet.
- Kostnadsmodeller.
- Asymptotisk komplexitet/notation.
- Analys av algoritmers tidskomplexitet.

Pseudokod

- Blandning av programmeringsspråk, matematisk notation och naturligt språk.
- Mål: fokusera på det viktiga, undvik onödiga detaljer.
- Ibland kan det vara bra med fler detaljer, ibland färre.

Mergesort

Pseudokod:

```
// Sorterar xs.
mergesort(xs) =
  if length of xs <= 1 then
    return xs
  else
    ys = first  half of xs
    zs = second half of xs
    return merge(mergesort(ys),
                 mergesort(zs))
```

Om $|\texttt{xs}|$ är udda:
Låt `ys` innehålla ett element mer än `zs`.

Mergesort

```
// Slår ihop två sorterade listor.
// Resultatet är sorterat.
merge(xs, ys) =
  if      xs is empty then return ys
  else if ys is empty then return xs
  else
    x = first element in xs
    y = first element in ys
    if x <= y then
      return merge(tail of xs, ys).addFirst(x)
    else
      return merge(xs, tail of ys).addFirst(y)
```

Mergesort

I kursboken:

- Mergesort för arrayer.
- Detaljerad kod.

Mergesort

Tidskomplexitet (om <= tar konstant tid):

- `merge`: Linjär i listornas längd: $\Theta(|\texttt{xs}| + |\texttt{ys}|)$.
- `mergesort`:

$$
\begin{aligned}
T(0) &= 1 \\
T(1) &= 1 \\
T(n) &= n + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right), \text{ om } n \geq 2
\end{aligned}
$$

Mergesort

Antag att $n = 2^k$, $k \in \mathbb{N}$.

$$
\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2(n/2 + 2T(n/4)) \\
&= 2n + 4T(n/4) \\
&= 2n + 4(n/4 + 2T(n/8)) \\
&= 3n + 8T(n/8) \\
&= \ldots \\
&= \log_2 n \cdot n + nT(n/n) \\
&= n \log_2 n + n \\
&= \Theta(n \log n)
\end{aligned}
$$

Mergesort

Är lösningen $L(n) = n \log_2 n + n$ korrekt?
Induktionsbevis:

- $k = 0$:

$$\begin{aligned} T(1) &= 1 \\ &= 0 + 1 \\ &= 1 \log_2 1 + 1 \\ &= L(1) \end{aligned}$$

Mergesort

Är lösningen $L(n) = n \log_2 n + n$ korrekt?
Induktionsbevis:

- $k > 0$:

$$
\begin{aligned}
T(2^k) &= 2^k + 2T(2^{k-1}) \\
&= 2^k + 2L(2^{k-1}) \\
&= 2^k + 2(2^{k-1} \log_2 2^{k-1} + 2^{k-1}) \\
&= k2^k + 2^k \\
&= 2^k \log_2 2^k + 2^k \\
&= L(2^k)
\end{aligned}
$$

Mergesort

Om $n \geq 1$ inte är en tvåpotens:

$$
\begin{aligned}
T(n) &= \\
T(2^{\log_2 n}) &\leq \quad \{T \text{ är monoton}\} \\
T(2^{\lceil \log_2 n \rceil}) &= \\
O(2^{\lceil \log_2 n \rceil} \log 2^{\lceil \log_2 n \rceil}) &\leq \\
O(2^{\log_2 n + 1} \log 2^{\log_2 n + 1}) &= \\
O(n \log n)
\end{aligned}
$$

Mergesort

Fråga

Hur många jämförelser utförs?

Mergesort

Fråga

Hur många jämförelser utförs?

Svar: $O(n \log n)$.

Divide and conquer

- Algoritmteknik: söndra och härska (divide and conquer).
- Skapar man två delproblem av halv storlek på linjär tid så får man en $O(n \log n)$-algoritm.

Animationer

http://www.sorting-algorithms.com/

Problem/algoritm

Problem: sortera en lista.

Algoritmer:

- Insertion sort.
- Mergesort.
- Quicksort.
- ...

Abstrakt datatyp/datastruktur

Abstrakt datatyp (matematisk abstraktion): lista.

Datastrukturer (implementation):

- Array (dynamisk?).
- Enkellänkad lista.
- ...

Listor, stackar och köer: ADT-er

ADT	Operationer (exkl konstruerare)
Stack	push, pop
Kö	enqueue, dequeue
Lista	add(x), add(x, i), remove(x), remove(i), get(i), set(i, x), contains(x), size, iterator
Iterator	hasNext, next

(Inte exakta definitioner.)

Listor, stackar och köer: datastrukturer

ADT	Implementationer
Lista	dynamisk array, enkellänkad lista, dubbellänkad lista
Stack	lista
Kö	lista, cirkulär array

ADT-er

- Kan använda en ADT för att implementera en annan.
- Även på tentan!

Listor, stackar och köer: tillämpningar

Diskussionsuppgift

Vad kan man ha de här ADT-erna till?

Listor, stackar och köer: tillämpningar

Stackar
- Implementera rekursion.
- Evaluera postfix-uttryck.
- Topologisk sortering (grafalgoritm).
- ...

Köer
- Skrivarköer.
- Simulering av ”riktiga” köer.
- Topologisk sortering (grafalgoritm).
- Bredden först-sökning (grafalgoritm).
- ...

Listor ...

Länkade listor

Många varianter:

- Objekt med pekare till första noden, kanske storlek.
- Pekare till sista noden?
- Enkellänkad, dubbellänkad?
- Vaktposter (sentinels)? Först/sist/både och?

Dubbellänkad lista med dubbla vaktposter

```
ListNode {
   A contents;
   ListNode next;  // null omm sista vakten.
   ListNode prev;  // null omm första vakten.
}
```

Dubbellänkad lista med dubbla vaktposter

```
DoublyLinkedList {
  ListNode first, last;  // Ej null.
  int size;

  // Konstruerar tom lista.
  DoublyLinkedList() {
    first      = new ListNode();
    last       = new ListNode();
    first.next = last;
    last.prev  = first;
    size       = 0;
  }
}
```

Dubbellänkad lista med dubbla vaktposter

Övning

Implementera metoden `addFirst`.

Tips: Rita först upp vad som ska hända.

Dubbellänkad lista med dubbla vaktposter

```
void addFirst(A x) {
  node             = new ListNode();
  node.contents    = x;
  node.next        = first.next;
  node.prev        = first;
  first.next.prev = node;
  first.next       = node;
  size++;
}
```

Inga specialfall. (Vad händer utan vaktposter?)

Invariant

- Egenskap som ”alltid” gäller för ett objekt.
- Exempel: `first != null`.
- Exempel: `last.next = null`.
- Invarianter bryts ibland temporärt
  när objekt uppdateras.

Precondition

- Krav som förväntas vara uppfyllt
  då metod anropas.
- Exempel: `merge` kräver att listorna är sorterade.

Postcondition

- Egenskap som är uppfylld efter anrop,
  givet preconditions och invarianter.
- Exempel: `merge(xs, ys)` är en sorterad lista,
  givet att `xs` och `ys` är sorterade listor.

Assertion

Man kan testa vissa egenskaper med ”assertions”.
Kan vara smidigt för att hitta/undvika fel i labbar.

```
$ cat Fail.java
public class Fail {
  public static void main (String[] args) {
    assert(args.length == 2);
  }
}
$ javac Fail.java
$ java Fail
$ java -ea Fail ett två
$ java -ea Fail
Exception in thread "main" java.lang.AssertionError
        at Fail.main(Fail.java:3)
```

Assertion

```
// Skapar ny /intern/ listnod.
// Precondition: prev != null && next != null.
ListNode(A contents, ListNode prev, ListNode next) {
  assert prev != null && next != null;

  this.contents = contents;
  this.prev     = prev;
  this.next     = next;
}
```

Nu leder

```
new ListNode(x,first.prev,first.next)
```

till ett `AssertionError` (med `-ea`),
inte ett kryptiskt fel senare.

Övning

Vad är fel? Hur kan man rätta till felet?

```
// Reverserar listan.
void reverse() {
  ListNode here = first.next;
  while (here.next != null) {
    ListNode next = here.next;
    here.next     = here.prev;
    here.prev     = next;
    here          = next;
  }
}
```

Korrekthet

Hur kan man förvissa sig om att man löst en uppgift korrekt?

- Bevis. Kan vara svårt, ta mycket tid.
- Tester. Kan gå snabbare.

Tester

När ni jonglerar pekare (på papper eller i dator):

- Testa gärna lösningen.
- Några representativa fall:
  - Tom lista.
  - Lista med några element.
  - ...

```
// Implementera en operation som lägger till en lista i slutet av en
// annan lista. I värsta fallet ska operationen ta konstant tid.
// Exempel: Om man lägger till [3, 4, 5] i slutet av [1, 2] ska
// resultatet bli [1, 2, 3, 4, 5].

// Enkellänkade listor med en "vaktpost" i början samt pekare till
// vaktposten och sista noden:
class List<A> {
    class ListNode<A> {
        A           contents;
        ListNode<A> next;
    }

    ListNode<A> head;  // Pekar på vaktposten.
    ListNode<A> tail;  // Pekar på vaktposten om listan är tom,
                       // annars på den sista riktiga noden.

    // Skapar en tom lista.
    List() {
        head = new ListNode<A>();
        tail = head;
    }

    // Lägger till ys sist i listan, utan att förstöra ys.
    void append(List<A> ys) {
        tail.next = ys.head.next;
    }
}
```

Sammanfattning

- Mergesort: söndra och härska.
- ADT-er/datastrukturer.
- Listor, stackar, köer.
- Länkade listor, pekarjonglering.
- Invarianter, assertions m m.
- Tester.