Föreläsning
Datastrukturer (DAT036)

Nils Anders Danielsson

2012-11-05

Repetition

Förra gången:

- Listor, stackar, köer.
- Länkade listor, pekarjonglering.

Idag:

- Cirkulära arrayer.
- Dynamiska arrayer.
- Amorterad tidskomplexitet.
- Träd.

Cirkulära arrayer

Implementationsteknik för köer.

Cirkulära arrayer

```
public class CircularArrayQueue<A> {
    private A[] queue; // Invariant: queue.length > 0.
    private int size;
    private int front; // Nästa element (eller rear).
    private int rear;  // Nästa lediga (eller front).

    public CircularArrayQueue(int capacity) {
        if (capacity <= 0) {
            throw new NonPositiveCapacityException();
        }
        queue = (A[]) new Object[capacity];
        size = front = rear = 0;
    }

    ...
}
```

Cirkulära arrayer

```
public void enqueue(A a) {
    if (size == queue.length) {
        throw new FullQueueException();
    }

    size++;
    queue[rear] = a;
    rear = (rear + 1) % queue.length;
}
```

Cirkulära arrayer

```
public A dequeue() {
    if (size == 0) {
        throw new EmptyQueueException();
    }

    size--;
    A a = queue[front];
    queue[front] = null;  // Undvik minnesläckor.
    front = (front + 1) % queue.length;

    return a;
}
```

Dynamiska arrayer

- Arrayer som ändrar storlek dynamiskt.
- Smidigt.
- Hur påverkas tidskomplexiteten?

Dynamiska arrayer

```
public class DynamicArrayList<A> {
    private A[] list;
    private int size;

    public DynamicArrayList() {
        list = (A[]) new Object[0];
        size = 0;
    }

    public void add(A a) {
        if (size == list.length) {
            list = Arrays.copyOf(list, 2 * size);
        }
        list[size++] = a;
    }
}
```

Dynamiska arrayer

```
public class DynamicArrayList<A> {
    private A[] list;
    private int size;

    public DynamicArrayList() {
        list = (A[]) new Object[0];
        size = 0;
    }

    public void add(A a) {
        if (size == list.length) {
            list = Arrays.copyOf(list, 2 * size);
        }
        list[size++] = a;
    }
}
```

Vad är fel?

Testa!

Dynamiska arrayer

- Värstafallstidskomplexitet: $\Theta(n)$.
- Men vi kopierar ju inte hela tiden...
- Ca $n/2$ billiga operationer, en dyr: i medeltal konstant tid.
- *Amorterad* tidskomplexitet $\Theta(1)$.

Amorterad tidskomplexitet

- Man kan låta tidiga, billiga operationer "betala" för dyra, sena.
- Efter dubblering:
  n snabba insättningar,
  1 dubblering.
- Låt de snabba insättningarna "betala" $\Theta(1)$ extra.
- Återstående kostnad för dubblering: $\Theta(1)$.

Amorterad tidskomplexitet, formellt

Operationer: $o_1, o_2, o_3 \ldots$
Faktisk tidskomplexitet: $t_1, t_2, t_3 \ldots$
Amorterad tidskomplexitet: $a_1, a_2, a_3 \ldots$

Krav:

$$\forall n. \sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} a_i$$

I varje steg är den *totala* amorterade tiden en övre gräns för den faktiska tiden.

Amorterad tidskomplexitet, formellt

Operationer:	$o_1, o_2, o_3...$
Faktisk tidskomplexitet:	$t_1, t_2, t_3...$
Amorterad tidskomplexitet:	$a_1, a_2, a_3...$

"Insättning tar konstant amorterad tid" betyder:

- Insättning har amorterad tid a (konstant).
- Övriga operationer har amorterade tider a_i.
- Så att kravet uppfylls för *varje* tänkbar operationssekvens $o_1, o_2, o_3....$

Amorterad tidskomplexitet, formellt

Operationer: $o_1, o_2, o_3...$
Faktisk tidskomplexitet: $t_1, t_2, t_3...$
Amorterad tidskomplexitet: $a_1, a_2, a_3...$

"Insättning tar konstant amorterad tid" betyder:

- Insättning har amorterad tid a (konstant).
- Övriga operationer har amorterade tider a_i.
- Så att kravet uppfylls för *varje* tänkbar operationssekvens $o_1, o_2, o_3....$

Notera att många operationer kan ignoreras (orelaterade, sätt $a_i = t_i$).

Potentialmetoden

- Potentialfunktion $\Psi \in \mathit{Tillstånd} \to \mathbb{R}$.
- Krav: $\forall s.\Psi(s) \geq \Psi(s_0)$. ($s_i$: Tillstånd efter o_i.)
- Definiera $a_i = t_i + \Psi(s_i) - \Psi(s_{i-1})$.

Kravet uppfylls:

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (t_i + \Psi(s_i) - \Psi(s_{i-1}))$$

$$= \Psi(s_n) - \Psi(s_0) + \sum_{i=1}^{n} t_i \geq \sum_{i=1}^{n} t_i$$

Potentialmetoden: exempel

Dynamiska arrayer:

- Välj potentialen så att dyra insättningar betalas av i tid.
- En konstant extrakostnad per insättning bör räcka.
- Vi kan bara använda de $\lceil n/2 \rceil$ senaste extrabetalningarna.

Potentialmetoden: exempel

Dynamiska arrayer:

- Välj potentialen så att dyra insättningar betalas av i tid.
- En konstant extrakostnad per insättning bör räcka.
- Vi kan bara använda de $\lceil n/2 \rceil$ senaste extrabetalningarna.

Potential (n: längd, c: kapacitet, arrayens längd):

$$\Psi((c, n)) = \kappa(n - \lfloor c/2 \rfloor),$$

för någon konstant $\kappa > 0$.

Potentialmetoden: exempel

Potential:

$$\Psi((c, n)) = \kappa(n - \lfloor c/2 \rfloor),$$

för någon konstant $\kappa > 0$.

- Jag ignorerar tillstånd då listan inte existerar.
- $\forall s.\Psi(s_0) = 0 \leq \Psi(s)$.

Potentialmetoden: exempel

Potential:

$$\Psi((c, n)) = \kappa(n - \lfloor c/2 \rfloor),$$

för någon konstant $\kappa > 0$.

- Kan se det som att vi lägger κ "kronor" på varje nytt element.
- Pengarna används då arrayen dubbleras.

Potentialmetoden: exempel

Potential:

$$\Psi((c, n)) = \kappa(n - \lfloor c/2 \rfloor),$$

för någon konstant $\kappa > 0$.

- Endast en operation påverkar potentialen (om vi ignorerar skräpsamling): insättning.
- Analysen kan sluta fungera om klassen utökas!

Potentialmetoden: exempel

Potential:

$$\Psi((c, n)) = \kappa(n - \lfloor c/2 \rfloor),$$

för någon konstant $\kappa > 0$.

Snabb insättning:

$$\begin{aligned} a_i &= t_i + \Psi(s_i) - \Psi(s_{i-1}) \\ &= \Theta(1) + \kappa((n+1) - \lfloor c/2 \rfloor) - \kappa(n - \lfloor c/2 \rfloor) \\ &= \Theta(1) + \kappa \end{aligned}$$

Konstant!

Potentialmetoden: exempel

Potential:

$$\Psi((c, n)) = \kappa(n - \lfloor c/2 \rfloor),$$

för någon konstant $\kappa > 0$.

Långsam insättning:

$$\begin{aligned} a_i &= t_i + \Psi(s_i) - \Psi(s_{i-1}) \\ &= \Theta(n) + \kappa((n+1) - \lfloor 2n/2 \rfloor) - \kappa(n - \lfloor n/2 \rfloor) \\ &= \Theta(n) + \kappa(1 - (n - \lfloor n/2 \rfloor)) \\ &= \Theta(n) + \kappa(1 - \lceil n/2 \rceil) \end{aligned}$$

Konstant om κ väljs tillräckligt stor!

Övning

Vad händer om arrayens längd tredubblas?

Övning

Vad händer om vi lägger till 10 element varje gång?

Övning

Utöka klassen med en metod `m`.
Insättning och `m` ska inte båda kunna ges konstant amorterad tidskomplexitet.

Mer om amorterad tidskomplexitet

- Samma operation kan ges olika amorterad tidskomplexitet i olika analyser.
- Vissa operationer långsamma: kan vara problematiskt i realtidssammanhang.

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`		
`add(x, i)`		
`remove(x)`		
`remove(i)`		
`get(i)`		
`set(i, x)`		
`contains(x)`		
`size`		
`iterator`		
`hasNext/next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`		
`remove(x)`		
`remove(i)`		
`get(i)`		
`set(i, x)`		
`contains(x)`		
`size`		
`iterator`		
`hasNext/next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`		
`remove(i)`		
`get(i)`		
`set(i, x)`		
`contains(x)`		
`size`		
`iterator`		
`hasNext`/`next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`		
`get(i)`		
`set(i, x)`		
`contains(x)`		
`size`		
`iterator`		
`hasNext`/`next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`		
`set(i, x)`		
`contains(x)`		
`size`		
`iterator`		
`hasNext/next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`	$O(1)$	$O(n)$
`set(i, x)`		
`contains(x)`		
`size`		
`iterator`		
`hasNext/next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`	$O(1)$	$O(n)$
`set(i, x)`	$O(1)$	$O(n)$
`contains(x)`		
`size`		
`iterator`		
`hasNext/next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`	$O(1)$	$O(n)$
`set(i, x)`	$O(1)$	$O(n)$
`contains(x)`	$O(n)$	$O(n)$
`size`		
`iterator`		
`hasNext/next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`	$O(1)$	$O(n)$
`set(i, x)`	$O(1)$	$O(n)$
`contains(x)`	$O(n)$	$O(n)$
`size`	$O(1)$	$O(1)$
`iterator`		
`hasNext/next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`	$O(1)$	$O(n)$
`set(i, x)`	$O(1)$	$O(n)$
`contains(x)`	$O(n)$	$O(n)$
`size`	$O(1)$	$O(1)$
`iterator`	$O(1)$	$O(1)$
`hasNext`/`next`		
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`	$O(1)$	$O(n)$
`set(i, x)`	$O(1)$	$O(n)$
`contains(x)`	$O(n)$	$O(n)$
`size`	$O(1)$	$O(1)$
`iterator`	$O(1)$	$O(1)$
`hasNext/next`	$O(1)$	$O(1)$
`remove`		

Listor: tidskomplexitet

	Dynamisk array	Dubbellänkad lista
`add(x)`	$O(1)$ amorterat	$O(1)$
`add(x, i)`	$O(n)$	$O(n)$
`remove(x)`	$O(n)$	$O(n)$
`remove(i)`	$O(n)$	$O(n)$
`get(i)`	$O(1)$	$O(n)$
`set(i, x)`	$O(1)$	$O(n)$
`contains(x)`	$O(n)$	$O(n)$
`size`	$O(1)$	$O(1)$
`iterator`	$O(1)$	$O(1)$
`hasNext/next`	$O(1)$	$O(1)$
`remove`	$O(n)$	$O(1)$

Listor: tidskomplexitet

Några kommentarer:

- `remove`, `contains`:
  Givet att jämförelser tar konstant tid.
- Indexbaserade operationer för
  dubbellänkad lista med pekare till sista noden:
  Bättre än $\Theta(n)$ om man vet att
  `i` pekar ”nära” början eller slutet av listan.

Collections i Java

- Java Collections Framework.
- ADT $\sim$ interface, datastruktur $\sim$ klass.

Träd

- Ett träd är tomt eller icketomt.
- Ett icketomt träd består av:
  - En rot (ev med innehåll).
  - Noll eller flera icke-tomma delträd (ofta ordnade).

Träd

Terminologi:

- Förälder, barn, syskon, barnbarn o s v.
- Löv.

Träd

En representation:

```
class Tree<A> {
  class TreeNode {
    A        contents;
    TreeNode firstChild;   // null om löv.
    TreeNode nextSibling;  // null om sista barnet.
  }

  TreeNode root;  // null om trädet är tomt.
}
```

Träd

Höjd:

- Tomma träd har höjd -1.
- Löv har höjd 0.
- Övriga träd har höjd
  1 + högsta barnträdets höjd.

Ungefär: Antalet steg från roten till djupaste lövet.

Träd

Höjd:

```
int height(TreeNode n) {
  if (n == null) {
    return -1;
  } else {
    largestHeight    = -1;
    TreeNode current = n.firstChild;
    while (current != null) {
      largestHeight = max(largestHeight,
                          height(current));
      current       = current.nextSibling;
    }
    return 1 + largestHeight;
  }
}
```

Träd

Uttryckskträd:

- Representerar aritmetiska uttryck.
- Tal eller variabler i löven.
- Operationer i noderna.

Binära träd

Binära träd:

- Max två barn.
- Skiljer på vänster och höger barn även om det bara finns ett.

Binära träd

En representation:

```
class BinaryTree<A> {
  class TreeNode {
    A        contents;
    TreeNode left;   // null om vänster barn saknas.
    TreeNode right;  // null om höger barn saknas.
  }

  TreeNode root;  // null om trädet är tomt.
}
```

Träd

Kan gå igenom ett träds noder i olika ordning:

Preorder Roten, barnträden från vänster till höger, vart och ett ”i preorder”.

Postorder Barnträden från vänster till höger, roten.

Inorder För binära träd:
vänster barnträd, roten, höger barnträd.

Level-order Roten,
noder på djup 1 från vänster till höger,
noder på djup 2 från vänster till höger, ...

Fråga

Vilken traverseringsordning är lämplig för att beräkna ett uttryckträds värde?

Sammanfattning

- Cirkulära arrayer.
- Dynamiska arrayer.
- Amorterad tidskomplexitet.
- Träd.

Nästa gång:

- Prioritetsköer.
- Binära heapar.
- Binomialheapar.