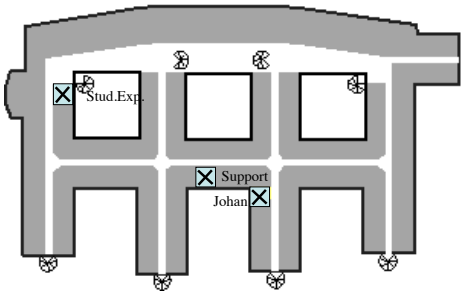
Datastrukturer och algoritmer

Föreläsning 1

Här sitter vi

MIT-huset våning 4



Personal

- Lärare
  - Johan Eliasson (johane@cs.umu.se)
- Handledare
  - Lucas Lindström
  - Tor Sterner-Johansson
  - Mikael Öhman

Innehåll

- Kurspresentation
  - Målsättning
  - Förkunskaper
  - Kursutvärdering
  - Upplägg
  - Översikt
- Föreläsning 1
  - Viktiga begrepp

Laborationer

- Tre stycken
  - OU2 Implementera tabeller (individuell)
  - OU4 Analysera algoritmer (individuell)
    - Experimentellt
    - Asymptotiskt (vi jobbar även med denna på en gruppövning)
  - OU1,3,5,6 Implementera en router (2 och 2). Görs som ett projekt med ett flertal inlämningar. Det första som ska göra är att ni ska hitta en grupp att jobba med (Klart senast på måndag 17:00)
  - Ingen särskild labsal är bokat för er så det är fritt fram att använda de obokade labben.

Mina målsättningar

- Alla ska höja sig ett par snäpp som programmerare, och inse att man har gjort det
- Förstå komplexitetsbegreppet
  - Tid och rum
- Ni ska ha lärt er några viktiga typer av algoritmer och datatyper
- Rolig kurs

Vad krävs för att nå dit?

- Ge kursen en ärlig chans
  - Kom i tid till föreläsningarna
  - Läs boken, förstå vad som står där (om något är oklart så fråga)
  - Gör övningar
  - Börja med laborationerna i tid
  - Utnyttja handledningen
  - Säg till om det är något som inte funkar

Förkunskaper

- Kunna implementera en godtycklig algoritmbeskrivning i Java (eller C)

Gruppövningar

- Gruppövning 1:
  - Övningar på det vi gått igenom tills dess
- Gruppövning 2:
  - Komplexitetsanalys (del av lab2)
- Gruppövning 3:
  - Övningar
- Gruppövning 4:
  - **Obligatorisk närvaro**. Redovisning av pågående arbete och reviderad projektplan
- Gruppövning 5:
  - Övningar

Websajten

- http://www.cs.umu.se/kurser/5DV043/VT09/
  - Schema
  - Uppdateringar
  - Formalia
  - Resultatredovisning
  - Gruppövningar
  - Föreläsningar
    - OH-bilder
  - Gamla tentor

Datastrukturer & Algoritmer

- Vi ska strukturera upp ert tänk kring program lite grann
- Genom att se strukturer och hitta likheter ska vi bygga mer komplexa program
- Ni ska lära er lite nya byggblock att bygga upp program av, och lära er i vilken situation de passar

Innehållsöversikt

- Algoritmer
  - Komplexitetsanalys, mm
  - Designprinciper
- Abstrakta datatyper
  - Stack, Kö, Listor, Träd, Graf, P-kö. Heap. Tabell, Sträng, Mängd, Sökträd, Tries, mm
- Sökning
  - Olika metoder och hur dom passar i olika lägen
- Sortering
  - 5-6 olika algoritmer, analys

Algoritmer

- Tids och rumsfrågor
  - Exekvering, implementering, underhåll
  - Tid vs rumskomplexitet
  - Analys av exekveringstider
    - Olika sätt, vad de står för och +/-
      - Experimentell
      - Asymptotisk
    - T(n) för en alg => O(n)
      - Primitiva operationer
      - Okulärbesiktning
    - Att beskriva algoritmer

Abstrakta DataTyper

- Beskrivningssätt
  - Organisation, Modell
  - Gränsytan (formell, informell)
- Grundbegrepp
  - Vad är en organisation, sorterad ADT, mm
  - Primär vs. sekundär struktur
  - Absolut vs. relativ komplexitet
- Sätt att implementera
  - +/-, när var hur
  - Komplexitet för de viktigaste operationerna
- Vart hittar vi ADT typiskt
  - Hur olika ADT hänger ihop med varandra
  - Algoritmer på ADT

Sökning (traversering)

- Vad är traversering
- Olika metoder för sökning i sekvenser
  - Linjär & Binär
  - +/-
  - Ställer de några krav på implementationen av ADTn

Sortering

- Varför sorterar vi?
  - Sorterad ADT vs sortering av data
  - Sökningen blir snabbare…ibland
- Hur kan vi sortera (olika former)
  - +/-
  - När passar dom?
  - Känna till någon bra alg. Bra
  - Stabil sortering

Design av algoritmer

- Idéer till 4 olika algoritmtyper
  - Brute force, Divide & Conquer, Greedy och Dynamisk programmering
  - Exempel på varje
  - +/-
  - Typiska användningar

Föreläsning 1

- Att bygga program ≈ bygga broar
- Brobygge
  - Specifikation
  - Beskrivning
  - Material
  - Verktyg
  - Uppföljning

Att bygga program

- Problembeskrivning…
- Systemdesign…
- Modelleringsverktyg
  - **Datatyper** – representera datat i programmet
  - **Algoritmer** – stegvis plan för att utföra något, modellera hur man löser problemet
  - **Kontrollstrukturer** – för att modellera flödet

Datatyper = op. + obj.

- Vilka objekt är det?
  - Vad vill vi modellera/representera/abstrahera
- Vad kan man göra med dem?
  - Vilka operationer/metoder

Gränssnitt

- Vad är ett gränssnitt?
  - Kontaktyta
  - Gränsen mellan två eller flera delar
  - Överenskommelse - oberoende av vem, vart, hur ska det ”passa”
    - Musköten 1700-talet

Gränssnitt

- Separerar
  - Funktion och implementationen
  - Användning och skapandet
  - Specifikationen och konstruktionen

Gränssnitt i ”datavetenskapen”

- Mellan centralenheter och periferienheter
- Mellan människan och maskinen
- Mellan mjukvarukomponenter
  - Funktioner/metoder
  - Datatyper
  -
  -

Begrepp

- Data - objekten som bär information
- Datatyp - ”värdemängd” samt operationer
- Sammansatt datatyp vs. enkel
- Homogen vs. Hetrogen
- Struktur
- Abstrakta datatyper
- Konstruerad datatyp
- Implementerad datatyp
- Fysisk – implementerad i språket/hårdvaran
- Konkret datatyp

Beskrivningssätt

- Modell - vardaglig, ”det man modellerar”
  - Kö
- Organisation
  - Den grundläggande naturen på objekten, linjärt ordnade, före och efter relation etc.
- Informell beskrivning
  - Gränsytan (nödvändig och minimal, kraftfull)
  - Informell beskrivning av operationernas funktion
  - Signaturdiagram
- Formell beskrivning - rent matematisk-logisk

Datastrukturer och algoritmer

Föreläsning 2

Innehåll

- Listor
  - Specifikation, Konstruktion
  - Algoritmmönster
- Riktade listor, länkade celler
  - Specifikation, Konstruktion
  - Dynamiska resurser

Lista

- Modell
  - Pärm
    - Bläddra, inspektera, lägga till, ta bort
- Konstruktion
  - Dynamiskt med hjälp av länkade celler
  - Statiskt med hjälp av fält/arrayer

Lista

- Ändligt antal linjärt ordnade element
- Första / sista element
- Före / efter relation
- Dynamisk datatyp
  - Struktur och storlek förändras under datatypens livslängd

Lista

- Generisk datatyp (polytyp)
  - Lista av *typ*
    - *Typ* kan vara av vilken typ som helst
- Homogen datatyp
  - Alla element har samma typ

Lista

- Element
  - Värde och position
- Struktur
  - Bortser från elementvärden
- Position
  - Plats i strukturen

Gränsyta till Lista

abstract datatype `List(val)`

auxiliary `pos`

```
Empty() -> List(val)
Insert(v:val,p:pos,l:List(val))->(List(val),pos)
Isempty (l:List(val)) -> Bool
Inspect (p:pos,l:List(val)) -> val
First (l:List(val)) -> pos
End (l:List(val)) -> pos
Next(p:pos,l:List(val)) -> pos
Previous(p:pos,l:List(val)) -> pos
Remove((p:pos,l:List(val))
                ->(List(val),pos)
```


Varför är specifikationen viktig?

- Nu kan vi skapa algoritmer för listor utan att behöva bry oss om ***hur*** listan verkligen implementeras.
- Ex, Söka efter elementet med värdet **v** i listan **list:**

```
If not isEmpty(list) Then
   p:= first(list)
   While Not (Next(p,list) = End(list)
         Or v = inspect(p, list)) Do
      p:= next(p, list)
   End
End
```


Tänkbar gränsyta i Java

```
public interface List{
   public boolean isEmpty();
   public Position insert(Object v, Position p);
   public Object inspect(Position p);
   public Position first();
   public Position end();
   public Position next(Pos p);
   public Position previous(Pos p);
   public Position remove(Pos p);
}
```


Implementation av gränsytan till Pos

```
public interface Position {
   public boolean eq(Position p);
}
```


Lista som Fält

- + Snabb inspektion av element
- - Fast reserverat utrymme
- - Kostsamt sätta in / ta bort element

Länkade strukturer

+ Insättning / borttagning går snabbt
+ Minnesutrymmet är proportionellt mot storleken
+ Allokera minne när det behövs
- Länkarna behöver också minnesutrymme
- Kommer bara åt listelement genom att traversera från listans början

n-länkad Cell

- Tippel som består av
  - ett värde
  - n stycken länkar
- Byggmaterial för andra datatyper
- *n*-länkad struktur
  - Objekt konstruerade med *n*-länkade celler
    - Listor, träd

1-Cell

```
private class Cell
{
   private Object data; //data portion
   private Cell next; //link to next node

   private Cell(Object dataPortion)
   {
      data = dataPortion;
      next = null;
   } //end constructor

   private Cell(Object dataPortion, Cell NextCell)
   {
      data = dataPortion;
      next = NextCell;
   } //end constructor
} //end Cell
```


Cell kan används för positioner

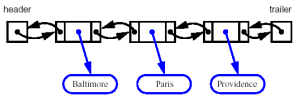
```
private class Cell implements Pos
{
   // se förra sidan för konstruktorer

   public boolean eq(Pos p) {
      return  (((Cell)p).next == this.next);
   }
}
```


Dubbellänkad Lista

- Länkar mellan elementen
- Elementen är en cell som består av länkar och värde

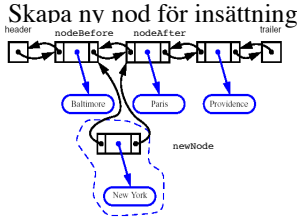
Dubbellänkad lista

- Före insättning



-Dubbellänkad lista

- Skapa ny nod för insättning



`newNode` skapas

`nodeAfter` = noden som ska vara efter den nya noden

`nodeBefore` = `nodeAfter`:s ”bakåt” länk

`newNode`:s ”framåt” länk = `nodeAfter`

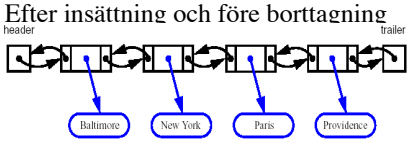
`newNode`:s ”bakåt” länk = `nodeBefore`

`nodeBefore`:s ”framåt” länk = `newNode`

`nodeAfter`:s ”bakåt” länk = `newNode`

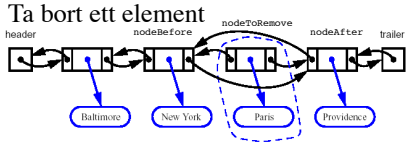
Dubbellänkad lista

- Efter insättning och före borttagning



Dubbellänkad lista

- Ta bort ett element



`nodeBefore` = `nodeToRemove`:s ”bakåt” länk

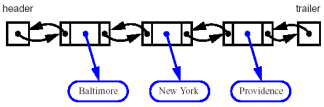
`nodeAfter` = `nodeToRemove`:s ”framåt” länk

`nodeBefore`:s ”framåt” länk = `nodeAfter`

`nodeAfter`:s ”bakåt” länk = `nodeBefore`

Dubbellänkad lista

- Efter borttagning



Riktad Lista

- Modell
  - Slalombana
- Kan bara flytta sig framåt
- Specialisering av Lista

Riktad Lista

- Previous och end behövs ej
- Isend lagts till

Gränsyta till Riktad Lista

abstract datatype **DList(val)**
auxiliary **pos**

```
Empty() -> DList(val)
Insert(v:val,p:pos,l:DList(val))
                    ->(DList(val),pos)
Isempty (l:DList(val)) -> Bool
Inspect (p:pos,l:DList(val)) -> val
First (l:DList(val)) -> pos
Isend (p:pos,l:DList(val)) -> Bool
Next(p:pos,l:DList(val)) -> pos
Remove((p:pos,l:DList(val))
              ->(DList(val),pos)
```


Riktad Lista

- Konstruerad som
  - Fält
  - Dubbellänkad Lista
  - Enkellänkad Lista
- Enkellänkad Lista
  - Mer ekonomisk

Enkellänkad Lista

- Problem vid insättning
- Lösning:
  - Representera position mha en länk till föregångarelementet
  - Listhuvud
    - Tomma objekt
    - Gränspositioner

Enkellänkad Lista

- Konstruktion utan huvud
  - Stopplänkvärde (nil, null)
  - Insättning före elementet X
    - Skapa en ny cell
    - Sätt in den efter X
    - Kopiera X:s värde till den nya cellen
    - Sätt X:s värde till v

Algoritmmönster

- Traversering
  - Besöker systematiskt alla element
- Sökning
  - Söker det första elementet som uppfyller ett bestämt villkor
- Filtrering
  - Filtrerar ut alla element som uppfyller ett bestämt villkor

Algoritmmönster

- Reduktion
  - Beräknar en funktion av objektets elementvärden
    - Ex. Summera alla tal i en lista
- Mappning
  - Transformera varje elementvärde i en datastruktur
    - Ex. multiplicera alla talen i en lista med 4

Länk

- Referens, pekare
- Objekt som refererar till annat objekt
- Konstrueras oftast som index i fält (kursor)
- Billigare kopiera länkar till objekt än objekten själva

Gränsyta till Länk

Abstract datatype	Link (obj)
Make	(*x* : obj) -> Link (obj)
Nil	( ) -> Link (obj)
Isnil	(*l* : Link (obj)) -> Bool
Follow	(*l* : Link (obj)) -> obj
Equal	(*l1*,*l2* : Link (obj)) -> Bool

Dynamiska resurser

- Skapar dataobjekt för tillfälliga behov
- `Kill`- lösgör resurser
  - Jämför med i `free` C
- `Create` - reserverar resurser
  - jämför med `malloc`, `calloc`, osv i C
- Vålnader
  - minne som är avallokerat, men som vi fortfarande refererar till (uppstår ej i java)
- Sophämtning
  - Identifierar och återvinner objekt som inte utnyttjas