ITK:P1 Föreläsning 7

Algoritmer och datastrukturer

DSV Marie Olsson

1

Datasamlingar

- Förra föreläsningen tittade vi på fält/arrayer
- Vid initieringen bestäms hur mycket data som ska rymmas i fältet
- När man skriver ett program så vet man inte alltid hur mycket data som sedan ska lagras
- Det vore därför ibland praktiskt med en datasamling som kunde utökas dynamiskt

2

Klassen Vector

- I Java har det sedan den allra första versionen funnits en lättanvänd dynamisk array
  `java.util.Vector`
- Skapas enkelt genom t ex:
  `Vector<Typ> v = new Vector<Typ>();`
- Instanser kan sedan läggas in och tas bort
  `add(Typ t);` `remove(Typ t);`

3

Klassen Vector

Används enligt:

```
import java.util.*;

public class Test {
  public static void main(String[] args) {
    Vector<String> vektor = new Vector<String>();
  }
}
```

4

Algoritmer

- En algoritm är beskrivningen för hur man löser ett givet problem
- Behöver inte vara skriven i programkod
- Som ett recept i en kokbok
- Den datalogiska definitionen:

”Ett ändligt antal instruktioner som löser ett problem.”

5

Algoritmer

- Algoritmer har funnits sedan länge inom områden som t ex arkitektur och matematik
- Här kommer en algoritm som är daterad till 1200-talet men som säkert är ännu äldre
- Leonardo Fibonaccis *fibonaccital*:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ...

6

Leonardo Fibonacci

- Om man betraktar en gran- eller tallkotte utgående från basen (fästpunkten), så bildar kottens fjäll spiraler såväl med- som motsols. Om kotten är hel, är antalet spiraler i kotten lika med Fibonacci-talen 5,8 eller 13.
- Fröna i en solros bildar även spiraler med- och motsols och antalet spiraler kan vara lika med Fibonacci-talen 34,55,89,144 och t.o.m. 233. Fibonacci-talen påträffas även i t.ex. snäckors spiralstrukturer.

http://www.edu.fi/svenska/oppimateriaalit/arkimatematiikkaa/fibona.html

7

Fibonacci-algoritmen

```
public class Fibonacci {
    static final int ANTAL = 10;

    public static void main(String[] args){
      int low   = 1;
      int high  = 1;
      for(int i = 1; i <= ANTAL; i++){
        System.out.println("Fibo " + i + ": "+ high);
         high += low;
         low = high-low;
  }
}
```

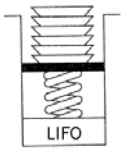
8

Abstrakta datatyper

- ADT:er
- En samling data som det går att utföra ett antal passande operationer på
- En ADT kan implementeras på olika sätt
- Det viktigaste är att det finns fungerande operationer
- I Java konstrueras ADT:er som klasser

9

LIFO



- En riktigt enkel algoritm
- Men mycket användbar
- LIFO = Last In First Out
- Lämplig ADT är en **stack**

10

Stack

- `java.util.Stack`
- `Stack<Typ> = new Stack<Typ>();`
- `Enkel att använda:`

```
public Typ push(Typ t);
public Typ pop();
public Typ peek();
public boolean empty();
```

11

FIFO

- I en simulering av biljettförsäljningen på en biograf vore det inte så kul för vissa besökare om de blev inlagda i en stack
- Här passar det bättre med
  First In First Out
- En ännu enklare algoritm där passande ADT heter **kö**/queue

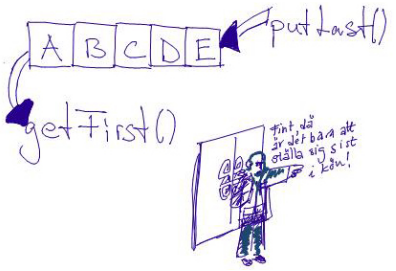
12

Kö

- Fr o m Java 1.5 finns det en färdig klass men det är lätt att bygga en egen då de viktiga metoderna är:
  putLast() ELLER enqueue()
  getFirst() ELLER dequeue()
- I **Lektion 4** bygger vi en egen kö med hjälp av **`java.util.Vector`**

13

Kö och FIFO



14

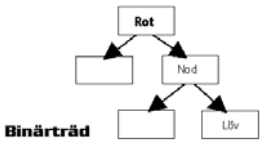
Länkade listor

- Dynamisk struktur
- Dubbellänkad lista
- En nackdel är ibland söktiden

15

Träd

- En struktur för snabbare åtkomst
- En mängd olika sorters träd



Binärträd

16

Hashtabell

- Hashtabeller = *associativa arrayer*
- Sökord är kopplade till värden
- Snabb åtkomst som är konstant oavsett antalet element som ingår i hashtabellen
- Från början fanns java.util.Hashtable
- I det senare JCF Java Collection Framework finns klasserna **HashMap** och **HashSet**
- Avancerade datastrukturer men lättanvända

17

Hashtabell

Skapa ett objekt enligt:

```
Map<String, Color> favoritFärger =
             new HashMap<String,Color>(23);
```

Lägg in värden:

```
favoritFärger.put(”Astrid”, Color.green);
favoritFärger.put(”Balthasar”, Color.red);
```

Plocka ut värden:

```
Color a = favoritFärger.get(”Astrid”);
Color b = favoritFärger.get(”Balthasar”);
```

18

Modulusoperatorn

- Heltalsdivision 9 / 5 = 1
- För att ta vara på resten från heltalsdivisioner så finns i de flesta språk en modulusoperator
- I Java representeras den av %
- Användbar på många sätt t ex

```
minuter  = tid / SEKUNDER_PER_MINUT;
sekunder = tid % SEKUNDER_PER_MINUT;
```

19

Villkorsoperatorn

- Ett arv från C och C++
- Går alltid att skriva om med en if – else

```
maxVärden[i] = (a > b) ? a : b;
```

- Samma som:

```
if (a > b)
  maxVärden[i] = a;
else
  maxVärden[i] = b;
```

20

Tack för idag

- Läs lite om den abstrakta datatypen **stack** i bokens kapitel 17
- Resten av detta kapitel spar vi till ITK:P2
- Exempel på hur man kan använda stackar och köer kommer i Lektion 4.

Tack för idag!

21