

## F07 - Tabell, relation, lexikon

5DV149 Datastrukturer och algoritmer  
Kapitel 6, 13, 16

Niclas Börlin  
[niclas.borlin@cs.umu.se](mailto:niclas.borlin@cs.umu.se)

2024-04-12 Fre

- ▶ Tabell
- ▶ Relation
- ▶ Lexikon
- ▶ OU3

## Tabell

- ▶ Modell:
  - ▶ Uppslagsbok
- ▶ Ändlig avbildning av **argument** eller **nycklar** på tabell-**värden**
  - ▶ Ingen begränsning på värdetypen
  - ▶ Nyckel-typen **nästan** obegränsad
    - ▶ Enda kravet är att **likhet** mellan två nycklar måste vara definierat
- ▶ **Sammansatt** datatyp — lagrar element
  - ▶ Varje element består av ett nyckel-tabellvärdepar
- ▶ **Oordnad** datatyp
  - ▶ Det finns inget första nyckel-värdepar
- ▶ **Dynamisk** datatyp
  - ▶ Storleken kan förändras under objektets livslängd



## Tabell, exempel

- ▶ Postadress:
  - ▶ Argument/nyckel: heltal
  - ▶ Värde: sträng
    - ▶ Postadress(90187) → "Umeå"
- ▶ Bilregister:
  - ▶ Argument/nyckel: sträng
  - ▶ Värde: post som beskriver bil
    - ▶ Bilregister("CBY328") → ("Toyota", "Avensis", 2013, diesel)
- ▶ Artikelregister:
  - ▶ Argument/nyckel: artikelnummer
  - ▶ Värde: post som beskriver artikeln
    - ▶ Artikelregister("32-2837") → ("Nätverkskabel", CAT 5, 79)

## Gränsyta till Tabell

```
abstract datatype Table(arg, val)
  Empty() → Table(arg, val)
  Insert(k: arg, v: val, t: Table(arg, val)) → Table(arg, val)
  Iempty(t: Table(arg, val)) → Bool
  Lookup(k: arg, t: Table(arg, val)) → (Bool, val)
  Remove(k: arg, t: Table(arg, val)) → Table(arg, val)
  Kill(t: Table(arg, val)) → ()
```

## Informell specifikation

- ▶ `Empty()` returnerar en tom tabell
- ▶ `Iempty(t)` avgör om tabellen `t` är tom
- ▶ `Lookup(k, t)` slår upp och returnerar värdet associerat med nyckeln `k` i tabellen `t`
- ▶ `Insert(k, v, t)` stoppar in värdet `v` i tabellen `t` och associerar det med nyckeln `k`
- ▶ `Remove(k, t)` tar bort värdet associerat med nyckeln `k` ur tabellen `t`

## Fält kontra Tabell

- ▶ De abstrakta datatyperna Fält och Tabell har likheter:
  - ▶ Bägge lagrar värden
  - ▶ Fältets `index` svarar mot tabellens `nyckel`
- ▶ Skillnader:
  - ▶ Begränsningar på argumenttypen/nyckeltypen:
    - ▶ Fält kräver att argumenttypen är diskret linjärt ordnad
    - ▶ Tabell kräver endast att `likhet` ska vara definierat
  - ▶ Tabell är en dynamisk datatyp, fält en statisk

## Nyckelvärden i Tabell

- ▶ Observera alltså att det enda som krävs av nyckeltypen är att `likhet` är definierat!
- ▶ I princip vilken typ som helst kan vara nyckeltyp
  - ▶ Strängar
  - ▶ Heltal
  - ▶ Poster
  - ▶ Listor
  - ▶ ...

## Konstruktion av tabell

### ► Fyra alternativ

- Tabell konstruerad som Fält
- Tabell konstruerad som Lista av par
- Tabell konstruerad som Hashtabell (senare)
- Tabell konstruerad som Binärt sökträd (senare)

### ► Egenskaper, bästa sökprestanda

Konstruktion	Osorterade nycklar	sorterade	Statisk/dynamisk
Fält	$O(n)$	$O(\log n)$	statisk
Lista av par	$O(n)$	$O(n)$	dynamisk
Hashtabell	$O(1)$	$O(1)$	statisk/dynamisk
Binärt sökträd	-	$O(\log n)$	dynamisk

## Tabell som Fält

### ► Tabell kan konstrueras som Fält under vissa förutsättningar

1. argumenttypen är **diskret linjärt ordnad**,
2. argumenten är relativt väl samlade
3. det går att hitta en **ODEF**-konstant, dvs. en konstant som betyder att tabellvärdet är "ej definierat",

### ► **Table-Lookup**(k, t) blir **Array-Inspect-value**(ix, t.a), där ix är uträknat från nyckeln k

### ► Exempel:

#### ► Postadress.

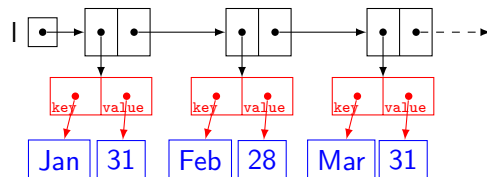
- Argument: heltal
- Intervall: 10000-99999
- ODEF: NULL eller tomma strängen

#### ► Bilregister

- Vi kan göra nyckeltypen "sex tecken" diskret linjärt ordnad om vi t.ex. definierar ordningen  $0 < \dots < 9 < A < \dots < Z$
- Men intervallet blir  $23^3 \cdot 10^2 \cdot 33 = 40151100$  element långt!

## Tabell som Lista av par

### ► Lista av par



### ► Tabellen använder en Lista

- Listan kan ha vilken konstruktion som helst (här visas 1-celler)
- Listans komponenter visas i svart

### ► Tabellen använder element som består av Poster (**structs**) med nyckel-värde-par

- Fältet **key** innehåller en länk till en *nyckel*
- Fältet **value** innehåller en länk till ett (*tabell*-)värde
- Tabellens komponenter visas i rött

### ► Nycklarna och tabellvärden stoppas in i tabellen av användaren

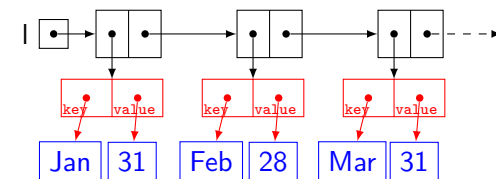
- Här består nyckeln av en sträng och värdet av ett heltal
- Användarens komponenter visas i blått

### ► Denna konstruktion är implementerad i kodbasen

## Tabell som Lista av par — dubletter

### ► Hur hanterar vi **dubletter**, dvs. två nyckel-värde-par med samma **nyckel**?

#### ► Ex. Stoppa in (Feb, 29)



### ► Vi får två huvudalternativ:

1. Kolla om det **redan finns** ett par med samma nyckel
  - **Byt ut** det gamla paret mot det nya, alt. det gamla tabellvärdet mot det nya
  - **Insert()** måste modifieras
2. Sätt in det nya paret **först** i listan utan att kolla
  - **Lookup()** måste leta framifrån och returnera första träffen
  - **Remove()** måste ta bort **alla** träffar

## Tillämpningar av Tabell (1)

- ▶ Benämna objekt, t.ex.
  - ▶ (Artikel-nr) 32-2837: Nätverkskabel
  - ▶ (Registration) CBY328: Toyota Avensis
- ▶ Associera **egenskaper** hos ett objekt med en nyckel, t.ex.
  - ▶ 32-2837:
    - Namn Nätverkskabel
    - Pris SEK 79
    - Hyllplacering E14
    - Lagerstatus 23 st

## Att jämföra nycklar

## Tillämpningar av Tabell (2)

- ▶ Representera **samband** mellan objekt, t.ex.
  - ▶ personer som äger en fastighet,
  - ▶ personer som är medlemmar av en klubb
- ▶ Kompilatorer och interpretatorer (symboltabeller)
  - ▶ Existens (är symbolen definierad?)
  - ▶ Attribut (värdetyp, räckvidd)
- ▶ Fält som Tabell (glesa matriser).
  - ▶ Nyckel: (rad, kolumn)
  - ▶ Ex. Matrisen

0	0	3
8	0	0
5	0	0

skulle lagras som

nyckel	värde
(2,1)	8
(3,1)	5
(1,3)	3

## Tabell, att jämföra nycklar

- ▶ Om nyckeltypen inte är inbyggd (t.ex. int, double) utan t.ex. Post måste man definiera och implementera en **likhetsoperation**
- ▶ T.ex. om nyckeln är en Post (artikel-nr, namn), definiera **likhet** som att **artikel-numren** är lika
- ▶ Om strukturen kräver att datat går att **sortera** behöver vi i stället definiera en **jämförelseoperation** som avgör om objektet **a** kommer före/lika/efter **b** i en sorteringsordning

## Definition av likhet

- ▶ Att definiera **likhet** är ofta inte en uppgift för datatypen, utan för **användaren** av datatypen
- ▶ Vad som är den bästa definitionen **varierar** ofta med tillämpningen
- ▶ Exempel:
  - ▶ För en nyckeltyp **artikelnummer**, ska "37-1463" och "371463" anses vara lika?
  - ▶ För en nyckeltyp **svenskt efternamn**, ska "Svensson" och "Svenson" anses vara lika?
  - ▶ För en nyckeltyp **flyttal**, ska 3.14156259 och 3.14156258 anses lika?
  - ▶ För en **Post** (artikelnummer, beskrivning), ska två poster anses lika om **artikelnumren** är lika eller måste **beskrivningarna** också vara lika?
  - ▶ För två strängpekare i språket C, ska strängarna anses lika om **innehållet** är lika, eller måste **adresserna** vara lika?

## Tippel, Post

## Blank

## Tippel, Post

- ▶ Två andra datatyper som liknar tabell
  - ▶ Associerar **argument** med **värden**
- ▶ Tippel (*tuple*) består av ***n* ordnade** element, ex. för koordinater  $a=(29, 4, 3)$ 
  - ▶ `Inspect-first(a)` → 29
  - ▶ `Inspect-second(a)` → 4
- ▶ Post (*record, struct*) är som abstrakt datatyp sett samma sak som Tippel, t.ex. för  $a=(32-2837, \text{Nätverkskabel}, 79)$ 
  - ▶ `Inspect-first(a)` → 32-2837
  - ▶ `Inspect-second(a)` → Nätverkskabel
- ▶ Skillnaden är att för Post brukar man använda **namn** för att ange fälten, för Tippel **heltal**

# Relation

- ▶ En **Relation** är en **egenskap** definierad för en **grupp** av **objekt**
- ▶ Exempel:
  - ▶ Objekten (Måndag, Tisdag, ..., Söndag) är relaterade till varandra via relationen **veckodag**
  - ▶ Objekten (Januari, Februari, ..., December) är alla relaterade till varandra via relationen **månad**
  - ▶ Objekten Måndag och September är **inte** relaterade till varandra

## Binära relationer

- ▶ En **binär** relation är en relation mellan **två** objekt
- ▶ För den binära relationen "mindre-än" med symbol  $<$  säger vi vanligen att " $a$  är mindre än  $b$ " ( $a < b$  i text) för t.ex.  $a = 4$  och  $b = 6$
- ▶ Formellt gäller att objektena  $a$  och  $b$  är **relaterade** till varandra enligt relationen "mindre-än" om och endast om  $a < b$ 
  - ▶ Motsatsen är att  $a$  och  $b$  är **inte** relaterade till varandra enligt relationen "mindre-än", vilket gäller om  $a \geq b$
- ▶ Ett annat uttryck är att  $a$  och  $b$  **har** eller **saknar** relationen "mindre-än"
  - ▶ Objekten  $a = 4, b = 6$  **har** relationen "mindre-än"
  - ▶ Objekten  $a = 9, b = 6$  **saknar** relationen "mindre-än"

## Relationstabeller (1)

- ▶ Relationer konstrueras ofta som tabeller
  - ▶ En nyckel, oftast heltal, används för att **enumerera** objekten som har relationen
  - ▶ Ex, persontabell

Nyckel	Namn
...	
3512	NB
3513	HB
...	

- ▶ Ex, fastighetstabell:

Nyckel	Namn
1	Bergsnäs 1:1
...	
42	Bergsnäs 2:4
...	

## Relationstabeller (2)

- ▶ Relationer mellan **olika** objektklasser kan konstrueras i en **egen** tabell med nyckelreferenser till objekten
  - ▶ Relationer kan också innehålla **attribut**
  - ▶ Varje rad i tabellen är en Tippel av nycklar eller attribut
    - ▶ Beskriver relationen mellan objekten

- ▶ Ex, fastighetsägartabell:

Fastighet	Person	Andel
...		
42	3512	0.5
42	3513	0.5
...		

- ▶ "Personen 3512 har en relation till fastigheten 42 med attribut 0.5" utläses "NB äger 50% av Bergsnäs 2:4"
- ▶ Är grunden för **relationsdatabaser**

## Blank

## Lexikon

- ▶ Ett **Lexikon** är som en tabell **utan tabellvärden**
- ▶ Lexikon är en s.k. **solitär** datatyp
  - ▶ Man kan göra **modifieringar av isolerade dataobjekt**, ex. lägga till, ta bort eller slå upp element, men **inte kombinera/bearbeta två eller flera** Lexikon
- ▶ Jämför den icke-solitära datatypen Mängd:
  - ▶ Det går att lägga till och ta bort element till både Mängd och Lexikon
  - ▶ Både en Mängd och ett Lexikon är ordnade datatyper
  - ▶ Det går att bilda **unionen** eller **snittet** av två mängder
  - ▶ Motsvarande för två Lexikon är inte definierat
    - ▶ "Det finns bara **ett** Lexikon av svenska ord"

## Experimentell komplexitetsanalys

## ► Experimentell

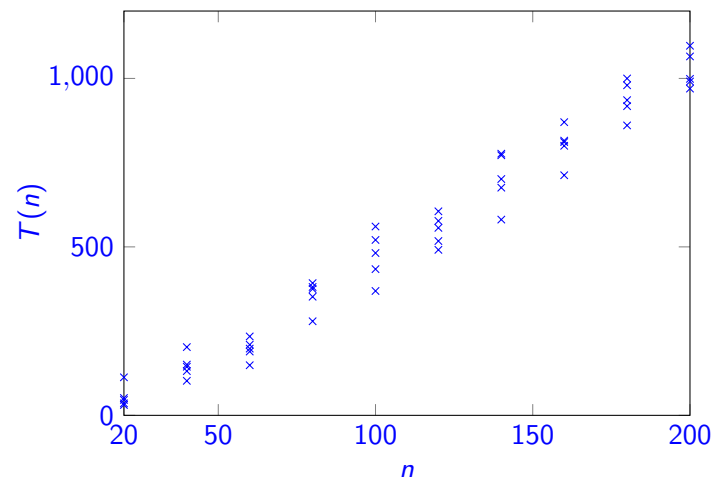
1. Kör programmet för olika **problemstorlekar**
2. **Mät tiden**
3. Uppskatta **trenden**

## ► Asymptotisk

1. Analysera algoritmen **teoretiskt**
2. Undersök vad som händer då  **$n$  blir stort**

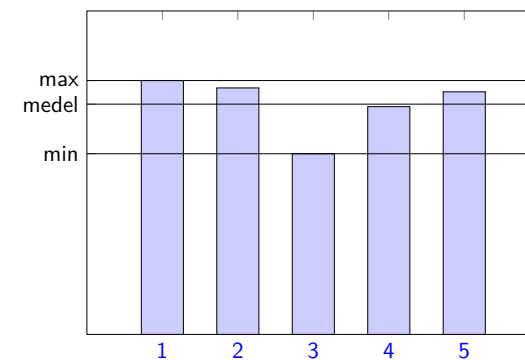
1. **Implementera** algoritmen
2. Kör programmet med varierande **datamängd**
  - Storlek
  - Sammansättning
3. Mät **tiden**  $T(n)$  då programmet körs
4. **Plotta**  $T(n)$ 
  - 4.1 Ansätt en **hypotes**, t.ex.  $g(n) = n^2$
  - 4.2 **Plotta**  $f(n) = T(n)/g(n)$
  - 4.3 Om  $f(n)$  går mot **positiv konstant** så är hypotesen troligen korrekt
  - 4.4 Om inte, ansätt en **annan hypotes**, t.ex.  $g(n) = n$

## Exempel på en plot



## Bästa, värsta, medel

- $T(n)$  för  $n = 80$ :



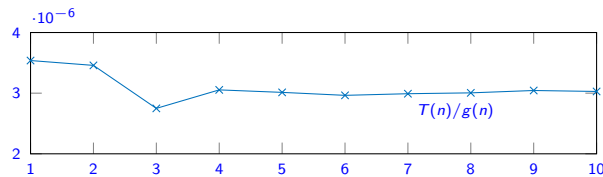
- Beroende på datats **sammansättning** kan algoritmen fungera **olika bra**
- Bubblesort för **redan sorterad lista** är  $O(n)$ 
    - I **medel-** och **värsta** fall  $O(n^2)$



## Kontrollera din slutsats

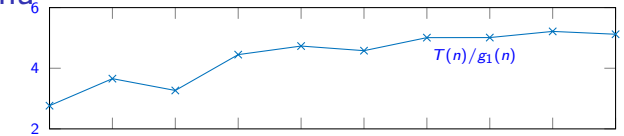
- ▶ Mät tiden för ett antal  $n$
- ▶ Ansätt ("gissa")  $g(n)$  från ordo-definitionen med stöd från den asymptotiska teorin
  - ▶ Om teorin säger att komplexiteten bör vara kvadratisk, ansätt  $g(n) = n^2$
- ▶ Plotta **uppmätt/hypotetisk tid** ( $T(n)/g(n)$ )
- ▶ Bör gå mot **positiv konstant** för stora  $n$  om  $g(n)$  är korrekt

$n$	1	2	3	4	5	6	7	8	9	10
$g(n) = n^2$	1	4	9	16	25	36	49	64	81	100
$T(n) (\cdot 10^{-6})$	3.54	14	25	49	75	107	147	192	247	303
$T(n)/g(n)$	3.54	3.46	2.75	3.05	3.01	2.96	2.99	3.01	3.04	3.03



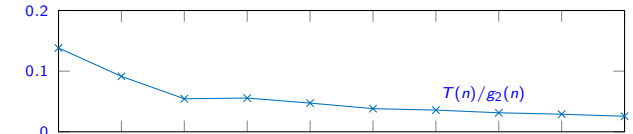
## Testa hypoteserna

$$g_1(n) = n$$



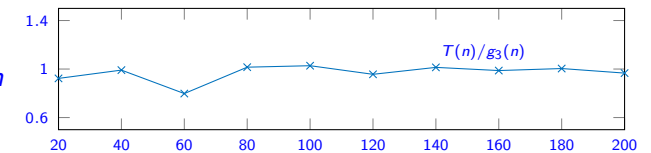
Fortsätter växa.  $g_1$  växer för långsamt!

$$g_2(n) = n^2$$



Fortsätter avta.  $g_2$  växer för snabbt!

$$g_3(n) = n \log n$$



Konvergerar.  $g_3$  växer lagom snabbt!  $n$

## Experimentell analys

- ▶ Fördelar:
  - ▶ Behöver inte **källkoden**
  - ▶ Behöver "bara" ett **körbart** program
- ▶ Begränsningar:
  - ▶ Måste **implementera** och testa algoritmen
  - ▶ Experimenten kan endast utföras på en **begränsad** (liten) mängd data
  - ▶ Man **kan missa** viktiga testdata (specialfördelningar)
  - ▶ Hård- och mjukvaran måste vara densamma för **alla körningar**
  - ▶ Modern, "intelligent", strömsparande mjuk- och hårdvara kan **variera hastigheten** på processorn

## Introduktion till OU3

- ▶ Samma gränsyta/interface — olika implementationer
- ▶ Table-as-list, Tabell som Lista av par
  - ▶ Given i kodbasen
- ▶ Move-to-front-table (MTFTable)
  - ▶ En liten modifikation av Table-as-list som ni ska skriva — utgå från Table-as-list
- ▶ Table-as-array
  - ▶ Skriv koden från början — ni ska använda er av Array

- ▶ Givet en tabellimplementation, implementera två andra!
  - ▶ En mindre variant — move-to-front-table
  - ▶ En helt annan variant — table-as-array
- ▶ Kör ett givet testprogram för alla tre tabellerna för att se att koden är korrekt och se hur tidsåtgången för olika operationer varierar
  - ▶ Ni får jobba i par
  - ▶ Jobbar ni i par ska ni dessutom implementera två mindre tabellvarianter (sorterade) till
  - ▶ Mindre arbete per person och störst pedagogisk vinst
- ▶ Rapport med bland annat en presentation av resultatet från körningen och en analys

## Att tänka på (1)

- ▶ Kopiera Table-as-list till Table-as-move-to-front-list och modifiera koden
- ▶ Råd: Använd **inte** en `kill_function` inuti tabellen.
- ▶ Table-as-array:
  - ▶ Dubbletthantering — tillåta dubletter i tabellen eller ej?
  - ▶ Kom ihåg att använda **exakt** samma gränsyta som i de övriga tabellerna
  - ▶ Ni behöver hålla reda på hur många element i fältet som är upptagna
    - ▶ Modifiera **struct table**
  - ▶ Inte tillåtet med "hål" i tabellen
    - ▶ Annars blir det svårt att hålla de givna komplexiteterna
- ▶ Om ni ska implementera sorterade versioner — de flesta funktionerna kommer vara identiska

## Att tänka på (2)

- ▶ Testprogrammet är också ett tidtagningsprogram
- ▶ Programmet testar tiden för  $n$  anrop med problemstorlek  $n$  till följande funktioner
  1. `Insert()`
  2. `Remove()`
  3. `Lookup()` för **icke existerande** nycklar
  4. `Lookup()` för **existerande** nycklar, slumpmässigt (likformigt) fördelade (*random lookups*)
  5. `Lookup()` för existerande nycklar med **sned** fördelning (en del nycklar slås upp oftare än andra) (*skewed lookups*)

## Att tänka på (3)

- Tidtagningen är för  $n$  anrop till en av funktionerna i gränsytan, t.ex. `Insert()` för en problemstorlek på  $n$ 
  - Det betyder att om den asymptotiska komplexiteten (för ett funktionsanrop) är t.ex. konstant ( $O(1)$ ) och testprogrammet ger tiden för  $n$  anrop så behöver ni använda  $g(n) = n \cdot 1$  i er experimentella analys
  - Om funktionen är  $O(n)$  behöver ni använda  $g(n) = n \cdot n = n^2$ , osv.
- Det finns bash-script som hjälper er att generera filer med testdata och matlab-script som hjälper er att generera grafer
  - Dessa är sparsamt dokumenterade, men skriv gärna frågor på handledningen och/eller i lämpligt discussionsforum

## Insidan

