F01 - Begrepp och fält (*array*)

5DV149 Datastrukturer och algoritmer

Niclas Börlin
niclas.borlin@cs.umu.se

2024-03-20 Ons

Introduktion

Innehållsöversikt

- Abstrakta datatyper:
  - Fält, Lista, Stack, Kö, Träd, Graf, Tabell, Mängd, ...
  - ... med tillhörande algoritmer, begrepp, relationer, mm
- Algoritmer:
  - Pseudokod som ett sätt att beskriva algoritmer
  - Tids- och rums-komplexitet
  - Designprinciper
- Sökning och sortering
- Testning och felsökning

Abstrakta DataTyper (ADT)

- Kursen visar på
  - Hur kan man beskriva en ADT på ett effektivt och entydigt sätt
  - Grundbegrepp kring ADT
  - Olika sätt att konstruera ADT:er
  - Hur olika ADT:er hänger ihop med varandra
  - Skillnad mellan abstraktion och implementation
- Fokus ligger på att använda datatyperna och förstå när man bör välja en given implementation
  - Tillämpnings- och språkberoende

Datatyper = objekt + operationer

- Vilka objekt vill vi använda?
  - Vad vill vi modellera / representera / abstrahera?
- Vad vill vi göra med dem?
  - Vilka operationer är aktuella?
- Exempel:
  - Heltal med operationerna plus (+), minus (-), etc.
  - Tabell och "lägg till", "slå upp", etc.
  - Kö och "stoppa in" (sist), "ta ut" (först), etc.
- Data — objekt som bär information
  - Tex en sträng som representerar namnet på ett hotell,
  - ett heltal som representerar antalet rum i hotellet, etc.

Gränssnitt (*interface*), definition

- Vad är ett gränssnitt?
  - Kontaktyta
  - Gränsen mellan två eller flera delar.
  - Överenskommelse — mellan konstruktör och användare
    - Jämför reglagen i en bil.
- Ett gränssnitt separerar "vad" från "hur"
  - Funktion från implementation
  - Specifikation från konstruktion

Gränssnitt i datavetenskap

- Mellan centralenheter och periferienheter
  - Ex. USB-sladd
- Mellan människan och maskinen
  - Ex. tangentbord, skärm
  - Ex. GUI (*graphical user interface*)
- På denna kurs: mellan mjukvarukomponenter
  - Funktioner
  - Datatyper
  - API (*application programming interface*)

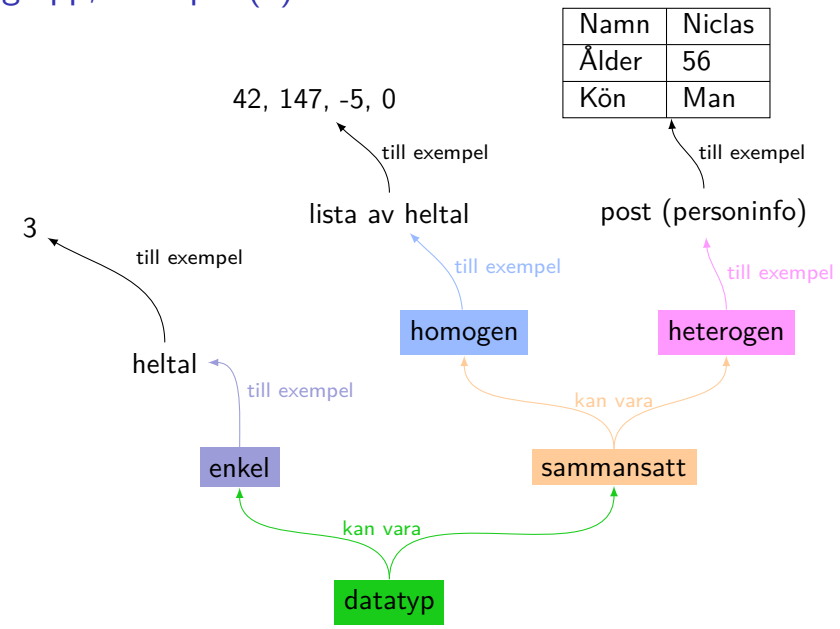
Grundläggande begrepp

Kapitel 2-3

Begrepp (1)

- Enkla datatyper
  - Saknar synlig inre struktur
  - Ex. heltal, flyttal, tecken
- Sammansatta datatyper
  - Består av delar (element), strukturerade på visst sätt, ex.
    - en lista av heltal,
    - en kö av printerjobb,
    - en struktur som beskriver ett hotell
  - En sammansatt datatyp är endera
    - homogen — alla element har samma typ, eller
    - heterogen — elementen kan ha har olika typ
  - En sammansatt datatyp är endera
    - ordnad — elementen har en inbördes ordning
      - det finns ett första element, ett nästa element, etc.
    - oordnad — elementen saknar inbördes ordning
      - det finns inget första element
  - De flesta sammansatta datatyperna är ordnade, t.ex. lista, fält,
    - Ett fåtal är oordnade, t.ex. mängd, post (`struct`)

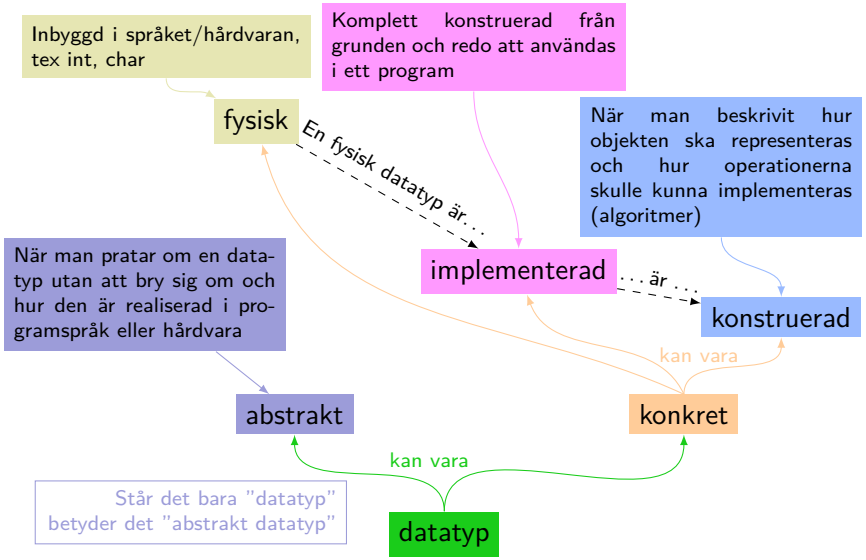
Begrepp, exempel (1)



Begrepp (2)

- Abstrakt datatyp
  - En datatyp definierad helt av sitt gränssnitt
  - Behöver inte ens vara implementerad!
- Konstruerad datatyp
  - Vi har bestämt hur datatypen ska representeras och organiseras internt
    - En Schackpjäs representerad som ett Heltal
    - En Kö som använder en Lista
- Implementerad datatyp
  - Koden finns, färdig att använda
- Fysisk datatyp
  - Implementerad i språket/hårdvaran, ex. 8-bitars heltal, 64-bitars flyttal, pekare, ...
- Konkret datatyp — samlingsnamn på konstruerad, implementerad, fysik datatyp
- Se kap 2.3 för fullständiga definitioner

Begrepp, exempel (2)



Ordning

- Vi kommer ofta att prata om en mängd av element är linjärt ordnade eller diskret linjärt ordnade
- Det innebär att:
  - Elementen är diskreta (ej kontinuerliga)
  - Det finns ett första element
  - För varje element x utom det första finns en unik föregångare y
  - Exempel: De naturliga talen 0, 1, ...
  - Motexempel: De reella talen (~~diskreta~~), färger (första färg?)
- Elementen är ordnade "på en linje"
  - Vi säger att en före-efter-relation är definierad:
    - a kommer före b omm det finns en sekvens

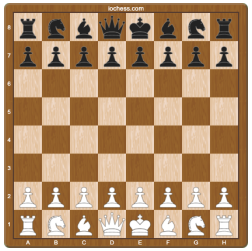
$$a = t_0, t_1, ..., t_k, t_{k+1} = b,$$

      där varje t_i är en föregångare till $t_{i+1}, i = 0, ..., k$
  - Motexempel: Punkter i 2D
- För en ändlig diskret linjär ordning gäller dessutom att
  - Det finns ett sista element
  - För varje element x utom det sista finns en unik efterföljare y
  - Exempel: Talen 1, 2,. . . , 7; månaderna januari, . . . , december.

Fält (*array*)

Kapitel 6

Fält



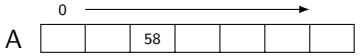
- Mental modell: Schackbräde eller excel-ark
  - Med hjälp av två parametrar — bokstav och siffra — kan man ange positionen för en bestämd ruta
- Organisation
  - Sammansatt datatyp — lagrar element
  - Homogen datatyp — alla elementvärden av samma typ
  - Ett n-dimensionellt fält organiserat som ett rätblock
  - Tillåts innehålla odefinierade elementvärden

Koordinater (index)

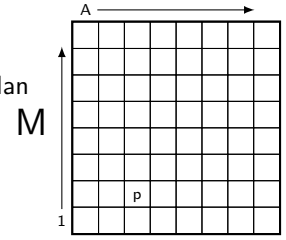
- Koordinatvärdena längs en enskild koordinataxel är en diskret linjärt ordnad typ, oftast heltal
- I praktiken säger vi att n koordinater (index): koordinat 1, koordinat 2, etc.
  - Ibland säger vi att vi har ett koordinatvärde som är en grupp av n värden

Olika slags fält

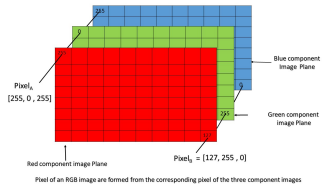
- En dimension: vektor
  - `A(2) = 58`
  - Index = (2)



- Två dimensioner: matris, schackbräde, svartvit bild
  - `M(2,C) = p`
  - Index = (2,C)
  - Notera att ordningen mellan axlarna och riktningen i *presentationen* är *godtycklig*



- Tre dimensioner: tensor, färgbild (RGB), volym
  - `I(0,0,R) = 255`
  - Index = `(0,0,R)`



Gränsyta till Fält (1)

```
abstract datatype Array(val, index)
  Create(lo, hi: index) → Array(val, index)
  Low(a: Array(val, index)) → index
  High(a: Array(val, index)) → index
  Set-value(i: index, v: val, a: Array(val, index))
                                  → Array(val, index)
  Has-value(i: index, a: Array(val, index)) → Bool
  Inspect-value(i: index, a: Array(val, index)) → val
  Kill(a: Array(val, index)) → ()
```


Gränsyta till Fält (2)

Datatypens namn (Array) och typparametrar som behövs för att definiera den

Visar att vi definierar en datatyp

```
abstract datatype Array(val, index)
  Create(lo, hi: index) → Array(val, index)
  Low(a: Array(val, index)) → index
  High(a: Array(val, index)) → index
  Set-value(i: index, v: val, a: Array(val, index))
                                  → Array(val, index)
  Has-value(i: index, a: Array(val, index)) → Bool
  Inspect-value(i: index, a: Array(val, index)) → val
  Kill(a: Array(val, index)) → ()
```

Datatypens metoder, deras parametrar och vad de returnerar

Fältets typparametrar

- Fält har två typparametrar
  - `val` - elementvärdetypen
    - Typen på elementen som ska lagras i Fältet
    - Kan vara vilken typ som helst
  - `index` - typen för det index som används
    - måste vara *n* st vars värden är diskret linjärt ordnade
    - alla operationer på index hanteras av indextypen (nästa index, förra index, etc.)
    - fältet har indexgränser som sätts när fältet skapas och kan därefter inte ändras, bara avläsas

Informell specifikation av Fält (1)

- `Create(lo, hi)` — skapa ett fält med `lo` och `hi` som gränser för index
  - Notera att storleken på fältet är från `lo` till och med `hi`!
    - `Create(10, 15)` har plats för 6 element
  - Initialt har alla element ett odefinierat värde
- `Low(a)` — returnera den undre indexgränsen för fältet `a`
- `High(a)` — returnera den övre indexgränsen för fältet `a`
- `Kill(a)` — lämnar tillbaka de resurser (minne) som Fältet använt

Informell specifikation av Fält (2)

- `Set-value(i, v, a)` — sätt värdet för elementet med index `i` i fältet `a` till `v`
- `Has-value(i, a)` — `True` omm (om och endast om) värdet i fältet `a` med index `i` är satt med `Set-value`
- `Inspect-value(i, a)` — returnera värdet för elementet med index `i` i fältet `a`

Exempel (1) — Endimensionell vektor

- En endimensionell vektor av längd 10 som indexeras från 0 och kan lagra flyttal (`double`) kan definieras som
  - `Array(double, int)`
- och skapas med
  - `a` ← `Create(0, 9)`
- Sätt första elementet (index 0) till 3.14:
  - `a` ← `Set-value(0, 3.14, a)`

Exempel (2) — Sudokufält

- Ett sudoku-fält kan definieras som
  - `Array(digit, index)`, där
  - `digit` är `(' ', '1', '2', ..., '9')`,
  - `index` är `(int, int)`
- och skapas med
  - `a` ← `Create((1,1), (9,9))`
- Sätt elementet på rad 1, kolumn 3 till tecknet '2':
  - `a` ← `Set-value((1,3), '2', a)`

Gränsytans operationer

- Operationer i gränsytan till `Array` kan delas upp i tre klasser
  - Konstruktorer (skapar eller förändrar objektet)
    - `Create`
    - `Set-value`
  - Inspektorer (avläser innehåll eller struktur)
    - `Low`
    - `High`
    - `Has-value`
    - `Inspect-value`
  - Destruktorer (monterar sönder objektet)
    - `Kill`

Egenskaper för Fält

- Ett fält är en generisk datatyp (polytyp)
  - Elementvärdetypen kan vara olika typer, t.ex. heltal, strängar, poster, etc.
    - Vi kan ha Fält av heltal, Fält av strängar, osv.
  - OBS! Ett givet Fält är fortfarande homogent, dvs. det kan inte innehålla både heltal och strängar
- Ett fält är en statisk datatyp
  - Storleken och indexgränserna bestäms när fältet skapas
    - Är därefter oförändrade under fältets livslängd
  - Elementen har bestämd plats, flyttas inte runt

Typalgoritm (1)

Pseudokod

```
1  Algoritm Array-find-max(v: Array)
2
3
4
5  mx ← Inspect-value(Low(v), v)
6
7  for i from Low(v) + 1 to High(v) do
8    if (mx < Inspect-value(i, v)) then
9      mx ← Inspect-value(i, v)
10
11
12
13 return mx
```

C-kod

```
int find_max(const int_array_1d *a)
{
    int lo = int_array_1d_low(a);
    int hi = int_array_1d_high(a);
    int mx = int_array_1d_inspect_value(a, lo);

    for (int i = lo + 1; i <= hi; i++) {
          if (mx < int_array_1d_inspect_value(a, i)) {
              mx = int_array_1d_inspect_value(a, i);
          }
    }

    return mx;
}
```


Typalgoritm (2)

Pseudokod

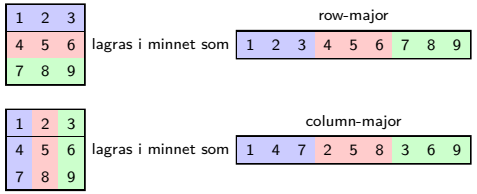
```
1  Algorithm main()
2
3  v ← Create(0, 3)
4  v ← Set-value(0, 4, v)
5  v ← Set-value(1, 6, v)
6  v ← Set-value(2, 2, v)
7  v ← Set-value(3, 8, v)
8
9  mx ← Inspect-value(Low(v), v)
10 for i from Low(v) + 1 to High(v) do
11   if (mx < Inspect-value(i, v)) then
12     mx ← Inspect-value(i, v)
13
14
15 Kill(v)
16 return mx
```

C-kod

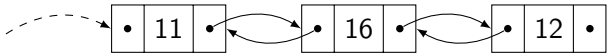
```
int main(void)
{
  int v[4] = { 4, 6, 2, 8 };




  int mx = v[0];
  for (int i = 1; i < 4; i++) {
    if (mx < v[i]) {
      mx = v[i];
    }
  }

  return mx;
}
```

Konstruktion av Fält

- Fysisk datatyp i många språk
- Inbyggda fält i C lagras som vektor
  - (n-dim Fält som 1-dim Fält)
  - "Vecklar" ut fältet
- Matriser lagras kolumnvis (Matlab) eller radvis (C).



- Fält kan konstrueras som Lista
  - Ineffektivt — måste traversera från början av listan för att nå element med index `i`



Tillämpningar Fält

- Spelplaner
  - Speciellt 2-dimensionella, t.ex. schack, Othello, luffarschack
- Tekniska beräkningar
  - Geometriska transformationer (grafik, datorspel, VR, datorseende)
    - Ex: translation T, rotation R, skalning S:

$$T = \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad S = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

  - Linjära ekvationssystem (simulering, flödesberäkningar, autopiloter, GNSS)
  - Bildberäkningar (kantdetektering, brusreducering, OCR)

https://commons.wikimedia.org/w/index.php?curid=44894482

Gles matris

- Gles matris — ett stort antal element är odefinierade eller har samma värde (ofta noll eller tomma strängen)
- Kommer ofta från stora tekniska problem
  - Skulle inte rymmas i minnet om man implementerade dem med normala fält
- Bara intresserade av de signifikanta värdena, t.e.x de nollskilda
- Kan konstrueras som Lista av Tabell, ex (rad, kol, val)
  - Sparar utrymme — tabellstorleken proportionell mot antalet tabellvärden
  - Sparar tid när man vill gå igenom endast de signifikanta elementen i matrisen
  - Andra operationer och algoritmer kan ta längre tid
- Ex. *sparse matrix* i Matlab, excel-dokument

Kodbasen

- Kodbasen innehåller tre olika implementationer av Fält (array)

`array_1d` generisk 1-dimensionellt, med heltal som index `Array(Link(val), int)`

`array_2d` generisk 2-dimensionellt, med heltal som index `Array(Link(val), (int, int))`

`int_array_1d` typat för heltal, 1-dimensionellt, med heltal som index `Array(int, int)`

- Datatypen `Link(val)` är en länk (typ pekare) till `val` och gås igenom på nästa föreläsning
- Kodbasen innehåller flera MWE (*Minimum Working Examples*) för varje datatyp
  - Värdefullt att kompilera och testköra dessa i debugger för att få förståelse för hur datatyperna kan användas
- Kodbasen innehåller ingen implementation av ett generiskt Fält med godtycklig dimension

Nu då?

- Registrera dig på kursen https://www.umu.se/student/
  - Om det inte funkar av nån anledning, kontakta studexp@cs.mu.se så hjälper de dig
- Workshop på fredag och tisdag
  - Övningar i terminal, kompilera från command line
  - Använda Debugger
- Eget arbete
  - Gör övningar i C för att repetera era kunskaper
  - Installera program
    - Fråga på diskussionssidan på Canvas om något strular