Algoritmiskt Tänkande

Testning

Dr. Johan Hagelbäck

johan.hagelback@lnu.se

http://aiguy.org

Linnæus University

Algoritmiskt Tänkande

Programarkitektur

- Tidiga beräkningsmaskiner (datorer?) hade en *fixed-program* arkitektur
- Instruktionerna för programvaran är inbyggda i själva hårdvaran, och endast data lagras i minnet
- Dessa maskiner kunde inte enkelt programmeras om eftersom de då behövde byggas om fysiskt
- En miniräknare (av det enklare slaget) kan också ses som en *fixed-program* arkitektur då den endast kan utföra matematiska beräkningar och inte kan programmeras om

von Neumann-arkitektur

- John von Neumann föddes 1903 i Ungern men var verksam i USA från 1930-talet
- von Neumann räknas ofta som den kanske främste matematikern under 1900-talet
- Han gjorde betydande bidrag inom kvantfysik, nationalekonomi och datavetenskap
- von Neumann anses ofta vara den som uppfann *stored-program* arkitekturen, även om principerna byggde på andras arbete

von Neumann-arkitektur

- Centralt för *stored-program* arkitekturen är att inte bara information kan lagras som data i minnet,
- … utan även instruktioner kan ses som data och lagras i minnet
- I artikeln ”First Draft of a Report on EDVAC” som publicerades 1945 beskrev von Neumann en dator med en aritmetikenhet (CPU), en kontrollenhet innehållande instruktionsregister och programräknare, externt minne och internt minne för att lagra både data och instruktioner

von Neumann-arkitektur

- Det som var nytt med von Neumanns arkitektur jämfört med andra samtida eller tidigare system är att:
  - Ett gemensamt minne används för både data och instruktioner
  - Vad som är data och vad som är instruktion avgörs av sammanhanget
  - Beräkningar sker sekventiellt
- Alan Turing beskrev en konkurrerande arkitektur som tydligt skiljer mellan data och instruktioner, vilket visade sig inte vara lika kraftfullt
- Att se instruktioner som data gav datorer helt andra möjligheter än någonsin tidigare!

von Neumann-flaskhalsen

- Modernare arkitekturer försöker lösa en del problem som von Neumanns arkitektur hade, främst ett problem kallat von Neumann-flaskhalsen
- Data och instruktioner läses sekventiellt från minnet
- von Neumann insåg fördelarna med parallella system, men ansåg att det skulle vara för komplicerat att konstruera ett sådant
- Principen om gemensamt minne är också något som anses vara ett problem, då alla programspråk ändå skiljer mellan data och instruktioner

Harvard-arkitektur

- Moderna datorer bygger på Harvard-arkitekturen
- Den har fysiskt separata bussar för instruktioner och data
- Arkitekturen använder både en programräknare och en datapekare, och de kan adressera helt olika minnesområden
- Detta medför att CPUn kan läsa instruktioner och läsa/skriva data samtidigt
- Den är därför snabbare än en von Neumann-arkitektur

Kombinerad arkitektur

- Moderna datorer har ofta en kombination av Harvard- och von Neumann-arkitekturen
- CPUn är internt av Harvard-typ, med ett cacheminne till databussen och ytterligare ett till instruktionsbussen
- De två cacheminnena arbetar mot ett gemensamt externt minne, vilket gör att kretsen utåt liknar en von Neumann-arkitektur

Kalkylark (spreadsheet)

Kalkylark (spreadsheet)

- Excel, Google Spreadsheet, Numbers etc. är väldigt användbara verktyg vi har svårt att klara oss utan
- Gemensamt är att de lagrar data i tabellform i celler
- Rader är numrerade och kolumner identifieras med bokstäver
- Cell C4 refererar till cellen på rad 4 kolumn 3
- Kalkylark är intressanta eftersom de är bra exempel på hur instruktioner kan ses som data
- Som exempel ska vi beräkna medeltid det tar att sortera en lista med heltal med några vanliga algoritmer för sortering

Exekveringstid

Run	Bubble	Quick	Selection	Insertion	Merge
1	17384	24	3258	3	30
2	17559	21	3386	3	27
3	17795	19	3344	4	28
4	17484	20	3417	3	28
5	17642	19	3358	3	30

- Tid i millisekunder för att sortera 100000 slumptal mellan 0 och 10000

Exekveringstid

C7 =SUMMA(C2:C6)

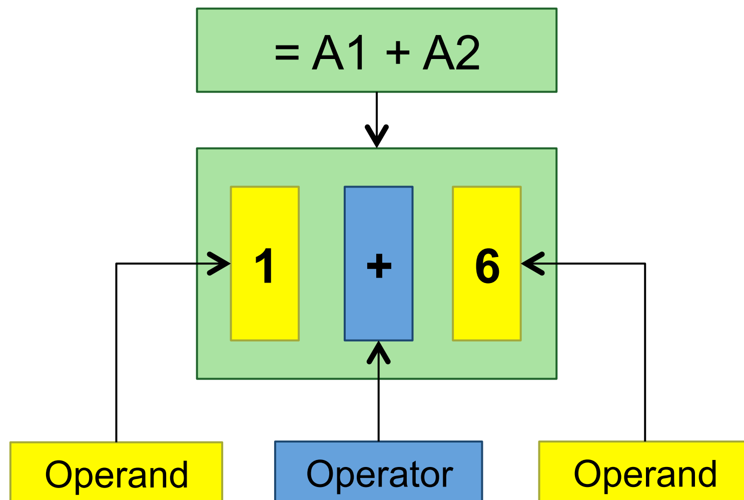
	A	B	C	D	E	F
1	Run	Bubble	Quick	Selection	Insertion	Merge
2	1	17384	24	3258	3	30
3	2	17559	21	3386	3	27
4	3	17795	19	3344	4	28
5	4	17484	20	3417	3	28
6	5	17642	19	3358	3	30
7	Sum	87864	103	16763	16	143
8						
9						

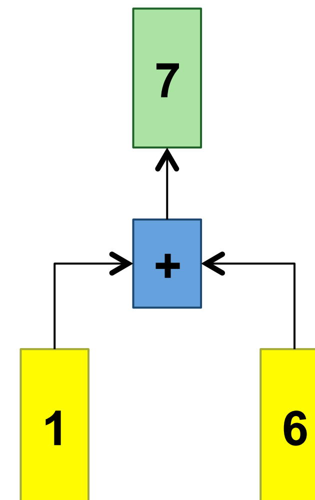
- Cell B7, C7, D7, E7 och F7 innehåller instruktioner (formler)
- Vi ser dock inte själva formlerna utan resultatet av beräkningarna
- Ändrar vi värdet i en cell uppdateras resultaten automatiskt

Formler/uttryck

- Formler beskriver indata och vilka beräkningar som ska göras för att producera utdata
- I programmering kallas det uttryck (expressions)
- Formler måste följa strukturella regler, kallat syntax
- En cell måste börja med = för att visa att vi vill använda en formel och inte bara skriva text
- Formler har följande element:
  - Tal: 3, -18, 3.141
  - Operatorer: + - / *
  - Cell referenser: A1, C4
  - Funktioner: SUMMA(), MIN(), …

Operatorer



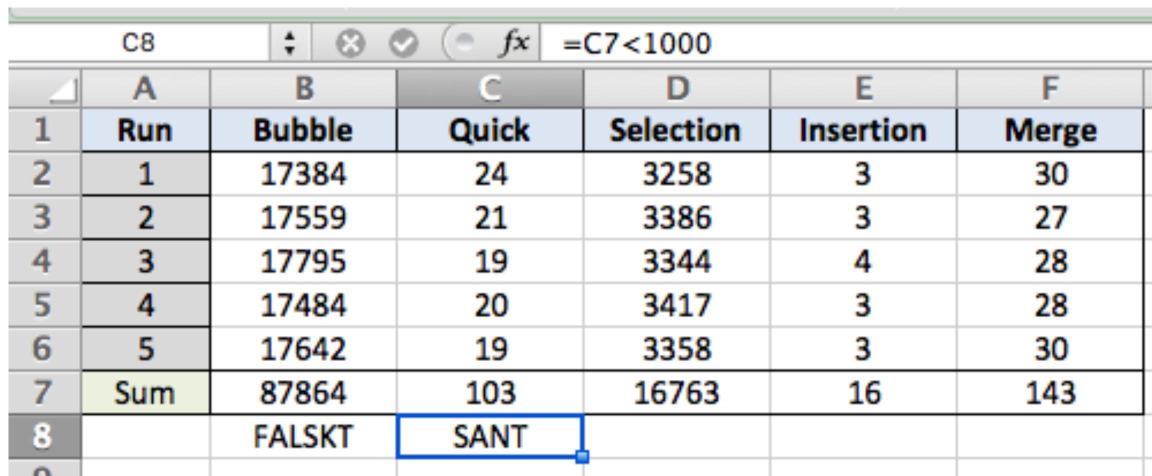


Struktur på formeln

Resultat av beräkningen

Operatorer

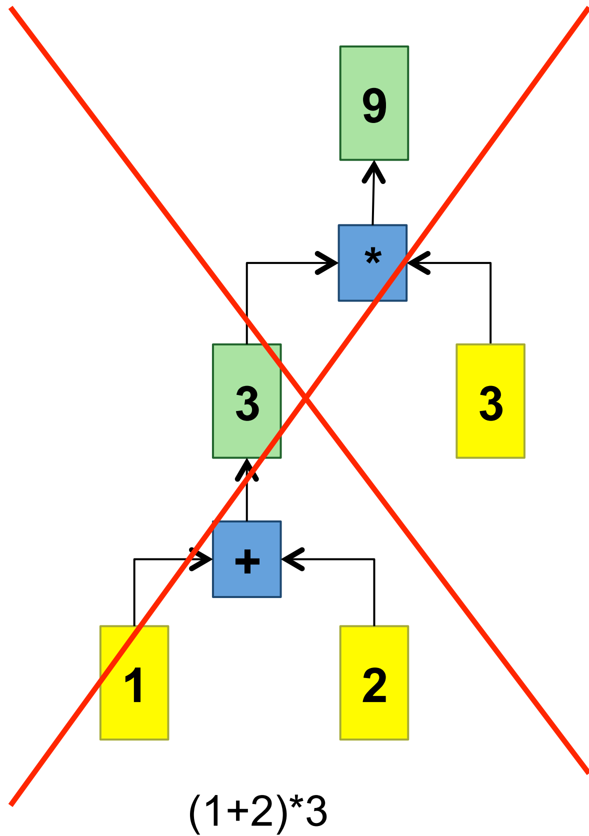
- Formler kan innehålla flera operatorer
- Beräkningarna följer prioriteringsreglerna i matematik
- Förutom aritmetiska operatorer kan även logiska jämförelseoperatorer användas:



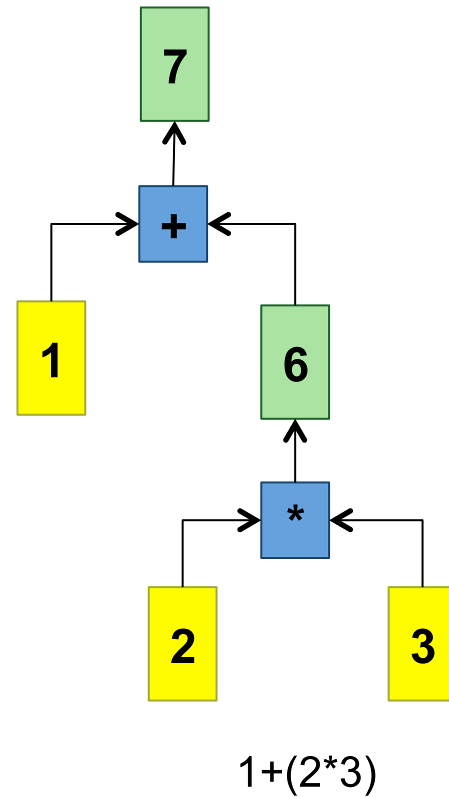
C8 =C7<1000

	A	B	C	D	E	F
1	Run	Bubble	Quick	Selection	Insertion	Merge
2	1	17384	24	3258	3	30
3	2	17559	21	3386	3	27
4	3	17795	19	3344	4	28
5	4	17484	20	3417	3	28
6	5	17642	19	3358	3	30
7	Sum	87864	103	16763	16	143
8		FALSKT	SANT			

Prioriteringsregler



1 + 2 * 3



Prioriteringsregler

- Alla operatorer har följande egenskaper:
  - Aritet (arity): antal argument i indata
  - Räkneordning (precedence): i vilken ordning operatorer prioriteras
  - Associativitet (associativity): anger i vilken ordning operatorer beräknas om de har samma prioritet
- Exempel:
  - Multiplikation tar två argument som indata
  - Den har högre prioritet än addition och subtraktion, och samma prioritet som division
  - Den är vänster-associativ, det vill säga operatorer med samma prioritet sorteras från vänster till höger

Medelvärde

B9 | =B7/B8

	A	B	C	D	E	F
1	Run	Bubble	Quick	Selection	Insertion	Merge
2	1	17384	24	3258	3	30
3	2	17559	21	3386	3	27
4	3	17795	19	3344	4	28
5	4	17484	20	3417	3	28
6	5	17642	19	3358	3	30
7	Sum	87864	103	16763	16	143
8	No runs	5	5	5	5	5
9	Average	17572,8	20,6	3352,6	3,2	28,6
10						

- Cell B9 använder Cell B7 och B8 som indata
- Beräkningen som görs i B9 bryr sig bara om själva resultatet i B7, inte att B7 innehåller en formel
- Formler ses alltså som data!

Flöde

	A	B
1	Run	Bubble
2	1	17384
3	2	17559
4	3	17795
5	4	17484
6	5	17642
7	Sum	87864
8	No runs	5
9	Average	17572,8
10		

Indata

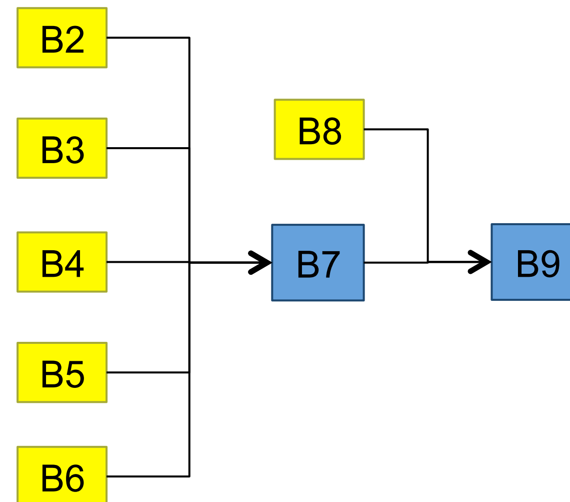
=B2+B3+B4+B5+B6

=ANTAL(B2:B6)

- För att utföra en beräkning måste beroenden analyseras, alltså vilka celler som måste beräknas före vi kan utföra en beräkning
- Internt bygger applikationen en beroende-graf

Beroende-graf

	A	B
1	**Run**	**Bubble**
2	1	17384
3	2	17559
4	3	17795
5	4	17484
6	5	17642
7	Sum	87864
8	No runs	5
9	Average	17572,8
10		



Funktioner

- Funktioner anropas med funktionsnamn (oftast med stora bokstäver) och inparametrar:

funktionsnamn (input1 , input2 , … , inputN)

- En inparameter kan vara:
  - Värde: 1, 52.3
  - Cell refens: B2, C7
  - Område: B2:B6 anger alla celler från B2 till och med B6

Område

	A	B	C	D	E	F
1	Run	Bubble	Quick	Selection	Insertion	Merge
2	1	17384	24	3258	3	30
3	2	17559	21	3386	3	27
4	3	17795	19	3344	4	28
5	4	17484	20	3417	3	28
6	5	17642	19	3358	3	30
7	Sum	87864	103	16763	16	143
8	No runs	5	5	5	5	5
9	Average	17572,8	20,6	3352,6	3,2	28,6

Kolumn (C2:C6)

	A	B	C	D	E	F
1	Run	Bubble	Quick	Selection	Insertion	Merge
2	1	17384	24	3258	3	30
3	2	17559	21	3386	3	27
4	3	17795	19	3344	4	28
5	4	17484	20	3417	3	28
6	5	17642	19	3358	3	30
7	Sum	87864	103	16763	16	143
8	No runs	5	5	5	5	5
9	Average	17572,8	20,6	3352,6	3,2	28,6

Block (D3:E5)

	A	B	C	D	E	F
1	Run	Bubble	Quick	Selection	Insertion	Merge
2	1	17384	24	3258	3	30
3	2	17559	21	3386	3	27
4	3	17795	19	3344	4	28
5	4	17484	20	3417	3	28
6	5	17642	19	3358	3	30
7	Sum	87864	103	16763	16	143
8	No runs	5	5	5	5	5
9	Average	17572,8	20,6	3352,6	3,2	28,6

Rad (B4:F4)

Vanliga funktioner

- De flesta kalkylark har ett antal vanliga funktioner:
  - MAX(B2:B6)
    Visar högsta värdet för cellerna inom området
  - MIN(B2:B6)
    Visar minsta värdet för cellerna inom området
  - SUMMA(B2:B6)
    Beräknar summan av cellerna inom området
  - MEDEL(B2:B6)
    Beräknar medelvärdet av cellerna inom området
  - ABS(B7)
    Beräknar absolutvärdet på en cell
  - …

Textbearbetning

Textbearbetning

- All data vi lagrar i minnet är inte tal
- En stor (största?) del är text:
  - Namn, statusinformation, email adress, titel, …
- Texter kallas strängar (string) i programmeringsvärlden
- En sträng är en ordnad sekvens av tecken:
  - Siffor, bokstäver och symboler (/ & + -% ? …)
  - Specialtecken som radbrytning (osynligt)
- Längden på en sträng är antal tecken i strängen

Strängar

- I programmering anges värdet på en sträng (texten) inom enkla eller dubbla citationstecken:
  - ”En sträng”
  - ’Också en sträng’
- Vissa programmeringsspråk accepterar båda, andra bara det ena:
  - ”en text” anger en sträng i Java
  - ’a’ anger datatypen char (bokstav) i Java
- Vanliga specialtecken är:
  - ”rad1\nrad2” anger radbrytningar
  - ”col1\tcol2” anger tabbar
- Vissa datorsystem använder ”\n\r” för radbrytning
  - Line feed + carriage return

Strängoperationer

- Det finns ett antal vanliga operationer vi kan utföra på strängar
- Dessa finns i samtliga programmeringsspråk men det kan variera hur vi använder dem

- Length:
  - Returnerar längden på en sträng:

```
> String name = "Kalle"
> print(name.length)
5
```

Strängoperationer

- Indexering:
  - En sträng är en sekvens av tecken (en array)
  - Vi kan plocka ut enskilda bokstäver med hakparenteser:

```
> String name = ”Kalle”
> print(name[1])
a
```

  - Observera att längden på strängen är 5, men index är mellan 0 och 4

Strängoperationer

- Konkatenering:
  - Kombinerar två strängar till en ny sträng:

```
> String firstname = ”Kalle”
> String lastname = ”P”
> String fullname = firstname + lastname
> print(fullname)
KalleP
```

Strängoperationer

- Substring:
  - Returnerar en del av en sträng:

```
> String name = ”Kalle”
> String first3 = name.substring(0,3)
> print(first3)
Kal
```

  - Inparametrar är start- och slutindex på delsträngen
  - Delsträngen blir mellan startindex och slutindex - 1

Strängoperationer

- Sökning:
  - Det finns (oftast) flera operationer för att söka i strängar
  - indexOf – returnerar index på första förekomsten av en delsträng i en sträng, eller -1 om delsträngen inte finns
  - lastIndexOf – liknar indexOf men returnerar sista förekomsten av en delsträng
  - contains – returnerar om en delsträng finns i en sträng eller ej (finns inte i alla system då indexOf kan användas i stället)

Strängoperationer

- Sökning:

```
> String name = ”Kalle”
> print(name.indexOf(”l”))
2
> print(name.lastIndexOf(”l”))
3
> print(name.contains(”al”))
true
> print(name.contains(”P”))
false
```

Strängoperationer

- Vi kan också kombinera operationer
- Ett vanligt användningsområde är att plocka ut delar av en sträng, till exempel lokal- och domändel i en email adress:

```
> String em = ”johan.hagelback@lnu.se”
> String local = em.substring(0,em.indexOf(”@”))
> print(local)
johan.hagelback
> String domain = em.substring(em.indexOf(”@”)+1,em.length)
> print(domain)
lnu.se
```

Mönster

Mönster

- Ett mönster (pattern) anger en struktur som en textsträng måste följa
- Det är användbart om vi vill kontrollera att en sträng uppfyller strukturella krav
- Exempel på strängar som måste uppfylla strukturella krav är:

Typ	Exempel
E-mail adress	johan.hagelback@lnu.se
Personnummer	20290504-1213 290504-1213
Datum	2029-05-04
Tid	23:59

Reguljära uttryck

- Ett reguljärt uttryck beskriver ett mönster som en sträng måste följa
- Det kan användas för att:
  - Kontrollera om en sträng är korrekt formaterad för dess syfte
  - Söka efter en viss typ av delsträngar i en text, till exempel hitta alla email adresser i ett dokument

Grundläggande regler

- Reguljära uttryck följer ett antal grundläggande regler:

Grundläggande regler
Ett enskilt tecken är ett mönster (undantaget några specialtecken), tecknet självt
Punkt (.) är ett mönster som representerar ett tecken, oavsett vilket
Om A och B båda är mönster, är också följande mönster: • AB: en sekvens av mönster A följt av B • A|B: antingen mönster A eller mönster B • (A): mönster A ses som en grupp

Grundläggande regler

- Följande mönster beskriver ett personnummer (10 siffror):

```
(0|1|2|3|4|5|6|8|9) (0|1|2|3|4|5|6|8|9) (0|1|2|3|4|5|6|8|9)
(0|1|2|3|4|5|6|8|9) (0|1|2|3|4|5|6|8|9) (0|1|2|3|4|5|6|8|9)
- (0|1|2|3|4|5|6|8|9) (0|1|2|3|4|5|6|8|9) (0|1|2|3|4|5|6|8|9)
(0|1|2|3|4|5|6|8|9)
```

- Uttrycket är rätt svårtolkat…
- Det blir också problem om delar kan ha olika längd, t.ex. lokaldelen på en email adress

Repetitionsregler

- För att komma runt dessa begränsningar kan vi använda repetitionsregler:

Repetitionsregler
A* anger noll eller flera upprepningar av mönstret A
A+ anger en eller flera upprepningar av A
A? anger noll eller en förekomster av A
A{m} anger exakt m repetitioner av A
A{m,n} anger minst m och som mest n repetitioner av A
A{m,} anger minst m repetitioner av A

- Följande tecken är därmed specialtecken och inte mönster:
  * + ? { }

Repetitionsregler

- Vi kan skriva om mönstret för personnummer med hjälp av repetitionsregler:

```
(0|1|2|3|4|5|6|8|9){6} - (0|1|2|3|4|5|6|8|9){4}
```

- Ibland anges ett + i stället för - för personer födda föregående århundrade
- Här måste vi vara noga med att + är ett specialtecken och inte ett mönster, så vi måste skriva \+

```
(0|1|2|3|4|5|6|8|9){6} (-|\+) (0|1|2|3|4|5|6|8|9){4}
```

Teckenklasser

- För att godkänna alla använder vi följande uttryck:

```
(0|1|2|3|4|5|6|8|9)
```

- Detta blir lite svåröverskådligt, särskilt om vi vill godkänna alla stora och små bokstäver:

```
(a|A|b|B|c|C|...
```

Teckenklasser

- Till vår hjälp har vi teckenklasser som omfattar alla tecken av en viss typ:

Teckenklasser
Hakparenteser anger en teckenklass, och vilka tecken som tillhör klassen. [0123456789] anger att alla siffror är tillåtna.
- definierar ett område. [0-9] anger att alla siffror är tillåtna, och [a-z,A-Z] att alla bokstäver, små som stora, är tillåtna
Specialklasser: • \d: en enskild siffra • \D: ett enskilt tecken som inte är en siffra • \w: en enskild bokstav eller siffra [a-z,A-Z,0-9] • \W: ett enskilt tecken som inte är en bokstav eller siffra • \s: ett white-space tecken (mellanslag, tab eller radbrytning) • \S: ett enskilt tecken som inte är white-space

Teckenklasser

- Vi kan snygga till vårt uttryck för personnummer ytterligare med hjälp av teckenklasser:

```
\d{6} - \d{4}
```

- Eller:

```
\d{6} (-|\+) \d{4}
```

Email adress

- Ett enkelt uttryck för en email adress är:

```
\w+ @ \w+ \. [a-z,A-Z]+
```

- Det finns dock flera specialfall som vårt uttryck inte hanterar, och ett komplett uttryck för email adresser är komplicerat:
  - Vissa specialtecken tillåts, till exempel - och _
  - Andra specialtecken tillåts inte, till exempel ? och @
  - Domändelen måste sluta med en godkänd domän, t.ex. .se eller .com
- Ett bättre, men inte helt komplett, uttryck är:

```
[a-z0-9_-]+(\.[a-z0-9_-]+)*@[a-z0-9_-]+(\.[a-z0-9_-]+)+
```

Sökning

- Reguljära uttryck kan användas för att hitta delar i en text som matchar ett uttryck
- Vi kan till exempel leta upp ord som skrivs med enbart STORA BOKSTÄVER
- För att hitta en sekvens av stora bokstäver använder vi följande uttryck:

```
[A-Z]+
```

- Detta kan dock inte användas för sökning eftersom hela texten måste matcha uttrycket
- Vi måste även ange vad som ska vara före och efter mönstret:

```
\W [A-Z]+ \W
```

Sökning

- Om vi vill hitta sekvenser med två eller flera ord med enbart stora bokstäver kan vi modifiera uttrycket något:

```
\W ([A-Z]+\W){2,}
```

- Teckenklassen \W anger alla tecken som inte är en bokstav eller siffra
- Ett alternativ är att i stället använda \s som tillåter white-space (mellanslag, tab eller radbrytning) men inga andra symboler
- Avvägning måste göras mellan att vara för specifik (och kanske missa några fall) eller att vara för förlåtande (och hitta en del fall som inte är korrekta)

Svenska tecken

- Varken [a-z,A-Z] eller \w tillåter svenska tecken
- I stället måste vi använda följande mönster:

```
[a-z,A-Z,åäöÅÄÖ]
```

Testning

Fel i applikationer

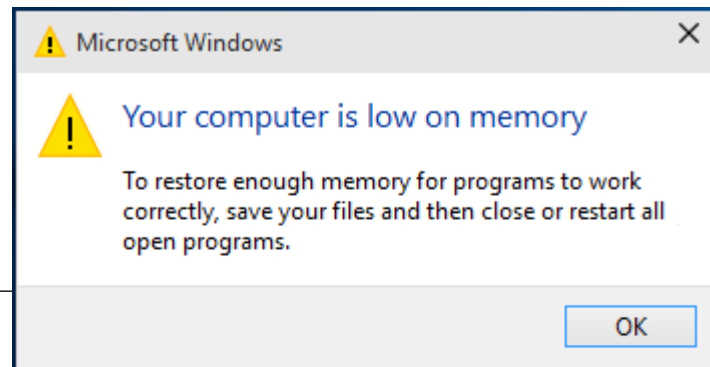
- Ni upptäcker att saldot på ert bankkonto inte stämmer efter att ni satt in pengar på kontot
- Vad är mest troliga källan till felet:
  - En dator har räknat fel
  - En banktjänsteman har skrivit in fel belopp

Fel i applikationer

- Det är oerhört sällsynt att fel beror på hårdvara
- Fel kan bero på buggar i programvaran
- Men mest troligt är att felet beror på den mänskliga faktorn
- Vanliga orsaker till fel kan vara:
  - Systemet ges inkorrekt indata
  - Systemets funktionalitet kan missförstås
  - Systemets output kan missförstås
  - Fysisk skada
  - Fel på hårdvaran
  - Fel i mjukvaran (buggar)
  - Intrångsförsök/hackerattack

Fel i applikationer

- Av dessa är det mest troligt att en användare matar in inkorrekt data
- Därför måste vi vara noga med att validera input data för att hitta de vanligaste felen
- Användare kan även missförstå hur applikationer fungerar:
  - Blanda ihop Reply och Reply All knapparna i epost klienten
- Användare kan också missförstå output eller meddelanden i en applikation



Fel i applikationer

- Fysiska skador menas skador som uppkommer genom mänsklig påverkan:
  - Stötar, fallskador, vatten eller kaffe, …
  - Oftast går datorn sönder om den blir fysiskt skadad
- Hårdvarufel är fel på hårdvaran som inte beror på mänsklig påverkan
  - Dessa är oerhört sällsynta
- Det vi som utvecklare kan påverka mest är mjukvarufel - buggar

Korrekthet

- När vi pratar om korrekthet (correctness) syftar vi oftast på mjukvaran
- Ni har garanterat stött på felaktig mjukvara:
  - Spel som buggar
  - Appar som kraschar
  - Word vill inte lägga din bild där du vill ha din bild
  - …
- Vad innebär egentligen korrekthet?

Korrekthet

- Mjukvaran gör det vi har programmerat den att göra
- Vi kan inte säga att mjukvaran i sig själv är korrekt eller felaktig
- Att vi inte råkar gillar hur något görs i Word betyder det inte att mjukvaran är felaktig
- Korrekthet måste därför ses utifrån en kontext
- Hur är det tänkt att mjukvaran ska fungera?
- Utifrån detta kan vi sedan avgöra om mjukvaran uppför sig korrekt eller felaktigt
- Vem avgör hur en mjukvara ska fungera?

Korrekthet

- Flera aktörer är inblandade i utveckling och användande av en mjukvara:
  – Utvecklare, designers, ingenjörer, …
  – Beställare/kunder
  – Användare
- Användare kan ha önskemål om hur en mjukvara ska fungera, men det är beställaren/kunden som avgör
- Beställare kan vara externa (anlita ett bolag för att utveckla något) eller interna (chefer och marknadsavdelning)
- Som utvecklare kan vi bara påverka hur något ska göras, inte vad det ska göra!

Korrekthet

- Vi har tidigare pratat om kravspecifikation och vikten av att ha detaljerade och väl beskrivna krav
- Det kan dock vara svårt att korrekt beskriva kundens önskemål:
  - Krav är dåligt formulerade
  - Missförstånd mellan utvecklare och beställare
  - Beställare kan ändra sig
  - Krav kan vara svåra att uppfylla
- Vi kan avgöra korrekthet utifrån:
  - Vad beställaren/kunden önskar
  - Vad kravspecifikationen säger

Verifiering och Validering

- Processen för att avgöra en mjukvaras korrekthet kallas Verifiering och Validering
- Validering säkerställer att vi har utvecklat det beställaren/ kunden önskar
- Verifiering säkerställer att mjukvaran uppfyller alla krav i kravspecifikationen
- Barry Boehm beskrev 1979 skillnaden mellan de båda så här:
  - Verification: “Are we building the product right?”
  - Validation: “Are we building the right product?”
- Verifiering är enklare än Validering
- Utvecklarteamet är ansvarigt för verifiering, beställare för validering

Validering

Validering

- En ofta avgörande del i valideringen är att blanda in beställaren i utvecklingsprocessen
- Detta görs ofta i en iterativ process där vi utvecklar mjukvaran i sprintar
- En sprint är oftast 1-2 veckor lång
- I början på en sprint plockas de krav ut som ska implementeras under sprinten
- I slutet av sprinten ska en körbar version av mjukvaran kunna visas för beställaren
- Beställaren får då se mjukvaran under hela utvecklingsprocessen

Användare?

- Ofta är det beställaren som bestämmer hur en applikation ska fungera
- En bra idé är att även fråga framtida användare vad de önskar av applikationen, och hur de tycker den ska fungera
- Flera stora IT projekt har varit mer eller mindre misslyckade för att användarna inte blivit inblandade i tillräckligt hög grad
  - System för sjukvård, polis, …

Validering

- Vi kan också använda olika typer av tester för att validera mjukvaran:
  - Beta testing: beställare och/eller användare testar och ger feedback på en nästan färdig version av mjukvaran
  - Usability testing: beställare och/eller användare testar gränssnittet och användarvänligheten
  - Acceptance test: beställare gör en sluttestning så att mjukvaran fungerar enligt önskemål

Verifiering

Verifiering

- Tanken med en kravspecifikation är att den ska vara en fysisk representation av beställarens/kundens önskemål
- En kravspecifikation bestämmer vad som ska utvecklas
- Som vi diskuterat tidigare ska krav vara:
  - Tydligt beskrivna, konsekventa och kompletta
- Nu lägger vi till ytterligare ett krav på kraven:
  - Krav ska vara verifierbara!

Verifierbar

- Med verifierbar menas att det ska gå att testa om kravet är uppfyllt eller ej
- Är följande krav möjliga att verifiera?
  - R1: mjukvaran måste vara säker
  - R2: alla användare ska kunna logga in med användarnamn och lösenord
  - R3: alla lösenord ska krypteras innan de sparas i databasen
  - R4: mjukvaran ska vara användarvänlig

Testning

- Det vanligaste sättet att verifiera om ett krav är uppfyllt eller ej är genom testning
- Testning innebär kortfattat att vi kör vår applikation och observerar hur den fungerar
- Observationerna jämförs sedan med kraven
- Ett lyckat test är när applikationen fungerar enligt de krav testet ska verifiera
- Ett misslyckat test är motsatsen – applikationen beter sig annorlunda än vad kraven säger

Testning

- Det är viktigt att ha i åtanke att testning aldrig helt kan garantera korrekthet!
- Testning har sina brister:
  - Det är svårt och kostsamt att testa allt i en mjukvara (coverage)
  - Testning görs ofta i en annan miljö än där applikationen sedan körs (verklig vs. simulering)
  - Det är ofta svårt att utforma tester för vissa saker, t.ex. säkerhetsintrång

Exempel på dyra fel

- Mars Climate Orbiter:
  - 1998 förlorade NASA kontrollen över Mars Climate Orbiter
  - Felet berodde på att utvecklarna glömt att konvertera från engelska enheter till metriska
  - Rymdsonden kostade 125 miljoner dollar att bygga
- Ariane 5:
  - Raketen Ariane 5 hade kraftigare och snabbare motorer än föregångaren Ariane 4
  - Ingenjörerna använde samma mjukvara men de snabbare motorerna resulterade i ett mjukvarufel, ett 64-bitars tal komprimerades till 16 bitar och mätfel uppstod
  - Raketen exploderade och lasten, en sond värd 500 miljoner dollar, gick förlorade

Exempel på dyra fel

- Heathrow terminal 5:
  - Bagagesystemet på Heathrows nya terminal 5 testades med 12000 testväskor
  - Det testades dock inte under verkliga scenarion, som att resenärer ibland plockar upp sina väskor igen för att de glömt att ta ut något
  - Detta gjorde att systemet kraschade och 42000 väskor kom bort
  - Över 500 flighter ställdes in under de 10 dagar det tog att få igång systemet igen

https://raygun.com/blog/10-costly-software-errors-history/

Exempel på testfall

- R2: alla användare ska kunna logga in med användarnamn och lösenord
- Testfall:

ID:	T2.1
Input:	User Name: admin Password: admin
Förväntat resultat:	Användaren loggas in i systemet

Exempel på testfall

- Vad är det för ”problem” med vårt testfall?
- Det testar att systemet fungerar när giltig login anges, men inte när ogiltig anges
- Vi behöver därför komplettera med ytterligare ett testfall:

ID:	T2.2
Input:	User Name: admin Password: kalle
Förväntat resultat:	Meddelande om fel användarnamn eller lösenord visas

Exempel på testfall

- Har vi full coverage på login-kravet nu?
- Det finns fler saker som vi eventuellt behöver testa:
  1. Vad händer om ett giltigt användarnamn men tomt lösenord anges?
  2. Vad händer om ”admin” byts ut mot ”Admin”?
     Är systemet case-sensitive?
  3. Fler än ett giltigt användarnamn och lösenord?
  4. Alla möjliga kombinationer av användarnamn och lösenord?
- Någonstans måste gränsen dras när det är för tidskrävande (och dyrt) att testa
- Punkt 1-3 är rimliga att testa, men inte 4

Välja testfall

- Vi kan inte testa allt!
- Testfallen bör vara representativa för användandet av systemet
- Övervägning måste göras mellan:
  - Tid/kostnad att implementera testet
  - Sannolikheten att det som testas inträffar
- Om vi testar ett par olika kombinationer av otillåtna användarnamn kan vi vara relativt säkra på att andra otillåtna användarnamn inte heller godkänns
- Vi skapar ett antal tester som samlas i en testsvit
- Testsviten testar hela systemet, men kan inte helt garantera korrekthet

Iterativ utveckling

- I iterativ utveckling implementerar vi ett antal krav i en sprint om 1-2 veckor
- Vi ska då också se till att vi har tester som täcker kraven som implementerats

- Att skriva bra tester kan vara lika svårt som att implementera funktionalitet!
- Utvecklare tränas i att implementera mjukvara, inte lika mycket i att hitta situationer när mjukvaran inte fungerar
- Oftast har företag dedikerade testare som bara jobbar med testning, och utvecklare som bara jobbar med utveckling

Typer av testning

- Testning delas ofta in i två typer:
- Black box testning:
  - Testare har inte tillgång till programkoden
  - Applikationen körs och verifieras utan att testarna vet hur den fungerar internt
  - Kallas ofta *funktionell testning*
- White box testning:
  - Testare har tillgång till programkoden och kan se hur mjukvaran fungerar internt

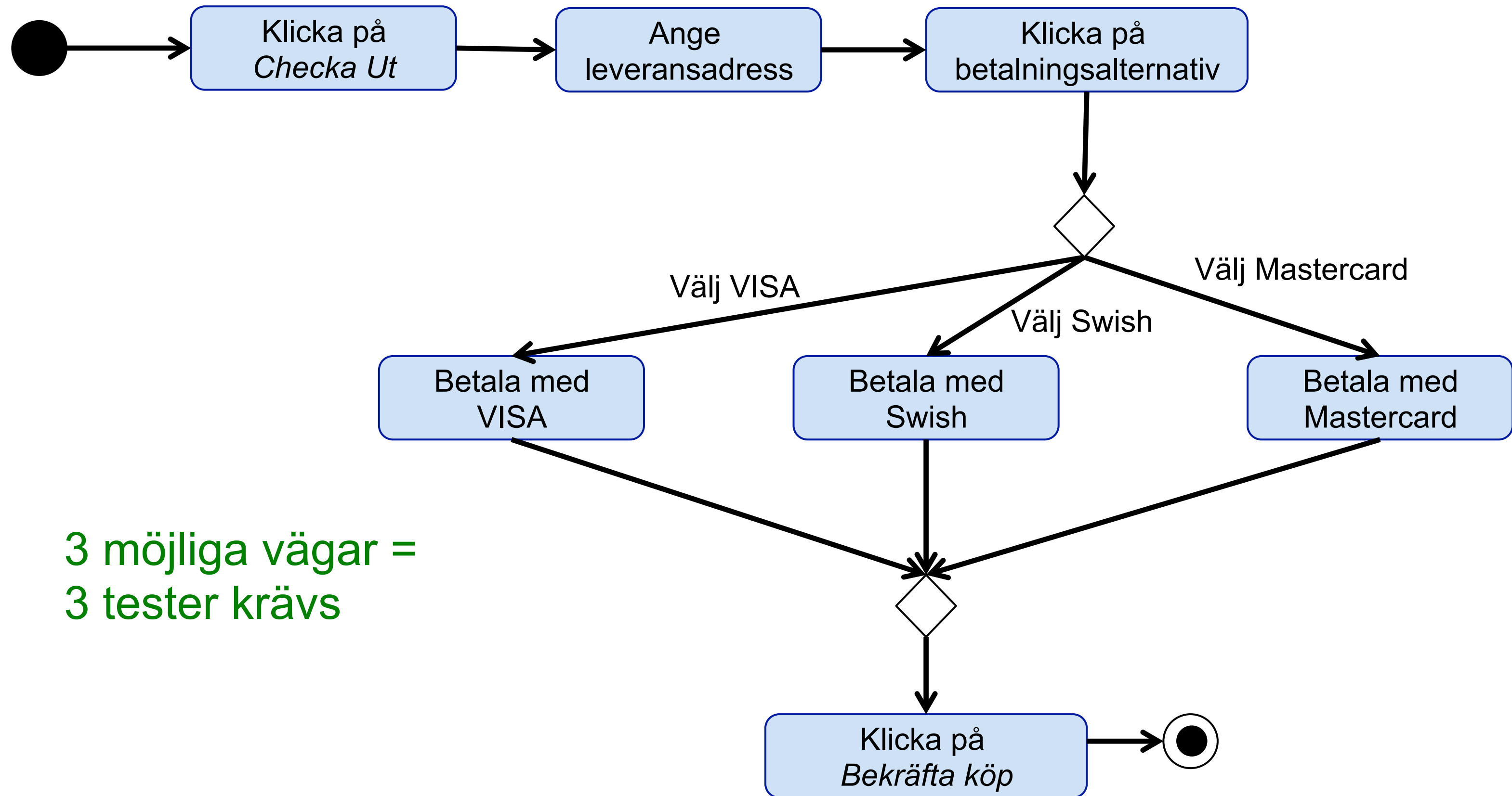
White box testning

- För att hitta bra testfall kan vi undersöka strukturen i programkoden (*structure testing*)
- Det hjälper oss att hitta test som täcker alla kod (*statement coverage*)
- Målet är att alla kodrader ska testas i åtminstone ett testfall
- Viktigt att ha i åtanke att bara för att ett stycke kod fungerar, kan det misslyckas under andra omständigheter (t.ex. annan indata)

White box testning

- Vi kan också undersöka kontrollflödet i programkoden för att hitta bra testfall (*path testing*)
- Till vår hjälp kan vi ha aktivitetsdiagram – de är oftast lättare att följa än flödet i programkoden
- Komplett path testing innebär att varje väg (path) från start till slut ska täckas av åtminstone ett testfall

Path testing



Black box testning

- Black box testning skiljer sig inte nämnvärt från white box testning
- Enda skillnaden är att vi inte kan granska koden för att hitta testfall som täcker all kod
- I stället baseras all testning på kraven
- En fördel med black box testning är att det krävs ingen eller lite programmeringserfarenhet

Black box testning

- En teknik vi kan använda vid black box testning är *equivalence partitioning*
- Det innebär att vi delar upp alla möjliga värden på indata i grupper så att varje grupp har något gemensamt med avseende på kraven
- Till exempel kan man ha olika roller i ett system
- Alla login med administratörsrättigheter kan vara i en grupp, och alla med användarrättigheter i en annan grupp, och alla obehöriga i en tredje grupp

Gränsvärden

- ”Bugs lurk in corners and congregate at boundaries” – Boris Beizer
- Vi kan till exempel ha en persondatabas där längd på personer anges som indata
- Normalfallet kan anses vara en längd mellan 155 och 185 cm
- Extremfall är mindre än 150 och över 200 cm
- Vårt system kan t.ex. acceptera längder mellan 140 och 230 cm

Gränsvärden

- Vi bör då testa:
  - Ett testvärde inom normalfallet
  - Ett testvärde inom nedre extremfallet
  - Ett testvärde inom övre extremfallet
  - Ett värde under nedre gränsen
  - Ett värde över övre gränsen
- Genom att testa gränsvärden ser vi om systemet fungerar korrekt med extreminput
- Detta kan vara lämpligt för t.ex. bilar. Hur beter sig bilen om vi kör nära maxhastighet, eller på extremt högt eller lågt varvtal?

Manuell och automatisk testning

- Testning kan utföras både manuellt eller automatiskt
- I manuell testning körs applikationen, och en testare utför ett antal steg och ser om resultatet matchar det förväntade resultatet
  - Oftast black box testning
- I automatisk testning körs programkod som testar annan kod i systemet
  - White box testning
- Oftast används båda två då det har olika för- och nackdelar