F05 - Algoritmer, debugging

5DV149 Datastrukturer och algoritmer
Kapitel 2.1-2.2

Niclas Börlin
niclas.borlin@cs.umu.se
Anna Jonsson
aj@cs.umu.se

2020-02-03 Mon

Algoritmer

- Vad är det?
- Beräkningsbarhet.

Algoritm som recept

- Går att likna vid ett "recept" som man följer för att lösa ett givet problem på ett strukturerat sätt.
  - Ingredienser (indata).
  - Verktyg.
  - Procedur (algoritm).
  - Resultat.
- En algoritm är en **ändlig** stegvis beskrivning av en **ändlig** process.

Cookbook:Seitan

Cookbook | Ingredients | Recipes

Seitan is a food made from gluten protein that has been extracted from wheat flour. It has a springy texture, with protein strands similar to those in chicken running through it, making it a reasonable replacement for meat in vegan or vegetarian dishes.

Chunks of seitan

Ingredients

- 10 oz (280g) High-gluten wheat flour (such as bread flour)
- Water

Procedure

Filtering half mask with exhalation valve (class: FFP3)

1. Add 2 cups water to 10 oz (280g) high-gluten wheat flour (such as bread flour).
2. Knead together until **well-combined** and elastic. It is advisable to keep hands wet so that the gluten doesn't stick to the hands.
3. Cover with water and place in refrigerator for 1 hour.
4. Knead under water until water is cloudy; dump cloudy water and replace with clear, cold water. In the beginning the dough can still easily dissolve underwater so it is advisable to only cautiously squeeze it against the bowl. Later you might find it easier to knead over the water, squeezing out the water, and only to dip the mass into the water to wash off the starch, if the mass contains too much water you easily wash out gluten as well. The enriched water can be used to gain the insoluble starch, which settles on the bottom of the container, and what remains can serve as a base for grain milk.
5. Continue kneading and replacing water until water remains clear after kneading.
6. Divide gluten mass into loaf- or roll-shaped halves. At this point, there are several things you can add to the mass.
   1. Adding soy sauce is recommended; this is traditional in the making of seitan.
   2. Spices. For example, the seitan can simulate Italian sausage by adding the correct spices, or adding poultry seasoning can make the seitan more similar to chicken or turkey.
   3. Adding nutritional yeast is recommended, particularly if those consuming the seitan are vegetarian, and especially vegan. The B12 found in some brands of nutritional yeast is a vitamin that is usually obtained through meats, eggs and/or dairy products.
7. Place halves in pot, cover with vegetable broth or *dashi*.
8. Bring to a boil.
9. Reduce heat to simmer; simmer for 1 - 1 1/2 hours.
10. Remove from heat and serve as desired or use in place of meat.

Algoritm

- **Texten** som beskriver algoritmen har en **fix** storlek.
- **Processen** kan **variera** i storlek.
- En algoritm kan ha olika kornighet (detaljnivå):
  - "Servera med vitlökssås" kontra beskriva hur vitlökssås tillverkas.
  - "Sortera listan av medlemmar" kontra beskriva hur sorteringen ska gå till.

Algoritm — Definition

En algoritm är en noggrann plan, en metod för att stegvis utföra något.

Krav på algoritmer

Ändlighet Algoritmen måste sluta (terminera).

Bestämdhet Varje steg måste vara entydigt.

Indata Måste ha noll eller flera indata.

Utdata Måste ha ett eller flera utdata.

Effektivitet/Genomförbarhet Varje steg i algoritmen måste gå att utföra på ändlig tid.

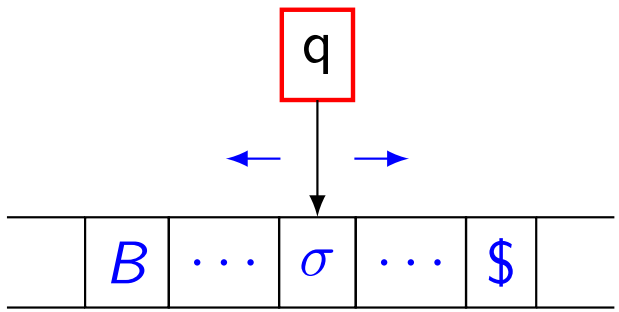
Algoritmer och beräkningsbarhet

- Uttrycket *beräkningsbar* har visat sig vara svårt att definiera exakt.
- En av de mest kända definitioner är (Alan Turing, 1936):

> En algoritm är beräkningsbar om och endast om det finns en *Turingmaskin* som löser problemet.

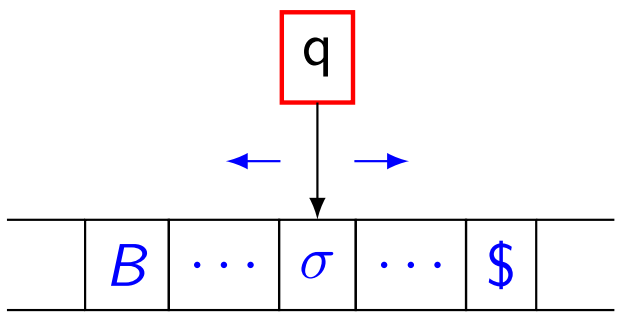
Turing-maskin (1)

- En Turing-maskin är en abstrakt, simpel modell av en beräkningsmaskin och består av:
  1. En centralenhet som kan befinna sig i ett ändligt antal olika tillstånd (*states*).
  2. Ett oändligt långt band indelat i celler som kan innehålla symboler. Symbolerna kommer från ett ändligt alfabet.
  3. Ett läs/skriv-huvud.
  4. En drivanordning för bandet.



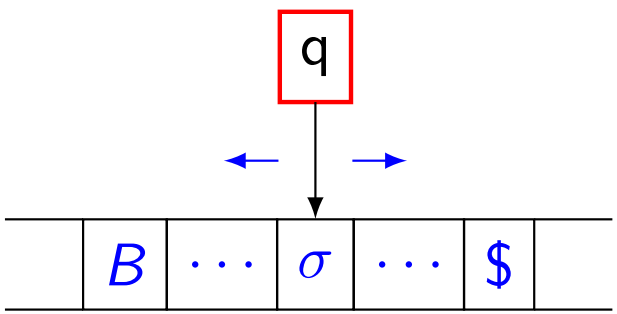
Turing-maskin (2)

- En Turing-masking kan utföra 4 operationer:
  1. Avläsa symbolen i cellen som är mitt för läs-/ skriv-huvudet (den *aktuella* cellen).
  2. (Radera och) skriva en ny symbol i den aktuella cellen.
  3. Flytta bandet en cell-längd framåt eller bakåt.
  4. Stanna maskinen.
- Turing-maskinen jobbar i diskreta beräkningssteg.
- I varje steg utförs en operation.
- Vilken operation Turing-maskinen utför bestäms av en övergångsfunktion (*transition function* eller *action table*).



Turing-maskin (3)

- Maskinen startar med ett *starttillstånd* och en given position på remsan.
- Maskinen avslutar beräkningen när ett *sluttillstånd* uppnås.
- Beräkningen kan misslyckas genom att maskinen aldrig uppnår sluttillståndet.
- Trots sin enkelhet så kan **varje dator-algoritm översättas till en Turing-maskin**.



Turing-maskin (4), exempel

- https://sv.wikipedia.org/wiki/Turingmaskin
- Tillstånd: s1, s2, s3, s4, s5, s6.
- Alfabet: 0, 1.
- Tomma symbolen: 0 (får finnas oändligt många gånger på remsan).
- Indatasymboler: 1.
- Starttillstånd: s1.
- Sluttillstånd: s6.

S	R	W	N	M	
s1	0	0	s6	N	Ingen (mer) etta att kopiera. Klar!
s1	1	0	s2	R	Kopiera denna etta till näst-nästa nolla högerut, lämna en nolla so
s2	1	1	s2	R	Leta vidare högerut efter första nollan.
s2	0	0	s3	R	Första nollan hittad. Gå vidare till s3, som hittar andra.
s3	1	1	s3	R	Leta vidare högerut efter andra nollan.
s3	0	1	s4	L	Andra nollan hittad, skriv en etta och gå tillbaka två nollor.
s4	1	1	s4	L	Leta vidare vänsterut efter första nollan på tillbakavägen.
s4	0	0	s5	L	Första nollan hittad. Gå vidare till s5 som hittar andra.
s5	1	1	s5	L	Leta vidare vänsterut efter andra nollan (den som s1 lämnade som
s5	0	1	s1	R	Nollan hittad. Skriv tillbaka den etta som s1 skrev över, och börja

Turing-maskin (5), exempel

Steg	Tillstånd	Remsa
1	s1	**1**1000
2	s2	0**1**000
3	s2	01**0**00
4	s3	010**0**0
5	s4	01**0**10
6	s5	0**1**010
7	s5	**0**1010
8	s1	1**1**010
9	s2	10**0**10
10	s3	100**1**0
11	s3	1001**0**
12	s4	100**1**1
13	s4	10**0**11
14	s5	1**0**011
15	s1	11**0**11
16	s6	-stopp-

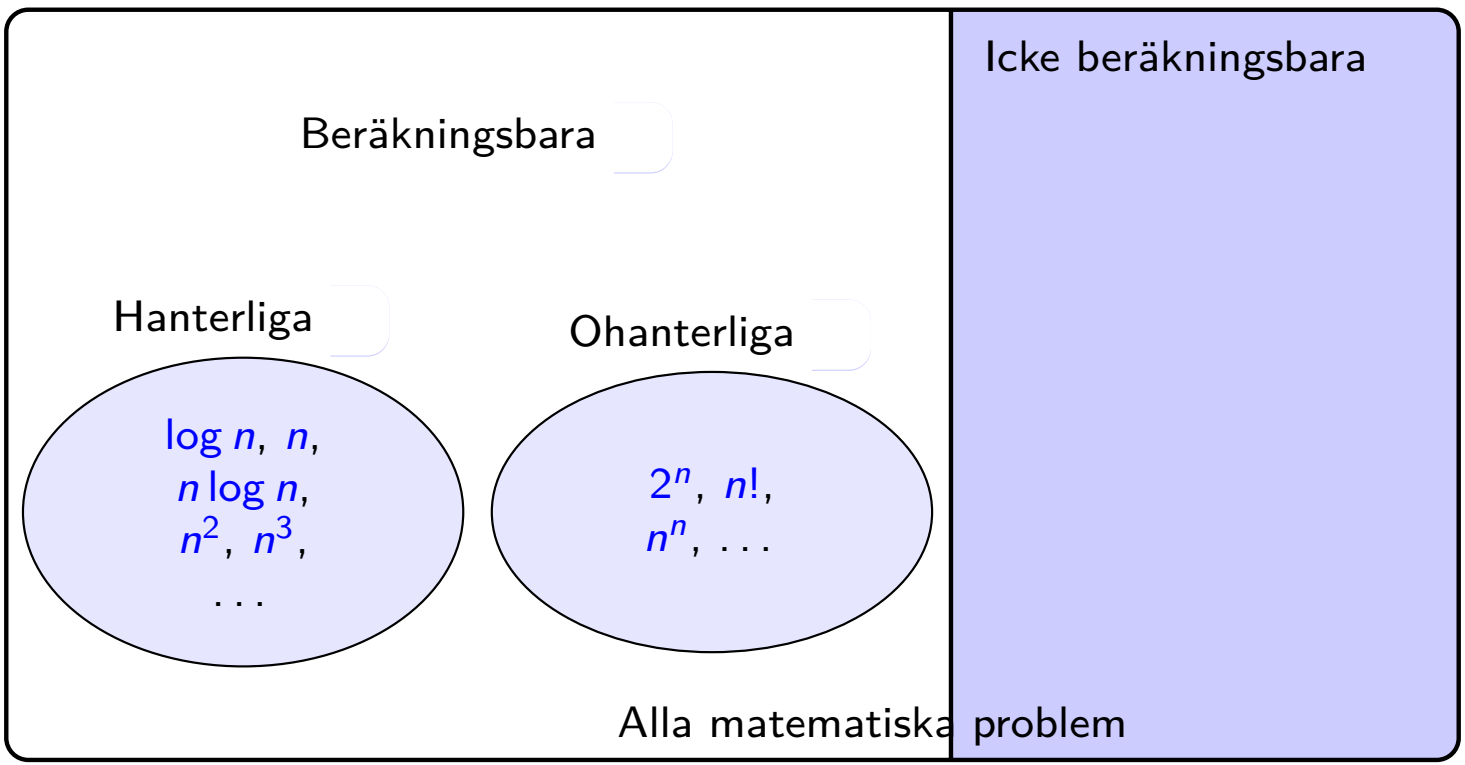
Turing-maskin, icke beräkningsbara problem

- Turing sökte bl.a. svaret på följande fråga: *Does a machine exist that can determine whether any arbitrary machine on its tape is "circular"(e.g., freezes, or fails to continue its computational task)*
- Svaret var: nej.
  - "Ta källkoden till ett godtyckligt C-program och avgör om det kan hamna i en oändlig loop."

Vad kan vi beräkna?

Beräkningsbara/hanterbara problem

- Icke beräkningsbara problem.
- Beräkningsbara, ohanterliga problem — superpolynom.
- Beräkningsbara, hanterliga problem — polynom.



Ohanterbarhet

- Många triviala att förstå och viktiga att lösa:
  - Schemaläggning.
  - Handelsresande.

Hur hanterar vi ohanterbarhet?

- Heuristik!
- Lösa nästan rätt problem exakt:
  - Förenkling.
- Lösa exakt problem nästan rätt:
  - Approximation.
- Exempel: Hitta snabbaste vägen från A till B.
  - Förenkling: Sök A–motorväg–B.
  - Approximation: Dra “rakt streck” närmaste vägen A–B på kartan. Justera strecket så att det går på vägar.

NP-kompletta problem

- En speciell klass av ohanterliga problem.
- Exempel:
  - Givet en mängd $\{M\}$ av heltal, finns det en icke-tom delmängd var summa är noll?
  - Generaliserad Sudoku ($n^2 \times n^2$ matris av $n \times n$-block).
- Ekvivalenta:
  - Transformeras till varandra.
  - Bevis för högst exponentiell komplexitet.
  - Saknar bevis för ohanterbarhet.
- Ett bevis att NP-kompletta problem är NP eller P (super-polynomiska eller polynomiska) är ett stort olöst problem inom datavetenskap och matematik.
- Ett av sju s.k. *Millennium Prize Problems*.

Felsökning

Debugging, Olika typer av fel i program

Syntaxfel Upptäcks av kompilator/interpretator.

Programkörningsfel (*run-time error*) Koden är syntaktiskt korrekt men programmet kraschar när man kör det.

Logiska fel Koden är syntaktiskt korrekt och programmet kör som det ska men resultatet blir inte det man tänkt sig.

Hur hittar man orsakerna till fel?

- Läsa meddelanden från kompilator/interpretator, koden och kommentarerna en extra gång.
- Läsa **första** felmeddelandet igen.
- Lägga till spårutskrifter i programmet.
- **Debugger**

Debugger

- En debugger gör det möjligt att
  - **Stega igenom** koden rad för rad (*single-step*).
  - Följa variablers **värden** och hur de **förändras** under körning (*watch*).
  - **Förändra** variablers värden under körning.
  - Sätta **brytpunkter** (*breakpoints*) som bestämmer var exekveringen ska stanna.
- Under stegningen kan man välja mellan att
  - **gå in** i subrutiner/procedurer (*step in*),
  - att **gå förbi** subrutinerna (*step over*),
  - **hoppa ut** ur en subrutin (*step out*).

Robusthet och effektivitet

- Vi vill ofta att program ska vara både robusta och effektiva.
  - Med robusthet menar vi att användaren kan begå misstag utan att det leder till katastrof.
    - Felaktig inmatning, fil som saknas, osv.
- Hårda robusthetskrav kan leda till försämrad effektivitet.
  - Om man bygger in felhantering i en datatyp så får man en robust lösning, men mindre effektiv.
    - T.ex. att alltid kolla indexgränser i en array.

Syfte med OU3

- OU3 handlar om att ni ska redovisa muntligt att ni kan hantera
  - en debugger (valfri grafisk) och
  - ett verktyg att hitta minnesläckor (`valgrind`).