

A dark blue slide with white text and a logo. The text is centered and reads: "DV2: ALGORITMER OCH PROBLEMLÖSNING" in a large, bold, sans-serif font. Below it, in a smaller, regular sans-serif font, is "Heap och Trie". At the bottom center is a small circular logo for Umeå University, featuring a stylized building and the text "UMU" and "1827". Below the logo, the text "UMEÅ UNIVERSITET" is written in a regular sans-serif font.

1

---

---

---

---

---


---

2

[illegible]

# MODUL 3 - TRÄDALGORITMER

- Lektion
  - Heap och Trie - Denna lektion
- Quiz 2
  - Träd och Binära sökträd (repetition från DV1-kursen)
  - Trädalgoritmer – Heap och Trie
- OU2
  - Huffman
  - Start på måndag



UMEÅ UNIVERSITET

3

---

---

---

---

---

---

## FRÅN DV1-KURSEN

- Träd
  - Ändlig, homogen, acyklisk, rekursiv datatyp
  - Begrepp: Förälder, barn, noder, löv, rot, grenar, ordnade, oordnade, riktade, höjd och djup
  - Begrepp binära träd: fulla och kompletta träd
  - Konstruktioner: Som länkade strukturer eller arrayer
- Trädalgoritmer
  - Traversering – Bredden-först och Djupet-först (Pre-, post- och inorder)
  - Binära sökträd
- Egen repetition utifrån behov



UMEÅ UNIVERSITET

4

---

---

---

---

---

---

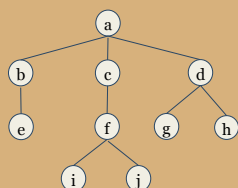
---

---

## UPPGIFT 2

Hur skulle trädet i figuren nedan traverseras om man använde sig av

- a) preorder,
- b) inorder och
- c) postorder?



5

---

---

---

---

---

---

---

---

## DAGENS FÖRELÄSNING

- Heap
- Trie
  - Inklusive allt som behövs för att lösa OU2 (förutom funktionspekare)



UMEÅ UNIVERSITET

6

---

---

---

---

---

---

---

---

## HEAP

- Heap/Hög – ett partiellt sorterat binärt träd
  - Etiketterna är sorterade efter en relation  $R$  så att  $a$  är förälder till  $b$  endast om  $a$  är före  $b$  i ordningen som ges av  $R$
  - Insättningar och borttagningar görs så att trädet hålls komplett
  - Insert  $O(\log n)$ , Delete-first  $O(\log n)$



7

---

---

---

---

---

---

---

## TILLÄMPNINGAR

- Utnyttjas som konstruktion av en prioritetskö
- Enkelt sätt att sortera något (Heapsort)
  - Stoppa in allt i en heap och plocka ut det igen



8

---

---

---

---

---

---

---

## HEAP – INSÄTTNING

- Stoppa in elementet på första lediga plats som håller trädet komplett
- Om elementet är på fel plats enligt sorteringsordning, byt plats med föräldern
  - Upprepa detta tills elementet hamnat rätt!
- Exempel, heap som lagrar tecken
  - Stoppa in bokstäverna P,R,I,O,R,I,T,E,T



9

---

---

---

---

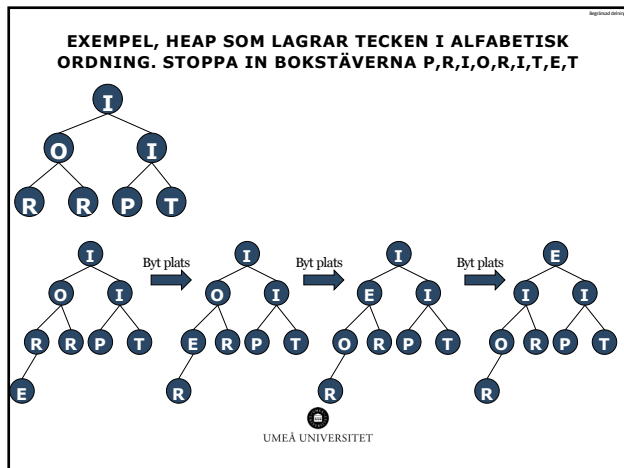
---

---

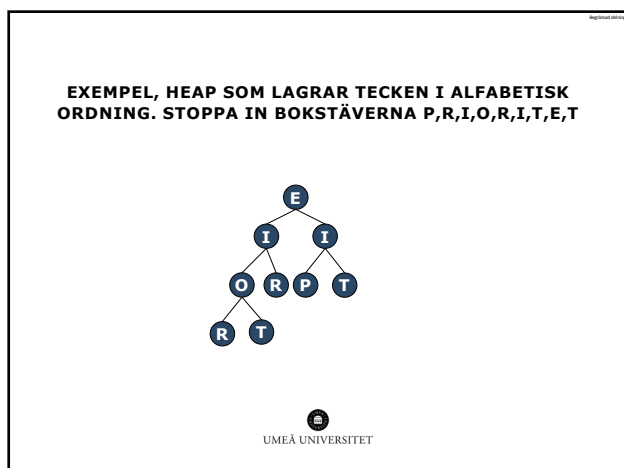
---



10



11



12

## HEAP – BORTTAGNING

- Ta bort roten och stoppa elementet som är "sist" (längst ned till höger) som ny rot
- Om elementet är på fel plats enligt sorteringsordning, byt plats med det minsta av barnen
  - Upprepa detta tills elementet hamnat rätt!



13

---

---

---

---

---

---

---

### Exempel, heap borttagning



14

---

---

---

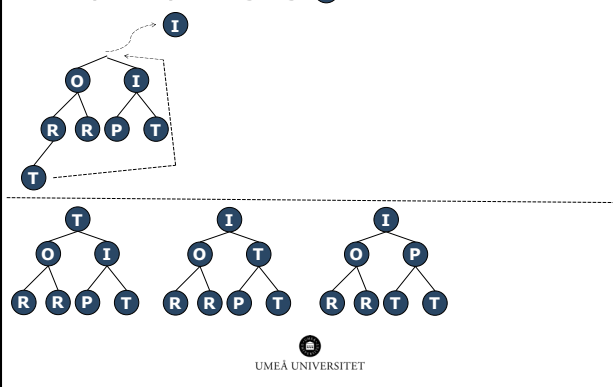
---

---

---

---

### Exempel, heap borttagning



15

---

---

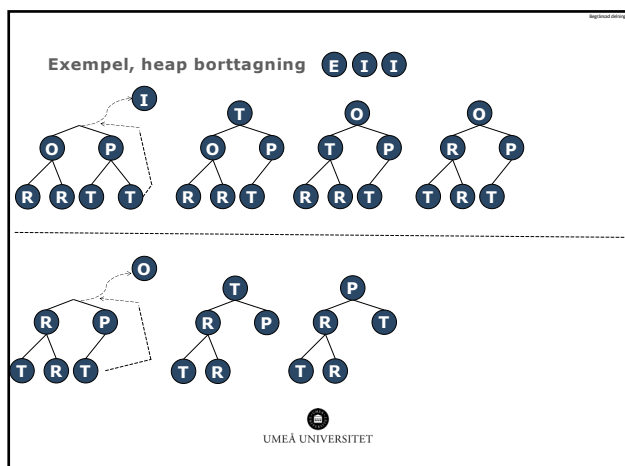
---

---

---

---

---



16

---

---

---

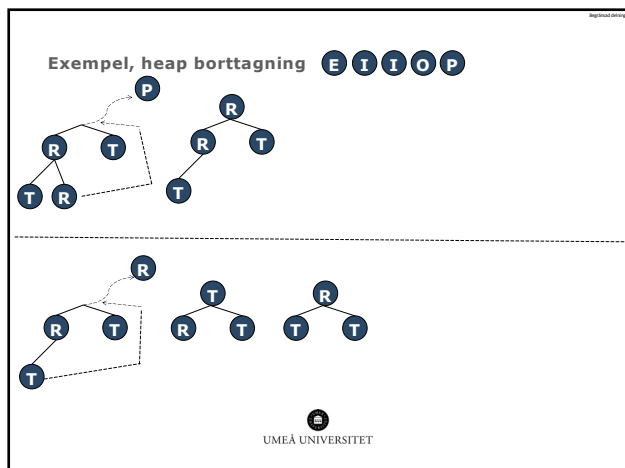
---

---

---

---

---



17

---

---

---

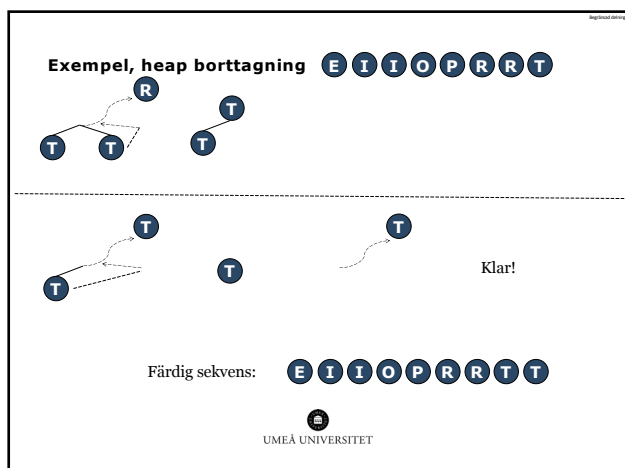
---

---

---

---

---



18

---

---

---

---

---

---

---

---

## UPPGIFT 6-8

### Uppgift 6 - insättning i en heap

Sätt in värdena 1 6 2 5 2 9 -2 4 i den ordning de står i en heap.  
Visa det resulterande trädet efter varje insättning.

### Uppgift 7 - insättning i en heap

Sätt in följande element (i given ordning) i en heap: 5, 3, 8, 2, 4, 1, 7, 9, 6. Rita upp hur hela trädet ser ut efter varje insättning.

### Uppgift 8 – borttagning ur en heap

Utgå från heapen i uppgift 6 och ta bort roten tre gånger. Rita upp hur hela trädet ser ut efter varje borttagning.

19

## TRIE

- Ytterligare en variant av träd. Vi har tidigare sett:
  - Oordnat träd där barnen till en nod bildar en mängd
  - Ordnat träd där barnen till en nod bildar en lista
- I Trie är barnen till en nod organiserade som tabellvärden i en tabell som hör till noden
- Trie kallas också för diskrimineringssträd, code-link tree eller radix-search tree



20

## ORGANISATION AV TRIE

- Man når barnen (delträden) direkt genom "namn", dvs argument (nycklar) i nodens tabell
  - När man ritat träd brukar nycklarna skrivas direkt intill motsvarande båg
- I en trie har tabellerna en och samma nyckeltyp, till exempel tecken
- I många tillämpningar av Trie saknar de inre noderna etiketter, träden är lövträd
- Trie är normalt nedåtriktad
- Binära träd kan ses som ett specialfall av Trie där nyckelvärdena är left och right



21

## INFORMELL SPECIFIKATION

- Två sätt:
  - Alternativ 1: Utgå från Urträdets specifikation och låt typparametern sibling ha värdet Tabell
    - Då hanteras insättning, borttagning och värdeskoll av Tabellen själv
    - I övrigt används de vanliga operationerna för att sätta in och ta bort barn etc.
  - Alternativ 2: Sätt in lämpliga tabelloperationer direkt i specifikationen av Trie
    - Insert-child blir tabellens Insert, Delete-child tabellens Remove och Child tabellens Lookup



UMEÅ UNIVERSITET

22

## KONSTRUKTION AV TRIE

- De flesta konstruktioner av träd går bra
  - Förutsatt att det går bra att byta ut de delar som hanterar barnen (till exempel som element i en lista) till att hantera dessa som tabellvärden i en tabell
  - Implementerar man tabellen som en vektor eller som en hashtabell får man effektiva Trie-implementationer



UMEÅ UNIVERSITET

23

## TILLÄMPNINGAR AV TRIE

- Används för att konstruera Lexikon av sekvenser eller Tabeller där nycklarna är sekvenser
- För sekvenser med element av typ A väljer vi en Trie med tabellnycklar av typ A
  - En sekvens motsvaras då av en väg i trädet från roten till ett löv
  - Man lägger till en slutmarkör i slutet av varje sekvens om en sekvens kan vara början på en annan sekvens
    - En annan variant är att ha etiketter i de inre noderna också
- Ett viktigt/vanligt specialfall är Lexikon/Tabell av textsträng = en lista eller vektor av tecken



UMEÅ UNIVERSITET

24



## FORTS...

- Fördelar med att använda Trie för Lexikon/Tabeller som lagrar sekvenser som startar med samma följd av elementvärden:
  - Kompakt sätt att lagra lexikonet/tabellen på
  - Sökningens tidskomplexitet proportionell mot sekvenslängden. (En jämförelse per elementtecken)
  - Den relativa komplexiteten är oberoende av lexikonets/tabellens storlek
    - Det blir inte "dyrare" att söka i ett stort lexikon jämfört med ett litet



25

---

---

---

---

---

---

---

## TILLÄMPNINGAR

- Stavningskontroll
  - Skapa ett trie med alla ord som finns i språket
- Översättningstabell
  - Löven innehåller motsvarande ord i ett annat språk
- Filsystem på Unix/PC
- Datakomprimering
  - LZ78 algoritmen – zip, gzip, png bland annat
  - Huffman kodning



26

---

---

---

---

---

---

---

## FILKOMPRIMERING

- ASCII-filer är textfiler där varje bokstav representeras av en 8-bitars ascii-kod
  - Det är alltså en fixlängdskodning
- Om man tittar på en textfil ser man att vissa bokstäver förekommer oftare än andra
  - E är vanligast i engelska – jmf Morse-alfabetet (s286 i kursboken)
- Om man lagrar vanligt förekommande tecken med färre bitar så skulle vi kunna spara utrymme



27

---

---

---

---

---

---

---

### FILKOMPRIMERING

- Kodningen måste ske så att man enkelt kan avkoda strängen entydigt med kännedom om hur de olika tecknen översätts
  - Exempel: Antag att de tre tecknen a, b och c kodas som 0, 1 respektive 01
    - Om en mottagare får strängen 001 vad betyder det? aab eller ac?
- Prefix-regeln: Ingen symbol kodas med en sträng som utgör prefix till en annan symbols kodsträng



UMEA UNIVERSITET

28

---

---

---

---

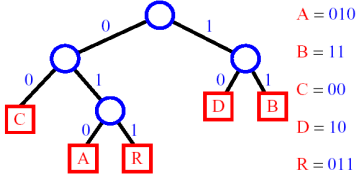
---

---

---

### VI ANVÄNDER EN TRIE!

- Bokstäverna lagras i löven
- Den vänstra kanten betyder 0
- Den högra betyder 1
- Vad betyder 01011011010000101001011011010?



A = 010

B = 11

C = 00

D = 10

R = 011

29

---

---

---

---

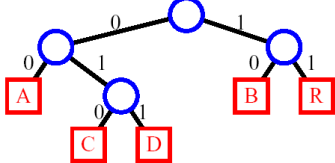
---

---

---

### VI VILL HA OPTIMAL KOMPRESSION!

- Trädet/tabellen bestämmer hur effektiv komprimeringen är
- Vårt träd gav 29 bitar för strängen "ABRACADABRA"
  - Den kan kortas ned till 24 bitar (23 minst):
- 001011000100001100101100 med träd



Notera att A och R fått kortare kod, samt C och D längre.

30

---

---

---

---

---

---

---

## HUFFMANKODNING

- Bygger upp ett optimalt träd från en frekvenstabell
- Börja med en serie träd bestående av ett enda löv
- Till varje löv associeras en symbol och en vikt = symbolens frekvens i texten som ska kodas
- Upprepa tills vi har ett enda stort träd:
  - Välj de två träd som har minst vikt i roten
  - Bygg ihop dem till ett träd där de blir barn till en ny nod
  - Den nya noden innehåller en vikt = summan av barnens vikter
- Den genererade kodtabellen måste skickas först i meddelandet
- Används bla för jpeg, mp3



31

---

---

---

---

---

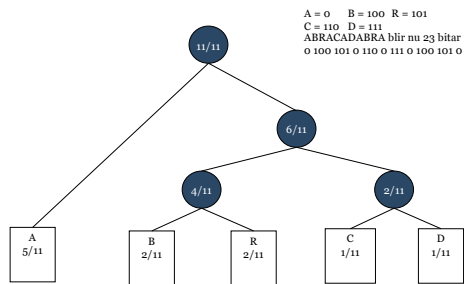
---

---

---

## EXEMPEL HUFFMANKODNING

- ABRACADABRA



32

---

---

---

---

---

---

---

---

## UPPGIFT 9

- Huffman-koda följande sträng: ddddoaaäärrrrllääättt (räkna fram en optimal binärkod för texten samt den trie som behövs för kodning/avkodning)

33

---

---

---

---

---

---

---

---

## TRIES FÖR STRÄNGAR

- **Insättning**
  - Startar i roten och går nedåt i trädet så länge det finns en matchande väg
  - När man hittar en skiljelinje, stanna och stoppa in resten av strängen som ett delträd
- **Borttagning**
  - I princip samma algoritm som insättning fast "tvärtom". Sök upp strängen som ska tas bort och radera nerifrån i trädet upp till första förgreningen



34

---

---

---

---

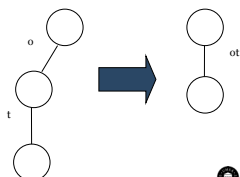
---

---

---

## KOMPRIMERADE TRIES

- Alla barnsnoder konverteras till att innehålla hela strängen/sekvensen som är under
- Vid insättning:
  - Kan behövas delas upp i två barn



35

---

---

---

---

---

---

---

## ANVÄNDNING

- Lempel-Ziv-Welch (LZW) i GIF-formatet, unix-kommandot compress
- DEFLATE-kompression (LZ77+Huffman) i gzip och PNG-formatet



36

---

---

---

---

---

---

---

## LEMPER-ZIV KODNING

- Kodning:
  - Låt frasen o vara strängen ""
  - Skanna igenom texten
    - Om du stöter på en "ny" bokstav lägg till den på toppnivån på trien
      - Stoppa in paret (nodeIndex, aktuellBokstav) i den komprimerade strängen
    - Om du stöter på en "gammal" bokstav gå nedåt i trien tills du inte kan matcha fler tecken, lägg till en nod i trien som representerar den nya strängen
      - Stoppa in paret (nodeIndex, sistaBokstaven) i den komprimerade strängen
- Exempel:
 

"how now brown cow in town."



UMEA UNIVERSITET

37

Startsträng: how\_now\_brown\_cow\_in\_town.

Fraser: 0

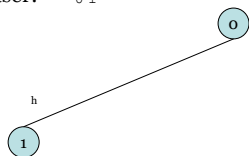


Kodad sträng:

38

Startsträng: how\_now\_brown\_cow\_in\_town.

Fraser: 0 1



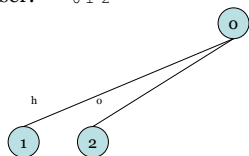
Kodad sträng:

0h

39

Startsträng: how\_now\_brown\_cow\_in\_town.

Fraser: 0 1 2



Kodad sträng:  
0h0o

40

---

---

---

---

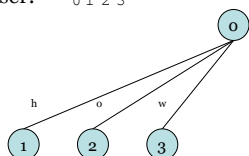
---

---

---

Startsträng: how\_now\_brown\_cow\_in\_town.

Fraser: 0 1 2 3



Kodad sträng:  
0h0o0w

41

---

---

---

---

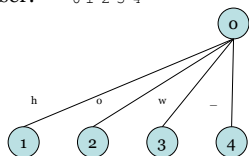
---

---

---

Startsträng: how\_now\_brown\_cow\_in\_town.

Fraser: 0 1 2 3 4



Kodad sträng:  
0h0o0w0\_

42

---

---

---

---

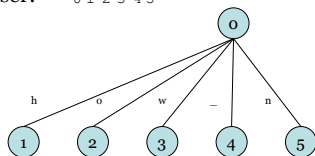
---

---

---

Startsträng: how now brown cow in town.

Fraser: 0 1 2 3 4 5



Kodad sträng:

0h0o0w0\_0n

43

---

---

---

---

---

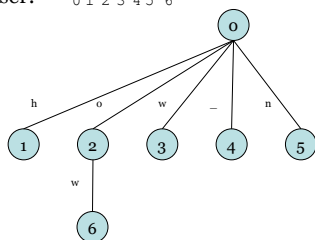
---

---

---

Startsträng: how now brown cow in town.

Fraser: 0 1 2 3 4 5 6



Kodad sträng:

0h0o0w0\_0n2w

44

---

---

---

---

---

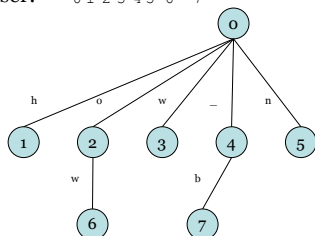
---

---

---

Startsträng: how now brown cow in town.

Fraser: 0 1 2 3 4 5 6 7



Kodad sträng:

0h0o0w0\_0n2w4b

45

---

---

---

---

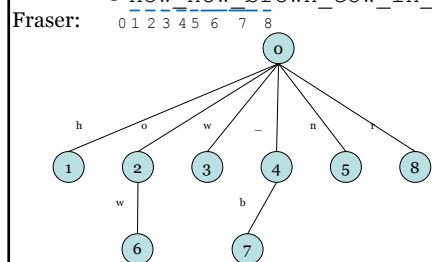
---

---

---

---

Startsträng: how now brown cow in town.



Kodad sträng:

0h0o0w0\_0n2w4b0r

46

---

---

---

---

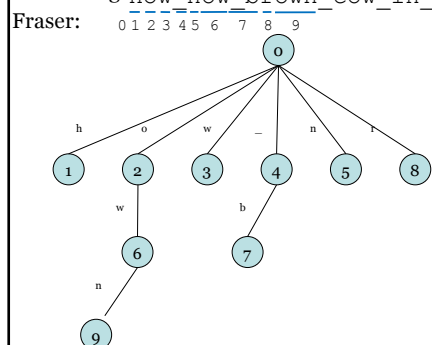
---

---

---

---

Startsträng: how now brown cow in town.



Kodad sträng:

0h0o0w0\_0n2w4b0r6n

47

---

---

---

---

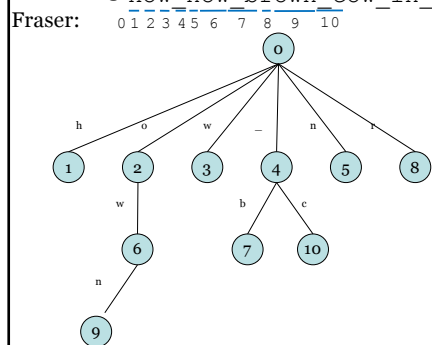
---

---

---

---

Startsträng: how now brown cow in town.



Kodad sträng:

0h0o0w0\_0n2w4b0r6n4c

48

---

---

---

---

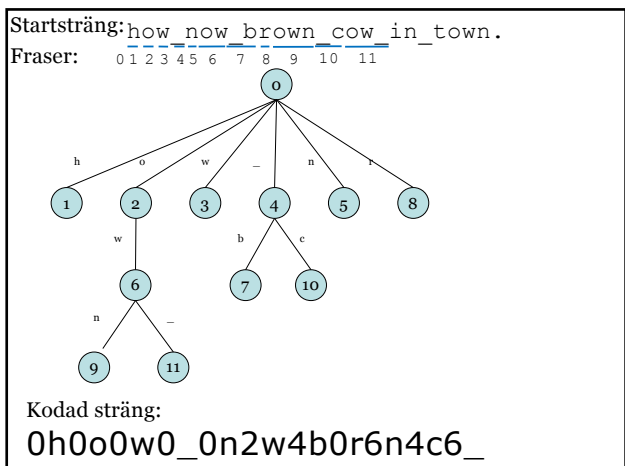
---

---

---

---





49

---

---

---

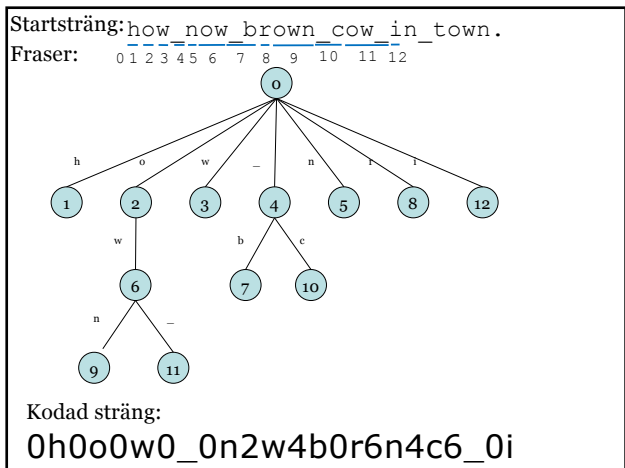
---

---

---

---

---



50

---

---

---

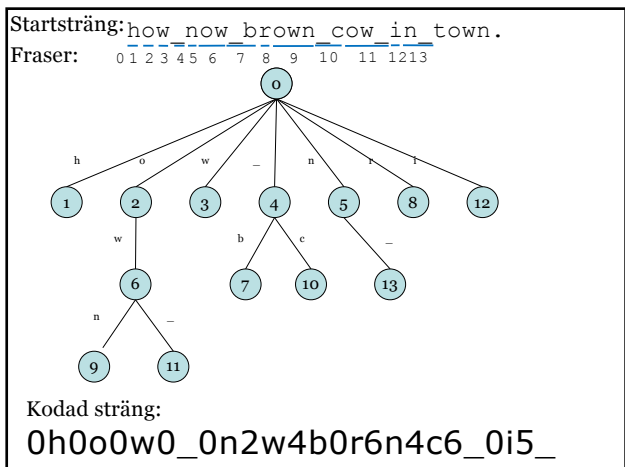
---

---

---

---

---



51

---

---

---

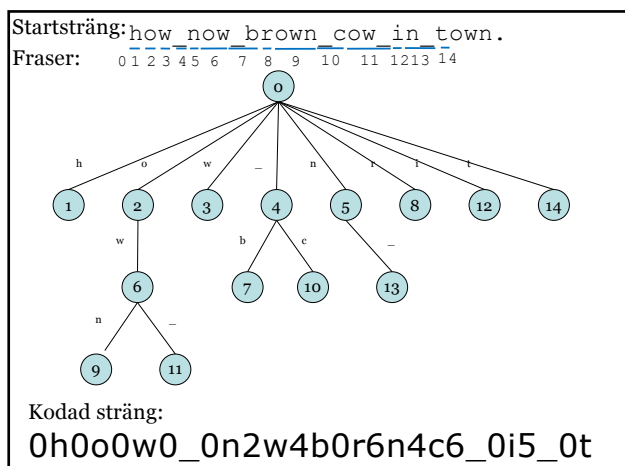
---

---

---

---

---



52

---

---

---

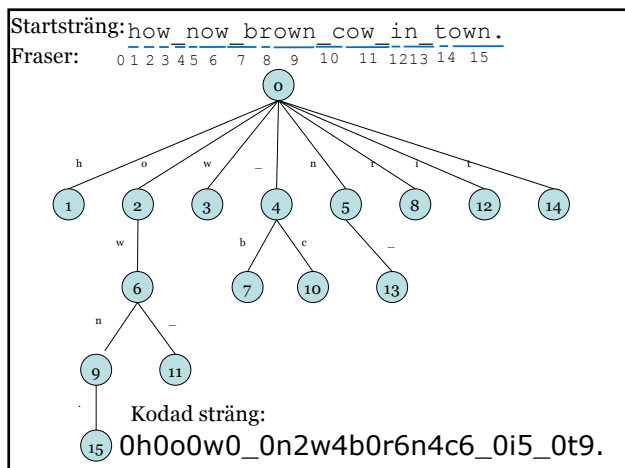
---

---

---

---

---



53

---

---

---

---

---

---

---

---

### LEMPER-ZIV KODNING

- Avkodning:
  - Varje gång du stöter på "o" i strängen lägg nästa bokstav i strängen direkt efter den föregående i den avkodade strängen
  - För varje index skiljt från o, stoppa in delsträngen som motsvaras av den noden i den avkodade strängen, följt av nästa tecken i den komprimerade strängen
  - Notera att man inte behöver skicka med trädet
- Exempel: 0h0o0w0\_0n2w4b0r6n4c6\_0i5\_0t9.

h	o	w		b	r	o	w		c	o	w		i	n		t	o	w	.
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15					

UMEÅ UNIVERSITET

54

---

---

---

---

---

---

---

---

## SAMMANFATTNING

- Heap
- Trie
  - Inklusive allt som behövs för att lösa OU2 (förutom funktionspekare)



UMEÅ UNIVERSITET

55

## NÄSTA GÅNG

- OU2 - Introduktion och nästa steg
  - Läs igenom specifikationen noggrant innan
  - Specifikationen publiceras (senast) 14/2 13:15
  - Genomförs enskilt eller i par
    - Bilda par senast på måndag den 17/2



UMEÅ UNIVERSITET

56