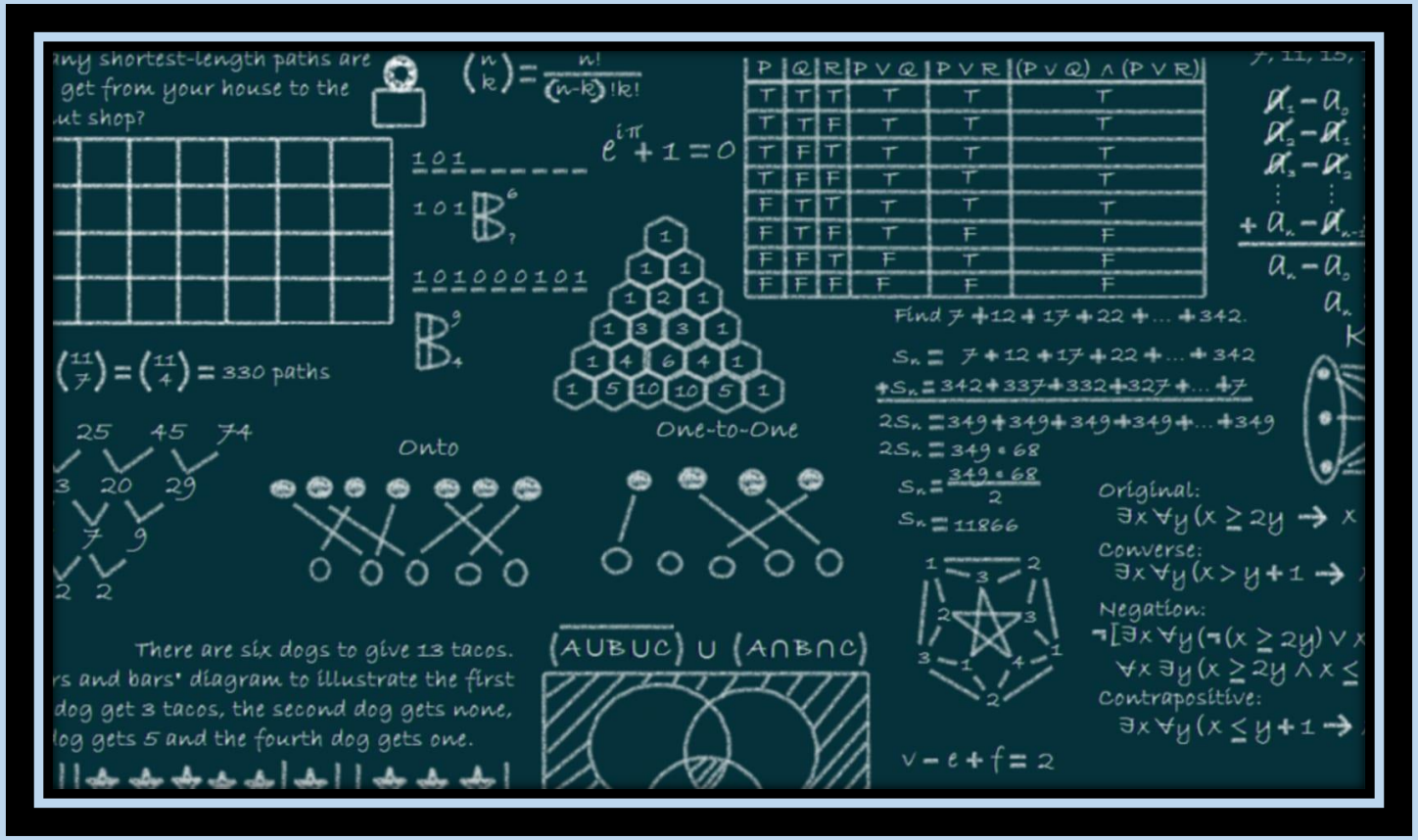


FÖRELÄSNING 4

Datastrukturer och algoritmer
KYH – 2022 HT

Andreas Nilsson Ström

Repetition

Uppvärmning!

Algoritm-Problem: Multipler av 3 och 5

- Om vi listar alla naturliga tal under 10 som är multipler av 3 och 5 så får vi 3, 5, 6 och 9. Summan av dessa multipler tal är 23.
- Hitta summan av alla multipler av 3 och 5 under 1000

Algoritm-Problem: Multipler av 3 och 5

- Om vi listar alla naturliga tal under 10 som är multipler av 3 och 5 så får vi 3, 5, 6 och 9. Summan av dessa multipler tal är 23.
- Hitta summan av alla multipler av 3 och 5 under 1000

- Detaljer
  - Naturligt tal = Heltal som inte är negativt. Dvs större eller lika med 0.
  - Om ett tal är en multipel av 3 eller 5 så ska vi lägga till det till vår summa
  - Testa alla tal under 1000

Algoritm-Problem: Multipler av 3 och 5

- Om vi listar alla naturliga tal under 10 som är multipler av 3 och 5 så får vi 3, 5, 6 och 9. Summan av dessa multipler tal är 23.
- Hitta summan av alla multipler av 3 och 5 under 1000

- Detaljer
  - Naturligt tal = Heltal som inte är negativt. Dvs större eller lika med 0.
  - Om ett tal är en multipel av 3 eller 5 så ska vi lägga till det till vår summa
  - Testa alla tal under 1000

- Svar: 233168

Algoritm-Problem: Multipler av 3 och 5

- Om vi listar alla naturliga tal under 10 som är multipler av 3 och 5 så får vi 3, 5, 6 och 9. Summan av dessa multipler tal är 23.
- Hitta summan av alla multipler av 3 och 5 under 1000

- Detaljer
  - Naturligt tal = Heltal som inte är negativt. Dvs större eller lika med 0.
  - Om ett tal är en multipel av 3 eller 5 så ska vi lägga till det till vår summa
  - Testa alla tal under 1000

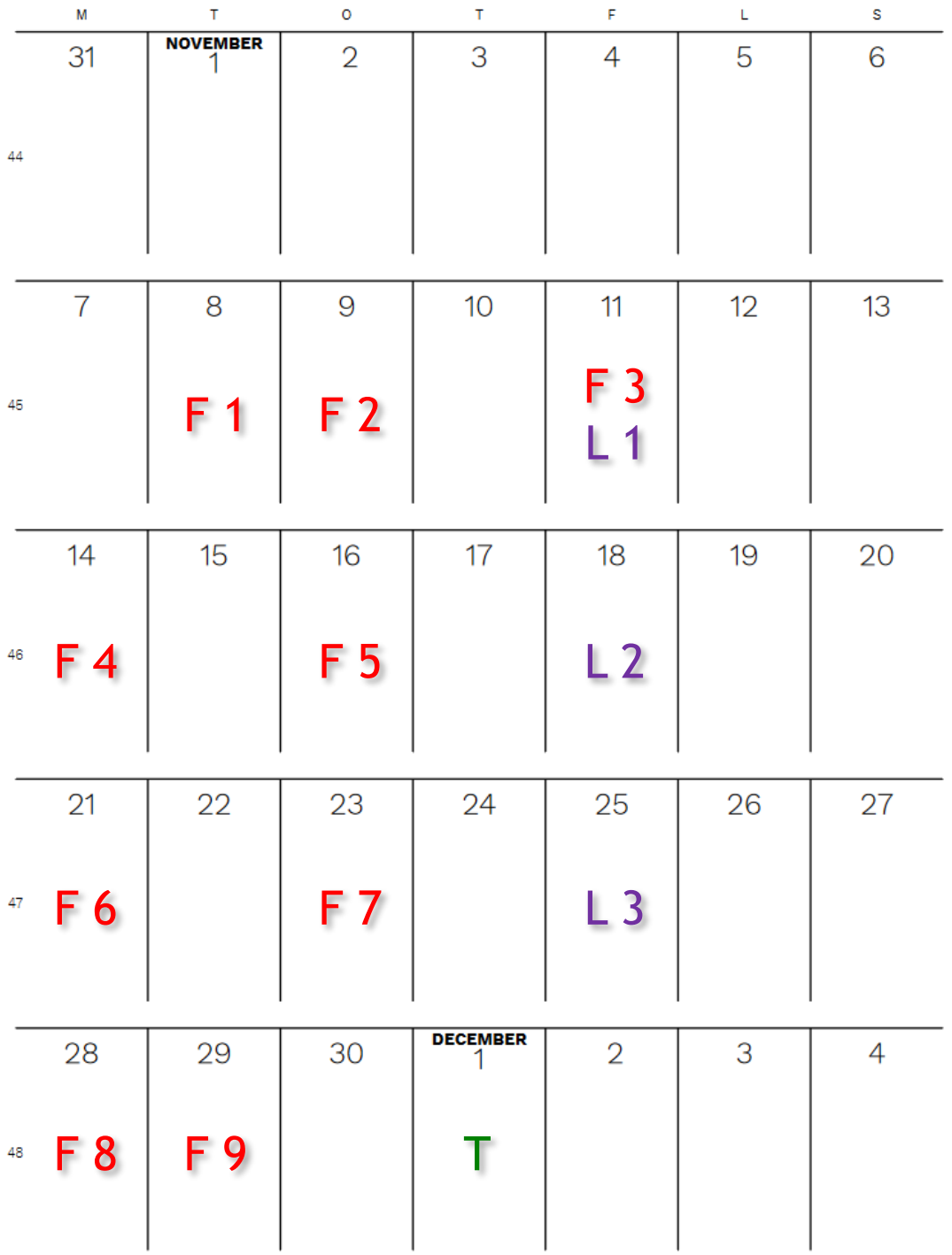
- Svar: 233168
- Källa: ProjectEuler.net, problem 1

Agenda

- Abstrakta datastrukturer
- Deque
- Sorterade listor
- Intro till Labb 2

Översikt v2 och 3

- F4 (idag): Abstrakta datastrukturer, Deque, Intro till sortering
- F5: Sortering och sökning
- F6: Rekursion
- F7: Träd och Grafer
- F8-F9: Repetition



Agenda

- Abstrakta datastrukturer
- Deque
- Sorterade listor
- Intro till Labb 2

Abstrakta datatyper (ADT)
- ADT vs datastruktur?

En Abstrakt DataTyp (ADT) är en specifikation – en mall.

- Den specifierar hur vi kan använda ett objekt – T.ex dess API
- Den består av metoder och konstruktorer, men också av en beskrivning i vanlig text
  - Ex: add() och remove() beter sig olika för stacker och köer
  - Den ska också innehålla information om komplexitet när det är viktigt

Abstrakta datatyper (ADT)
- ADT vs datastruktur?

En datastruktur är en implementering av en Abstrakt DataTyp

- Den specificerar hur objektet fungerar internt
- Den behöver inte var skriven i ett specifikt språk
  - Kan ofta vara specificerad i pseudokod
  - Ex: En stack eller kö kan vara implementerad som en länkad lista eller dynamisk array (Lista i Python), oavsett programspråk

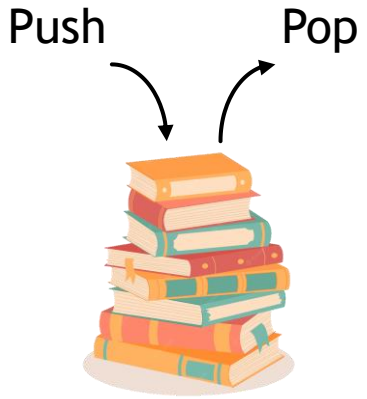
Bara metoderna räcker inte!

- Det räcker inte att bara lista vilka metoder som finns!
  De här har samma interface:

```
class Queue:          class Stack:
    add(item)             add(item)
    remove()              remove()
```

- Men de ska fungera olika!
- Vad förväntar vi oss för output här?
- Därför måste en specifikation av en ADT och innehålla en beskrivning i text av hur metoderna beter sig

Stack och Kö



- Stack - **LIFO** ("**last in first out**")
  - Push (Kallas även append, add, add_first, …)
  - Pop (Kallas även remove, remove_first, …)

Stack:

```
push(x)     # Lägger x på toppen av stacken
pop()       # Tar bort och ger tillbaka elementet från toppen
```

- Queue - **FIFO** ("**first in first out**")
  - Enqueue
  - Dequeue

Queue:

```
enqueue(x)  # Lägger till x sist i kön
dequeue()   # Tar bort och ger tillbaka elementet först i kön
```

"Kärn"-metoder och övriga metoder

Vi vill ju också ha fler är bara kärnmetoderna add och remove

- Ex. för Stack och Kö så vill vi veta om de är tomma
  - Alternativt kan vi förlita oss på att fånga fel, men det är inte alltid en bra vana
- Vi vill också kunna göra en "sneak peek" utan att ta bort elementet
  - (Det kan vi göra med en pop+push på en stack, men det är att slösa med resurser)
- ..Och hitta storleken

Stack:

```
    push(x)    # Lägger x på toppen av stacken
    pop()      # Tar bort och ger tillbaka elementet från toppen

    is_empty() # True om stacken är tom
    peek()     # Ger tillbaka elementet från toppen, utan att ta bort
    size()     # Ger antalet element i stacken
```

Osorterade abstrakta datatyper

- Stack och Kö är båda sorterade datastrukturer
- Vad har vi för osorterade strukturer vi har stött på?
  - Bag (Från förra föreläsningen)
  - Set
  - Maps

Osorterat

- Sets

```
Set:
    contains(item)  # tests if x is in the set
    add(item)       # adds x to the set (if it's not already there)
    remove(item)    # removes x from the set (if it's there)
```

- Maps ("Dictionary" / "Dict" i Python)

```
Map:
    contains(key)      # true if there is an association for the key
    get(key)           # returns the value associated with the key
    put(key, value)    # associates the key with the value
    remove(key)        # removes the value associated with the key
```

Agenda

- Abstrakta datastrukturer
- Deque
- Sorterade listor
- Intro till Labb 2

Stack + Queue = Deque

- Uttalas "Deck"
- Kommer från "Double-Ended Queue"
  - Kombination av features från olika ADT

- Exempel på ADT för Deque:

Deque:

```
    add_first(x)   # Lägger till x på vänstra sidan av dequen
    add_last(x)    # Lägger till x på högra sidan av dequen
    remove_first() # Tar bort och ger tillbaka värdet längst till vänster
    remove_last()  # Tar bort och ger tillbaka värdet längst till höger

    is_empty()     # True om dequen är tom
    size()         # Ger antalet element i dequen
```

Stack + Queue = Deque

Så varför använda en stack eller kö när vi har en deque?

- Implementeringen kan skilja sig, vilket betyder att operationer kan vara långsammare (eller använda mer minne) med en mer komplex ADT
- Det är bra att veta exakt vilken ADT din algoritm / ditt program behöver: Om det räcker med en stack, var tydlig med det.
- (Men när vi implementerar algoritmen kan vi fortfarande använda en deque om vi vill)

Diskussion:

- Hur skulle vi skriva tester för en Deque-klass?

Deque:

```
    add_first(x)    # Lägger till x på vänstra sidan av dequen
    add_last(x)     # Lägger till x på högra sidan av dequen
    remove_first()  # Tar bort och ger tillbaka värdet längst till vänster
    remove_last()   # Tar bort och ger tillbaka värdet längst till höger

    is_empty()      # True om dequen är tom
    size()          # Ger antalet element i dequen
```

Diskussion:

- Hur skulle vi skriva tester för en Deque-klass?
- Först behöver vi kanske veta att (inte hur) den underliggande datatypen fungerar
  - Försäkra oss om att den har tester, så alla head, tail osv fungerar som de ska
- Unittester: Testa metoderna isolerat
- Integrationstester: Testa metoderna i kombination med varandra

Övning 1

- Implementera en Deque-klass
  - Antingen med hjälp av en länkad lista, eller en vanlig Python-lista

Deque:

```
    add_first(x)   # Lägger till x på vänstra sidan av dequen
    add_last(x)    # Lägger till x på högra sidan av dequen
    remove_first() # Tar bort och ger tillbaka värdet längst till vänster
    remove_last()  # Tar bort och ger tillbaka värdet längst till höger

    is_empty()     # True om dequen är tom
    size()         # Ger antalet element i dequen
```

Övning 2

- Implementera en Stack och en Kö som använder Deque-klassen under ytan

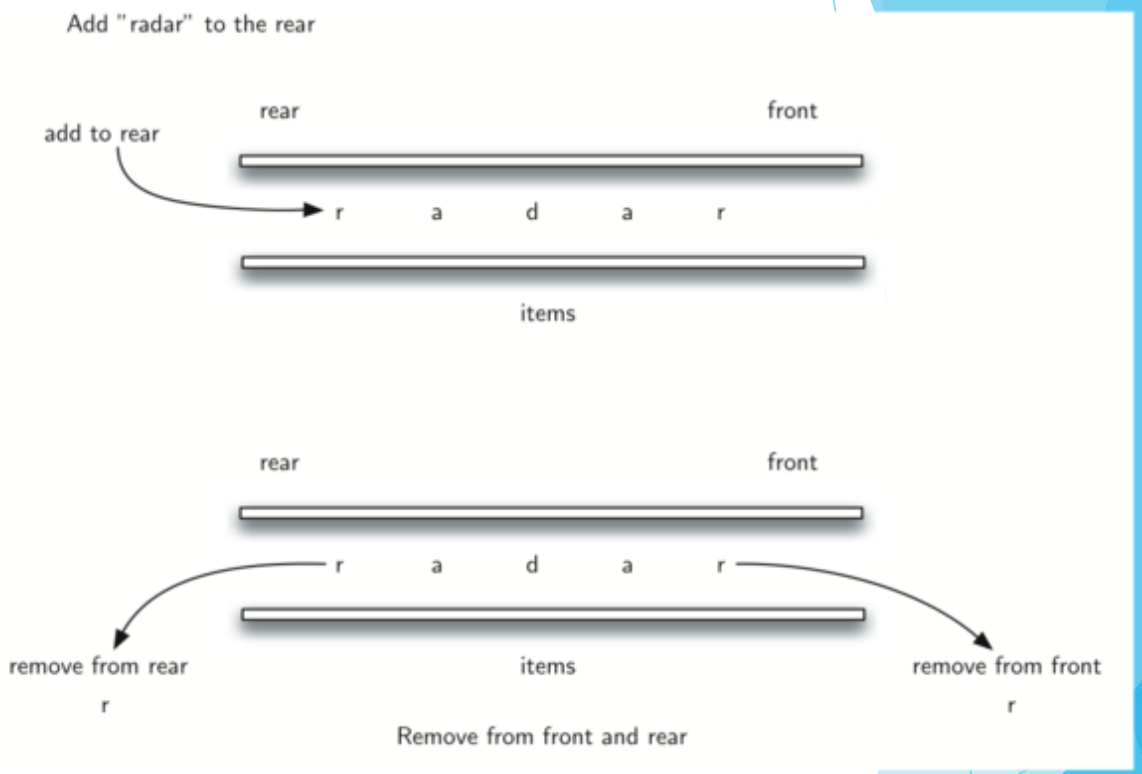
```
Stack:
    push(x)    # Lägger x på toppen av stacken
    pop()      # Tar bort och ger tillbaka elementet från toppen

Queue:
    enqueue(x) # Lägger till x sist i kön
    dequeue()  # Tar bort och ger tillbaka elementet först i kön


För båda:
    size()
    is_empty()
    peek()
```

Övning 3

- Skriv en metod som testar om ett givet ord är en palindrom
  - is_palindrome(string)
- Tips:
  - Använd en deque för att lagra bokstäverna i strängen. (Loopa igenom och lägg till tecken för tecken i datastrukturen)
  - Lös uppgiften genom att ta bort ett tecken från vänster, och ett från höger samtidigt. Om de är samma så kan vi fortsätta. Om de skiljer så är det inte en palindrom.
  - Lösningen är klar när det finns 1 eller 0 tecken kvar



- is_palindrome("racecar") == True
- is_palindrome("tacocat") == True
- is_palindrome("palindrome") == False

Agenda

- Abstrakta datastrukturer
- Deque
- Sorterade listor
- Intro till Labb 2

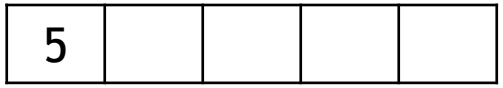
Sorterade listor

- Tanke: Hur ser vi till att en lista alltid är sorterad?
  - Sorterar den när någon använder den?
  - Sorterar vid skapande?
  - Sorterar vid uppdatering?

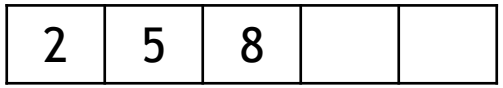
Sorterade listor

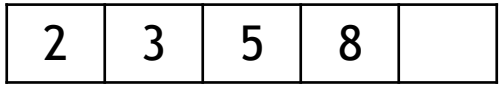
- Grundfall: Starta med en tom lista: 

- Infoga värden ett åt gången, se till så nya värden sätts in på rätt plats

add(5) ⟶ 

add(2) ⟶ 

add(8) ⟶ 

add(3) ⟶ 

Agenda

- Abstrakta datastrukturer
- Deque
- Sorterade listor
* Intro till Labb 2

Inlämning 2

- Sortering av kortlek
- Se detaljer på Omniway
- Deadline på fredag kl 23:59
- Länk till startkod på GitHub
- Inspiration från boken. (LinkedList, men principen fungerar liknande för vanliga listor)

Mer läsning

Bok

- **Problem Solving with Algorithms and Data Structures Using Python**
- Länk
- Dagens material: Kapitel 1.5, 3.4, 3.11, 3.15-3.18

- Nästa gång: Sortering och sökning
  - Om man vill läsa i förväg:
  - Sökning: Kapitel 5.2-5.4
  - Sortering: Första delarna av: Kapitel 5.6, 5.9, 5.12-5.16