Week - 1

L1.1

Video-1 30/6/21

CLASSTIME Pg. No. 1
Date / /

1. Introduction to Datasets

- patterns of computation becomes easier
  1. School data - name, gender, DOB, city, MM MP MC MTotal etc
  2. Shopping data - shop name, cust name, item, category
  3. Word data - word, type of speech, no of letters
- this data to illustrate CT

Video-2

2. Introduc^{n} to course

- CT is basics for computer to how to write prog
- every computer or program requires steps ⇒ details ⇒ depends on how much you have instructed
- programming → lang. to talk to computer
- pattern - sequence of steps. Eg. Books, Clothes

⇒ Video 3 & 4 are repetitions of 1 & 2 respectively

Video-5 L1.2

5. Concept of variables, iterators & filtering

- to know how many students in a class? HHH ⇒ 5
- Iteration - going thru the sequence of objects, u have and for each obj. u keep a count (gen. from 0 init.)
- Variable - quantity is a no. (maybe) which can Δ eventually. (value keeps changing)
- Eg. Avg. marks of maths = sum of all maths marks / total students
- Keeping track of 2 or more variables in single iteration.
- in the word data ⇒ want only verbs ⇒ concept of filtering ⇒ to catch some particular thing / from data

Video-6

6. Tutorial for Video - 5

- Eg. Count the no. of bills which have -
  (a) no food item purchase — AF count = 0 X ... 10
  (b) only " " " — NF count = 0 / ... 6

- Eg. Calculate the avg. word length of dword data
  Word count = 65
  Total letters = 329 Avg = TLC / WC = 329 ~ 5 letters / 65

Video-7 & 1.3

7. Iterations using combination of filtering conditions
- some prop. we'll take up ⇒ then iterate
- class data ⇒ Eg. how many Girls from Chennai
  - gender, city ⇒ 2 iterations + 2 filters (5/30)
  - now to do in 1 step ⇒ 'and' condition
    both should be true ⇒ to select the card
- Eg. born btw. the half year → Male = 8
  → Female = 7
- Eg. Girls / Boys doing better in Maths
  - not look at max.
  - avg. of boys / girls and then compare
  - boys / girls count, boys / girls Maths marks in 1 iteration
- algorithm problem - systematic procedure

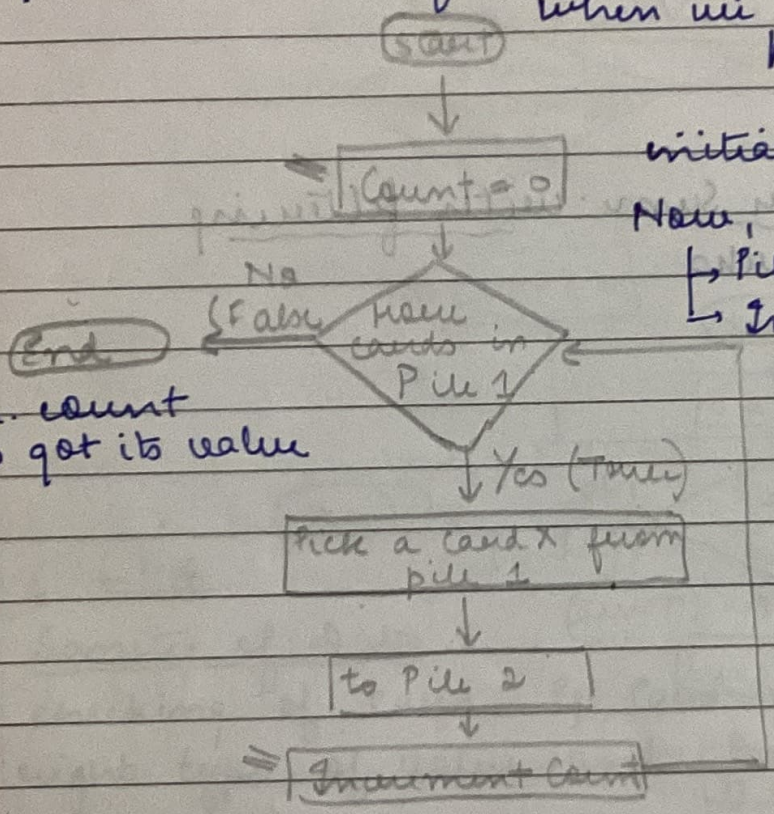
Video-8

8. Tutorial for Video-7
- Eg. Word data - Eg. no. of adjectives, > 4 letters
  - LargeAdjectiveCount = 7
- Bill data - Eg. How much Sunitaan spend —
  - SVStores - 843 BigBazaar - 0 Sun Gym - 694
- Word data - Eg. compare the size of avg. preposition and avg. pronoun (in one iteration)
  - PrepCount 10 ✓ PrepLength 27
  - ProCount 7 ProLength 17
  - After that calculate avg. for comparing

Video-9 L 1.4

9. Introduc" to Flowcharts

- there is initializa" step, then we keep repeating
- step wise procedures in formal way
  - ↳ Flowcharts and Pseudocode
- symbols in flowcharts
  - Process / Activity ⇒ Set of op^ns that s the valu of data.
  - Flowline / Arrow ⇒ Shows the order of execu" of program steps
  - Decision ⇒ determines which path the program will take
  - Terminal ⇒ indicates 'start' or 'end'
- Eg. Flowchart for counting cards
  when we start, cards are in a pile → pile 1.

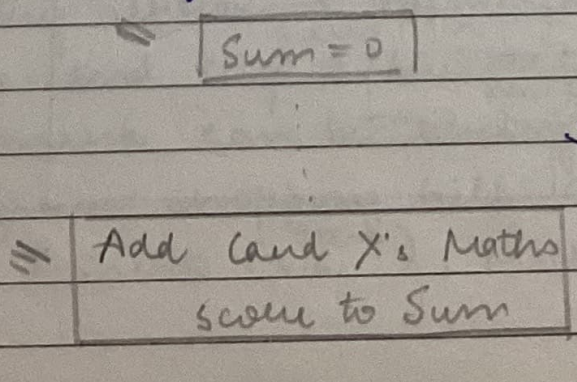


initializing the count to 0.
Now, iteration starts
↳ Pile 1 → Pile 2
↳ Inc. the value of var.

Var. count has got its value

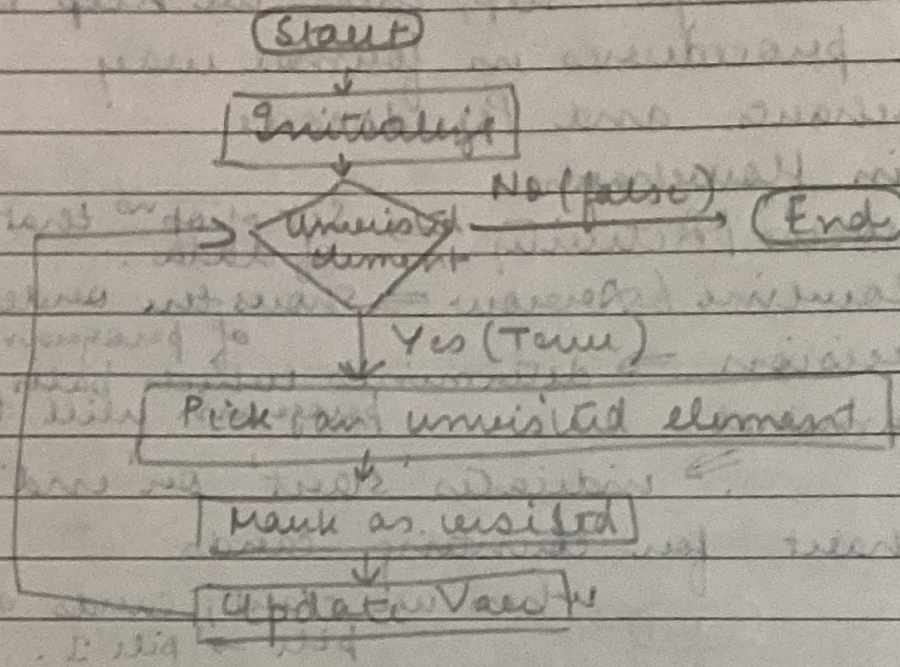
go back to Start"

- Eg. Flowchart for sum of maths marks



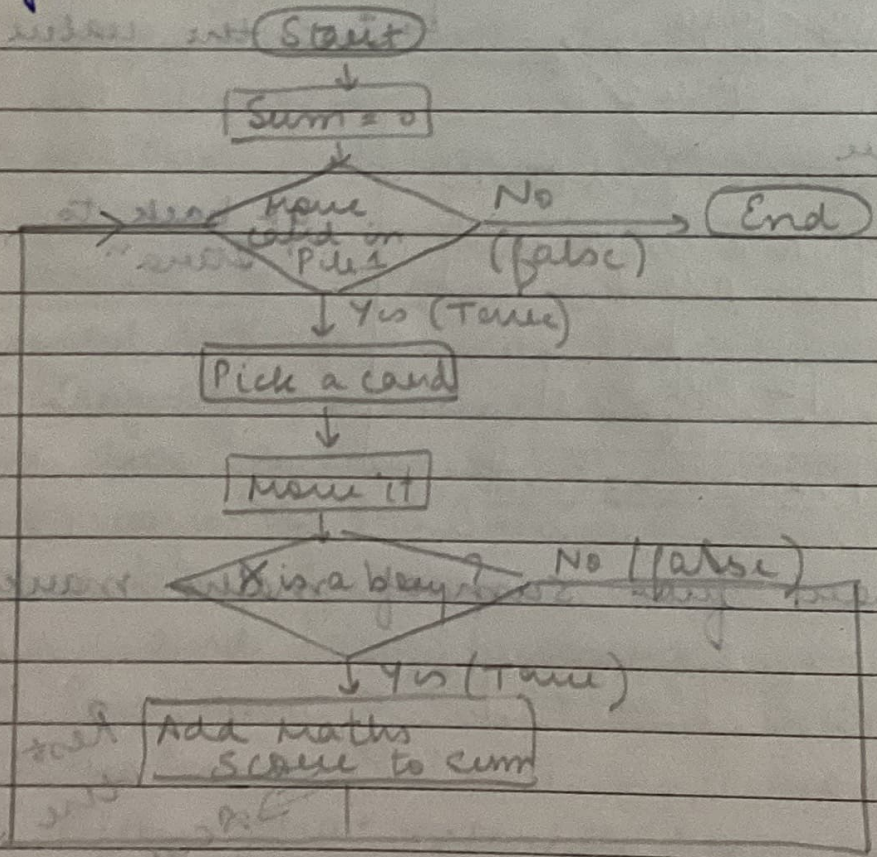
Rest remains the same
⇒ Accumulate the sum

- Generic flowchart for iteration



Video-10 L1.5

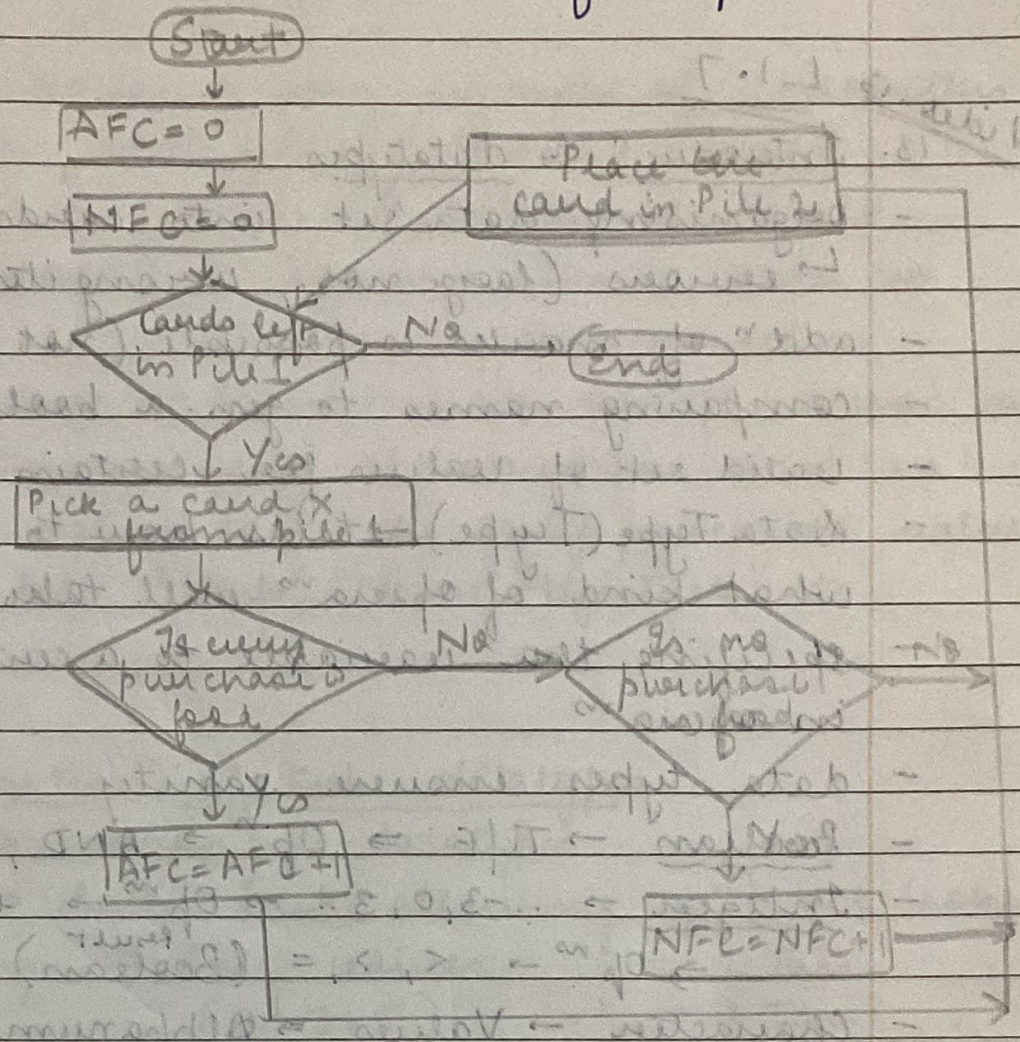
10. Flowchart for Sum with filtering - Boy Maths marks



Video-11

11. Tutorial

- Eg. All Food Purchase and no food purchase.



Video-12 L1.6

12. Sanity of Data

- checking of card. Eg. Card no. → cannot be 0
- right type of value and further they should be in range
- typing mistakes → sanity of number [Std. is wrong]
- data entry error → name pasted in city
- Eg. D.O.b. and city ⇒ swapped values
- Total mark can be wrong (added wrongly)
- Errors from shopping bill data - Wrong item name, T-Shirts (3.6 ~~Item~~), Milk (Qty → 2.1 ~~L~~), etc.

- Word data set - card no. (3F), considered, Swami → Noun (x Adj.), open (letter count → 6x)

Video 13 L 1.7

13. Introduc^{n} to datatypes

- organising data set into cards
  ↳ errors (large nos., wrong item, extra decimal)
- add^{n} of marks is possible (not multiplica^{n})
- comparing names to gen. a boolean type (T/F)
- valid set of values for certain elements
- Data Type (Type) → tells how to use that element, what kind of opera^{ns} will take place
- specifing the variable to a certain, constraints in opera^{ns}
- data types ensure sanity
- Boolean → T/F ⇒ Op^{n} → AND, OR
- Integer → ..-3, 0, 3.. ⇒ Op^{ns} → +, −, ×, ÷ → some constraint
  ⇒ Op^{ns} → <, >, = (Result Boolean)
  $\frac{21}{7} = 3 \rightarrow \frac{22}{7}, 3$
- Character → Values ⇒ Alphanumeric A, a, 1, 0, etc
  ⇒ Spl. charac. . , ; , $, #, @
  ⇒ Op^{n} → = (Result Boolean)

Video 14 L 1.8

14. Subtypes of basic datatypes

- Sequence number → 0 - Range (Reasonable)
  ↳ no integer op^{ns} doesn't make sense.
- Marks → +ve, 0 - 100 (For one subject)
  ↳ + (add), − but ×, ÷ no sense
- Count → +ve, 0 - Max Range
  ↳ +, − ⇒ Integer Result, <, >, = ⇒ Boolean

All these were integers

- Characters → Gender → Values M/F
  ↳ =, Boolean Result

- String - Values → any sequence of characters
  ↳ char in string? ↳ Boolean Result
  ↳ = ⇒ Boolean Result
- Name (Strings) → Values → strings with no spl. characters
- City (Strings) - Strings with no spl. charac. value
- Words - Strings with alphanumeric and . , ; : ...
- Category - Values can take only prescribed values like "Verb", "Noun", "Adjective," etc.

L1.9 Transformation of sub-datatypes

- we have to work on sub-datatypes
- Integer ⇒ Date (like allowing op^n) but looks like string ⇒ Date values (0-365) 0 → 1 Jan, 1 → 2 Jan ...

print(0) = "1 Jan"	Op^n	Result
print(31) = "1 Feb"	print	String
	<, >, =	Boolean

- Int ⇒ Frac. marks 62.5 → what to do?
- We can use type for real nos. ⇒ Float
  ↳ we might have 2 decimal places
  ↳ multiply with 100 ⇒ it becomes int. ⇒ now perform integer operations

62.5 → 6250

print(6250) = "62.5"	Op^n	Result
	print	string
	+ -	Marks
	>, <, =	Boolean

- Int ⇒ Amount ⇒ same 27.50 ⇒ 2750 ...

- Int. ⇒ Quantity ⇒ could be frac.ⁿ also
  ↳ 1.25 → 125 stored, print (125) = "1.25"

L1.10 Introducⁿ to complex datatypes

- 2 ways of bundle → record & list
- record → data items with no. of elements
  ↳ data type with multiple field ↳ name, value
  ↳ also c/d struct & tuple
- sanity of field values is ensured by data-type specificaⁿ

	Name of field	Type
Marks card (Record)	Seq. no.	Seq No
	name	Name
	gender	Gender
	dob	Date
	city	City
	M marks	4 Marks
	P marks	
	C "	
	T "	

-	Word in Para (Record)	word category Letter count	Word Category Count

- List - a seq. of data elements (a seq. of record)
  ↳ Marks list → is the data type for our data set of all marks cards
  → Each element in the seq. is of marks card record data type
- Shopping bill (Record) — total bill value ?? Amount; items ?? Item list
  Item ⇒ Name, Category, Cost, Quantity
- Record types → collecⁿ with same or diff. type
- List type - seq. of items, with same data type.

Week-2

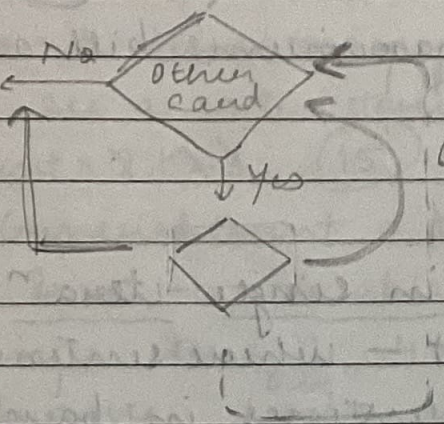
Needs Rev.

CLASSTIME Pg. No. 9
Date / /

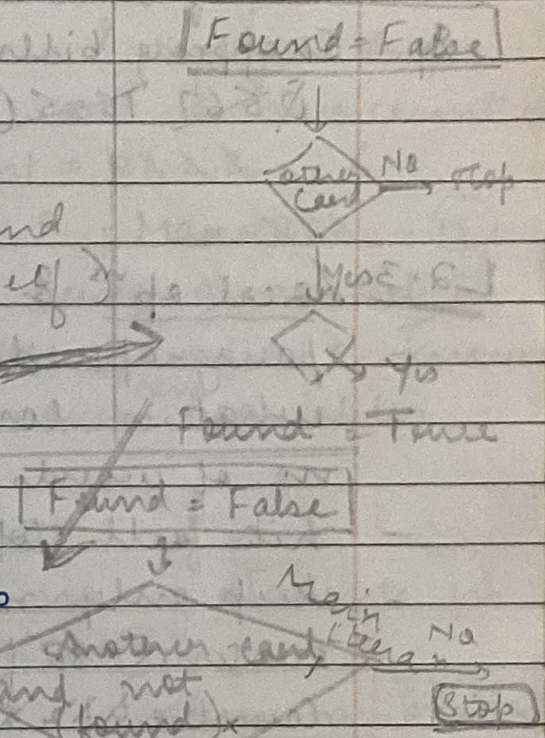
L2.1 Conditional termination in Iteration

- some search for a specific student ⇒ for criteria
Eg. Student with >90 in all subj. → stop or not?
Eg. any customer → food + apparel + toiletry (3 ✓) → 1 stop
Eg. First word in third sentence. Scount = 2 × 2 → Now word
↳ go on to second sentence find the word in third sentence ⇒ Word No. 12 ⇒ no need to go ahead

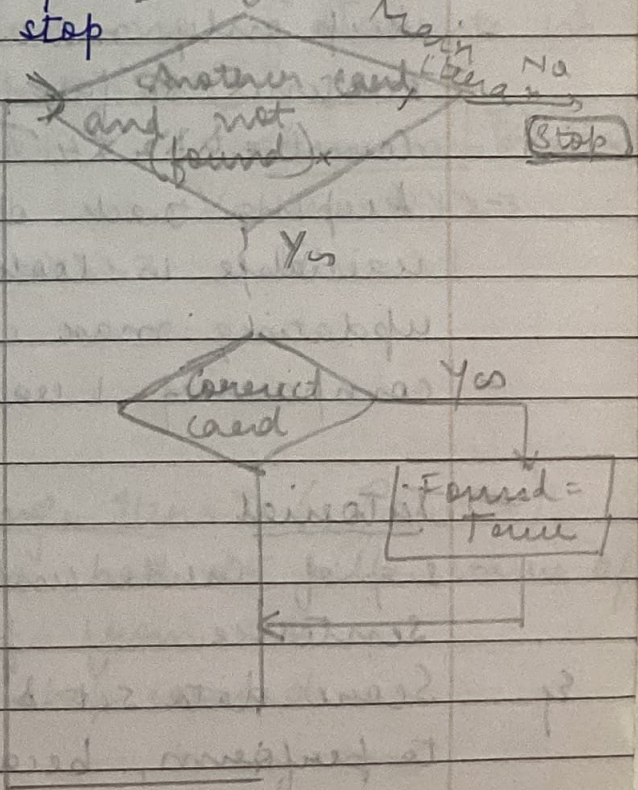
Bit unclear



Find / Search Pattern



- conditioning itself in looping
- this gives the way to stop iteration with a termination
- if I stop, that means I found it ⇒ no student then found would still be false
- exist or not ⇒ kind of Check



L2.2 Local operations & max in single iteration (Part 1)

Eg. Best total marks — ~~270~~ ~~253~~ ~~276~~ (281)
- continue check & when marks beats it, replace it
- keep variable - replace the last mark.
- filtering + checking against the variable

Tutorial

Eg. Words data → find the word with maximum no. of letters → max = ~~2~~ ~~2~~ ~~8~~ ~~7~~ (11)

Eg. Shopping bills → maximum bill amount = ~~0~~ ~~567~~ ~~7525~~ (4174)

L2.3 Local ops & max in single iteration (Part 2)

- paragraph data set — when sentences ends, what is longest sentence in para → has most no. of words
- hit full stop → keep count seperate → better way
- keep only max. no. of words till then, longest sentence so far, Max = ~~0~~ ~~4~~ ~~7~~ ~~8~~ ~~20~~ (25)
  Count = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ; ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ... so on
- keeping track of longest sentence, count variable is looking changed (restarting) and updating max count, in 1 iteration
- comparing 1 variable with other variable

⇒ Tutorial
- eg. of words until a fullstop is taken to be sentence

Eg. Score data set & shopping bill is bit problematic to perform, because words are in a paragraph so they are in a ordered way

Eg. Scores dataset are arranged in a month way. To find out max no. of students born in a month
Max = 0 1 2 3 4 Count = 0 1 ; 0 1 ; 0 1 2 3 4 ... so on

L2.4 Local Op^ns & Max in single itera^n (Part 3)
- which shop is doing well → how many people go there? × → how many bills they generate
- we don't have many stores? segregate shops
- if we see a new store, we start a new count
  SV Count = 0 1 2 3 ... 15 BB Count = 0 1 2 3 ... 6
  Sun General Count = 0 1 2 3 ... 9 ; Max = 0 1 2 3 ... 15
- more & more variables are being generated ↑
- in single itera^n, 4 variable, we didn't know in advance, 1 max & unknown no. of variables

→ Tutorial
- Q find the city with max. no. of students in scores dataset ; Max = 0 1 2 3 4 5 6 7 8
  Erode = 0 1 2 Salem = 0 1 Chennai = 0 1 2 3 4 5 6 7 8
  Madurai = 0 1 2 3 4 5 Ambur = 0 1 Vellore = 0 1 2 3
  Bengaluru = 0 1 2 3 4 5 Trichy = 0 1 2 3 Teni = 0 1
  Nagercoil = 0 1

L2.5 Local Op^ns and Max in Single Itera^n (Part 4)
- Shopping bills → SV stores had 15 bills, value of bills was less, but diff. from BB
- total value of bill the shop earned

SV = ₹567+1341 = ~~1908~~ ... 7120 GG = ₹~~354~~ ... ~~4009~~
BB = ₹1525 = ~~5699~~ 12486 Max = ₹~~567~~ ~~1525~~ ~~1908~~ 1248

- 1 itera^n to add all bills and compare with the max and further update it.

⇒ Tutorial

- words data set ⇒ which part of speech contributes most charac. to the text? Letter count

Pronoun = ₹2...
Verb = ₹3...35
Adjective = ₹9...65
Adverb = ₹9..13
Conjunc^n = ₹3..9

Max = ₹2~~3~~~~7~~ = ~~13~~~~24~~ ... 145
Noun = ₹~~8~~~~13~~24 ... 145
Preposi^n = ₹2 ... 27
Article = ₹3 ... 18

L2.6 Max in 1 & 2 itera^n (non-nested)

- which caud is most freq. in paragraph dataset; see each caud and make new variables accordingly
  ₹t = ₹1 Was ₹1 Monday ₹1 Morning ₹1 ... so on
- creating a variable for each word
- count is becoming large, very tricky to not make mistakes
- 2nd itera^n, keep variable max at change it as and when req.
- so we can do both the itera^n in 1 go
  Max = 6 (the)
- we don't know which word, but that can be done by seperate variable

CLASSTIME | Pg. No.
Date / /

⇒ Tutorial
- Shopping bill → no. of bills / customer
- when we see new customer, we will make a new variable each time, if that person comes again, we'll add the count
- we can do the max. and inc. no. of variables in 1 iteration Max = 4 (iteration) → Ahmed

L2.7 Max. in single iteration w/o losing info.
- so who got max marks? not only max. marks, but the person who is associated to it.
- if we update max, and change card no. (unique)
  Max Maths = ~~68 87~~ 97 Max. Card No = ~~1 2~~ 11, 23
- 2 people who got max. maths marks, if we got 99 marks, then replace both people.
- SEO → applicn of max. frequency, keep relating unique words gives best results
- Google engines gives best relevant words
- we need to keep 2 iterations → 1st to find freq. count; 2nd to use as an application

⇒ Tutorial
- max. bill amt. & which customer has paid that max. amt.
  Max. bill = ~~567 1525~~ 4774 Cus. = ~~None~~ ~~Sumeetram~~ Sudeep Akshaya

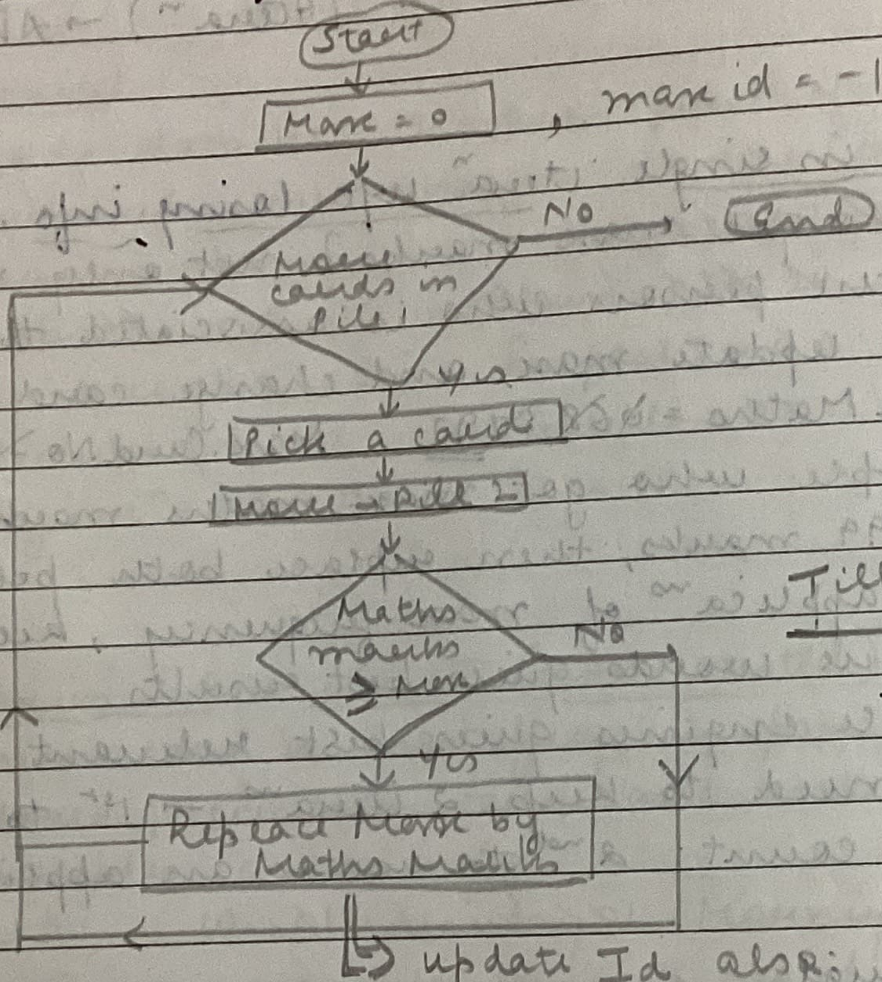
⇒ Tutorial
- initialisation of variables; count = 0; sum = 0
  Name = (String) None
- Min - - - - Max in chemistry scores

initialize to 94 ✗

- Min (43) . . . - - - Max (93)
  initialize to 35 ✗

0 . . . Range 100
Max initialize — Min initialize

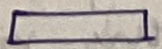
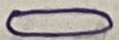
L2.8 Flowchart for max marks



Till here same

- keep track of card which has the max marks
- we can keep diff. no. of people who had scored highest marks
- append of a & b is a,b (helps to create a list)

L2.9 Introduc" to Pseudocode

- pseudocode - textual represenla" of procedures
- node types - process 
  - decision 
  - terminal 
- arrows indicate opera" flow
- adv → visual represenla" of computa", easy to understand
- disadv. → size problem, collabora" restricted, version cng. (modifica" needed)

```
Step 0  Start
  "  1  Initialize count = 0
  "  2  check cards in Pile 1
  "  3  If no more card, Step 8
  "  4  Pick a card x
  "  5  Move to Pile 2
  "  6  Increment count
  "  7  go to step 2
  "  8  End
```

- succint nota" for computational processes
- better textual represenla" for conditional & repeated execution.

```
Start
Count = 0
while (Pile 1 has more cards) {
  Pick a card X from pile 1
  Move X to pile 2
  Increment Count
}
End
```

L2.10 Pseudocode with iteration & filtering

- with always, not use start & end
- Sum of maths marks

```
Sum = 0
while (Pile 1 has cards) {
    Pick a card X
    Move X to Pile 2
    Sum = Sum + X.Maths
}
```

this is a assignment statement ⟸ Eg. C = C + 1

Maths field of Card X

- sum of boys math marks

```
Sum = 0
while (Pile 1) {
    Pick a card X
    Move to Pile 2
    if (X.Gender == M) {
        Sum = Sum + X.Maths
    }
}
```

== Equality
= Assignment

- sum of both girls & boys marks

```
BoySum, GirlSum = 0
while (Pile 1) {
    Pick a card X
    Move to Pile 1
    if (X.Gender == M) {
        BoySum = BoySum + X.Maths
    }
    else {
        GirlSum = GirlSum + X.Maths
    }
}
```

alternative branch for conditional execution

- find the max. maths marks

```
MaxM = 0
while (Pile 1) {
    Pick a card
    Move to Pile 2
    if (X.Maths > MaxM) {
        MaxM = X.Maths
        MaxCard = X.Id
    }
}
```

who is that person

MaxCard = -1

CLASSTIME Pg. No.
Date / /

→ Tutorial

- count = 0
  while (Pile 1 has cards) {
    Card x = Pile 2
  if (x.TownCity == "Chennai") {
    if (x.Gender == "Female") {
      count = count + 1;
    }
  }
  Move x to Pile 2
}

- we could do 2 if statements in 1 way
  if (X.TownCity == "Chennai" & X.Gender == "Female")

Week-3

CLASSTIME Pg. No.
Date / /

L 3.1 Sequentia" of datasets in table

- each card ~~has~~ is a unit of info., diff attributes or fields → organise as a table
- each card is a row now + 9 columns
- new card → new row
- we can do with same with word dataset
  Id Word Type Length
- what 're the attributes in shopping bills?
- some are 1 but items, category and so on
- which ill create a variable no. of columns
  → no longer a neat table
- variable no. of row? tag to unique Id
- some things get duplicated multiple times
- with this, problem ↑s as entries increase
- we can create multiple tables

Id	Shop	Customer	Table

Id	Item	Cat.	Qty	Price

the Id links the 2 tables

⇒ Tutorial

Eg. LongNoun = 0 ; Table 1 (cards) ; Read the first
ShortNoun = 0 ; Table 2 (empty) ; Row of table 1

Noun → >7 ⇒ LN = LN + 1
→ else ⇒ SN = SN + 1

LN = 8 x 2 3 x 7
SN = 8 x 2 3 x 5 8 8 9
+ 14

L 3.2 Below avg. students in 2 itera" (non-nested) & graded collaca"

- we want to find avg., and then who are below avg. ⇒ most probably should be half
- count = 1, 2, ... 30 — 217.1 / 30 = avg. = 72.36
  sum = 68 130 ... 2171
- so who 're below avg. students?

BA count = 1, 2, 3 ... 15 (30/2) (anything 72 or <72
↳ it's not necessary always half

- grp. the student, how do they have done? → grades
Range → lowest to highest real marks
(20) (97) 78 ↳ 4 equal parts

- for finding min. → min = 100 68 62 ... 42

Range → 42 97
55 69 83
37
(20)
14

avg. was 72.36

A 84-97 → 8
B 70-83 → 10
C 56-69 → 9
D 42-55 → 3

- this gives a better pic. → how class is doing

⇒ Tutorial

Eg. all the bills above avg.
1. find avg:
2 2nd criteria → who 'r

count = 1, 2, ... 30
sum = 2587 ... 24413

24413
30
813.76

→ so, now those bills who 're above 813.76
AAvg count = 8

Eg. Low exp. 1
Med. exp. 26
High exp 3

For that
Max. = 4174
Min = 5000 ... 42

42 1420 2798 4174
1419 2797

4133
3
1378

L3.3 Systematic Procedure of Hypothesis Verificaⁿ

- freq. words are gen. smaller and large words are less freq.
- $\frac{\text{sum}}{\text{count}} = \frac{329}{65} \sim 5$ — 5> is small word, 5< is long word

it was to his he the of and	High Monday Freq. but long
this. confirms short (8) ✓	(1) ✗ Long ✓
came long thing ✗	headmaster teacher wednesday so this proves that
(15)	(23) freq. words are gen. smaller
they refute	confirms

47 distinct words → 31 confirms
↳ 16 refute

hypothesis is fine

- there is correlaⁿ btw. length of word & freq. of how do they occur

L3.4 Three Prizes Problem

- award prizes to best student ; top 3 students of a class ; min. both → 1 girl & 1 boy

1 →
2 → } criterions
3 →

- Top 3 marks → keep current 3 aside

281 → M
276 → F
261 → F

-

maths			phy			chem		
1	2	3	1	2	3	1	2	3

- go on replacing & checking

CLASSTIME Pg. No.
Date / /

L 3.5 Intro. to procedures & parameters

- compare the highest T with highest marks of each subj. added up., if diff. is less, that means that highest topper is doing better in all subj.
- this is a procedure → repeatedly executing; some set of instruc^{ns} + changing the field name (parameter)
- max → procedure, P, C, M, Totals → parameter

Eg. Max (Maths) ⇒ 97
Func^{n}

L 3.6 Pseudocode for procedures & parameters (Part 1)

```
Procedure SumMaths (gen)
    Sum = 0
    while (Pile 1 has cards) {
      Pick card X
      Move X to pile 2
       if (X.Gender == gen) {
           Sum = Sum + X.Maths
       }
    }
    return (Sum)
End SumMaths
```

(gen, fld) ← gets assigned
value for field name
gen(F)
X.fld

△ procedure to Sum up Maths marks

⇒ 1 GirlChemSum = SumMarks (F, Chem.)
2 BoyChem Sum = " " (B, Chem)

```
if (1 > 2) {
  "Compa. to girls"
}
else {
  Compa to boys
}
```

- procedure may not return a value, like updating marks → it is a separate statement
- procedures are pseudocode-templates that work in diff. situations, they help to modularize pseudocodes

L3.1 Pseudocode for Parameters (Part 2)

Eg. - is there a single student who is best performer across subj.
- same computation with a parameter to determine the field

```
Procedure MaxMarks (fld)
    MaxVal = 0
    while (not EOF) {
        if (x.fld > MaxVal) {
            MaxVal = x.fld
        }
    }
    return (MaxVal)
end

if MaxTotal == SubjTotal {
    Single Topper = True
}
else {
    Single topper = False
}
```

→ Tutorial

- 3 common errors

Eg. SumBB, CountBB = AddIfBB(X, SumBB, CountBB)

Procedure AddIfBB(.., .., ..)

- #1 Wrong Procedure Name ; name used should be same in both places
- #2 Wrong no. of parameters ; if we have 3 parameters in procedure, take same in the code also
- #3 Incorrect order of parameters ; order is essential at both the places ; assignments will be wrong.

L3.8 Pseudocode for 3 Peujes Problems

- Top 3 Peujes → basic → is total marks
  → at least 1 subj. topper
  → intop 3 → min 1 boy & min 1 girl

Procedure Top3Marks (subj)

max, 2nd max, 3rd max = 0

while (Pick 1)

Pick a cand

if (X.Subj > max) {

3rd max = 2nd max ; 2nd max = max , max = X.Sub

}

if (max > X.Subj > 2nd max)

return (max, 2nd max, 3rd max)

End Top3Marks

- but, we record only 3rd max ; return (3rd Max)
- we should keep track of card no. that has the highest marks

Pseudocode

max = 0 , 2nd max = 3rd max = 0

maxid = ; 2nd maxid = 3rd maxid = -1

maths 3 = Top 3 Marks (Maths)

phy 3 = " (Phy)

chem 3 = " (Chem)

update Id & max. marks

Procedure SubjTopper
(card, MMark, PMark, CMark)
if (card. Maths ≥ Maths or Phy or Chem {
return (True)
}
else {
return (False)
}
End SubjTopper

- gender problem? 3 boys, knock off boy (last) and replace it till we find the girl

L-3.9 Side effects of procedures

- procedure to sum up any type of marks
- what abt. the set of cards? File is fixed.
- 3rd Parameter deck → Seen Deck, the procedure should also restore deck
- procedure modifies some data during compute

- seq. of cards may be disturbed, it might matter when some dec./inc. order is impt.
- each procedure comes with a contract
  → Functionality → parameters will be passed
    ↳ expected in return
  → Side effects → is it predictable
- contract specifies interface → can a procedure implementation (code) provide interface is unaffected
- there are places where side-effects are undesirable
  - no guarantee needed
  - undesirable
  - side effects are goal

Week 4

→ Tutorial
- OR statement Part of speech == Noun or Verb
- OR is true if even 1 statement is true
- The cond^n can both be true simultaneously? They are mutually exclusive.
- Not mutually exclusive ... don't split that
- AND is true iff. both the statements are true

L4.1 Concept of nested iteration (Naive)
- birthday paradox → 25 people → more 50%. probability → same month & date
- 1st iteration - 1st card → inner iteration
  - Abhinav → whole class
  - Akshaya Abhijeet → whole class → this pair won't help
  - its a iteration within iteration
- 2 customers who shop in a same shop

L4.2 Binning concept of nested iteration
- grp students by months; seperated out into bins by months → get 12 bins; each bin do same double iteration
- cards per bin decrease; the double iteration becomes easier
- ~~but~~ if use binning wisely so that it eases task; take clue from the data

CLASSTIME Pg. No.
Date / /

⇒ Tutorial

Q. Find score cards same maths with same peace

Erode = 0, 18
Salem = 1
Ambur = 6
Bangalore = 8, 9, 11, 21, 23
Teni = 15
Chennai = 2, 3, 5, 10, 12, 17, 25, 26
Madurai = 4, 13, 24, 27, 28
Vellore = 7, 20, 29
Trichy = 14, 16, 19
Nagercoil = 22

Naive methods → 435 comparisons

$\frac{n \times (n-1)}{2}$ ⇒ n no. of rows in that table

Binning methods → only 55 comparisons drastically red., if data is reasonably distributed

L4.3 Impt. of binning

- red. no. of comparisons
- Some computa~ req. comparisons of each card with all other cards
- Obs. that if we can organize cards into bins
- formal way of determining the reduc^n in comp.

A B C D E

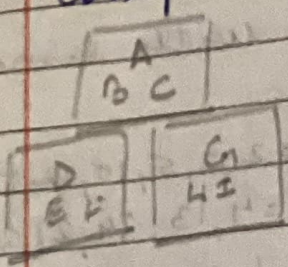
A→B, A→C, A→D, A→E, B→C, B→D, B→E, C→D, D→E

$\frac{N(N-1)}{2}$ for N obj., no. of comparison

(10)

- this is same as no. of ways of choosing 2 obj. from N obj. $^NC_2 = \frac{N(N-1)}{2}$
- the no. of comparison grows exponentially

- computer $\approx 10^8$ comparisons / 1 sec.



$9(8) = 36$ reduced comp. by a 4

so for each bin $\frac{3(2)}{2} = 3$

$3+3+3 = 9$

- for N items $\frac{N \times (N-1)}{2}$
- if we use k bins of equal size, no. of items in each bin $= \frac{N}{k}$
- Now, Total comparisons $= \frac{N \times \left(\frac{N}{k}-1\right)}{2}$
- Reduc^n by a factor $= \frac{(N-1)}{\left(\frac{N}{k}-1\right)}$

L4.4 Concept of binning to avoid complexity

- loyal customer - visited many times to a same shop → bin as we go along
- if 1 card that means too less info.
- separate all customers into bins and iterate to check → space is needed, but time decreases

L4.5 Concept of Fair Teams

- help each other in studies → grp. study
- pair of students X. Maths > Y. Maths
  & X. Phy < Y. Phy

- pairs which have same totals 150 + 25 bins
- binning is a generic method, comparisons reduced; inside the bin we again compared to make the pairs

L4.6 Procedure to find same d.o.b
- birthday → parameter; check along the whole deck

Eg. 30th April → return (False)
13th May → but 30th April, we have already checked → return in such a way that specifics are marked

L4.7 Procedure to find pronoun with their nouns
- when down to get pronoun, then went backward to find noun ⟹ now directly move further to find new pronoun
- keep a user name
- nested loop → iterating over the card, then started the inner loop

⇒ Tutorial

cvcvccvcv
reluctant

```
i = 1                  while(i ≤ X.letterCount)
V = 0                      if ith letter X.word is vowel
C = 0                          V = V+1
                           else
                               C = C+1

                           i = i+1
```

→ Summary

- iteration → most commonly used in CT, it represents the procedures of some task repeat
  initialize → count → stop
- variables → ∆ the values, at each iteration, it ∆
- initialization & update of vars. are done thru assignment statements
- terminals ⬭, process ▭, decision ◇
- use of flowcharts & pseudocode
- iteration to systematically go thru a set of items
- sanity of diff. data fields of the items
- basic data types → boolean, int., char, etc.
- list → all data items typically have the same datatype
- record → multiple named fields, each can be of diff. datatype
- filtering - a decision at each repeated step
- sometimes we have to write the same piece of code again & again with small diff.
  ↳ use procedure (small diff. → parameter)
  Eg. find max for each subject
- most common use of an iteration is to create an accumulation
- doing 2 iterations - one nested within other
- we can reduce the number of comparison by using binning methods

Week - 5

CLASSTIME Pg. No.
Date / /

L5.1 Intro. to collec^n & list data struc.

- collec^n is sets ; relationships are relations
- comp. has memory RAM → Random Access Memory → u can go to that place directly & change it
- no side-effect, with the help of bookmark
- list → indexing the positions, we can make several lists

Chennai list (outer iter^n) may list = students born in May from Chennai (nested iter^n) (inner)

L5.2 Pseudocode for lists

- variables keep track of intermediate values
- simplest collec^n is a list → seq. of values, nota^n for lists

[1, 13, 2] ["cane", "school"] [] → empty set

- append 2 lists → seq. impt. → combine

l1 is [1, 3] l2 [2, 17, 1] l1 ++ l2 → [1, 3, 2, 17, 1]

- extend l with item x l = l ++ [x] (list of length 1)
- list of student in may, list of students from Chennai

```
Maylist = [ ]
while (Table 1 has rows) {
    Read first
    if (x.MOB == "May") {
        Maylist = maylist ++ [x.Seq]
    }
}
```

- we need to iterate over a list

```
foreach x in l {
    do sth. with x
}
```

```
may chemaiclist = [ ]
foreach x in Maylist {
    foreach y in chennaiclist {
        if (x == y) {
            mclist = mclist ++ [x]
        }
    }
}
```

L5.3 Operations on data collected in three periods

- Top 3 in atleast 1 subject
- maths, phy, chem. set ⇒ any student who managed to come in all
- we do not disturb the original card set - that is key thing -

L5.4 Pseudocode for 3 Periods problem

⇒ Procedure Top3Marks (Subj)
  ↳ max, 2nd max, 3rd max
  ↳ we get return of third max (cutoff)

⇒ cutoffs M/P/C mp/c list = [ ]
  while (table!)
    if (X.Maths >= cutoff M)
      mclist = mclist ++ [X. Seq. No.]
    :

⇒ we got three lists

⇒ mp list [ ] foreach x in mclist {
    foreach y in plist {
      if x == y {
        mplist = mplist ++ [x]
      }
    }
  }

- now we'll match mp list & c list and append to form mpc list
- lists are useful to collect items that share some property

LS.5 Insertion Sort & Ordered List

- arranging in some order → find a place to insert → building up a new list
- insertion sort → we are inserting it along with sorting it with the things ⇒ (nested iteration)
- building up a ordered list.
- we sort the bookmark
  → (0, 210) ⟹ Ordered list
  → (1, 198), (0, 210)
  → (3, 173), (2, 188), (1, 198), (0, 210) - - - -
- each ordered list keeps a track of set of card in inc. / dec. order.

LS.6 Pseudocode for insertion list

- sorting a list make computations easier
- finding top k values, finding duplicates, grping by percentiles (quantiles)
- insertion sort - create a 2nd sorted list, start with an empty list, repeatedly insert next value
- list l in arranged order → insert a new element x at a appropriate place

CLASSTIME Pg. No.
Date / /

```
Procedure list (l, x)
    newl = []
    inserted = False

    foreach z in l {
        if not(inserted) {
            if (x <= z) {
                newl = newl ++ [x]
                inserted = True
            }
        }
        newl = newl ++ [z]
    }
    if (not(inserted)) {
        newl = newl ++ [x]
    }
    return (newl)
End
```

⇒
```
sorted list = []
foreach z in l {
    sorted list = insert (list, z)
}
return (sorted list)
```

- invariant - second list is always sorted
  - [] is sorted, since it is empty
- insertion sort is a natural sorting algorithm

CLASSTIME Pg. No.
Date / /

L5.7 Systematic procedure for hypothesis verificⁿ

- the person who does good in maths is good in phy. also
- distribute in grades → we had a hypothesis
- either there is confirm / refute
- this hypothis isn't much associaⁿ

L5.8 Pseudocode for systematic procedure

- list eg. → correlating student's performance
- algorithm → assign grades
  - → count A of Maths in A of Phy
  - → count of B of Maths in A/B of Phy
  - → use count to confirm / refute

⇒ **Tutorial**

- length (l) ⇒ length [20, 30, 40, 50, 10] → 5
- first (l) ⇒ first [20, 30, 40, 50, 10] → 20
  - ↳ if empty → ND
- last (l) ⇒ last [20, 30, 40, 50, 10] → 10
- rest (l) ⇒ rest [20, 30, 40, 50, 10]
  - ↳ [30, 40, 50, 10]
- init (l)
- member (l, e)

⇒ rest (l) = l - first (l)

⇒ init [20, 30, 40, 50, 10] = [20, 30, 40, 50]

⇒ init (l) = l - last (l)

⇒ member ([20, 30, 40], 30) ⇒ True

([20, 30, 40], 10) ⇒ False

Date / /

→ Tutorial

- X → books insertion

sBooks = []
updated version

someBooks = []
" = Insert [2, someBooks]

L5.9 Intro. to Train Dataset

→ train & station dataset
- pair of train city ⟺ city (paired train)
- MWF → days that train moves [01201, 02202]
- next data → list of cities and all the train that pass them it

WEEK - 6

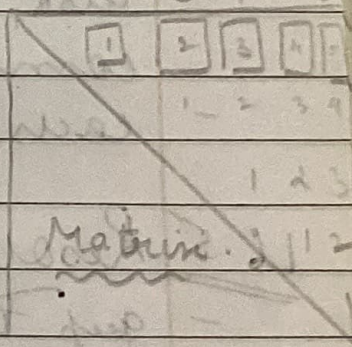
CLASSTIME Pg. No.
Date / /

16.1 Rela^n of customers - Part 1

- Segmenta^n - segment of customers for ad
- Abhinaya — Appareal — Food — Toiletries (Rows of items)
  Neerja — 2 — 7a
  Akshaya — 4 — 6 — 5 ... so on
- we got a table, rows are customers, columns are categories
- Radha, Akhil, Julia, George ... → Food only
- 1 grp. with more than 10' & 1 grp with 10' or below 10
- atleast everybody buys food across all bills irrespective of the grp.

16.2 Rela^n of customers - Part 2

- 6 customes - each compared with everyone. 15 comparisons will be there
- compute the distance & the pair which has the least difference becomes a similar customer
- Rectilinear / Manhattan distance



- tables are good way of organizing things
- gruping in some 1 columns

16.3 Intro. to dictionary data struc.

- which shop gives the max. no. of food items
- iterate them card and in inner iterat^n, we count all the food items in that list
  ↳ SV stores

- which customer buys most food?, there are many customers — how to handle?
- so the customer who buys the max. food item is from the shop that sells max food item — thus, hypothesis ✓

L6.4 Concept of dictionary to solve birthday problem

- dictionary — indexing, index is full b.o.d
- build dictionary in 1 go

DOB	Card No.
3 June	1
5 May	3
.... so on	

→ only 1 b.o.d. in common and also dictionary told that there aren't any more duplicates

any element which has 2 entries?

- also, keep track of diff. birthdays that come up
- each year in dictionary store. can be accessed using key

L6.5 Resolve pronoun with its matching noun

- grp. together by some category
- pronoun & noun - tracks

Pronoun - 0, 9 ... 61

Noun ... 2, 3, 4 ... 64

- 9 → his ; go before 2, 3, 4 and it should sensible also

4 ← 9 11 ...

- dictionary by category — and we are storing the seq. no. as key
- partitioning each word into any of the categories ⟹ binning

L6.6 Pseudocode for dictionaries

- a list has seq. of values, can iterate them list
- dictionary stores key-value pairs
  - chem marks (value) for each student (key)
  - source sta" (value) of train route (key)
- present the key to extract the value
  m = chemmarks ["Rahul"] ⟹ random access allowed
- seq. of 'key: value' pairs {"k": 92, "L": 86}
- empty dictionary {}
  - chemmarks ["Rahul"] = 92 (either replaced or added)
- the dictionary must exist to create a new entry

```
chem = {}
while (more rows){
    Read first row
    name = X.Name
    marks = X.Chem
    Chem[name] = marks
    Move X to row 2
}
```

- keys(d) is the list of keys of dictionary

```
foreach k in keys(d) {
    do stn
}
```

```
total = count = 0
for each k in keys(chem) {
    total = total + chem[k]
    count = count + 1
}
chemavg = total / count
```

eg.

- use of a dictionary is to accumulate values

num["kohli"] = num["kohli"] + score → update

but, in beginning

num["kohli"] = score → assign

is key(d, k) → returns True if k is a key in d

```
if is key (num, "kohl") {
        update
} else {
        first assign
}
```

- 1 Problem ⇒ Take time proportional to size of the dictionary ⇒ random access

L6.7 Pseudocode for real-time eqs. using dictionary

- find the customer who buys highest no. of food item

```
food D {}
while (Table 1 has rows) {
    read the first row
    Customer = X. Cust Name
    items = X. Items
```

CLASSTIME | Pg. No.
Date / /

```
for each row {
    if (row.category == "Food") {
        if (isKey (foodD, cust)) {
            foodD[cust] = foodD[cust]+1
        }
        else {
            foodD[cust] = 1
        }
    }
    Move row → table 2
}
```

L6.8 Dic. comparison to find common elements

- train dataset | — dic. — train is the key
  2 | — dic. — stn is the key
- Mangalore directly to (x) Pune

02283 } Stations list
12224 }

Now which all trains reach Pune

Best should be Mumbai

L6.9 Procedure to find rela^n btw. customers

- 1 contractor keeps the comp. list of all values
  ↳ sub-contractor works for 1 person and iterate thru 1 card
- sub-contractor doesn't have to do anything abt. the name
- this becomes modular & easy, the decision-making is now divided
- parameter is the card, return value is high
- side-effect → call for sth., he does sth-else

L6.10 Side effects for list & dictionary

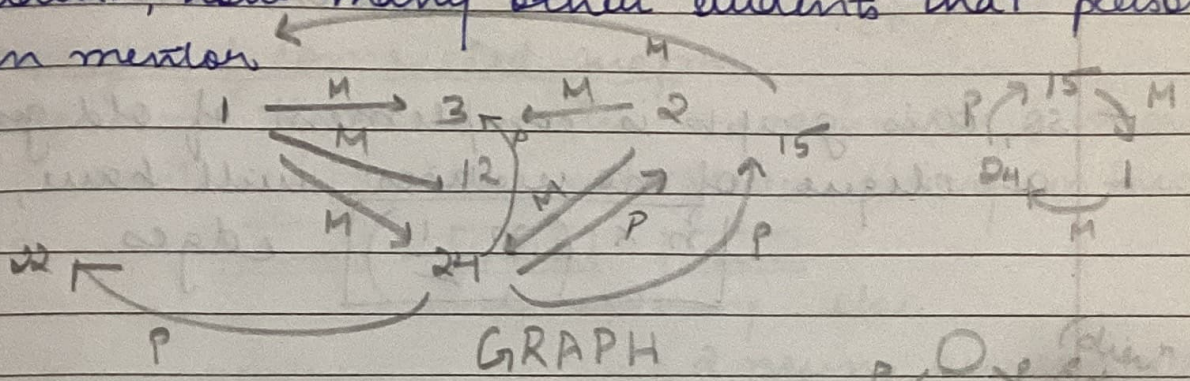
- compare 2 lists for duplicate items → nested loop
- what if lists are sorted out? use first() and rest() to cut down list to be scanned
- now, 2nd list has been modified inside the procedure → side-effect
- for this, we can copy the main list to my own copy and change it accordingly
- in some cases, side effects are intended, like in places to update the persn. collection

Week-7

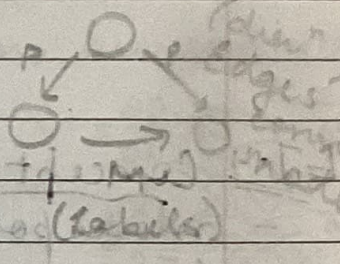
CLASSTIME | Pg. No.
Date / /

L7.1 Intro. to Graph Data Struc.

- A —S→ B (A: student, helps; B: student in subj. S)
  A's Mark in S − B's Mark in S btw. [10, 20]
- all pairs, hence, nested iteran
  11 —M→ [7, 16, 6, 27, 21, 4] and so on find for each student, how many other students that person can mention



- cycles of length 2/3
  with list of adjacents / neighbours



L7.2 Intro. to matrices & implementan of matrix

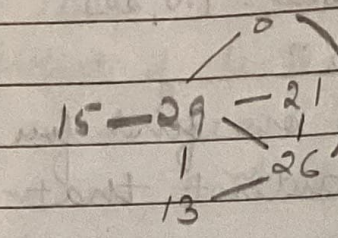
- list keeps a seq. of values, no random access, for value at i, scan (i−1) elements
- dictionary stores key−value pairs, supports random access
- Matrix — 2d table, m rows, n columns
  ↳ rows, columns are numbered from 0
  ↳ $0 \le i \le m-1$, $0 \le j \le n-1$ ⇒ matrix[i][j]
- outer key → rows, inner key → columns
  ↳ creates a nested dictionary
- create → mymat = Create matrix (30, 45)
- sort (key(d)) — ascending order

```
foreach r in sort(keys(matrix)) {
    " c " sort(keys(matrix[0])) {
        do stuff
}}
```

L7.3 Undirected graphs & cliques

- similar [0,9] → thus, undirect, only simile

Eg. 22 — 13
↳ undirected edge



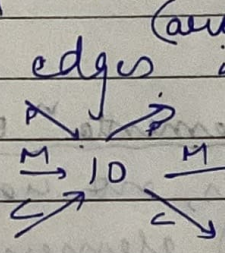
Clique → congregate together and exclude other people

- so, this graph is complement of old graph
- a clique of n vertices will have,

$$\left[n \times \left(\frac{n-1}{2} \right) \right] \text{ edges}$$

L7.4 Concept of popular students using graph

- more popular, in middle, they get help from many people and they'll help many people
- no. of edges (around) in nodes is degree — in degree, out degree



and so on
complete the list

L7.5 Pseudocodes for real-time eqs using graphs

- A can mentor B, diff. btw. 10 - 20 marks
- create a dictionary for marks in subj.
- create a mentoring graph, represent as a matrix

M[i][j] = 1 - edge from i → j
M[i][j] = 0 - no edge

- dictionary helps to keep the list of things explicitly w/o any repetition
- iterate them matrix to aggregate info. from the graph

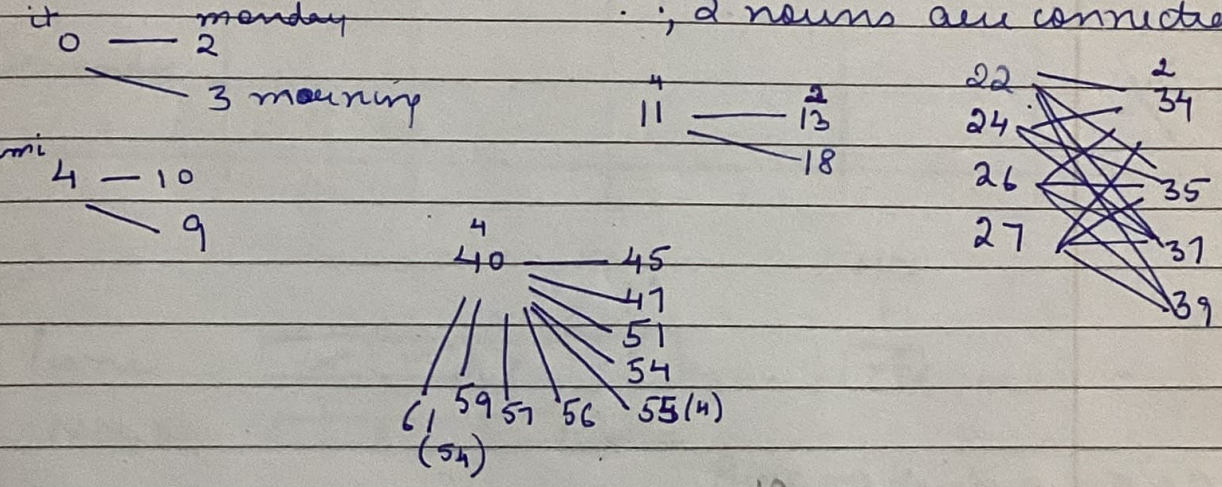
L7.6 Concept of connected graph

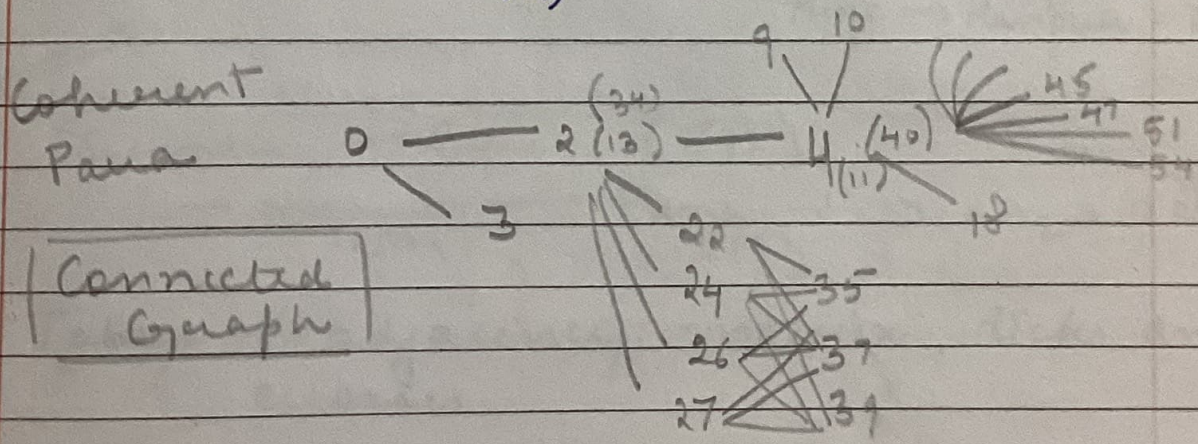
- Rule → 2 noun with a verb btw., in same sentence, which not other thing in btw.

Noun | Verb | Noun

and nothing else

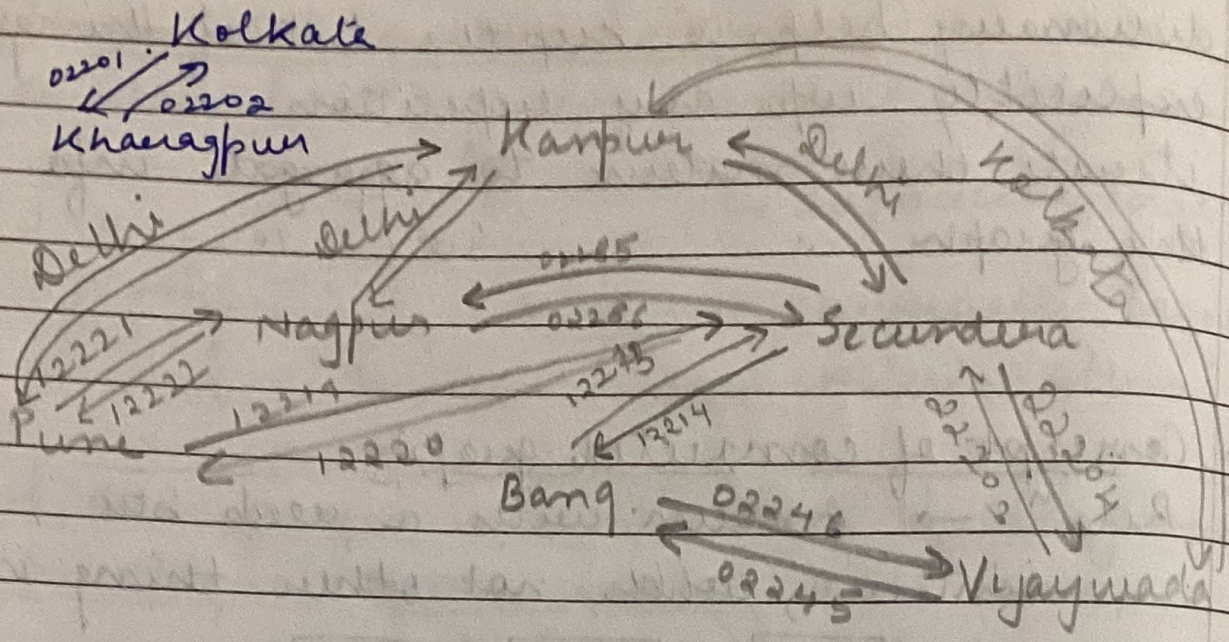
∴, 2 nouns are connected





L7.7 Represent direct trains using graph

- Rule btw. stations with a edge

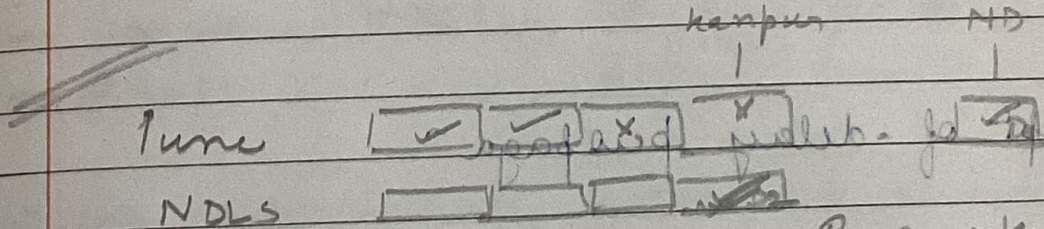

Kolkata
02201
02202
Khanagpur
Kanpur
Delhi
Delhi
Kolkata
Delhi
02165
02166
12221
12222
Nagpur
Secunderabad
12213
12214
12220
12215
12214
Pune
Bang
02246
02245
Vijayawada

Week-8

CLASSTIME | Pg. No.
Date / /

L8.1 Represent graph as adjacency matrix

To → / From ↓	K	Na	NDLS	P	S
Kolkata	X	10222 — 1131 km 9962 — 113	12273 — 1529 (21) 12263 — 1520	existed	—
Nagpur		X			09286 — 577
NDLS			X		
Pune			12263 — 1520	X	
Secunder.					X



Pune → Kanpur = $D_1 + D_2$

Secunder. → Pune (597)

Nagpur (579) → Pune (889) (1520)

- Tables to adjacency matrix, ticks and crosses

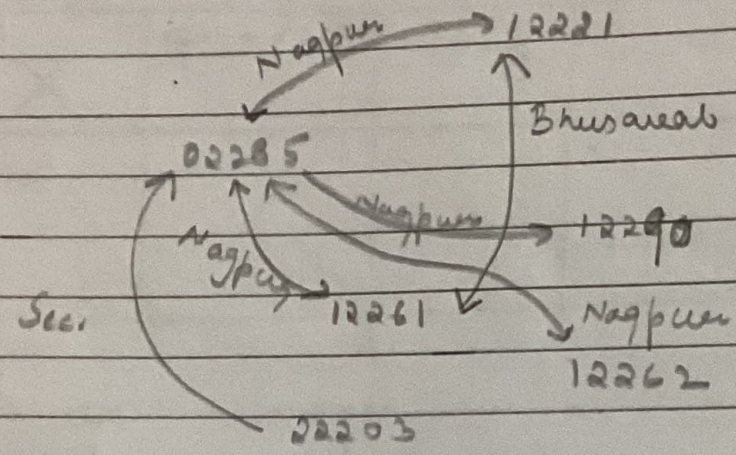
	Sec	Vij	NDLS	Kan	Kol
B	688 D.	738	1686 1667 ✓		
S				439 440	
ND				1014 1090	
Kol					1218
Vij.					

CLASSTIME | Pg. No.
Date / /

B → V → Kol → Kan = 728 + 1218 + 1014 = 2960
B → Sec → ND → Kan = 688 + 1667 + 439 = 2794

- Matrix multiplicⁿ — adds values

L8.2 Constrⁿ of graph where trains are nodes and staⁿˢ are edges



- concept of delay propagaⁿ

L8.3 Pseudocode for finding route btw 2 staⁿˢ

- compute pair of staⁿˢ connected by a direct train

```
train[t][start], train[t][end]
stations {}
foreach t in keys(trains) {
    stations[[trains[t][start]] = True
                                = True
}
n = length(keys(stations))
direct = create matrix (n,n)
```

- 1 hop route - stations A & B such that u can reach B from A by changing 1 train
- we can modify to direct to 1 hop (at most)
- 2 hops - $i \longrightarrow k \longrightarrow j$ (via 0)
- we can extend 'nhops' to allow 1 more hop
- Path - seq. of edges from A → B. A direct edge is a path of length 1
- N nodes — shortest paths from A → B has at most (N − 1) edges
- Transitive closure - pair of nodes connected by a path

CLASSTIME Pg. No.
Date / /

L8.4 Pseudocode for edge labelled graph

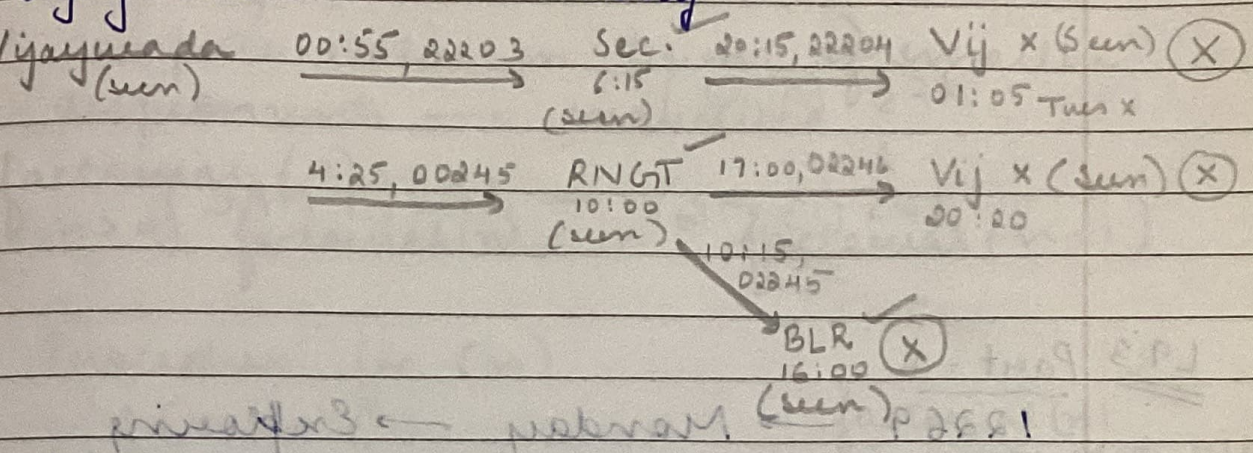
- keep a track connecting stations
  - each entry in the matrix is now a dictionary
  - initial empty - no direct connection
  - add a key for each train
- for each direct train record the distance it travels trains[t][distance]
- compute the shortest distance
- graph directlist[i][j] = 1 (is the shortest distance)
  = 0 (no direct train

Week - 9

CLASSTIME Pg. No.
Date / /

L9.1 DFS (Depth First search) & recursive procedure call — Part 1

Vijayawada → Monday



- depth first search - goes deeper and deeper
  - u have starting time → sta^n

explore (Vijayawada, MON, 00:00) | Vij
explore(Sec, MON, 6:15) ex (RNGT, MON, 10:00) | Sec, RNG

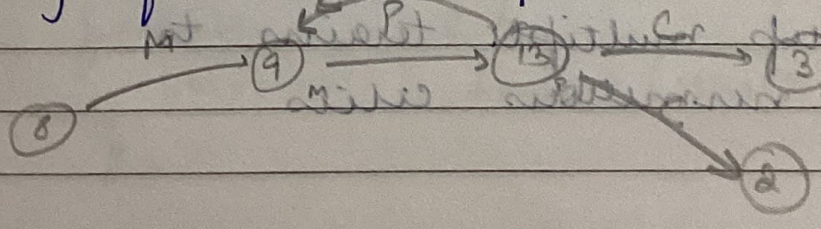
Seen — recursive procedure it calls itself

→ recursion → procedures call itself

→ DFS creates a tree with edges of graph, gives the order of journey
- keeping track of seen

L9.2 Part - 2

- mentioning graph
- arrange for each subj. in decreasing order



- finding the graph as we go along

- last 2 nodes.

dFs (8, prev) M

↙

dFs (9, M)

↓

dFs (13, MP) → dFs (9, MPC)

L9.3 Part - 3

12269 → Monday → Exploring connecting trains

Ex (12269, M, prev)

vij ↓

Ex (02246, M, 22:31)

↓ Bhub.

Ex (02202, Tu, 20:15)

vij ↓

Ex (22004, Tu, 1:45)

- spanning tree → reaches all the stns, the graph is included in this

L9.4 Part - 4

- 24 hrs → Explore (Bhuvanal) Visited cities
- alternate btw. trains & stns and explore the cities & also pick up multiple trains to reach numerous cities

Visited Trains

L9.5 Pseudocode for recursion

- many computations — inductively
  - directly return the value → base
  - compute value in terms of smaller arguments — inductive

factorial (0) = 1

n > 0, factorial(n) = n * factorial(n-1)

```
Procedure fac (n)
  if (n == 0) {
    return(1)
  }
  else {
    return (n * fac(n-1))
  }
End
```

factorial (n) is suspended till factorial (n-1) returns the value.

- base → empty list

inductive → compute value in terms first element & rest

If l == [], sum is 0

or first(l) + rest(l)

or last(l) + init(l)

```
if (l == []) {
  return(0)
}
else
  return (first(l) + ListSum(rest(l)))
}
```

- build up a sorted prefix, extend it by inserting the next element in the correct pos^n

```
sorted l = []
  foreach z in l {
    sortlist = Insert (sorted list, z)
  }
```

- insert^n sort, inductively ⇒ list of length 1 or less is sort, for longer, insert 1st(l) into rest (l)

```
if (length(L) <= 1) {
    return(l)
}
else {
    return (Insert (Insertsort (rest(l)))
}
```

- w/o properly defined base cases, recursive procedures will never terminate ⇒ suspend the current computation till the recursion computation terminates

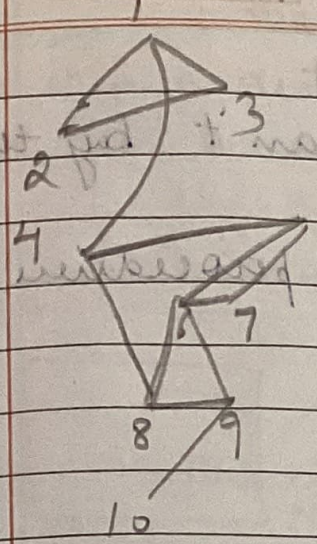
Lg.6 Pseudocode for DFS & recursive procedure

- what are vertices from node i?
- start from i, visit a neighbor j. ⇒ bfs
- dfs → i → j → k → [- - . and so on
- maintain info about visited nodes in a dictionary visited → not to explore again

```
visited = {}
visited = DFS(graph, visited, i)
keys(visited) is set of nodes that can be
    reached from i

Procedure DFS(graph, visited, i)
visited[i] = True
foreach j in columns(graph) {
    if (graph[i][j] == 1 & not(j key(visited))
        visited = DFS(graph, visited, j)
    }
}
return(visited)
End
```

⇒ keys(visited) includes all nodes, the graph is connected



visited = { 4 : True, 1 : True,
2 : True, 3 : True, 5 : True, 6 : True
7 : True, 8 : True, 9 : True,
10 : True }

DFS(graph, visited, 4)
DFS(" , " , 1)
DFS(" , " , 2)
DFS(" , " , 3)
DFS(" , " , 5)
DFS(" , " , 6)
DFS(" , " , 7)
DFS(" , " , 8)
DFS(" , " , 9)
DFS(" , " , 10)

Week-10

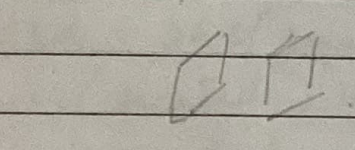
L10.1 Concept of Encapsulation & Object

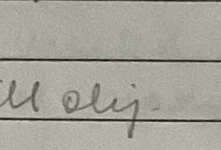
- Manual comes with TV, u can't buy the manual seperately
- other aspects of organising procedures
- all the cards are inside the box → object
- Capsule consist all the data and procedures in the box → encapsulation
- it relates of OOP (object-oriented programming)
- object → some stores of things and do some operations on it.

Score dataset - obj. is whole stacks

Shopping Bill" → 1 shop is 1 obj.

- 1 customer be 1 obj.





L10.2 Concept of class & its instances c/d obj.

- procedures are inside the obj.

train → trains → 1 obj.

data set → stations → 1 obj.

- depending on what is the ques" we have to choose obj wisely
- 1 card of train dataset is a template of generating trains

CLASSTIME Pg. No.
Date / /

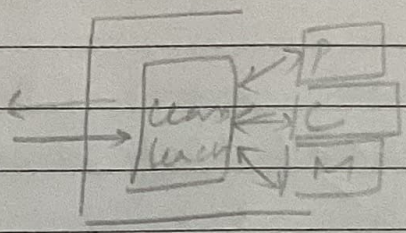
→ 1 train card in train dataset

- it is a design of an obj. → class → not a obj. itself but it represents an obj.

12262 → class
new train; Wednesday → Train
Clock → obj
3:00pm, Wed
Tatanagar - Bilaspur
Where New?

L10.3 Concept of abstraction

- from where, these obj. comes from?
- we create obj. that obj. creates other obj. if needed.



L10.4 Applica" of class, obj, encapsula" & abstrac" - Part 1

Jhansi ← Kanpur (same day) Tuesday

?

12214 (Mon)
12269 (Mon)
10:22 12270 (Tues) DLI
12260 (Mon)
12273 (Mon)
12259 (Mon) 06:30
12274 (Tues)

CLASSTIME | Pg. No.
Date / /

L10.5 Part-2

SEEN STN
KGP 01:15
KOL 4:15
TATA 11:35
BLSP 17:35

Explicit STN
KGP

SEEN TRAIN
02202 (Tw)
12262 (W)
12261 (W)
12261 (Thu)
12298 (Thu)

Explicit TRAIN
02202 (Tw)
12262

ExS (KGP, W, 01:10)

ExT (02202, Tw)

ExS (KOL, W, 04:15)

ExT (12262, W)

ExS (TATA, W, 11:35) ✗

ExS (BLSP, W, 17:36) ✗

L10.6 Decompos^n of study grp problem
- 4 obj. - CT, PhyT, MathT, ChemT
- decomposing the scorecard into 4 obj.

L10.7 Formalized notn and summary of encapsuln

- It seems to be more natural to attach procedures to the data element
- Encapsulate the datas elements & procedures into 1 package, which is clld an obj., i.e., oop
- The procedures encapsulated in the obj. act as an interfaces thru which the ext. world interacts with obj.
- The obj. hides details of implementn from ext. worlds, seperates "what" from "how"
- procedures was also doing the same things, taking parameters & returns values, hiding was also there, but procedures have undesirable side-effects, and we can't store the value
- datatypes —
  - basic — int, char, boolean
  - subtype of datatype restricts values & opern
  - compound — record, list, string
  - collcn of cards is also a datatype

Procedures

- Avg marks takes a field as parameter
- Eg. AvgTotal = Avgmarks (total)
  AvgMaths = Avgmarks (Maths)

Encapsuln

- datatype ClassAvg has a procedure avg() within it.
- We call avg() for each obj. of Class Avg datatype
- Eg. AvgTotal = CT.Avg()
  AvgMaths = Mat.Avg()

- if the dataset is not static, means a students gets added to the class

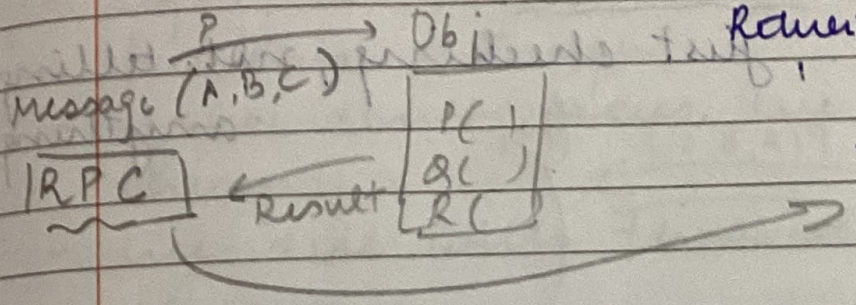
- Class Ave should be able to restrict access to its fields, done by private / public field
- encapsulation provides ↑ use of modularisation when compared to procedures

Week - 11

CLASSTIME | Pg. No.
Date / /

L 11.1 Concept of msg. passing using Remote Procedure Call (RPC)

- we wait to get the ans. — u r stuck
- we're telling the obj., the procedure is inside the object
- counter from two-way communica"
- flight / hotel → tray thing, u put your thing and then u get it later
- the sub-contractor can be in a diff. city
- Remote Procedure Call $X = P(A, B, C)$

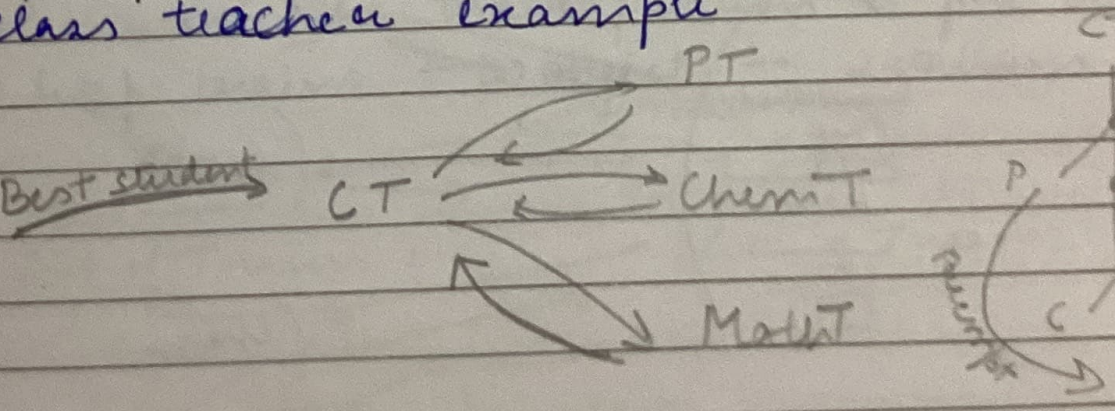


Eg. Ordering food from a restaurant, Cache memory

- phenomenal way of abstrac"

L 11.2 Concept of concurrent execu" using poling and preemp"

- class teacher example

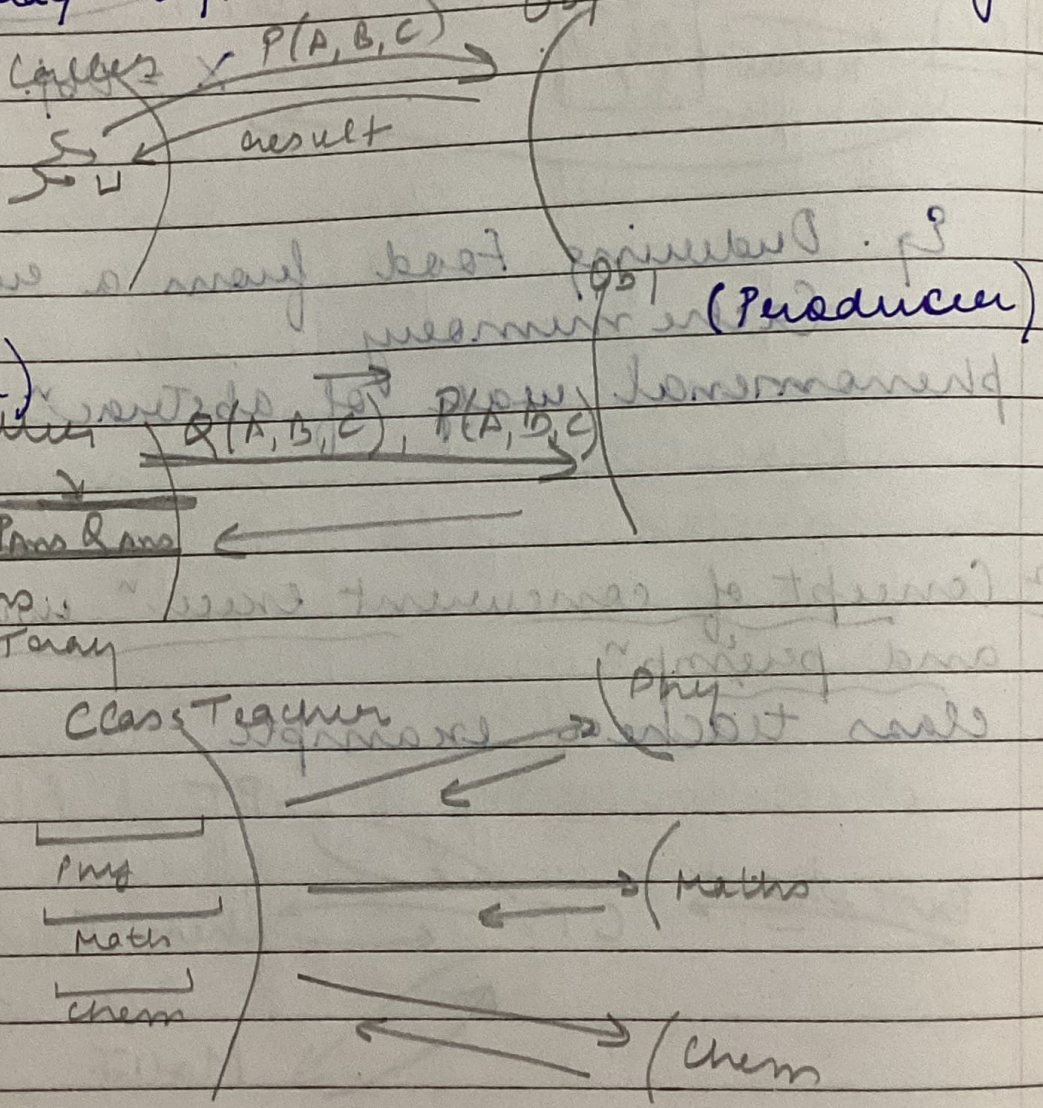


- things are going in kind of parallel

- concurrent → diff. things happening at the same time
  - ↳ multiple parts are active and each thing is doing sth. else
- preemption - interrupting sth. more impt. first, kind of pausing sth. and do sth. else
- multiple threads → first, kind of pausing thing started, and simultaneously u started sth. else

L11.3 Prod.-Consumer Problem

- A → B
  Obj ← Array Obj. , just checking, not telling anything

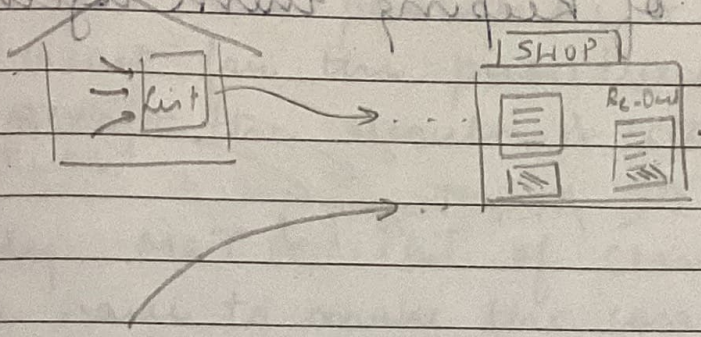


- atomic - reading & writing should happen simultaneously (no concurrency)

CLASSTIME Pg. No.
Date / /

LI1.4 Real time applicaⁿˢ of concurrency

- shopping bill data — customers are not under each on shops

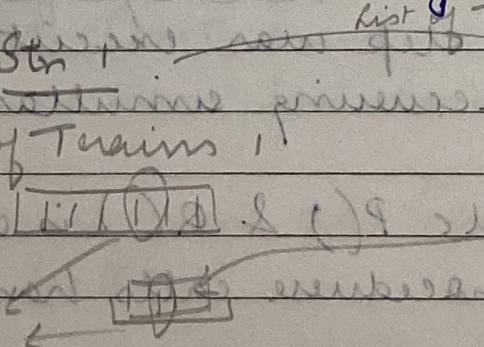


Concurrency like anything

- no. of possibilities increases
- seq. of concurrent things can be arranged but this is tough
- if we don't focus, we might loose up things
- we deal with these things in real-life always

LI1.5 Applicaⁿˢ of concurrency to speedup the execⁿ

- Train dataset
- all trains are running
- at any time, multiple trains are running
- Due to a staⁿ, how many trains you have.
- resources like track / station is pre-occupied
- 1 hop trains → 2 object → 2 stations



L11.6 Concept of Msg. Broadcasting

- Send a msg. to everyone together
- broadcasting the question — to everyone
- agreement of keeping them together

Broadcast

Atomic Read ← Ans. Today

Atomic write

L11.7 Concurrency involved in input/output operations

- browser → search query → submit ans.
- I/O (Input - Output) - interaction
- Forms (write sth.) - with restriction
  → Submit button (with sanity)
- And result is shown, then computer gives the output
- This I/O work is also concurrency with a atomic model.
- Eg. Chatting also

L11.8 Formalized Notation & Summary of Concurrency

- till now, only 1 step was executed at a time
- multiple lines occuring simultaneously → concurrency
- 2 procedure calls P() & Q() is concurrent
- Concurrency → procedure calls has to be unbundled
- The step B = P(A) has many actions

- For obj. X & Y : X.P(A) & Y.P(A) may be executed concurrently ⟹ related to OOP
- B = X.P(A) has actions —
  1. start the procedure & pass parameters
  2. wait for the procedure to comp: -
  3. Accept the result & store the result

Eg: 2 obj. MaT & PhT of class Aee datatype (we have to make the concurrent)
- we need start, ready & result (average)

MaT.start(average)
PhT.start(average)
wait(MaT.ready(average) & PhT.ready(avg))
Aee Maths = MaT.result(average)
Aee Physics = PhT.result(average)

-
- if, now want to add new students to both MaT & PhT

MaT.start(news, new M1)
PhT.start(news, new M2)

⇒ MaT.start(average)
MaT.start(adds, new M1)
wait(MaT.ready(average) & MaT.ready(adds))
Aee Maths = MaT.result(average)

↳ we can add aee ready & add ready fields

- to prevent corrupt^n of the list, we have to make sure that we do only 1 list op^n at a time. We say that list field is atomic.
- for different coordinat^n, the obj. MAT should be shared.
- Polling - The caller started the procedure and then just waited for them to finish
- Pre-emption - It interfere your work & stop it
- Prod.-Consumer Model - I keep the things in tray (buffer) & consumer will take it from there.
- this prevents unwanted race conditions btw. read & write of the buffer

Week-12

CLASSTIME Pg. No.
Date / /

L12.1 Tutorial -1 (Pseudocodes)

Eg. Find the no. of students based on total marks

Eg. Find the no. of students based on marks & city / town

⇒ Nested iteration

Eg. Find the no. of students above avg. marks

⇒ multiple non-nested iteration

↳ 2nd iteration ans. is dependant on the ans of 1st iteration

Eg. Find the no. of pair of students who score same marks in maths

Eg. Find subj. topper of 3 subjects

⇒ same code for 3 subj. → (increment) into a parameterised function subj. as parameter

CLASSTIME Pg. No.
Date / /

- Advantages of procedure –
(i) avoid repetition of code
(ii) divide complex problem into smaller one
(iii) makes it easy to read the code
(iv) modify of procedure becomes easier

- return
terminates the procedure & flow of code goes back to the procedure call

exitloop
terminates the only loop from which the statement has been called

```
A = is even(num)
Procedure is Even(n)
  if (n % 2 == 0) {
      return (True)
  }
End is even
```

```
i = 1, j = 1, n ← 4, sum = 0
while (i < n) {
    if (i == j) {
        exitloop
    }
    sum = sum + j
    j = j + 1
}
i = i + 1
}
```

L12.2 Tutorial – 2 (Lists & Dictionaries)

- list is collec^n of values in seq. order, can store values of diff. datatypes, duplicates allowed, it preserves the order of element
[] notation
- l = [] or length(l) == 0, denotes l is empty
- list = list ++ list
↳ append & extend

- append → merges 2 lists
  $l_1 = [1]$ $l_2 = ["a"]$ $l_1 ++ l_2 = [1, "a"]$
- extend → adds element to list
  $l_1 = [1,2,3]$ $l_1 ++ [x] = [1,2,3,4]$
  $x = 4$
  input

- [] → length = 0
- [1,2,3,4] → length(l) = 4
  rest(l) = [2,3,4] init(l) = [1,2,3]

returns/output types:
- length(l) ⇒ always integer
- rest(l) = init(l) ⇒ always list
- first(l) = rest(l) ⇒ depends on element

- Insertion Sort
  input list = [5,2,1,3,4], sorted list = []
  foreach x in input list {
    sorted list = insert an element (sl, x)
  }
  Procedure

- Dictionary - it is a collection of element stored as key: value pair.
- It can store duplicate values, but keys are unique, no order preserved.

- {} notation, value is accessed by d[key] notation
- keys(d) → returns a list of keys of dictionary

d = {"a": 1, "b": 2}
keys(d) = ["a", "b"]

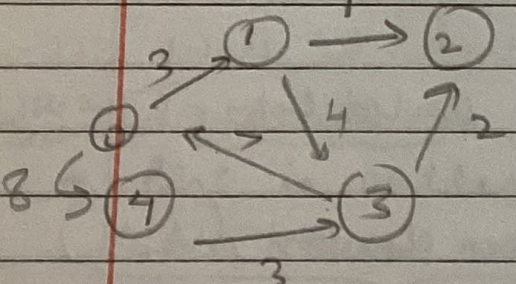
- iskey(d, k) → True if k is a key of a dictionary

iskey(d, "a") → return (True)

Index	Key
- List has indices	- Dic has keys
- it indicates the pos^n of element	- unique entity to identify the key
- it always integer	- any datatype
l = [10, 20, 30]	d = {1: 10, 2: 20}
index l(0) = 10	d[2] = 20

L12.3 Tutorial -3 (Graph & Adjacency Matrix)

- edge - labelled directed graph



	0	1	2	3	4
0	0	3	0	0	8
1	0	0	1	4	0
2	0	0	0	0	0
3	7	0	2	0	0
4	0	0	0	3	0

matrix = { 0: {0:0, 1:3, 2:0, 3:0, 4:8},
1: {0:0, 1:0, 2:1, 3:4, 4:0},
2: ... Similarly
3: ... for other two
4: ... }

matrix[0][1] == 3
" [0][4] == 8 } list of edges

→ Procedure it matrix (rows, col) → 2-D -
↳ produces a matrix of row × col

```
matrix = []
i = 0
while (i < rows) {
    matrix[i] = []
    j = 0
    while (j < cols) {
        matrix[i][j] = 0
        j = j + 1
    }
    i = i + 1
}
return (matrix
```

L12.4 Tutorial - 4 (Recursion & DFS)

- Recursion - is a process in which a procedure calls itself. It simplifies the pseudo code.
- Problems with base problem only can be changed to recursive call.

Eg. To find the factorial of number

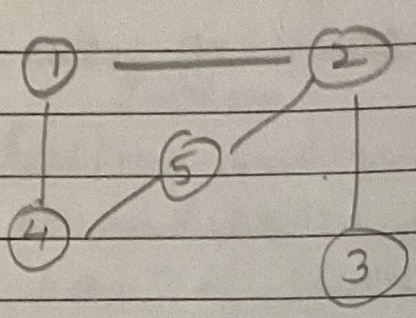
CLASSTIME Pg. No.
Date / /

```
Procedure fac(n)
    fact = 1
while (n > 0) {
    fact = fact * n
    n = n - 1
}
return (fact)
```

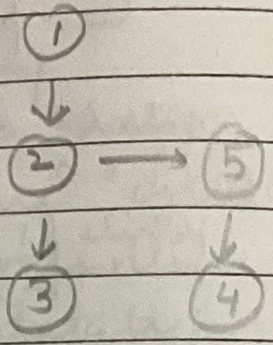
→

```
if (n == 0) {
return n (1)
}
else {
return (n * fac(n-
}
```

- DFS → recursive algorithm



Input: Graph



Output: Tree

L12.5 Final Summary

- Express problem solutions as seq. of steps
- find common patterns btw solu^n
- data sc. problems → on dataset
- first scanning to collect relevant info.
- processing the info.
- finally, organising the relationships in a form
- Iterator - that scans the dataset (most powerful construct)
- Variable - storage units
- Assignment statements & accumulators
- Pseudocodes & Flowcharts

- basic datatypes - boolean, int, char, string
- subtypes for more constraint
- List → items are of same datatype
- Dictionary → Record → multiple datatype
  ↳ we can keeping adding fields fields
- concept of filtering
- procedure - to seperate out the code
- parameter - vars. used for procedure
- this makes the code compact & easy to manage
- sequential itera^n - to find above avg. students
- nested itera^n - binning to end. time
  ↳ and store in list, dictionary, graph
- foreach iterates them values in a list
- insertion sort - is a natural sorting algorith

```
foreach k in keys(d) {
    Do sth with d[k]
}
```

Dictionary provides random access

- graphs represent relationships
- matrices are 2-d tables → m[i][j]
- Recursion - base case & inductive step
  ↳ suspend the current computa^n till the recursive computa^n terminates
- Encapsula^n - packs procedures with data elements
  ↳ provides more modularisa^n
  ↳ disadj. → obj. need to created
- concurrency → allows several tasks to be executed simultaneously
  - polling
  - producer-consumer model

Atomic — Input/Output type