CT WEEK 7 : Dictionary and Table

Lecture 1 : Examples to introduce dictionary

Week-7 (L.7.1) (Examples to Introduce Dictionary)

* Dictionary is a Collection where you have Values but for each Value there is a way to get to that Value so it's like a key.

(key : Value) pair

- { 2 : True }
  - Seen
  - key (Index)
  - Value

Math Pairs
[16, 4], [16, 25]

List → (Seq of Value)

10 True

- 16 : True

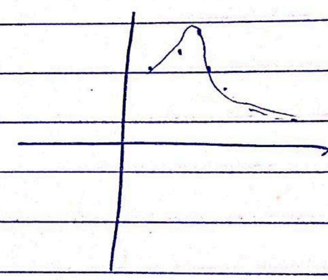
- Use of dictionary
  - # Nested Iteration, when we want to track more than 2 card.

- Key - Unique.
- Value can be anything.

∘
letter count

2 : ///
3 : /////

10 : //



QN :

Lecture 2 : Concept of dictionary to solve birthday paradox problem

L.7.2. (Concept of 'Dictionary' to solve Birthday Paradox Problem)

* 2 students have same birthday or not. ?

Check Using Dictionary → # Date as key - (Day / Month)

- Check how many have same birthday.

Jan – Jun	July – Dec
	17 Jul - 22
	23 Jul - 14
	22 Sep - 15
5 May - 3	7 Nov - 0
	12 Nov - 4
3 Jun - 1	

↳ One Item with 2 element is what we are looking for.

Lecture 3 : Pseudocode for dictionaries

Pseudocode: Introducing dictionaries

Indexed collections

- A list keeps a sequence of values
- Can iterate through a list, but random access is not possible
  - To get the value at position i, need to start at the beginning and walk down $i - 1$ steps
- A dictionary stores key-value pairs. For instance
  - Chemistry marks (value) for each student (key)
  - Source station (value) of a train route (key)
- Present the key to extract the value — takes the same time for all keys, random access
  - `m = chemMarks["Rahul"]`
  - `s = sourceStation["10215"]`

Pseudocode for dictionaries

- At a "raw" level, sequence of `key:value` pairs within braces
  - `{"Rahul":92, "Clarence":73, "Ritika":89}`
- Empty dictionary, `{}`
- Access value by providing key within square brackets
  - `s = sourceStation["10215"]`
- Assigning a value — replace value or create new key-value pair
  - `chemMarks["Rahul"] = 92`
- Dictionary must exist to create new entry
  - Initialize as `d = {}`

Example

Collect Chemistry marks in a dictionary

```
chemMarks = {}
while (Table 1 has more rows) {
  Read the first row X in Table 1
    name = X.Name
    marks = X.ChemistryMarks
    chemMarks[name] = marks
  }
  Move X to Table 2
}
```

Processing dictionaries

- How do we iterate through a dictionary?
- `keys(d)` is the list of keys of `d`

  ```
  foreach k in keys(d) {
     Do something with d[k]
  }
  ```
- Example
  - Compute average marks in Chemistry

```
total = 0
count = 0
foreach k in keys(chemMarks) {
   total = total + chemMarks[k]
   count = count + 1
}
chemavg = total/count
```

Checking for a key

- Typical use of a dictionary is to accumulate values
  - `runs["Kohli"]`, runs scored by Virat Kohli
- Process a data set with runs from different matches
- Each time we see an entry fo Kohli, update his score
  - `runs["Kohli"] = runs["Kohli"] + score`
- What about the first score for Kohli?
  - Create a new key and assign score
  - `runs["Kohli"] = score`
- How do we know whether to create a fresh key or update an existing key?
- `isKey(d,k)` — returns `True` if `k` is a key in `d`, `False` otherwise
- Typical usage

```
if isKey(runs,"Kohli"){
  runs["Kohli"] = runs["Kohli"]
                    + score
}
else{
  runs["Kohli"] = score
}
```

- Implementing `isKeys(d,k)`
  - Iterate through `keys(d)` searching for the key `k`
- Takes time proportional to size of the dictionary
- Instead, assume `isKeys(d,k)` is given to us, works in constant time
  - Random access

```
Procedure isKey(D,k)
    found = False
    foreach key in keys(D) {
      if (key == k) {
        found = True
        exitloop
      }
    }
    return(found)
End isKey
```

Summary

- A dictionary stores a collection of key:value pairs
- Random access — getting the value for any key takes constant time
- Dictionary is sequence `{k1:v1, k2:v2, ..., kn:vn}`
- Usually, create an empty dictionary and add key-value pairs

```
d = {}
d[k1] = v1
d[k7] = v7
```

- Iterate through a dictionary using `keys(d)`

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

- `isKey(d,k)` reports whether `k` is a key in `d`

```
if isKey(d,k){
  d[k] = d[k] + v
}
else{
  d[k] = v
}
```

QN : 4

Lecture 4 : Relations among customers based on their spending patterns (Part 1)

L.7.4 (Relations among Customers based on their Spending Patterns (Part 1))

*. Customer Similarity :- Similar purchasing Patterns,
↳ Category.

Customer	Category1	C-2	C-3
Cus-1	2	0	3

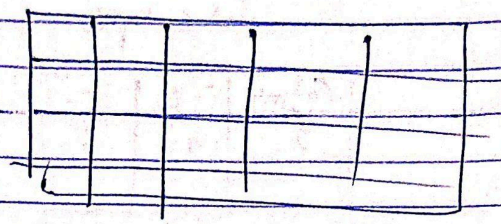
- People who purchased one Item.
- ——— Few Item.
- ——— More Item

QN : 3,4

Lecture 5 : Relations among customers based on their spending patterns (Part 2)

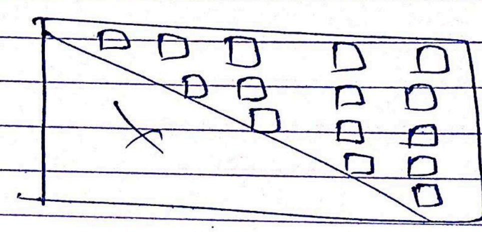
3① L.7.5 (Relation Among Customers based on their spending Patterns (Part-2))

- Distance — Difference column wise



1.2	2.3		
1.3	2.4	3,4	
1,4	2.5	3,5	4,5
1.5	2.6	3,6	4,6
1.6			5,6

↳ Subtract each Items No and add to get distance



32

1 2 3

1.2,3

12 23

13

~~11~~ 21 31

12 ~~22~~ ~~32~~

13 23 33

Lecture 6 : Introduction to dictionary data structure

1.7.6 :- (Introduction to dictionary data Structure)

Food sold for each Shop ~

CV = 0 → 8 → 14 → 17

SVN = 0 → 1

B14 = 0

- Exists → Update
- Not exist → Create and assigns

Dictionary

ABC → 0

ACB → 1

222 → 100

QN : 2,3,4

Lecture 7 : Pseudocodes for real-time examples using dictionaries

Pseudocode: Dictionaries, Examples

Customers buying food items

- Find the customer who buying the highest amount of food items
- Create a dictionary to store food purchases
  - Customer names as keys
  - Number of food items purchased as values

```
foodD = {}
while (Table 1 has more rows) {
    Read the first row X in Table 1
    customer = X.CustomerName
    items = X.Items
    foreach row in items {
      if (row.Category == "Food") {
        if (isKey(foodD,customer)) {
          foodD[customer]
                  = foodD[customer] + 1
        }
        else {
          foodD[customer] = 1
        }
      }
    }
    Move X to Table 2
}
```

Birthday paradox

- Find a birthday shared by more than one student
- Create a dictionary with dates of births as keys
- Record duplicates in a separate dictionary

```
birthdays = {}
duplicates = {}
while (Table 1 has more rows) {
    Read the first row X in Table 1
    dob = X.DoB
    if (isKey(birthdays,dob)) {
      duplicates[dob] = True
    }
    else {
      birthdays[dob] = True
    }
    Move X to Table 2
}
```

- Find a birthday shared by more than one student
- Create a dictionary with dates of births as keys
- Record duplicates in a separate dictionary
- If we want to record the names of those who share the birthday, store a list of student ids against each date of birth

```
birthdays = {}
duplicates = {}
while (Table 1 has more rows) {
    Read the first row X in Table 1
    dob = X.DoB
    seqno = X.SeqNo
    if (isKey(birthdays,dob)) {
      duplicates[dob] = True
      birthdays[dob] =
          birthdays[dob] ++ [seqno]
    }
    else {
      birthdays[dob] = [seqno]
    }
    Move X to Table 2
}
```

- Find a birthday shared by more than one student
- Create a dictionary with dates of births as keys
- Record duplicates in a separate dictionary
- If we want to record the names of those who share the birthday, store a list of student ids against each date of birth
- Can also store the students associated with each date of birth as a dictionary

```
birthdays = {}
duplicates = {}
while (Table 1 has more rows) {
   Read the first row X in Table 1
   dob = X.DoB
   seqno = X.SeqNo
   if (isKey(birthdays,dob)) {
     duplicates[dob] = True
     birthdays[dob][seqno] = True
   }
   else {
     birthdays[dob] = {}
     birthdays[dob][seqno] = True
   }
   Move X to Table 2
}
```

Resolving pronouns

- Resolve each pronoun to matching noun
  - Nearest noun preceding the pronoun
- Create a dictionary with part of speech as keys, sorted list of card numbers as values.

```
partOfSpeech = {}
partOfSpeech['Noun'] = []
partOfSpeech['Pronoun'] = []

while (Table 1 has more rows) {
    Read the first row X in Table 1
    if (X.PartOfSpeech == 'Noun') [
      partOfSpeech['Noun'] =
          partOfSpeech['Noun']
          ++ [X.SerialNo]
    }
    if (X.PartOfSpeech == 'Pronoun') [
      partOfSpeech['Pronoun'] =
          partOfSpeech['Pronoun']
          ++ [X.SerialNo]
    }
    Move X to Table 2
}
```

- Resolve each pronoun to matching noun
  - Nearest noun preceding the pronoun
- Create a dictionary with part of speech as keys, sorted list of card numbers as values.
- Iterate through the dictionary to match pronouns
- Note that `partOfSpeech['Noun']` and `partOfSpeech['Pronoun']` are both sorted in ascending order of `SerialNo`

```
matchD = {}
foreach p in partOfSpeech['Pronoun'] {
    matched = -1
    foreach n in partOfSpeech['Noun'] {
      if (n < p) {
        matched = n
      }
      else {
        exitloop
      }
    }
    matchD[p] = matched
}
```

QN : 2,3

PA : 2,3,4,5,7,8,9,10
GA : 4,7,10