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}`

Pseudocode for dictionaries

- At a "raw" level, sequence of `key:value` pairs within braces
  - {`"Rahul":92, "Clarence":73, "Ritika":89`}
- Empty dictionary, {}

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"]`

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`

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 = {}`

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?

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]
}
```

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

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`

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`

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?

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

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
}
```

Checking for a key

- Implementing `isKeys(d,k)`
  - Iterate through `keys(d)` searching for the key `k`

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

Checking for a key

- Implementing `isKeys(d,k)`
  - Iterate through `keys(d)` searching for the key `k`
- Takes time proportional to size of the dictionary

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

Checking for a key

- 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
}
```