IIT Madras

ONLINE DEGREE

Pseudocode: List example, correlating student performance

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

- Assign grades {A,B,C,D} in both subjects
  - “Perform well in Maths” — grade B or above
  - “Perform at least as well in Physics” — Physics grade $\geq$ Maths grade

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

- Assign grades {A,B,C,D} in both subjects
  - "Perform well in Maths" — grade B or above
  - "Perform at least as well in Physics" — Physics grade $\geq$ Maths grade

- Algorithm

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

- Assign grades {A,B,C,D} in both subjects
  - “Perform well in Maths” — grade B or above
  - “Perform at least as well in Physics” — Physics grade $\geq$ Maths grade
- Algorithm
  - Assign grades in each subject

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

- Assign grades {A,B,C,D} in both subjects
  - "Perform well in Maths" — grade B or above
  - "Perform at least as well in Physics" — Physics grade $\geq$ Maths grade
- Algorithm
  - Assign grades in each subject
  - Construct lists of students with grades A and B in both subjects — four lists

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

- Assign grades $\{$A,B,C,D$\}$ in both subjects
  - “Perform well in Maths” — grade B or above
  - “Perform at least as well in Physics” — Physics grade $\geq$ Maths grade

- Algorithm
  - Assign grades in each subject
  - Construct lists of students with grades A and B in both subjects — four lists
  - Count students in A list for Maths who are also in A list for Physics

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

- Assign grades {A,B,C,D} in both subjects
  - “Perform well in Maths” — grade B or above
  - “Perform at least as well in Physics” — Physics grade $\geq$ Maths grade

- Algorithm
  - Assign grades in each subject
  - Construct lists of students with grades A and B in both subjects — four lists
  - Count students in A list for Maths who are also in A list for Physics
  - Count students in B list for Maths who are also in A list or B list for Physics

Correlating marks in Maths and Physics

- We want to test the following hypothesis

  Student who perform well in Mathematics perform at least as well in Physics

- Assign grades {A,B,C,D} in both subjects
  - "Perform well in Maths" — grade B or above
  - "Perform at least as well in Physics" — Physics grade $\geq$ Maths grade

- Algorithm
  - Assign grades in each subject
  - Construct lists of students with grades A and B in both subjects — four lists
  - Count students in A list for Maths who are also in A list for Physics
  - Count students in B list for Maths who are also in A list or B list for Physics
  - Use these counts to confirm or reject the hypotheisis

Assigning grades

- Assign grades {A,B,C,D} approximately at quartile boundaries
    - Top 25% get A, next 25% get B, next 25% get C, bottom 25% get D

Assigning grades

- Assign grades {A,B,C,D} approximately at quartile boundaries
  - Top 25% get A, next 25% get B, next 25% get C, bottom 25% get D
- To calculate quartiles, extract marks as a list and sort the list

Assigning grades

- Assign grades {A,B,C,D} approximately at quartile boundaries
    - Top 25% get A, next 25% get B, next 25% get C, bottom 25% get D
- To calculate quartiles, extract marks as a list and sort the list
- Need to identify the students — each entry in the marks list is a pair `[StudentId,Marks]`

Assigning grades

- Assign grades {A,B,C,D} approximately at quartile boundaries
  - Top 25% get A, next 25% get B, next 25% get C, bottom 25% get D
- To calculate quartiles, extract marks as a list and sort the list
- Need to identify the students — each entry in the marks list is a pair `[StudentId,Marks]`
- Procedure to extract marks information as a list for a subject

```
Procecdure BuildMarksList(field)
    marksList = []
    while (Table 1 has more rows) {
      Read the first row X in Table 1
      marksList = marksList ++
                        [[X.SeqNo, X.field]]
      Move X to Table 2
    }
    return(marksList)
End BuildMarksList
```

Assigning grades

- Assign grades {A,B,C,D} approximately at quartile boundaries
    - Top 25% get A, next 25% get B, next 25% get C, bottom 25% get D
- To calculate quartiles, extract marks as a list and sort the list
- Need to identify the students — each entry in the marks list is a pair `[StudentId,Marks]`
- Procedure to extract marks information as a list for a subject
- Get the marks lists for Maths and Physics

```
Procedure BuildMarksList(field)
    marksList = []
    while (Table 1 has more rows) {
      Read the first row X in Table 1
      marksList = marksList ++
                        [[X.SeqNo, X.field]]
      Move X to Table 2
    }
    return(marksList)
End BuildMarksList
```

```
mathsList = BuildMarksList(Mathematics)

physicsList = BuildMarksList(Physics)
```

Assigning grades . . .

- Use insertion sort for mathList and physicsList?
    - Entries are [id,marks]
    - To compare [i1,m1] and [i2,m2], only look at m1, m2

Assigning grades . . .

- Use insertion sort for `mathList` and `physicsList`?
  - Entries are `[id,marks]`
  - To compare `[i1,m1]` and `[i2,m2]`, only look at `m1`, `m2`
- Extracting values at the beginning and end of a list
  - `first(l)` and `last(l)`

    `first([1,2,3,4])` is `1`,
    `last([1,2,3,4])` is `4`

Assigning grades . . .

- Use insertion sort for `mathList` and `physicsList`?
    - Entries are `[id,marks]`
    - To compare `[i1,m1]` and `[i2,m2]`, only look at `m1`, `m2`
- Extracting values at the beginning and end of a list
    - `first(l)` and `last(l)`

      `first([1,2,3,4])` is `1`,
      `last([1,2,3,4])` is `4`
    - The remainder of the list is given by `rest(l)` and `init(l)`, respectively

      `rest([1,2,3,4])` is `[2,3,4]`,
      `init([1,2,3,4])` is `[1,2,3]`

Assigning grades . . .

- Use insertion sort for mathList and physicsList?
  - Entries are [id,marks]
  - To compare [i1,m1] and [i2,m2], only look at m1, m2
- Extracting values at the beginning and end of a list
  - first(l) and last(l)

    first([1,2,3,4]) is 1,
    last([1,2,3,4]) is 4
  - The remainder of the list is given by rest(l) and init(l), respectively

    rest([1,2,3,4]) is [2,3,4],
    init([1,2,3,4]) is [1,2,3]
- Modify SortedListInsert

```
Procedure SortedListInsert(l,x)
   newList = []
   inserted = False

   foreach z in l {
     if (not(inserted)) {
       if (last(x) < last(z)) {
         newList = newList ++ [x]
         inserted = True
       }
     }
     newList = newList ++ [z]
   }

   if (not(inserted)) {
     newList = newList ++ [x]
   }

   return(newList)
End SortedListInsert
```

Assigning grades . . .

- InsertionSort uses updated SortedListInsert

```
sortedMathsList =
           InsertionSort(mathsList)
sortedPhysicsList =
           InsertionSort(physicsList)
```

Assigning grades . . .

- Assign grades to a sorted list by quartile

```
Procedure SimpleGradeAssignment(l)

End SimpleGradeAssignment
```

Assigning grades . . .

- Assign grades to a sorted list by quartile
  - `length(l)` returns number of elements in `l`
  - Compute quartile boundaries based on class size

```
Procedure SimpleGradeAssignment(l)
    classSize = length(l)
    q4 = classSize/4
    q3 = classSize/2
    q2 = 3*classSize/4
```

Assigning grades . . .

- Assign grades to a sorted list by quartile
  - length(l) returns number of elements in l
  - Compute quartile boundaries based on class size
  - Initialize list for each grade

```
Procedure SimpleGradeAssignment(l)
    q4 = ..., q3 = ..., q2 = ...

    gradeA = []
    gradeB = []
    gradeC = []
    gradeD = []
```

Assigning grades . . .

- Assign grades to a sorted list by quartile
  - length(l) returns number of elements in l
  - Compute quartile boundaries based on class size
  - Initialize list for each grade
  - Assign grades based on the position in the list

```
Procedure SimpleGradeAssignment(l)
   q4 = ..., q3 = ..., q2 = ...
   gradeA = [], ..., gradeD = []
   position = 0
   foreach x in l {
     if (position > q2) {
       gradeA = gradeA ++ [first(x)]
     }
     if (position > q3 and position <= q2) {
       gradeB = gradeB ++ [first(x)]
     }
     if (position > q4 and position <= q3) {
       gradeC = gradeC ++ [first(x)]
     }
     if (position <= q4) {
       gradeD = gradeD ++ [first(x)]
     }
     position = position + 1
   }
   return([gradeA,gradeB,gradeC,gradeD])
End SimpleGradeAssignment
```

Assigning grades . . .

- Assign grades to a sorted list by quartile
  - `length(l)` returns number of elements in `l`
  - Compute quartile boundaries based on class size
  - Initialize list for each grade
  - Assign grades based on the position in the list
- `SimpleGradeAssignment` returns a list containing four lists, for the four grades

```
Procedure SimpleGradeAssignment(l)
   q4 = ..., q3 = ..., q2 = ...
   gradeA = [], ..., gradeD = []
   position = 0
   foreach x in l {
     if (position > q2) {
       gradeA = gradeA ++ [first(x)]
     }
     if (position > q3 and position <= q2) {
       gradeB = gradeB ++ [first(x)]
     }
     if (position > q4 and position <= q3) {
       gradeC = gradeC ++ [first(x)]
     }
     if (position <= q4) {
       gradeD = gradeD ++ [first(x)]
     }
     position = position + 1
   }
   return([gradeA,gradeB,gradeC,gradeD])
End SimpleGradeAssignment
```

Assigning grades . . .

- Assign grades corresponding to Maths and Physics marks

```
mathsGrades =
    SimpleGradeAssignment(sortedMathsList)
physicsGrades =
    SimpleGradeAssignment(sortedPhysicsList)
```

Assigning grades . . .

- Assign grades corresponding to Maths and Physics marks

- Unpack the four lists into four separate lists

```
mathsGrades =
    SimpleGradeAssignment(sortedMathsList)
physicsGrades =
    SimpleGradeAssignment(sortedPhysicsList)

mathsAGrades = first(mathsGrades)
mathsBGrades = first(rest(mathsGrades))
mathsCGrades = last(init(mathsGrades))
mathsDGrades = last(mathsGrades)

physicsAGrades = first(physicsGrades)
physicsBGrades = first(rest(physicsGrades))
physicsCGrades = last(init(physicsGrades))
physicsDGrades = last(physicsGrades)
```

Test the hypothesis

- Check how many students with A in Maths confirm the hypothesis
  - `exitloop` prematurely terminates a `foreach` loop

```
confirm = []
reject = []
foreach x in mathsAGrades {
    found = False
    foreach y in physicsAGrades {
      if (x == y) {
        confirm = confirm ++ [x]
        found = True
        exitloop
      }
    }
    if (not(found)) {
      reject = reject ++ [x]
    }
}
```

Test the hypothesis

- Check how many students with A in Maths confirm the hypothesis
  - exitloop prematurely terminates a foreach loop
- Check how many students with B in Maths confirm the hypothesis

```
foreach x in mathsBGrades {
  found = False
  foreach y in physicsAGrades {
    if (x == y) {
      confirm = confirm ++ [x]
      found = True, exitloop
    }
  }
  if (not(found)) {
    foreach y in physicsBGrades {
      if (x == y) {
        confirm = confirm ++ [x]
        found = True, exitloop
      }
    }
  }
  if (not(found)) {
    reject = reject ++ [x]
  }
}
```

Test the hypothesis

- Check how many students with A in Maths confirm the hypothesis
  - exitloop prematurely terminates a foreach loop
- Check how many students with B in Maths confirm the hypothesis
- Finally check length(confirm) against length(confirm)+length(reject) to decide if the hypothesis holds

```
foreach x in mathsBGrades {
   found = False
   foreach y in physicsAGrades {
     if (x == y) {
       confirm = confirm ++ [x]
       found = True, exitloop
     }
   }
   if (not(found)) {
     foreach y in physicsBGrades {
       if (x == y) {
         confirm = confirm ++ [x]
         found = True, exitloop
       }
     }
   }
   if (not(found)) {
     reject = reject ++ [x]
   }
}
```

Summary

- Sorting was used to identify quartiles for grade assignment
- Need to modify the comparison function based on the items in the list
- `length(l)` returns number of elements in `l`
- New functions to extract first and last items of a list
    - `first(l)` and `rest(l)`
    - `last(l)` and `init(l)`
- `exitloop` to abort a `foreach` loop