Pseudocode: Working with Graphs

Mentoring

- Student A can mentor student B in a subject if A has higher marks, but not too much higher are between
  - Difference is between 10 and 20 marks

Mentoring

- Student A can mentor student B in a subject if A has higher marks, but not too much higher are between
  - Difference is between 10 and 20 marks
- Create a dictionary for marks in subject
  - `mathMarks = ReadMarks(Mathematics)`

```
Procedure ReadMarks(subj)
  marks = {}
  while (Table 1 has more rows) {
    Read the first row X in Table 1
    marks[X.Seqno] = X.subj
  }
  Move X to Table 2
  return(marks)
End ReadMarks
```

Mentoring

- Student A can mentor student B in a subject if A has higher marks, but not too much higher are between
  - Difference is between 10 and 20 marks
- Create a dictionary for marks in subject
  - `mathMarks = ReadMarks(Mathematics)`
- Create a mentoring graph for a subject
  - Represent as a matrix
  - `M[i][j] = 1` — edge from `i` to `j`
  - `M[i][j] = 0` — no edge from `i` to `j`

```
Procedure CreateMentorGraph(marks)
  n = length(keys(marks))
  mentorGraph = CreateMatrix(n,n)
  foreach i in keys(marks){
    foreach j in keys(marks){
      ijMarksDiff = marks[i] - marks[j]
      if (10 ≤ ijMarksDiff and
            ijMarksDiff ≤ 20) {
        mentorGraph[i][j] = 1
      }
    }
  }
  return(mentorGraph)
End CreateMentorGraph(marks)
```

Pairing students in study groups

- A can mentor student B in one subject and B can mentor A in the other

Pairing students in study groups

- A can mentor student B in one subject and B can mentor A in the other
- Study groups in Maths and Physics
  - Create mentoring graphs for each

```
mathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)

mathMentorGraph =
          CreateMentorGraph(mathMarks)
phyMentorGraph =
          CreateMentorGraph(phyMarks)
```

Pairing students in study groups

- A can mentor student B in one subject and B can mentor A in the other
- Study groups in Maths and Physics
  - Create mentoring graphs for each
- Use the mentoring graphs to pair off students

```
mathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)

mathMentorGraph =
          CreateMentorGraph(mathMarks)
phyMentorGraph =
          CreateMentorGraph(phyMarks)

paired = {}

foreach i in rows(mathMentorGraph) {
  foreach j in columns(mathMentorGraph) {
    if (mathMentorGraph[i][j] == 1 and
          phyMentorGraph[j][i] == 1 and
          not(isKey(paired,i)) and
          not(isKey(paired,j))) {
      paired[i] = j
      paired[j] = i
    }
  }
}
```

Popular students

- A student who can be mentored by many other students is popular

Popular students

- A student who can be mentored by many other students is popular
- Create mentoring graphs for all three subjects

```
CodemathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)
chemMarks = ReadMarks(Chemistry)

mathMentorGraph =
        CreateMentorGraph(mathMarks)
phyMentorGraph =
        CreateMentorGraph(phyMarks)
chemMentorGraph =
        CreateMentorGraph(chemMarks)
```

Popular students

- A student who can be mentored by many other students is popular
- Create mentoring graphs for all three subjects
- Count incoming mentoring edges for each student

```
popularity = {}
foreach i in keys(mathMarks) {
  popularity[i] = 0
  foreach j in keys(mathMarks) {
    if (mathMentorGraph[j][i] == 1){
      popularity[i] = popularity[i] + 1
    }
    if (phyMentorGraph[j][i] == 1){
      popularity[i] = popularity[i] + 1
    }
    if (chemMentorGraph[j][i] == 1){
      popularity[i] = popularity[i] + 1
    }
  }
}
```

Popular students

- A student who can be mentored by many other students is popular
- Create mentoring graphs for all three subjects
- Count incoming mentoring edges for each student
- Avoid duplicates
  - Explicitly keep track of mentors for each student and count them

```
mentors = {}
popularity = {}
foreach j in columns(mathMentorGraph) {
  mentors[j] = {}
  foreach i in rows(mathMentorGraph) {
    if (mathMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
    if (phyMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
    if (chemMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
  }
  popularity[j]
       = length(keys(mentors[j]))
}
```

Similar students

- Two students are similar if they have similar marks in all subjects
  - Difference is within 10 marks
- Dictionaries with marks in each subject

```
mathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)
chemMarks = ReadMarks(Chemistry)
```

Similar students

- Two students are similar if they have similar marks in all subjects
  - Difference is within 10 marks
- Dictionaries with marks in each subject

```
mathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)
chemMarks = ReadMarks(Chemistry)
```

- Create a similarity graph

```
Procedure
    CreateSimilarityGraph(marks1,mark2,marks3)
  n = length(keys(marks1))
  similarityGraph = CreateMatrix(n,n)
  foreach i in keys(marks1){
    foreach j in keys(marks1){
      ijDiff1 = abs(marks1[i] - marks1[j])
      ijDiff2 = abs(marks2[i] - marks2[j])
      ijDiff3 = abs(marks3[i] - marks3[j])
      if (ijDiff1 ≤ 10 and
            ijDiff2 ≤ 10 and
            ijDiff3 ≤ 10)}
        similarityGraph[i][j] = 1
      }
    }
  }
  return(similarityGraph)
End CreateSimilarityGraph(marks)
```

Summary

- Graphs are a useful way to represent relationships
  - Add an edge from `i` to `j` if `i` is related to `j`
- Use matrices to represent graphs
  - `M[i][j] = 1` — edge from `i` to `j`
  - `M[i][j] = 0` — no edge from `i` to `j`
- Iterate through matrix to aggregate information from the graph