Pseudocode: Edge Labelled Graphs

Trains

- Train dataset — information about trains and stations
  - Each train is a route list of stations
  - Each station is a route list of trains passing through
- Compute pairs of stations connected by a direct train
- Represent start and end station of trains in a nested dictionary
  - `trains[t][start]`, `trains[t][end]`

Train

M W Th Su

12259

Dist.	Station Name	Arr.	Dep.	Day
0	Kolkata	--	18:31	1
266	Dhanbad	21:52	21:56	1
667	Varanasi	02:20	02:30	2
1014	Kanpur	06:25	06:30	2
1453	New Delhi	11:31	11:45	2
1628	Loharu	14:23	14:26	2
1678	Sadulpur	15:03	15:06	2
1736	Churu	16:06	16:08	2
1779	Ratangarh	17:00	17:02	2
1844	Sri Dungargarah	17:47	17:48	2
1916	Bikaner	19:15	--	2

Station

Nagpur

Train No.	Arr.	Dep.	Days
12261	04:10	04:15	M W Th F
12221	04:10	04:15	Tu Su
02286	05:15	05:20	Tu Sa
12270	05:15	05:20	W Su
12289	07:20	--	M Tu W Th F Sa Su
12290	--	20:40	M Tu W Th F Sa Su
12269	21:00	21:06	M F
02285	21:00	21:06	Th Su
12222	23:15	23:20	Th Sa
12262	23:15	23:20	M Tu W F

Adding information to direct route graph

- Stations A and B such that a train starts at A and ends at B
- Create a matrix to record direct routes
  - Compile the list of stations from `trains`
  - Map stations to row, column indices
  - Populate the matrix

```
Procedure DirectRoutes(trains)
  stations = {}
  foreach t in keys(trains) {
    stations[trains[t][start]] = True
    stations[trains[t][end]] = True
  }
  n = length(keys(stations))
  direct = CreateMatrix(n,n)
  stnindex = {}
  i = 0
  foreach s in keys(stations) {
    stnindex[s] = i
    i = i+1
  }
  foreach t in keys(trains){
    i = stnindex[trains[t][start]]
    j = stnindex[trains[t][end]]
    direct[i][j] = 1
  }
  return(direct)
End DirectRoutes
```

Adding information to direct route graph

- Stations A and B such that a train starts at A and ends at B
- Create a matrix to record direct routes
  - Compile the list of stations from `trains`
  - Map stations to row, column indices
  - Populate the matrix
- Keep track of trains connecting stations
  - Each entry in the matrix is now a dictionary
  - Initially empty dictionary — no direct connection
  - Add a key for each train connecting a pair of stations

```
Procedure LabelledDirectRoutes(trains)
  :
  :
  foreach r rows(direct) {
    foreach c columns(direct) {
      direct[i][j] = {}
    }
  }
  foreach t in keys(trains){
    i = stnindex[trains[t][start]]
    j = stnindex[trains[t][end]]
    direct[i][j][t] = True
  }
  return(direct)
End DirectRoutes2
```

Adding information to direct route graph

- Stations A and B such that a train starts at A and ends at B
- Create a matrix to record direct routes
  - Compile the list of stations from `trains`
  - Map stations to row, column indices
  - Populate the matrix
- Keep track of trains connecting stations
  - Each entry in the matrix is now a dictionary
  - Initially empty dictionary — no direct connection
  - Add a key for each train connecting a pair of stations

```
Procedure LabelledDirectRoutes(trains)
  :
  :
  foreach r rows(direct) {
    foreach c columns(direct) {
      direct[i][j] = {}
    }
  }
  foreach t in keys(trains){
    i = stnindex[trains[t][start]]
    j = stnindex[trains[t][end]]
    direct[i][j][t] = True
  }
  return(direct)
End DirectRoutes2
```

- Information about trains is recorded as a label on the edge
  - Edge-labelled graph

Distances between stations

- For each direct train record the distance it travels
  - Add an extra key, `trains[t][distance]`

Distances between stations

- For each direct train record the distance it travels
  - Add an extra key, `trains[t][distance]`
- Compute the shortest distance by direct trains
  - Trains may take different routes between the same pair of stations

Distances between stations

- For each direct train record the distance it travels
  - Add an extra key, `trains[t][distance]`
- Compute the shortest distance by direct trains
  - Trains may take different routes between the same pair of stations
- Graph `directdist`
  - Edge label `directdist[i][j]` is the shortest direct distance between `i` and `j`

Distances between stations

- Compute the shortest distance by direct trains
- Edge-labelled graph `directdist`

```
Procedure DirectDistance(trains)
  :
  directdist = CreateMatrix(n,n)
  :
  foreach t in keys(trains){
    i = stn2idx[trains[t][start]]
    j = stn2idx[trains[t][end]]
    if (directdist[i][j] == 0) {
      directdist[i][j] = trains[t][distance]
    }
    else
      directdist[i][j] =
          min(directdist[i][j],trains[t][distance])
    }
  }
  return(directdist)
End DirectDistance
```

Distances between stations

- Compute the shortest distance by direct trains
- Edge-labelled graph `directdist`
- When we discover a direct route for the first time, set the distance

```
Procedure DirectDistance(trains)
  ⋮
  directdist = CreateMatrix(n,n)
  ⋮
  foreach t in keys(trains){
    i = stn2idx[trains[t][start]]
    j = stn2idx[trains[t][end]]
    if (directdist[i][j] == 0) {
      directdist[i][j] = trains[t][distance]
    }
    else
      directdist[i][j] =
          min(directdist[i][j],trains[t][distance])
    }
  }
  return(directdist)
End DirectDistance
```

Distances between stations

- Compute the shortest distance by direct trains
- Edge-labelled graph `directdist`
- When we discover a direct route for the first time, set the distance
- If we find a new direct route between an already connected pair, update the distance to the minimum

```
Procedure DirectDistance(trains)
  ⋮
  directdist = CreateMatrix(n,n)
  ⋮
  foreach t in keys(trains){
    i = stn2idx[trains[t][start]]
    j = stn2idx[trains[t][end]]
    if (directdist[i][j] == 0) {
      directdist[i][j] = trains[t][distance]
    }
    else
      directdist[i][j] =
          min(directdist[i][j],trains[t][distance])
    }
  }
  return(directdist)
End DirectDistance
```

One hop distance

- We start with the matrix of direct distances

```
Procedure OneHopDistance(directdist)
  n = length(keys(directdist))
  onehopdist = CreateMatrix(n,n)
  foreach i in rows(directdist) {
    foreach j in columns(directdist) {
      onehopdist[i][j] = directdist[i][j]
      foreach k in columns(directdist) {
        if (directdist[i][k] > 0 and
            directdist[k][j] > 0) {
          newdist = directdist[i][k]
                    + directdist[k][j]
          if (onehopdist[i][j] > 0){
            onehopdist[i][j] =
              min(newdist,onehopdist[i][j])
          }
          else{
            onehopdist[i][j] = newdist
  } } } } }
  return(onehopdist)
End OneHopDistance
```

One hop distance

- We start with the matrix of direct distances
- Initialize one hop distance to direct distance

```
Procedure OneHopDistance(directdist)
  n = length(keys(directdist))
  onehopdist = CreateMatrix(n,n)
  foreach i in rows(directdist) {
    foreach j in columns(directdist) {
      onehopdist[i][j] = directdist[i][j]
      foreach k in columns(directdist) {
        if (directdist[i][k] > 0 and
            directdist[k][j] > 0) {
          newdist = directdist[i][k]
                    + directdist[k][j]
          if (onehopdist[i][j] > 0){
            onehopdist[i][j] =
              min(newdist,onehopdist[i][j])
          }
          else{
            onehopdist[i][j] = newdist
  } } } } }
  return(onehopdist)
End OneHopDistance
```

One hop distance

- We start with the matrix of direct distances
- Initialize one hop distance to direct distance
- Each time we discover a one hop route, update the one hop distance

```
Procedure OneHopDistance(directdist)
  n = length(keys(directdist))
  onehopdist = CreateMatrix(n,n)
  foreach i in rows(directdist) {
    foreach j in columns(directdist) {
      onehopdist[i][j] = directdist[i][j]
      foreach k in columns(directdist) {
        if (directdist[i][k] > 0 and
            directdist[k][j] > 0) {
          newdist = directdist[i][k]
                    + directdist[k][j]
          if (onehopdist[i][j] > 0){
            onehopdist[i][j] =
              min(newdist,onehopdist[i][j])
          }
          else{
            onehopdist[i][j] = newdist
  } } } } }
  return(onehopdist)
End OneHopDistance
```

One hop distance

- We start with the matrix of direct distances
- Initialize one hop distance to direct distance
- Each time we discover a one hop route, update the one hop distance
- Iterate this to find length of the shortest path between each pair of stations
  - Modify the transitive closure calculation to record minimum distance path

```
Procedure OneHopDistance(directdist)
  n = length(keys(directdist))
  onehopdist = CreateMatrix(n,n)
  foreach i in rows(directdist) {
    foreach j in columns(directdist) {
      onehopdist[i][j] = directdist[i][j]
      foreach k in columns(directdist) {
        if (directdist[i][k] > 0 and
            directdist[k][j] > 0) {
          newdist = directdist[i][k]
                    + directdist[k][j]
          if (onehopdist[i][j] > 0){
            onehopdist[i][j] =
              min(newdist,onehopdist[i][j])
          }
          else{
            onehopdist[i][j] = newdist
  } } } } }
  return(onehopdist)
End OneHopDistance
```

Summary

- We can represent extra information in a graph via edge labels
  - Train name, distance
  - For each edge in the graph, replace 1 in the matrix by the edge label
- Iteratively update labels
  - Compute shortest distance path between each pair of stations