Graph Algorithms

M269 Unit 5

Contents

1 Agenda

- Welcome and introductions
- Session on M269 Unit 5 *Optimisation*
- Graph definitions and representations
- Python: List comprehensions, Named Tuples
- Topological Sort for directed acyclic graphs
- Dijkstra's Shortest Path Algorithm
- Prim's Minimum Spanning Tree Algorithm
- Dynamic Programming
- Implementations in Structured English, Python and Haskell (Optional)
- *Note* there is more material here than we can cover — some is for optional interest
- Slides/Notes are at pmolyneux.co.uk/OU/M269/M269TutorialNotes/M269TutorialGraphs/
- **Recording** Meeting > Record Meeting... ✔

Introductions — Me

- *Name* Phil Molyneux
- *Background* Physics & Maths, Operational Research, Computer Science
- *First programming languages* Fortran, BASIC, Pascal
- *Favourite Software*
  - Haskell — pure functional programming language
  - Text editors TextMate, Sublime Text — previously Emacs

- Word processing in LaTeX — all these slides and notes
- Mac OS X

- *Learning style* — I read the manual before using the software

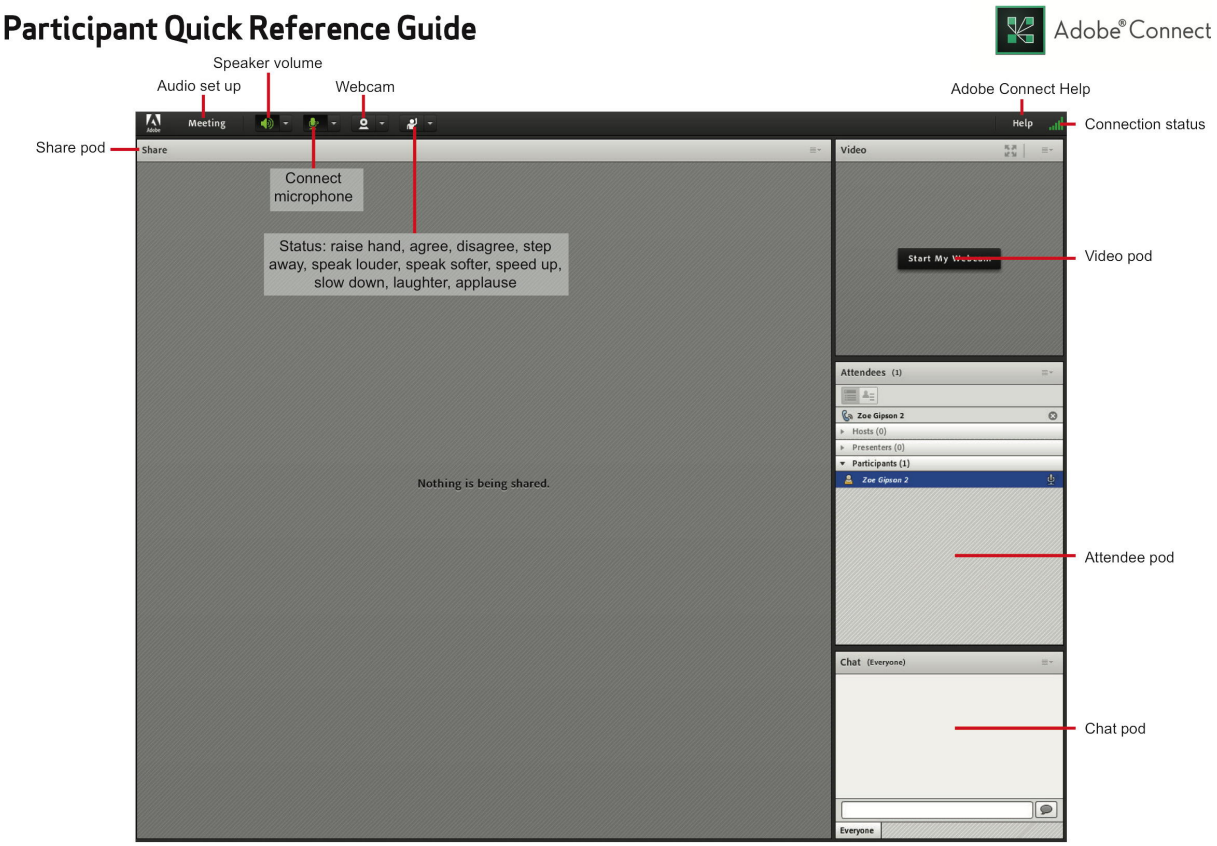
Introductions — You

- *Name* ?
- *New topics last month* ?
- *M269 Unit 5 Graph Algorithm topics you want covered* ?
- *Learning style* ?
- *Other OU courses* ?
- *Anything else* ?
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)

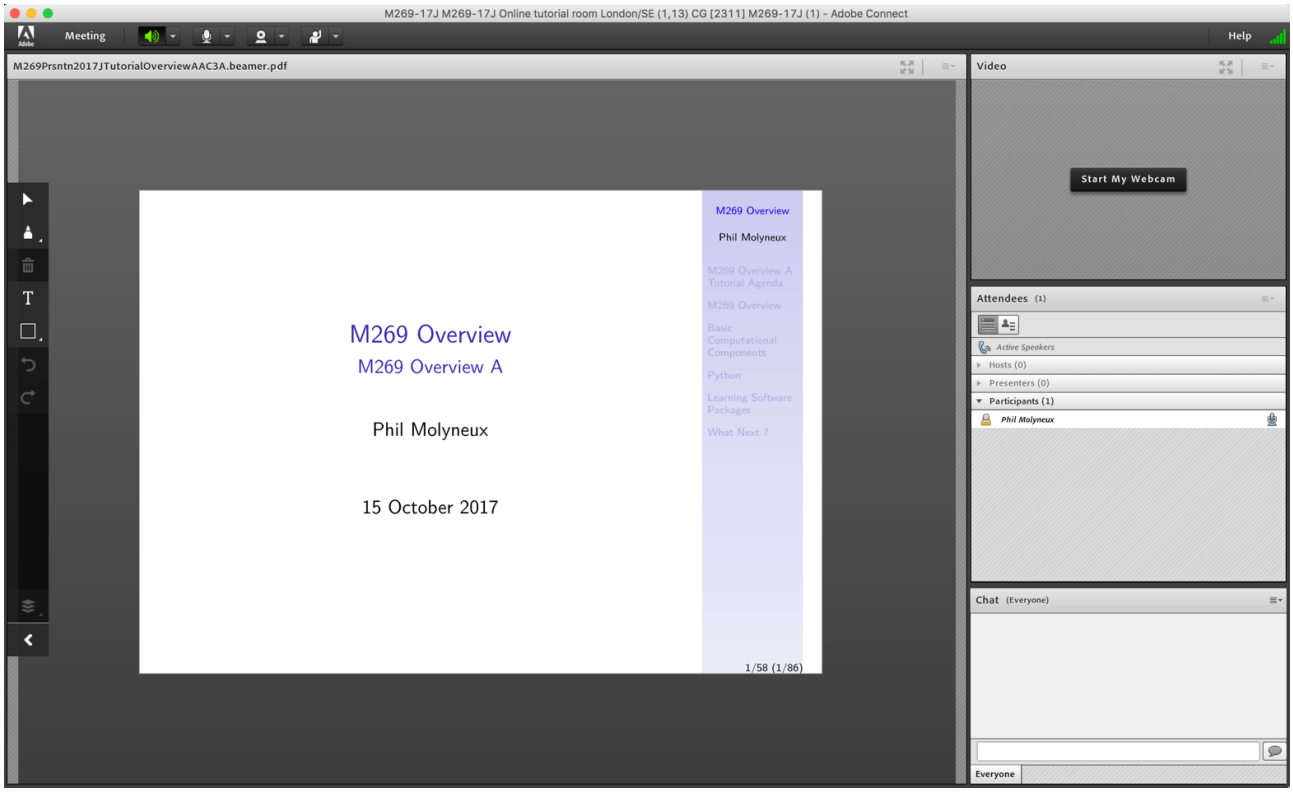
2 Adobe Connect Interface and Settings

2.1 Adobe Connect Interface — Student View

Adobe Connect Interface — Student Quick Reference



Adobe Connect Interface — Student View



2.2 Adobe Connect Settings

Adobe Connect Settings

- **Everybody: Audio Settings** Meeting › Audio Setup Wizard. . .
- **Audio** Menu bar › Audio › Microphone rights for Participants ✔
- Do not *Enable single speaker mode*
- **Drawing Tools** Share pod menu bar › Draw (1 slide/screen)
- Share pod menu bar › Menu icon › Enable Participants to draw ✔ gray
- Meeting › Preferences › Whiteboard › Enable Participants to draw ✔
- Cancel hand tool . . . Do not enable green pointer. . .
- Meeting › Preferences › Attendees Pod ✘ *Raise Hand notification*
- Meeting › Preferences › Display Name Display First & Last Name
- **Cursor** Meeting › Preferences › General tab › Host Cursors › Show to all attendees ✔ (default *Off*)
- Meeting › Preferences › Screen Share › Cursor › Show Application Cursor
- **Webcam** Menu bar › Webcam › Enable Webcam for Participants ✔
- **Recording** Meeting › Record Meeting. . . ✔

Adobe Connect — Access

- **Tutor Access**

  TutorHome › M269 Website › Tutorials

  Cluster Tutorials › M269 Online tutorial room

  Tutor Groups › M269 Online tutor group room

  Module-wide Tutorials › M269 Online module-wide room

- **Attendance**

  TutorHome › Students › View your tutorial timetables

- **Beamer Slide Scaling** 440% (422 x 563 mm)
- **Clear Everyone's Status**

  Attendee Pod › Menu › Clear Everyone's Status

- **Grant Access** and send link via email

  Meeting › Manage Access & Entry › Invite Participants. . .

- **Presenter Only Area**

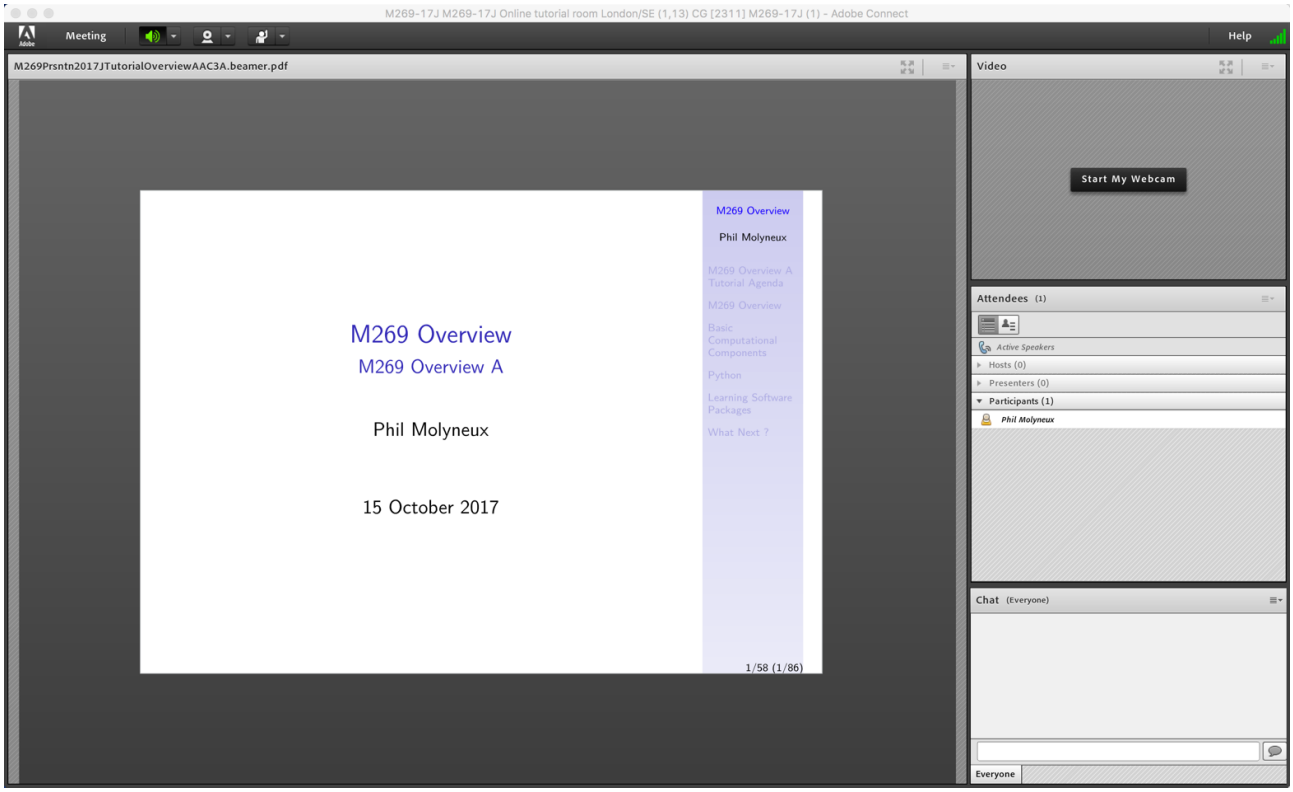
  Meeting › Enable/Disable Presenter Only Area

Adobe Connect — Keystroke Shortcuts

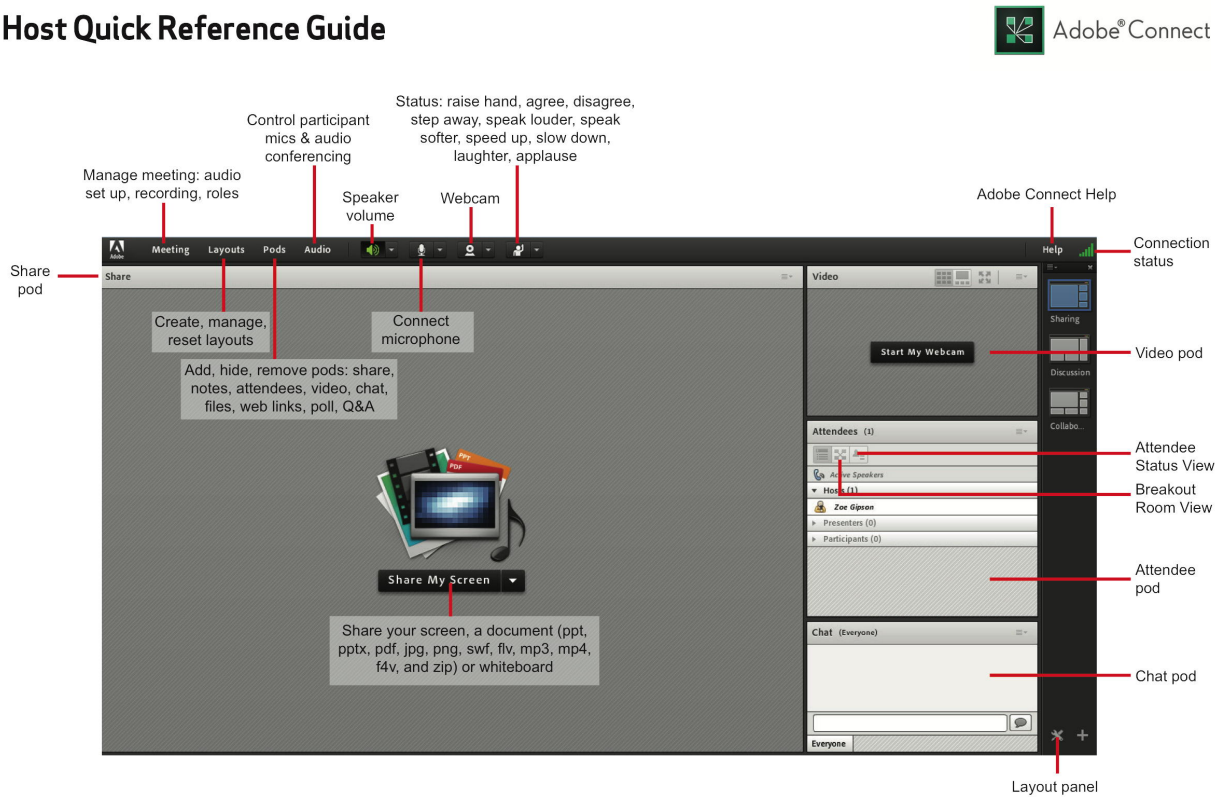
- Keyboard shortcuts in Adobe Connect
- **Toggle Mic** ⌘ + M (Mac), Ctrl + M (Win) (On/Disconnect)
- **Toggle Raise-Hand status** ⌘ + E
- **Close dialog box** ⎋ (Mac), Esc (Win)
- **End meeting** ⌘ + \

2.3 Adobe Connect Interface — Student & Tutor Views

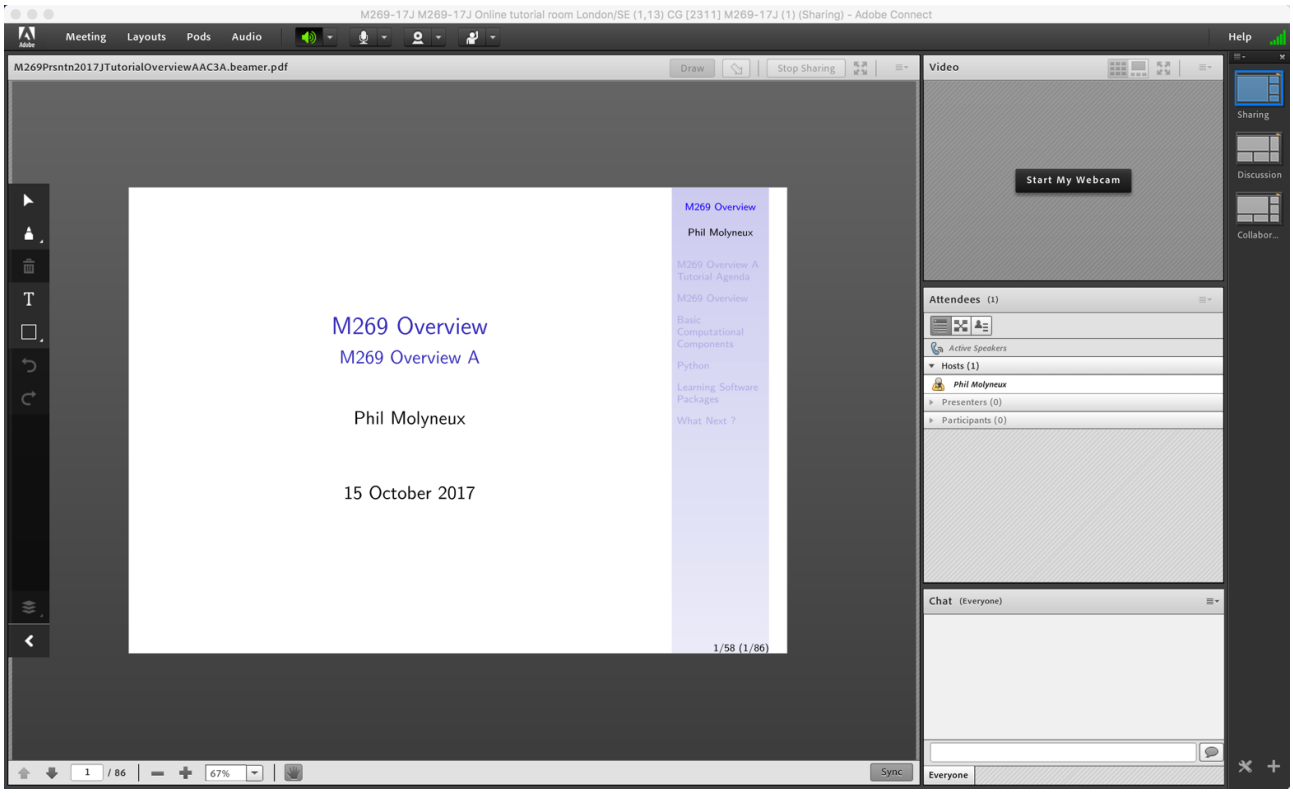
Adobe Connect Interface — Student View (default)



Adobe Connect Interface — Tutor Quick Reference



Adobe Connect Interface — Tutor View



2.4 Adobe Connect — Sharing Screen & Applications

- Share My Screen › Application tab › Terminal for Terminal
- Share menu › Change View › Zoom in for mismatch of screen size/resolution (Participants)
- (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- Leave the application on the original display
- Beware blued hatched rectangles — from other (hidden) windows or contextual menus
- Presenter screen pointer affects viewer display — beware of moving the pointer away from the application
- First time: System Preferences › Security & Privacy › Privacy › Accessibility

2.5 Adobe Connect — Ending a Meeting

- *Notes for the tutor only*
- **Student:** Meeting › Exit Adobe Connect
- **Tutor:**
- **Recording** Meeting › Stop Recording ✔
- **Remove Participants** Meeting › End Meeting... ✔
  - Dialog box allows for message with default message:
  - *The host has ended this meeting. Thank you for attending.*
- **Recording availability** *In course Web site for joining the room, click on the eye icon in the list of recordings under your recording* — edit description and name
- **Meeting Information** Meeting › Manage Meeting Information — can access a range of information in Web page.
- **Attendance Report** see course Web site for joining room

2.6 Adobe Connect — Invite Attendees

- **Provide Meeting URL** Menu › Meeting › Manage Access & Entry › Invite Participants...
- **Allow Access without Dialog** Menu › Meeting › Manage Meeting Information provides new browser window with *Meeting Information* Tab bar › Edit Information
- Check *Anyone who has the URL for the meeting can enter the room*
- Default *Only registered users and accepted guests may enter the room*
- **Reverts to default next session but URL is fixed**
- Guests have blue icon top, registered participants have yellow icon top — same icon if URL is open

- See Start, attend, and manage Adobe Connect meetings and sessions

2.7 Layouts

- **Creating new layouts** example *Sharing* layout
- Menu > Layouts > Create New Layout... > Create a New Layout dialog > Create a new blank layout and name it *PMolyMain*
- New layout has no Pods but does have Layouts Bar open (see Layouts menu)
- **Pods**
- Menu > Pods > Share > Add New Share and resize/position — initial name is *Share n*
- **Rename Pod** Menu > Pods > Manage Pods... > Manage Pods > Select > Rename or Double-click & rename
- Add Video pod and resize/reposition
- Add Attendance pod and resize/reposition
- Add Chat pod — name it *PMolyChat* — and resize/reposition
- Dimensions of **Sharing** layout (on 27-inch iMac)
  - Width of Video, Attendees, Chat column 14 cm
  - Height of Video pod 9 cm
  - Height of Attendees pod 12 cm
  - Height of Chat pod 8 cm
- **Duplicating Layouts** does *not* give new instances of the Pods and is probably not a good idea (apart from local use to avoid delay in reloading Pods)

2.8 Chat Pods

- **Format Chat text**
- Chat Pod > menu icon > My Chat Color
- Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black
- Note: Color reverts to Black if you switch layouts
- Chat Pod > menu icon > Show Timestamps

Go to Table of Contents

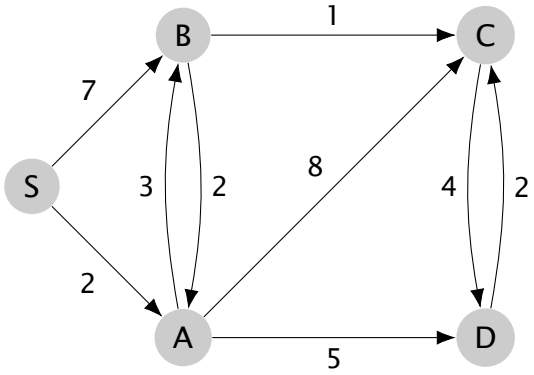
3 M269 Graph Algorithms

3.1 Graph Definitions

- A *Graph*, G, consists of a pair: a set of *vertices*, V, and a set of *edges*, E, where an edge (u, v) represents a connection between two vertices, u and v

- Equivalently, a graph is a set of objects together with a relation over that set
- Edges may have *direction* — that is, the relation is not symmetric — a graph with directed edges is called a *digraph*
- Informally, graphs are represented as diagrams (see below)
- If G = (V, E) is a *weighted digraph* then there is a function w :: E → $\mathbb{R}$ which maps edges to real numbers.
- If e = (u, v) we write w(u, v) for w(e)

Example Digraph



3.2 Graph Representation

- What operations do we want on graphs ?
- How can we implement a representation of graphs and the operations efficiently ?
- Common representations
  - Adjacency list — a linear structure holds every vertex together with a list of successor vertices and the weights of the successor edges.
  - Adjacency matrix — 2 dimensional array of values of dimension |V| × |V| where both coordinates u and v are vertices and the entry (u, v) is the weight of the edge (if it exists)
- Additional points:
  - A vertex may have other data: name, label with data (shortest path predecessors, distance, . . .)
  - An edge may have other data: weight, status (on shortest path, minimum spanning tree, . . .)

Graph Operations

Activity 1 Graph Operations

- In the space below give a graph operation indicating whether it is a creator, inspector or modifier and give its pre and post conditions

Go to Answer

Answer 1 Graph Operations

- Answer 1 Graph Operations — see next slide

Go to Activity

- *emptyGraph* returns an empty graph
- *mkGraph* takes a list of vertices, and a list of edges and returns a graph
- *isEmptyGraph* takes a graph and returns True if and only if the graph is empty.
- *vertices* takes a graph and returns the vertices
- *edges* takes a graph and returns the edges
- *succLists* takes a graph and returns a list of pairs of vertices and lists of successor edges
- *predLists* takes a graph and returns a list of pairs of vertices and lists of predecessor edges
- *startVertices* takes a graph and returns a list of vertices with no predecessors
- *endVertices* takes a graph and returns a list of vertices with no successors
- *removeVertex* takes a vertex and a graph and returns a graph with the vertex removed.
- Further service functions:
  - *esRemoveV* takes a vertex and a list of edges and returns the list of edges with the vertex removed.
  - *esStartV* takes a vertex and a list of edges and returns the list of edges where the given vertex is the start of an edge
  - *esEndV* takes a vertex and a list of edges and returns the list of edges where the given vertex is the end of an edge

Graph Representation 01

- `Adjacency matrix` Assign a unique label to each vertex and construct an $n \times n$ matrix of values in which (i, j) is x if $(i, j) \in E$ and x is its label, (i, i) is 0 and all other entries are ∞
- The adjacency matrix for the previous example digraph is:

	S	A	B	C	D
S	0	2	7	∞	∞
A	∞	0	3	8	5
B	∞	2	0	1	∞
C	∞	∞	∞	0	4
D	∞	∞	∞	2	0

Graph Representation 02

- The explicit adjacency list or matrix representations are biased towards the procedural view of programming.
- A functional view looks for an *inductive* definition (as we had with trees)
- Functional view:
  - A graph is either the empty graph or
  - a graph extended by a new node v together with its label and with edges to those of v's successors and predecessors *that are already in the graph*
- See FGL — A Functional Graph Library and Erwig (2001)
- M269 Python examples use *adjacency lists* to represent graphs.
- The Haskell examples in these notes use a simple (but inefficient) representation to illustrate the algorithms.

4 Algorithm Descriptions & Implementations

- The algorithms are described in a mix of Structured English, Python and Haskell
- The Python and Haskell code does not use any advanced features but may use some features not mentioned in M269
- In Python the code may use:
  - List comprehensions (tutorial), List comprehensions (reference) — a neat way of expressing iterations over a list, came from Miranda
  - Named tuples — a *Factory Function* for tuple with named fields — *quick & dirty* objects
- The Haskell syntax is defined as it is used — novel concepts may be:
  - Algebraic Data Types — just name your user defined data type and name its elements — magic!
  - Explicit type specifications — Haskell has a very powerful type system that can help spot errors.
  - List comprehensions — as above

4.1 List Comprehensions

List Comprehensions — Python

- **List Comprehensions** provide a concise way of performing calculations over lists (or other iterables)
- Example: Square the even numbers between 0 and 9

```
Python3>>> [x ** 2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
Python3>>> [(x,y) for x in range(4)
...               for y in range(4)
...               if x % 2 == 0
...                  and y % 3 == 0]
[(0, 0), (0, 3), (2, 0), (2, 3)]
Python3>>>
```

- In general

```
  [expr for target1 in iterable1 if cond1
        for target2 in iterable2 if cond2 ...
        for targetN in iterableN if condN ]
```

- Lots example usage in the algorithms below

List Comprehensions — Haskell

- **List Comprehensions** provide a concise way of performing calculations over lists
- Example: Square the even numbers between 0 and 9

```
GHCi> [x^2 | x <- [0..9], x `mod` 2 == 0]
[0,4,16,36,64]
GHCi>
```

- In general

```
  [expr | qual1, qual2,..., qualN]
```

- The qualifiers `qual` can be
  - Generators `pattern <- list`
  - Boolean guards — acting as filters
  - Local declarations with `let decls` for use in `expr` and later generators and boolean guards

Activity 2 (a) Stop Words Filter

- Stop words are the most common words that most search engines avoid: `'a','an','the','th`
- Using list comprehensions, write a function `filterStopWords` that takes a list of words and filters out the stop words
- Here is the initial code

```
11  sentence \
12   = "the quick brown fox jumps over the lazy dog"

14  words = sentence.split()

16  wordsTest \
17   = (words == ['the', 'quick', 'brown'
18               , 'fox', 'jumps', 'over'
19               , 'the', 'lazy', 'dog'])

21  stopWords \
22   = ['a','an','the','that']
```

Go to Answer

Activity 2 (a) Stop Words Filter

```
sentence \
 = "the␣quick␣brown␣fox␣jumps␣over␣the␣lazy␣dog"

words = sentence.split()

wordsTest \
 = (words == ['the', 'quick', 'brown'
             , 'fox', 'jumps', 'over'
             , 'the', 'lazy', 'dog'])

stopWords \
 = ['a','an','the','that']
```

- Notice the Python Explicit line joining with (\<nl>) and Python Implicit line joining with (`(...)`)
- The backslash (\) must be followed by an end of line character (<nl>)
- The ('␣') symbol represents a space (see Unicode U+2423 Open Box)

Go to Answer

Activity 2 (b) Transpose Matrix

- A matrix can be represented as a list of rows of numbers
- We *transpose* a matrix by swapping columns and rows
- Here is an example

```
matrixA \
 = [[1, 2, 3, 4]
   ,[5, 6, 7 ,8]
   ,[9, 10, 11, 12]]

matATr \
 = [[1, 5, 9]
   ,[2, 6, 10]
   ,[3, 7, 11]
   ,[4, 8, 12]]
```

- Using list comprehensions, write a function `transMat`, to transpose a matrix

Go to Answer

Activity 2 (c) List Pairs in Fair Order

- Write a function which takes a pair of positive integers and outputs a list of all possible pairs in those ranges
- If we do this in the simplest way we get a bias to one argument
- Here is an example of a bias to the second argument

```
yBiasLstTest \
 = (yBiasListing(5,5)
     == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
        , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
        , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
        , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
        , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Answer

Activity 2 (c) List Pairs in Fair Order

- Rewrite the function which takes a pair of positive integers and outputs a list of all possible pairs in those ranges

- The output should treat each argument *fairly* — any initial prefix should have roughly the same number of instances of each argument
- Here is an example output

```
fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
        , (0, 1), (1, 0)
        , (0, 2), (1, 1), (2, 0)
        , (0, 3), (1, 2), (2, 1), (3, 0)
        , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])
```

Go to Answer

Activity 2 (c) List Pairs in Fair Order

- Rewrite the function which takes a pair of positive integers and outputs a list of lists of all possible pairs in those ranges
- The output should treat each argument *fairly* — any initial prefix should have roughly the same number of instances of each argument — further, the output should be segment by each initial prefix (see example below)
- Here is an example output

```
fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
        , [(0, 1), (1, 0)]
        , [(0, 2), (1, 1), (2, 0)]
        , [(0, 3), (1, 2), (2, 1), (3, 0)]
        , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

Go to Answer

Answer 2 (a) Stop Words Filter

- Answer 2 (a) Stop Words Filter
- *Write here:*

Answer 2 (a) Stop Words Filter

- Answer 2 (a) Stop Words Filter

```
def filterStopWords(words) :
  nonStopWords \
   = [word for word in words
           if word not in stopWords]
  return nonStopWords


filterStopWordsTest \
 = filterStopWords(words) \
    == ['quick', 'brown', 'fox'
      , 'jumps', 'over', 'lazy', 'dog']
```

Go to Activity

Answer 2 (b) Transpose Matrix

- Answer 2 (b) Transpose Matrix
- *Write here:*

Answer 2 (b) Transpose Matrix

- Answer 2 (b) Transpose Matrix

```
49  def transMat(mat) :
50    rowLen = len(mat[0])
51    matTr \
52     = [[row[i] for row in mat] for i in range(rowLen)]
53    return matTr

55  transMatTestA \
56   = (transMat(matrixA)
57      == matATr)
```

- Note that a list comprehension is a valid expression as a target expression in a list comprehension
- The code assumes every row is of the same length

Go to Activity

Answer 2 (b) Transpose Matrix

- Note the differences in the list comprehensions below

```
38  matrixA \
39   = [[1, 2, 3, 4]
40     ,[5, 6, 7 ,8]
41     ,[9, 10, 11, 12]]
```

```
Python3>>> [[row[i] for row in matrixA]
...                  for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
Python3>>> [row[i] for row in matrixA
...                for i in range(4)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Python3>>> [row[i] for i in range(4)
...                for row in matrixA]
[1, 5, 9, 2, 6, 10, 3, 7, 11, 4, 8, 12]
Python3>>> [[row[i] for i in range(4)]
...                  for row in matrixA]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

Go to Activity

Answer 2 (b) Transpose Matrix

- Answer 2 (b) Transpose Matrix
- The Python NumPy package provides functions for N-dimensional array objects
- For transpose see numpy.ndarray.transpose

```
Python3>>> import numpy as np
Python3>>> ar = np.array([[1,2],[3,4]])
Python3>>> ar
array([[1, 2],
       [3, 4]])
Python3>>> arT = ar.transpose()
Python3>>> arT
array([[1, 3],
       [2, 4]])
Python3>>> ar
array([[1, 2],
       [3, 4]])
Python3>>> ar.shape
(2, 2)
```

Go to Activity

Answer 2 (c) List Pairs in Fair Order

- Answer 2 (c) List Pairs in Fair Order — first version
- Write here

```
yBiasLstTest \
 = (yBiasListing(5,5)
     == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
         , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
         , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
         , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
         , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Activity

Answer 2 (c) List Pairs in Fair Order

- Answer 2 (c) List Pairs in Fair Order
- This is the *obvious* but biased version

```
def yBiasListing(xRng,yRng) :
  yBiasLst \
   = [(x,y) for x in range(xRng)
            for y in range(yRng)]
  return yBiasLst

yBiasLstTest \
 = (yBiasListing(5,5)
     == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
         , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
         , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
         , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
         , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)])
```

Go to Activity

Answer 2 (c) List Pairs in Fair Order

- Answer 2 (c) List Pairs in Fair Order — second version
- Write here

```
fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
         , (0, 1), (1, 0)
         , (0, 2), (1, 1), (2, 0)
         , (0, 3), (1, 2), (2, 1), (3, 0)
         , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])
```

Go to Activity

Answer 2 (c) List Pairs in Fair Order

- Answer 2 (c) List Pairs in Fair Order — second version
- This works by making the sum of the coordinates the same for each prefix

```
def fairListing(xRng,yRng) :
  fairLst \
   = [(x,d-x) for d in range(yRng)
              for x in range(d+1)]
  return fairLst

fairLstTest \
 = (fairListing(5,5)
     == [(0, 0)
         , (0, 1), (1, 0)
         , (0, 2), (1, 1), (2, 0)
         , (0, 3), (1, 2), (2, 1), (3, 0)
         , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]
```

Go to Activity

Answer 2 (c) List Pairs in Fair Order

- Answer 2 (c) List Pairs in Fair Order — third version
- Write here

```
fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
        , [(0, 1), (1, 0)]
        , [(0, 2), (1, 1), (2, 0)]
        , [(0, 3), (1, 2), (2, 1), (3, 0)]
        , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

Go to Activity

Answer 2 (c) List Pairs in Fair Order

- Answer 2 (c) List Pairs in Fair Order — third version
- The *inner loop* is placed into its own list comprehension

```
def fairListingA(xRng,yRng) :
  fairLstA \
   = [[(x,d-x) for x in range(d+1)]
               for d in range(yRng)]
  return fairLstA

fairLstATest \
 = (fairListingA(5,5)
     == [[(0, 0)]
        , [(0, 1), (1, 0)]
        , [(0, 2), (1, 1), (2, 0)]
        , [(0, 3), (1, 2), (2, 1), (3, 0)]
        , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])
```

Go to Activity

Python & Haskell Tutorials

- Python tutorials:
  - Beginner's Python Tutorial
  - Python Programming
  - Non-Programmer's Tutorial for Python 3
  - Non-Programmer's Tutorial for Python 2.6
- Haskell Tutorials:
  - Haskell Wikibook
  - What I Wish I Knew When Learning Haskell
  - Haskell Meta-tutorial
  - Learn You a Haskell for Great Good
  - Real World Haskell

4.2 Python Graph Representation

```
7  from collections import namedtuple
8
9  Vertex = namedtuple('Vertex'
10                     ,['vtxName'])
11
12 Edge = namedtuple('Edge'
13                   ,['startVtx','endVtx'])
```

- This is from Python/M269TutorialGraphs2020J.py
- Reserved identifiers are shown in this color
- User defined data constructors such as Vertex and Edge are shown in that color
- Vertex is a *named tuple* with named fields — a *quick and dirty* object — recommended by Guido van Rossum (checked 16 January 2016)
- **Health Warning:** these notes may not be totally consistent with syntax colouring.

Example Graphs — Python

```
17 ta = Vertex('TA')
18 tb = Vertex('TB')
19 tc = Vertex('TC')
20 td = Vertex('TD')
21 te = Vertex('TE')
22 tf = Vertex('TF')
23 tg = Vertex('TG')
24 th = Vertex('TH')
25
26 eg01Vs = [ta,tb,tc,td,te,tf,tg,th]
27
28 eg01Es = [(ta,tb),(tg,tb),(tg,th),(tb,tc)
29          ,(tb,tf),(tf,th),(tc,td),(td,te),(te,th)]
30
31 eg01Gr = (eg01Vs, eg01Es)
32
33 eg02Es = [(ta,tb),(tb,tc),(tc,ta)] # cycles
34
35 eg02Gr = ([ta,tb,tc], eg02Es)
```

- Used ordinary tuples for edges here

Graph Service Functions — Python

```
39 def vertices(gr):
40   return gr[0]
41
42 def edges(gr):
43   return gr[1]
44
45 def esStartV(v,es):
46   return [edge for edge in es if edge[0] == v]
47
48 def esEndV(v,es):
49   return [edge for edge in es if edge[1] == v]
50
51 def esRemoveV(v,es):
52   return [edge for edge in es
53               if edge[0] != v and edge[1] != v]
```

- Choice of service function (or class methods) is a design issue — a bit of a fudge here (to avoid complexity in these notes)

```
55 def succLists(gr):
56   return [(v, esStartV(v, (edges(gr))))
57           for v in vertices(gr)]

59 def predLists(gr):
60   return [(v, esEndV(v, (edges(gr))))
61           for v in vertices(gr)]

63 def isEmptyGraph(gr):
64   return gr[0] == [] and gr[1] == []

66 def startVertices(gr):
67   return [pLst[0] for pLst in predLists(gr)
68                   if pLst[1] == []]

70 def endVertices(gr):
71   return [sLst[0] for sLst in succLists(gr)
72                   if sLst[1] == []]
```

```
74 def removeVertex(v, gr):
75   vs = gr[0]
76   vs1 = vs[:]
77   if v in vs1:
78     vs1.remove(v)
79   es = gr[1]
80   es1 = esRemoveV(v,es)
81   return (vs1,es1)
```

- Note that `vs1` at line 76 is a (shallow) copy of `vs`
- If vertices had more structure we might have to write a function to do a proper copy

4.3 Haskell Graph Representation

```
1 module M269TutorialGraphs2020J where
2   import Data.List
3   import Data.Maybe
```

1. A Haskell script starts with a module header which starts with the reserved identifier, `module` followed by the module name, `M269TutorialGraphs2020J`
2. The module name must start with an upper case letter and is the same as the file name (without its extension of *.lhs*)
3. Haskell uses *layout* (or the *off-side rule*) to determine scope of definitions, similar to Python
4. The body of the module follows the reserved identifier `where` and starts with two `import` declarations
5. These import the built-in libraries `Data.List` and `Data.Maybe`
6. We use the `sort` function from `Data.List`.
7. The `Maybe` datatype from `Data.Maybe` will be used at the end of this script to implement the graphs with data.

Example Graphs — Haskell

```
5   data Vertex = TA | TB | TC | TD | TE | TF | TG | TH
6                 deriving (Eq, Show, Read)

8   type Edge = (Vertex, Vertex)
```

```
10  type Graph = ([Vertex],[Edge])
```

- The reserved identifier `data` introduces the name of a user defined type `Vertex`
- Following the `=` are the names or constructors of the elements of the type — in this case the constructors take no arguments — these names are invented by the programmer — the only Haskell requirement is that they start with an upper case letter
- The `deriving` part of the declaration automates the production of instances of the *Type Classes* `Eq` for equality and `Show` and `Read` for writing and reading.
- The reserved identifier `type` introduces *type synonym declarations*

```
12  eg01VS = [TA, TB, TC, TD, TE, TF, TG, TH]

14  eg01ES = [(TA,TB),(TG,TB),(TG,TH),(TB,TC)
15            ,(TB,TF),(TF,TH),(TC,TD),(TD,TE),(TE,TH)]

17  eg01Gr = (eg01VS, eg01ES)

19  eg02VS = [TA, TB, TC]

21  eg02ES = [(TA,TB),(TB,TC),(TC,TA)] -- cycles

23  eg02Gr = (eg02VS, eg02ES)
```

Graph Service Functions — Haskell

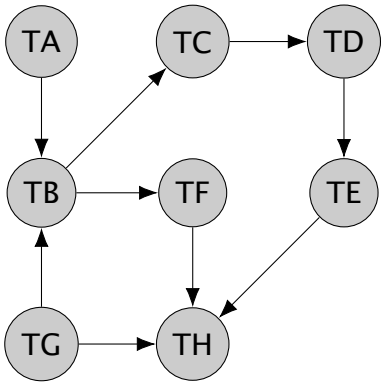
```
25  vertices :: Graph -> [Vertex]
26  vertices (vs,es) = vs

28  edges :: Graph -> [Edge]
29  edges (vs,es) = es


32  esStartV :: Vertex -> [Edge] -> [Edge]
33  esStartV v es = [(v1,v2) | (v1,v2) <- es, v1 == v]

35  esEndV :: Vertex -> [Edge] -> [Edge]
36  esEndV v es = [(v1,v2) | (v1,v2) <- es, v2 == v]

38  esRemoveV :: Vertex -> [Edge] -> [Edge]
39  esRemoveV v es = [(v1,v2) | (v1,v2) <- es,
40                              v1 /= v && v2 /= v]
```

```
42  succLists :: Graph -> [(Vertex,[Edge])]
43  succLists gr = [(v,esStartV v (edges gr))
44                  | v <- vertices gr]

46  predLists :: Graph -> [(Vertex,[Edge])]
47  predLists gr = [(v,esEndV v (edges gr))
48                  | v <- vertices gr]

50  isEmptyGraph :: Graph -> Bool
51  isEmptyGraph (vs,es) = vs == [] && es == []

53  startVertices :: Graph -> [Vertex]
54  startVertices gr = [v | (v, []) <- predLists gr]

56  endVertices :: Graph -> [Vertex]
57  endVertices gr = [v | (v, []) <- succLists gr]

59  removeVertex :: Vertex -> Graph -> Graph
60  removeVertex v (vs,es) = (delete v vs, esRemoveV v es)
```

5 Topological Sort

- A *topological sort* of a directed acyclic graph (DAG) is a linear ordering of its vertices so that for any directed edge (u, v), u comes before v in the ordering
- See en.wikipedia.org/wiki/Topological_sorting
- A topological ordering is possible for a graph if and only if it is a DAG
- Any DAG has at least one topological ordering
- If a Hamiltonian path exists (a path visiting every node in a graph exactly once) then the graph has exactly one topological ordering

Topological Sort — Example Graph



- Find all the topological orderings on this digraph

5.1 Topological Sort — Algorithm

- `topSorts` takes a graph, `gr` and returns a list of lists of vertices (all the topological sorts of the graph)
- If the graph is empty, it returns a list containing just the empty list — *Note: not just the empty list*
- Obtain a list of all the start vertices of `gr`
- If the list of start vertices is empty, then the graph has a cycle — so raise an error and stop
- Otherwise for each start vertex, `v`
  - Join it to `ts`
  - where `ts` is one of the topological sorts of `gr` with `v` removed

Topological Sort — Algorithm — Python

```
85 def topSorts(gr):
86   if isEmptyGraph(gr):
87     return [[]]
88   elif startVertices(gr) == []:
89     raise RuntimeError('Cycle in the graph')
90   else:
```

```
91      return [[v] + ts
92              for v in startVertices(gr)
93              for ts in topSorts(removeVertex(v,gr))]
```

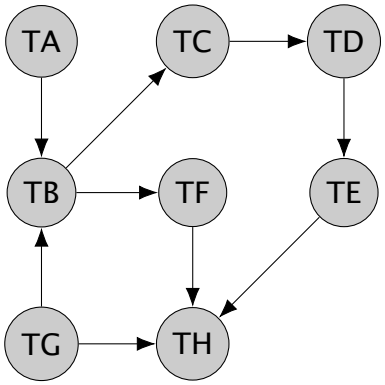
Topological Sort — Algorithm — Haskell

```
62  topSorts :: Graph -> [[Vertex]]
63  topSorts gr
64    | isEmptyGraph gr = [[]]
65    | startVertices gr == [] = error "Cycle in the graph"
66    | otherwise
67        = [v : ts | v <- startVertices gr,
68                    ts <- topSorts (removeVertex v gr)]
```

5.2 Topological Sort Example 01

Activity 3 Trace Exercise

- Trace the development of the topological sort algorithm in the following graph



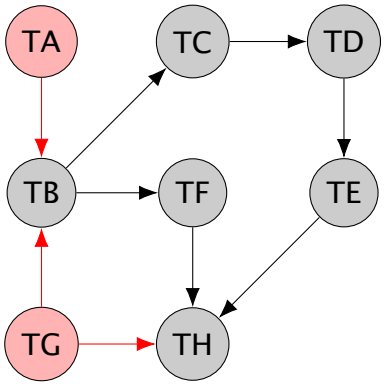
Go to Answer

Answer 3 Trace Exercise

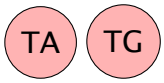
- Answer 3 Trace Exercise
- See the following slides

Go to Activity

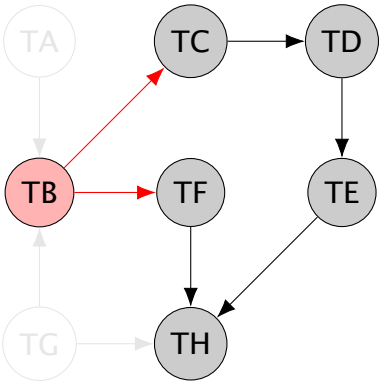
Step 1 Initial Graph



- Start vertices

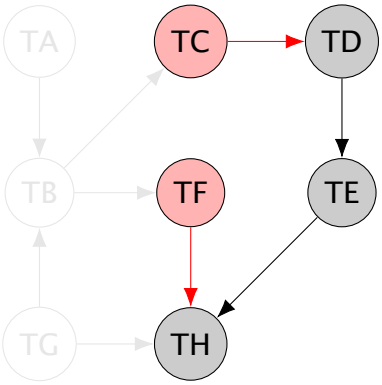


Step 2 Remove Vertices TA, TG

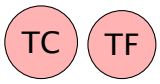


- Start vertices

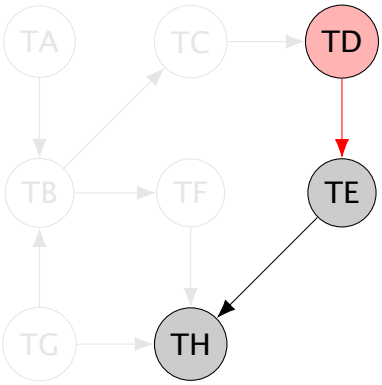
Step 3 Remove Vertex TB



- Start vertices



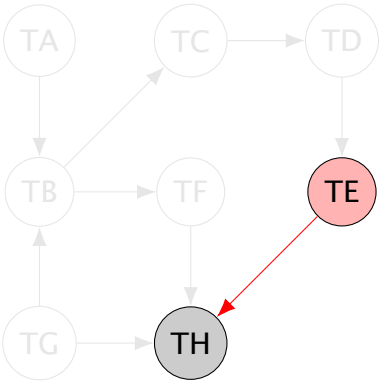
Step 4 Remove Vertices TC, TF



- Start vertices

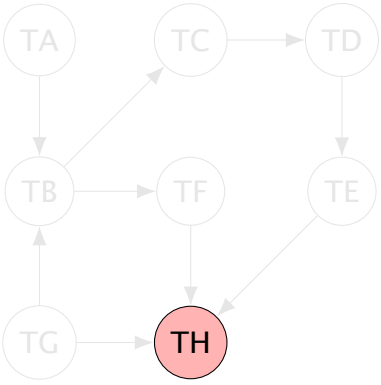
- Note: Step 4 to 6 has 4 combinations (see below)

Step 5 Remove Vertex TD



- Start vertices

Step 6 Remove Vertex TE



- Start vertices

- Step 7 would be the empty graph (not drawn)

Topological Sort — Output

```
topSortsEG01GrTest \
  = (topSorts(eg01Gr)
     == [[ta,tg,tb,tc,td,te,tf,th]
        ,[ta,tg,tb,tc,td,tf,te,th]
        ,[ta,tg,tb,tc,tf,td,te,th]
        ,[ta,tg,tb,tf,tc,td,te,th]
        ,[tg,ta,tb,tc,td,te,tf,th]
        ,[tg,ta,tb,tc,td,tf,te,th]
        ,[tg,ta,tb,tc,tf,td,te,th]
        ,[tg,ta,tb,tf,tc,td,te,th]])
```

- Note how the step 4 to 6 combinations get enumerated
- Note that a vertex `ta` would be displayed as

  `Vertex(vtxName='TA')`
- Notice the Python Explicit line joining with (`\<nl>`) and Python Implicit line joining with (`(...)`)
- The backslash (`\`) must be followed by an end of line character (`<nl>`)

```
70  topSortsEG01GrTest
71    = topSorts eg01Gr
72      == [[TA,TG,TB,TC,TD,TE,TF,TH]
73         ,[TA,TG,TB,TC,TD,TF,TE,TH]
74         ,[TA,TG,TB,TC,TF,TD,TE,TH]
75         ,[TA,TG,TB,TF,TC,TD,TE,TH]
76         ,[TG,TA,TB,TC,TD,TE,TF,TH]
77         ,[TG,TA,TB,TC,TD,TF,TE,TH]
78         ,[TG,TA,TB,TC,TF,TD,TE,TH]
79         ,[TG,TA,TB,TF,TC,TD,TE,TH]]
```

- Note how the step 4 to 6 combinations get enumerated

6 Dijkstra's Algorithm

6.1 Dijkstra's Algorithm — Description

Sources

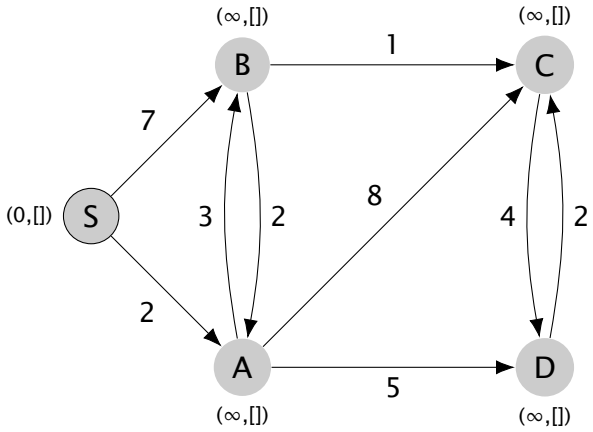
- From https://www.cse.ust.hk/~dekai/271/ (Lecture 10)
- Cormen et al. (2009, chapter 24) — Cormen et al. (2009, page 648) has a footnote explaining the origin of the term *relaxation*
- Sedgewick and Wayne (2011)
- Miller and Ranum (2011, section 7.8)
- *A Functional Graph Library* http://web.engr.oregonstate.edu/~erwig/fgl/ (Erwig, 2001)
- Rabhi and Lapalme (1999, chapter 7)

Dijkstra's Algorithm — Structured English

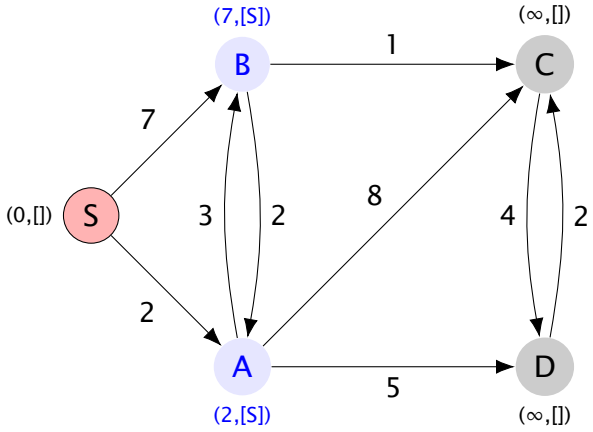
```
dijkstra(gr,weight,s)
  for u in vertices(gr)
    dist(u) = Infinity
    label(u) = Temp
  dist(s) = 0
  pred(s) = None
  q = makePriorityQ(vertices(gr))

  while not isEmptyPQ(q)
    u = extractMinPQ(q)
    for v in adj(gr,u)
      if (label(v) == Temp
          and dist(u) + weight(edge(u,v)) < dist(v))
        dist(v) = dist(u) + weight(edge(u,v))
        q = decreaseKeyPQ(q,v,dist(v))
        pred(v) = u
    label(u) = Permanent
```

6.2 Dijkstra's Algorithm Example 01

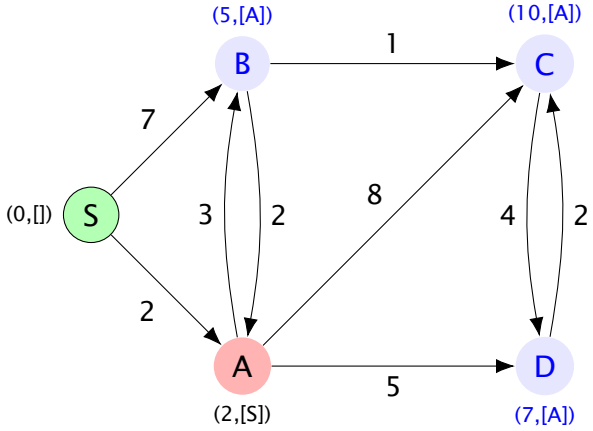
Step 0 Initialisation



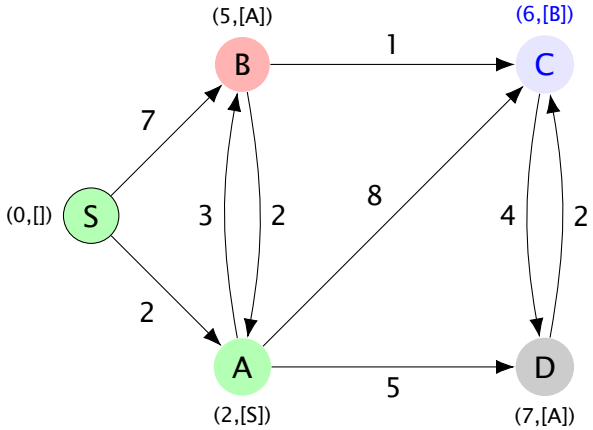
Step 1 Process S



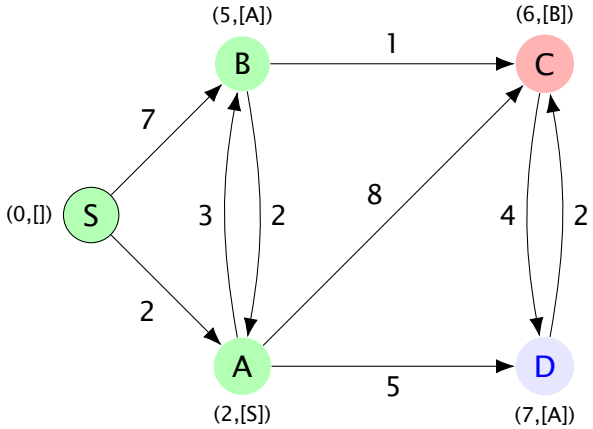
Step 2 Process A



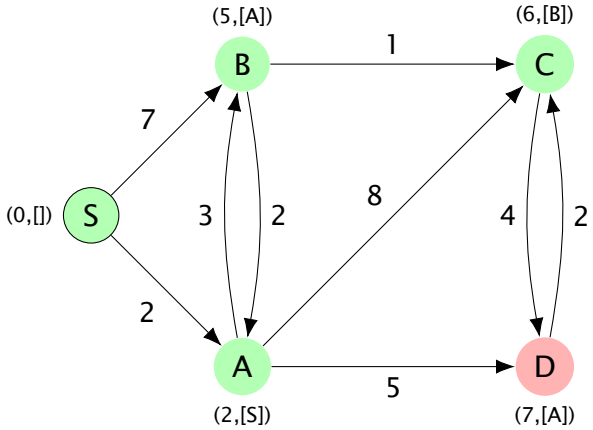
Step 3 Process B



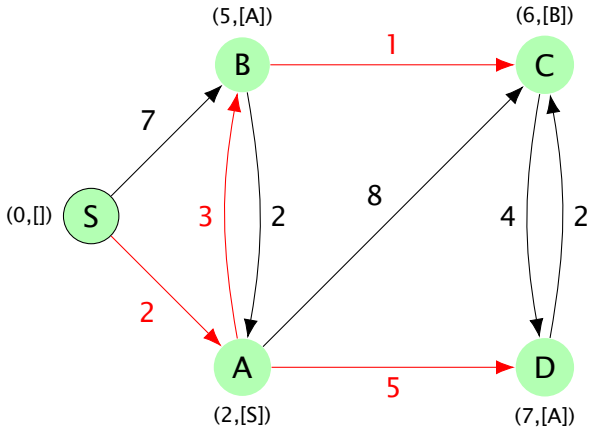
Step 4 Process C



Step 5 Process D



Shortest Path Tree Edges

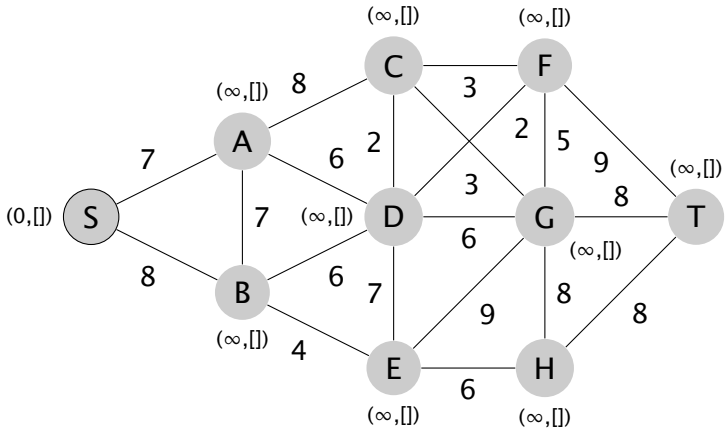


6.3 Dijkstra's Algorithm — Further points

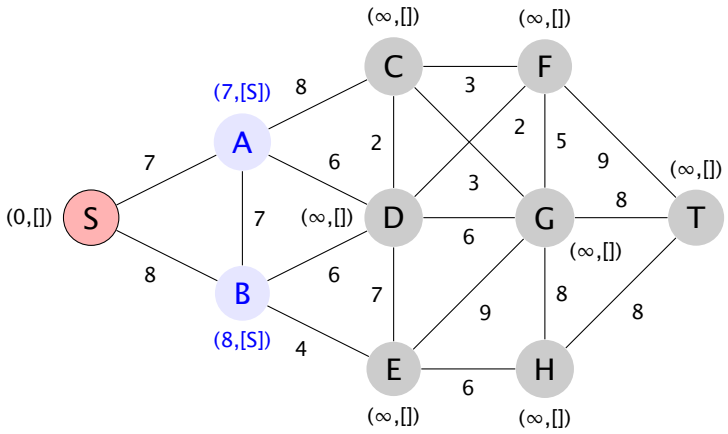
- See presentation at http://www.ukuug.org/events/agm2010/ShortestPath.pdf
- The algorithm as given assumes unique shortest paths — what if there are multiple shortest paths ? Modify the algorithm to accommodate this — change the weight on some edge to test this in the above example (change the weight of edge (A,C) to 4, for example)
- Implement a priority queue for Dijkstra's algorithm
- Material essentially comes from Cormen et al. (2009, Chp 24)

6.4 Dijkstra's Algorithm Example 02

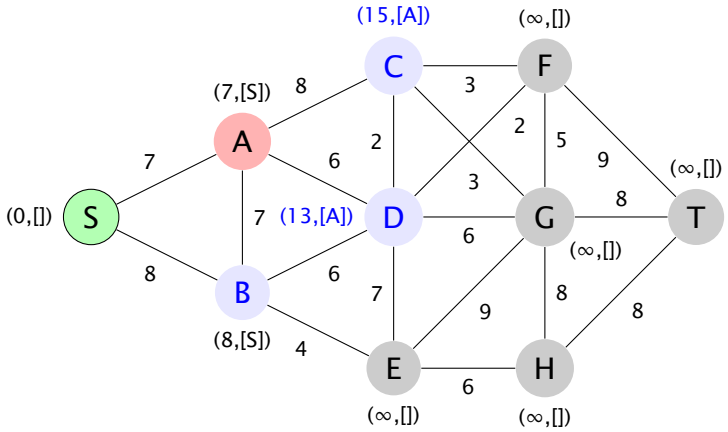
Step 0 Initialisation



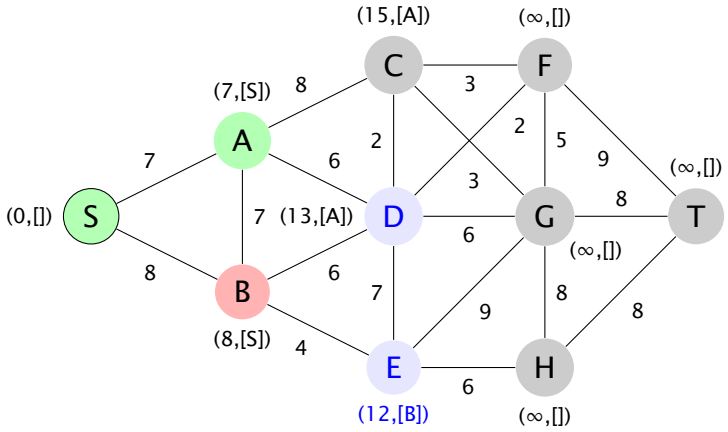
Step 1 Process S



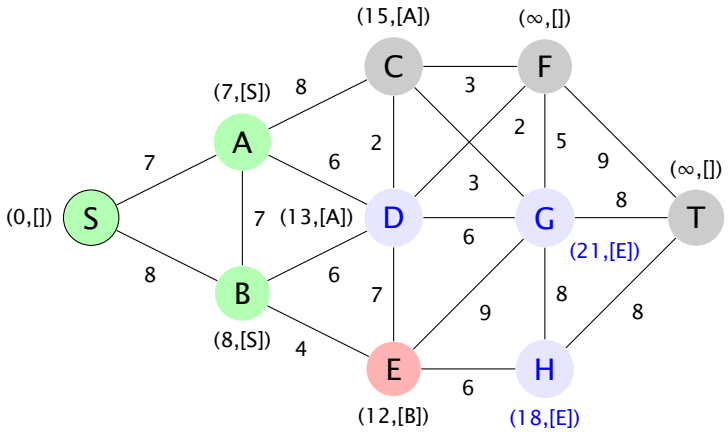
Step 2 Process A



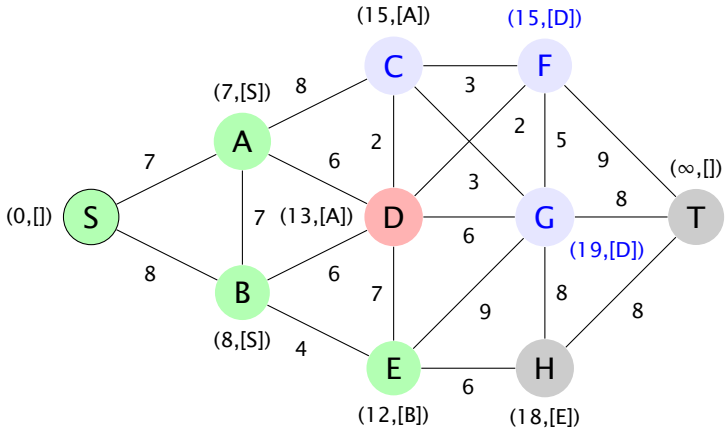
Step 3 Process B



Step 4 Process E

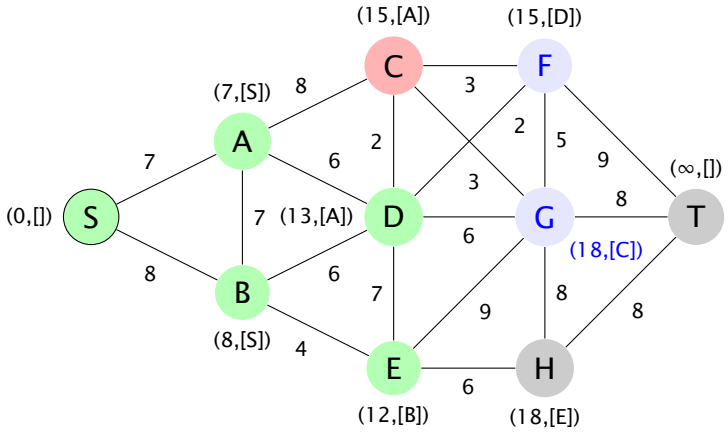


Step 5 Process D

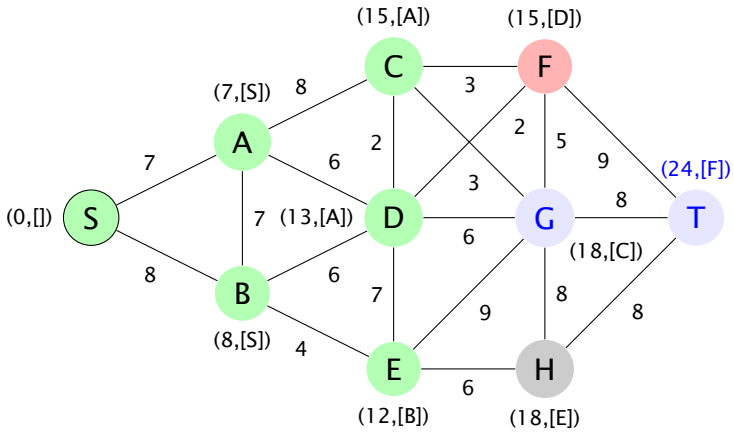


- Vertex C should have label (15,[A,D]) if we record multiple shortest routes
- How do we change the algorithm ?

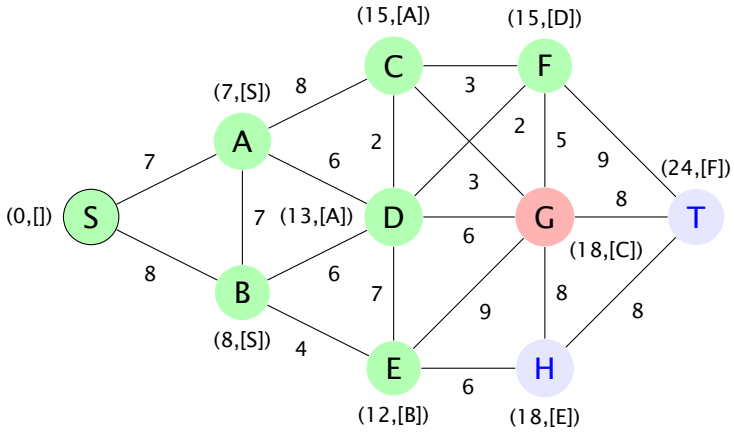
Step 6 Process C (or F)



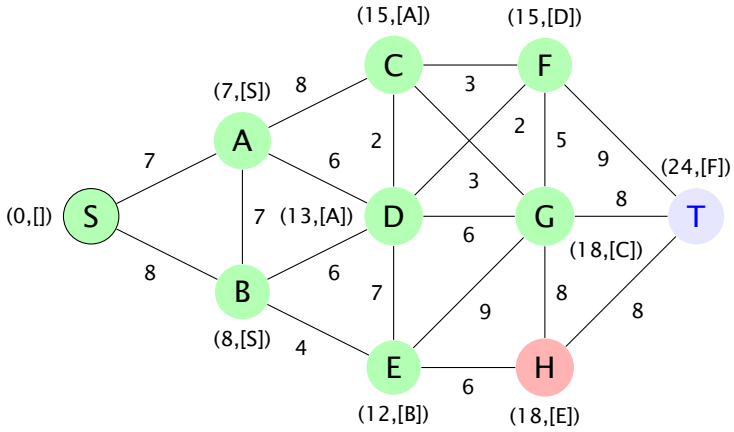
Step 7 Process F



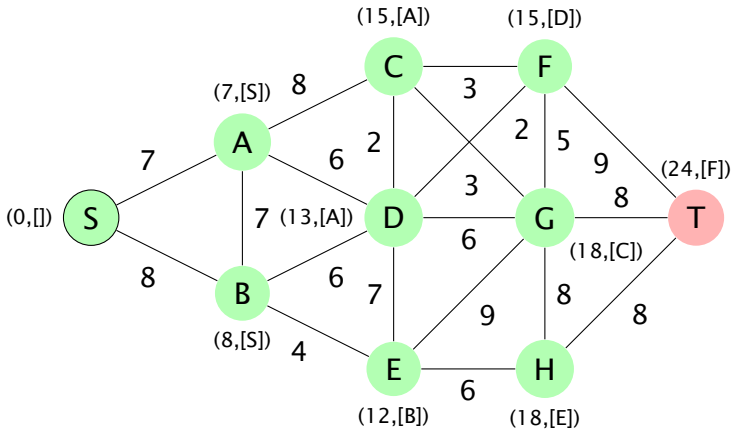
Step 8 Process G (or H)



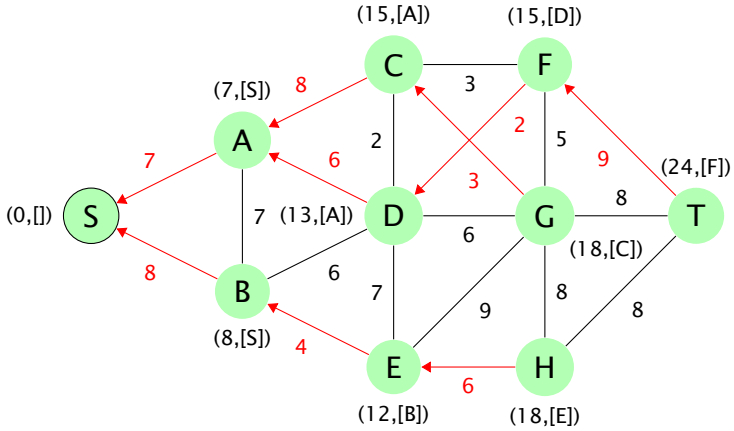
Step 9 Process H



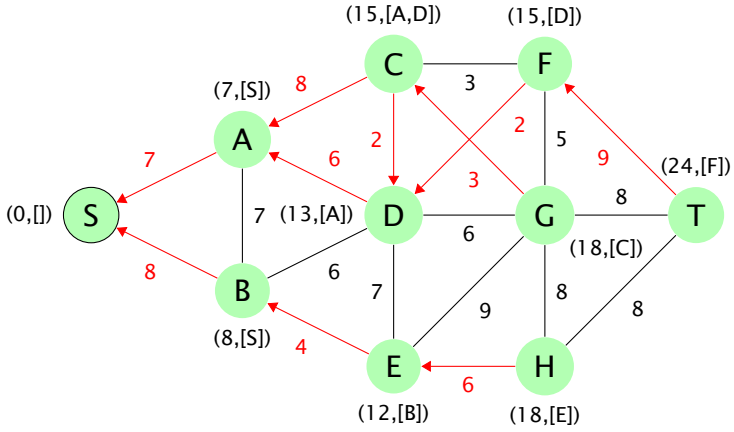
Step 10 Process T



Shortest Path Tree



Shortest Path Graph



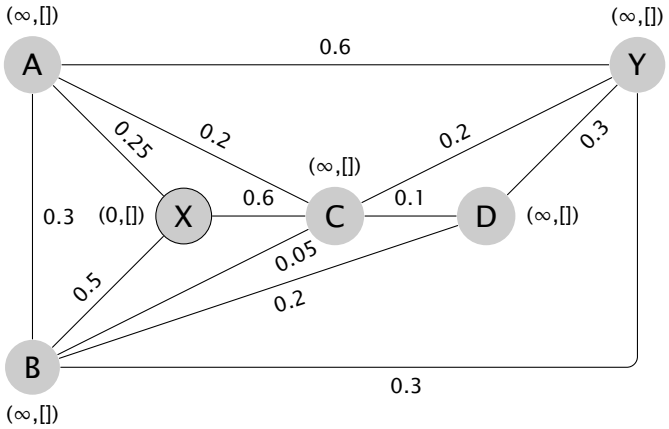
6.5 Dijkstra's Algorithm Example 03

Problem Description

- In the following graph, the weight on each edge represents the probability of failing while traversing the edge

- *Problem:* find the path that maximises the chance of traversing from X to Y

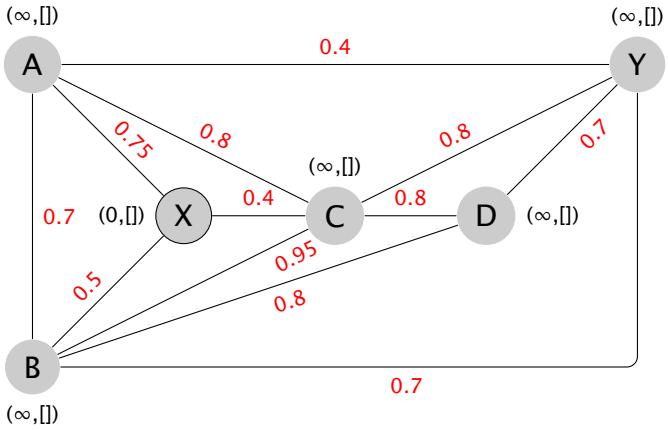
Step 0 Initialisation



Formulation as Shortest Path

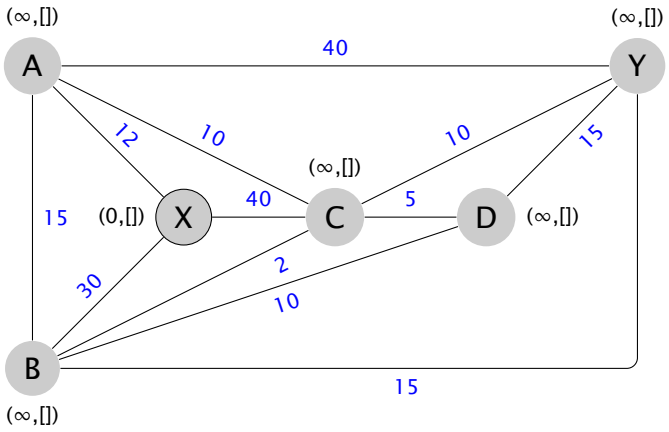
- Let $p_{(i,j)}$ be probability of failing on edge (i, j)
- The probability of not failing is $x_{(i,j)} = 1 - p_{(i,j)}$
- Over any path $x_{(i,j)}$ are independent so problem is to maximise probability of not failing $\prod_{(i,j)\in path} x_{(i,j)}$
- Equivalently, if $y_{(i,j)} = \log x_{(i,j)}$ then problem is to maximise $\sum_{(i,j)\in path} y_{(i,j)}$
- Alternatively, since $y_{(i,j)} \in (-\infty, 0]$ as $x_{(i,j)} \in [0, 1]$ then let $z_{(i,j)} = -100y_{(i,j)}$ and minimise $\sum_{(i,j)\in path} z_{(i,j)}$

Step 0 Reformulation (a)



- The numbers in red are the probabilities of not failing
- $x_{(i,j)} = 1 - p_{(i,j)}$

Step 0 Reformulation (b)



- The numbers in blue are negated scaled logs of $x_{(i,j)}$
- $z_{(i,j)} = -100 \log_{10} x_{(i,j)}$

7 Prim's Algorithm

7.1 Prim's Algorithm — Description

Prim's Algorithm — Structured English

```
prim(gr,weight,r)
  for u in vertices(gr)
    key(u) = Infinity
    label(u) = Temp
  key(r) = 0
  pred(r) = None
  q = makePriorityQ(vertices(gr))

  while not isEmptyPQ(q)
    u = extractMinPQ(q)
    for v in adj(gr,u)
      if (label(v) == Temp
          and weight(edge(u,v)) < key(v))
        key(v) = weight(edge(u,v))
        q = decreaseKeyPQ(q,v,key(v))
        pred(v) = u
    label(u) = Permanent
```

Dijkstra's and Prim's Algorithms — Comparison

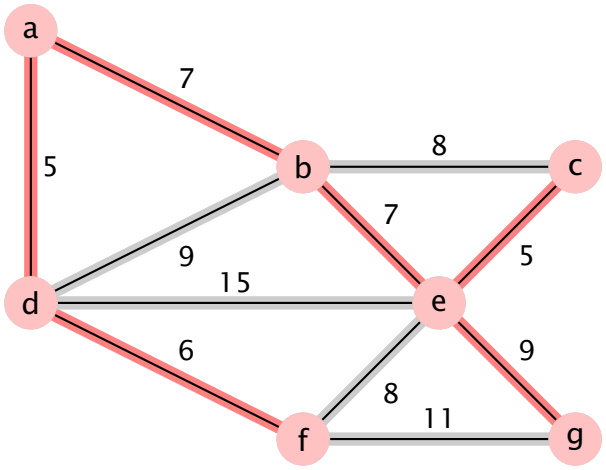
- Both are examples of *greedy* algorithms
- They choose the next best edge to add to the permanently labelled set
- The algorithms are very similar
- Process each vertex, v, in turn from a priority queue
- Examine all vertices adjacent to v and perform *relaxation*
- *relaxation* means updating the distances or keys
- For the term *relaxation* see Cormen et al. (2009, page 648) has a footnote explaining the origin of the term *relaxation*

7.2 Prim's Algorithm — Example

From http://www.texample.net/tikz/examples/prims-algorithm/ — note that the layers had to be declared inside the frame.

See https://www.cse.ust.hk/~dekai/271/ (Lecture 7) and Cormen et al. (2009, chapter 23), Sedgewick and Wayne (2011), Miller and Ranum (2011, section 7.8), (Rabhi and Lapalme, 1999, section 7.5)

Tutorial Material — Prim's algorithm



8 Dynamic Programming

8.1 Fibonacci Sequence

- The Fibonacci Numbers or Sequence invented by Leonardo Fibonacci Pisano, who also popularized the Hindu-Arabic numeral system via his 1202 book *Liber Abaci (Book of Calculations)*
- Defined by the recurrence relation
- $F_n = F_{n-1} + F_{n-2}$
- $F_0 = 0, \quad F_1 = 1$ (or originally, $F_1 = 1, \quad F_2 = 1$)
- The Fibonacci numbers have lots of interesting properties
- In programming, often used to illustrate ideas about recurrence relations and recursion

Fibonacci Sequence — Python Recursive (1)

- Recursive definitions are recursive algorithms

```
1 def fibs1(n):
2   return fibs1indent(n,0)
3
4 def fibs1indent(n,indent):
5   indentStr = '␣'*indent
```

```
  print(indentStr + 'fibs(' + str(n) + ')')

  if n == 0 :
    return 0
  elif n == 1 :
    return 1
  else:
    return (fibs1indent(n - 2, indent + 2)
          + fibs1indent(n - 1, indent + 2))
```

```
Python3>>> fibs1(5)
fibs(5)
  fibs(3)
    fibs(1)
    fibs(2)
      fibs(0)
      fibs(1)
  fibs(4)
    fibs(2)
      fibs(0)
      fibs(1)
    fibs(3)
      fibs(1)
      fibs(2)
        fibs(0)
        fibs(1)
5
Python3>>>
```

- This take 15 calls to `fibs1()`, 5 calls to `fibs(1)`, 3 calls to `fibs(0)`

Fibonacci Sequence — Python Recursive (2)

- Recursion but remembering enough to avoid most

```
def fibs2(n):
  return fibs2indent(n,0)

def fibs2indent(n,indent,first=0,second=1):
  indentStr = '␣'*indent
  print(indentStr + 'fibs(' + str(n) + ')')

  if n == 0 :
    return [first]
  else:
    return ([first]
            + fibs2indent(n - 1, indent + 2
                          , second, first + second))
```

```
Python3>>> fibs2(5)
fibs(5)
  fibs(4)
    fibs(3)
      fibs(2)
        fibs(1)
          fibs(0)
[0, 1, 1, 2, 3, 5]
Python3>>>
```

- This take 6 calls to `fibs2()`
- `fibs2()` $T(n) = T(n-1) + 1 = n + 1$
- `fibs1()` $T(n) = T(n-1) + T(n-2) + 1 = 2F_{n+1} - 1$
- `fibs1(1)` gets called F_n times
- `fibs1(0)` gets called F_{n-1} times
- So the recursion tree has $F_n + F_{n-1} = F_{n+1}$ leaves

- Since the tree is full it must have $2F_{n+1} - 1$ nodes
- (it is the sum of a geometric series)

Fibonacci Sequence — Closed-form Solution

- $F_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$
- Euler-Binet Formula
- A formula for Fib(n)
- ϕ is the Golden mean
- $\phi = \frac{1}{\phi - 1} = \frac{1+\sqrt{5}}{2}$
- Also known as the Golden ratio
- $\phi \stackrel{\text{def}}{=} \frac{a}{b} = \frac{a+b}{a}$
- From How to Compute Fibonacci Numbers? and Fibonacci Formulae

```
fib (n + k) = fib (n-1) * fib k + fib n * fib (k+1)
```

- Proof by induction

```
fib (n + 2 + k)
  = fib (n+k) + fib (n+k+1)                    -- by fib
  = fib (n-1) * fib k + fib n * fib (k+1)
    + fib n * fib k + fib (n+1) * fib (k+1)    -- by hyp
  = fib (n+1) * fib k + fib (n+2) * fib (k+1) -- by fib
```

- We now derive a method to compute `fib` in O(log n)

```
fib (2n+1) = (fib n)^2 + (fib (n+1))^2    -- (1)
fib (2n+2) = fib n * fib (n+1) + fib (n+1) * fib (n+2)
  = fib n * fib (n+1) + fib (n+1) * (fib n + fib (n+1))
  = 2 * fib n * fib (n+1) + (fib (n+1))^2  -- (2)
fib 2n = 2 * fib n * fib (n+1) - (fib n)^2 -- (3)
  -- (3) = (2) - (1)
```

```
fib2v :: Int -> (Int,Int)
fib2v 0 = (0,1)
fib2v n
  | n `mod` 2 == 0  = (c,d)
  | otherwise       = (d,c+d)
  where
  (a,b) = fib2v (n `div` 2)
  c     = 2 * a * b - a * a
  d     = a * a + b * b
```

8.2 DP Formulation

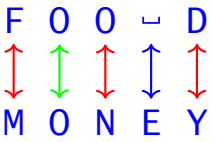
- Write a recursive algorithm for the problem
- Build solutions to the recurrence from the bottom up
  - Identify subproblems
  - Choose a data structure to memoize intermediate results

- Identify dependencies between subproblems
- Find an evaluation order so that each subproblem comes after the subproblems it depends on.
- Implement the algorithm for the evaluation order

8.3 Edit Distance

8.4 Edit Distance Definitions

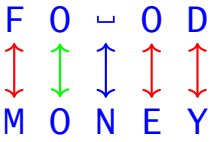
- **Edit Distance** or **Levenshtein Distance** is the minimum number of letter insertions, deletions and substitutions required to transform one word into another.
- Example FOOD → MOOD → MOND → MONED → MONEY
- A better way of displaying the transformation is in the next diagram

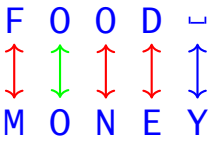


- **Exercise** find two more 4 step transformations. Are there more ?

Edit Distance — Example 1

- Two further transformations of 4 step transformations of FOOD into MONEY are





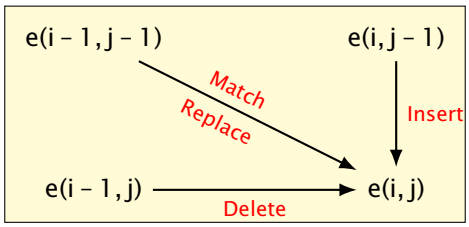
- How do we find such transformations in general ?

8.5 Edit Distance — Recursive Algorithm

- **Notation**
- s and t are names for the *start* and *target* strings
- Let s(i), t(j) denote the *prefixes* of s and t of lengths i and j
- Let s[i], t[j] denote the letters at index i and j of s and t — remember that Python and Haskell index from 0 not 1
- Let e(i, j) denote the edit distance between prefixes s(i) and t(j)

- We now describe recursively how to transform s(i) to t(j)

Edit Distance Table Dependencies



- **Deletion** — transform s(i – 1) to t(j) and delete the last character of s(i) which is s[i – 1]
- **Insertion** — transform s(i) to t(j – 1) and insert the next character required for t(j) which is t[j – 1]
- **Match/Replace** — transform s(i – 1) to t(j – 1) and match or replace the last character of s(i) with the last character of t(j) that is, s[i – 1] with t[j – 1]
- Note: you have to be careful whether you are using 0 or 1 based indexing
- The edit distance will be the minimum of the three possibilities
- This also determines the predecessor nodes (1 to 3)

Edit Distance — Base cases & Table Display

- The shortest way to convert s(i) to an empty string ϵ is by i deletions
- The shortest way to convert an empty string ϵ to t(j) is by j insertions
- **Table Display** — note that in the Haskell (but not the Python), the edit distance table is calculated with (i, j) indexing rows and columns but table is displayed with (j, i) indexing rows and columns
- That means that in the table, the row characters are the source and the column characters are the target, while in the display the column characters are the source.
- This seems to be the convention is most texts but may lead to some tricky thinking when formatting the diagrams
- **Note** The next version will change this design decision.

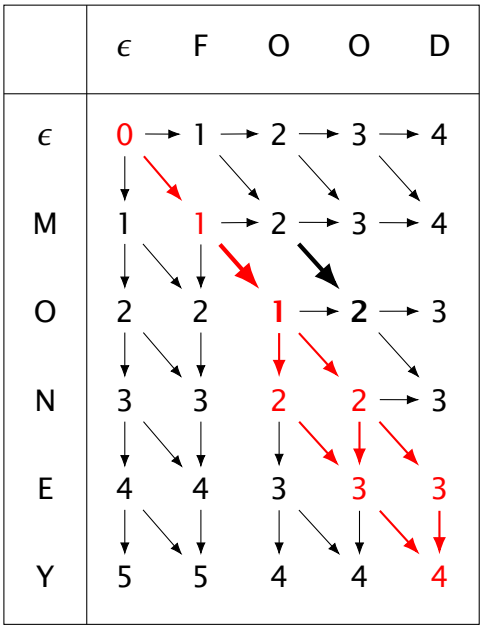
Edit Distance — Recursive Definition

$$e(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} e(i-1,j)+1 \\ e(i,j-1)+1 \\ e(i-1,j-1)+ \\ \quad \text{if } s[i-1] \neq t[j-1] \text{ then } 1 \text{ else } 0 \end{cases} & \text{otherwise} \end{cases}$$

- For simplicity we have assumed the cost of deletion, insertion, or replacement is 1 and the cost of a match is 0 (made more general in the implementation below)

- the running time of the algorithm is exponential in the length of s and t — but we do not care since are going to implement it bottom up.
- The implementation below also provides the graph of the transformations by calculating the predecessors to each node in the table.

Edit Distance — Example 1 Table



8.6 Edit Distance — Implementation

- If we calculate the edit distance table in the standard row-major order — row by row, each row from left to right
- then when we reach an entry, the entries it depends on are already available
- We develop the algorithm below in both Haskell and Python
- We have used *list comprehensions* in both for iterations — so they should look fairly similar
- *Note* both implementations could be optimised further
- *Health Warning* you are not expected to write the code for this problem — see the comments on Python_activity_5.11.py — this is more of a reading/comprehension exercise

8.7 Edit Distance — Haskell Implementation

Edit Distance — Haskell Implementation — Type Declarations

```
81  type Dist = Int
82  type Pred = (Int,Int)
83  type EditDistCell = (Dist, [Pred])
84  type EditDistTable = [[EditDistCell]]
```

- The `type` declarations introduce some type aliases

Edit Distance — Haskell Implementation — Service Functions

```
86  edCellDist :: EditDistCell -> Dist
87  edCellDist  (x, ps) = x

89  edCellPreds :: EditDistCell -> [Pred]
90  edCellPreds (x, ps) = ps
```

- `edCellDist` and `edCellPreds` take apart an `EditDistCell`

```
92  getEdCell :: EditDistTable -> (Int,Int) -> EditDistCell
93  getEdCell edTbl (i,j) = edTbl!!i!!j

95  getEdDist :: EditDistTable -> (Int,Int) -> Dist
96  getEdDist  edTbl (i,j)
97    = edCellDist  (getEdCell edTbl (i,j))

99  getEdPreds :: EditDistTable -> (Int,Int) -> [Pred]
100 getEdPreds edTbl (i,j)
101   = edCellPreds (getEdCell edTbl (i,j))
```

Edit Distance — Haskell Implementation — Costs

```
103 -- Cost of deletion, insertion, substitution
104 delC, insC, subC :: Dist
105 delC = 1
106 insC = 1
107 subC = 1
```

- The cost of deletion, insertion and replacement are defined here
- The cost of a match is assumed to be 0

Edit Distance — Haskell Implementation — Algorithm

```
109 editDistTblDP :: String -> String -> EditDistTable
110 editDistTblDP s t
111   = edTbl
112     where
113     edTbl = [ [ edTblCell (i,j) | j <- [0..length t] ]
114               | i <- [0..length s] ]
115     edTblCell (0,0) = (0,[])
116     edTblCell (i,0) = (i * delC, [(i-1,0)])
117     edTblCell (0,j) = (j * insC, [(0,j-1)])
118     edTblCell (i,j)
119       = (minD, preds)
120         where
121         possPreds = [(delD, (i-1,j))
122                     ,(insD, (i,j-1))
123                     ,(subD, (i-1,j-1))
124                     ]
125         delD = getEdDist edTbl (i-1,j) + delC
126         insD = getEdDist edTbl (i,j-1) + insC
127         subD = getEdDist edTbl (i-1,j-1)
128                + (if s!!(i-1) == t!!(j-1) then 0 else subC)
129         minD = minimum [delD, insD, subD]
130         preds = [(i,j) | (x,(i,j)) <- possPreds, x == minD]
```

Edit Distance — Haskell Implementation — Examples

```haskell
edTblTF = editDistTblDP "TREES" "FOREST"
edTblTrTF
  = transpose (editDistTblDP "TREES" "FOREST")

edTblAA = editDistTblDP "ALGORITHM" "ALTRUISTIC"
edTblTrAA
  = transpose (editDistTblDP "ALGORITHM" "ALTRUISTIC")

edTblFM = editDistTblDP "FOOD" "MONEY"
edTblTrFM
  = transpose (editDistTblDP "FOOD" "MONEY")
```

- We use `transpose` since the edit distance table is displayed with the characters of the start string in columns and the target string in the rows with i indexing the columns and j the rows
- This may lead to some tricky thinking when formatting the diagram but it seems to be the convention.

Edit Distance — Haskell Implementation (2)

```haskell
editDistTblDP01 :: String -> String -> EditDistTable
editDistTblDP01 s t
  = edTbl
    where
    edTbl = [ [ setEdTblCell (edTbl,s,t) (i,j)
              | j <- [0..length t] ]
              | i <- [0..length s] ]
```

- Defining a subsidiary function `setEdTblCell` avoids having nested `where` clauses
- `setEdTblCell` takes the edit distance table `edTbl`, the start and target strings, and a pair of table indices and returns the table cell.
- The list comprehension definition here is recursive and works because the entries are calculated in row-major order.

```haskell
setEdTblCell :: (EditDistTable, String, String)
                -> (Int, Int) -> EditDistCell
setEdTblCell (edTbl,s,t) (0,0) = (0,[])
setEdTblCell (edTbl,s,t) (i,0) = (i * delC, [(i-1,0)])
setEdTblCell (edTbl,s,t) (0,j) = (j * insC, [(0,j-1)])
setEdTblCell (edTbl,s,t) (i,j)
  = (minD, preds)
    where
    possPreds = [(delD, (i-1,j))
                ,(insD, (i,j-1))
                ,(subD, (i-1,j-1))
                ]
    delD = getEdDist edTbl (i-1,j) + delC
    insD = getEdDist edTbl (i,j-1) + insC
    subD = getEdDist edTbl (i-1,j-1)
           + (if s!!(i-1) == t!!(j-1) then 0 else subC)
    minD = minimum [delD, insD, subD]
    preds = [(i,j) | (x,(i,j)) <- possPreds, x == minD]
```

```haskell
edTblTF01 = editDistTblDP01 "TREES" "FOREST"
edTblTrTF01
  = transpose (editDistTblDP01 "TREES" "FOREST")

edTblAA01 = editDistTblDP "ALGORITHM" "ALTRUISTIC"
edTblTrAA01
  = transpose (editDistTblDP01 "ALGORITHM" "ALTRUISTIC")

edTblFM01 = editDistTblDP "FOOD" "MONEY"
edTblTrFM01
  = transpose (editDistTblDP01 "FOOD" "MONEY")
```

- `transpose` is used to switch the rows and columns for display
- This may lead to some tricky thinking about formatting the diagram

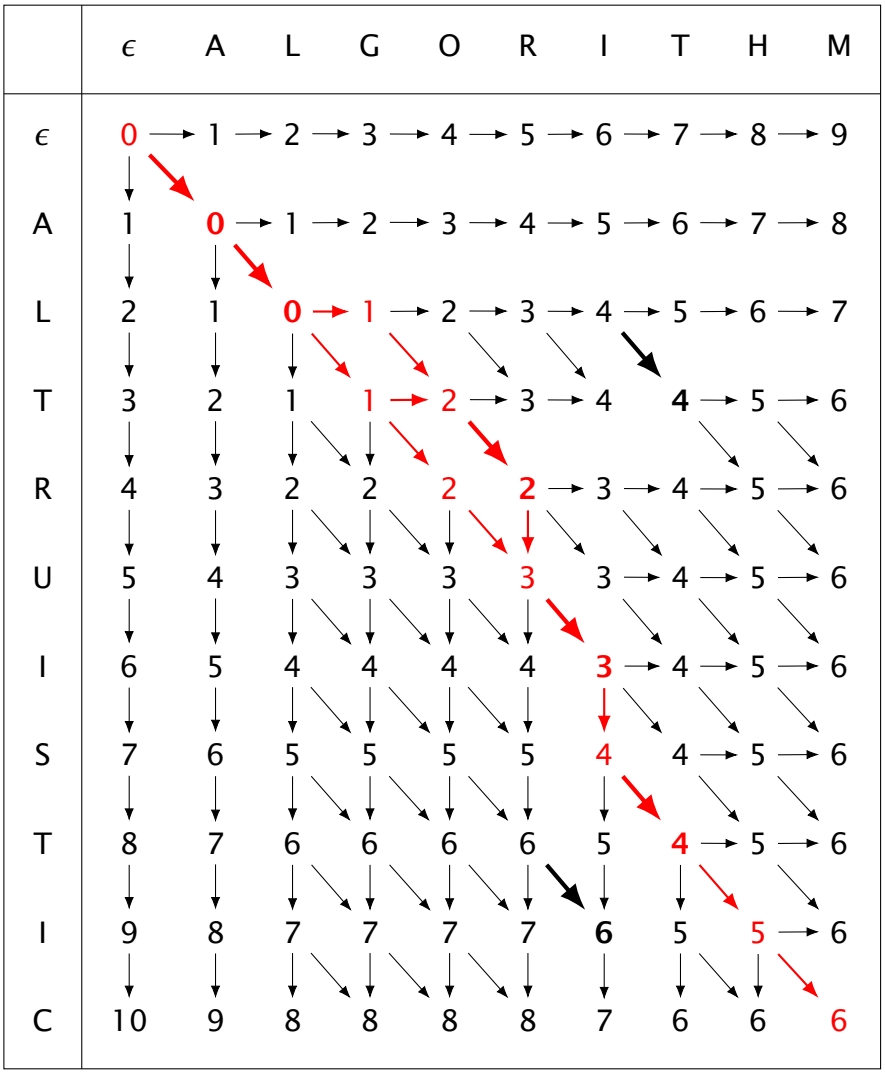
8.8 Edit Distance — Python Implementation

- The Python file `Python_activity_5.11.py` for Python activity 5.11 contains the code for the Dynamic Programming solution for the Levenshtein Edit Distance problem.
- It uses a double `for` loop to do the equivalent calculations of the edit distance table
- TODO rewrite using Python list comprehensions and compare with the Haskell version
- *Note* that the bulk of the code in 5.11 is to format the output and display one path.
- The Haskell to generate the edit table diagrams in these notes is available but not given here since it generates LaTeX TikZ/PGF which is not part of this course.

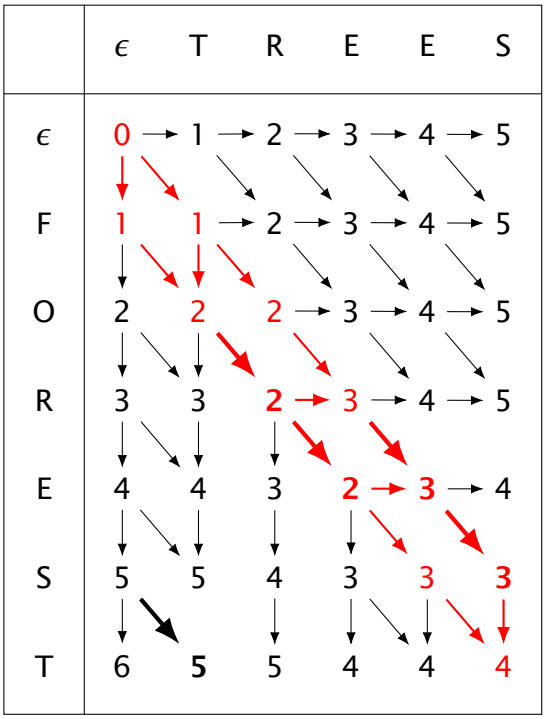
8.9 Edit Distance Examples

- For `editDistance "ALGORITHM" "ALTRUISTIC"`
  - What is the edit distance ?
  - How many different routes are there ?
  - Give two examples laid out as in the `editDistance "FOOD" "MONEY"` example
- repeat the above with `editDistance "TREES" "FOREST"` (harder!)

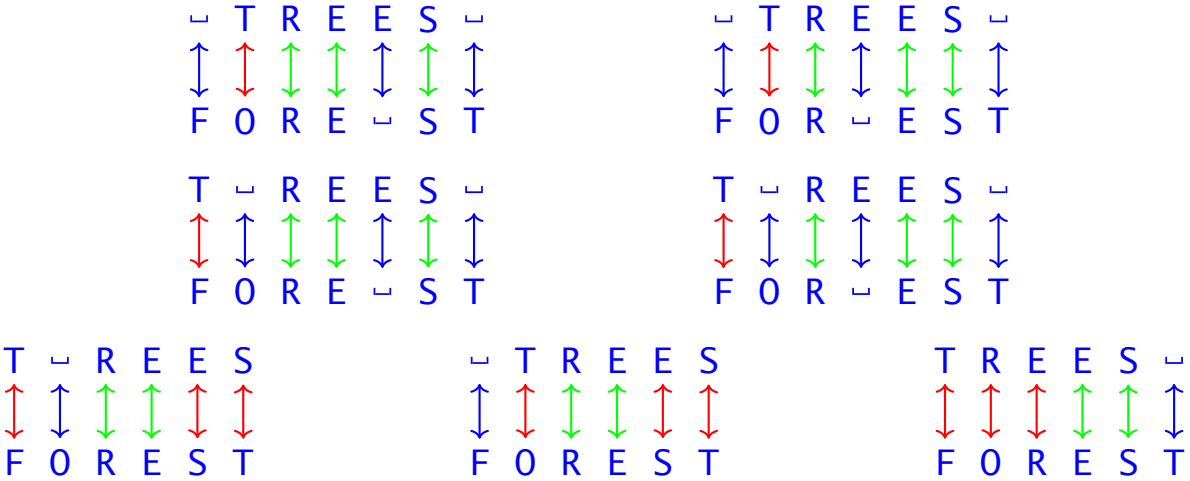
Memoization Table for `editDistance "ALGORITHM" "ALTRUISTIC"`



Memoization Table for `editDistance "TREES" "FOREST"`



- Alternative ways of representing the collection of transformation operations were given in subsection 8.4
- Here is a representation of the edit distance graph with match, delete/insert, and replace color arrows



- The transformations can also be represented as a sequence of operations
- Note that the collection of operations could be used in any order — the end result is the same

8.10 Edit Distance Sources

- Jeff Erickson Algorithms — these notes are based on the materials on this site
- Wikibooks Levenshtein (12 February 2016)
- Wikipedia: Levenshtein (12 February 2016)
- Linux Magazine March 2016 Issue 184 Better Finds — article on egrep which uses the Levenshtein algorithm
- Rosetta Code: Levenshtein Distance many implementations
- Peter Norvig How to Write a Spelling Corrector (9 February 2016)
- Stack Overflow: Edit Distance in Haskell (9 February 2016) — main source of Haskell code in this section
- *Levenshtein Algorithm* www.levenshtein.net (10 February 2016)
- *Levenshtein Demo* odur.let.rug.nl/~kleiweg/lev/ (10 February 2016)
- Wikipedia: Damerau-Levenshtein distance (12 February 2016) allows for transposition of two adjacent characters

8.11 Edit Distance Diagram Construction

- `editDistTblDP` takes two strings and returns an `EditDistTable`
- The diagrams illustrate `EditDistTable` with the cells laid out in a table with arrows
- The diagrams use the markup language LaTeX with the PGF/TikZ package
- Note: the Haskell to LaTeX and TikZ/PGF diagram code is not part of M269 and is only here for interest
- Here are several small examples

Memoization Table for `editDistTblDP "ON" "NO"`

	ϵ	O	N
ϵ	0	1	2
N	1	1	1
O	2	1	2

Memoization Table for `editDistTblDP "OH" "NO"`

	ϵ	O	H
ϵ	0	1	2
N	1	1	2
O	2	1	2

Memoization Table for `editDistTblDP "IN" "OUT"`

	ϵ	I	N
ϵ	0	1	2
O	1	1	2
U	2	2	2
T	3	3	3

LaTeX Code for `editDistTblDP "OH" "NO"`

- The LaTeX code is in a `tikzpicture` environment
- LaTeX TikZ style definitions
- Styles for nodes and arrows
- Edit path nodes and arrows are red and thick
- Free substitutions (matches) have bold nodes and ultra thick arrows

```
    [ePath/.style={red,thick}
    ,frSub/.style={font=\bfseries}
    ,efrSb/.style={ePath,frSub}
    ,orArr/.style={-Latex}
    ,frArr/.style={orArr,ultra thick}
    ,eoArr/.style={ePath,orArr}
    ,efArr/.style={ePath,frArr}
    ]
```

- TikZ Matrix code

- The double rows/columns is a hack to ease the frame placement
- See discussion at end

```
% TikZ Matrix code
\matrix (edMat) [matrix of nodes
                ,nodes in empty cells
                ,ampersand replacement=\&
                ,nodes={minimum size=\STtextNodeWidth}]
{
  \&             \&  \&             \&  \&             \&  \&             \& \\
  \&             \&  \&  $\epsilon$ \&  \&           0 \&  \&           H \& \\
  \&             \&  \&             \&  \&             \&  \&             \& \\
  \&  $\epsilon$ \&  \&           0 \&  \&           1 \&  \&           2 \& \\
  \&             \&  \&             \&  \&             \&  \&             \& \\
  \&           N \&  \&           1 \&  \&           1 \&  \&           2 \& \\
  \&             \&  \&             \&  \&             \&  \&             \& \\
  \&           0 \&  \&           2 \&  \&           1 \&  \&           2 \& \\
  \&             \&  \&             \&  \&             \&  \&             \& \\
};
```

- LaTeX TikZ Arrows code
- All arrows are given the orArr style
- TODO: modify the data structure and code to add the edit paths and free substitutions

```
% TikZ Arrows code
\draw[orArr] (edMat-4-4) -- (edMat-4-6);
\draw[orArr] (edMat-4-6) -- (edMat-4-8);

\draw[orArr] (edMat-4-4) -- (edMat-6-4);
\draw[orArr] (edMat-4-4) -- (edMat-6-6);
\draw[orArr] (edMat-6-6) -- (edMat-6-8);
\draw[orArr] (edMat-4-6) -- (edMat-6-8);

\draw[orArr] (edMat-6-4) -- (edMat-8-4);
\draw[orArr] (edMat-6-4) -- (edMat-8-6);
\draw[orArr] (edMat-8-6) -- (edMat-8-8);
\draw[orArr] (edMat-6-6) -- (edMat-8-8);
```

- LaTeX TikZ Frame code

```
% TikZ frame code
\draw (edMat-1-1.center) -- (edMat-1-9.center)
       -- (edMat-9-9.center) -- (edMat-9-1.center)
       -- cycle;
\draw (edMat-3-1.center) -- (edMat-3-9.center);
\draw (edMat-1-3.center) -- (edMat-9-3.center);
```

Generating PGF/TikZ from the Edit Table

- We generate the PGF/TikZ from editDistTblDP
- By convention we display the transpose of the table
- Remember that the transpose edit cells have lists of predecessors indexed in the original table
- This makes the computation more awkward but it appears to be the convention
- This computation does not include the edit paths nor the free substitutions
- That would be better included in the edit table computation
- TODO: Include edit paths and free substitutions in the edit table computation

Offsets, Scales and Constants

```
iOffset = 4  -- offset for original string
jOffset = 4  -- offset for target string
iScale  = 2  -- scale for original string
jScale  = 2  -- scale for target string

spc = ' '  -- space char
nl  = '\n' -- new line char
```

TikZ Arrow Code (1)

- The code for drawing arrows between nodes is built on the code for one arrow.
- edCellTrPredToArrow takes a pair of style name and matrix name,
- a pair of offset and scale data for original and target string,
- the coordinates of the destination node and a predecessor
- and returns the TikZ code for the arrow

```
edCellTrPredToArrow (styleName,matName)
  ((iOff,iScl),(jOff,jScl)) (j,i) (a,b)
 = "\\draw" ++ "[" ++ styleName ++ "]" ++ [spc]
   ++ "(" ++ matName ++ "-"
   ++ show (jOff + jScl*b)
     ++ "-" ++ show (iOff + iScl*a) ++ ")"
   ++ "␣--␣"
   ++ "(" ++ matName ++ "-"
   ++ show (jOff + jScl*j)
     ++ "-" ++ show (iOff + iScl*i) ++ ");"
   ++ "\n"
```

TikZ Arrow Code (2)

- We use edCellTrPredToArrow to build a function to take a transposed edit table and return all the arrows as a TikZ string
- edCellTrPredToArrow01 is a *helper* function to avoid repeat typing
- edCellTrPredsToArrows01 takes the list of predecessors in a cell and returns the arrows (as a string)
- edCellTrToArrows01 takes a cell and returns the arrows
- edRowTrToArrows01 takes a row and returns the arrows
- edTblTrToArrows01 takes the complete table and returns the arrows

```
edCellTrPredToArrow01
 = edCellTrPredToArrow
     ("orArr","edMat")
     ((iOffset,iScale),(jOffset,jScale))

edCellTrPredsToArrows01  (j,i) ps
 = concat (map (edCellTrPredToArrow01 (j,i)) ps)

edCellTrToArrows01  (j,i) (d,ps)
 = edCellTrPredsToArrows01 (j,i) ps

edRowTrToArrows01 j cs
 = concat [edCellTrToArrows01 (j,i) c
          | (i,c) <- zip [0..length cs - 1] cs]
          ++ "\n"

edTblTrToArrows01 edTblTr
```

```
219     = concat [edRowTrToArrows01 j cs
220               | (j,cs) <- zip [0..length edTblTr - 1]
221                               edTblTr]
```

```
223   -- Test Edit Table to Arrows
224   edTblTrToArrows01TF
225    = putStr (edTblTrToArrows01 edTblTrTF)

227   edTblTrToArrows01AA
228    = putStr (edTblTrToArrows01 edTblTrAA)

230   edTblTrToArrows01FM
231    = putStr (edTblTrToArrows01 edTblTrFM)
```

TikZ Matrix Code (1)

- edCellTrDistToNode takes a distance and returns a node string
- strToNode takes a string and returns a node string
- **Health Warning** the code below needs rewriting to make it easier to read

```
233   ndWidth = 13 -- string width of a node
234   -- nWidth = (length zStr + 1) + 2
235   -- spc = ' ' -- spc is defined above

237   ampRepStr = "\\&"

239   zStr = "$\\epsilon$"

241   matrixPreamble nm ampRepStr
242    = "\\matrix␣(" ++ nm ++ ")␣" ++ "[matrix␣of␣nodes"
243      ++ "\n" ++ nSpcs
244      ++ ",nodes␣in␣empty␣cells"
245      ++ "\n" ++ nSpcs
246      ++ ",ampersand␣replacement=" ++ ampRepStr
247      ++ "\n" ++ nSpcs
248      ++ ",nodes={minimum␣size=\\STtextNodeWidth}]"
249      ++ "\n"
250      where
251      nSpcs = replicate (11 + length nm) spc
```

```
253   -- PGF/TikZ Matrix functions

255   edCellTrDistToNode :: Int -> String
256   edCellTrDistToNode d
257    = replicate n spc ++ dStr ++ [spc]
258      where
259      dStr = show d
260      n    = ndWidth - 1 - length dStr

262   strToNode str
263    = replicate n spc ++ str ++ [spc]
264      where
265      n = ndWidth - 1 - length str
```

TikZ Matrix Code (2)

```
267   strToPrefixRow str
268    = "␣␣" ++ ampRepStr
269      ++ replicate ndWidth spc ++ ampRepStr
270      ++ "␣␣" ++ ampRepStr
271      ++ replicate ndWidth spc ++ ampRepStr
272      ++ concat ["␣␣" ++ ampRepStr
273                 ++ replicate ndWidth spc ++ ampRepStr
274                | c <- str]
275      ++ "␣\\\\\n"
276      ++ "␣␣" ++ ampRepStr
```

```
277       ++ replicate ndWidth spc ++ ampRepStr
278       ++ "␣␣" ++ ampRepStr
279       ++ replicate (ndWidth - 11) spc
280          ++ "$\\epsilon$␣"++ ampRepStr
281       ++ concat ["␣␣" ++ ampRepStr
282                  ++ replicate (ndWidth - 2) spc
283                     ++ [c] ++ [spc] ++ ampRepStr
284                 | c <- str]
285       ++ "␣\\\\\n"
```

```
287   strToSuffixRow str
288    = "␣␣" ++ ampRepStr
289      ++ replicate ndWidth spc ++ ampRepStr
290      ++ "␣␣" ++ ampRepStr
291      ++ replicate ndWidth spc ++ ampRepStr
292      ++ concat ["␣␣" ++ ampRepStr
293                 ++ replicate ndWidth spc ++ ampRepStr
294                 | c <- str]
295      ++ "␣\\\\\n"

297   colNoTrToPrefixCol zStr str j
298    = "␣␣" ++ ampRepStr ++
299        (if j == 0
300        then replicate (ndWidth - 1 - length zStr) spc
301             ++ zStr ++ [spc]
302        else replicate (ndWidth - 2) spc
303             ++ [str!!(j - 1)] ++ [spc])
304      ++ ampRepStr
```

```
306   edCellTrToNodes (d,ps)
307    = "␣␣" ++ ampRepStr
308      ++ edCellTrDistToNode d ++ ampRepStr

310   edRowTrToNodes zStr str j cs
311    = "␣␣" ++ ampRepStr
312      ++ replicate ndWidth spc ++ ampRepStr
313      ++ concat ["␣␣" ++ ampRepStr
314                 ++ replicate ndWidth spc ++ ampRepStr
315                 | c <- cs]
316      ++ "␣\\\\\n"
317      ++ colNoTrToPrefixCol zStr str j
318      ++ concat [edCellTrToNodes c | c <- cs]
319      ++ "␣\\\\\n"

321   edTblTrToNodes zStr str edTblTr
322    = concat [edRowTrToNodes zStr str j cs
323             | (j,cs) <- zip [0..length edTblTr - 1]
324                             edTblTr]
```

```
326   -- Test usage
327   edTblTrToNodesTF
328    = putStr (edTblTrToNodes zStr "FOREST" edTblTrTF)
```

TikZ Matrix Code (3)

```
330   edTblTrToMatrix nm zStr s t
331    = matrixPreamble nm ampRepStr
332      ++ "{\n"
333      ++ strToPrefixRow s
334      ++ edTblTrToNodes zStr t edTblTr
335      ++ strToSuffixRow s
336      ++ "};\n"
337      where
338      edTblTr = transpose (editDistTblDP s t)
```

```
340   -- Test code for entire matrix
341   edTblTrToMatrixTF
342    = putStr (edTblTrToMatrix "edMat" zStr "TREES" "FOREST")

344   edTblTrToMatrixAA
345    = putStr (edTblTrToMatrix "edMat" zStr "ALGORITHM" "ALTRUISTIC")
```

```
edTblTrToMatrixFM
 = putStr (edTblTrToMatrix "edMat" zStr "FOOD" "MONEY")
```

TikZ Picture Code (1)

```
-- Construction of tikzpicture

tikzPreamble
 = "\\begin{tikzpicture}" ++ [nl]
   ++ "  [ePath/.style={red,thick}" ++ [nl]
   ++ "  ,frSub/.style={font=\\bfseries}" ++ [nl]
   ++ "  ,efrSb/.style={ePath,frSub}" ++ [nl]
   ++ "  ,orArr/.style={-Latex}" ++ [nl]
   ++ "  ,frArr/.style={orArr,ultra thick}" ++ [nl]
   ++ "  ,eoArr/.style={ePath,orArr}" ++ [nl]
   ++ "  ,efArr/.style={ePath,frArr}" ++ [nl]
   ++ "  ]"

tikzEnd
 = "\\end{tikzpicture}" ++ [nl]
```

TikZ Picture Code (2)

```
tikzFrmCoord nm loc i j
 = "(" ++ nm ++ "-" ++ show j ++ "-"
    ++ show i ++ "." ++ loc ++ ")"

tikzFrame nm loc s t
 = "\\draw " ++ (tikzFrmCoord nm loc 1 1) ++ " -- "
     ++ (tikzFrmCoord nm loc xMax 1)
     ++ [nl] ++ dSpcs ++ " -- "
     ++ (tikzFrmCoord nm loc xMax yMax) ++ " -- "
     ++ (tikzFrmCoord nm loc 1 yMax)
     ++ [nl] ++ dSpcs ++ " -- cycle;" ++ [nl]
   ++ "\\draw " ++ (tikzFrmCoord nm loc 1 (jScale + 1))
     ++ " -- " ++ (tikzFrmCoord nm loc xMax (jScale + 1))
     ++ ";" ++ [nl]
   ++ "\\draw " ++ (tikzFrmCoord nm loc (iScale + 1) 1)
     ++ " -- " ++ (tikzFrmCoord nm loc (iScale + 1) yMax)
     ++ ";" ++ [nl]
   where
     xMax = iOffset + 1 + iScale * (length s)
     yMax = jOffset + 1 + jScale * (length t)
     dSpcs = replicate 6 spc
```

TikZ Picture Code (3)

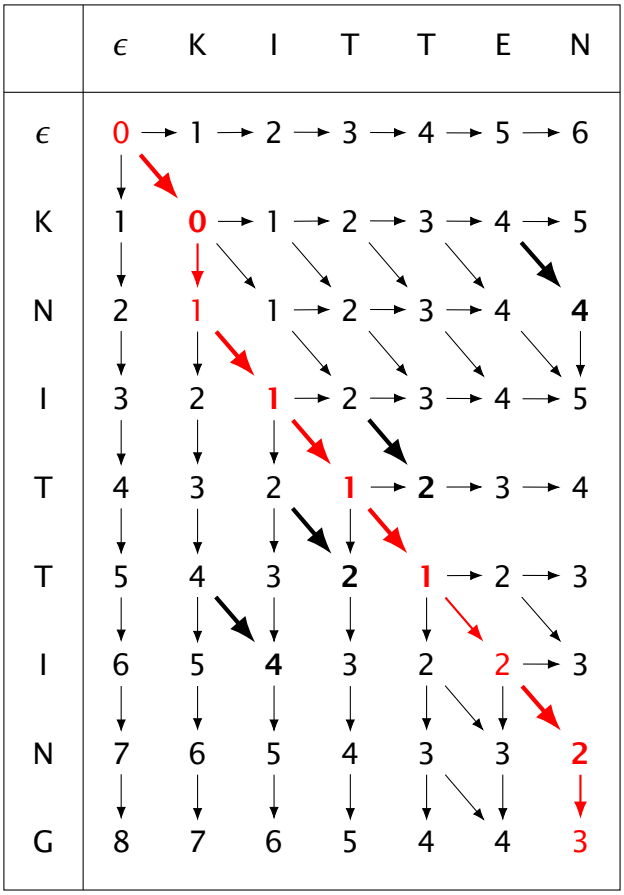
```
edTblTrToTikz nm loc zStr s t
 = tikzPreamble ++ [nl]
   ++ [nl] ++ "% TikZ Matrix code" ++ [nl]
   ++ edTblTrToMatrix nm zStr s t ++ [nl]
   ++ [nl] ++ "% TikZ Arrows code" ++ [nl]
   ++ edTblTrToArrows01 edTblTr ++ [nl]
   ++ [nl] ++ "% TikZ frame code" ++ [nl]
   ++ tikzFrame nm loc s t ++ [nl]
   ++ tikzEnd
   where
     edTblTr = transpose (editDistTblDP s t)
```

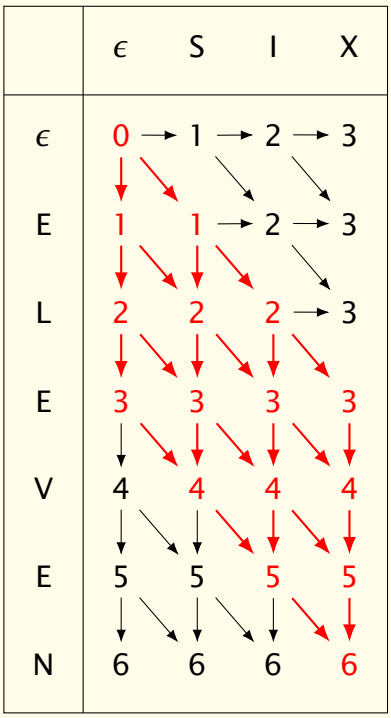
Test Code Matrix Frame

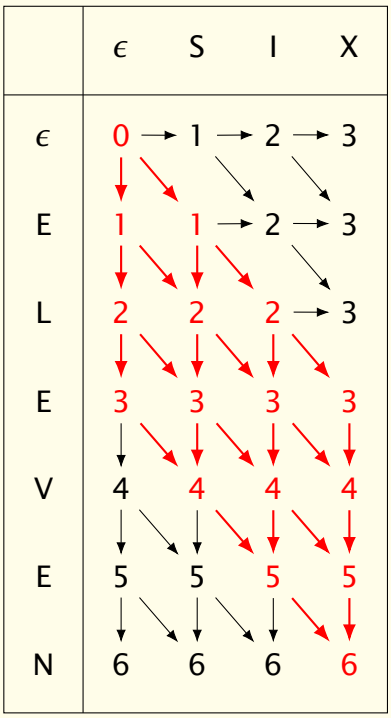
```
400  -- Test code for matrix frame lines

402  tikzFrameFM
403   = putStr (tikzFrame "edMat" "center"
404             "FOOD" "MONEY")

406  tikzFrameAA
407   = putStr (tikzFrame "edMat" "center"
408             "ALGORITHM" "ALTRUISTIC")

410  tikzFrameTF
411   = putStr (tikzFrame "edMat" "center"
412             "TREES" "FOREST")
```

Test Code TikZ Picture

```
414  -- Text code for tikzpicture

416  edTblTrToTikzFM
417   = putStr (edTblTrToTikz "edMat" "center" zStr
418             "FOOD" "MONEY")

420  edTblTrToTikzAA
421   = putStr (edTblTrToTikz "edMat" "center" zStr
422             "ALGORITHM" "ALTRUISTIC")

424  edTblTrToTikzTF
425   = putStr (edTblTrToTikz "edMat" "center" zStr
426             "TREES" "FOREST")

428  edTblTrToTikzKK
429   = putStr (edTblTrToTikz "edMat" "center" zStr
430             "KITTEN" "KNITTING")

432  edTblTrToTikzSE
433   = putStr (edTblTrToTikz "edMat" "center" zStr
434             "SIX" "ELEVEN")
```


ϵ K I T T E N
ϵ 0 1 2 3 4 5 6
K 1 0 1 2 3 4 5
N 2 1 1 2 3 4 4
I 3 2 1 2 3 4 5
T 4 3 2 1 2 3 4
T 5 4 3 2 1 2 3
I 6 5 4 3 2 2 3
N 7 6 5 4 3 3 2
G 8 7 6 5 4 4 3


ϵ S I X
ϵ 0 1 2 3
E 1 1 2 3
L 2 2 2 3
E 3 3 3 3
V 4 4 4 4
E 5 5 5 5
N 6 6 6 6


ϵ S I X
ϵ 0 1 2 3
E 1 1 2 3
L 2 2 2 3
E 3 3 3 3
V 4 4 4 4
E 5 5 5 5
N 6 6 6 6

9 M269 Library

9.1 M269 Library — Overview

- Implementations in Python of some M269 algorithms and data structures
- M269 Library contains
  - bst.py — Binary Search Tree
  - deque.py — Double-ended Queue
  - digraph.py — Directed graph
  - graph.py — Graph (discrete mathematics)
  - hash_table.py — Hash table
  - priority_queue.py — Priority queue
  - queue.py — Queue (abstract data type)
  - sort.py — Sorting algorithm
  - stack.py — Stack (abstract data type)

10 Future Work

- Saturday, 13 March 2021 tutorial online
- Wednesday, 24 March 2021 TMA02 due
- Sunday, 11 April 2021 online tutorial Unit 6 Logic
- Wednesday 28 April 2021 iCMA46 due
- Sunday, 2 May 2021 online tutorial Unit 7 Computability, Complexity
- Sunday, 16 May 2021 online tutorial exam revision
- Saturday, 22 May 2021 tutorial online
- Tuesday 25 May 2021 iCMA47 due
- Tuesday 8 June 2022 Exam
- Please email me with any requests for particular topics

11 Web Sites & References

- **Tree (Graph Theory)** http://en.wikipedia.org/wiki/Tree_(graph_theory)
- **Graph Theory** http://en.wikipedia.org/wiki/Graph_theory
- **Dekai Wu Algorithms course** https://www.cse.ust.hk/~dekai/271/
- **Jeff Erickson Algorithms** http://jeffe.cs.illinois.edu/teaching/algorithms/

11.1 Web Sites for Dynamic Programming

- Stack Exchange http://cs.stackexchange.com/questions/645/deciding-on-sub-problems-for-dynamic-programming
- *Jeff Erickson Algorithms* http://jeffe.cs.illinois.edu/teaching/algorithms/
- Cormen et al. (2009, page 407) has same example as *Jeff Erickson*
- *Edit Distance HaskellWiki* https://wiki.haskell.org/Edit_distance
- *Edit Distance in Haskell* http://stackoverflow.com/questions/5515025/edit-distance-algorithm-in-haskell-performance-tuning
- *Wikibooks Algorithm implementation — Levenshtein Distance* http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance
- *Wikipedia Levenshtein Distance* http://en.wikipedia.org/wiki/Levenshtein_distance
- *Andrew McCallum* http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/Lec6-EditDistance.pdf — beware inconsistent (i,j)
- https://web.stanford.edu/class/cs124/lec/med.pdf
- *Needleman-Wunsch algorithm* http://en.wikipedia.org/wiki/Needleman\T1\textendash algorithm
- *Peter Norvig Spell Checker*http://norvig.com/spell-correct.html (from http://stackoverflow.com/questions/2460177/edit-distance-in-python)
- LaTeX and PGF/TikZ code for table
  - *LaTeX Stack Exchange — How do I draw horizontal and vertical lines for a TikZ matrix* http://tex.stackexchange.com/questions/134209/how-do-i-draw-horizontal-and-vertical-lines-for-a-tikz-matrix
  - *LaTeX Stack Exchange — Table-like lines in TikZ matrix* http://tex.stackexchange.com/questions/204358/table-like-lines-in-tikz-matrix — but requires adjustment for non-zero column sep and row sep and for pgflinewidth

References

Bird, Richard (1998). *Introduction to Functional Programming using Haskell*. Prentice Hall, second edition. ISBN 0134843460.

Bird, Richard and Phil Wadler (1988). *Introduction to Functional Programming*. Prentice Hall, first edition. ISBN 0134841972.

Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; and Clifford Stein (2009). *Introduction to Algorithms*. MIT Press, third edition. ISBN 0262533057. URL http://mitpress.mit.edu/books/introduction-algorithms.

Dijkstra, Edsger W (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Erwig, Martin (2001). Inductive graphs and functional graph algorithms. *Journal of Functional Programming*, 11:467–492. ISSN 1469-7653. doi:10.1017/S0956796801004075. URL https://web.engr.oregonstate.edu/~erwig/fgl/.

Hudak, Paul; John Hughes; Simon Peyton Jones; and Phil Wadler (2007). A History of Haskell: Being Lazy with Class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 12–1–12–55. ACM New York, NY, USA.

Lee, Gias Kay (2013). Functional Programming in 5 Minutes. Web. http://gsklee.im, URL http://slid.es/gsklee/functional-programming-in-5-minutes.

Lutz, Mark (2011). *Programming Python*. O'Reilly, fourth edition. ISBN 0596158106. URL http://learning-python.com/books/about-pp4e.html.

Lutz, Mark (2013). *Learning Python*. O'Reilly, fifth edition. ISBN 1449355730. URL http://learning-python.com/books/about-lp5e.html.

Marlow, Simon and Simon Peyton Jones (2010). Haskell Language and Library Specification. Web. URL http://www.haskell.org/haskellwiki/Language_and_library_specification.

Miller, Bradley W. and David L. Ranum (2011). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle Associates Inc, second edition. ISBN 1590282574. URL http://interactivepython.org/courselib/static/pythonds/index.html.

O'Donnell, John; Cordelia Hall; and Rex Page (2006). *Discrete Mathematics Using a Computer*. Springer, second edition. ISBN 1846282411. URL http://www.dcs.gla.ac.uk/~jtod/discrete-mathematics/.

Okasaki, Chris (1998). *Purely Functional Data Structures*. Cambridge University Press. ISBN 0-521-63124-6.

O'Sullivan, Bryan; John Goerzen; and Donald Stewart (2008). *Real World Haskell*. O'Reilly, first edition. ISBN 0596514980. URL http://book.realworldhaskell.org/.

Rabhi, Fethi and Guy Lapalme (1999). *Algorithms: A Functional Programming Approach*. Addison-Wesley, second edition. ISBN 0201596040. URL http://www.iro.umontreal.ca/~lapalme/Algorithms-functional.html.

Sedgewick, Robert and Kevin Wayne (2011). *Algorithms*. Addison Wesley, fourth edition. ISBN 032157351X. URL http://algs4.cs.princeton.edu/home/.

Thompson, Simon (2011). *Haskell the Craft of Functional Programming*. Addison Wesley, third edition. ISBN 0201882957. URL http://www.haskellcraft.com/craft3e/Home.html.

van Rossum, Guido and Fred Drake (2011a). *An Introduction to Python*. Network Theory Limited, revised edition. ISBN 1906966133.

van Rossum, Guido and Fred Drake (2011b). *The Python Language Reference Manual*. Network Theory Limited, revised edition. ISBN 1906966141.