

Graphs and Greedy Algorithms

M269 Tutorial

Contents

1	Agenda	2
2	Adobe Connect	3
2.1	Interface	3
2.2	Settings	4
2.3	Sharing Screen & Applications	5
2.4	Ending a Meeting	5
2.5	Invite Attendees	6
2.6	Layouts	7
2.7	Chat Pods	8
2.8	Web Graphics	8
2.9	Recordings	8
3	M269 Graph Algorithms	9
3.1	Graph Definitions	9
3.2	Graph Representation	9
	Activity 1 Graph Operations	10
4	Algorithm Descriptions & Implementations	11
4.1	List Comprehensions	12
	Activity 2 List Comprehension Exercises	12
4.2	Python Graph Representation	18
4.3	Python Graph Representation from 21J	19
4.3.1	Graph Representation Choices	19
4.3.2	DiGraph Class	20
4.3.3	Weighted DiGraph Class	22
4.3.4	Undirected Graph Class	24
4.3.5	Weighted Undirected Graph Class	25
4.3.6	Drawing Graphs	27
4.3.7	Enumerations: Subsequences, Combinations	28
5	Topological Sort	30
5.1	Topological Sort — Algorithm	30
5.2	Topological Sort Example 01	31
	Activity 3 Trace Exercise	31
6	Dijkstra's Algorithm	34
6.1	Dijkstra's Algorithm — Description	34
6.2	Dijkstra's Algorithm Example 01	35
6.3	Dijkstra's Algorithm — Further points	37
6.4	Dijkstra's Algorithm Example 02	37
6.5	Dijkstra's Algorithm Example 03	41
7	Prim's Algorithm	43

7.1 Prim's Algorithm — Description	43
7.2 Prim's Algorithm — Example	44
8 Greedy Algorithms	44
8.1 Interval Scheduling	44
9 Future Work	47
10 Web Sites & References	47
10.1 Shortest Paths	48
10.2 Web Sites for Dynamic Programming	48
References	49
Python Code Index	51
Pseudocode Index	52
M269 DiGraph Code Index	53
M269 Weighted DiGraph Code Index	54
M269 Undirected Graph Code Index	55
M269 Weighted Undirected Graph Code Index	56
Diagrams Index	57

1 Agenda

- Welcome and introductions
- Session on M269 Graph, Greedy & DP Algorithms
- 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/M269FolderSync/M269TutorialNotes/M269Tutoria05
- **Recording**   ✓

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 L^AT_EX — all these slides and notes
 - Mac OS X
- *Learning style* — I read the manual before using the software

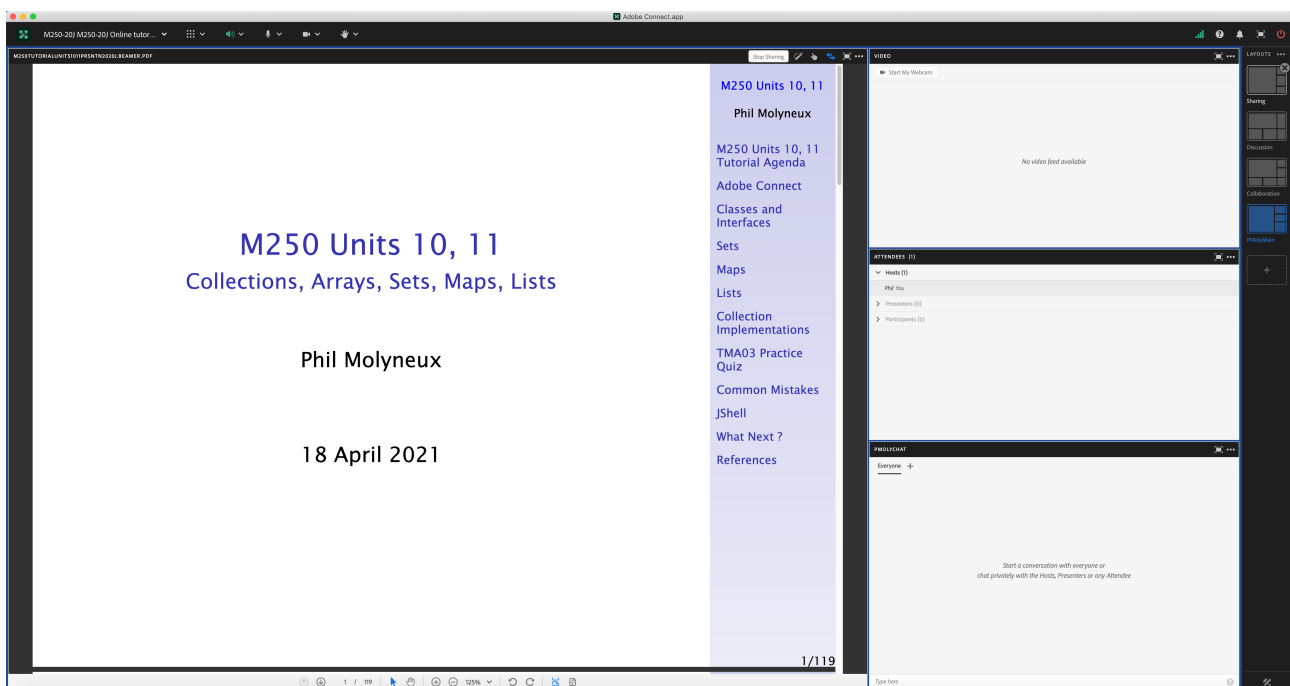
Introductions — You

- Name ?
- New topics last month ?
- M269 Graph. Greedy & Dynamic Programming 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)

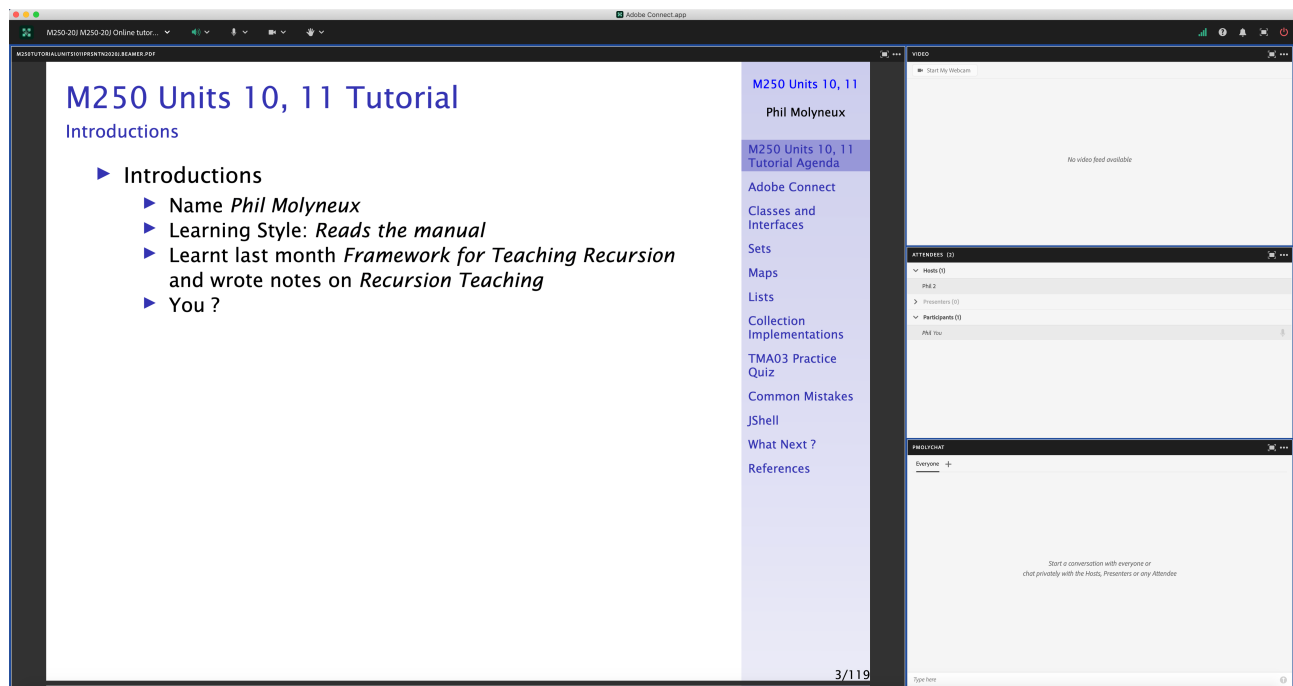
2 Adobe Connect Interface and Settings

2.1 Adobe Connect Interface

Adobe Connect Interface — Host View



Adobe Connect Interface — Participant View



2.2 Adobe Connect Settings

Adobe Connect — Settings

- **Everybody** **Menu bar** **Meeting** **Speaker & Microphone Setup**
- **Menu bar** **Microphone** **Allow Participants to Use Microphone** ✓
- Check Participants see the entire slide including slide numbers bottom right **Workaround**
 - **Disable Draw** **Share pod** **Menu bar** **Draw icon**
 - **Fit Width** **Share pod** **Bottom bar** **Fit Width icon** ✓
- **Meeting** **Preferences** **General** **Host Cursor** **Show to all attendees**
- **Menu bar** **Video** **Enable Webcam for Participants** ✓
- Do not *Enable single speaker mode*
- Cancel hand tool
- Do not enable green pointer
- **Recording** **Meeting** **Record Session** ✓
- **Documents** Upload PDF with drag and drop to share pod
- Delete **Meeting** **Manage Meeting Information** **Uploaded Content** and **check filename** **click on delete**

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 — 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 blue 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**

2.4 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.
- **Delete File Upload** Meeting > Manage Meeting Information > Uploaded Content tab select file(s) and click Delete
- **Attendance Report** see course Web site for joining room

2.5 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](#)
- Click on the link sent in email from the Host
- Get the following on a Web page
- As *Guest* enter your name and click on Enter Room

**Adobe Connect**

M269-21J Online tutorial room
London/SE (1,13) CG [2311] (M269-21J)
(1)

Guest Registered User

Name
Guest Name

By entering a Name & clicking "Enter Room", you agree that you have read and accept the [Terms of Use](#) & [Privacy Policy](#).

Enter Room

- See the *Waiting for Entry Access* for *Host* to give permission

**Adobe Connect**

Waiting for Entry Access

This is a private meeting. Your request to enter has been sent to the host. Please wait for a response.

- *Host* sees the following dialog in *Adobe Connect* and grants access

Guest entry ⓘ
1 guest would like to enter the room. Do you want to allow or deny entry to incoming guests?

Guest Name (guest) ✓ ✕ ✉

Allow everyone Deny everyone Close

2.6 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 *PMolyShare*
- **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 — rename 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)
- **Auxiliary Layouts** name *PMolyAuxOn*
 - Create new Share pod
 - Use existing Chat pod
 - Use same Video and Attendance pods

2.7 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**

2.8 Graphics Conversion for Web

- Conversion of the screen snaps for the installation of Anaconda on 1 May 2020
- Using GraphicConverter 11
- **File** > **Convert & Modify** > **Conversion** > **Convert**
- Select files to convert and destination folder
- Click on **Start selected Function** or  + 

2.9 Adobe Connect Recordings

- **Menu bar** > **Meeting** > **Preferences** > **Video**
- **Aspect ratio** > **Standard (4:3)** (not Wide screen (16:9) default)

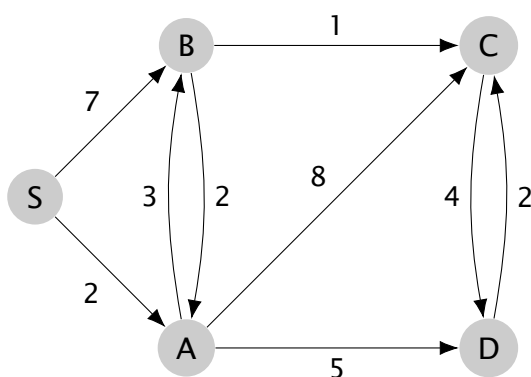
- **Video quality** **Full HD** (1080p not High default 480p)
- **Recording** **Menu bar** **Meeting** **Record Session** ✓
- **Export Recording**
- **Menu bar** **Meeting** **Manage Meeting Information**
- **New window** **Recordings** **check Tutorial** **Access Type button**
- **check Public** **check Allow viewers to download**
- **Download Recording**
- **New window** **Recordings** **check Tutorial** **Actions** **Download File**

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 $w :: E \rightarrow \mathbb{R}$ which maps edges to real numbers.
- If $e = (u, v)$ we write $w(u, v)$ for $w(e)$

Example Digraph


[ToC](#)

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 $|V| \times |V|$ 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.

[ToC](#)

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

```

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']

```

- Notice the [Python Explicit line joining](#) with (`\<n1>`) and [Python Implicit line joining](#) with (`(...)`)
- The [backslash](#) (`\`) must be followed by an [end of line character](#) (`<n1>`)
- 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

```

38 matrixA \
39     = [[1, 2, 3, 4]
40         , [5, 6, 7, 8]
41         , [9, 10, 11, 12]]
43 matATr \
44     = [[1, 5, 9]
45         , [2, 6, 10]
46         , [3, 7, 11]
47         , [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

```

68 yBiasLstTest \
69 = (yBiasListing(5,5)
70   == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
71       , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
72       , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
73       , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
74       , (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

```

81 fairLstTest \
82 = (fairListing(5,5)
83   == [(0, 0)
84       , (0, 1), (1, 0)
85       , (0, 2), (1, 1), (2, 0)
86       , (0, 3), (1, 2), (2, 1), (3, 0)
87       , (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

```

94 fairLstATest \
95 = (fairListingA(5,5)
96   == [[(0, 0)]
97       , [(0, 1), (1, 0)]
98       , [(0, 2), (1, 1), (2, 0)]
99       , [(0, 3), (1, 2), (2, 1), (3, 0)]
100      , [(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

```

24 def filterStopWords(words) :
25     nonStopWords \
26     = [word for word in words
27         if word not in stopWords]

```

```

28     return nonStopWords

31 filterStopWordsTest \
32     = filterStopWords(words) \
33     == ['quick', 'brown', 'fox'
34         , '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

```

69 yBiasLstTest \
70   = (yBiasListing(5,5)
71       == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
72           , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
73           , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
74           , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
75           , (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

```

63 def yBiasListing(xRng,yRng) :
64     yBiasLst \
65     = [(x,y) for x in range(xRng)
66         for y in range(yRng)]
67     return yBiasLst
69 yBiasLstTest \
70   = (yBiasListing(5,5)
71       == [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)
72           , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
73           , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
74           , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
75           , (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

```

83 fairLstTest \
84   = (fairListing(5,5)
85       == [(0, 0)
86           , (0, 1), (1, 0)
87           , (0, 2), (1, 1), (2, 0)
88           , (0, 3), (1, 2), (2, 1), (3, 0)
89           , (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

```

77 def fairListing(xRng,yRng) :
78     fairLst \
79     = [(x,d-x) for d in range(yRng)
80         for x in range(d+1)]
81     return fairLst

83 fairLstTest \
84 = (fairListing(5,5)
85    == [(0, 0)
86        , (0, 1), (1, 0)
87        , (0, 2), (1, 1), (2, 0)
88        , (0, 3), (1, 2), (2, 1), (3, 0)
89        , (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

```

97 fairLstATest \
98 = (fairListingA(5,5)
99    == [[(0, 0)]
100        , [(0, 1), (1, 0)]
101        , [(0, 2), (1, 1), (2, 0)]
102        , [(0, 3), (1, 2), (2, 1), (3, 0)]
103        , [(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

```

91 def fairListingA(xRng,yRng) :
92     fairLstA \
93     = [[(x,d-x) for x in range(d+1)]
94         for d in range(yRng)]
95     return fairLstA

97 fairLstATest \
98 = (fairListingA(5,5)
99    == [[(0, 0)]
100        , [(0, 1), (1, 0)]
101        , [(0, 2), (1, 1), (2, 0)]
102        , [(0, 3), (1, 2), (2, 1), (3, 0)]
103        , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]])

```

[Go to Activity](#)[ToC](#)

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
9 Vertex = namedtuple('Vertex'
10                    , ['vtxName'])
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')

26 eg01Vs = [ta, tb, tc, td, te, tf, tg, th]

28 eg01Es = [(ta, tb), (tg, tb), (tg, th), (tb, tc)
29           , (tb, tf), (tf, th), (tc, td), (td, te), (te, th)]

31 eg01Gr = (eg01Vs, eg01Es)

33 eg02Es = [(ta, tb), (tb, tc), (tc, ta)] # cycles
35 eg02Gr = ([ta, tb, tc], eg02Es)

```

- Used ordinary tuples for edges here

Graph Service Functions — Python

```

39 def vertices(gr):
40     return gr[0]

42 def edges(gr):
43     return gr[1]

45 def esStartV(v,es):
46     return [edge for edge in es if edge[0] == v]

48 def esEndV(v,es):
49     return [edge for edge in es if edge[1] == v]

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

[ToC](#)

4.3 Python Graph Representation from 21J

4.3.1 Graph Representation Choices

- A graph is a pair of sets of nodes and edges, possibly with information attached to nodes and edges such as labels, weights, durations or distances — this is the mathematical view of graphs
- Algorithms also need to consider representations for the efficiency of the operations — M269 discusses several graph representations:
 - *Edge list representation*

- *Adjacency matrix representation*
- *Adjacency list representation*
- The implementation is given for *directed graphs* or *digraphs* and *undirected graphs* using adjacency list representations

[ToC](#)

4.3.2 DiGraph Class

- The following code is from [M269TutorialGraphs2021JDiGraph.py](#) which is from [m269_digraph.py](#) modified only for layout

```

10 import networkx
11 from typing import Hashable

13 class DiGraph:
14     """A directed graph with hashable node objects.

16     Edges are between different nodes.
17     There's at most one edge from one node to another.
18     """

```

```

20 def __init__(self):
21     self.out = dict() # a map of nodes to their out-neighbours

23 def has_node(self, node: Hashable) -> bool:
24     """Return True if and only if the graph has the node."""
25     return node in self.out

27 def has_edge(self, start: Hashable, end: Hashable) -> bool:
28     """Return True if and only if edge start -> end exists.

30     Preconditions: self.has_node(start) and self.has_node(end)
31     """
32     return end in self.out[start]

```

```

34 def add_node(self, node: Hashable) -> None:
35     """Add the node to the graph.

37     Preconditions: not self.has_node(node)
38     """
39     self.out[node] = set()

41 def add_edge(self, start: Hashable, end: Hashable) -> None:
42     """Add edge start -> end to the graph.

44     If the edge already exists, do nothing.

46     Preconditions:
47     self.has_node(start) and self.has_node(end) and start != end
48     """
49     self.out[start].add(end)

```

- Note `add` is a `set` method that does not raise an error if the argument is a node already present

```

51 def remove_node(self, node: Hashable) -> None:
52     """Remove the node and all its attached edges.

54     Preconditions: self.has_node(node)
55     """
56     self.out.pop(node)
57     for start in self.out:
58         self.remove_edge(start, node)

60 def remove_edge(self, start: Hashable, end: Hashable) -> None:
61     """Remove edge start -> end from the graph.

```

```

63     If the edge doesn't exist, do nothing.
64
65     Preconditions: self.has_node(start) and self.has_node(end)
66     """
67     self.out[start].discard(end)

```

- Note `discard` is a `set` method that does not raise an error if the argument is a node that is not present
- `pop` is a `dict` and a `set` operation
- Note this version of `remove_node` has a bug — remove the edges to the node first

```

69 def nodes(self) -> set:
70     """Return the graph's nodes."""
71     all_nodes = set()
72     for node in self.out:
73         all_nodes.add(node)
74     return all_nodes
75
76 def edges(self) -> set:
77     """Return the graph's edges as a set of pairs (start, end)."""
78     all_edges = set()
79     for start in self.out:
80         for end in self.out[start]:
81             all_edges.add( (start, end) )
82     return all_edges

```

```

84 def out_neighbours(self, node: Hashable) -> set:
85     """Return the out-neighbours of the node.
86
87     Preconditions: self.has_node(node)
88     """
89     return set(self.out[node]) # return a copy
90
91 def out_degree(self, node: Hashable) -> int:
92     """Return the number of out-neighbours of the node.
93
94     Preconditions: self.has_node(node)
95     """
96     return len(self.out[node])

```

```

98 def in_neighbours(self, node: Hashable) -> set:
99     """Return the in-neighbours of the node.
100
101     Preconditions: self.has_node(node)
102     """
103     start_nodes = set()
104     for start in self.out:
105         if self.has_edge(start, node):
106             start_nodes.add(start)
107     return start_nodes
108
109 def in_degree(self, node: Hashable) -> int:
110     """Return the number of in-neighbours of the node.
111
112     Preconditions: self.has_node(node)
113     """
114     return len(self.in_neighbours(node))

```

```

116 def neighbours(self, node: Hashable) -> set:
117     """Return the in- and out-neighbours of the node.
118
119     Preconditions: self.has_node(node)
120     """
121     return self.out_neighbours(node).union(self.in_neighbours(node))
122
123 def degree(self, node: Hashable) -> int:
124     """Return the number of in- and out-going edges of the node.
125
126     Preconditions: self.has_node(node)
127     """
128     return self.in_degree(node) + self.out_degree(node)

```

```

130 def draw(self) -> None:
131     """Draw the graph."""
132     if type(self) == DiGraph:
133         graph = networkx.DiGraph()
134     else:
135         graph = networkx.Graph()
136         graph.add_nodes_from(self.nodes())
137         graph.add_edges_from(self.edges())
138         networkx.draw(graph, with_labels=True,
139             node_size=1000, node_color='lightblue',
140             font_size=12, font_weight='bold')

```

```

142 from collections import deque

144 def bfs(graph: DiGraph, start: Hashable) -> DiGraph:
145     """Return the subgraph traversed by a breadth-first search.

146     Preconditions: graph.has_node(start)
147     """
148     # changes from traversed function noted in comments
149     visited = DiGraph()
150     visited.add_node(start)
151     unprocessed = deque() # set -> deque
152     for neighbour in graph.out_neighbours(start):
153         unprocessed.append( (start, neighbour) ) # add -> append
154     while len(unprocessed) > 0:
155         edge = unprocessed.popleft() # pop -> popleft
156         previous = edge[0]
157         current = edge[1]
158         if not visited.has_node(current):
159             visited.add_node(current)
160             visited.add_edge(previous, current)
161             for neighbour in graph.out_neighbours(current):
162                 unprocessed.append( (current, neighbour) ) # add -> append
163     return visited

```

```

166 def dfs(graph: DiGraph, start: Hashable) -> DiGraph:
167     """Return the subgraph traversed by a depth-first search.

168     Preconditions: graph.has_node(start)
169     """
170     visited = DiGraph()
171     visited.add_node(start)
172     unprocessed = [] # deque -> list
173     for neighbour in graph.out_neighbours(start):
174         unprocessed.append( (start, neighbour) )
175     while len(unprocessed) > 0:
176         edge = unprocessed.pop() # popleft -> pop
177         previous = edge[0]
178         current = edge[1]
179         if not visited.has_node(current):
180             visited.add_node(current)
181             visited.add_edge(previous, current)
182             for neighbour in graph.out_neighbours(current):
183                 unprocessed.append( (current, neighbour) )
184     return visited

```

[ToC](#)

4.3.3 Weighted DiGraph Class

```

187 import math

189 class WeightedDiGraph(DiGraph):
190     """A weighted directed graph with hashable node objects.

191     Edges are between different nodes.
192     There's at most one edge from one node to another.
193     Edges have weights, which can be floats or integers.
194     """
195
197     def add_node(self, node: Hashable) -> None:

```

```

198 """Add the node to the graph.
199
200 Preconditions: not self.has_node(node)
201 """
202 self.out[node] = dict() # a map of out-neighbours to weights
203
204 def add_edge(self, start: Hashable, end: Hashable, weight: float) -> None:
205 """Add edge start -> end, with the given weight, to the graph.
206
207 If the edge already exists, set its weight.
208
209 Preconditions:
210 self.has_node(start) and self.has_node(end) and start != end
211 """
212 self.out[start][end] = weight

```

```

214 def weight(self, start: Hashable, end: Hashable) -> float:
215 """Return the weight of edge start -> end or infinity if it doesn't exist.
216
217 Preconditions: self.has_node(start) and self.has_node(end)
218 """
219 if self.has_edge(start, end):
220     return self.out[start][end]
221 else:
222     return math.inf
223
224 def remove_edge(self, start: Hashable, end: Hashable) -> None:
225 """Remove edge start -> end from the graph.
226
227 If the edge doesn't exist, do nothing.
228
229 Preconditions: self.has_node(start) and self.has_node(end)
230 """
231 if self.has_edge(start, end):
232     self.out[start].pop(end)

```

```

234 def edges(self) -> set:
235 """Return the graph's edges as a set of triples (start, end, weight)."""
236 all_edges = set()
237 for start in self.out:
238     for (end, weight) in self.out[start].items():
239         all_edges.add( (start, end, weight) )
240 return all_edges
241
242 def out_neighbours(self, node: Hashable) -> set:
243 """Return the out-neighbours of the node.
244
245 Preconditions: self.has_node(node)
246 """
247 return set(self.out[node].keys())

```

```

249 def draw(self) -> None:
250 """Draw the graph."""
251 if type(self) == WeightedDiGraph:
252     graph = networkx.DiGraph()
253 else:
254     graph = networkx.Graph()
255     graph.add_nodes_from(self.nodes())
256     for (node1, node2, weight) in self.edges():
257         graph.add_edge(node1, node2, w=weight)
258     pos = networkx.spring_layout(graph)
259     networkx.draw(graph, pos, with_labels=True,
260                 node_size=1000, node_color='lightblue',
261                 font_size=12, font_weight='bold')
262     networkx.draw_networkx_edge_labels(graph, pos,
263                                     edge_labels=networkx.get_edge_attributes(graph, 'w'))

```

```

265 from heapq import heappush, heappop
266
267 def dijkstra(graph: WeightedDiGraph, start: Hashable) -> WeightedDiGraph:
268 """Return a shortest path from start to each reachable node.
269
270 Preconditions:
271 - graph.has_node(start)

```

```

272 - node objects are comparable
273 - no weight is negative
274 """
275 visited = WeightedDiGraph()
276 visited.add_node(start)

278 # create min-priority queue of tuples (cost, (A, B, weight))
279 # cost is total weight from start to B via shortest path to A
280 unprocessed = [] # min-priority queue
281 for neighbour in graph.out_neighbours(start):
282     weight = graph.weight(start, neighbour)
283     heappush(unprocessed, (weight, (start, neighbour, weight)) )

285 while len(unprocessed) > 0:
286     info = heappop(unprocessed)
287     cost = info[0]
288     edge = info[1]
289     previous = edge[0]
290     current = edge[1]
291     weight = edge[2]

293     if not visited.has_node(current):
294         visited.add_node(current)
295         visited.add_edge(previous, current, weight)
296         for neighbour in graph.out_neighbours(current):
297             weight = graph.weight(current, neighbour)
298             edge = (current, neighbour, weight)
299             heappush(unprocessed, (cost + weight, edge) )
300 return visited

```

ToC

4.3.4 Undirected Graph Class

- The following code is from [M269TutorialGraphs2021JUngraph.py](#) which is from [m269_ungraph.py](#) modified only for layout

```

10 from typing import Hashable
12 class UndirectedGraph(DiGraph):
13     """An undirected graph with hashable node objects.
14
15     There's at most one edge between two different nodes.
16     There are no edges between a node and itself.
17     """

19     def add_edge(self, node1: Hashable, node2: Hashable) -> None:
20         """Add an undirected edge node1-node2 to the graph.
21
22         If the edge already exists, do nothing.
23
24         Preconditions: self.has_node(node1) and self.has_node(node2)
25         """
26         super().add_edge(node1, node2)
27         super().add_edge(node2, node1)

29     def remove_edge(self, node1: Hashable, node2: Hashable) -> None:
30         """Remove edge node1-node2 from the graph.
31
32         If the edge doesn't exist, do nothing.
33
34         Preconditions: self.has_node(node1) and self.has_node(node2)
35         """
36         super().remove_edge(node1, node2)
37         super().remove_edge(node2, node1)

39     def edges(self) -> set:
40         """Return the graph's edges as a set of pairs.
41
42         Postconditions: for every edge A-B,
43         the output has either (A, B) or (B, A) but not both

```

```

44     """
45     all_edges = set()
46     for node1 in self.out:
47         for node2 in self.out[node1]:
48             if (node2, node1) not in all_edges:
49                 all_edges.add( (node1, node2) )
50     return all_edges

52 def in_neighbours(self, node: Hashable) -> set:
53     """Return all nodes that are adjacent to the node.

54     Preconditions: self.has_node(node)
55     """
56     return self.out_neighbours(node)

59 def neighbours(self, node: Hashable) -> set:
60     """Return all nodes that are adjacent to the node.

61     Preconditions: self.has_node(node)
62     """
63     return self.out_neighbours(node)

```

```

66 def in_degree(self, node: Hashable) -> int:
67     """Return the number of edges attached to the node.

68     Preconditions: self.has_node(node)
69     """
70     return self.out_degree(node)

73 def degree(self, node: Hashable) -> int:
74     """Return the number of edges attached to the node.

75     Preconditions: self.has_node(node)
76     """
77     return self.out_degree(node)

```

[ToC](#)

4.3.5 Weighted Undirected Graph Class

```

80 class WeightedUndirectedGraph(WeightedDiGraph):
81     """A weighted undirected graph with hashable node objects.

82     There's at most one edge between two different nodes.
83     There are no edges between a node and itself.
84     Edges have weights, which may be integers or floats.
85     """
86

```

```

88 def add_edge(self, node1: Hashable, node2: Hashable, weight: float) -> None:
89     """Add an edge node1-node2 with the given weight to the graph.

90     If the edge already exists, do nothing.

91     Preconditions: self.has_node(node1) and self.has_node(node2)
92     """
93     super().add_edge(node1, node2, weight)
94     super().add_edge(node2, node1, weight)

98 def remove_edge(self, node1: Hashable, node2: Hashable) -> None:
99     """Remove edge node1-node2 from the graph.

100     If the edge doesn't exist, do nothing.

101     Preconditions: self.has_node(node1) and self.has_node(node2)
102     """
103     super().remove_edge(node1, node2)
104     super().remove_edge(node2, node1)

```

```

108 def edges(self) -> set:
109     """Return the graph's edges as a set of triples (node1, node2, weight).

110     Postconditions: for every edge A-B,

```

```

112     the output has either (A, B, w) or (B, A, w) but not both
113     """
114     all_edges = set()
115     for start in self.out:
116         for (end, weight) in self.out[start].items():
117             if (end, start, weight) not in all_edges:
118                 all_edges.add( (start, end, weight) )
119     return all_edges

```

```

121 def in_neighbours(self, node: Hashable) -> set:
122     """Return all nodes that are adjacent to the node.
123
124     Preconditions: self.has_node(node)
125     """
126     return self.out_neighbours(node)
127
128 def neighbours(self, node: Hashable) -> set:
129     """Return all nodes that are adjacent to the node.
130
131     Preconditions: self.has_node(node)
132     """
133     return self.out_neighbours(node)
134
135 def in_degree(self, node: Hashable) -> int:
136     """Return the number of edges attached to the node.
137
138     Preconditions: self.has_node(node)
139     """
140     return self.out_degree(node)
141
142 def degree(self, node: Hashable) -> int:
143     """Return the number of edges attached to the node.
144
145     Preconditions: self.has_node(node)
146     """
147     return self.out_degree(node)

```

```

149 from heapq import heappush, heappop
150
151 def prim(graph: WeightedUndirectedGraph, start: Hashable) -> WeightedUndirectedGraph:
152     """Return a minimum spanning tree of graph, beginning at start.
153
154     Preconditions:
155     - graph.has_node(start)
156     - graph is connected
157     - node objects are comparable
158     """
159     visited = WeightedUndirectedGraph()
160     visited.add_node(start)
161
162     unprocessed = []
163     for neighbour in graph.neighbours(start):
164         weight = graph.weight(start, neighbour)
165         heappush(unprocessed, (weight, start, neighbour) )

```

```

167 while len(unprocessed) > 0:
168     edge = heappop(unprocessed)
169     weight = edge[0]
170     previous = edge[1]
171     current = edge[2]
172     if not visited.has_node(current):
173         visited.add_node(current)
174         visited.add_edge(previous, current, weight)
175         for neighbour in graph.neighbours(current):
176             weight = graph.weight(current, neighbour)
177             heappush(unprocessed, (weight, current, neighbour) )
178 return visited

```

- Note that the *priority queue* `heapq` does the work of making the next smallest weight edge available — it is always the first element of `unprocessed`

4.3.6 Drawing Graphs

- The provided graph code gives two [draw](#) methods:
- For Weighted DiGraph or Undirected Graph see line 249, page 23,
- For Unweighted see line 130, page 22,
- [NetworkX](#) is a Python package for the creation, manipulation and study of networks
- [Matplotlib](#) is a Python library for creating static, animated, and interactive visualizations
- Matplotlib is used by NetworkX
- Some of the examples in these notes explicitly use `savefig(fname)` from `matplotlib.pyplot` to save the current figure to an external file
see [matplotlib.pyplot.savefig](#)
see also [matplotlib.pyplot.show](#)
- NetworkX [Drawing reference introduction](#) states that it provides basic functionality for visualising graphs but its main aim is to enable graph analysis
- The examples in M269 use the Matplotlib interface commands
- It mentions the tools [Cytoscape](#), [Gephi](#), [Graphviz](#), and for LaTeX typesetting, [PGF/TikZ](#)
- All of the packages are big and require reading the documentation — for example, the PGF/TikZ manual is 1321 pages (version 3.1.9a, 11 January 2022) (used in this document for most diagrams)
- You are not expected to learn any of the visualisation software but it may be worth noting some points about the provided [draw](#) method
- The code for the [draw](#) method is repeated on line 249, page 27

```

249 def draw(self) -> None:
250     """Draw the graph."""
251     if type(self) == WeightedDiGraph:
252         graph = networkx.DiGraph()
253     else:
254         graph = networkx.Graph()
255     graph.add_nodes_from(self.nodes())
256     for (node1, node2, weight) in self.edges():
257         graph.add_edge(node1, node2, w=weight)
258     pos = networkx.spring_layout(graph)
259     networkx.draw(graph, pos, with_labels=True,
260                  node_size=1000, node_color='lightblue',
261                  font_size=12, font_weight='bold')
262     networkx.draw_networkx_edge_labels(graph, pos,
263                                     edge_labels=networkx.get_edge_attributes(graph, 'w'))

```

- The line numbers are in gray to indicate this is a repeat of the code listing

```

258 pos = networkx.spring_layout(graph)

```

- [spring_layout](#) positions nodes using [Fruchterman-Reingold force-directed algorithm](#)
- If several layouts are possible then each run of the program will cycle through possible layouts
- To have reproducible sequences of layout use an explicit `seed=n` where n is some fixed value.

- Code in context at line 258, page 27,

```
259 networkx.draw(graph, pos, with_labels=True,
260               node_size=1000, node_color='lightblue',
261               font_size=12, font_weight='bold')
```

- `draw_networkx` draws the graph with Matplotlib with various options
- If `pos` is not specified a *spring layout* will be computed
- `with_labels` set to `True` to draw labels on the nodes
- `odelist`, `edgelist` draw only the specified nodes, edges
- Code in context at line 259, page 27

```
259 networkx.draw_networkx_edge_labels(graph, pos,
260                                  edge_labels=networkx.get_edge_attributes(graph, 'w'))
```

- `draw_networkx_edge_labels` draws edge labels
- `label_pos` position of edge label along edge (0=head, 0.5=center, 1=tail)
- Code in context at line 262, page 27,
- See also `draw_networkx_nodes`, can take a `odelist`
- See also `draw_networkx_edges`, can take an `edgelist`
- Show the graphic in the Notebook cell with the code

```
%matplotlib inline
```

- Save graphic to PNG format file in current folder

```
import matplotlib.pyplot as plt
```

```
graph = WeightedUndirectedGraph()
```

```
graph.draw()
plt.savefig("M269TMA02Q3bGraphC.png")
```

- `savefig` in `matplotlib.pyplot` saves the current figure
- See also `savefig` in `matplotlib.figure`

ToC

4.3.7 Enumerations: Subsequences, Combinations

- M269 21J TMA02 Part 2 has questions that refer to calculating subsequences (or subsets) and combinations of numbers of elements from a list
- It uses the `combinations` function from the `itertools` module of the Python *Functional Programming Modules*
- It may be useful to review some simple programs that implement the same functions, but less efficiently — it may help understand the concepts
- The following code is in the same Python script as Morse Code [M269BinaryTrees2021JMorseCode.py](#) (but probably should be with the graph algorithm notes)

- The notes here give example implementations of
 - All subsequences of a list (a surrogate for subsets)
 - Two versions of combinations
- The notes use [list comprehensions](#) — a nice alternative to loops or explicit recursion ([list comprehension reference](#))
- Subsequences of a list are all possible subsequences of elements from the list

```
AnPython3>>> subSeqsM([1,2,3])
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]]
```

```
249 def subSeqsM(xs) :
250     if xs == [] :
251         return []
252     else :
253         return ([[xs[0]] + rs if b else rs
254                 for b in [False, True]
255                 for rs in subSeqsM(xs[1:])])
```

- If the list `xs` is empty there is one subsequence: the empty list
- Otherwise you can choose the first element followed by any of the subsequences of the rest of the list
 - or ignore the first element and take any of the subsequences of the rest of the list
- See notes on List Comprehensions in the Graphs notes (mine)
- Combinations takes a list and an integer and return all subsequences of the list of that length
- Version using list comprehension instead of [map](#)

```
AnPython3>>> combsM01([1,2,3,4,5],3)
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4], [2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

```
271 def combsM01(xs, k) :
272     if k == 0 :
273         return []
274     elif xs == [] :
275         return []
276     else :
277         return ([[xs[0]] + ys for ys in combsM01(xs[1:], k-1)]
278                 + combsM01(xs[1:], k))
```

- If `k` is 0 then there is one combination, the empty list
- If the list is empty (and $k \geq 1$) then there are none
- Otherwise choose the first element followed by $(k-1)$ combinations of the rest of the list
 - or ignore the first element and choose `k` combinations of elements from the rest of the list
- Combinations takes a list and an integer and return all subsequences of the list of that length

```
AnPython3>>> combsM([1,2,3,4,5],3)
[[1, 2, 3], [1, 2, 4], [1, 2, 5], [1, 3, 4], [1, 3, 5], [1, 4, 5], [2, 3, 4], [2, 3, 5], [2, 4, 5], [3, 4, 5]]
```

```

258 def combsM(xs, k) :
259     if k == 0 :
260         return [[]]
261     elif xs == [] :
262         return []
263     else :
264         return (list(map(lambda ys : ([xs[0]] + ys), combsM(xs[1:], k-1)))
265                + combsM(xs[1:], k))

```

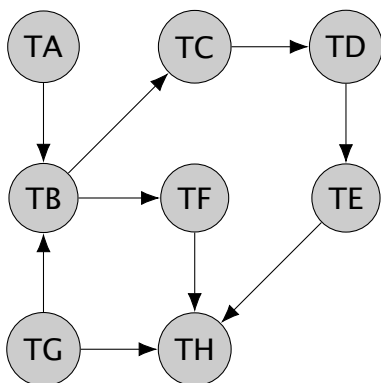
- Same as the list comprehension version (sort of)
- `map` takes a function and a list and applies the function to every element of the list
- Here the function is expressed as a `lambda` expression (an anonymous function)
- We need to convert the result to a list since `map` creates an iterable (explanation required?)

[ToC](#)

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

```

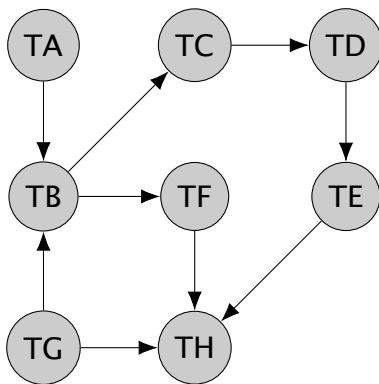
85 def topSorts(gr):
86     if isEmptyGraph(gr):
87         return [[]]
88     elif startVertices(gr) == []:
89         raise RuntimeError('Cycle_in_the_graph')
90     else:
91         return [[v] + ts
92                 for v in startVertices(gr)
93                 for ts in topSorts(removeVertex(v,gr))]
```

[ToC](#)

5.2 Topological Sort Example 01

Activity 3 Trace Exercise

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

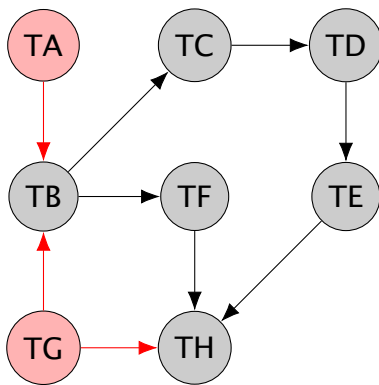

[Go to Answer](#)

Answer 3 Trace Exercise

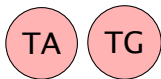
- Answer 3 Trace Exercise
- See the following slides

[Go to Activity](#)

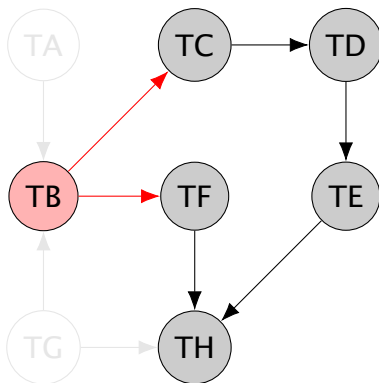
Step 1 Initial Graph



- Start vertices



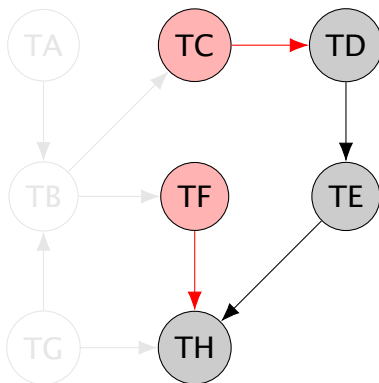
Step 2 Remove Vertices TA, TG



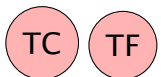
- Start vertices



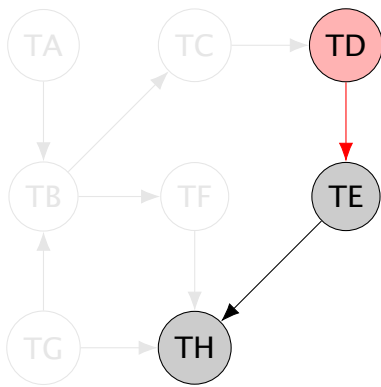
Step 3 Remove Vertex TB



- Start vertices



Step 4 Remove Vertices TC, TF

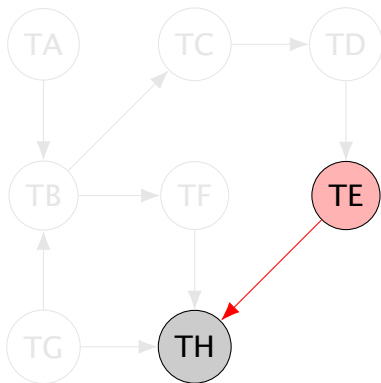


- Start vertices



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

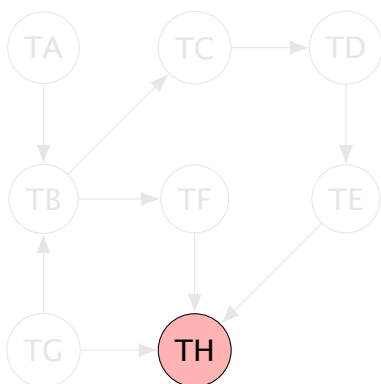
Step 5 Remove Vertex **TD**



- Start vertices



Step 6 Remove Vertex **TE**



- Start vertices



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

Topological Sort — Output

```

97 topSortsEG01GrTest \
98   = (topSorts(eg01Gr)
99     == [[ta,tg,tb,tc,td,te,tf,th]
100        , [ta,tg,tb,tc,td,tf,te,th]
101        , [ta,tg,tb,tc,tf,td,te,th]
102        , [ta,tg,tb,tf,tc,td,te,th]
103        , [tg,ta,tb,tc,td,te,tf,th]
104        , [tg,ta,tb,tc,td,tf,te,th]
105        , [tg,ta,tb,tc,tf,td,te,th]
106        , [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 (`\<n1>`) and [Python Implicit line joining](#) with (`(...)`)
- The [backslash](#) (`\`) must be followed by an [end of line character](#) (`<n1>`)

[ToC](#)

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

```

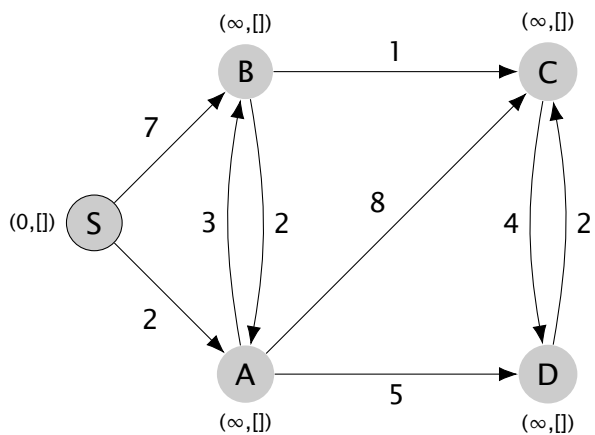
dijkstra(gr, weight, s)
  for u in vertices(gr)
    dist(u) = Infinity
    label(u) = Temp
  dist(s) = 0
  pred(s) = None
  q = makePriorityQ(vertices(gr))

  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

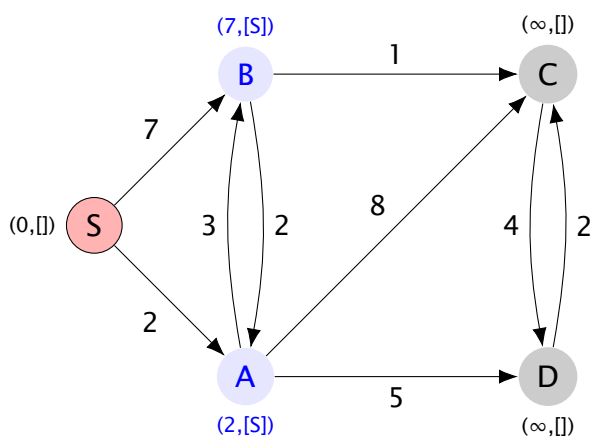
```

6.2 Dijkstra's Algorithm Example 01

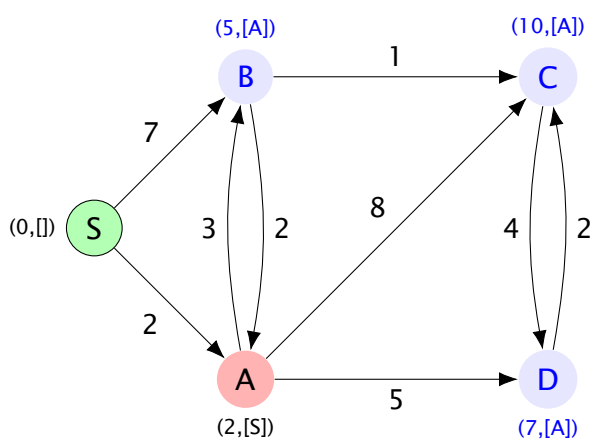
Step 0 Initialisation



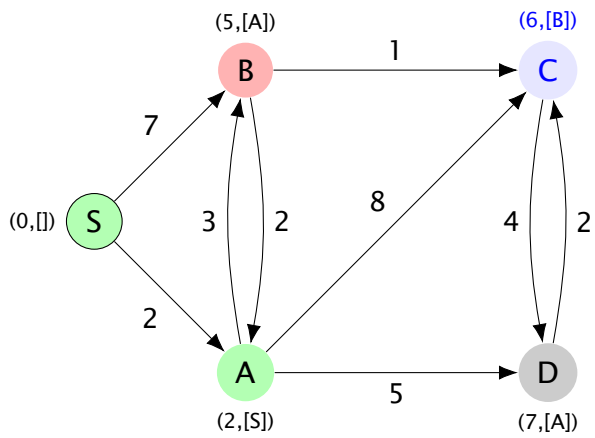
Step 1 Process S



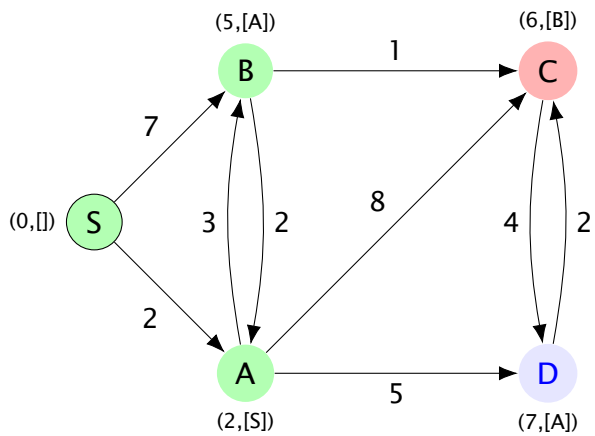
Step 2 Process A



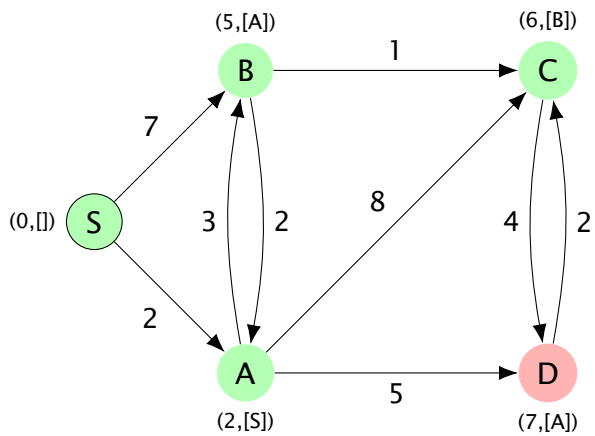
Step 3 Process B



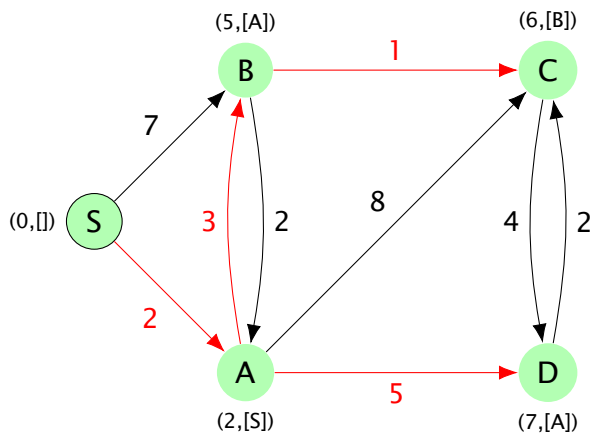
Step 4 Process C



Step 5 Process D



Shortest Path Tree Edges



ToC

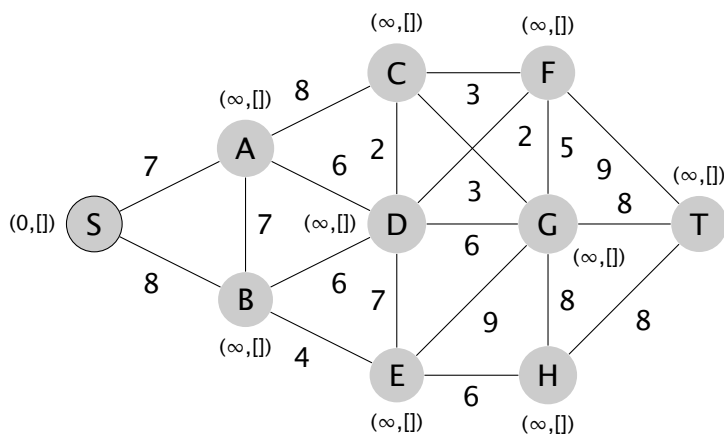
6.3 Dijkstra's Algorithm — Further points

- 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. \(2009, Chp 24\)](#)

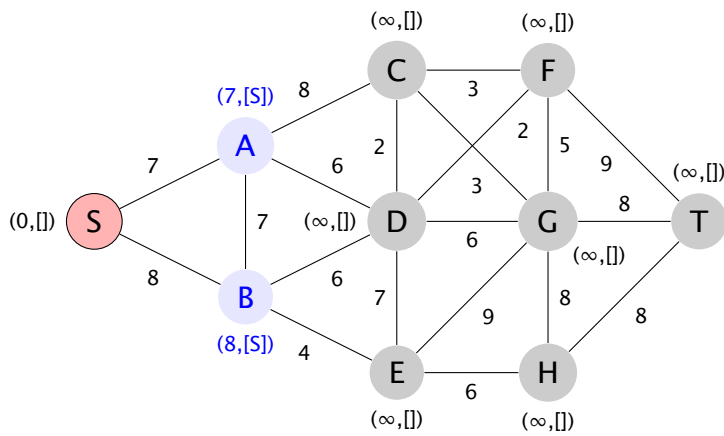
ToC

6.4 Dijkstra's Algorithm Example 02

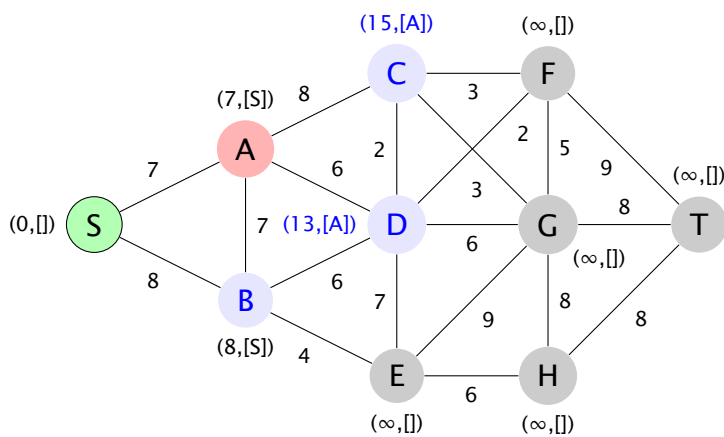
Step 0 Initialisation



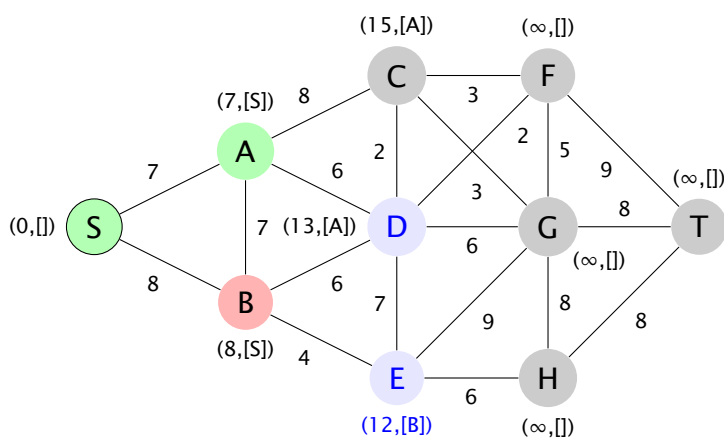
Step 1 Process S



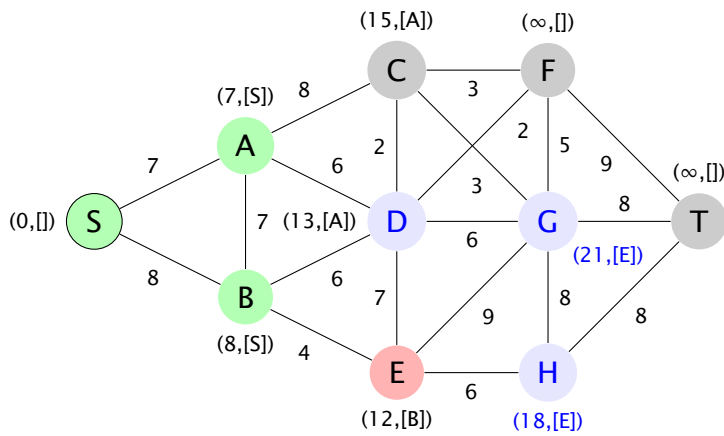
Step 2 Process A



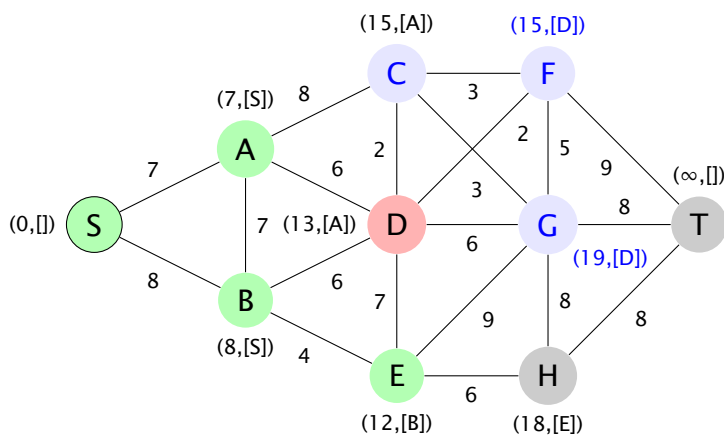
Step 3 Process B



Step 4 Process E

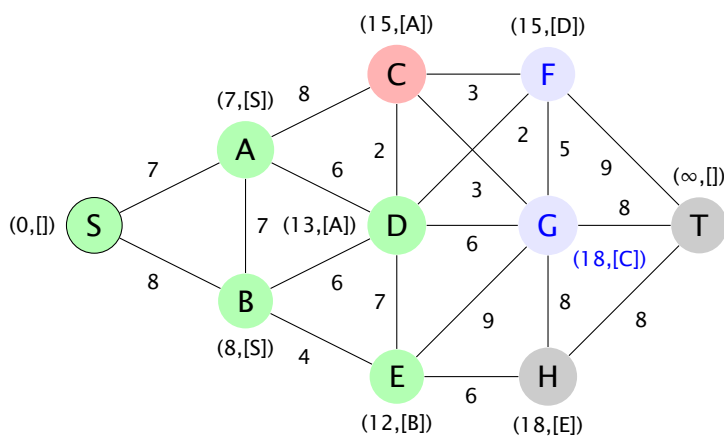


Step 5 Process D

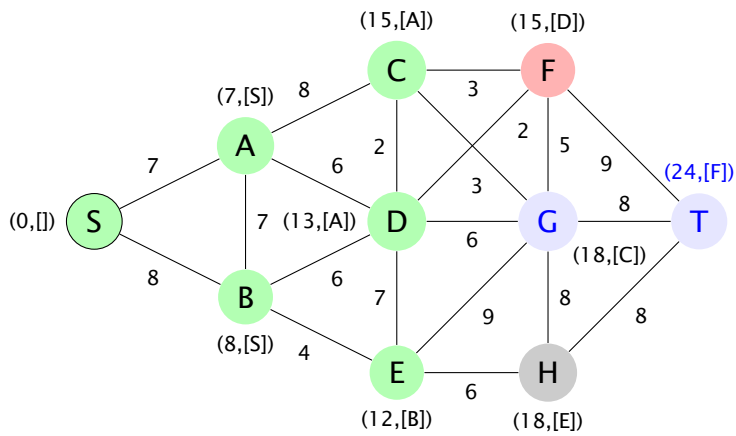


- Vertex **C** should have label **(15, [A, D])** if we record multiple shortest routes
- How do we change the algorithm ?

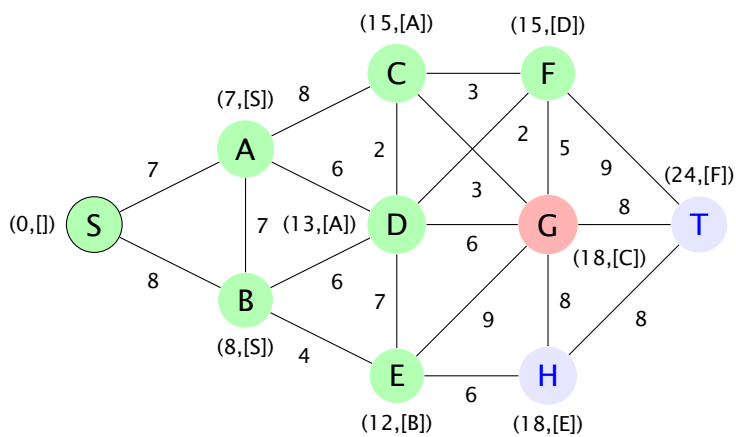
Step 6 Process C (or F)



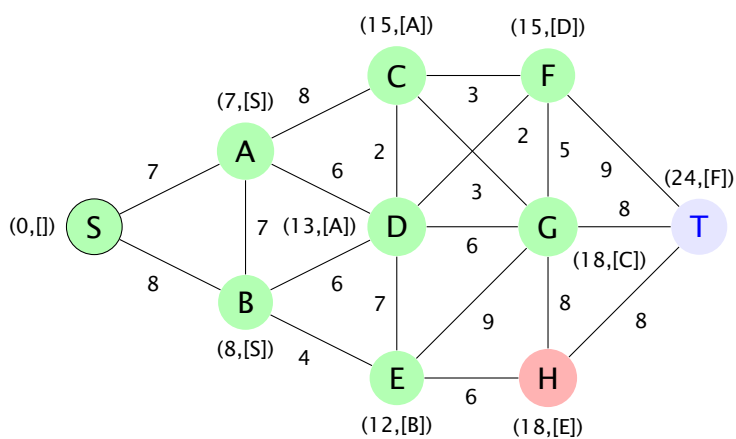
Step 7 Process F



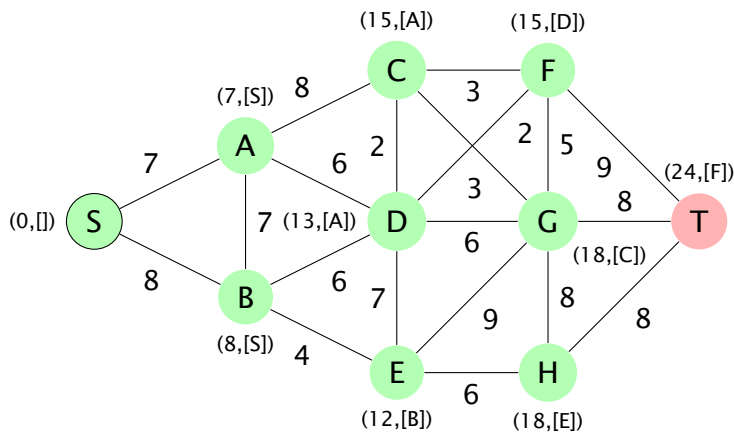
Step 8 Process G (or H)



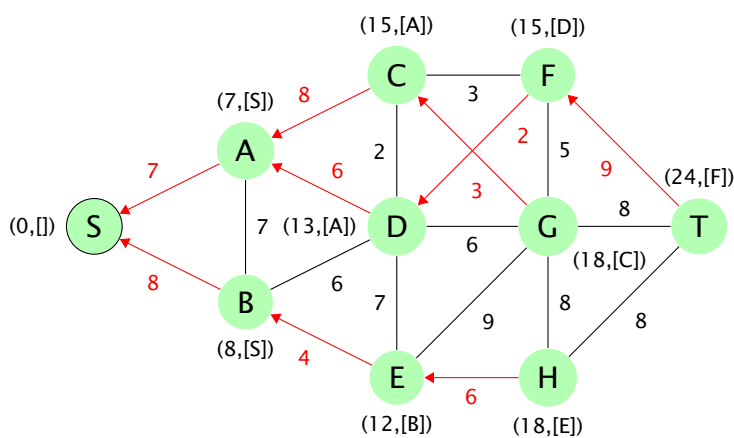
Step 9 Process H



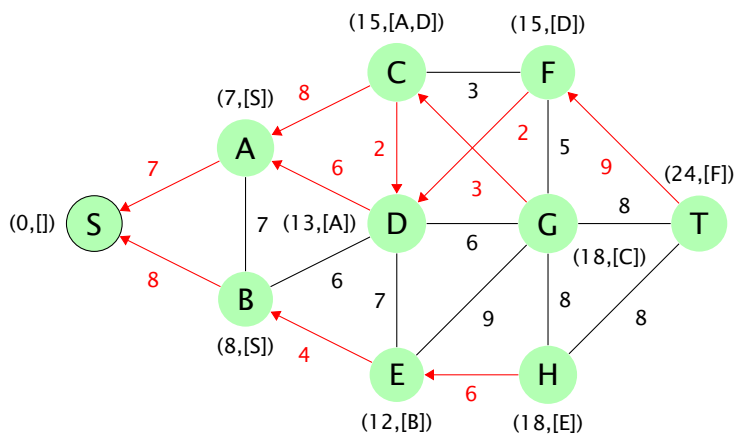
Step 10 Process T



Shortest Path Tree



Shortest Path Graph

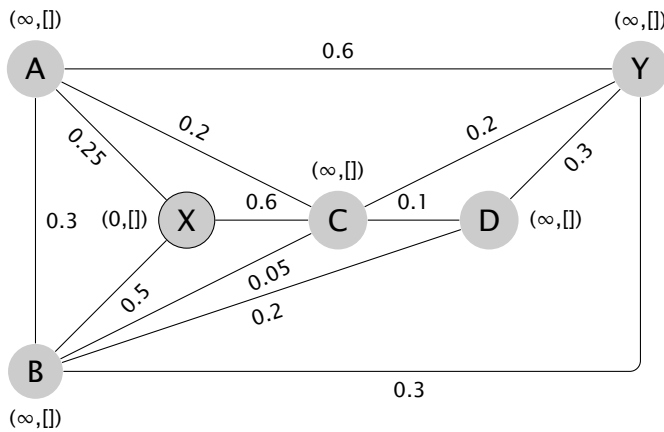

[ToC](#)

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

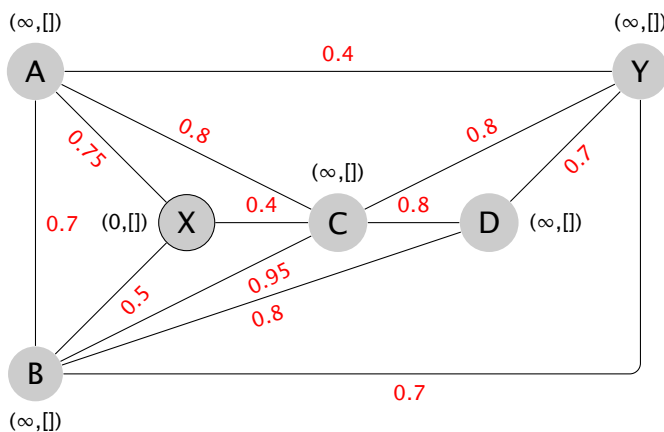
Step 0 Initialisation



Formulation as Shortest Path

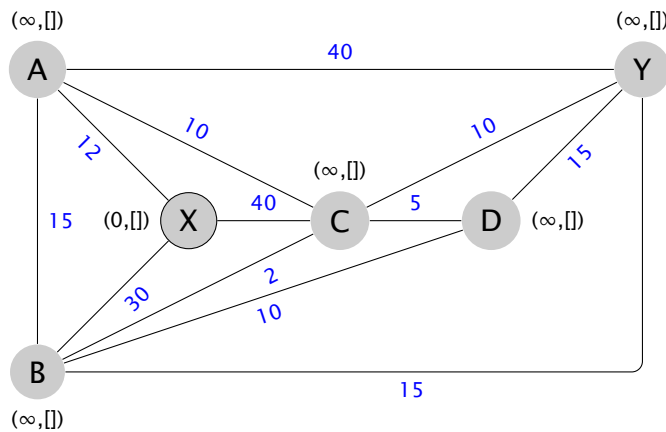
- Let $p_{(i,j)}$ be probability of failing on edge (i,j)
- The probability of not failing is $x_{(i,j)} = 1 - p_{(i,j)}$
- Over any path $x_{(i,j)}$ are independent so problem is to maximise probability of not failing $\prod_{(i,j) \in \text{path}} x_{(i,j)}$
- Equivalently, if $y_{(i,j)} = \log x_{(i,j)}$ then problem is to maximise $\sum_{(i,j) \in \text{path}} y_{(i,j)}$
- 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 $\sum_{(i,j) \in \text{path}} z_{(i,j)}$

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 $x_{(i,j)}$
- $z_{(i,j)} = -100 \log_{10} x_{(i,j)}$

[ToC](#)

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*

[ToC](#)

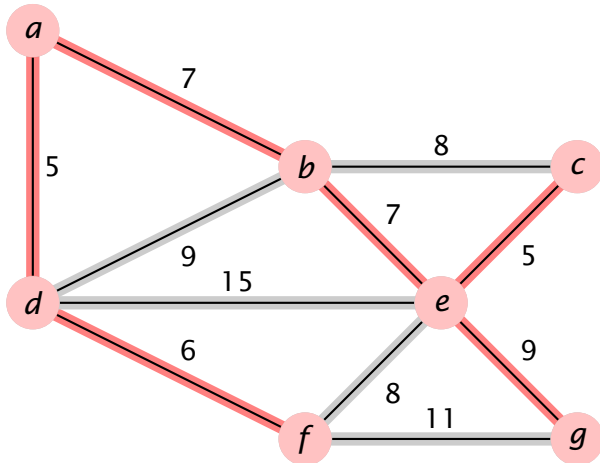
7.2 Prim's Algorithm — Example

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

Example Graph 01 *egPrimGraph00*



ToC

8 Greedy Algorithms

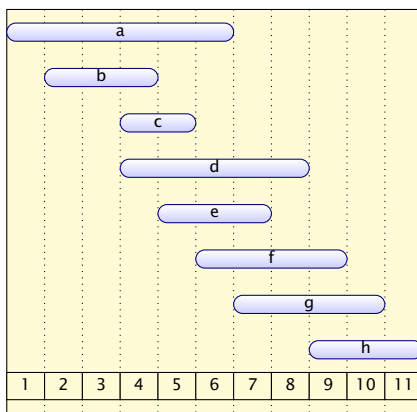
8.1 Interval Scheduling

- [Greedy algorithms](#) follow the problem solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum
- In general this rarely works — but it does in some cases including
 - [Dijkstra's algorithm](#) and [A* search algorithm](#) for graph search and shortest path finding
 - [Kruskal's algorithm](#) and [Prim's algorithm](#) for constructing minimum spanning trees of a given connected graph
 - [Interval scheduling](#) or [Activity selection problem](#) to find the maximum number of activities that do not clash with each other
- If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods such as [dynamic programming](#).
- [Interval scheduling](#)
- Job j starts at s_j and finishes at f_j
- Two jobs are compatible if they do not overlap

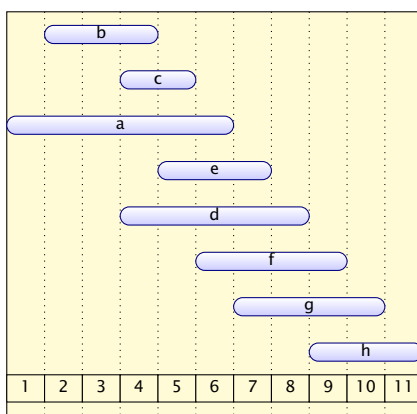
- **Q** What is the maximum subset of mutually compatible jobs?
- **Greedy template** Consider jobs in some order. Take each job provided it is compatible with the ones already taken.
- **Exercise** What orderings can we have ?
- Example from [Greedy algorithms: Interval scheduling](#)
- **Greedy template** Consider jobs in some order. Take each job provided it is compatible with the ones already taken.
- **Earliest start time** Consider jobs in ascending order of start time s_j
- **Earliest finish time** Consider jobs in ascending order of finish time f_j
- **Shortest interval** Consider jobs in ascending order of interval length $f_j + 1 - s_j$
- **Fewest conflicts** For each job, count the number of conflicting jobs c_j and schedule in ascending order of conflicts c_j
- For the jobs given below, produce an ordering by each of the greedy templates (above) and the schedule produced
- Each triple in the list below means (name, s_j, f_j) where the times are inclusive

jobs

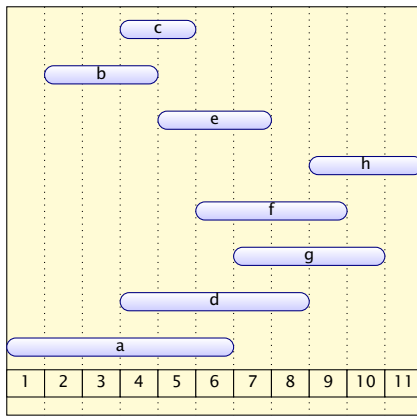
= [(a,1,6), (b,2,4), (c,4,5), (d,4,8),
(e,5,7), (f,6,9), (g,7,10), (h,9,11)]



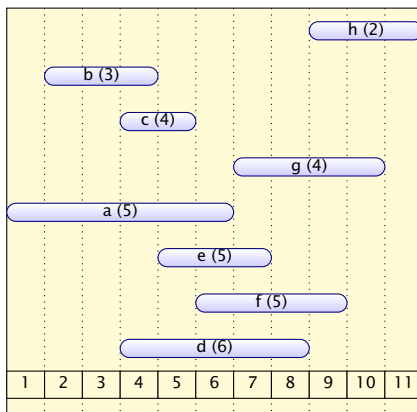
- Schedule jobs **a, g** (2 jobs)



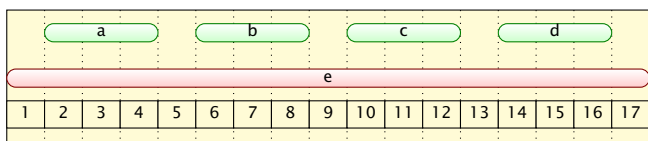
- Schedule jobs **b, e, h** (3 jobs)



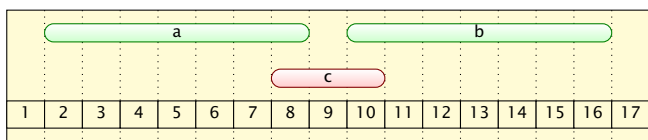
- Schedule jobs **c**, **h** (2 jobs)



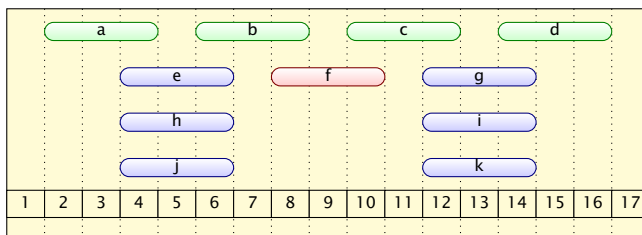
- Schedule jobs **h**, **b**, **e** (3 jobs)
- For each of the following *Greedy Templates* produce a counter example to show it may not produce the optimal schedule
- **Earliest start time** Consider jobs in ascending order of start time s_j
- **Shortest interval** Consider jobs in ascending order of interval length $f_j + 1 - s_j$
- **Fewest conflicts** For each job, count the number of conflicting jobs c_j and schedule in ascending order of conflicts c_j



- **e** dominates the optimal schedule by starting earlier and overlapping the others



- **c** dominates the optimal schedule y being shorter and overlapping the other two



- **f** dominates the optimal schedule by only having two conflicts and overlapping **b** and **c**

Order by Earliest Finish Time — Optimality Proof

- Basic structure of correctness proof:
- Assume that there is an optimal solution that is different from the greedy solution.
- Find the *first* difference between the two solutions.
- Argue that we can exchange the optimal choice for the greedy choice without making the solution worse (although the exchange might not make it better).
- This argument implies by induction that some optimal solution *contains* the entire greedy solution, and therefore *equals* the greedy solution.
- Sometimes, an additional step is required to show no optimal solution *strictly* improves the greedy solution.
- See [Jeff Erickson: Algorithms](#)
- Proof also in [Interval Scheduling](#) and [Greedy Algorithms](#)
- The slides at [Kevin Wayne: Greedy Algorithms](#) are from [Kleinberg and Éva Tardos \(2013\)](#)

ToC

9 Future Work

- Thursday, 13 March 2025 TMA02
- Sunday, 6 April 2025 Tutorial Online (Module wide) Dynamic Programming
- Sunday, 27 April 2025 Tutorial Online (Module wide) Computability and Complexity
- Sunday, 4 May 2025 Tutorial Online Review of course material for TMA03
- Thursday, 22 May 2025 TMA03

10 Web Sites & References

- Tree (Graph Theory) [http://en.wikipedia.org/wiki/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/>

10.1 Shortest Paths

- Dijkstra's Algorithm
 - [Dijkstra's shortest path algorithm](#)
- Bellman-Ford
 - [Bellman-Ford Algorithm](#)
 - [Bellman Ford Algorithm \(Simple Implementation\)](#)
 - [Haskell Hackage Data.BellmanFord](#)
 - [Data.IGraph](#) from [igraph](#) [igraph Reference Manual Chapter 13 Structural Properties of Graphs](#) [R igraph manual pages: Shortest \(directed or undirected\) paths between vertices](#) [IGraph/M: a Mathematica interface for igraph](#)

[ToC](#)

10.2 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-Wunsch_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>. 34, 37, 43, 44, 48
- 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/>. 11, 34
- 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.
- Kleinberg, Jon and Éva Tardos (2013). *Algorithm Design*. Pearson. ISBN 1292023945. URL <https://www.cs.princeton.edu/~wayne/kleinberg-tardos/>. 47
- 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-1p5e.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>. 34, 44
- 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>. 34, 44
- Sedgewick, Robert and Kevin Wayne (2011). *Algorithms*. Addison Wesley, fourth edition. ISBN 032157351X. URL <http://algs4.cs.princeton.edu/home/>. 34, 44
- 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.

Python Code Index

Index for some (but not all) of the python code. Note that the index commands are placed after any *listings* environment containing the code to be indexed.

combsM, [30](#)
combsM01, [29](#)

Edge, [18](#)
edges, [19](#)
endVertices, [19](#)
esEndV, [19](#)
esRemoveV, [19](#)
esStartV, [19](#)

fairListing, [17](#)
fairListingA, [17](#)
filterStopWords, [15](#)

isEmptyGraph, [19](#)

predLists, [19](#)

removeVertex, [19](#)

startVertices, [19](#)
subSeqsM, [29](#)
succLists, [19](#)

topSorts, [31](#)
transMat, [15](#)

Vertex, [18](#)
vertices, [19](#)

yBiasListing, [16](#)

[ToC](#)

Pseudocode Index

Index for some of the pseudo code. Note that the index commands are placed after any *listings* environment containing the code to be indexed.

dijkstra, [35](#)

prim, [43](#)

[ToC](#)

M269 DiGraph Code Index

Index for some of the M269 DiGraph code. Note that the index commands are placed after any *listings* environment containing the code to be indexed.

add_edge, [20](#)

add_node, [20](#)

bfs, [22](#)

degree, [22](#)

dfs, [22](#)

DiGraph, [20](#)

draw, [22](#)

edges, [21](#)

has_edge, [20](#)

has_node, [20](#)

in_degree, [21](#)

in_neighbours, [21](#)

neighbours, [22](#)

nodes, [21](#)

out, [20](#)

out_degree, [21](#)

out_neighbours, [21](#)

remove_edge, [21](#)

remove_node, [21](#)

[ToC](#)

M269 Weighted DiGraph Code Index

Index for some of the M269 Weighted DiGraph code. Note that the index commands are placed after any *listings* environment containing the code to be indexed.

`add_edge`, [23](#)

`add_node`, [23](#)

`dijkstra`, [24](#)

`draw`, [23](#)

`edges`, [23](#)

`out_neighbours`, [23](#)

`remove_edge`, [23](#)

`weight`, [23](#)

`WeightedDiGraph`, [23](#)

[ToC](#)

M269 Undirected Graph Code Index

Index for some of the M269 Undirected Graph code. Note that the index commands are placed after any *listings* environment containing the code to be indexed.

`add_edge`, [24](#)

`degree`, [25](#)

`edges`, [25](#)

`in_degree`, [25](#)

`in_neighbours`, [25](#)

`neighbours`, [25](#)

`remove_edge`, [24](#)

`UndirectedGraph`, [24](#)

[ToC](#)

M269 Weighted Undirected Graph Code Index

Index for some of the M269 Weighted Undirected Graph code. Note that the index commands are placed after any *listings* environment containing the code to be indexed.

`add_edge`, [25](#)

`degree`, [26](#)

`edges`, [26](#)

`in_degree`, [26](#)

`in_neighbours`, [26](#)

`neighbours`, [26](#)

`prim`, [26](#)

`remove_edge`, [25](#)

`WeightedUndirectedGraph`, [25](#)

[ToC](#)

Diagrams Index

Index for some of the PGF/TikZ diagrams. Note that the indexing commands are placed after the diagram code

egDigraph, [9](#)
egDijkstraGraph0100, [35](#)
egDijkstraGraph0101, [35](#)
egDijkstraGraph0102, [35](#)
egDijkstraGraph0103, [36](#)
egDijkstraGraph0104, [36](#)
egDijkstraGraph0105, [36](#)
egDijkstraGraph0106, [37](#)
egDijkstraGraph0200, [37](#)
egDijkstraGraph0201, [38](#)
egDijkstraGraph0202, [38](#)
egDijkstraGraph0203, [38](#)
egDijkstraGraph0204, [39](#)
egDijkstraGraph0205, [39](#)
egDijkstraGraph0206, [39](#)
egDijkstraGraph0207, [40](#)
egDijkstraGraph0208, [40](#)
egDijkstraGraph0209, [40](#)
egDijkstraGraph0210, [41](#)
egDijkstraGraph02SPG, [41](#)
egDijkstraGraph02SPT, [41](#)
egDijkstraGraph0300, [42](#)
egDijkstraGraph0300a, [42](#)
egDijkstraGraph0300b, [43](#)
egGantt01Conflicts, [46](#)
egGantt01ConflictsCntr, [47](#)
egGantt01EFT, [45](#)
egGantt01EST, [45](#)
egGantt01ESTcntr, [46](#)
egGantt01ShortInt, [46](#)
egGantt01ShortIntCntr, [46](#)
egPrimGraph00, [44](#)
egTopSortGraph, [30](#)
egTopSortGraph00, [31](#)
egTopSortGraph01, [32](#)
egTopSortGraph02, [32](#)
egTopSortGraph03, [32](#)
egTopSortGraph04, [33](#)
egTopSortGraph05, [33](#)
egTopSortGraph06, [33](#)

[ToC](#)