

Graphs and Greedy Algorithms



Topic

Phil Molyneux

8 March 2026

M269 Graph Algorithms

Agenda & Aims

- ▶ 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 [/M269TutorialNotes/M269Tutorial20260308GraphsGreedPrsntn2025J/](https://www.philmolyneux.com/M269TutorialNotes/M269Tutorial20260308GraphsGreedPrsntn2025J/)
- ▶ **Recording**   ✓

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

M269 Tutorial

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

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

M269 Tutorial

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)*

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

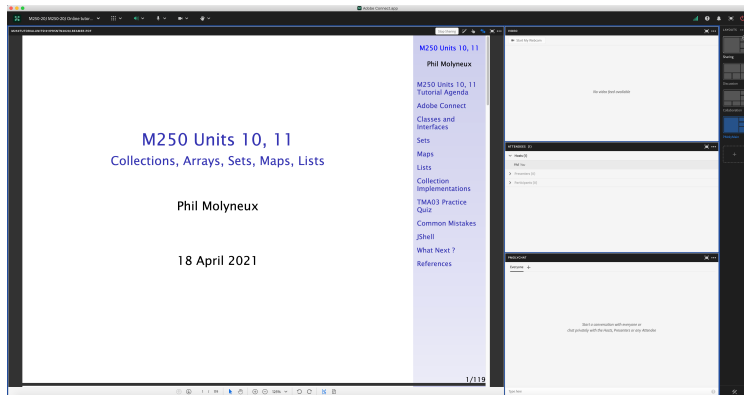
Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect

Interface — Host View



Graphs and Greedy Algorithms

Phil Molyneux

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect

Interface — Participant View

M250 Units 10, 11 Tutorial
Introductions

- ▶ Introductions
 - ▶ Name *Phil Molyneux*
 - ▶ Learning Style: *Reads the manual*
 - ▶ Learnt last month *Framework for Teaching Recursion* and wrote notes on *Recursion Teaching*
 - ▶ You ?

M250 Units 10, 11 Tutorial Agenda

Adobe Connect Classes and Interfaces

Sets

Maps

Lists

Collection Implementations

TMA03 Practice Quiz

Common Mistakes

JShell

What Next ?

References

3/119

Index

Members Not Available

Participants

Start a conversation with any participant privately with the tools, Presence or any toolbar

Graphs and Greedy Algorithms

Phil Molyneux

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect

Settings

- ▶ **Everybody** *Menu bar* *Meeting* *Speaker & Microphone Setup*
- ▶ *Menu bar* *Microphone* *Allow Participants to Use Microphone* ✓
- ▶ Check Participants see the entire slide **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*

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithms

Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

▶ 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

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm





Greedy Algorithms

Future Work

Web Sites & References

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**  + ****

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect Interface

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**

Agenda

Adobe Connect
Interface
Settings

Sharing Screen & Applications

Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

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

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

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](#)

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms


Future Work

Web Sites & References

Adobe Connect

Entering a Room as a Guest (1)

- ▶ 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

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect

Entering a Room as a Guest (2)

- ▶ 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.

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

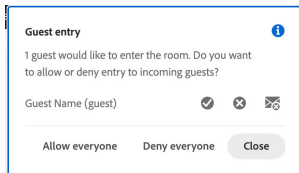
Future Work

Web Sites & References

Adobe Connect

Entering a Room as a Guest (3)

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



Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect

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

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect

Layouts

- ▶ 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

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm

Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms



Future Work

Web Sites & References

Adobe Connect

Chat Pods

- ▶ **Format Chat text**

- ▶  menu icon 

- ▶ Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black

- ▶ Note: Color reverts to Black if you switch layouts

- ▶  menu icon 

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm




Greedy Algorithms

Future Work

Web Sites & References

Graphics Conversion

PDF to PNG/JPG

- ▶ Conversion of the screen snaps for the installation of Anaconda on 1 May 2020
- ▶ Using GraphicConverter 1.1
- ▶ 
- ▶ Select files to convert and destination folder
- ▶ Click on  or 

Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Adobe Connect Recordings

Exporting 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*

Agenda

Adobe Connect

- Interface
- Settings
- Sharing Screen & Applications
- Ending a Meeting
- Invite Attendees
- Layouts
- Chat Pods
- Web Graphics

Recordings

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

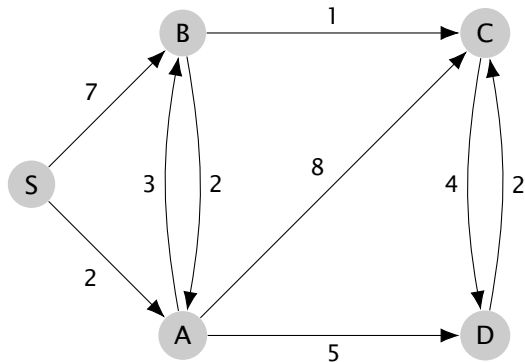
Graph Algorithms

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 $w(e)$

Graph Algorithms

Example Digraph egDigraph



Agenda

Adobe Connect

M269 Graph Algorithms

Graph Definitions

Graph Representation

Algorithm

Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Graph Algorithms

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 Algorithms

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](#)

Graph Algorithms

Answer 1 Graph Operations

- ▶ Answer 1 Graph Operations — see next slide

▶ Go to Activity

Graph Algorithms

Graph Operations 01

- ▶ *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

Graph Algorithms

Graph Operations 02

- ▶ *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 Algorithms

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 Algorithms

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.

Algorithm Descriptions & Implementations

Overview

- ▶ 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

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

List Comprehension Exercises

Activity 2 (a) Stop Words Filter

- ▶ **Stop words** are the most common words that most search engines avoid: 'a', 'an', 'the', 'that', ...
- ▶ 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](#)[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[List Comprehensions](#)[Python Graph Representation](#)
[Python Graph Representation from 21J](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Future Work](#)[Web Sites & References](#)

List Comprehension Exercises

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](#)[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[List Comprehensions](#)[Python Graph Representation](#)[Python Graph Representation from 21J](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Future Work](#)[Web Sites & References](#)

List Comprehension Exercises

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](#)[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[List Comprehensions](#)[Python Graph Representation](#)[Python Graph Representation from 21J](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Future Work](#)[Web Sites & References](#)

List Comprehension Exercises

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](#)[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[List Comprehensions](#)[Python Graph Representation](#)[Python Graph Representation from 21J](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Future Work](#)[Web Sites & References](#)

List Comprehension Exercises

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](#)[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[List Comprehensions](#)[Python Graph Representation](#)[Python Graph Representation from 21J](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Future Work](#)[Web Sites & References](#)

List Comprehension Exercises

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](#)[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[List Comprehensions](#)[Python Graph Representation](#)[Python Graph Representation from 21J](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Future Work](#)[Web Sites & References](#)

List Comprehension Exercises

Answer 2 (a) Stop Words Filter

- ▶ Answer 2 (a) Stop Words Filter
- ▶ *Write here:*
- ▶ Answer 2 continued on next slide

▶ Go to Activity

List Comprehension Exercises

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

List Comprehension Exercises

Answer 2 (b) Transpose Matrix

- ▶ Answer 2 (b) Transpose Matrix
- ▶ *Write here:*
- ▶ Answer 2 continued on next slide

▶ Go to Activity

List Comprehension Exercises

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
- Answer 2 continued on next slide

► Go to Activity

List Comprehension Exercises

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](#)

List Comprehension Exercises

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)
```

List Comprehension Exercises

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

List Comprehension Exercises

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](#)

List Comprehension Exercises

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

List Comprehension Exercises

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](#)

List Comprehension Exercises

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

List Comprehension Exercises

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](#)[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[List Comprehensions](#)[Python Graph Representation](#)[Python Graph Representation from 21J](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Future Work](#)[Web Sites & References](#)

Algorithm Descriptions & Implementations

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](#)

Graph Representation

Python

```
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](#)
- ▶ **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

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Graph Service Functions

Python (1)

```
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)

Graph Service Functions

Python (2)

```
55 def sucLists(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 sucLists(gr)
72             if sLst[1] == []]
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Graph Service Functions

Python (3)

```
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

Python Graph Representation from 21J

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

Python 21J Adjacency List Representation

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.
15
16     Edges are between different nodes.
17     There's at most one edge from one node to another.
18     """
```

DiGraph Class

Constructor, Inspectors

```
20 def __init__(self):
21     self.out = dict() # a map of nodes to their out-neighbours
22
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
26
27 def has_edge(self, start: Hashable, end: Hashable) -> bool:
28     """Return True if and only if edge start -> end exists.
29
30     Preconditions: self.has_node(start) and self.has_node(end)
31     """
32     return end in self.out[start]
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

DiGraph Class

Add Node,Edge

```
34 def add_node(self, node: Hashable) -> None:
35     """Add the node to the graph.
36
37     Preconditions: not self.has_node(node)
38     """
39     self.out[node] = set()
40
41 def add_edge(self, start: Hashable, end: Hashable) -> None:
42     """Add edge start -> end to the graph.
43
44     If the edge already exists, do nothing.
45
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

DiGraph Class

Remove Node,Edge

```
51 def remove_node(self, node: Hashable) -> None:
52     """Remove the node and all its attached edges.
53
54     Preconditions: self.has_node(node)
55     """
56     self.out.pop(node)
57     for start in self.out:
58         self.remove_edge(start, node)
59
60 def remove_edge(self, start: Hashable, end: Hashable) -> None:
61     """Remove edge start -> end from the graph.
62
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

DiGraph Class

Get Nodes,Edges

```
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

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
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

DiGraph Class

Out Neighbours, Degrees

```
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])
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

DiGraph Class

In Neighbours, Degrees

```
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))
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class
Weighted Undirected Graph Class

Drawing Graphs
Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

DiGraph Class

Neighbours, Degree

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

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

DiGraph Class

Draw DiGraph

```
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')
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

DiGraph Class

Breadth First Search

```
142 from collections import deque
144 def bfs(graph: DiGraph, start: Hashable) -> DiGraph:
145     """Return the subgraph traversed by a breadth-first search.
146
147     Preconditions: graph.has_node(start)
148     """
149     # changes from traversed function noted in comments
150     visited = DiGraph()
151     visited.add_node(start)
152     unprocessed = deque() # set -> deque
153     for neighbour in graph.out_neighbours(start):
154         unprocessed.append( (start, neighbour) ) # add -> append
155     while len(unprocessed) > 0:
156         edge = unprocessed.popleft() # pop -> popleft
157         previous = edge[0]
158         current = edge[1]
159         if not visited.has_node(current):
160             visited.add_node(current)
161             visited.add_edge(previous, current)
162         for neighbour in graph.out_neighbours(current):
163             unprocessed.append( (current, neighbour) ) # add -> append
164     return visited
```

DiGraph Class

Depth First Search

```
166 def dfs(graph: DiGraph, start: Hashable) -> DiGraph:
167     """Return the subgraph traversed by a depth-first search.
169     Preconditions: graph.has_node(start)
170     """
171     visited = DiGraph()
172     visited.add_node(start)
173     unprocessed = [] # deque -> list
174     for neighbour in graph.out_neighbours(start):
175         unprocessed.append( (start, neighbour) )
176     while len(unprocessed) > 0:
177         edge = unprocessed.pop() # popleft -> pop
178         previous = edge[0]
179         current = edge[1]
180         if not visited.has_node(current):
181             visited.add_node(current)
182             visited.add_edge(previous, current)
183             for neighbour in graph.out_neighbours(current):
184                 unprocessed.append( (current, neighbour) )
185     return visited
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class
Weighted Undirected Graph Class

Drawing Graphs
Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted DiGraph Class

Initial Code, Add Node, Edge

```
187 import math
189 class WeightedDiGraph(DiGraph):
190     """A weighted directed graph with hashable node objects.
191
192     Edges are between different nodes.
193     There's at most one edge from one node to another.
194     Edges have weights, which can be floats or integers.
195     """
196
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
```

Weighted DiGraph Class

Weight, Remove Edge

```
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     Preconditions: self.has_node(start) and self.has_node(end)
217     """
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     If the edge doesn't exist, do nothing.
227     Preconditions: self.has_node(start) and self.has_node(end)
228     """
229
230     if self.has_edge(start, end):
231         self.out[start].pop(end)
```

Agenda

Adobe Connect
M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class
Weighted Undirected Graph Class

Drawing Graphs

Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted DiGraph Class

Weight, Remove Edge

```
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

242 def out_neighbours(self, node: Hashable) -> set:
243     """Return the out-neighbours of the node.

245     Preconditions: self.has_node(node)
246     """
247     return set(self.out[node].keys())
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted DiGraph Class

Draw

```
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'))
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class
Weighted Undirected Graph Class

Drawing Graphs
Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted DiGraph Class

Shortest Path: Dijkstra (1)

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

▶ Shortest Path Dijkstra continued on next slide

Weighted DiGraph Class

Shortest Path: Dijkstra (2)

```
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
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Python 21J Adjacency List Representation

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.
15     There's at most one edge between two different nodes.
16     There are no edges between a node and itself.
17     """
```

Undirected Graph Class

Add and Remove Edge

```
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)
28
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)
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Undirected Graph Class

Edges, Neighbours

```
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
51
52 def in_neighbours(self, node: Hashable) -> set:
53     """Return all nodes that are adjacent to the node.
54
55     Preconditions: self.has_node(node)
56     """
57     return self.out_neighbours(node)
58
59 def neighbours(self, node: Hashable) -> set:
60     """Return all nodes that are adjacent to the node.
61
62     Preconditions: self.has_node(node)
63     """
64     return self.out_neighbours(node)
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Undirected Graph Class

In Degree, Degree

```
66 def in_degree(self, node: Hashable) -> int:
67     """Return the number of edges attached to the node.
69     Preconditions: self.has_node(node)
70     """
71     return self.out_degree(node)
73 def degree(self, node: Hashable) -> int:
74     """Return the number of edges attached to the node.
76     Preconditions: self.has_node(node)
77     """
78     return self.out_degree(node)
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted Undirected Graph Class

Initial Code

```
80 class WeightedUndirectedGraph(WeightedDiGraph):
81     """A weighted undirected graph with hashable node objects.
83     There's at most one edge between two different nodes.
84     There are no edges between a node and itself.
85     Edges have weights, which may be integers or floats.
86     """
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs
Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted Undirected Graph Class

Add and Remove Edge

```
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
91     If the edge already exists, do nothing.
92
93     Preconditions: self.has_node(node1) and self.has_node(node2)
94     """
95     super().add_edge(node1, node2, weight)
96     super().add_edge(node2, node1, weight)
97
98 def remove_edge(self, node1: Hashable, node2: Hashable) -> None:
99     """Remove edge node1-node2 from the graph.
100
101     If the edge doesn't exist, do nothing.
102
103     Preconditions: self.has_node(node1) and self.has_node(node2)
104     """
105     super().remove_edge(node1, node2)
106     super().remove_edge(node2, node1)
```

Weighted Undirected Graph Class

Edges

```
108 def edges(self) -> set:
109     """Return the graph's edges as a set of triples (node1, node2, weight).
110
111     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
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs
Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted Undirected Graph Class

In Neighbours, Neighbours, In Degree, Degree

```
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)
```

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations: Subsequences, Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Weighted Undirected Graph Class

Minimum Spanning Tree: Prim (1)

```
149 from heapq import heappush, heappop
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) )
```

- ▶ Minimum Spanning Tree Prim continued on next slide

Weighted Undirected Graph Class

Minimum Spanning Tree: Prim (2)

```
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`

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs
Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Drawing Graphs

Weighted DiGraph

- ▶ The provided graph code gives two `draw` methods:
- ▶ For Weighted DiGraph or Undirected Graph see line 249, slide 72,
- ▶ For Unweighted see line 130, slide 66,
- ▶ `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`

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Drawing Graphs

NetworkX, Matplotlib

- ▶ 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](#), slide [87](#)

Drawing Weighted DiGraph

draw method

```
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

Draw Method

Spring Layout

```
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, slide 87,

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

List Comprehensions

Python Graph Representation

Python Graph Representation from 21J

Graph Representation Choices

DiGraph Class

Weighted DiGraph Class

Undirected Graph Class

Weighted Undirected Graph Class

Drawing Graphs

Enumerations:
Subsequences,
Combinations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Draw Method

NetworkX draw function

```
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](#), slide [87](#)

Draw Method

NetworkX Draw Edge Labels function

```
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](#), slide [87](#),
- ▶ See also [draw_networkx_nodes](#), can take a [nodelist](#)
- ▶ See also [draw_networkx_edges](#), can take an [edgelist](#)

Agenda

[Adobe Connect](#)

[M269 Graph Algorithms](#)

[Algorithm Descriptions & Implementations](#)

[List Comprehensions](#)

[Python Graph Representation](#)

[Python Graph Representation from 21J](#)

[Graph Representation Choices](#)

[DiGraph Class](#)

[Weighted DiGraph Class](#)

[Undirected Graph Class](#)

[Weighted Undirected Graph Class](#)

[Drawing Graphs](#)

[Enumerations, Subsequences, Combinations](#)

[Topological Sort](#)

[Dijkstra's Algorithm](#)

[Prim's Algorithm](#)

[Greedy Algorithms](#)

[Future Work](#)

[Web Sites & References](#)

Draw Method

Inline and Externalizing Graphics

- ▶ 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`

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`)

Enumerations

Subsequences

- ▶ 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)

Enumerations

Combinations (1)

- ▶ 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

Enumerations

Combinations (2)

- ▶ 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?)

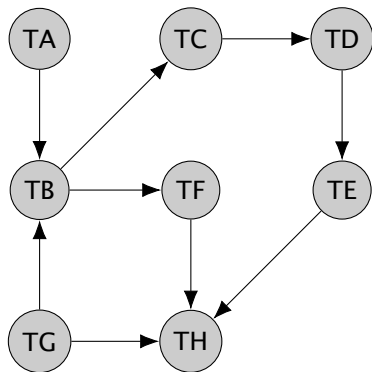
Topological Sort

Definition

- ▶ 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 egTopSortGraph



- Find all the topological orderings on this digraph

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))]
```

Graphs and Greedy Algorithms

Phil Molyneux

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Topological Sort — Algorithm

Topological Sort Example 01

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

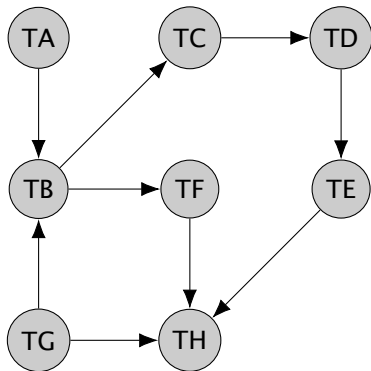
Future Work

Web Sites & References

Topological Sort Example 01

Activity 3 Trace Exercise egTopSortGraph00

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



▶ Go to Answer

Topological Sort Example 01

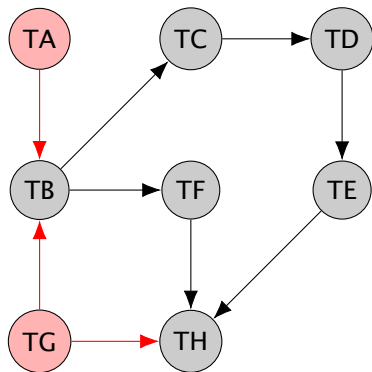
Answer 3 Trace Exercise

- ▶ Answer 3 Trace Exercise
- ▶ See the following slides

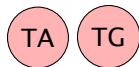
▶ Go to Activity

Topological Sort Example 01

Step 1 Initial Graph egTopSortGraph01



► Start vertices



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort
Topological Sort – Algorithm

Topological Sort Example 01

Dijkstra's Algorithm

Prim's Algorithm

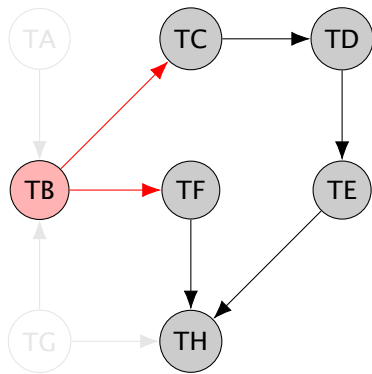
Greedy Algorithms

Future Work

Web Sites & References

Topological Sort Example 01

Step 2 Remove Vertices TA, TG egTopSortGraph02

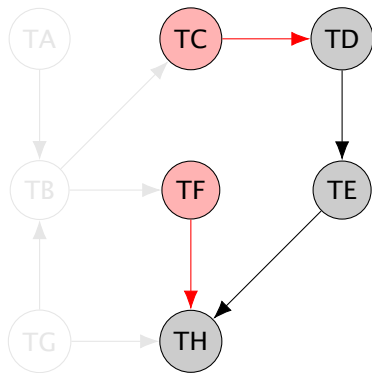


► Start vertices

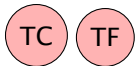


Topological Sort Example 01

Step 3 Remove Vertex TB egTopSortGraph03

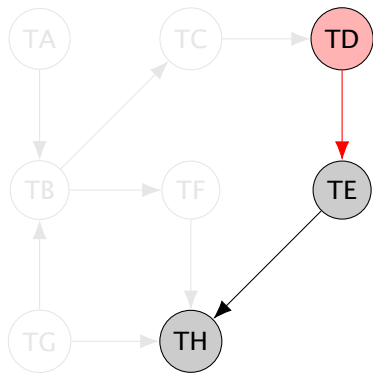


► Start vertices



Topological Sort Example 01

Step 4 Remove Vertices TC, TF egTopSortGraph04



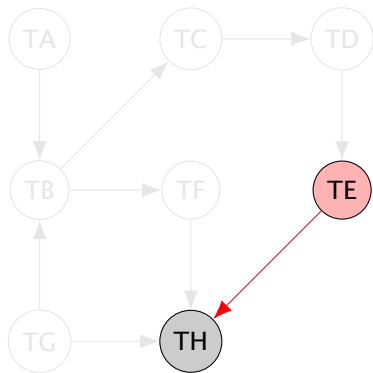
- ▶ Start vertices



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

Topological Sort Example 01

Step 5 Remove Vertex TD egTopSortGraph05

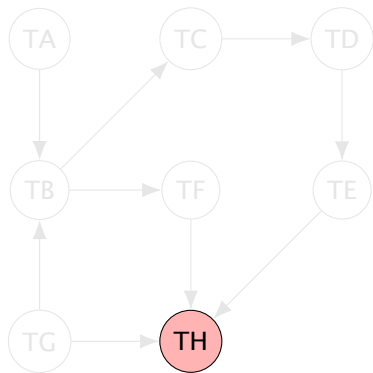


► Start vertices



Topological Sort Example 01

Step 6 Remove Vertex TE egTopSortGraph06



- ▶ Start vertices



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

Topological Sort — Example

Output — Python

```

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 (`\<n>`) and **Python Implicit line joining** with (`(...)`)
- ▶ The **backslash** (`\`) must be followed by an **end of line character** (`<n>`)

Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort
Topological Sort — Algorithm

Topological Sort Example 01

Dijkstra's Algorithm

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm

Sources

- ▶ From <https://www.cse.ust.hk/~dekai/271/> (Lecture 10)
- ▶ Cormen (2009, chapter 24) — Cormen (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/>
- ▶ Rabhi and Lapalme (1999, chapter 7)

Dijkstra's Algorithm

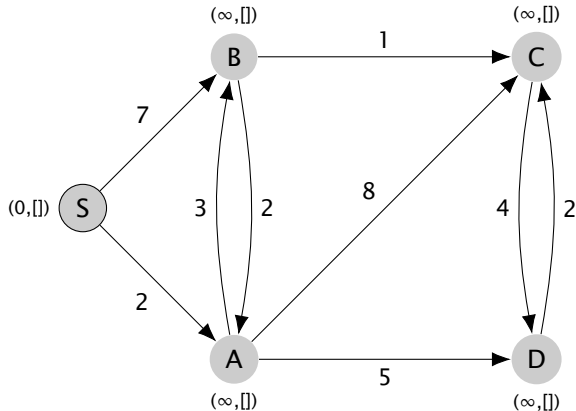
Structured English

```
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
```

Dijkstra's Algorithm Example 01

Step 0 Initialisation egDijkstraGraph0100



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

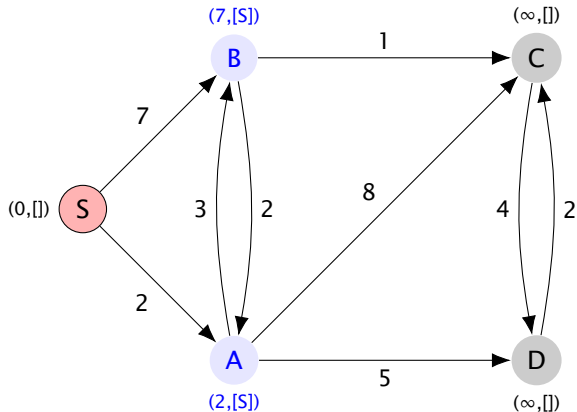
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example

Step 1 Process S egDijkstraGraph0101



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm — Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm — Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

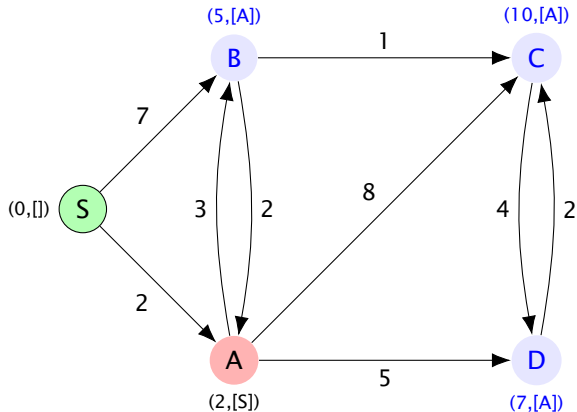
Greedy Algorithms

Future Work

Web Sites & References

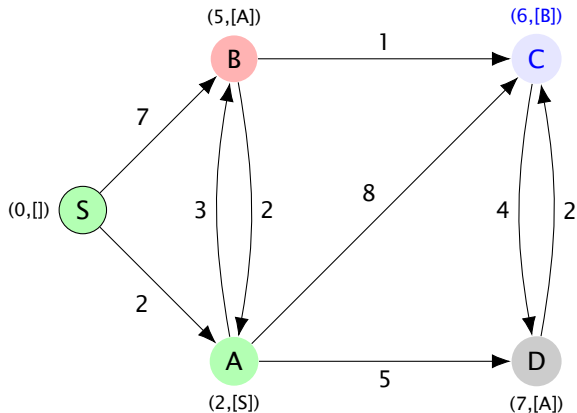
Dijkstra's Algorithm Example

Step 2 Process A egDijkstraGraph0102



Dijkstra's Algorithm Example

Step 3 Process B egDijkstraGraph0103



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

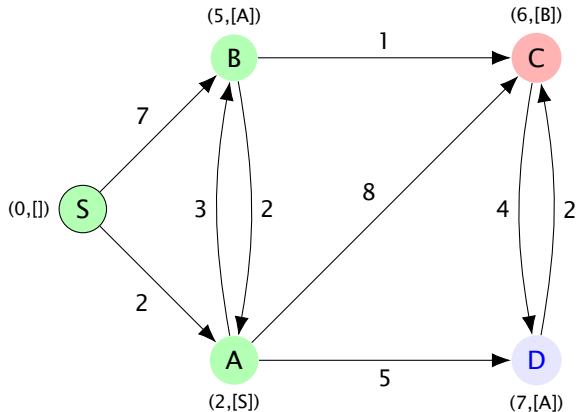
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example

Step 4 Process C egDijkstraGraph0104



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm — Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm — Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

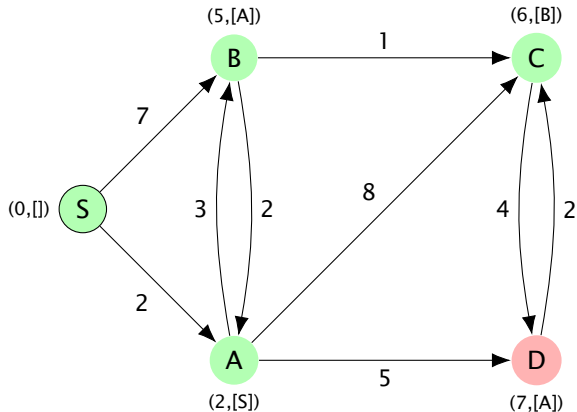
Greedy Algorithms

Future Work

Web Sites & References

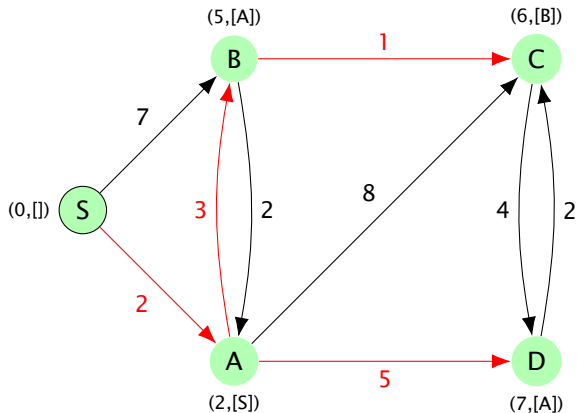
Dijkstra's Algorithm Example

Step 5 Process D egDijkstraGraph0105



Dijkstra's Algorithm Example

Shortest Path Tree Edges egDijkstraGraph0106



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm — Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm — Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

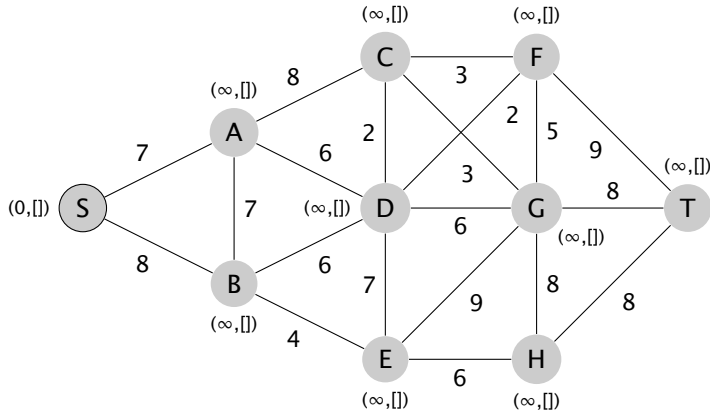
Dijkstra's Algorithm

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, chp 24

Dijkstra's Algorithm Example 02

Step 0 Initialisation egDijkstraGraph0200



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

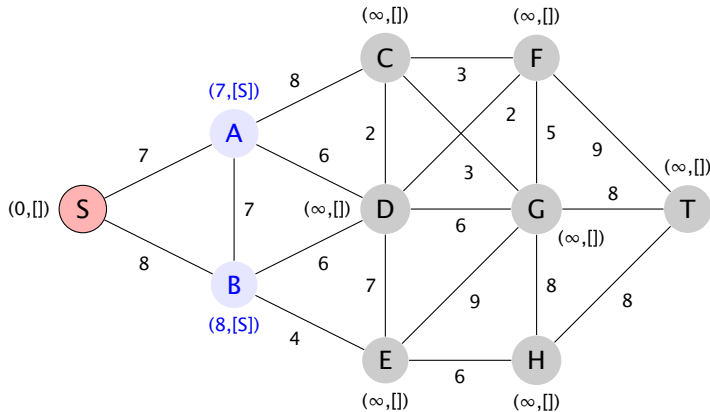
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 1 Process S egDijkstraGraph0201



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

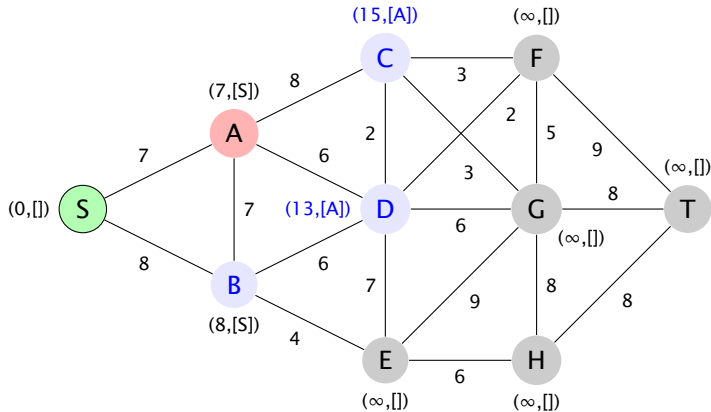
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 2 Process A egDijkstraGraph0202



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

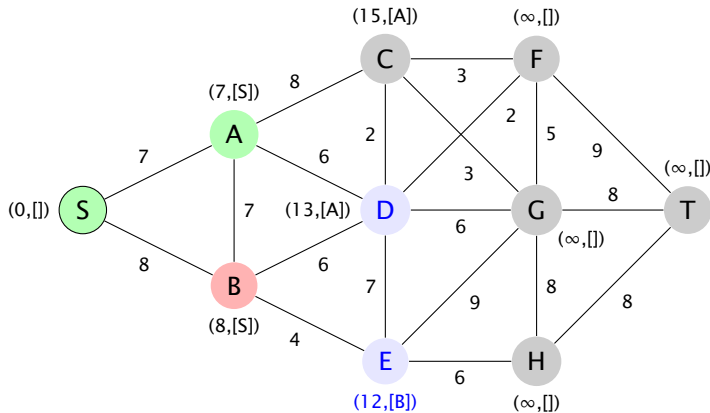
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 3 Process B egDijkstraGraph0203



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

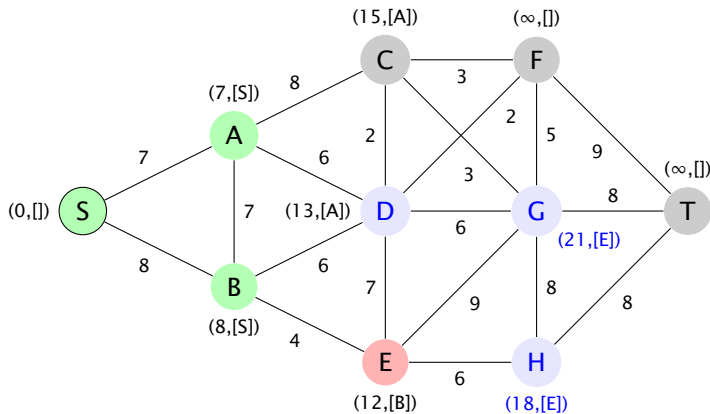
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 4 Process E egDijkstraGraph0204



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

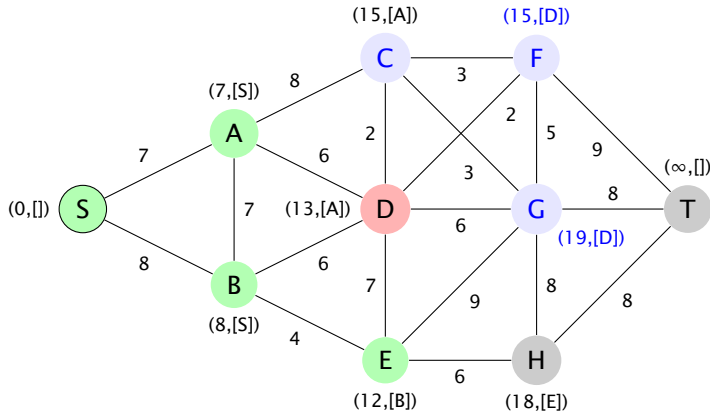
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

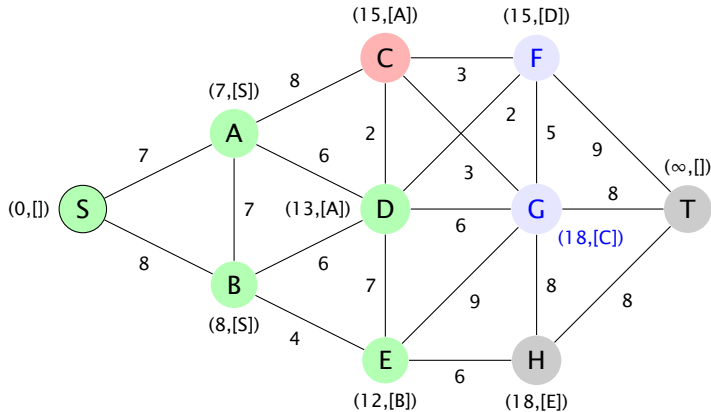
Step 5 Process D egDijkstraGraph0205



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

Dijkstra's Algorithm Example 02

Step 6 Process C (or F) egDijkstraGraph0206



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

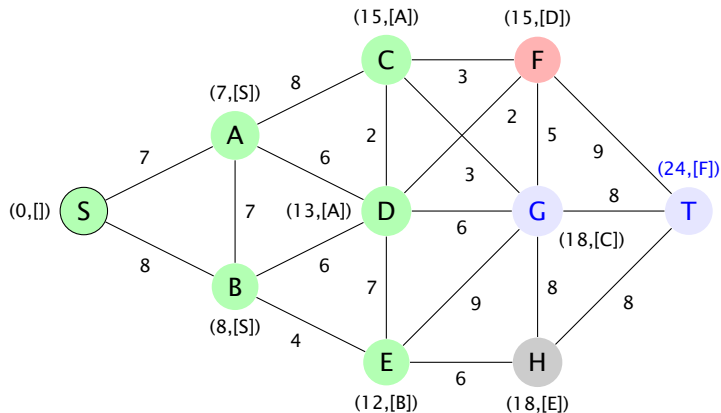
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 7 Process F egDijkstraGraph0207



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

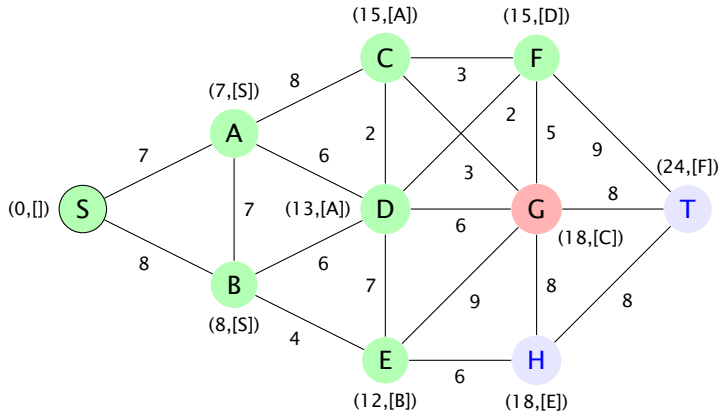
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 8 Process G (or H) egDijkstraGraph0208



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

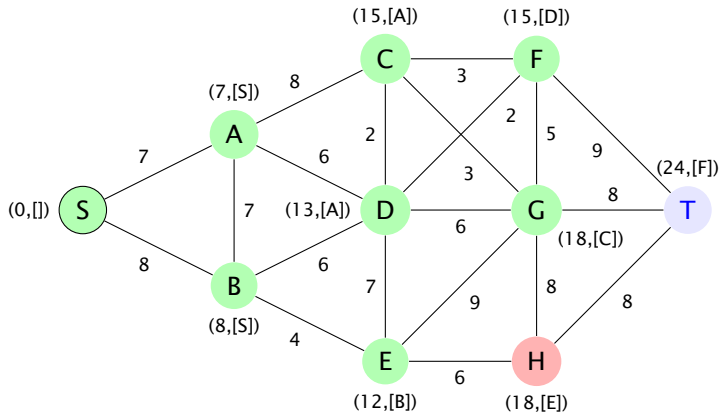
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 9 Process H egDijkstraGraph0209



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

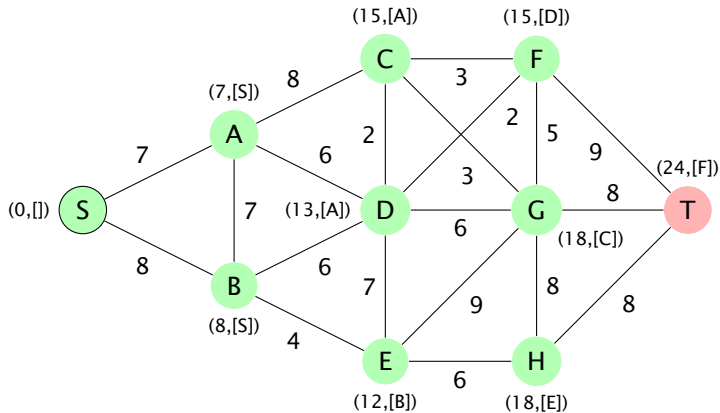
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Step 10 Process T egDijkstraGraph0210



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

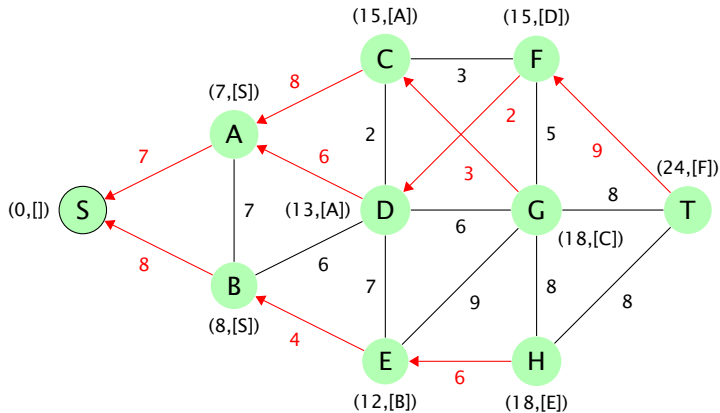
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Shortest Path Tree egDijkstraGraph02SPT



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

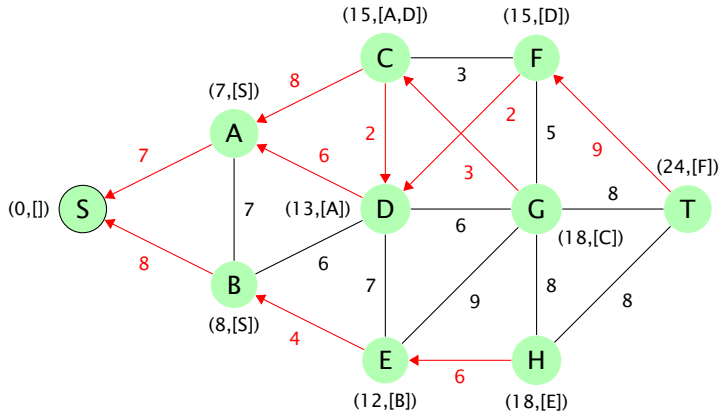
Greedy Algorithms

Future Work

Web Sites & References

Dijkstra's Algorithm Example 02

Shortest Path Graph egDijkstraGraph02SPG



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

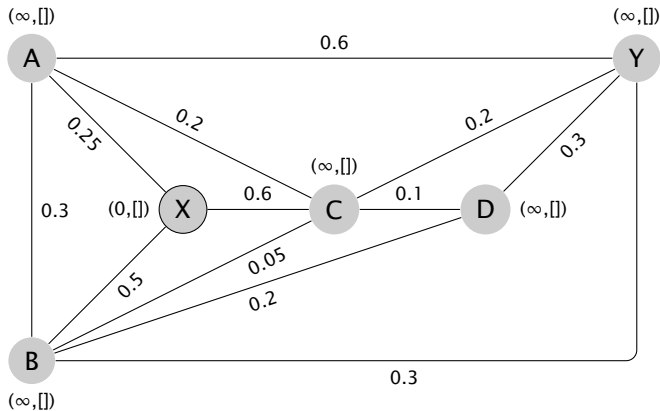
Dijkstra's Algorithm Example 03

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

Dijkstra's Algorithm Example 03

Step 0 Initialisation egDijkstraGraph0300



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Dijkstra's Algorithm – Description

Dijkstra's Algorithm Example 01

Dijkstra's Algorithm – Further points

Dijkstra's Algorithm Example 02

Dijkstra's Algorithm Example 03

Prim's Algorithm

Greedy Algorithms

Future Work

Web Sites & References

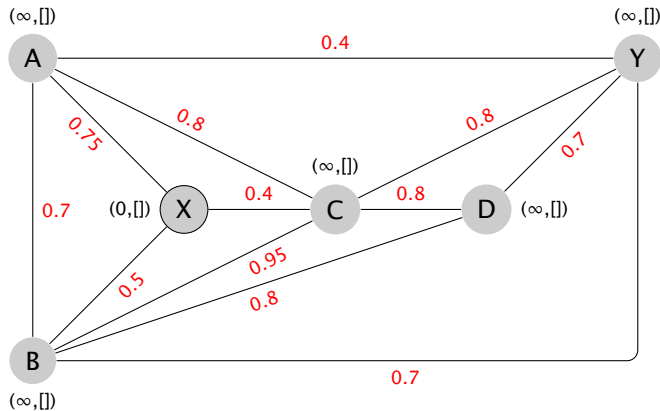
Dijkstra's Algorithm Example 03

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 $\sum_{(i,j) \in \text{path}} z_{(i,j)}$

Dijkstra's Algorithm Example 03

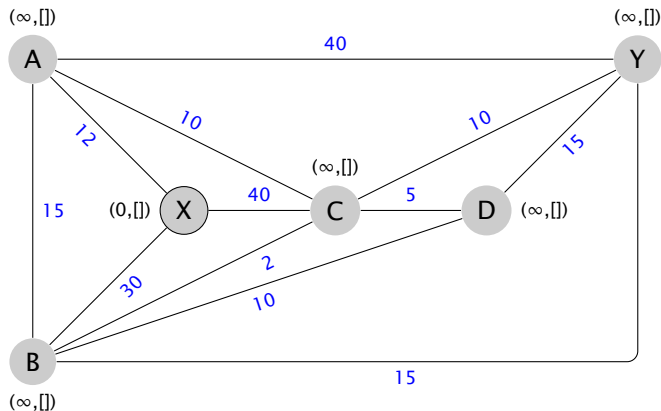
Step 0 Reformulation (a) egDijkstraGraph0300a



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

Dijkstra's Algorithm Example 03

Step 0 Reformulation (b) egDijkstraGraph0300b



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

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
```

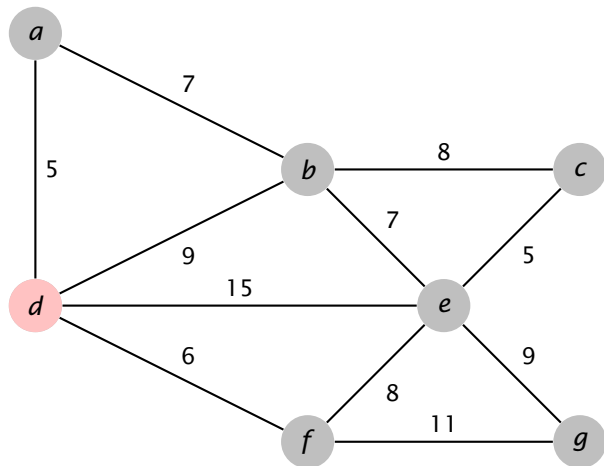
Dijkstra's and Prim's Algorithms

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 (2009, page 648) has a footnote explaining the origin of the term *relaxation*

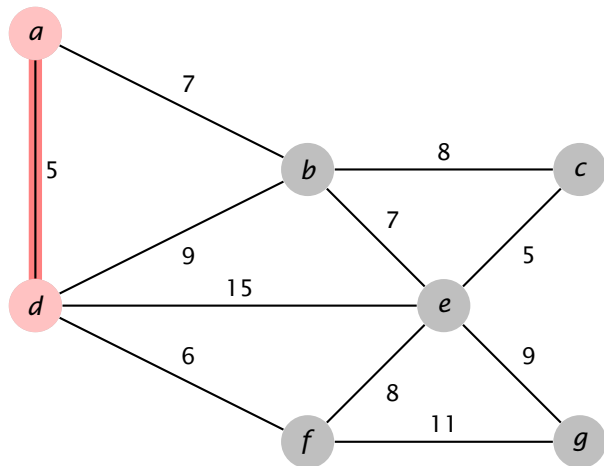
Tutorial Material — Prim's algorithm

Example Graph 01 egPrimGraph00



Tutorial Material — Prim's algorithm

Example Graph 01 egPrimGraph00



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Prim's Algorithm — Description

Prim's Algorithm — Example

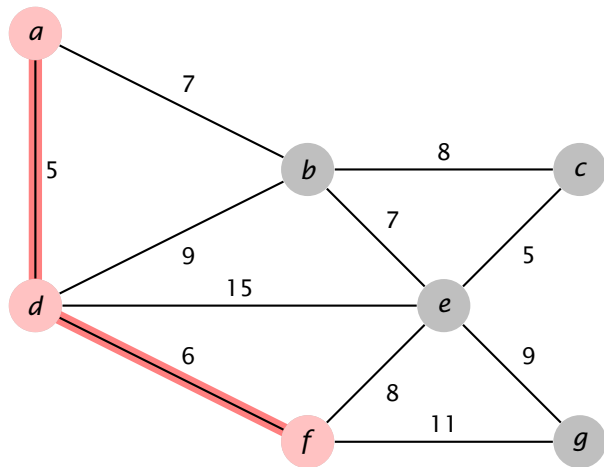
Greedy Algorithms

Future Work

Web Sites & References

Tutorial Material — Prim's algorithm

Example Graph 01 egPrimGraph00



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Prim's Algorithm — Description

Prim's Algorithm — Example

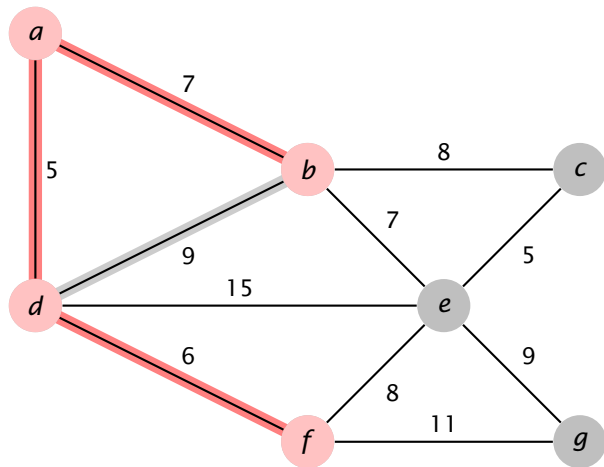
Greedy Algorithms

Future Work

Web Sites & References

Tutorial Material — Prim's algorithm

Example Graph 01 egPrimGraph00



Agenda

Adobe Connect

M269 Graph Algorithms

Algorithm Descriptions & Implementations

Topological Sort

Dijkstra's Algorithm

Prim's Algorithm

Prim's Algorithm — Description

Prim's Algorithm — Example

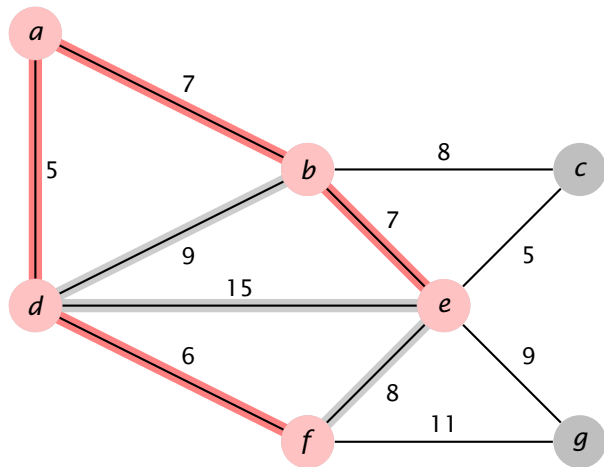
Greedy Algorithms

Future Work

Web Sites & References

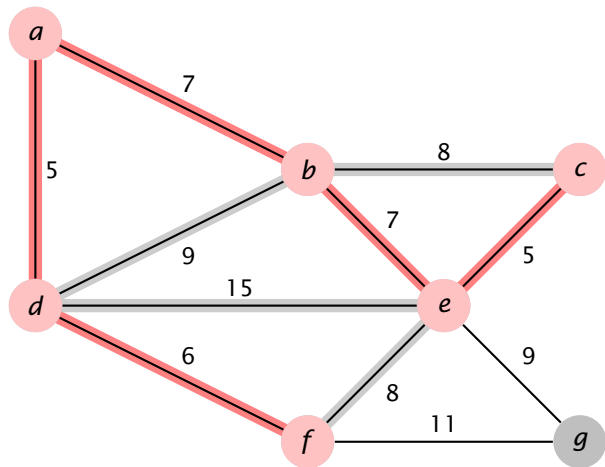
Tutorial Material — Prim's algorithm

Example Graph 01 egPrimGraph00



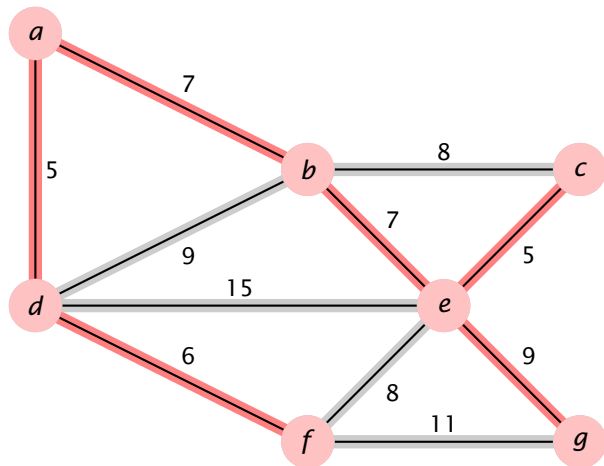
Tutorial Material — Prim's algorithm

Example Graph 01 egPrimGraph00



Tutorial Material — Prim's algorithm

Example Graph 01 egPrimGraph00



Greedy Algorithms

Overview

- ▶ **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**.

Greedy Algorithms

Interval Scheduling

- ▶ [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 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

Interval Scheduling

Example

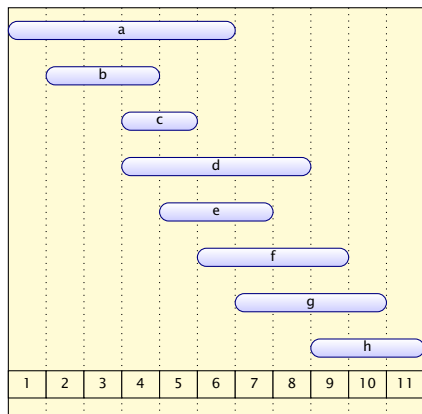
- ▶ For the jobs given below, produce an ordering by each of the greedy templates (above) and the schedule produced
- ▶ Each triple in the list below means $(name, s_i, f_i)$ where the times are inclusive

```
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)]
```

[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Interval Scheduling](#)[Future Work](#)[Web Sites & References](#)

Interval Scheduling

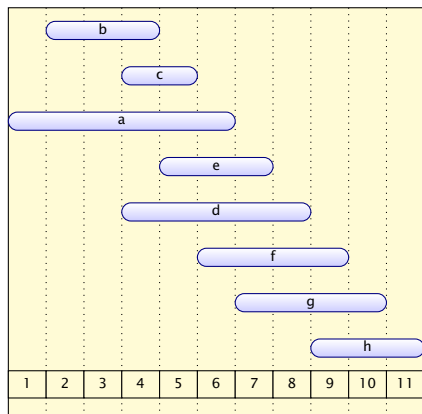
Order by Earliest Start Time eg Gantt01 EST



► Schedule jobs **a**, **g** (2 jobs)

Interval Scheduling

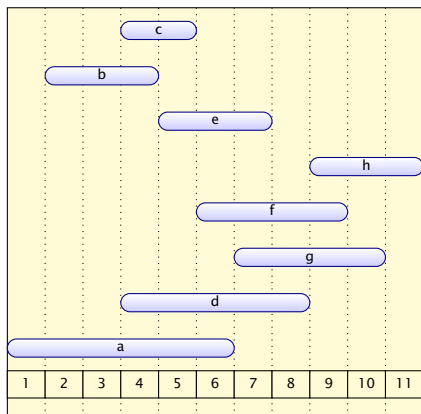
Order by Earliest Finish Time eg Gantt01EFT



► Schedule jobs **b, e, h** (3 jobs)

Interval Scheduling

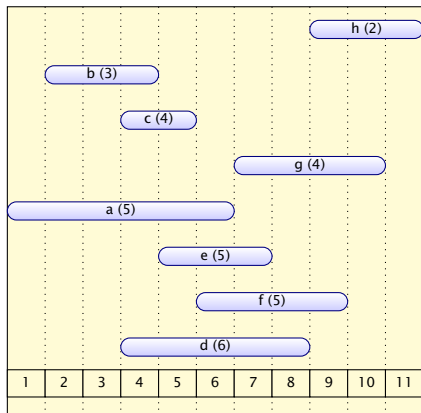
Order by Shortest Interval egGantt01 ShortInt



► Schedule jobs **c**, **h** (2 jobs)

Interval Scheduling

Order by Fewest Conflicts egGantt01Conflicts



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

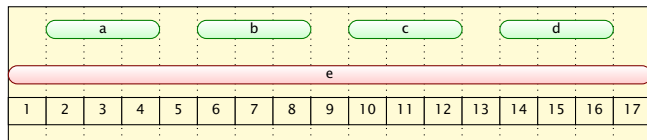
Interval Scheduling

Counter Examples

- ▶ 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

Interval Scheduling

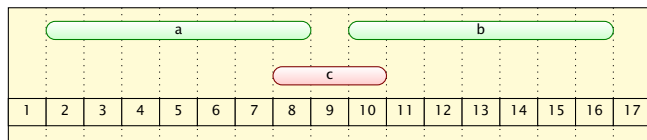
Order by Earliest Start Time — Counter Example egGantt01 ESTcctr



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

Interval Scheduling

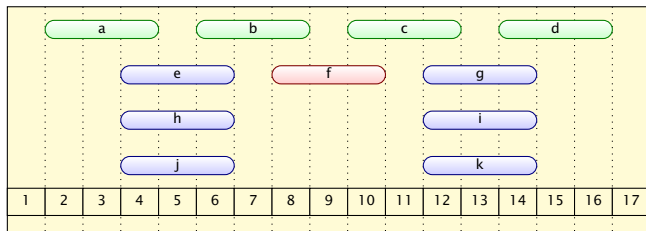
Order by Shortest Interval — Counter Example egGantt01ShortIntCnt



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

Interval Scheduling

Order by Fewest Conflicts — Counter Example egGantt01 ConflictsCntr



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

Interval Scheduling

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 (2013)

[Agenda](#)[Adobe Connect](#)[M269 Graph Algorithms](#)[Algorithm Descriptions & Implementations](#)[Topological Sort](#)[Dijkstra's Algorithm](#)[Prim's Algorithm](#)[Greedy Algorithms](#)[Interval Scheduling](#)[Future Work](#)[Web Sites & References](#)

Future Work

Graph algorithms, Greed, Logic, Computability

- ▶ Thursday, 12 March 2026 TMA02
- ▶ Sunday, 12 April 2026 Tutorial Online (Module wide)
Dynamic Programming
- ▶ Sunday, 26 April 2026 Tutorial Online (Module wide)
Computability and Complexity
- ▶ Sunday, 3 May 2026 Tutorial Online Review of course
material for TMA03
- ▶ Thursday, 28 May 2026 TMA03

References

Graph Theory

- ▶ **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/>

References

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

References

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/>
- ▶ [?](#), 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