

Dynamic Programming

M269 Tutorial with Commentary 2023

Contents

Commentary 1	2
1 Agenda	2
2 Adobe Connect	3
2.1 Interface	3
2.2 Settings	4
2.3 Sharing Screen & Applications	5
2.4 Ending a Meeting	6
2.5 Invite Attendees	6
2.6 Layouts	7
2.7 Chat Pods	8
2.8 Web Graphics	8
2.9 Recordings	8
Commentary 2	9
3 DP Introduction	9
4 List Comprehensions	10
4.1 List Comprehensions	10
Activity 1 List Comprehension Exercises	10
Commentary 3	16
5 Dynamic Programming	16
5.1 Fibonacci Sequence	16
5.1.1 Fibonacci Closed Form	18
5.1.2 Fibonacci Alternative Calculation	20
5.2 DP Formulation	21
5.3 Edit Distance	21
5.4 Edit Distance Definitions	21
5.5 Edit Distance: Recursive	22
5.6 Edit Distance — Implementation	24
5.7 Edit Distance: Haskell	24
5.8 Edit Distance: Python	27
5.9 Edit Distance Examples	29
5.10 Edit Distance Diagram Construction	31
Commentary 4	39
6 Future Work	39
7 Web Sites & References	39
7.1 Edit Distance Sources	39

7.2 Algorithm Texts & Web Sites	40
7.3 Analysis of Algorithms	40
7.4 Graph Algorithms	40
7.5 Shortest Paths	40
7.6 Web Sites for Dynamic Programming	41
References	41
Python Code Index	44
Haskell Code Index	45
Diagrams Index	46

Commentary 1

1 Agenda, Aims and Topics

- Overview of aims of tutorial
- Note selection of topics
- Recursion is used throughout the topics
- Points about my own background and preferences
- Adobe Connect slides for reference

[ToC](#)

1 Agenda

- Welcome and introductions
- Dynamic Programming — introduction
- Python: List comprehensions, Named Tuples
- Implementations in Structured English, Python and Haskell (Optional)
- *Note* there is more material here than we can cover — some is for optional interest
- Not covered in this session: section 2 Adobe Connect notes, sections 5.1.1 Fibonacci closed form, 5.1.2 Fibonacci Alternative Calculation, sections 5.7 Edit Distance — Haskell Implementation, 5.10 Edit Distance Diagram Construction
- Slides/Notes are at pmolyneux.co.uk/OU/M269FolderSync/M269TutorialNotes/M269Tutorial06
- **Recording** Meeting Record Meeting... ✓

Introductions — Phil

- *Name* Phil Molyneux
- *Background*
 - Undergraduate: Physics and Maths (Sussex)

- Postgraduate: Physics (Sussex), Operational Research (Brunel), Computer Science (University College, London)
 - Worked in Operational Research, Business IT, Web technologies, Functional Programming
- *First programming languages* [Fortran](#), [BASIC](#), [Pascal](#)
- *Favourite Software*
 - [Haskell](#) — pure functional programming language
 - Text editors [TextMate](#), [Sublime Text](#) — previously [Emacs](#)
 - Word processing in [L^AT_EX](#) — all these slides and notes
 - [Mac OS X](#)
- *Learning style* — I read the manual before using the software

Introductions — You

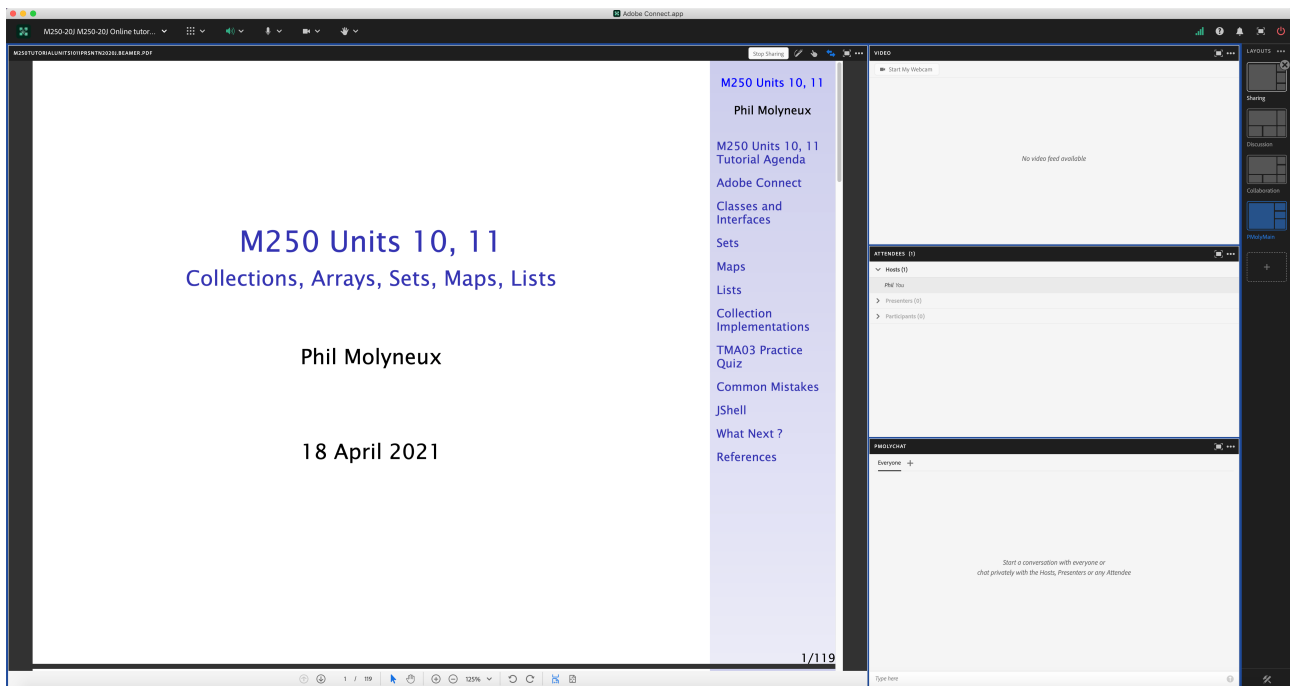
- *Name ?*
- *Favourite software/Programming language ?*
- *Favourite [text editor](#) or [integrated development environment \(IDE\)](#)*
- [List of text editors](#), [Comparison of text editors](#) and [Comparison of integrated development environments](#)
- *Other OU courses ?*
- *Anything else ?*

[ToC](#)

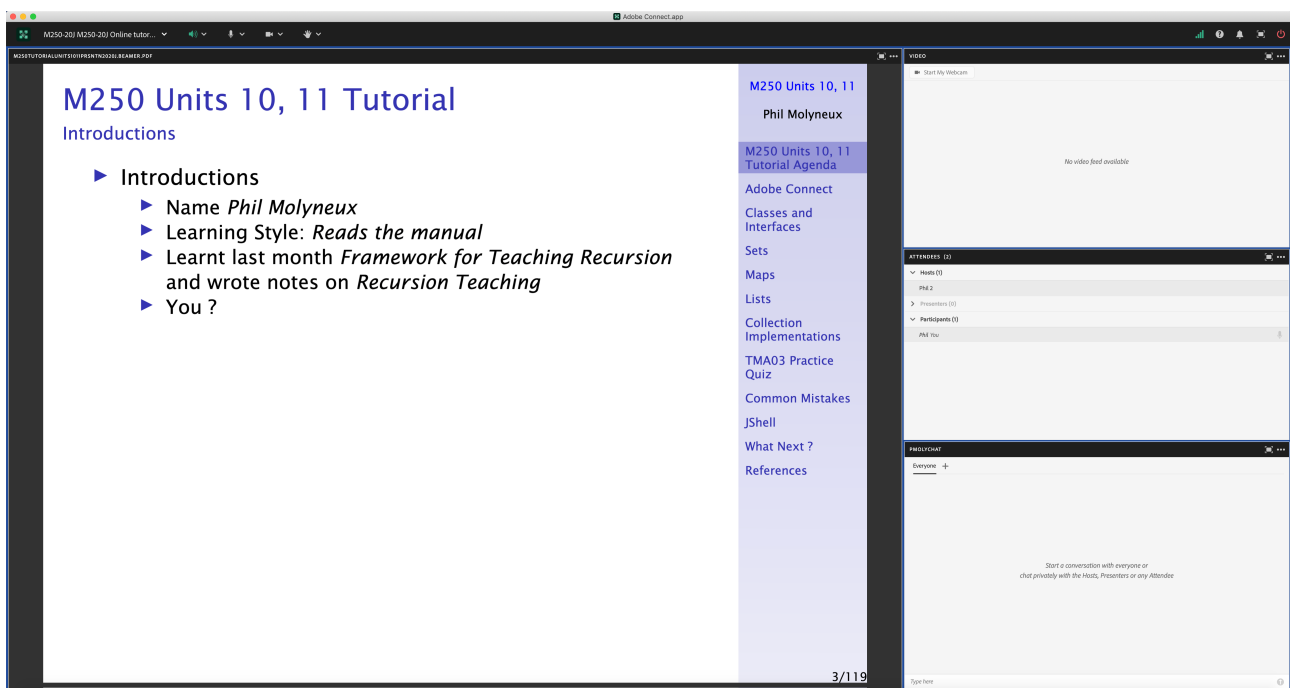
2 Adobe Connect Interface and Settings

2.1 Adobe Connect Interface

Adobe Connect Interface — Host View



Adobe Connect Interface — Participant View



2.2 Adobe Connect Settings

Adobe Connect — Settings

- **Everybody** **Menu bar** **Meeting** **Speaker & Microphone Setup**
- **Menu bar** **Microphone** **Allow Participants to Use Microphone** ✓
- Check Participants see the entire slide including slide numbers bottom right **Workaround**
 - **Disable Draw** **Share pod** **Menu bar** **Draw icon**
 - **Fit Width** **Share pod** **Bottom bar** **Fit Width icon** ✓

- **Meeting** > Preferences > General > **Host Cursor** > Show to all attendees
- **Menu bar** > Video > Enable **Webcam** for Participants ✓
- Do not *Enable single speaker mode*
- Cancel hand tool
- Do not enable green pointer
- **Recording** > Meeting > Record Session ✓
- **Documents** Upload PDF with drag and drop to share pod
- Delete > Meeting > Manage Meeting Information > Uploaded Content and > check filename > click on delete

Adobe Connect — Access

• Tutor Access

TutorHome > M269 Website > Tutorials

Cluster Tutorials > M269 Online tutorial room

Tutor Groups > M269 Online tutor group room

Module-wide Tutorials > M269 Online module-wide room

• Attendance

TutorHome > Students > View your tutorial timetables

• Beamer Slide Scaling 440% (422 x 563 mm)

• Clear Everyone's Status

Attendee Pod > Menu > Clear Everyone's Status

• Grant Access and send link via email

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

• Presenter Only Area

Meeting > Enable/Disable Presenter Only Area

Adobe Connect — Keystroke Shortcuts

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

2.3 Adobe Connect — Sharing Screen & Applications

- Share My Screen > Application tab > Terminal for [Terminal](#)
- Share menu > Change View > Zoom in for mismatch of screen size/resolution (Participants)

- (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- Leave the application on the original display
- Beware blue hatched rectangles — from other (hidden) windows or contextual menus
- Presenter screen pointer affects viewer display — beware of moving the pointer away from the application
- First time: **System Preferences** > **Security & Privacy** > **Privacy** > **Accessibility**

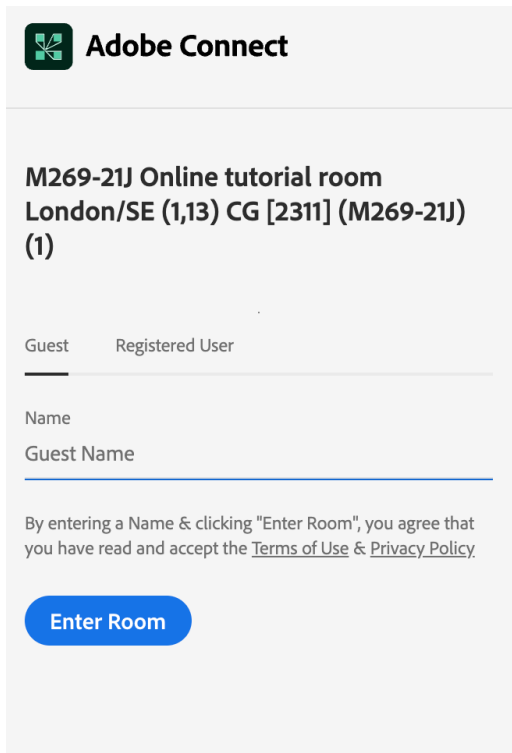
2.4 Adobe Connect — Ending a Meeting

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

2.5 Adobe Connect — Invite Attendees

- **Provide Meeting URL** **Menu** > **Meeting** > **Manage Access & Entry** > **Invite Participants...**
- **Allow Access without Dialog** **Menu** > **Meeting** > **Manage Meeting Information** provides new browser window with *Meeting Information* **Tab bar** > **Edit Information**
- Check *Anyone who has the URL for the meeting can enter the room*
- Default *Only registered users and accepted guests may enter the room*
- **Reverts to default next session but URL is fixed**
- Guests have blue icon top, registered participants have yellow icon top — same icon if URL is open
- See [Start, attend, and manage Adobe Connect meetings and sessions](#)
- Click on the link sent in email from the Host
- Get the following on a Web page

- As *Guest* enter your name and click on **Enter Room**



Adobe Connect

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

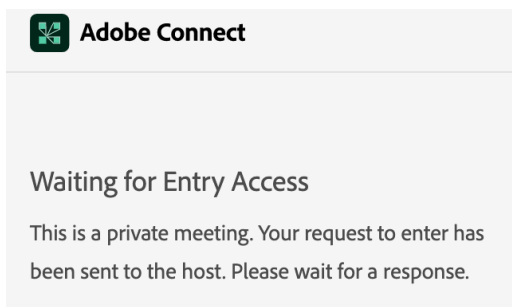
Guest Registered User

Name
 Guest Name

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

Enter Room

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

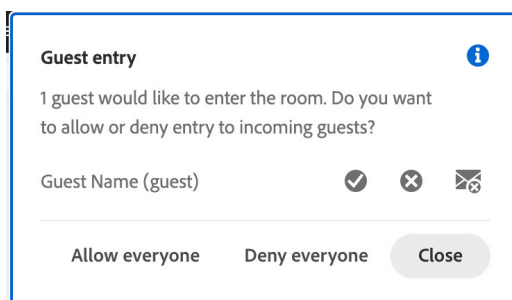


Adobe Connect

Waiting for Entry Access

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

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



Guest entry

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

Guest Name (guest) ✓ ✕ 🔒

Allow everyone Deny everyone Close

2.6 Layouts

- **Creating new layouts** example *Sharing* layout
- **Menu** > **Layouts** > **Create New Layout...** > **Create a New Layout dialog** > **Create a new blank layout** and name it *PMolyMain*
- New layout has no Pods but does have Layouts Bar open (see Layouts menu)

- **Pods**
- **Menu** > **Pods** > **Share** > **Add New Share** and resize/position — initial name is *Share n*
- **Rename Pod** **Menu** > **Pods** > **Manage Pods...** > **Manage Pods** > **Select** > **Rename** or **Double-click & rename**
- Add Video pod and resize/reposition
- Add Attendance pod and resize/reposition
- Add Chat pod — name it *PMolyChat* — and resize/reposition
- Dimensions of **Sharing** layout (on 27-inch iMac)
 - Width of Video, Attendees, Chat column 14 cm
 - Height of Video pod 9 cm
 - Height of Attendees pod 12 cm
 - Height of Chat pod 8 cm
- **Duplicating Layouts** does *not* give new instances of the Pods and is probably not a good idea (apart from local use to avoid delay in reloading Pods)
- **Auxiliary Layouts** name *PMolyAuxOn*
 - Create new Share pod
 - Use existing Chat pod
 - Use same Video and Attendance pods

2.7 Chat Pods

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

2.8 Graphics Conversion for Web

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

2.9 Adobe Connect Recordings

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

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

[ToC](#)

Commentary 2

2 DP Introduction, List Comprehensions

- Overview of Dynamic Programming
- Obtain recursive algorithm for problem but implement bottom up
- Introduction to list comprehensions as a concise way of iterating over lists (or other iterables)
- List comprehension exercises with solutions

[ToC](#)

3 Dynamic Programming — Introduction

- **Dynamic Programming** invented by **Richard Bellman** in the 1950s
- *Programming* meant *planning* the sequence of decisions
- *Dynamic* suggested evolution of the system over time
- Attributes: (1) optimal substructure (2) overlapping subproblems
- Divide and conquer has non-overlapping subproblems — a tree-structure
- DP has an acyclic graph structure
- Dynamic Programming process:
- Obtain a recursive solution
- Two ways to avoid subproblems being calculated more than once:
- **Memoization** uses the top-down recursive structure but preserves subproblems in a table for subsequent retrieval
- Tabulation works bottom-up which orders the computations so that simplest results calculated first and dependent problems follow on in some order

[ToC](#)

4 List Comprehensions

4.1 List Comprehensions

List Comprehensions — Python

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

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

- In general

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

- Lots example usage in the algorithms below

List Comprehensions — Haskell

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

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

- In general

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

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

Activity 1 (a) Stop Words Filter

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

```
11 sentence \
12   = "the_quick_brown_fox_jumps_over_the_lazy_dog"
14 words = sentence.split()
```

```

16 wordsTest \
17   = (words == ['the', 'quick', 'brown'
18               , 'fox', 'jumps', 'over'
19               , 'the', 'lazy', 'dog'])
21 stopWords \
22   = ['a', 'an', 'the', 'that']

```

[Go to Answer](#)**Activity 1 (a) Stop Words Filter**

```

11 sentence \
12   = "the_quick_brown_fox_jumps_over_the_lazy_dog"
14 words = sentence.split()
16 wordsTest \
17   = (words == ['the', 'quick', 'brown'
18               , 'fox', 'jumps', 'over'
19               , 'the', 'lazy', 'dog'])
21 stopWords \
22   = ['a', 'an', 'the', 'that']

```

- Notice the [Python Explicit line joining](#) with (`\<n1>`) and [Python Implicit line joining](#) with (`(...)`)
- The [backslash](#) (`\`) must be followed by an [end of line character](#) (`<n1>`)
- The (`' '`) symbol represents a space (see [Unicode U+2423 Open Box](#))

[Go to Answer](#)**Activity 1 (b) Transpose Matrix**

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

```

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

```

- Using list comprehensions, write a function [transMat](#), to transpose a matrix

[Go to Answer](#)**Activity 1 (c) List Pairs in Fair Order**

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

```

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

```

```

71     , (1, 0), (1, 1), (1, 2), (1, 3), (1, 4)
72     , (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)
73     , (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)
74     , (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]]

```

[Go to Answer](#)

Activity 1 (c) List Pairs in Fair Order

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

```

81 fairLstTest \
82   = (fairListing(5,5)
83     == [(0, 0)
84         , (0, 1), (1, 0)
85         , (0, 2), (1, 1), (2, 0)
86         , (0, 3), (1, 2), (2, 1), (3, 0)
87         , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])

```

[Go to Answer](#)

Activity 1 (c) List Pairs in Fair Order

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

```

94 fairLstATest \
95   = (fairListingA(5,5)
96     == [[(0, 0)]
97         , [(0, 1), (1, 0)]
98         , [(0, 2), (1, 1), (2, 0)]
99         , [(0, 3), (1, 2), (2, 1), (3, 0)]
100        , [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]]

```

[Go to Answer](#)

Answer 1 (a) Stop Words Filter

- Answer 1 (a) Stop Words Filter
- *Write here:*

Answer 1 (a) Stop Words Filter

- Answer 1 (a) Stop Words Filter

```

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

31 filterStopWordsTest \
32   = filterStopWords(words) \

```

```

33 == ['quick', 'brown', 'fox'
34     , 'jumps', 'over', 'lazy', 'dog']

```

[Go to Activity](#)**Answer 1 (b) Transpose Matrix**

- Answer 1 (b) Transpose Matrix
- *Write here:*

Answer 1 (b) Transpose Matrix

- Answer 1 (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
54
55 transMatTestA \
56 = (transMat(matrixA)
57    == matATr)

```

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

[Go to Activity](#)**Answer 1 (b) Transpose Matrix**

- Note the differences in the list comprehensions below

```

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

```

```

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

```

[Go to Activity](#)**Answer 1 (b) Transpose Matrix**

- Answer 1 (b) Transpose Matrix
- The Python [NumPy](#) package provides functions for N-dimensional array objects
- For transpose see [numpy.ndarray.transpose](#)

```

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

```

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

[Go to Activity](#)**Answer 1 (c) List Pairs in Fair Order**

- Answer 1 (c) List Pairs in Fair Order — first version
- Write here

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

[Go to Activity](#)**Answer 1 (c) List Pairs in Fair Order**

- Answer 1 (c) List Pairs in Fair Order
- This is the *obvious* but biased version

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

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

[Go to Activity](#)**Answer 1 (c) List Pairs in Fair Order**

- Answer 1 (c) List Pairs in Fair Order — second version
- Write here

```
83 fairLstTest \
84 = (fairListing(5,5)
85    == [(0, 0)
86        , (0, 1), (1, 0)
87        , (0, 2), (1, 1), (2, 0)
88        , (0, 3), (1, 2), (2, 1), (3, 0)
89        , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])
```

[Go to Activity](#)**Answer 1 (c) List Pairs in Fair Order**

- Answer 1 (c) List Pairs in Fair Order — second version

- This works by making the sum of the coordinates the same for each prefix

```

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

83 fairLstTest \
84 = (fairListing(5,5)
85    == [(0, 0)
86        , (0, 1), (1, 0)
87        , (0, 2), (1, 1), (2, 0)
88        , (0, 3), (1, 2), (2, 1), (3, 0)
89        , (0, 4), (1, 3), (2, 2), (3, 1), (4, 0)])

```

[Go to Activity](#)

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — third version
- Write here

```

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

```

[Go to Activity](#)

Answer 1 (c) List Pairs in Fair Order

- Answer 1 (c) List Pairs in Fair Order — third version
- The *inner loop* is placed into its own list comprehension

```

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

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

```

[Go to Activity](#)

[ToC](#)

Python & Haskell Tutorials

- Python tutorials:
 - [Beginner's Python Tutorial](#)
 - [Python Programming](#)
 - [Non-Programmer's Tutorial for Python 3](#)
 - [Non-Programmer's Tutorial for Python 2.6](#)

- Haskell Tutorials:
 - [Haskell Wikibook](#)
 - [What I Wish I Knew When Learning Haskell](#)
 - [Haskell Meta-tutorial](#)
 - [Learn You a Haskell for Great Good](#)
 - [Real World Haskell](#)

[ToC](#)

Commentary 3

3 DP Introduction, List Comprehensions

- Fibonacci sequence
- Recursive definition and simple recursive program
- Recursive but more efficient program
- Original implementation in Haskell (optional)
- Implementation in Python
- Edit Distance examples
- Edit Distance diagram construction — generating the $\text{\LaTeX}$ for the diagrams (optional)

[ToC](#)

5 Dynamic Programming

5.1 Fibonacci Sequence

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

Fibonacci Sequence — Python Recursive (1)

- Recursive definitions are recursive algorithms

```
1 def fibs1(n):  
2     return fibs1indent(n,0)  
4 def fibs1indent(n,indent):
```

```

5 indentStr = '_'*indent
6 print(indentStr + 'fibs(' + str(n) + ')')

8 if n == 0 :
9     return 0
10 elif n == 1 :
11     return 1
12 else:
13     return (fibs1indent(n - 2, indent + 2)
14             + fibs1indent(n - 1, indent + 2))

```

```

1 Python3>>> fibs1(5)
2 fibs(5)
3   fibs(3)
4     fibs(1)
5     fibs(2)
6       fibs(0)
7       fibs(1)
8   fibs(4)
9     fibs(2)
10      fibs(0)
11      fibs(1)
12    fibs(3)
13      fibs(1)
14      fibs(2)
15        fibs(0)
16        fibs(1)
17 5
18 Python3>>>

```

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

Fibonacci Sequence — Python Recursive (2)

- Recursion but remembering enough to avoid most

```

1 def fibs2(n):
2     return fibs2indent(n,0)

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

8     if n == 0 :
9         return [first]
10    else:
11        return ([first]
12                + fibs2indent(n - 1, indent + 2
13                              , second, first + second))

```

```

1 Python3>>> fibs2(5)
2 fibs(5)
3   fibs(4)
4     fibs(3)
5     fibs(2)
6       fibs(1)
7       fibs(0)
8 [0, 1, 1, 2, 3, 5]
9 Python3>>>

```

- This takes 6 calls to `fibs2()`
- `fibs2()` $T(n) = T(n - 1) + 1 = n + 1$
- `fibs1()` $T(n) = T(n - 1) + T(n - 2) + 1 = 2F_{n+1} - 1$
- `fibs1(1)` gets called F_n times
- `fibs1(0)` gets called F_{n-1} times

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

5.1.1 Fibonacci Closed Form

- $F_n = \frac{\phi^n - (1 - \phi)^n}{\sqrt{5}}$
- [Euler-Binet Formula](#)
- [A formula for Fib\(n\)](#)
- ϕ is the [Golden mean](#)
- $\phi = \frac{1}{\phi - 1} = \frac{1 + \sqrt{5}}{2}$
- Also known as the [Golden ratio](#)
- $\phi \stackrel{\text{def}}{=} \frac{a}{b} = \frac{a+b}{a}$

Proofs of Euler-Binet Formula

- [ProofWiki: Euler-Binet Formula](#) gives 4 proofs:
 - (1) Proof by induction — straightforward but doesn't really tell you how to get the formula in the first place
 - (2) Proof by matrix algebra — find the eigenvalues and eigenvectors of a matrix that generates the Fibonacci sequence
 - (3) Proof from the [ProofWiki: Binet Form](#)

$$U_n = mU_{n-1} + U_{n-2} \text{ where } U_0 = 0 \text{ and } U_1 = 1$$
 - (4) Proof from the [ProofWiki: Generating Function for Fibonacci Numbers](#)

$$G(z) = \frac{z}{1 - z - z^2}$$

Matrix Algebra Proof of Euler-Binet Formula

- Let $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
- Then $A^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$
- Proof by induction
- $A^1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix}$
- Assume $A^k = \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix}$
- Then $A \times A^k = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{pmatrix}$

$$= \begin{pmatrix} F_{k+1} + F_k & F_k + F_{k-1} \\ F_{k+1} & F_k \end{pmatrix} = \begin{pmatrix} F_{k+2} & F_{k+1} \\ F_{k+1} & F_k \end{pmatrix}$$

- Demonstrate the eigenvalues of A are ϕ and $\hat{\phi} = 1 - \phi$
- Solve $\det(\lambda I - A) = 0$
- $\det \left(\lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \right) = 0$
- $\det \begin{pmatrix} \lambda - 1 & -1 \\ -1 & \lambda \end{pmatrix} = 0$
- $\lambda^2 - \lambda - 1 = 0$
- $ax^2 + bx + c = 0$ has roots $= \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- Hence $\phi = \frac{1 + \sqrt{5}}{2}$ and $\hat{\phi} = 1 - \phi = \frac{1 - \sqrt{5}}{2}$
- and $A \begin{pmatrix} \phi \\ 1 \end{pmatrix} = \begin{pmatrix} \phi + 1 \\ \phi \end{pmatrix} = \phi \begin{pmatrix} \phi \\ 1 \end{pmatrix}$ as $\phi^2 - \phi - 1 = 0$
- and $A \begin{pmatrix} \hat{\phi} \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{\phi} + 1 \\ \hat{\phi} \end{pmatrix} = \hat{\phi} \begin{pmatrix} \hat{\phi} \\ 1 \end{pmatrix}$ as $\hat{\phi}^2 - \hat{\phi} - 1 = 0$
- Hence $\begin{pmatrix} \phi \\ 1 \end{pmatrix}$ is an **eigenvector** of A with **eigenvalue** ϕ
- and $\begin{pmatrix} \hat{\phi} \\ 1 \end{pmatrix}$ is an **eigenvector** of A with **eigenvalue** $\hat{\phi}$
- We have to have $\det(\lambda I - A) = 0$ **noninvertible**, singular
- Proof: assume $X \neq 0$, $AX = \lambda X$ and $(\lambda I - A)$ is invertible
- Then $X = IX = ((\lambda I - A)^{-1}(\lambda I - A))X$

$$= (\lambda I - A)^{-1}((\lambda I - A)X)$$

$$= (\lambda I - A)^{-1}0$$

$$= 0 \text{ contradiction}$$
- Hence $A^n \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} = \phi^n \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$
- and $A^n \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} = \hat{\phi}^n \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$
- Also $A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$
- Also $\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} - \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$

- Hence $\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \left(\begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} - \begin{pmatrix} \hat{\phi} \\ \frac{1}{\sqrt{5}} \end{pmatrix} \right)$

$$= \phi^n \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} - \hat{\phi}^n \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$$
- Hence $A^n \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} = \phi^n \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$
- and $A^n \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} = \hat{\phi}^n \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$
- Also $A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$
- Also $\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} - \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$
- Hence $\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = A^n \left(\begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} - \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} \right)$

$$= \phi^n \begin{pmatrix} \frac{\phi}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} - \hat{\phi}^n \begin{pmatrix} \frac{\hat{\phi}}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix}$$

$$= \frac{1}{\sqrt{5}} \begin{pmatrix} \phi^n \cdot \phi - \hat{\phi}^n \cdot \hat{\phi} \\ \phi^n - \hat{\phi}^n \end{pmatrix}$$

$$= \frac{1}{\sqrt{5}} \begin{pmatrix} \phi^{n+1} - \hat{\phi}^{n+1} \\ \phi^n - \hat{\phi}^n \end{pmatrix}$$
- Hence $F_n = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)$

ToC

5.1.2 Fibonacci Alternative Calculation

- From [How to Compute Fibonacci Numbers?](#) and [Fibonacci Formulae](#)

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

- Proof by induction

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

- We now derive a method to compute [fib](#) in $O(\log n)$

```

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

```

```

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

```

[ToC](#)

5.2 DP Formulation

- Write a recursive algorithm for the problem
- Build solutions to the recurrence from the bottom up
 - Identify subproblems
 - Choose a data structure to memoize intermediate results
 - Identify dependencies between subproblems
 - Find an evaluation order so that each subproblem comes after the subproblems it depends on.
 - Implement the algorithm for the evaluation order

[ToC](#)

5.3 Edit Distance

5.4 Edit Distance Definitions

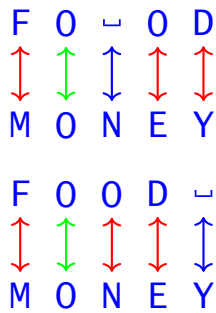
- **Edit Distance** or **Levenshtein Distance** is the minimum number of letter insertions, deletions and substitutions required to transform one word into another.
- Example **FOOD** → **MOOD** → **MOND** → **MONED** → **MONEY**
- A better way of displaying the transformation is in the next diagram

F	O	O	↖	D
↑	↑	↑	↑	↑
M	O	N	E	Y

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

Edit Distance — Example 1

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



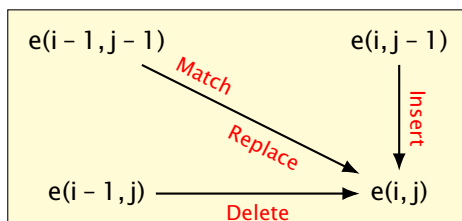
- How do we find such transformations in general ?

[ToC](#)

5.5 Edit Distance — Recursive Algorithm

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

Edit Distance Table Dependencies



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

Edit Distance — Base cases & Table Display

- The shortest way to convert $s(i)$ to an empty string ϵ is by i deletions
- The shortest way to convert an empty string ϵ to $t(j)$ is by j insertions

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

Edit Distance — Recursive Definition

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

- For simplicity we have assumed the cost of deletion, insertion, or replacement is 1 and the cost of a match is 0 (made more general in the implementation below)
- the running time of the algorithm is exponential in the length of s and t — but we do not care since we are going to implement it bottom up.
- The implementation below also provides the graph of the transformations by calculating the predecessors to each node in the table.

Edit Distance — Example 1 Table

edDistTblFoodMoney

	€	F	O	O	D
€	0 → 1 → 2 → 3 → 4				
M	1 ↓ 1 → 2 → 3 → 4				
O	2 ↓ 2 ↓ 1 → 2 → 3				
N	3 ↓ 3 ↓ 2 ↓ 2 → 3				
E	4 ↓ 4 ↓ 3 ↓ 3 ↓ 3				
Y	5 ↓ 5 ↓ 4 ↓ 4 ↓ 4				

5.6 Edit Distance — Implementation

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

5.7 Edit Distance — Haskell Implementation

```

1 module M269TutorialDynamicProgCmnty2023 where
2 import Data.List
3 import Data.Maybe

```

1. A Haskell script starts with a module header which starts with the reserved identifier, `module` followed by the module name, `M269TutorialDynamicProgCmnty2023`
2. The module name must start with an upper case letter and is the same as the file name (without its extension of `.lhs`)
3. Haskell uses *layout* (or the *off-side rule*) to determine scope of definitions, similar to Python
4. The body of the module follows the reserved identifier `where` and starts with two `import` declarations
5. These import the built-in libraries `Data.List` and `Data.Maybe`
6. We use the `sort` function from `Data.List`.
7. The `Maybe` datatype from `Data.Maybe` will be used at the end of this script to implement the graphs with data.

Edit Distance — Haskell Implementation — Type Declarations

```

5 type Dist = Int
6 type Pred = (Int,Int)
7 type EditDistCell = (Dist, [Pred])
8 type EditDistTable = [[EditDistCell]]

```

- The reserved identifier `type` introduces *type synonym declarations* to improve readability
- The finite-precision integer type `Int` covers the range $[-2^{63} = -9223372036854775808, 2^{63} - 1 = 9223372036854775807]$ (machine dependent)
- `(t1, t2)` is the type of a pair of `t1` and `t2`
- `[t3]` is the type of a list of elements all of type `t3`
- `[[t3]]` type of a list of lists of elements of type `t3`

Edit Distance — Haskell Implementation — Service Functions

```

10 edCellDist :: EditDistCell -> Dist
11 edCellDist (x, ps) = x

13 edCellPreds :: EditDistCell -> [Pred]
14 edCellPreds (x, ps) = ps

```

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

```
f :: t1 -> t2
```

- The above is a [type signature](#) for the variable `f`
- This specifies `f` has a function type which takes an element of type `t1` and returns an element of type `t2`
- The definition of `f` may be given without a type signature, in which case the system will infer the most general type
- Note that every function takes exactly one input and returns one output

```

16 getEdCell :: EditDistTable -> (Int,Int) -> EditDistCell
17 getEdCell edTbl (i,j) = edTbl!!i!!j

19 getEdDist :: EditDistTable -> (Int,Int) -> Dist
20 getEdDist edTbl (i,j)
21   = edCellDist (getEdCell edTbl (i,j))

23 getEdPreds :: EditDistTable -> (Int,Int) -> [Pred]
24 getEdPreds edTbl (i,j)
25   = edCellPreds (getEdCell edTbl (i,j))

```

- Function application is denoted by juxtaposition, is left associative and is more tightly binding than (almost) anything else
 - write `f x` and not `f (x)`
 - say `f x` as *f applied to x*
 - `f x y` means `(f x) y`

This notational convention has huge advantages — discuss and also see [Currying](#) and [Functional Programming in 5 Minutes](#)

- To be consistent, the function type arrow `(->)` is right associative
 - `f :: a -> b -> c` means `f :: a -> (b -> c)`
- Lists are denoted `[1,2,3]`, the empty list `[]`
- `(:)` prefixes an element to a list, `1:[2,3] == [1,2,3]`
- Parentheses over-ride precedence
- `(!!)` is the list indexing operator, 0-origin
 - `[1,2,3]!!1 == 2`

Edit Distance — Haskell Implementation — Costs

```

27 -- Cost of deletion, insertion, substitution
28 delC, insC, subC :: Dist
29 delC = 1
30 insC = 1
31 subC = 1

```

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

Edit Distance — Haskell Implementation — Algorithm

```

33 editDistTb1DP :: String -> String -> EditDistTable
34 editDistTb1DP s t
35   = edTb1
36   where
37     edTb1 = [ [ edTb1Cell (i,j) | j <- [0..length t] ]
38               | i <- [0..length s] ]
39     edTb1Cell (0,0) = (0,[])
40     edTb1Cell (i,0) = (i * delC, [(i-1,0)])
41     edTb1Cell (0,j) = (j * insC, [(0,j-1)])
42     edTb1Cell (i,j)
43       = (minD, preds)
44       where
45         possPreds = [(delD, (i-1,j))
46                     , (insD, (i,j-1))
47                     , (subD, (i-1,j-1))
48                     ]
49         delD = getEdDist edTb1 (i-1,j) + delC
50         insD = getEdDist edTb1 (i,j-1) + insC
51         subD = getEdDist edTb1 (i-1,j-1)
52               + (if s!!(i-1) == t!!(j-1) then 0 else subC)
53         minD = minimum [delD, insD, subD]
54         preds = [(i,j) | (x,(i,j)) <- possPreds, x == minD]

```

Edit Distance — Haskell Implementation — Examples

```

56 edTb1TF = editDistTb1DP "TREES" "FOREST"
57 edTb1TrTF
58   = transpose (editDistTb1DP "TREES" "FOREST")
59
60 edTb1AA = editDistTb1DP "ALGORITHM" "ALTRUISTIC"
61 edTb1TrAA
62   = transpose (editDistTb1DP "ALGORITHM" "ALTRUISTIC")
63
64 edTb1FM = editDistTb1DP "FOOD" "MONEY"
65 edTb1TrFM
66   = transpose (editDistTb1DP "FOOD" "MONEY")

```

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

Edit Distance — Haskell Implementation (2)

```

67 editDistTb1DP01 :: String -> String -> EditDistTable
68 editDistTb1DP01 s t
69   = edTb1
70   where
71     edTb1 = [ [ setEdTb1Cell (edTb1,s,t) (i,j)
72                   | j <- [0..length t] ]
73               | i <- [0..length s] ]

```

- Defining a subsidiary function `setEdTb1Cell` avoids having nested `where` clauses
- `setEdTb1Cell` takes the edit distance table `edTb1`, the start and target strings, and a pair of table indices and returns the table cell.

- The list comprehension definition here is recursive and works because the entries are calculated in row-major order.

```

75 setEdTblCell :: (EditDistTable, String, String)
76             -> (Int, Int) -> EditDistCell
77 setEdTblCell (edTbl, s, t) (0,0) = (0, [])
78 setEdTblCell (edTbl, s, t) (i,0) = (i * delC, [(i-1,0)])
79 setEdTblCell (edTbl, s, t) (0,j) = (j * insC, [(0,j-1)])
80 setEdTblCell (edTbl, s, t) (i,j)
81   = (minD, preds)
82   where
83     possPreds = [(delD, (i-1,j))
84                  ,(insD, (i,j-1))
85                  ,(subD, (i-1,j-1))]
86               ]
87     delD = getEdDist edTbl (i-1,j) + delC
88     insD = getEdDist edTbl (i,j-1) + insC
89     subD = getEdDist edTbl (i-1,j-1)
90           + (if s!!(i-1) == t!!(j-1) then 0 else subC)
91     minD = minimum [delD, insD, subD]
92     preds = [(i,j) | (x,(i,j)) <- possPreds, x == minD]

```

```

94 edTblTF01 = editDistTblDP01 "TREES" "FOREST"
95 edTblTrTF01
96   = transpose (editDistTblDP01 "TREES" "FOREST")
97
98 edTblAA01 = editDistTblDP "ALGORITHM" "ALTRUISTIC"
99 edTblTrAA01
100   = transpose (editDistTblDP01 "ALGORITHM" "ALTRUISTIC")
101
102 edTblFM01 = editDistTblDP "FOOD" "MONEY"
103 edTblTrFM01
104   = transpose (editDistTblDP01 "FOOD" "MONEY")

```

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

[ToC](#)

5.8 Edit Distance — Python Implementation

- The Python file [Python_activity_5.11.py](#) for Python activity 5.11 contains the code for the Dynamic Programming solution for the Levenshtein Edit Distance problem.
- It uses a double `for` loop to do the equivalent calculations of the edit distance table
- TODO rewrite using Python list comprehensions and compare with the Haskell version
- *Note* that the bulk of the code in 5.11 is to format the output and display one path.
- The Haskell to generate the edit table diagrams in these notes is available but not given here since it generates LaTeX TikZ/PGF which is not part of this course.
- The Python code below is based on the Haskell code (elsewhere in this document)
- **Health Warning** the code here should be regarded as near pseudo-code since a few of the Haskell features do not translate directly
- In particular, the Haskell code depended to some extent on the default [lazy evaluation strategy](#) which is a particular way of implementing non-strict evaluation
- Strict evaluation evaluates arguments of functions (except in special cases or by special constructs)

- Non-strict evaluation evaluates argument to functions at most once and if possible not at all — it evaluates on need
- Python does struct evaluation unless you use yield or other generators
- Haskell is non-strict unless you use strictness annotations
- This means the Python code below may need modifying
- Service functions
- Edit Distance Table is an array of cells
- Edit Distance cell is a pair of distance and a list of predecessors

```

21 def edCellDist(cell) :
22     return cell[0]

24 def edCellPreds(cell) :
25     return cell[1]

```

- functions to get specific cells
- `edTbl` is a list of list of cells

```

30 def getEdCell(edTbl, i, j) :
31     return edTbl[i][j]

33 def getEdDist(edTbl, i, j) :
34     return edCellDist(edTbl, i, j)

36 def getEdPreds(edTbl, i, j) :
37     return edCellPreds(edTbl, i, j)

```

- Cost of deletion, insertion, substitution

```

41 delC = 1
42 insC = 1
43 subC = 1

```

- Calculating the Edit Distance Table
- Calculate the entry in each cell
- `setEdTblCell` takes the from string, `s`, the to string, `t`, the cell indices, `i, j`, and returns a cell

```

52 def setEdTblCell(edTbl, s, t, i, j) :
53     if i == 0 and j == 0 :
54         return (0, [])
55     elif j == 0 :
56         return (i * delC, [(i-1, 0)])
57     elif i == 0 :
58         return (j * insC, [(0, j-1)])
59     else :
60         possPreds = [(delD, (i-1, j)),
61                     (insD, (i, j-1)),
62                     (subD, (i-1, j-1))]
63         delD = getEdDist(edTbl, i-1, j) + delC
64         insD = getEdDist(edTbl, i, j-1) + insC
65         subD = (getEdDist(edTbl, i-1, j-1)
66                 + (0 if s[i-1] == t[j-1] else subC))
67         minD = min([delD, insD, subD])
68         preds = [(i, j) for (x, (i, j)) in possPreds if x == minD]
69

```

- Calculate the Edit Distance Table in row-major order

```

73 def editDistTb1DP (s, t) :
74     edTb1 = [ [ setEdTb1Cell (edTb1, s, t, i, j)
75                 for j from range(len(t) + 1) ]
76                 for i from range(len(s) + 1) ]
78     return edTb1

```

[ToC](#)

5.9 Edit Distance Examples

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

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

edDistTb1AlgorithmAltruistic

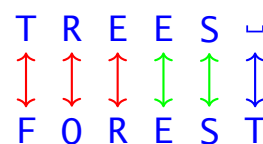
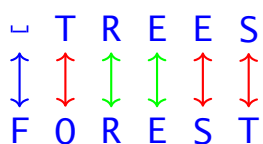
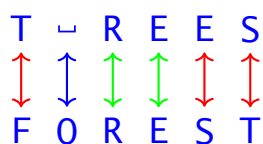
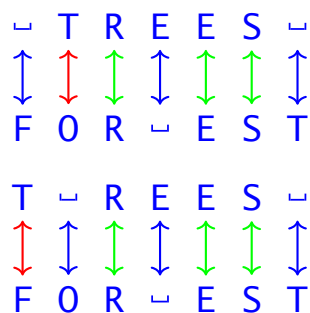
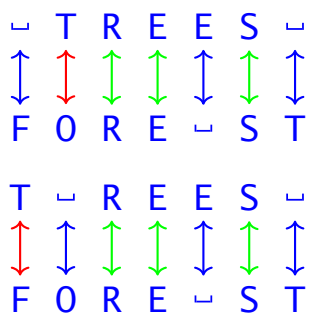
	ε	A	L	G	O	R	I	T	H	M
ε	0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9									
A	1	0 → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8								
L	2	1	0 → 1 → 2 → 3 → 4 → 5 → 6 → 7							
T	3	2	1	1 → 2 → 3 → 4 → 5 → 6						
R	4	3	2	2	2 → 3 → 4 → 5 → 6					
U	5	4	3	3	3	3 → 4 → 5 → 6				
I	6	5	4	4	4	4	3 → 4 → 5 → 6			
S	7	6	5	5	5	5	4	4 → 5 → 6		
T	8	7	6	6	6	6	5	4	4 → 5 → 6	
I	9	8	7	7	7	7	6	5	5 → 6	
C	10	9	8	8	8	8	7	6	6	6

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

edDistTblTreesForest

	ε	T	R	E	E	S
ε	0 → 1 → 2 → 3 → 4 → 5					
F	1	1 → 2 → 3 → 4 → 5				
O	2	2	2 → 3 → 4 → 5			
R	3	3	2 → 3 → 4 → 5			
E	4	4	3	2 → 3 → 4		
S	5	5	4	3	3 → 4	
T	6	5	5	4	4	4

- Alternative ways of representing the collection of transformation operations were given in subsection 5.4
- Here is a representation of the edit distance graph with **match**, **delete/insert**, and **replace** color arrows



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

TREES
 ↓insert
 FTREES
 ↓replace
 FOREES
 ↓delete
 FORES
 ↓insert
 FOREST

TREES
 ↓delete
 TRES
 ↓replace
 ORES
 ↓insert
 FORES
 ↓insert
 FOREST

[ToC](#)

5.10 Edit Distance Diagram Construction

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

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

`edDistTblOnNo`

	ε	O	N
ε	0 → 1 → 2		
N	1 ↓ 1 ↘ 1		
O	2 ↓ 1 ↘ 2		

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

`edDistTblOhNo`

	ε	O	H
ε	0 → 1 → 2		
N	1 ↓ 1 ↘ 2		
O	2 ↓ 1 ↘ 2		

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

edDistTblInOut

	ϵ	I	N
ϵ	0 → 1 → 2 ↓ ↘ ↘		
O	1 1 → 2 ↓ ↘ ↘		
U	2 2 2 ↓ ↘ ↘		
T	3 3 3 ↓ ↘ ↘		

LaTeX Code for **editDistTblDP** "OH" "NO"

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

```

2 [ePath/.style={red,thick}
3 ,frSub/.style={font=\bfseries}
4 ,efrSb/.style={ePath,frSub}
5 ,orArr/.style={-Latex}
6 ,frArr/.style={orArr,ultra thick}
7 ,eoArr/.style={ePath,orArr}
8 ,efArr/.style={ePath,frArr}
9 ]

```

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

```

11 % TikZ Matrix code
12 \matrix (edMat) [matrix of nodes
13 ,nodes in empty cells
14 ,ampersand replacement=\&
15 ,nodes={minimum size=\STextNodeWidth}]
16 {
17 \& \& \& \& \& \& \& \& \&
18 \& \& \& $\epsilon$ \& \& 0 \& \& H \& \&
19 \& \& \& \& \& \& \& \& \&
20 \& $\epsilon$ \& \& 0 \& \& 1 \& \& 2 \& \&
21 \& \& \& \& \& \& \& \& \&
22 \& N \& \& 1 \& \& 1 \& \& 2 \& \&
23 \& \& \& \& \& \& \& \& \&
24 \& 0 \& \& 2 \& \& 1 \& \& 2 \& \&
25 \& \& \& \& \& \& \& \& \&
26 };

```

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

```

29 % TikZ Arrows code
30 \draw[orArr] (edMat-4-4) -- (edMat-4-6);
31 \draw[orArr] (edMat-4-6) -- (edMat-4-8);

```

```

33 \draw[orArr] (edMat-4-4) -- (edMat-6-4);
34 \draw[orArr] (edMat-4-4) -- (edMat-6-6);
35 \draw[orArr] (edMat-6-6) -- (edMat-6-8);
36 \draw[orArr] (edMat-4-6) -- (edMat-6-8);

38 \draw[orArr] (edMat-6-4) -- (edMat-8-4);
39 \draw[orArr] (edMat-6-4) -- (edMat-8-6);
40 \draw[orArr] (edMat-8-6) -- (edMat-8-8);
41 \draw[orArr] (edMat-6-6) -- (edMat-8-8);

```

- LaTeX TikZ Frame code

```

45 % TikZ frame code
46 \draw (edMat-1-1.center) -- (edMat-1-9.center)
47      -- (edMat-9-9.center) -- (edMat-9-1.center)
48      -- cycle;
49 \draw (edMat-3-1.center) -- (edMat-3-9.center);
50 \draw (edMat-1-3.center) -- (edMat-9-3.center);

```

Generating PGF/TikZ from the Edit Table

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

Offsets, Scales and Constants

```

106 iOffset = 4 -- offset for original string
107 jOffset = 4 -- offset for target string
108 iScale  = 2 -- scale for original string
109 jScale  = 2 -- scale for target string

111 spc = ' ' -- space char
112 nl  = '\n' -- new line char

```

TikZ Arrow Code (1)

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

```

114 edCellTrPredToArrow (styleName,matName)
115 ((iOff,iSc1),(jOff,jSc1)) (j,i) (a,b)
116 = "\draw" ++ "[" ++ styleName ++ "]" ++ [spc]
117 ++ "(" ++ matName ++ "-"
118 ++ show (jOff + jSc1*b)
119 ++ "-" ++ show (iOff + iSc1*a) ++ ")"
120 ++ "\draw" ++ matName ++ "-"
121 ++ "(" ++ matName ++ "-"

```

```

122 ++ show (j0ff + jSc1*j)
123 ++ "-" ++ show (i0ff + iSc1*i) ++ "; "
124 ++ "\n"

```

TikZ Arrow Code (2)

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

```

126 edCellTrPredToArrow01
127 = edCellTrPredToArrow
128   ("orArr", "edMat")
129   ((i0ffset, iScale), (j0ffset, jScale))

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

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

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

142 edTblTrToArrows01 edTblTr
143 = concat [edRowTrToArrows01 j cs
144           | (j,cs) <- zip [0..length edTblTr - 1]
145                 edTblTr]

```

```

147 -- Test Edit Table to Arrows
148 edTblTrToArrows01TF
149 = putStr (edTblTrToArrows01 edTblTrTF)

151 edTblTrToArrows01AA
152 = putStr (edTblTrToArrows01 edTblTrAA)

154 edTblTrToArrows01FM
155 = putStr (edTblTrToArrows01 edTblTrFM)

```

TikZ Matrix Code (1)

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

```

157 ndWidth = 13 -- string width of a node
158 -- nWidth = (length zStr + 1) + 2
159 -- spc = ' ' -- spc is defined above

161 ampRepStr = "\\&"

163 zStr = "$\\epsilon$"

165 matrixPreamble nm ampRepStr

```

```

166 = "\\matrix_" ++ nm ++ ")" ++ "[matrix_of_nodes"
167 ++ "\n" ++ nSpcs
168 ++ ",nodes_in_empty_cells"
169 ++ "\n" ++ nSpcs
170 ++ ",ampersand_replacement=" ++ ampRepStr
171 ++ "\n" ++ nSpcs
172 ++ ",nodes={minimum_size=\\STtextNodeWidth}]"
173 ++ "\n"
174 where
175 nSpcs = replicate (11 + length nm) spc

```

```

177 -- PGF/TikZ Matrix functions

179 edCellTrDistToNode :: Int -> String
180 edCellTrDistToNode d
181 = replicate n spc ++ dStr ++ [spc]
182 where
183 dStr = show d
184 n = ndWidth - 1 - length dStr

186 strToNode str
187 = replicate n spc ++ str ++ [spc]
188 where
189 n = ndWidth - 1 - length str

```

TikZ Matrix Code (2)

```

191 strToPrefixRow str
192 = "" ++ ampRepStr
193 ++ replicate ndWidth spc ++ ampRepStr
194 ++ "" ++ ampRepStr
195 ++ replicate ndWidth spc ++ ampRepStr
196 ++ concat [" " ++ ampRepStr
197 ++ replicate ndWidth spc ++ ampRepStr
198 | c <- str]
199 ++ " \\\\n"
200 ++ "" ++ ampRepStr
201 ++ replicate ndWidth spc ++ ampRepStr
202 ++ "" ++ ampRepStr
203 ++ replicate (ndWidth - 11) spc
204 ++ "$\\epsilon$" ++ ampRepStr
205 ++ concat [" " ++ ampRepStr
206 ++ replicate (ndWidth - 2) spc
207 ++ [c] ++ [spc] ++ ampRepStr
208 | c <- str]
209 ++ " \\\\n"

```

```

211 strToSuffixRow str
212 = "" ++ ampRepStr
213 ++ replicate ndWidth spc ++ ampRepStr
214 ++ "" ++ ampRepStr
215 ++ replicate ndWidth spc ++ ampRepStr
216 ++ concat [" " ++ ampRepStr
217 ++ replicate ndWidth spc ++ ampRepStr
218 | c <- str]
219 ++ " \\\\n"

221 colNoTrToPrefixCol zStr str j
222 = "" ++ ampRepStr ++
223 (if j == 0
224 then replicate (ndWidth - 1 - length zStr) spc
225 ++ zStr ++ [spc]
226 else replicate (ndWidth - 2) spc
227 ++ [str!!(j - 1)] ++ [spc])
228 ++ ampRepStr

```

```

230 edCellTrToNodes (d,ps)
231 = "" ++ ampRepStr
232 ++ edCellTrDistToNode d ++ ampRepStr

234 edRowTrToNodes zStr str j cs
235 = "" ++ ampRepStr

```

```

236 ++ replicate ndWidth spc ++ ampRepStr
237 ++ concat ["_"] ++ ampRepStr
238 ++ replicate ndWidth spc ++ ampRepStr
239 | c <- cs]
240 ++ "_\\\\\\n"
241 ++ colNoTrToPrefixCol zStr str j
242 ++ concat [edCellTrToNodes c | c <- cs]
243 ++ "_\\\\\\n"
244
245 edTblTrToNodes zStr str edTblTr
246 = concat [edRowTrToNodes zStr str j cs
247 | (j,cs) <- zip [0..length edTblTr - 1]
248 edTblTr]
249
250 -- Test usage
251 edTblTrToNodesTF
252 = putStr (edTblTrToNodes zStr "FOREST" edTblTrTF)

```

TikZ Matrix Code (3)

```

254 edTblTrToMatrix nm zStr s t
255 = matrixPreamble nm ampRepStr
256 ++ "{\n"
257 ++ strToPrefixRow s
258 ++ edTblTrToNodes zStr t edTblTr
259 ++ strToSuffixRow s
260 ++ "};\n"
261 where
262 edTblTr = transpose (editDistTblDP s t)
263
264 -- Test code for entire matrix
265 edTblTrToMatrixTF
266 = putStr (edTblTrToMatrix "edMat" zStr "TREES" "FOREST")
267
268 edTblTrToMatrixAA
269 = putStr (edTblTrToMatrix "edMat" zStr "ALGORITHM" "ALTRUISTIC")
270
271 edTblTrToMatrixFM
272 = putStr (edTblTrToMatrix "edMat" zStr "FOOD" "MONEY")

```

TikZ Picture Code (1)

```

274 -- Construction of tikzpicture
275
276 tikzPreamble
277 = "\\begin{tikzpicture}" ++ [n1]
278 ++ "[ePath/.style={red,thick}" ++ [n1]
279 ++ "[frSub/.style={font=\\bfseries}" ++ [n1]
280 ++ "[efrSb/.style={ePath,frSub}" ++ [n1]
281 ++ "[orArr/.style={-Latex}" ++ [n1]
282 ++ "[frArr/.style={orArr,ultra_thick}" ++ [n1]
283 ++ "[eoArr/.style={ePath,orArr}" ++ [n1]
284 ++ "[efArr/.style={ePath,frArr}" ++ [n1]
285 ++ "]"
286
287 tikzEnd
288 = "\\end{tikzpicture}" ++ [n1]

```

TikZ Picture Code (2)

```

290 tikzFrmCoord nm loc i j
291 = "(" ++ nm ++ "-" ++ show j ++ "-"
292 ++ show i ++ "." ++ loc ++ ")"
293
294 tikzFrame nm loc s t
295 = "\\draw_" ++ (tikzFrmCoord nm loc 1 1) ++ "___"
296 ++ (tikzFrmCoord nm loc xMax 1)
297 ++ [n1] ++ dSpCs ++ "___"
298 ++ (tikzFrmCoord nm loc xMax yMax) ++ "___"

```

```

299     ++ (tikzFrmCoord nm loc 1 yMax)
300     ++ [nl] ++ dSpcs ++ "\_--\_cycle;" ++ [nl]
301     ++ "\\draw_" ++ (tikzFrmCoord nm loc 1 (jScale + 1))
302     ++ "\_--_" ++ (tikzFrmCoord nm loc xMax (jScale + 1))
303     ++ ";" ++ [nl]
304     ++ "\\draw_" ++ (tikzFrmCoord nm loc (iScale + 1) 1)
305     ++ "\_--_" ++ (tikzFrmCoord nm loc (iScale + 1) yMax)
306     ++ ";" ++ [nl]
307   where
308     xMax = iOffset + 1 + iScale * (length s)
309     yMax = jOffset + 1 + jScale * (length t)
310     dSpcs = replicate 6 spc

```

TikZ Picture Code (3)

```

312   edTblTrToTikz nm loc zStr s t
313   = tikzPreamble ++ [nl]
314     ++ [nl] ++ "%_TikZ_Matrix_code" ++ [nl]
315     ++ edTblTrToMatrix nm zStr s t ++ [nl]
316     ++ [nl] ++ "%_TikZ_Arrows_code" ++ [nl]
317     ++ edTblTrToArrows01 edTblTr ++ [nl]
318     ++ [nl] ++ "%_TikZ_frame_code" ++ [nl]
319     ++ tikzFrame nm loc s t ++ [nl]
320     ++ tikzEnd
321   where
322     edTblTr = transpose (editDistTblDP s t)

```

Test Code Matrix Frame

```

324   -- Test code for matrix frame lines
325
326   tikzFrameFM
327   = putStr (tikzFrame "edMat" "center"
328     "FOOD" "MONEY")
329
330   tikzFrameAA
331   = putStr (tikzFrame "edMat" "center"
332     "ALGORITHM" "ALTRUISTIC")
333
334   tikzFrameTF
335   = putStr (tikzFrame "edMat" "center"
336     "TREES" "FOREST")

```

Test Code TikZ Picture

```

338   -- Text code for tikzpicture
339
340   edTblTrToTikzFM
341   = putStr (edTblTrToTikz "edMat" "center" zStr
342     "FOOD" "MONEY")
343
344   edTblTrToTikzAA
345   = putStr (edTblTrToTikz "edMat" "center" zStr
346     "ALGORITHM" "ALTRUISTIC")
347
348   edTblTrToTikzTF
349   = putStr (edTblTrToTikz "edMat" "center" zStr
350     "TREES" "FOREST")
351
352   edTblTrToTikzKK
353   = putStr (edTblTrToTikz "edMat" "center" zStr
354     "KITTEN" "KNITTING")
355
356   edTblTrToTikzSE
357   = putStr (edTblTrToTikz "edMat" "center" zStr
358     "SIX" "ELEVEN")

```

edDistTblKittenKnitting

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

edDistTblSixEleven

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

Commentary 4

4 Future Work, Other Examples, References

- Future work — Tutorials and TMAs
- Other examples
- References and other sources
- **Colophon**
- LaTeX with Beamer, Listings and other packages
- Index of Python code and diagrams
- PGF/TikZ for the diagrams
- External copies of the diagrams as PDF with tight bounding boxes are available

[ToC](#)

6 Future Work

- Future tutorial and TMA dates
- **Other DP examples**
- Longest common subsequence
- Knapsack
- TSP

[ToC](#)

7 Web Sites & References

7.1 Edit Distance Sources

- [Jeff Erickson Algorithms](#) — basis of these notes
- [Wikibooks Levenshtein](#) (12 February 2016)
- [Wikipedia: Levenshtein](#) (12 February 2016)
- Linux Magazine March 2016 Issue 184 [Better Finds](#) — article on egrep which uses the Levenshtein algorithm
- [Rosetta Code: Levenshtein Distance](#)
- Peter Norvig [How to Write a Spelling Corrector](#) (9 February 2016)
- [Stack Overflow: Edit Distance in Haskell](#) (9 February 2016) — main source of Haskell code in this section
- *Levenshtein Algorithm* www.levenshtein.net (10 August 2023)
- *Levenshtein Demo* let.rug.nl/~kleiweg/lev/ (10 August 2023)
- [Edit distance \(Levenshtein-Distance\) algorithm explanation](#) (10 August 2023)

- [Wikipedia: Damerau-Levenshtein distance](#) (12 February 2016) allows for transposition of two adjacent characters

[ToC](#)

7.2 Algorithm Texts & Web Sites

- [Jeff Erickson Algorithms](#)
- [Cormen et al. \(2022\) *Introduction to Algorithms*](#)
- [Sedgewick and Wayne \(2011\) *Algorithms*](#) see [Algorithms](#), 4th edition
- [Bird and Gibbons \(2020\) *Algorithm Design with Haskell*](#)

7.3 Analysis of Algorithms

- [Big O notation](#) (2 April 2022)
- [The Algebra of Big-O](#) (2 April 2022)
- [Python Time Complexity](#) — also summarised in [Software Summaries](#)
- [Computational Complexity of a List Comprehension](#) (10 August 2023) — as it says: *The structure of a list comprehension makes it easy to determine computational complexity.*

This is from [An Introduction to Functional Programming, Lazy Evaluation, and Streams in Python](#) (10 August 2023)

- [Big-O Cheat Sheet](#) (10 August 2023)

7.4 Graph Algorithms

- [Graph Theory: Trees](#)
- [Graph Theory](#)
- [Dekai Wu Algorithms course](#)
- <http://jeffe.cs.illinois.edu/teaching/algorithms/> Jeff Erickson Algorithms

[ToC](#)

7.5 Shortest Paths

- Dijkstra's Algorithm
 - [Dijkstra's shortest path algorithm](#)
- Bellman-Ford
 - [Bellman-Ford Algorithm](#)
 - [Bellman Ford Algorithm \(Simple Implementation\)](#)
 - [Haskell Hackage Data.BellmanFord](#)
 - [Data.IGraph](#)

from [igraph](#)

- [igraph Reference Manual Chapter 13 Structural Properties of Graphs](#)
- [R igraph manual pages: Shortest \(directed or undirected\) paths between vertices](#)
- [IGraph/M: a Mathematica interface for igraph](#)

[ToC](#)

7.6 Web Sites for Dynamic Programming

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

[ToC](#)

References

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

- Bird, Richard (2014). *Thinking Functionally with Haskell*. Cambridge University Press. ISBN 1107452643. URL <http://www.cs.ox.ac.uk/publications/books/functional/>.
- Bird, Richard and Jeremy Gibbons (2020). *Algorithm Design with Haskell*. Cambridge University Press. ISBN 9781108869041. URL <http://www.cs.ox.ac.uk/publications/books/adwh/>. 40
- Bird, Richard and Phil Wadler (1988). *Introduction to Functional Programming*. Prentice Hall, first edition. ISBN 0134841972.
- Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; and Clifford Stein (2009). *Introduction to Algorithms*. MIT Press, third edition. ISBN 0262533057. URL <http://mitpress.mit.edu/books/introduction-algorithms>. 41
- Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; and Clifford Stein (2022). *Introduction to Algorithms*. MIT Press, fourth edition. ISBN 9780262046305. URL <https://mitpress.mit.edu/books/introduction-algorithms-fourth-edition>. 40
- Dijkstra, Edsger W (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.
- Hudak, Paul; John Hughes; Simon Peyton Jones; and Phil Wadler (2007). A History of Haskell: Being Lazy with Class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 12–1–12–55. ACM New York, NY, USA.
- Lee, Gias Kay (2013). Functional Programming in 5 Minutes. Web. <http://gsklee.im>, URL <http://slid.es/gsklee/functional-programming-in-5-minutes>.
- Lutz, Mark (2011). *Programming Python*. O'Reilly, fourth edition. ISBN 0596158106. URL <http://learning-python.com/books/about-pp4e.html>.
- Lutz, Mark (2013). *Learning Python*. O'Reilly, fifth edition. ISBN 1449355730. URL <http://learning-python.com/books/about-1p5e.html>.
- Marlow, Simon and Simon Peyton Jones (2010). Haskell Language and Library Specification. Web. URL http://www.haskell.org/haskellwiki/Language_and_library_specification.
- Miller, Bradley W. and David L. Ranum (2011). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle Associates Inc, second edition. ISBN 1590282574. URL <http://interactivepython.org/courselib/static/pythonds/index.html>.
- O'Donnell, John; Cordelia Hall; and Rex Page (2006). *Discrete Mathematics Using a Computer*. Springer, second edition. ISBN 1846282411. URL <http://www.dcs.gla.ac.uk/~jtod/discrete-mathematics/>.
- Okasaki, Chris (1998). *Purely Functional Data Structures*. Cambridge University Press. ISBN 0-521-63124-6.
- O'Sullivan, Bryan; John Goerzen; and Donald Stewart (2008). *Real World Haskell*. O'Reilly, first edition. ISBN 0596514980. URL <http://book.realworldhaskell.org/>.
- Rabhi, Fethi and Guy Lapalme (1999). *Algorithms: A Functional Programming Approach*. Addison-Wesley, second edition. ISBN 0201596040. URL <http://www.iro.umontreal.ca/~lapalme/Algorithms-functional.html>.
- Sedgewick, Robert and Kevin Wayne (2011). *Algorithms*. Addison Wesley, fourth edition. ISBN 032157351X. URL <http://algs4.cs.princeton.edu/home/>. 40

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

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

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

[ToC](#)

Python Code Index

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

delC, [28](#)

edCellDist, [28](#)
edCellPreds, [28](#)
editDistTblDP, [29](#)

fairListing, [12](#), [14](#), [15](#)
fairListingA, [12](#), [15](#), [15](#)
fairLstATest, [12](#), [15](#)
fairLstTest, [12](#), [14](#), [15](#)
fib1indent, [17](#)
fibs1, [17](#)
fibs2, [17](#)
fibs2indent, [17](#)
filterStopWords, [13](#)
filterStopWordsTest, [13](#)

getEdCell, [28](#)
getEdDist, [28](#)

getEdPreds, [28](#)

insC, [28](#)

matAtr, [11](#)
matrixA, [11](#), [13](#)

sentence, [11](#), [11](#)
setEdTblCell, [28](#)
stopWords, [11](#), [11](#)
subC, [28](#)

transMat, [13](#)
transMatTestA, [13](#)

words, [11](#), [11](#)
wordsTest, [11](#), [11](#)

yBiasListing, [12](#), [14](#), [14](#)
yBiasLstTest, [12](#), [14](#)

[ToC](#)

Haskell Code Index

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

ampRepStr, [35](#)
colNoTrToPrefixCol, [35](#)
delC, [26](#)
Dist, [24](#)
edCellDist, [25](#)
edCellPreds, [25](#)
edCellTrDistToNode, [35](#)
edCellTrPredsToArrows01, [34](#)
edCellTrPredToArrow, [34](#)
edCellTrPredToArrow01, [34](#)
edCellTrToArrows01, [34](#)
edCellTrToNodes, [36](#)
EditDistCell, [24](#)
EditDistTable, [24](#)
editDistTblDP, [26](#)
editDistTblDP01, [26](#)
edRowTrToArrows01, [34](#)
edRowTrToNodes, [36](#)
edTblAA, [26](#)
edTblAA01, [27](#)
edTblFM, [26](#)
edTblFM01, [27](#)
edTblTF, [26](#)
edTblTF01, [27](#)
edTblTrAA, [26](#)
edTblTrAA01, [27](#)
edTblTrFM, [26](#)
edTblTrFM01, [27](#)
edTblTrTF, [26](#)
edTblTrTF01, [27](#)
edTblTrToArrows01, [34](#)
edTblTrToArrows01AA, [34](#)
edTblTrToArrows01FM, [34](#)
edTblTrToArrows01TF, [34](#)
edTblTrToMatrix, [36](#)
edTblTrToMatrixAA, [36](#)
edTblTrToMatrixFM, [36](#)
edTblTrToMatrixTF, [36](#)
edTblTrToNodes, [36](#)
edTblTrToNodesTF, [36](#)
edTblTrToTikz, [37](#)
edTblTrToTikzAA, [38](#)
edTblTrToTikzFM, [38](#)
edTblTrToTikzKK, [38](#)
edTblTrToTikzSE, [38](#)
edTblTrToTikzTF, [38](#)
getEdCell, [25](#)
getEdDist, [25](#)
getEdPreds, [25](#)
insC, [26](#)
iOffset, [33](#)
iScale, [33](#)
jOffset, [33](#)
jScale, [33](#)
matrixPreamble, [35](#)
ndWidth, [35](#)
n1, [33](#)
Pred, [24](#)
setEdTblCell, [27](#)
spc, [33](#)
strToNode, [35](#)
strToPrefixRow, [35](#)
strToSuffixRow, [35](#)
subC, [26](#)
tikzEnd, [36](#)
tikzFrame, [37](#)
tikzFrameAA, [37](#)
tikzFrameFM, [37](#)
tikzFrameTF, [37](#)
tikzFrmCoord, [37](#)
tikzPreamble, [36](#)
zStr, [35](#)

Diagrams Index

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

edDistTblAlgorithmAltruistic, [29](#)

edDistTblFoodMoney, [23](#)

edDistTblInOut, [32](#)

edDistTblKittenKnitting, [38](#)

edDistTblOhNo, [31](#)

edDistTblOnNo, [31](#)

edDistTblSixEleven, [38](#)

edDistTblTreesForest, [30](#)

[ToC](#)