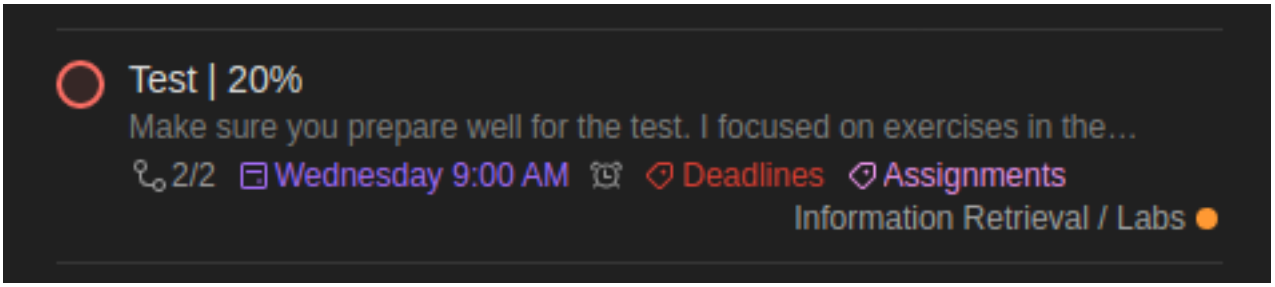
Information Retrieval

professor: Michel P Schellekens
email: m.schellekens@cs.ucc.ie
old lecture recordings: https://ucc.instructure.com/courses/48669/pages/old-panopto-recordings-for-this-course-organised-by-week?module_item_id=1513237

Examination and Assessment

Total Marks 100

- Written Examination 80 Marks - Semester 1 Written Exam
  - ◇ Paper 1: 1.5hr paper - Written Questions (80 Marks)
- Continuous Assessment 20 Marks
  - ◇ In-class Test - in-class test (**20 Marks**)



- Wednesday **9am** in **[G09]**

- **Test Announcements**:
  - ◇ Make sure you prepare well for the test. I focused on exercises in the last lecture and the tutorial this week (w6).
    I uploaded the solutions under weeks 5 and 6 in the Module on Lecture notes.
    The material will involve Markov chains and page ranking which needs consistent focus to master
    as well as the scoring we have just covered.
  - ◇ Bring paper and pen to write the answers in class.
    Bring Calculator.
  - ◇ As to contents: everything we covered up to and including lecture 9 (page ranking) needs to be known.
    The material that we have covered is marked in red under the lecture notes (we did *not* cover lectures 7 and 8,
    but we did cover lecture 9 which is included in the red-marked part).

TEST Notes

To Learn:
- **Levenshtein Distance**
- **Heap's Law**
  - ◇ log calculations
- **Postings Compression**
- **Matrices**
- **Random Walk**
- **Precision and Recall**

todo:

maybe???:
- Entropy and Huffman encoding: link ??
- Zipf's Law ??
  - ◇ tutorial: link and link

what this?:
- ko-sign similarity ??

Levenshtein Algorithm

- Levenshtein Distance **Algorithm** lecture:
  - ◇ part1 setup: https://web.microsoftstream.com/video/7ded103c-c240-4e06-bf7e-07570ccb51b5
  - ◇ part2 algorithm: https://web.microsoftstream.com/video/1bf23e60-6470-4fff-9e12-fd11e4d0629b
  - ◇ part3 example: https://web.microsoftstream.com/video/4eeb072c-a8d4-4ffc-9631-67f240ca592d

Levenshtein distance: Algorithm

```
LEVENSHTEINDISTANCE(s1, s2)
 1  for i ← 0 to |s1|
 2  do m[i, 0] = i
 3  for j ← 0 to |s2|
 4  do m[0, j] = j
 5  for i ← 1 to |s1|
 6  do for j ← 1 to |s2|
 7      do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Each cell of Levenshtein matrix

cost of getting here from my upper left neighbor (copy or replace)	cost of getting here from my upper neighbor (delete)
cost of getting here from my left neighbor (insert)	the minimum of the three possible "movements"; the cheapest way of getting here

```
 6  do for j ← 1 to |s2|
 7      do if s1[i] = s2[j]
 8            then m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]}
 9            else  m[i, j] = min{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1}
10  return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

```
do for j ← 1 to |s2|
    do if s1[i] = s2[j]
        then m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]}
        else  m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]+1}
return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

```
do for j ← 1 to |s2|
    do if s1[i] = s2[j]
        then m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]}
        else  m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]+1}
return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

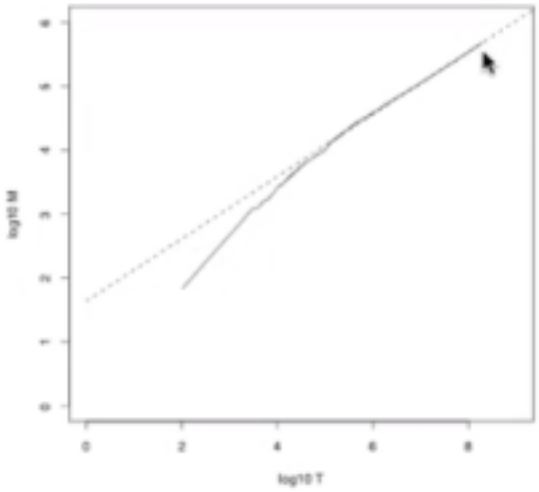
```
do for j ← 1 to |s2|
    do if s1[i] = s2[j]
        then m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]}
        else  m[i,j] = min{m[i-1,j]+1, m[i,j-1]+1, m[i-1,j-1]+1}
return m[|s1|, |s2|]
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Heap's Law

- Heap's Law
  - ◇ lectire: https://web.microsoftstream.com/video/2540efbc-696a-4ba6-977c-b4becda174a7
  - ◇ tutorial: https://web.microsoftstream.com/video/06b37f89-6bb2-4c40-8617-7b968943d0bf
  - ◇ review “log” calculations

Heaps’ law for Reuters



K = 44 and b = 0.49

$M = kT^b$

Vocabulary size M as a function of collection size T (number of tokens) for Reuters-RCV1The dashed line $\log_{10} M = 0.49 * \log_{10} T + 1.64$ is the best least squares fit. Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$.

Precision And Recall

wiki link: https://en.wikipedia.org/wiki/Precision_and_recall

note:
- has to be between 0 and 1
  - ie. between 0% and 100%

Precision [edit]

In the field of information retrieval, precision is the fraction of retrieved documents that are relevant to the query:

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

For example, for a text search on a set of documents, precision is the number of correct results divided by the number of all returned results.

Precision takes all retrieved documents into account, but it can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called *precision at n* or *P@n*.

Precision is used with recall, the percent of *all* relevant documents that is returned by the search. The two measures are sometimes used together in the F_1 Score (or f-measure) to provide a single measurement for a system.

Note that the meaning and usage of "precision" in the field of information retrieval differs from the definition of accuracy and precision within other branches of science and technology.

Recall [edit]

In information retrieval, recall is the fraction of the relevant documents that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

For example, for a text search on a set of documents, recall is the number of correct results divided by the number of results that should have been returned.

In binary classification, recall is called sensitivity. It can be viewed as the probability that a relevant document is retrieved by the query.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore, recall alone is not enough. One needs to measure the number of non-relevant documents also, for example by also computing the precision.

Postings Compression

- Encoding
  - ◇ lecture: https://web.microsoftstream.com/video/bf5cadff-fbe0-4188-a00c-3e5a4592bf43
    - → Variable Byte Code from 00:00 to 05:00
    - → Gamma Coding from 05:00 to end

- Variable Byte Code

Variable byte (VB) code

- Used by many commercial/research systems
- Dedicate 1 bit (high bit) to be a continuation bit *c*.
- If the gap *G* fits within 7 bits, binary-encode it in the 7 available bits and set *c* = 1.
- Else: encode lower-order 7 bits and then use one or more additional bytes to encode the higher order bits using the same algorithm.
- At the end set continuation bit of the last byte to 1 (*c* = 1) and of the other bytes to 0 (*c* = 0).

VB code examples

824 is 1100111000 in binary

10 bits do not fit in one byte. So take the last seven bits and fill a byte, which starts with bit 1: this bit indicates that there are no more bytes following the given byte: 10111000
Store the remaining bits, i.e. 110 in a new byte (padding it with 0000 to fill the seven bits, and add a continuation bit at the front: 0) 00000110
Now concatenate both: 00000110 10111000

docIDs	824		829	215406		
gaps			5	214577		
VB code	00000110	10111000	10000101	00001101	00001100	10110001

- Gamma Coding

Gamma codes for gap encoding

- You can get even more compression with another type of variable length encoding: bitlevel code.
- Gamma code is the best known of these.
- First, we need unary code to be able to introduce gamma code.
- Unary code
  - Represent *n* as *n* 1s with a final 0.
  - Unary code for 3 is 1110
  - Unary code for 40 is 11111111111111111111111111111111111111110
  - Unary code for 70 is:

11111111111111111111111111111111111111111111111111111111111111111111110

38

Gamma code

- Represent a gap G as a pair of length and offset.
- Offset is the gap in binary, with the leading bit chopped off.
- For example 13 → 1101 → 101 = offset
- Length is the length of offset.
- For 13 (offset 101), this is 3.
- Encode length in unary code: 1110.
- Gamma code of 13 is the concatenation of length and offset: 1110101.

Gamma code examples

number	unary code	length	offset	γ code
0	0			
1	10	0		0
2	110	10	0	10,0
3	1110	10	1	10,1
4	11110	110	00	110,00
9	1111111110	1110	001	1110,001
13		1110	101	1110,101
24		11110	1000	11110,1000
511		111111110	11111111	111111110,11111111
1025		11111111110	0000000001	11111111110,0000000001

Matrices

- Matrices
  - ◇ Lectures:
    - → Matrix Multiplication @15min onwards
      - ⇒ https://ucc.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=514d7b82-416c-4ba1-b414-ac4700c0a7db
    - → Multiplication demonstration @first 6 min
      - ⇒ https://ucc.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=1931c2c4-9b2a-4ea3-bdf4-ac4c009a6432
    - → Transpose of a Matrix
      - ⇒ https://ucc.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=b029caa0-467d-464b-a16d-ac4700c8ce76

- Matrix Multiplication example:

Example 10 *Consider the matrices A and B given below.*

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 3 & 2 & 1 \\ 1 & 2 & 3 \\ 1 & 0 & 1 \end{bmatrix}$$

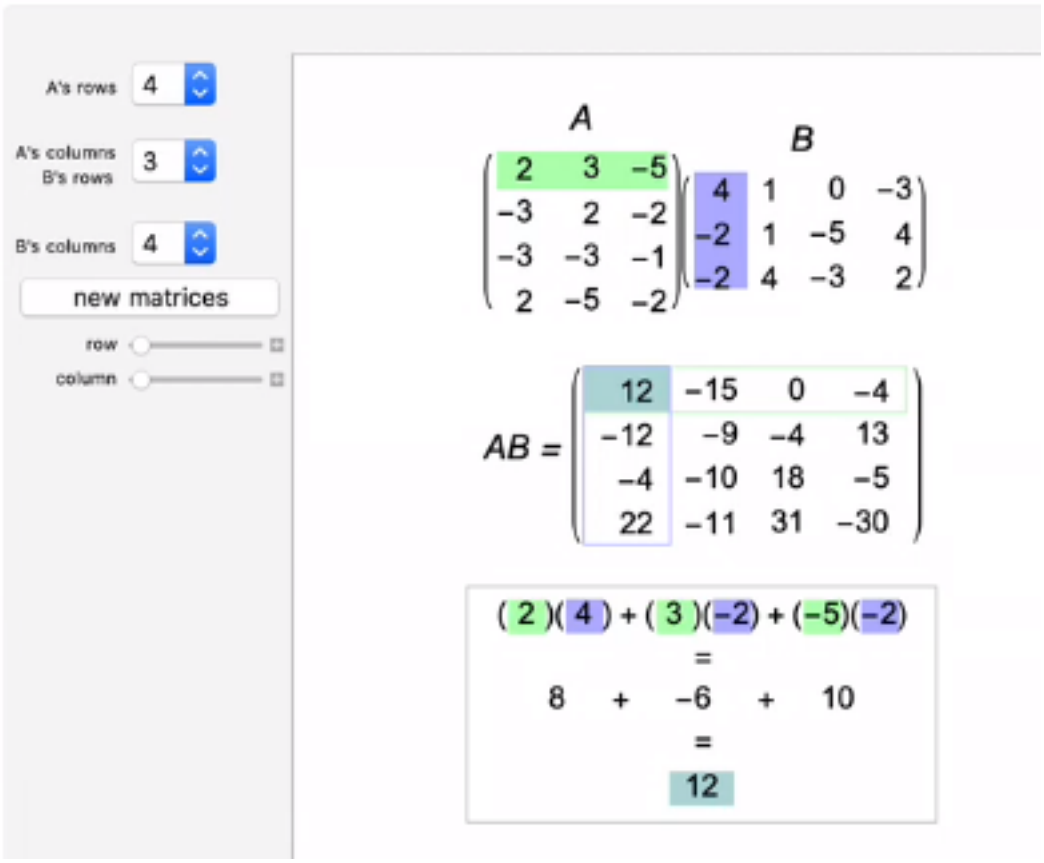
To compute $AB_{2,3}$, i.e. the entry in the product matrix AB in second row and third column, multiply the elements in row 2 of matrix A one-by-one with the elements in column 3 of matrix B:

$$3 \times 1, 2 \times 3, 1 \times 1$$

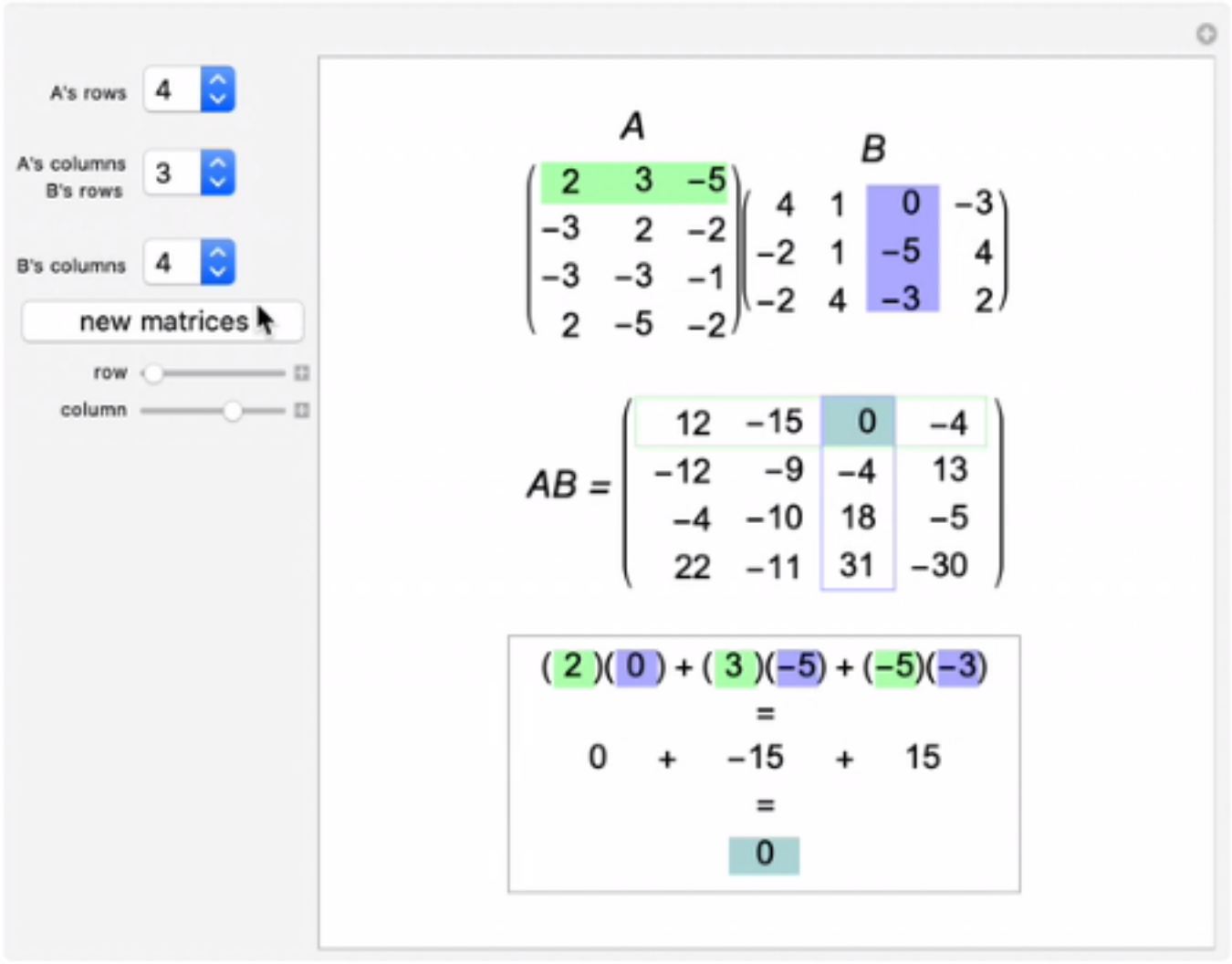
Adding up the outcomes yields the entry

$$AB_{2,3} = 3 \times 1 + 2 \times 3 + 1 \times 1 = 10$$

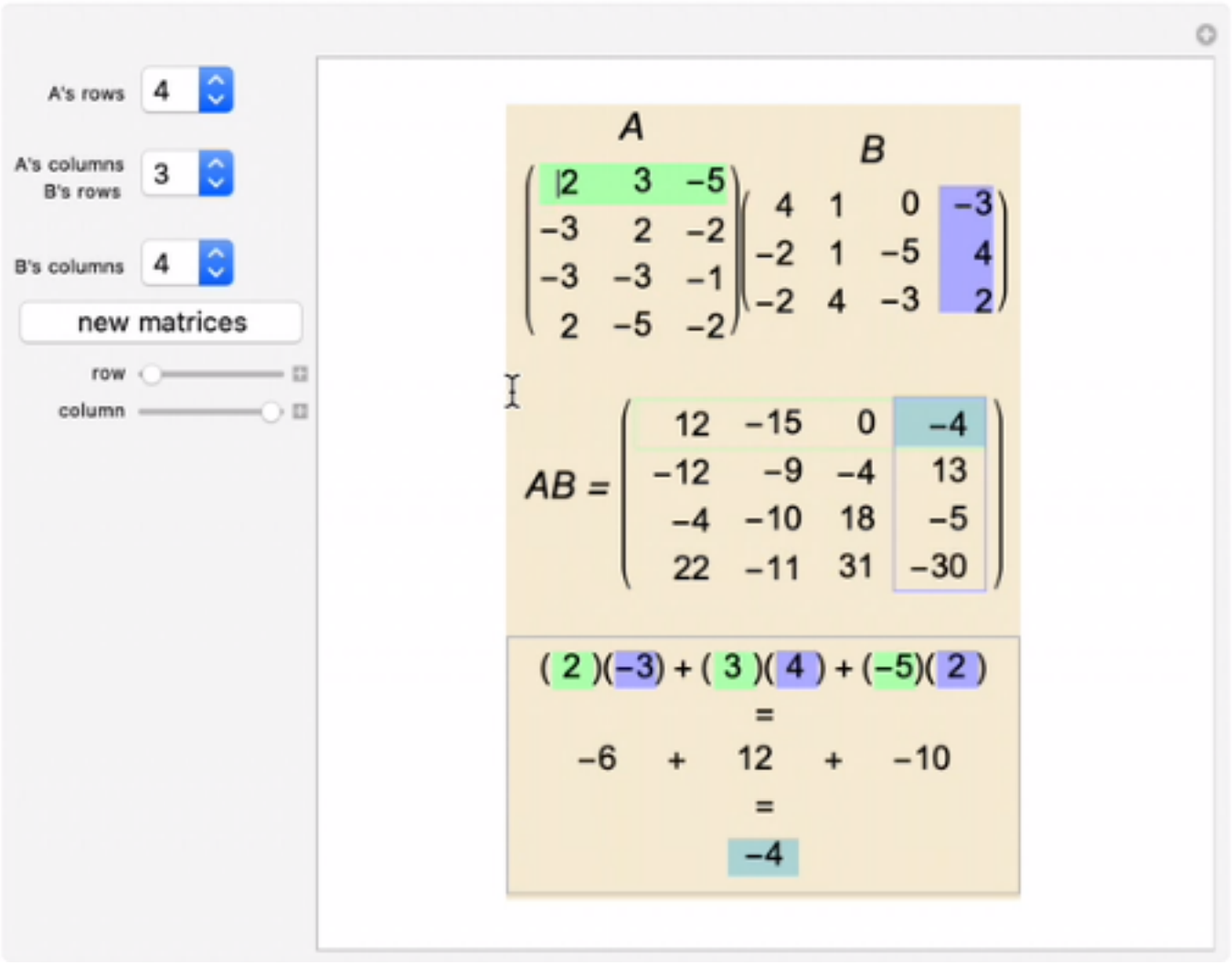
Matrix Multiplication



Matrix Multiplication



Matrix Multiplication



- Transpose

2.2.1 Transpose of a matrix

The *transpose* A^t of a matrix A is the result of interchanging i-the columns and i-th rows in A for each $i \in \{1, \ldots, n\}$. Formally: $A^t = (a_{ji})$. For instance

$$\text{if } A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ then } A^t = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

$$\text{if } B = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{6} & \frac{1}{2} & \frac{2}{6} \\ 0.12 & 0.53 & 0.35 \end{bmatrix} \text{ then } B^t = \begin{bmatrix} 0 & \frac{1}{6} & 0.12 \\ 1 & \frac{1}{2} & 0.53 \\ 0 & \frac{2}{6} & 0.35 \end{bmatrix}$$

The following properties hold. We omit the proofs. Verify these hold on some examples.

$$(AB)^t = B^t A^t$$
$$(A + B)^t = A^t + B^t$$

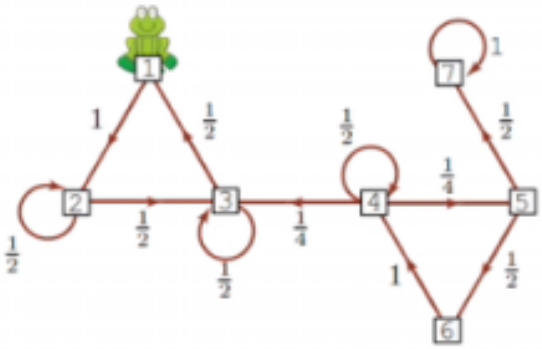
Random Walk | Markov Chains

- Random Walk
  ◇ example: https://ucc.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=745d09f5-3c25-48ce-ac79-ac4d00c04ff3
  ◇ calculation: https://ucc.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=ae0b5ef1-5fd8-4904-90bd-ac4d00e1d8b4
  ◇

Example 2 (Random walk)
Carry out a "random walk" through a city as follows: at each intersection involving, say, K multiple roads[5], pick one of these roads[6] to continue on with equal probability $\frac{1}{K}$[7]. The resulting walk forms a Markov chain. The current state, i.e. the last intersection arrived at, suffices to determine the next road you embark on. Prior intersections do not matter (memory-freeness).

The following illustrates a similar type of random walk, for which there are 7 'states' (lily pads)[8]

In matrix P the element p_{57} $(= \frac{1}{2})$ is the probability that, when starting in state 5, the next jump takes the frog to state 7.



$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

- calculation:

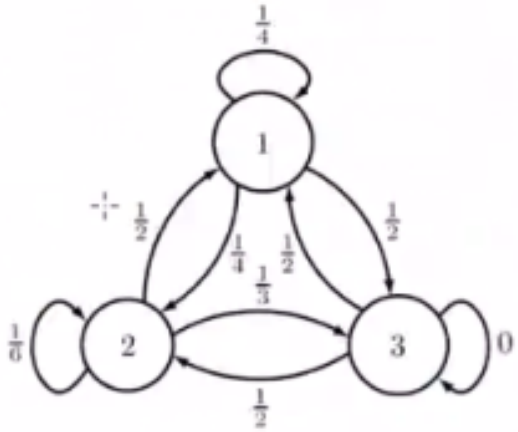


Figure 1: Random walk, induced by a frog jumping. The nodes in the graph represent the lily pads.

$$P = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{6} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

a) Check that this matrix is stochastic. Can you explain why this property holds?

We use the following notation, displaying the matrix P by row-vectors

$$\mathbf{v}_1 = \begin{bmatrix} p_{11} & p_{12} & p_{13} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} p_{21} & p_{22} & p_{23} \end{bmatrix}, \mathbf{v}_3 = \begin{bmatrix} p_{31} & p_{32} & p_{33} \end{bmatrix}$$

and by column vectors

$$\mathbf{w}_1 = \begin{bmatrix} p_{11} \\ p_{21} \\ p_{31} \end{bmatrix}, \mathbf{w}_2 = \begin{bmatrix} p_{12} \\ p_{22} \\ p_{32} \end{bmatrix}, \mathbf{w}_3 = \begin{bmatrix} p_{13} \\ p_{23} \\ p_{33} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} p_{11} & p_{12} & p_{13} \end{bmatrix} \\ \begin{bmatrix} p_{21} & p_{22} & p_{23} \end{bmatrix} \\ \begin{bmatrix} p_{31} & p_{32} & p_{33} \end{bmatrix} \end{bmatrix} = P = \begin{bmatrix} \begin{bmatrix} p_{11} \\ p_{21} \\ p_{31} \end{bmatrix} & \begin{bmatrix} p_{12} \\ p_{22} \\ p_{32} \end{bmatrix} & \begin{bmatrix} p_{13} \\ p_{23} \\ p_{33} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 \end{bmatrix}$$

b) The probabilities are:

$$p_{21}p_{12} = \frac{1}{2}\frac{1}{4} = \frac{1}{8}$$

$$p_{21}p_{13}p_{32} = \frac{1}{2}\frac{1}{2}\frac{1}{2} = \frac{1}{8}$$

$$p_{22}p_{23}p_{33} = \frac{1}{6}\frac{1}{3}0 = 0$$

c) Starting from pad 1, the frog can stay on pad 1 or jump to pad 2 or jump to pad 3. From each of those pads, it can once again transfer to any of the other three pads (by staying on the pad or by jumping to one of the other two pads). The probabilities for the frog to transfer to each of the other pads in 2 steps, starting from pad 1, hence are:

from pad 1 back to pad 1 in 2 steps:

$$p_{11}p_{11} + p_{12}p_{21} + p_{13}p_{31}$$

which in matrix multiplication becomes:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{21} \\ p_{31} \end{bmatrix} = \mathbf{v}_1\mathbf{w}_1$$

from pad 1 to pad 2 in 2 steps:

$$p_{11}p_{12} + p_{12}p_{22} + p_{13}p_{32}$$

which in matrix multiplication becomes:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \end{bmatrix} \begin{bmatrix} p_{12} \\ p_{22} \\ p_{32} \end{bmatrix} = \mathbf{v}_1\mathbf{w}_2$$

from pad 1 to pad 3 in 2 steps:

$$p_{11}p_{13} + p_{12}p_{23} + p_{13}p_{33}$$

which in matrix multiplication becomes:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \end{bmatrix} \begin{bmatrix} p_{13} \\ p_{23} \\ p_{33} \end{bmatrix} = \mathbf{v}_1\mathbf{w}_3$$

Hence we can summarise the probabilities of reaching any pad, starting from pad 1, as the result of the matrix multiplication:

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \end{bmatrix} P = \mathbf{v}_1 P = \begin{bmatrix} \mathbf{v}_1\mathbf{w}_1 & \mathbf{v}_1\mathbf{w}_2 & \mathbf{v}_1\mathbf{w}_3 \end{bmatrix}$$

We can repeat the same argument where the frog starts on pad 2 instead and, again, for the frog starting in pad 3.

For pad 2, we obtain:

$$\begin{bmatrix} p_{21} & p_{22} & p_{23} \end{bmatrix} P = \mathbf{v}_2 P = \begin{bmatrix} \mathbf{v}_2\mathbf{w}_1 & \mathbf{v}_2\mathbf{w}_2 & \mathbf{v}_2\mathbf{w}_3 \end{bmatrix}$$

And for pad 3:

$$\begin{bmatrix} p_{31} & p_{32} & p_{33} \end{bmatrix} P = \mathbf{v}_3 P = \begin{bmatrix} \mathbf{v}_3\mathbf{w}_1 & \mathbf{v}_3\mathbf{w}_2 & \mathbf{v}_3\mathbf{w}_3 \end{bmatrix}$$

In summary (see part A of the notes, :

$$\begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix} P = \begin{bmatrix} \mathbf{v}^1\mathbf{w}^1 & \dots & \mathbf{v}^1\mathbf{w}^n \\ \vdots & & \vdots \\ \mathbf{v}^n\mathbf{w}^1 & \dots & \mathbf{v}^n\mathbf{w}^n \end{bmatrix} = P^2$$

Each entry P^2_{ij} represents the probability of reaching pad j from pad i in 2 steps.

Similarly

Each entrty P^k_{ij} represents the the probability of reaching pad j from pad i in k steps.

Hence, if all of these values P^k converges to a fixed value as k increases (i.e. in the limit as $k \to \infty$), then P^∞_{ij} represents the probability of visiting pad j when starting from pad i during a continuously running random walk. It is this number that we will show exists, i.e. we will show that this limits always exists for a random walk, and it will represent the "ease of access" of each lily pad by the frog[10].