Übungen zu Algorithmen und Programmentwicklung für die Biologische Chemie

Michael T. Wolfinger

(based on slides by Sebastian Will)

Research Group Bioinformatics and Computational Biology
Department for Theoretical Chemistry
Währingerstrasse 17, 1090 Vienna, Austria

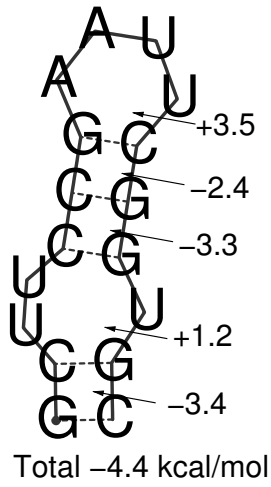
Sommersemester 2021

Random Sequences

Motivation Random Sequences

Random sequences allow calculating statistical measures for the 'significance' of an observation (compared to the random background).

RNA structure prediction



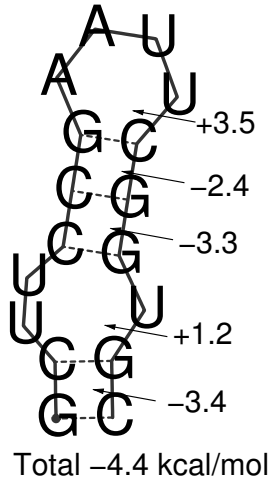
Energy of a structure = sum of loop energies

di-nucleotides are important

Motivation Random Sequences

Random sequences allow calculating statistical measures for the 'significance' of an observation (compared to the random background).

RNA structure prediction



Energy of a structure = sum of loop energies

di-nucleotides are important

$$z = \frac{E - \mu}{\sigma}$$

The *z score* normalizes the observed RNA structure stability (energy E) by mean μ and standard deviation σ.

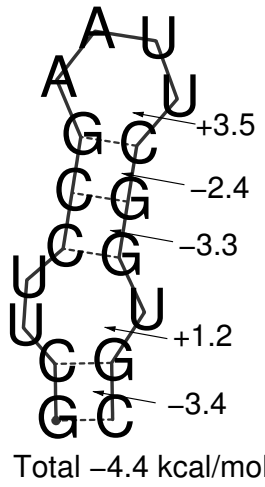
Estimate μ and σ from random sequences with

- nucleotide frequencies, or
- di-nucleotide content.

Motivation Random Sequences

Random sequences allow calculating statistical measures for the 'significance' of an observation (compared to the random background).

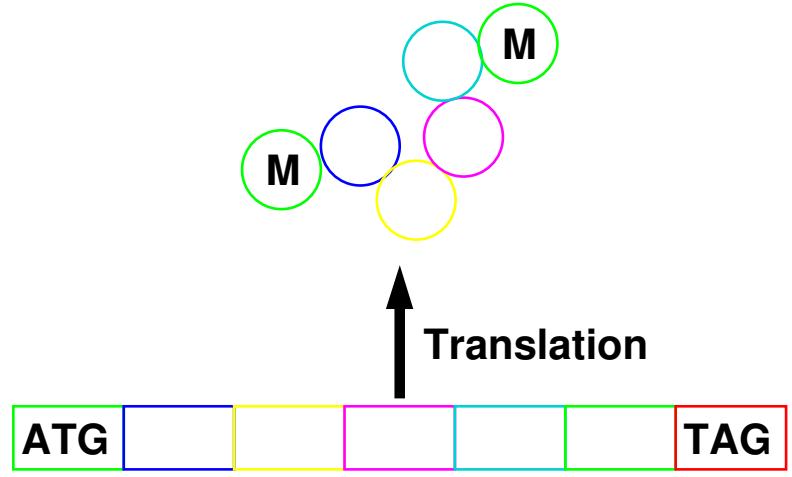
RNA structure prediction



Energy of a structure = sum of loop energies

di-nucleotides are important

Protein coding genes



RNA template is translated into amino acids.

tri-nucleotides are important

1 Frequency Estimation

Given a sequence the frequency of each nucleotide can be calculated as the fraction of the nucleotide count normalized by the sequence length.

```
CUUUUGCUAG
     |
     V
A => 1/10 = 0.1     UGCACGCUGA     A =>  5/40 = 0.125
C => 2/10 = 0.2 -> ACCUUCAUCU -\ C => 17/40 = 0.425
G => 2/10 = 0.2     UCUCCCUCCC -/ G =>  6/40 = 0.15
U => 5/10 = 0.5     GUGCAUCUGC     U => 12/40 = 0.3
```

1 Frequency Estimation

Given a sequence the frequency of each nucleotide can be calculated as the fraction of the nucleotide count normalized by the sequence length.

```
CUUUUGCUAG
     |
     V
A => 1/10 = 0.1     UGCACGCUGA     A =>  5/40 = 0.125
C => 2/10 = 0.2 -> ACCUUCAUCU -\ C => 17/40 = 0.425
G => 2/10 = 0.2     UCUCCCUCCC -/ G =>  6/40 = 0.15
U => 5/10 = 0.5     GUGCAUCUGC     U => 12/40 = 0.3
```

The generated sequences have on average the same nucleotide frequency as the given sequence.

Correct Mononucleotide Shuffling?

Given a sequence of length n and n different symbols, there are $n!$ possible random sequences; each of them should be created equally often!

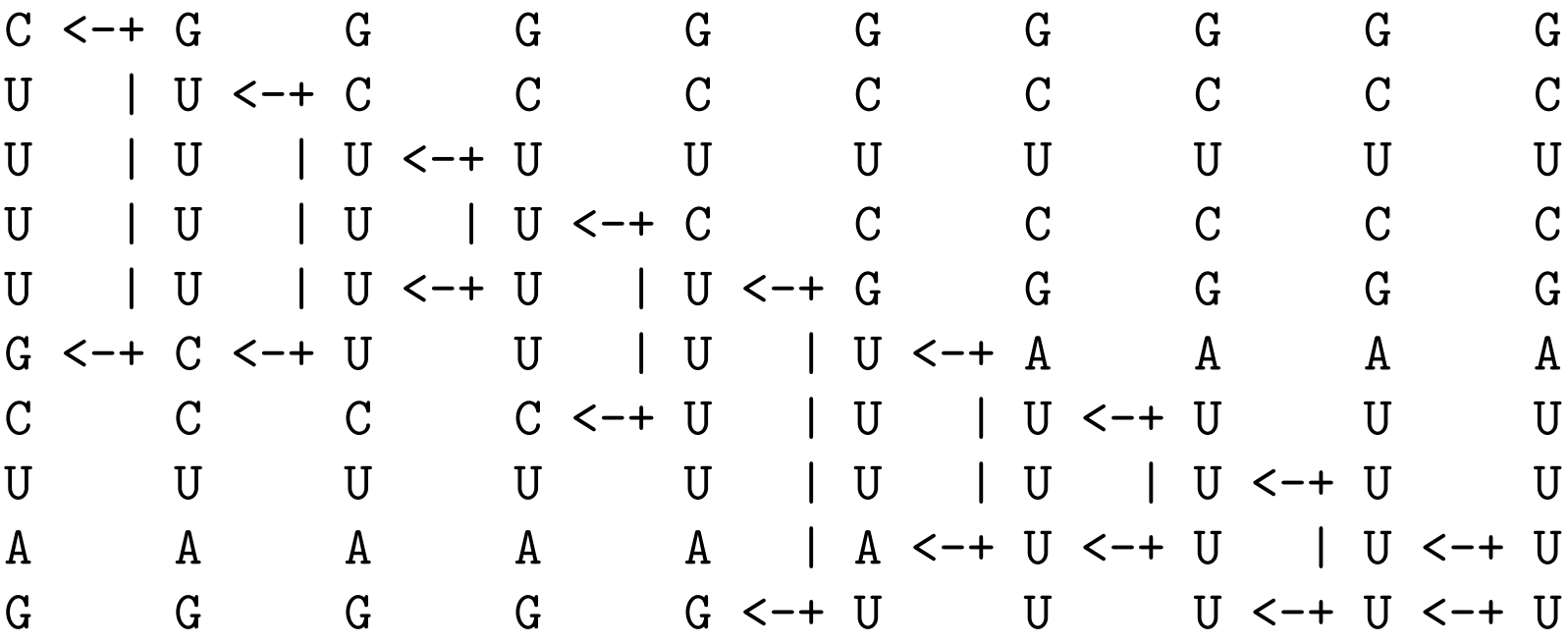
General strategy:

For $i = 0$ to $n - 1$: swap $S[i]$ and $S[j]$ for random j

- **Naive Approach:** select j randomly from the entire range. This creates a non-uniform distribution!
- **Fisher-Yates** select j between the current and the last position, $i \leq j \leq n$ (including the current one!)
- **Sattolo's Approach:** select j between the current and the last position, $i < j \leq n$. This cannot create all permutations! Only permutation cycles.

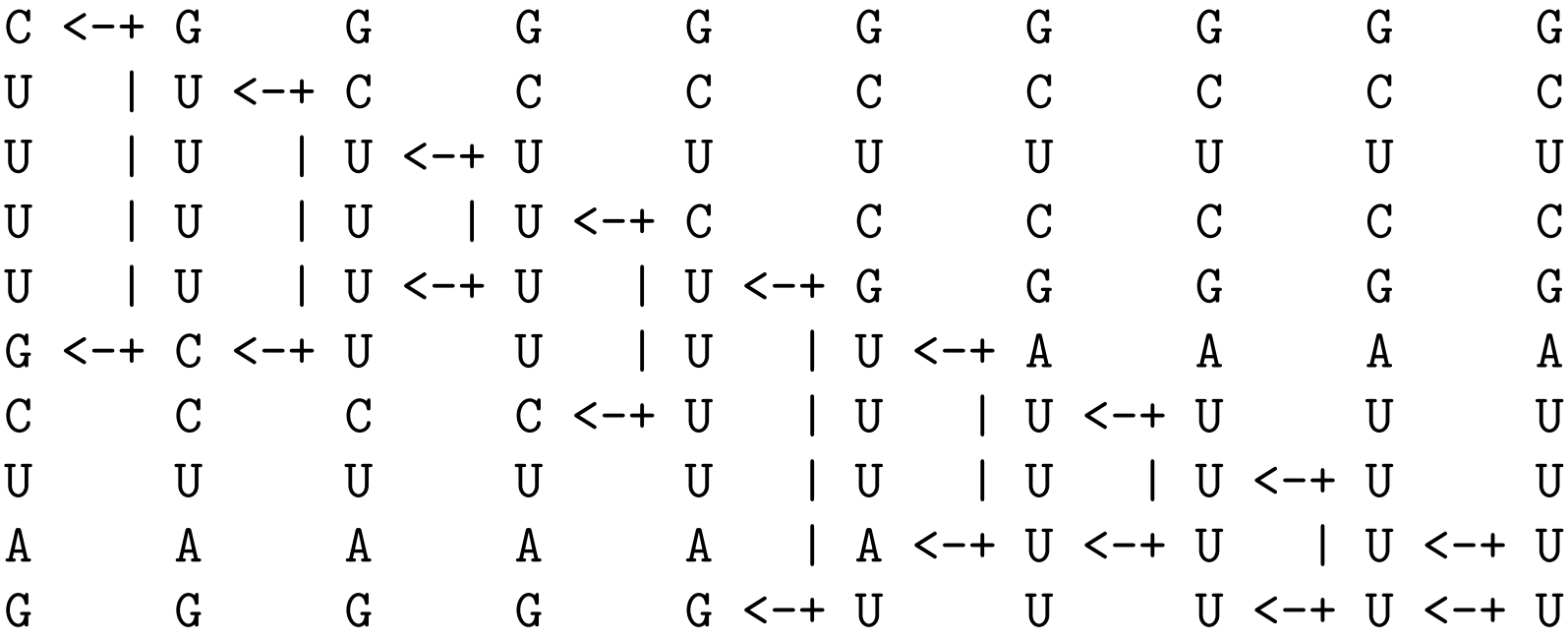
2 Knuth-Fisher-Yates Shuffling

The nucleotide of each position is kept or exchanged by one downstream chosen uniformly at random. This procedure is repeated $n - 1$ times, where n is the sequence length.



2 Knuth-Fisher-Yates Shuffling

The nucleotide of each position is kept or exchanged by one downstream chosen uniformly at random. This procedure is repeated $n - 1$ times, where n is the sequence length.



The generated sequences have exactly the same nucleotide frequency as the given sequence.

3 k-let Shuffling

k-let shuffling generates sequences with exactly the same frequency of sub-sequences of length k.

3 k-let Shuffling

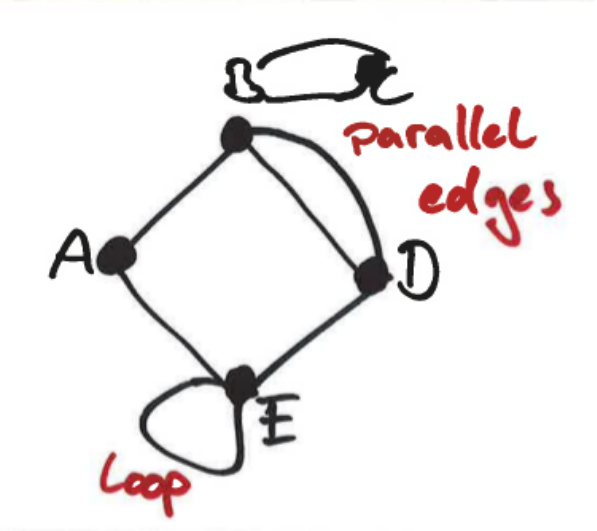
k-let shuffling generates sequences with exactly the same frequency of sub-sequences of length k.
We need a little graph theory to explain the method.

3 k-let Shuffling

k-let shuffling generates sequences with exactly the same frequency of sub-sequences of length k.
We need a little graph theory to explain the method.

Multi-graph

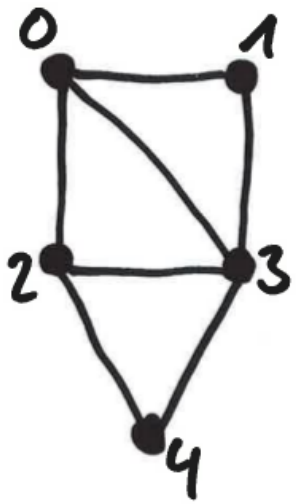
A graph G is defined as a set of vertices and edges that connect the vertices. G is a muti-graph if it contains loops and parallel edges.



Useful graph representation

Index vertices; assign **degrees**, i.e. number of edges associated with a vertex.

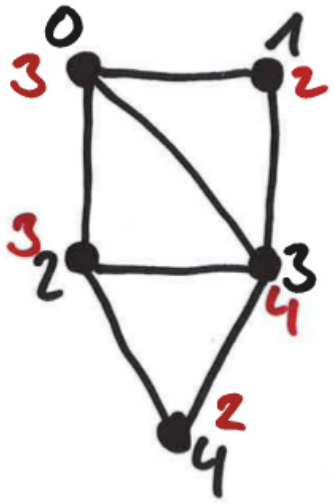
Graph representation:



Useful graph representation

Index vertices; assign **degrees**, i.e. number of edges associated with a vertex.

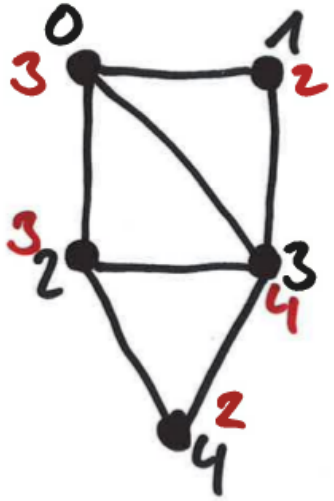
Graph representation:



Useful graph representation

Index vertices; assign **degrees**, i.e. number of edges associated with a vertex.

Graph representation:

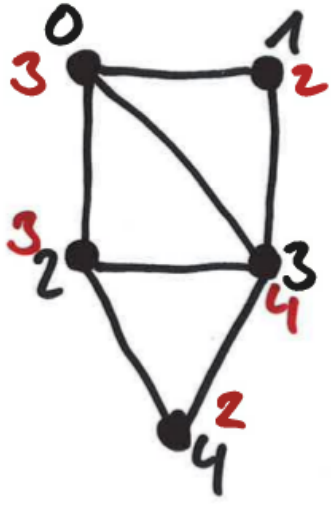


Adjacency matrix representation:

Useful graph representation

Index vertices; assign **degrees**, i.e. number of edges associated with a vertex.

Graph representation:



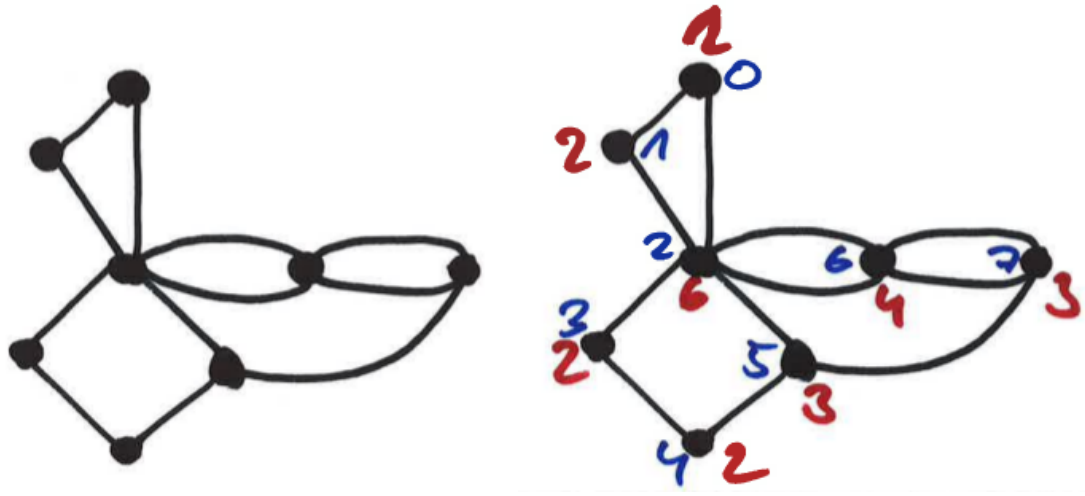
Adjacency matrix representation:

	0	**1**	**2**	**3**	**4**	**degree**
0	0	1	1	1	0	3
1	1	0	0	1	0	2
2	1	0	0	1	1	3
3	1	1	1	0	1	4
4	0	0	1	1	0	2

Eulerian Properties

Eulerian walk

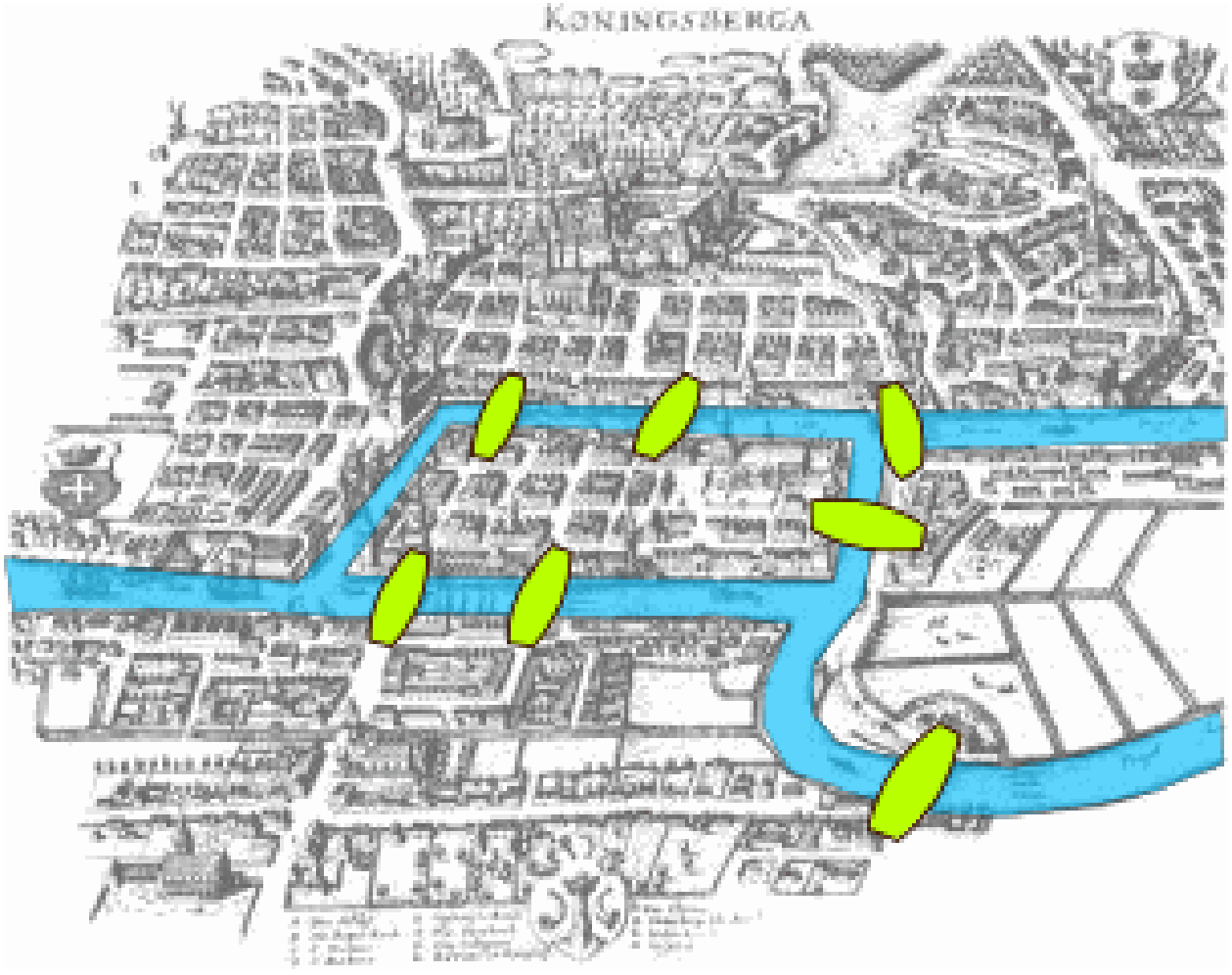
A walk in G is called Eulerian if it includes each edge in G exactly once.



Eulerian circuit

A Eulerian circuit in G is a closed Eulerian walk
(closed = start and end at same vertex).

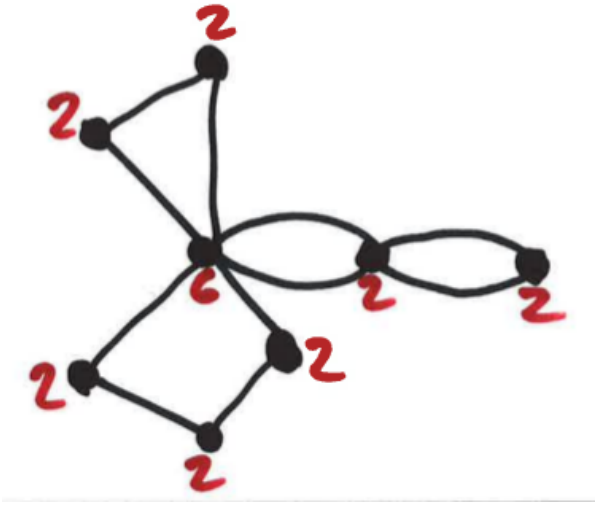
The bridges of Königsberg (1736)



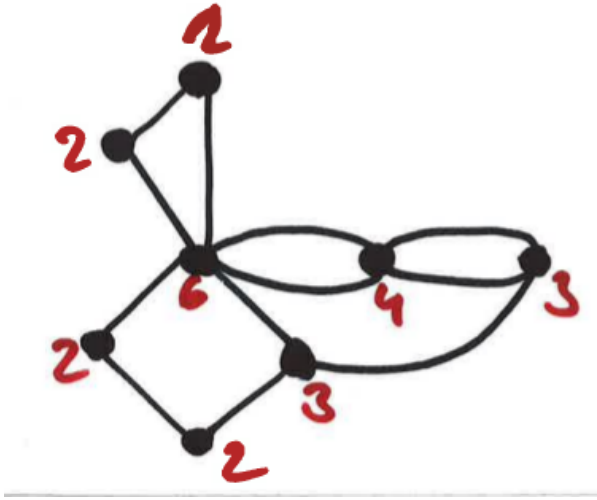
Existence?

A simple algorithm can be used to check if a connected graph G contains an Eulerian walk or circuit:

1. count the vertives with odd degree
2. `if` counter is 0 return "Eulerian circuit exists"
   `elsif` counter is 2 return "Eulerian walk exists"
   `else` return "no Eulerian walk or circuit exists"



counter = 0
$\Rightarrow$ Eulerian circuit



counter = 2
$\Rightarrow$ Eulerian walk

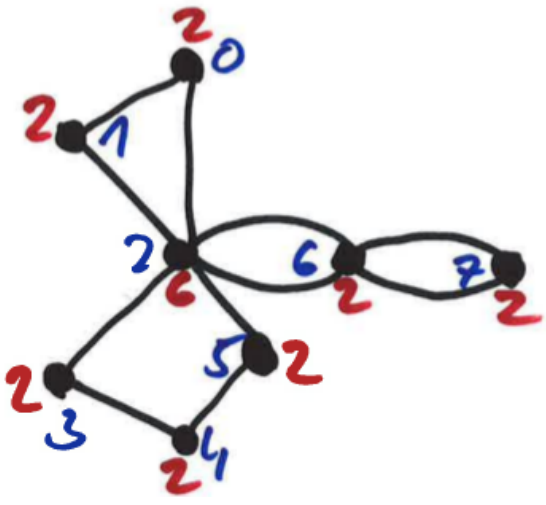
Eulerian circuit algorithm

If the graph contains an Eulerian circuit it can be generated using the following algorithm:

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are not equal to 0 and therefore edges exist return to step 1
5. combine generated circuits to a single Eulerian circuit

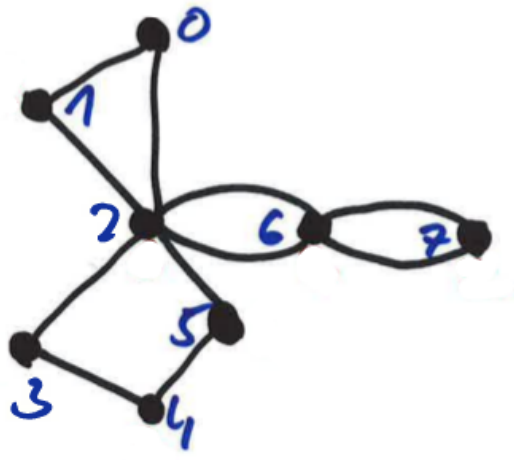
Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit



Example

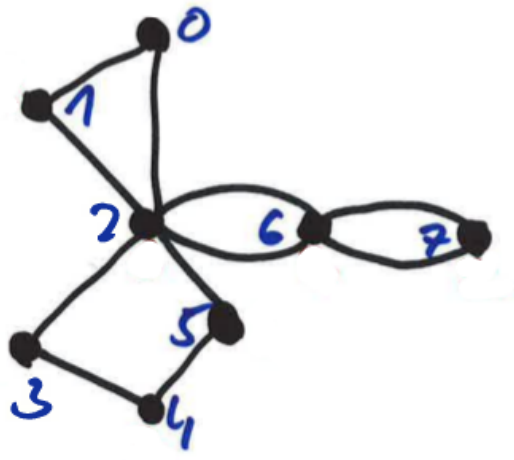
1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit



	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	1	1	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

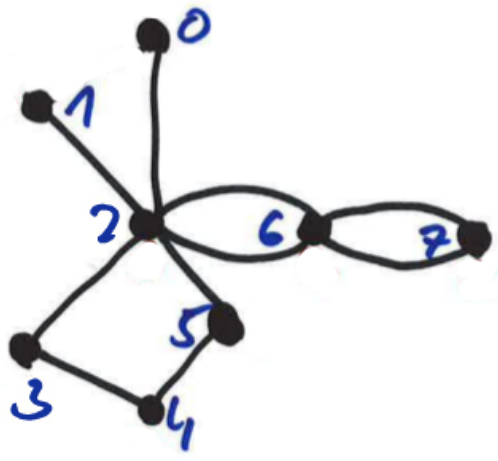


	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	1	1	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

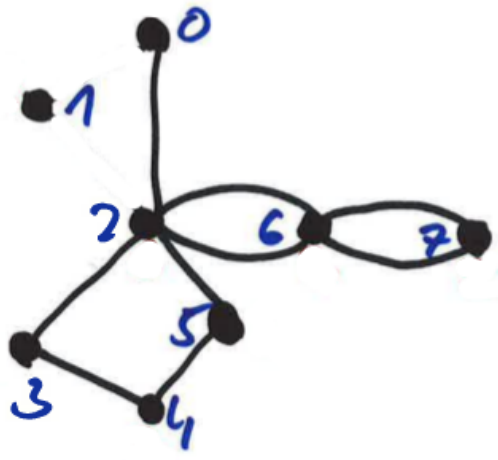


	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0
2	1	1	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

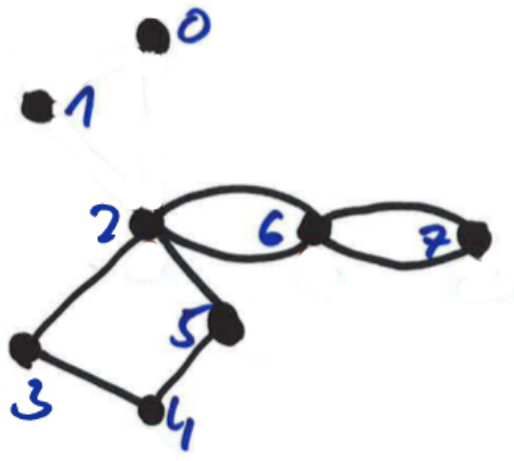


	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	1	0	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

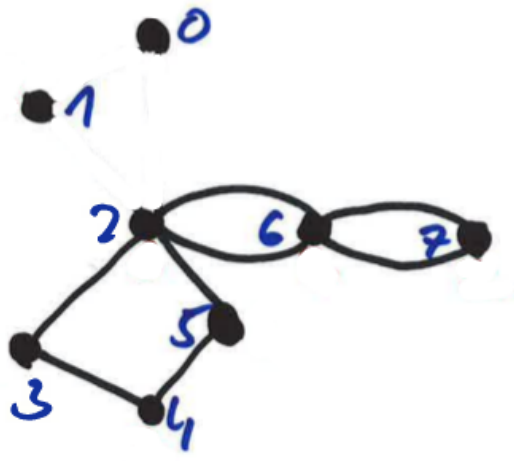


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0;

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

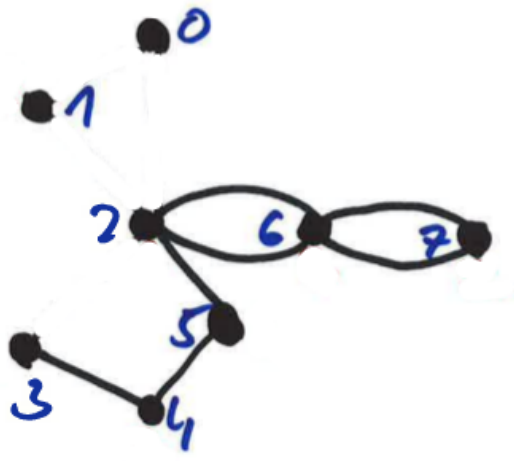


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

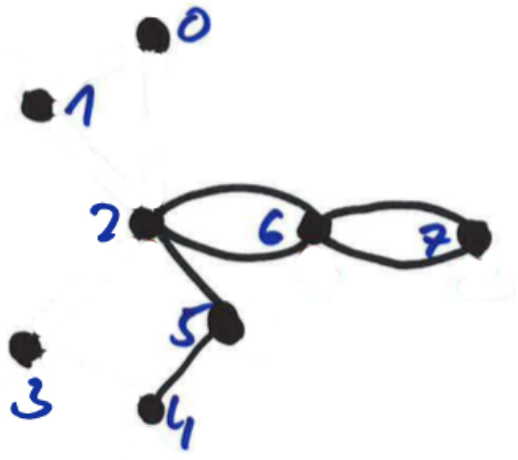


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	2	0
3	0	0	0	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

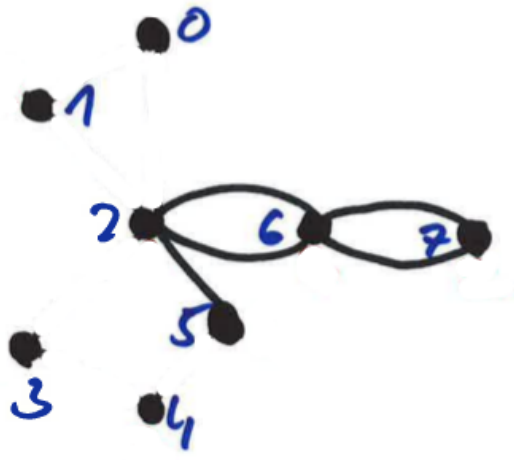


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3, 4

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

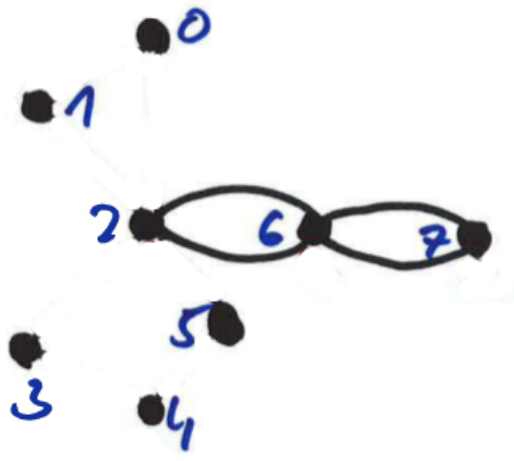


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3, 4, 5

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

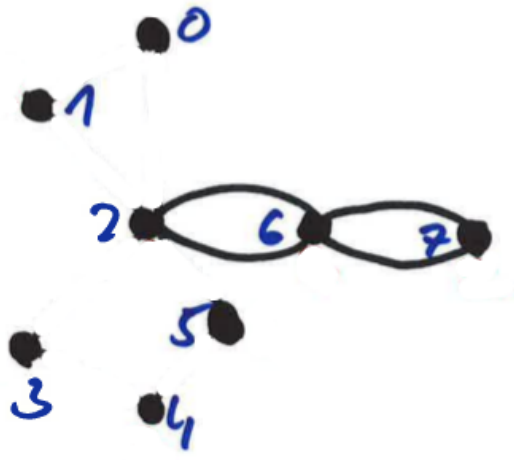


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3, 4, 5, 2;

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

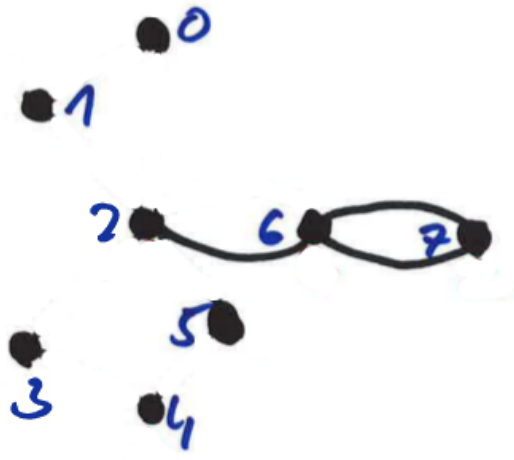


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3, 4, 5, 2; 2

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

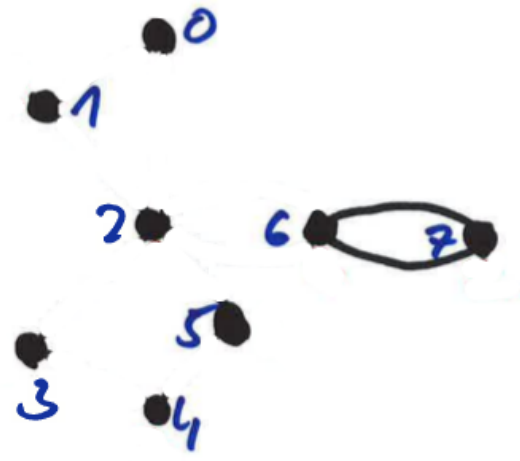


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3, 4, 5, 2; 2, 6

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

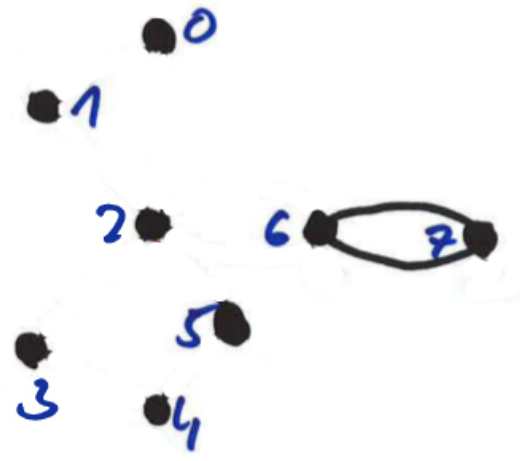


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3, 4, 5, 2; 2, 6, 2;

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

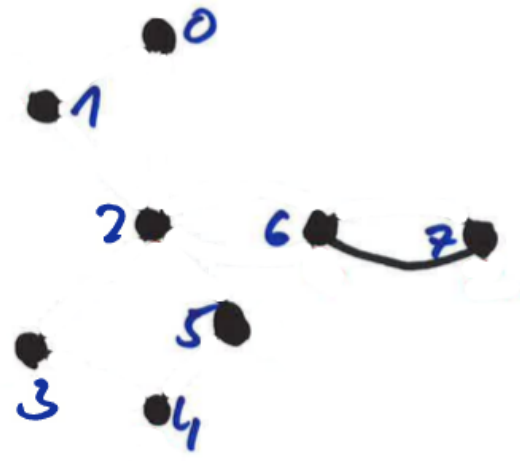


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	2
7	0	0	0	0	0	0	2	0

0, 1, 2, 0; 2, 3, 4, 5, 2; 2, 6, 2; 6

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

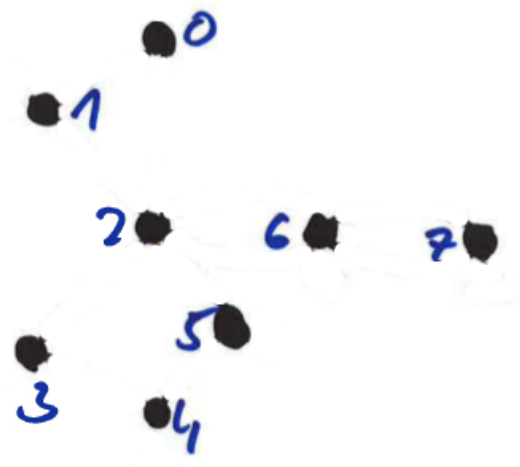


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	1	0

0, 1, 2, 0; 2, 3, 4, 5, 2; 2, 6, 2; 6, 7

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit

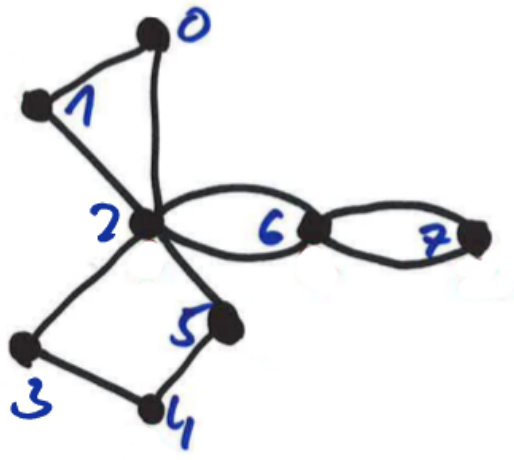


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

0, 1, 2, 0; 2, 3, 4, 5, 2; 2, 6, 2; 6, 7, 6;

Example

1. select (random) start vertex
2. walk along the edges until you reach the start vertex again
3. decrement the corresponding edge counts in the adjacency matrix
4. until all edge counts are unequal to 0 and therefore edges exist return to 1
5. combine generated circuits to a single Eulerian circuit



A valid Euler circuit is 0, 1, 2, 3, 4, 5, 2, 6, 7, 6, 2, 0;

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

0, 1, 2, 0; 2, 3, 4, 5, 2; 2, 6, 2; 6, 7, 6;

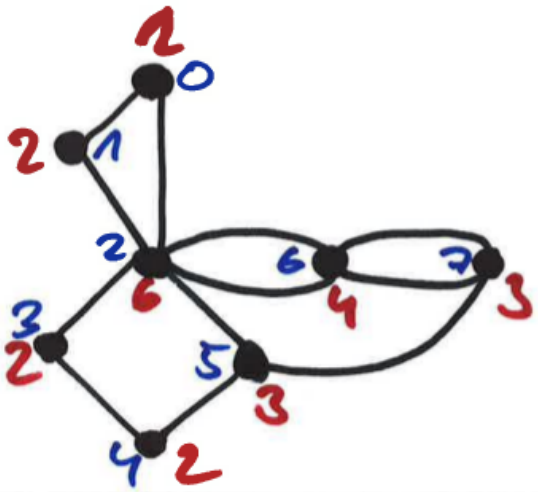
Eulerian walk algorithm

The algorithm to generate an Eulerian walk is based on the Eulerian circuit detection algorithm and works as follows:

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph
6. join the walk from the start to the end point and all Eulerian circuits into one Eulerian walk

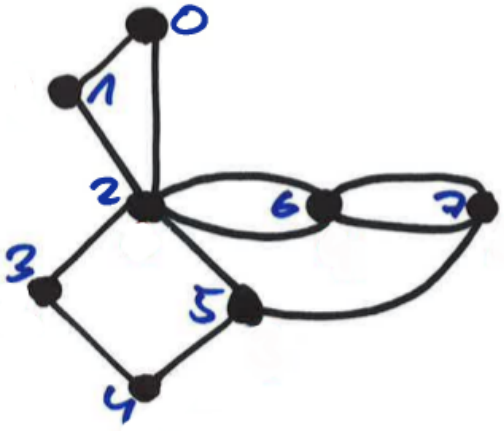
Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph



Example

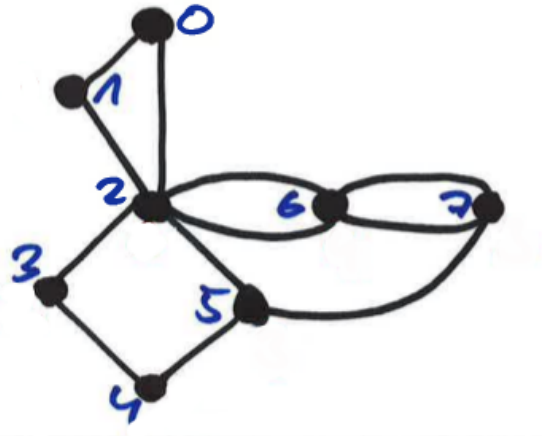
1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph



	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	1	1	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

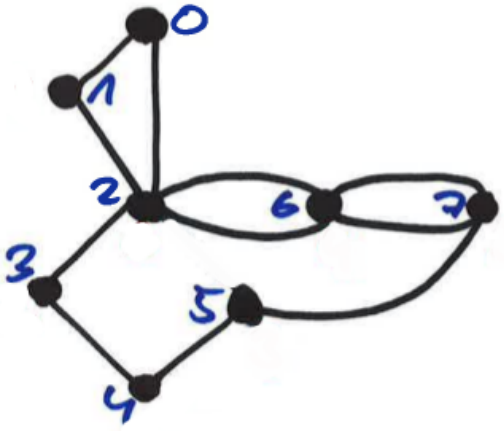


	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	1	1	0	1	0	1	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	1	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

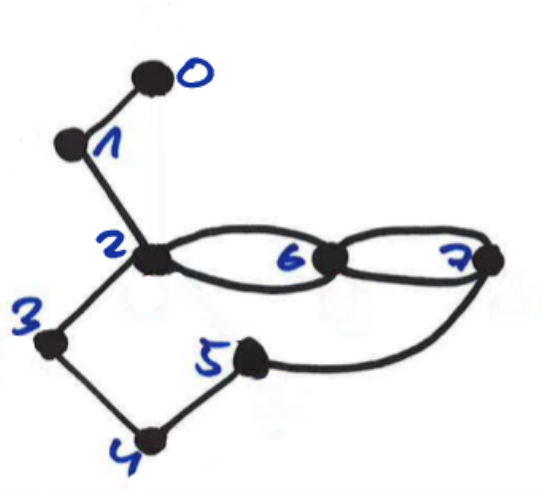


	0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	1	1	0	1	0	0	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	0	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5, 2

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

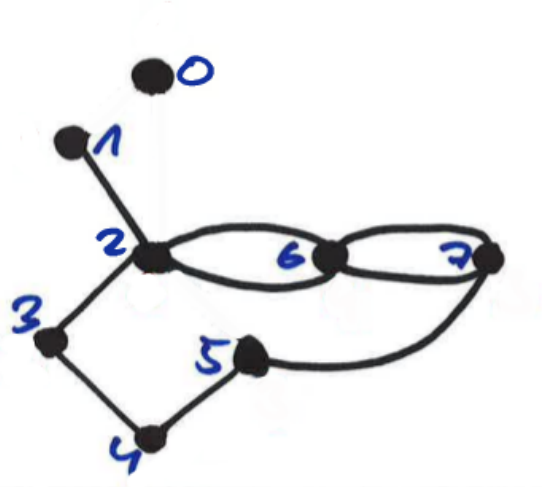


	0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
2	0	1	0	1	0	0	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	0	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5, 2, 0

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

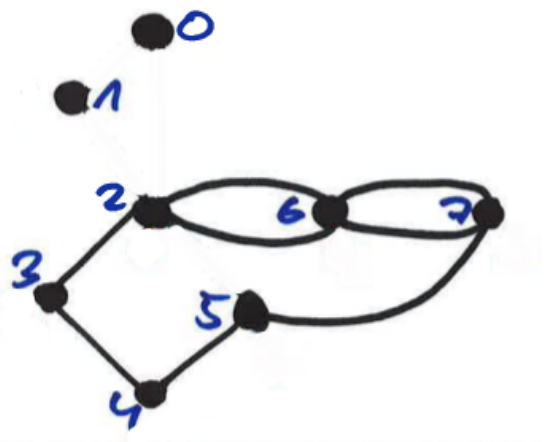


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
2	0	1	0	1	0	0	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	0	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5, 2, 0, 1

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

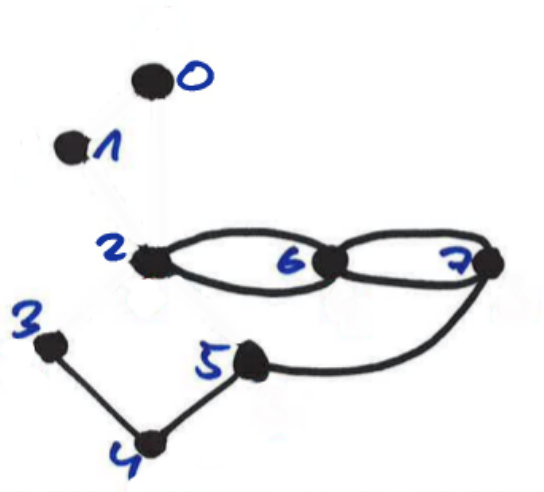


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	2	0
3	0	0	1	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	0	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5, 2, 0, 1, 2,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

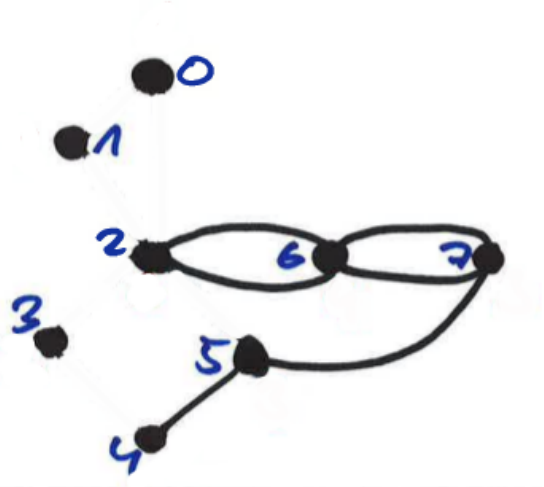


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0
3	0	0	0	0	1	0	0	0
4	0	0	0	1	0	1	0	0
5	0	0	0	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5, 2, 0, 1, 2, 3,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

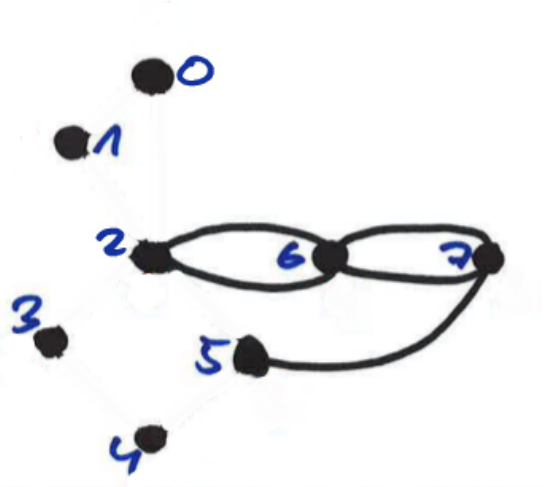


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	0	0	0	0	1	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5, 2, 0, 1, 2, 3, 4,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

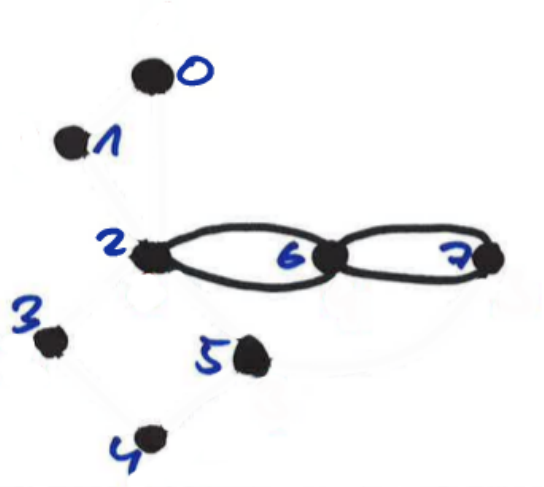


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**1**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**1**	2	0

5, 2, 0, 1, 2, 3, 4, 5,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

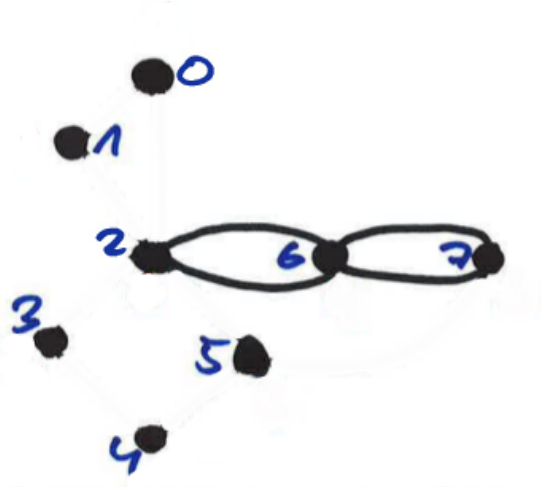


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**0**	2	0

5, 2, 0, 1, 2, 3, 4, 5, 7;

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

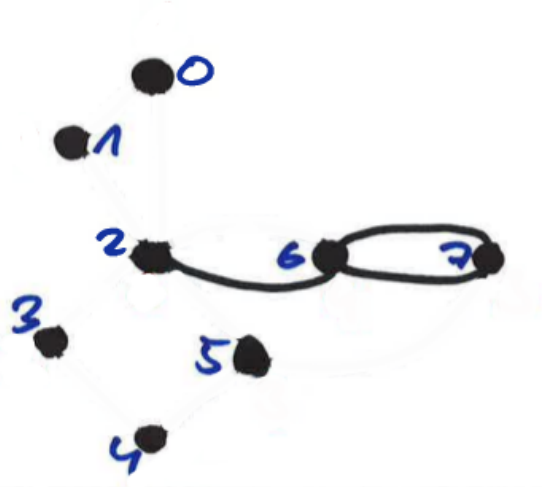


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	2	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	2	0	0	0	0	2
7	0	0	0	0	0	**0**	2	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

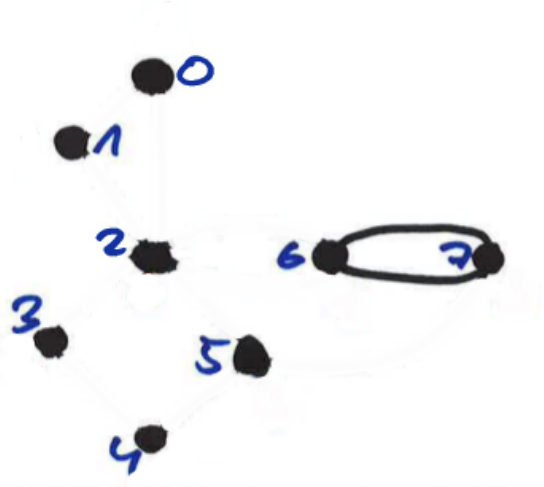


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	1	0	0	0	0	2
7	0	0	0	0	0	**0**	2	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2, 6,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

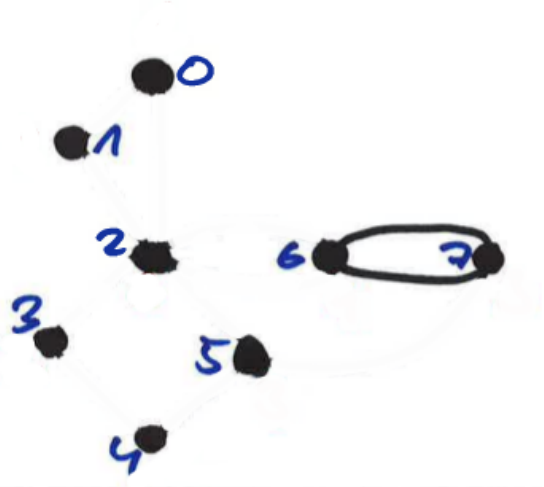


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	0	0	0	0	0	2
7	0	0	0	0	0	**0**	2	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2, 6, 2;

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

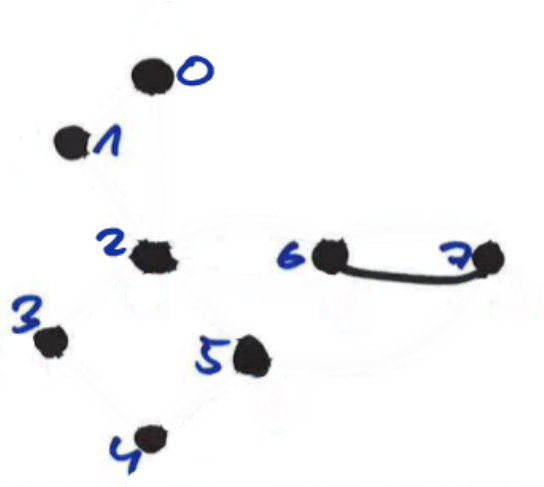


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	0	0	0	0	0	2
7	0	0	0	0	0	**0**	2	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2, 6, 2; 6,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

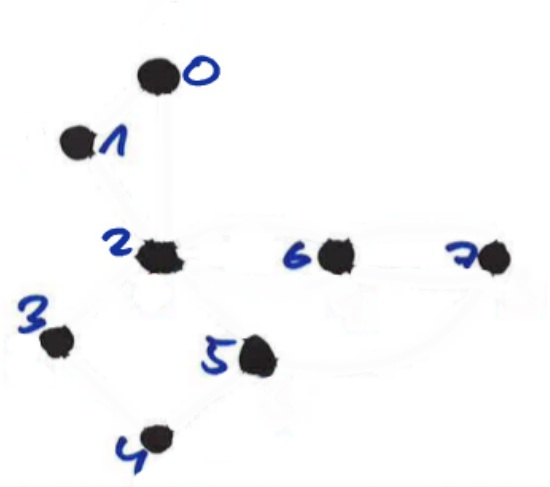


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	**0**	1	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2, 6, 2; 6, 7,

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

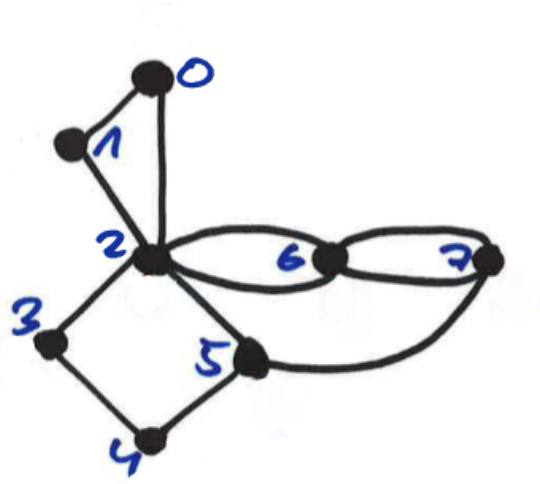


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	**0**	0	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2, 6, 2; 6, 7, 6;

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph

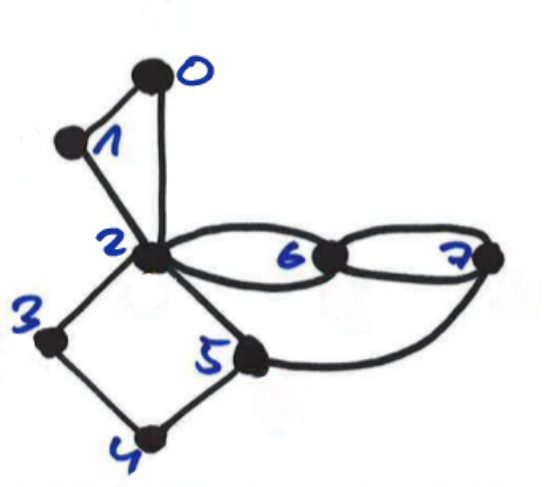


	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	**0**	0	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2, 6, 2; 6, 7, 6;

Example

1. use one of the vertex with odd degree as starting point
2. determine the other vertex with odd degree as end point
3. walk through the graph until you reach the end point
4. decrement the corresponding edge counts in the adjacency matrix
5. use the Eulerian circuit algorithm to explore the remaining graph



A valid Euler walk is 5, 2, 6, 7, 6, 2, 0, 1, 2, 3, 4, 5, 7

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	**0**
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	**0**	0	0

5, 2, 0, 1, 2, 3, 4, 5, 7; 2, 6, 2; 6, 7, 6;

Directed graph extensions

Eulerian circuit

A connected directed graph contains an Eulerian circuit if for each vertex: in-degree = out-degree.
Same Algorithm as for undirected graphs except that each edge has a direction.

Directed graph extensions

Eulerian circuit

A connected directed graph contains an Eulerian circuit if for each vertex: in-degree = out-degree.
Same Algorithm as for undirected graphs except that each edge has a direction.

Eulerian walk

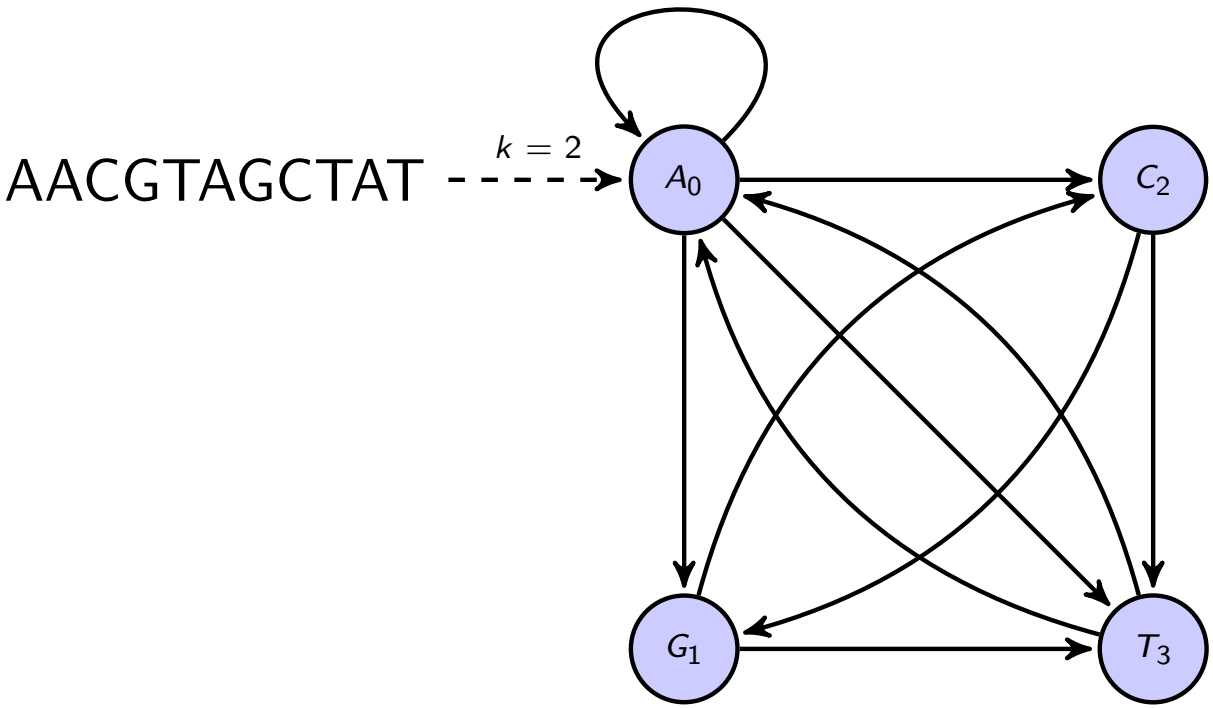
A directed graph has a Eulerian walk iff

- exactly one vertex (start): out-degree = in-degree + 1
- exactly one vertex (end): in-degree = out-degree + 1
- for all other vertices: in-degree = out-degree

k-let Shuffling

Step 1: Convert input sequence to multigraph where each vertex is a (k-1)-let of the input sequence.

k = 2 (dinucleotide shuffling)

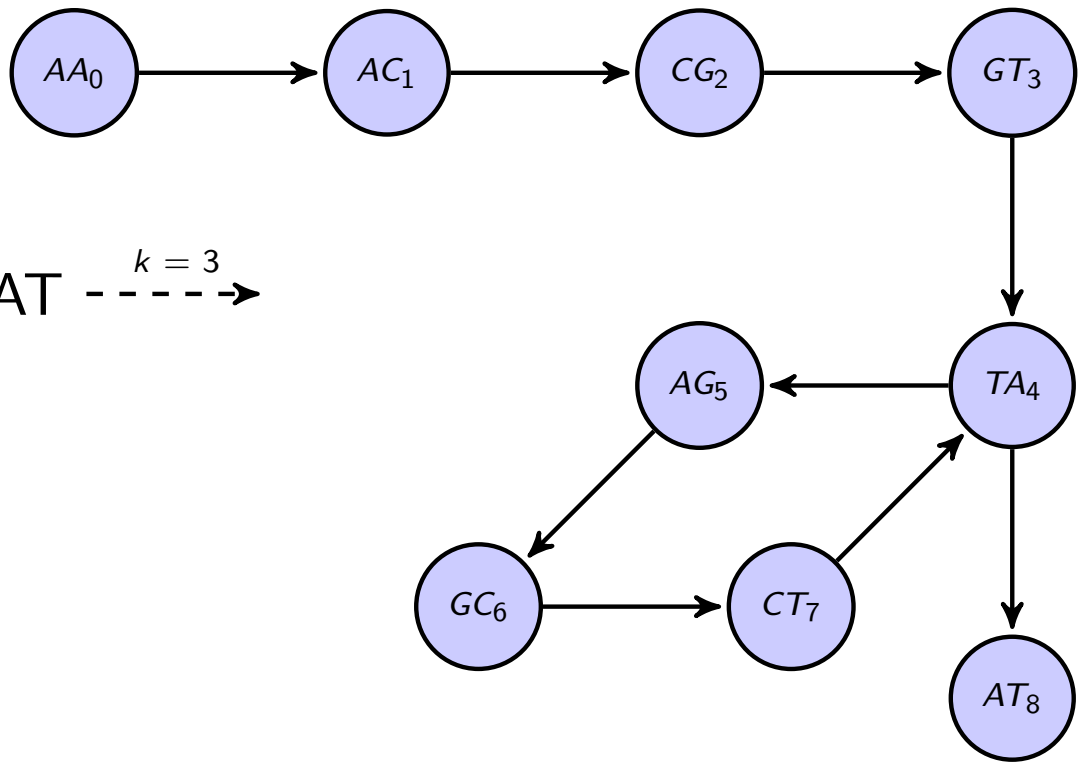


k-let Shuffling

Step 1: Convert input sequence to multigraph where each vertex is a (k-1)-let of the input sequence.

k = 3 (trinucleotide shuffling)

AACGTAGCTAT $\xrightarrow{k=3}$

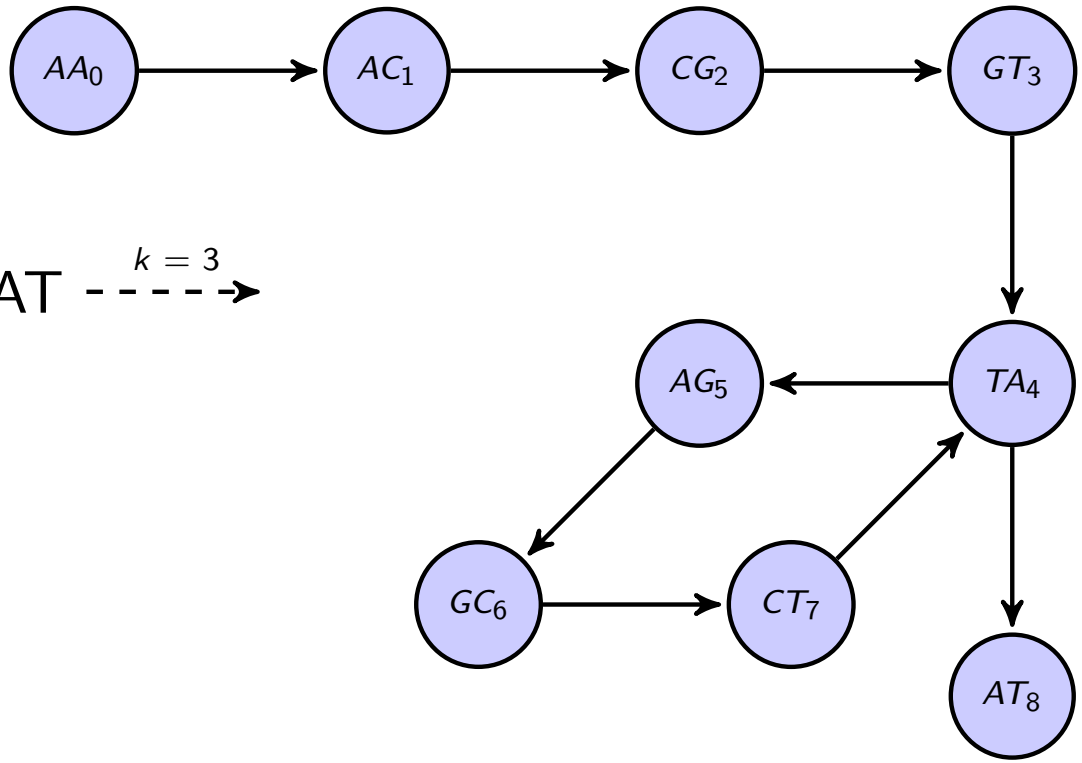


k-let Shuffling

Step 1: Convert input sequence to multigraph where each vertex is a (k-1)-let of the input sequence.

k = 3 (trinucleotide shuffling)

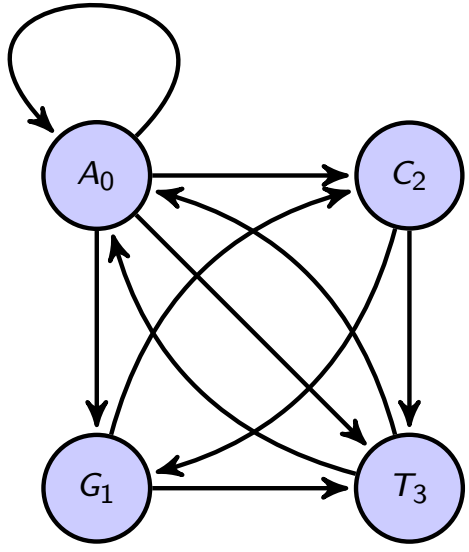
AACGTAGCTAT $\xrightarrow{k=3}$



In principle, one could now produce k-let shuffles, by randomly selecting Euler paths. Any problems!?

Uniform k-let Shuffling

Step 2: Create data structure to store multigraph information.




```
#          sym,id, neighbors, seen, next
tab[0] = ['A', 0, [0,1,2,3],    0, udef]
tab[1] = ['C', 1,     [2,3],    0, udef]
tab[2] = ['G', 2,     [3,1],    0, udef]
tab[3] = ['T', 3,     [0,0],    0, udef]
```

Uniform k-let Shuffling

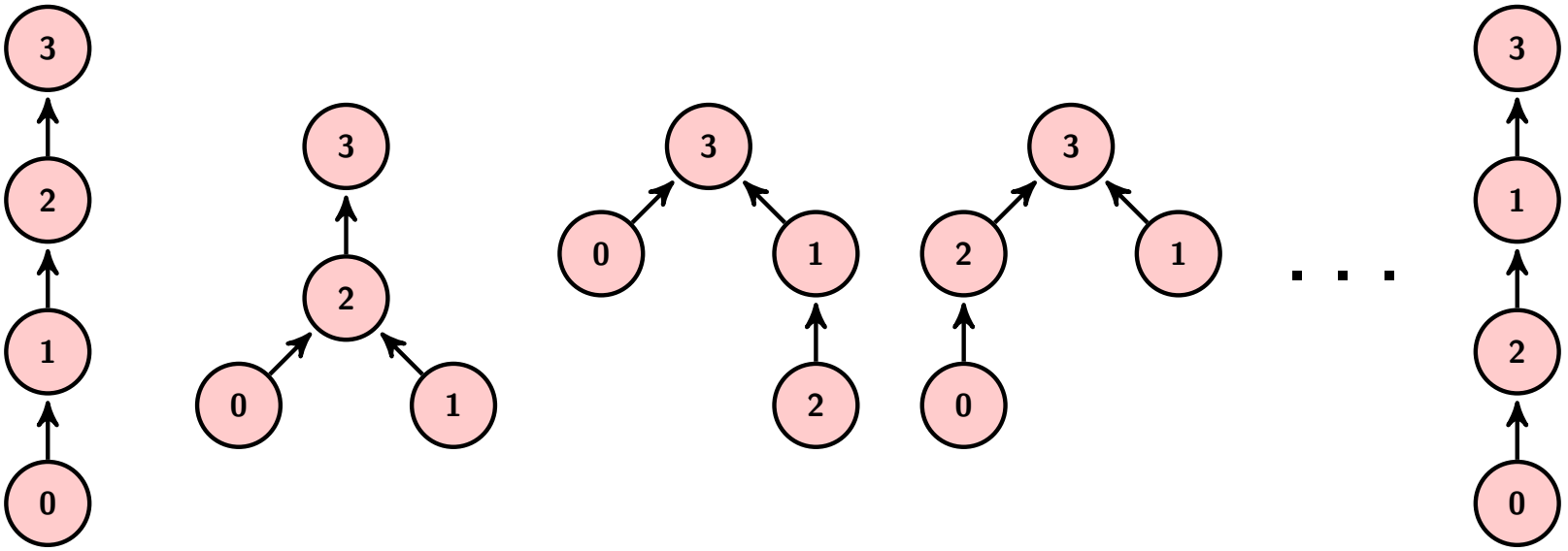
Step 3: Create random spanning tree (arborescence). The root is the last and starting vertex is the first k-let of the sequence (Wilson's algorithm).

```
#          sym,id, neighbors, seen, next
tab[0] = ['A', 0, [0,1,2,3],    0, udef]
tab[1] = ['C', 1,     [2,3],    0, udef]
tab[2] = ['G', 2,     [3,1],    0, udef]
tab[3] = ['T', 3,     [0,0],    0, udef]
```

Uniform k-let Shuffling

Step 3: Create random spanning tree (arborescence). The root is the last and starting vertex is the first k-let of the sequence (Wilson's algorithm).

```
#          sym,id, neighbors, seen, next
tab[0] = ['A', 0, [0,1,2,3],    0, udef]
tab[1] = ['C', 1,     [2,3],    0, udef]
tab[2] = ['G', 2,     [3,1],    0, udef]
tab[3] = ['T', 3,     [0,0],    0, udef]
```



Uniform k-let Shuffling

Step 3: Create random spanning tree (arborescence). The root is the last and starting vertex is the first k-let of the sequence (Wilson's algorithm).

```
#          sym,id, neighbors, seen, next
tab[0] = ['A', 0, [0,1,2,3],    0, udef]
tab[1] = ['C', 1,     [2,3],    0, udef]
tab[2] = ['G', 2,     [3,1],    0, udef]
tab[3] = ['T', 3,     [0,0],    0, udef]
```

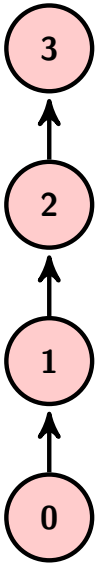
How to create/select a random spanning tree T?

"for each node i (except the end node) randomly select one out edge (i, j) w/o generating cycles" (set `next[i]:=j`)

Uniform k-let Shuffling

Step 4: Store selected spanning tree in data structure (column `next`)

```
#          sym,id, neighbors, seen, next
tab[0] = ['A', 0, [0,1,2,3],    1,    1]
tab[1] = ['C', 1,     [2,3],    1,    2]
tab[2] = ['G', 2,     [3,1],    1,    3]
tab[3] = ['T', 3,     [0,0],    1,    T]
```



Uniform k-let Shuffling

Step 5: Shuffle each `neighbors[i]` such that `next[i]` occurs as last entry

```
#          sym,id, neighbors, seen, next
tab[0] = ['A', 0, [2,0,3,1],    1,    1]
tab[1] = ['C', 1,     [3,2],    1,    2]
tab[2] = ['G', 2,     [1,3],    1,    3]
tab[3] = ['T', 3,     [0,0],    1,    T]
```

Uniform k-let Shuffling

Step 6: Start from first (k-1)-let of the input sequence; in each step, walk from node *i* to the first entry in `neighbor[i]`, remove the entry and continue.

```
#          sym,id, neighbors, seen, next
tab[0] = ['A', 0, [2,0,3,1],    1,    1]
tab[1] = ['C', 1,     [3,2],    1,    2]
tab[2] = ['G', 2,     [1,3],    1,    3]
tab[3] = ['T', 3,     [0,0],    1,    T]
```

$0 \to 2 \to 1 \to 3 \to 0 \to 0 \to 3 \to 0 \to 1 \to 2 \to 3$ AGCTAATACGT

The assignments

A4 `https://github.com/TBIAPBC/APBC2021/tree/master/A4`

This assignment is due on 26 April 10 AM !

(GitHub pull request must be placed by this time at latest)

Happy hacking!