

Learning objectives ⓘ Resources for students ⓘ Resources for Tutors ⓘ Show all ⓘ Reduce all ⓘ

TOPIC

Instructions

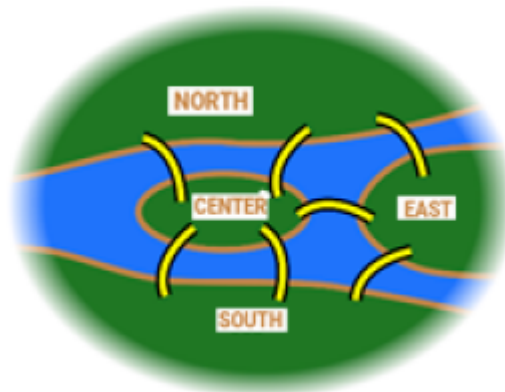
QUIZ:PROSIT: SMALL DETOUR

Your study focus within the scope of ADEME was not randomly chosen. In fact, CesiCDP is already engaged in another assignment, which is funded by the Grand EST region and part of an ambitious project around smart cities. The aim is to address a specific problem faced by local authorities:

Every day, the technical teams head to the streets of municipalities in the region for equipping traditional street lamps with connected devices in order to make street lighting 'smart'. Each team designs its own itinerary 'on a trial basis' but there is no guarantee that this route is 'optimal'. By proposing a program that calculates the best itinerary, this will mean big savings in terms of fuel consumption and time required for each route. And one can easily imagine other useful applications for this program.

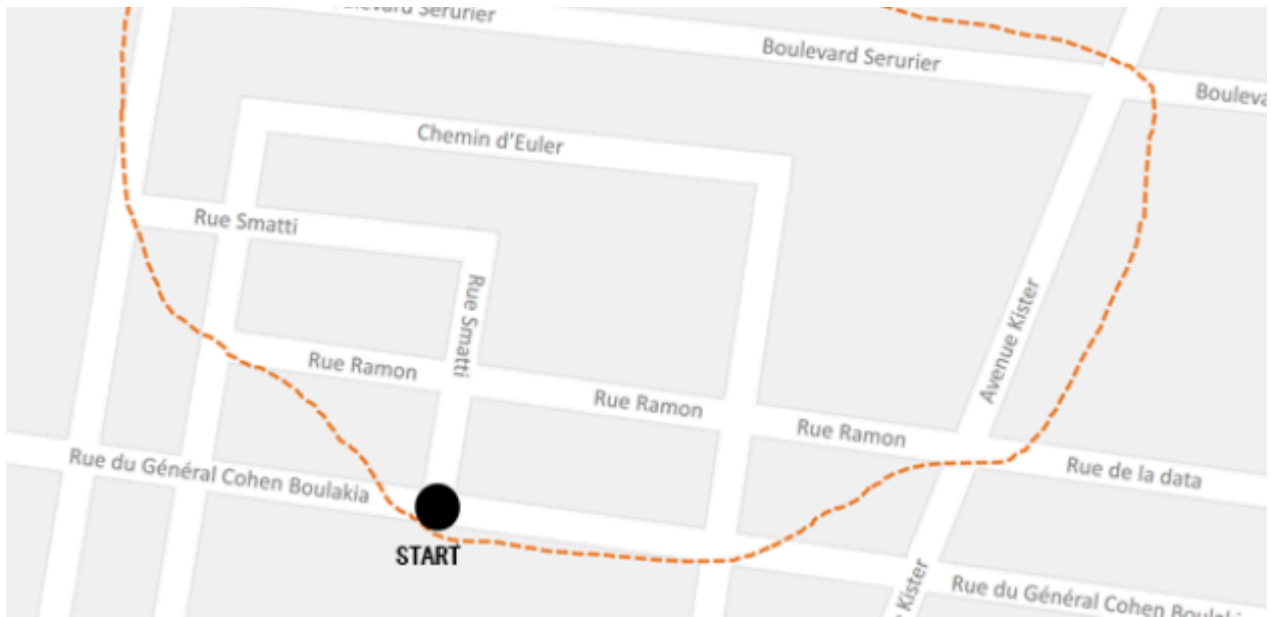
The themes and the algorithmic approach of these two projects are therefore very similar, so it has been decided that you will be involved in the latter until the end of the week as a prelude to your own assignment. Agathe, the person in charge of this project, is particularly looking forward to your programming expertise. As for you, the ambition is clearly to be able to reuse this work in your own project.

She shares her thoughts on the subject: 'This reminds me of that mathematical problem you find in magazines'. And then she starts drawing on a flipchart:



Agathe: 'The goal is to find a walking route through the streets of this charming city that allows, from any given starting point, to cross each bridge exactly once, and to return to the starting point'.

We can indeed see the similarities of the problem by comparing this with a map example of an urban area to be covered by a route:



As for you, you immediately picture yourself modeling this with a graph. The associated program should not be that hard to write. As you think about the algorithm to be set up and the form of the solution to be found, you notice that things will still have to be properly planned. In fact, you would normally want to create the path crossing each area (or crossroad) only once, whereas it is the path crossing each bridge (or street) only once that is expected.

Agathe, who had been standing by her flipchart trying to solve her bridge riddle, then takes advantage of your bewilderment to add more to it: 'You know what? I think it's impossible...'

You find it hard to imagine, especially since you have just come across a resource on the Minimum Spanning Tree algorithm, which seems to perfectly address the situation. This must depend on the map layout. One can easily imagine 4 streets arranged in a square configuration. In this case, there is a solution, which is quite obvious.

But Agathe's problem is far from solved: if there is no solution to find a path that only crosses each street once, the program will have to return something to optimally guide the teams in their maintenance routes.

The other concern Agathe has is about the performance of your algorithm. 'Having to wait two weeks for it to provide the solution is unacceptable...'

But she is not yet aware of your programming skills, so you commit to them demonstrate by providing her with the computation time by counting the number of milliseconds of CPU occupation by the process needed to run the program.

However, she is still sceptical: 'That will provide a first insight, that's true. But it's not that simple. There, we have an area with only a few streets and crossings. But what will happen once the program has to scan the whole city?'

It's true that the more streets there are... unless this depends on the number of crossroads instead? This complexity would have to be determined theoretically in order to anticipate the use of your algorithm on even larger cities. It would be necessary to know at what length the algorithm can no longer be used.

▼ Resources for students

▼ Resources in English

- How Königsberg bridges changed mathematics - Dan Van der Vieren [🔗](#) EN
- [🔗](#)Graph Theory: Euler Paths and Euler Circuits [🔗](#) - EN
- Introduction to Graph Theory [🔗](#) - EN (chapters 2 and 3)
- Algorithmic complexity [🔗](#) - EN
- Graphical representation of asymptotic complexities (jupyter notebook) [zip] [📄](#) - EN (translated by the author)
- Algorithmic complexity part 1 - Asymptotic complexity [🔗](#) , video Benjamin COHEN BOULAKIA (2020) - FR (explanations in FR by the author of the preceding resource)

EDUCATIONAL GUIDE

▼ Tutoring help

Learning Objectives

- Identifying the concepts and uses of the graph theory
 - Listing and describing the constituent elements of a graph (vertex, arc or edge, weighting) and their parameters (degree, order, connectivity)
 - Distinguishing between directed and undirected graphs
 - Differentiating between Eulerian and Hamiltonian graphs
 - Recognizing specific problems that can be modeled by a graph (scheduling, vehicle routing, networks...)
- Modeling a specific problem using a graph
 - Modeling the data of a specific problem in the different possible representations of a graph (sagittal representation, adjacency list, adjacency matrix)
 - Illustrating the notions of Eulerian cycle, Eulerian path and Eulerian graph
 - Proving that a graph is Eulerian (Euler's Theorem)
- Solving a specific problem using a graph
 - Demonstrating the search algorithm for an Eulerian cycle
 - Demonstrating the algorithm for the Chinese Postman Problem
 - Presenting the implementations of the different possible representations of a graph (sagittal representation, adjacency list, adjacency matrix)
 - Creating a program for finding the 'optimal' solution to a specific problem modeled by a graph
- Analyzing the complexity of an algorithm
 - Explaining the notion of an algorithm's asymptotic complexity
 - Calculating the complexity of an algorithm, and representing said complexity in asymptotic form
 - Analyzing the experimental computation time of an algorithm
 - Arranging the different classes of asymptotic complexity in a specific order (from quasi-linear to exponential)

Prosit philosophy and false leads

The prosit philosophy is to use graphs as a modeling and problem-solving tool. Students must present a wide range of problems that can be modeled by a graph in the prosit outcome and feedback. They must then focus on the Eulerian graphs and the Chinese Postman Problem, which are related to the specific problem of our prosperity:








1. Do we use Euler's theorem to find out whether there is an itinerary that crosses all the streets in the area to be covered exactly once?
2. If so, then we search for this itinerary using the search algorithm of an Eulerian cycle.
3. If the answer is no, we then search for an itinerary that crosses all the streets by crossing the same street less than 2x times using the Chinese Postman algorithm.

The Minimum Spanning Tree is a false lead here. Students should realize this quickly as they have studied a field of application for the MST in the 2nd year: routing. The same goes for the Hamiltonian cycle notion, which is also a false lead.









The Workshop focuses on the execution of a program for finding an Eulerian cycle, all done in a Jupyter Notebook.

Descriptions of resources

Resources in EN

- How Königsberg bridges changed mathematics - Dan Van der Vieren  EN
-  Graph Theory: Euler Paths and Euler Circuits  - EN
- Introduction to Graph Theory  - EN (chapters 2 and 3)
- Algorithmic complexity  - EN
- Graphical representation of asymptotic complexities (jupyter notebook) [zip]  - EN (translated by the author)
- Algorithmic complexity part 1 - Asymptotic complexity , video Benjamin COHEN BOULAKIA (2020) - FR (explanations in FR by the author of the preceding resource)

Original resources in FR

- Shining in society: The 7 bridges of Königsberg (video) 
very direct and fun video explanation of the königsberg bridge problem
- Bac revision: Eulerian chain, Eulerian cycle (video) 
short explanatory video on the Eulerian cycle
- Bac Revision: Euler's Theorem (video) 
short explanatory video on Euler's theorem
- Scholarvox reference work: Discovering Graphs and Graph Algorithms, Christian Laforest, EDP SCIENCES 
(chapters 1 to 9)
complete work on graphs
- Scholarvox reference work: Algorithms, Cormen, Dunod  : complexity (chapter 2), graphs (chapters 5 and 6)
reference work on the entire algorithmic part, the targeted chapters deal with the complexities of an algorithm and graphs
- Graphical representation of asymptotic complexities (jupyter notebook) [zip]  - EN
jupyter notebook allowing to visualize the asymptotic curves of the complexity quantities
- Engineering Techniques - Graph Theory [pdf] 
Technical reference resource for engineers - In FR Only, removed from EN course
- Zest of Knowledge - Graph and Graph Representation 
Synthetic online course on "Graphs and graph representation" on Zestedesavoir.com (2016). In FR only, removed from the EN course

Notes for the tutor, tricks and tips

Regarding the issues in this prosit, we particularly address a specific problem, which is basically finding an Eulerian cycle (finding a cycle in a graph that crosses all the EDGES of this graph exactly once).

Finding an Eulerian cycle is an 'easy' problem to solve by a computer program. We will discuss, in the next prosit, a problem that seems very similar to the problem of this prosit, and which consists in finding a Hamiltonian cycle (finding a cycle that crosses all the VERTICES of the graph exactly once). In contrast, this problem is 'hard' to solve by a computer program.

The temporal and spatial algorithmic complexity of a problem (and in particular the classes P and NP) will therefore be addressed in the next prosit. However, the matter relating to the time performance of an algorithm is addressed as of this

✓ Prosit Kick-Off

✓ Tutoring help

Discovering and clarifying the situation

- *Unknown words:*

Smart city, connected devices, smart, itinerary, optimal, program, Graph Theory, algorithms, Minimum Spanning Tree, efficient, temporal complexity/time complexity, asymptotic

- *Context:*

Before starting the ADEME project, the company decided to place us working on a project in partnership with the Grand-EST region, which consists of implementing a solution to help the technical teams optimize their routes in the context of installing connected street lamps.

Analyzing the need

- *Problem:*

Apparently, there isn't always a path that meets all requirements. How to find the optimal one?

It seems difficult to estimate the computation time required by the algorithm to provide the solution

- *Constraints:*

The program must return an itinerary proposal

The algorithm must work regardless of the city layout

The program must perform the calculation within a reasonable time

- *Deliverables:*

A program that provides an 'optimal' itinerary for the technical teams' route when there is one.

A development plan for the program, proposing an 'optimal' itinerary even if it is impossible to cross each street exactly once.

An assessment of the time required by the algorithm to provide the solution.

Generalization

- - Modeling and problem-solving using graph theory.

Possible solutions

- What exactly is meant by 'optimal' itinerary? Note for tutors: try to make students understand that the team must return to its starting point, crossing all the streets in area A exactly once, when possible.
- Do you know of any problems similar to the one addressed in the prosit? In a building, with electricity?
- In network routing? In parcel delivery?
- In what form should the graph be represented as input parameter for the program?
- How can graphs help us model the situation?
- Try to transform area A of the map provided by Agathe into a graph!
- What do we observe in these graphs? Specially on the edges that represent the streets/bridges?
- If we don't measure the performance of an algorithm according to the number of CPU cycles, how can we do this? Can we derive this just by studying the algorithm?

Advanced Algorithms Preparation Project 1 - PBL Loop 2 - PBL Loop 3 - PBL Loop 4 - PBL Loop

- Whether it is the number of streets or crossings, how can we describe the computation time of an algorithm if we don't know this number before hand? How can we generally describe its performance?

Action plan

1. Theoretical study of the problem

- Expressing the problem as a graph Establishing the expected solution format
- Finding a way to determine if it is possible to cross each street in a city only once based on our plan
- Determining the right problem-solving algorithm, if this is possible.

If this is not possible (cross the same street 2 times, if necessary)

2. 2. Program development

- Choosing a data structure for graphs
- Implementing the algorithm using this structure
- Analyzing performance

Implementing the algorithm by adding a count (the CPU usage time or the number of loop revolutions)

Theoretically analyzing the number of instructions in the worst-case scenario

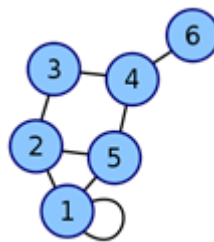
✓ Prosit Outcome and Feedback

✓ Tutoring help

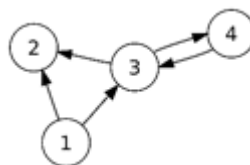
Definitions

- **Useful reminders about graphs**

A graph is a **set of vertices** (or nodes). As some vertices may be connected to each other by **edges**, we say that these vertices are **adjacent**. When a vertex is connected to itself by an edge, this edge is also called a **loop**. In the graph below, there are 6 vertices (numbered from 1 to 6) as well as 8 edges: between 1 and 2 (which are therefore adjacent), between 2 and 3... (including a loop on vertex 1). We often note that V is the set of vertices in this graph, and E is the set of edges



This graph is **undirected** since the edges have no direction, so we can say that we can go from vertex 1 to vertex 2 but also from vertex 2 to vertex 1. We talk about a directed graph when the edges are **directed** (the edges are also called arcs in this case), as in the graph below. In this graph, we can say that we can go from vertex 3 to vertex 2 but not from vertex 2 to vertex 3.



The **order** of a graph is its number of vertices. The **degree** of a vertex is the number of edges to which said vertex is connected. In the previous graph, the order is 4 and the degree of vertices can be represented by the table below.

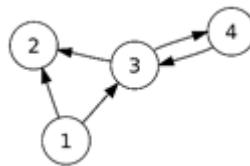
Degree	2	2	4	2
---------------	---	---	---	---

A **path** (resp. a **trail**) of **length** n is a sequence of n edges allowing to connect two vertices of an undirected (resp. directed) graph. When the starting point of a path (resp. of a trail) is also its ending point, this is known as a **cycle** (resp. a circuit).

An undirected (resp. directed) graph is **connected** when any pair of vertices can be connected by a path (resp. a trail).

Our 1st graph is connected since there is indeed a path between each pair of vertices (in other words, all vertices are mutually reachable). However, our 2nd graph is not connected since there is no trail between vertex 4 and vertex 1, for example.

A **weighted** graph is a graph in which each arc or edge is associated with a **weight** (a numerical value).



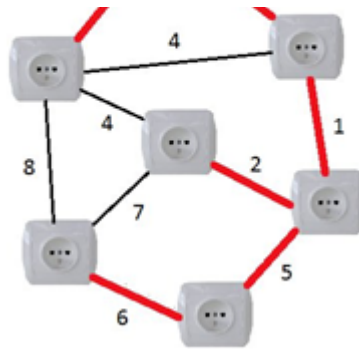
These values can represent the distance between two vertices, the link capacity, its cost...

- **Specific problems that can be modeled by graphs**

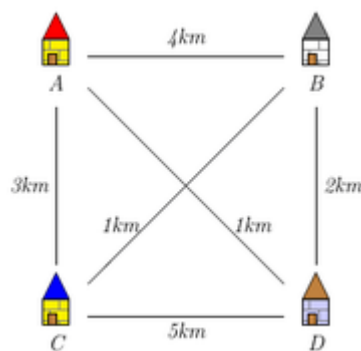
The graph theory is a popular **modeling** and **problem-solving** tool in a variety of fields, from basic sciences to specific technological applications. There are many applications:

- **Transport networks in general (electrical, fluid, IT and telecommunications applications, vehicles)** . In this case, by considering weightings, it is possible to take into account the latency (eg the distance in a road network), or the capacity of a link (the number of lanes in a road, the bandwidth of a network cable). In this type of problem, we can search for the shortest paths (GPS, IP routing...), organize vehicle routes (this is the case of the prosit if we consider only one vehicle), broadcast paths...
- **Location** (location of warehouses in the goods distribution networks, antennas...), in order to optimize their positioning (eg to maximize the geographical coverage of a 5G antenna network)
- **Task scheduling and resource allocation** (crew rostering problems in airlines, assembly line optimization).
- **Social media** used to describe interactions between individuals (statistically speaking, not necessarily people), for example to produce predictive epidemiological dissemination models. In this case, we are often dealing with very large graphs.

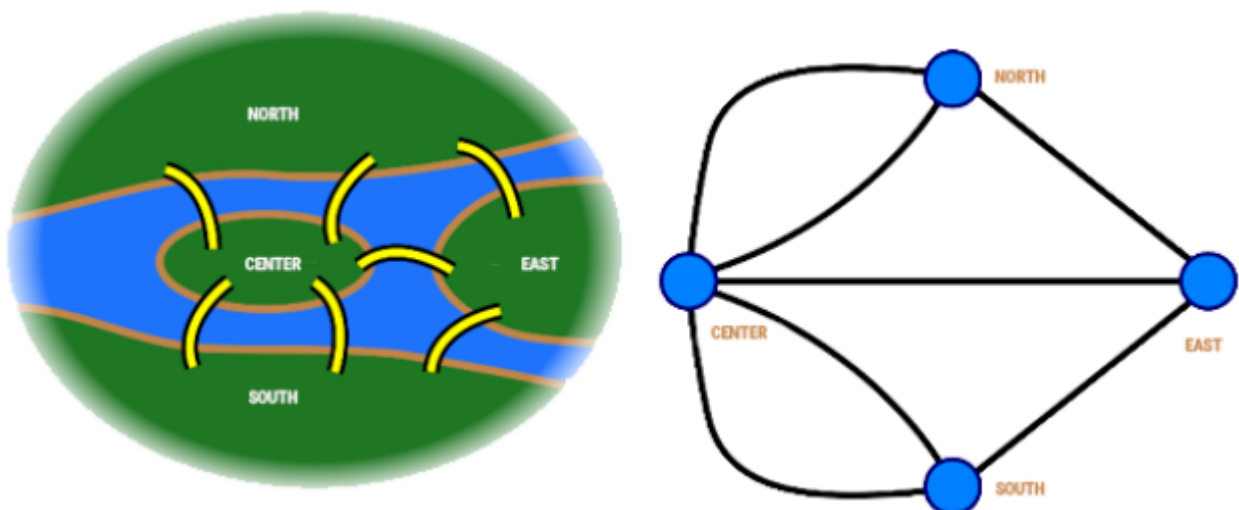
Example 1: Imagine that you want all your sockets to be connected to the mains using the **minimum amount** of electrical wire in order to **minimize the installation cost**. As a result, as shown by the graph below, you will associate a node of your graph with each socket, and an edge with each wire that it is possible (but not required) to install, having the distance between each socket as the value of the edge. **Your job is to choose the shortest set of cables to connect all the sockets.** Here the set of the shortest wires is coloured red. This corresponds to the search for the minimum spanning tree: *we can clearly see here how the MST is a false lead for the prosit issue.*



Example 2: The Travelling Salesman Problem (see graph below) that, given a list of cities and distances between all city pairs, determines the shortest path which visits each city exactly once, and ends in the city of departure. This corresponds to the search for a Hamiltonian cycle, the core topic in the next prosit (*and we don't say anything else, especially regarding the complexity aspect*).



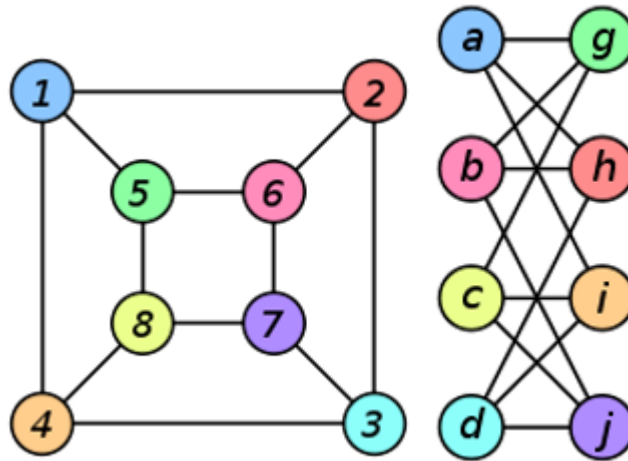
Example 3: The seven bridges of Königsberg problem is a historical mathematical problem solved by Euler in 1736. The city of Königsberg (nowadays Kaliningrad) is built around two islands located on the Pregel River and connected to each other by a bridge. Six other bridges connect the riverbanks either to one island or the other, as represented in the diagram below. The problem consists in **determining whether or not there is a walking route through the streets of Königsberg that allows, from any given starting point, to cross each bridge exactly once, and to return to its starting point**, knowing that we can only run through the Pregel River by crossing the bridges.



Therefore, this problem can be represented in the form of a graph where each vertex represents an area of the city and each edge represents a bridge that joins two areas (see above).

- **Graph representation**

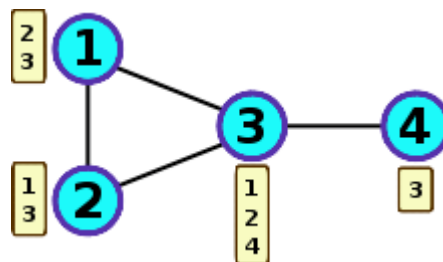
below.



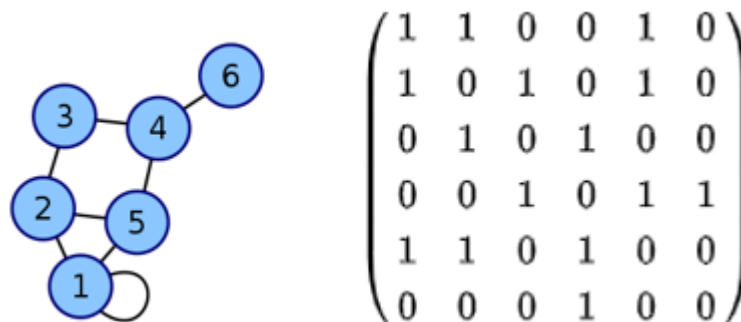
The major disadvantage of the sagittal representation of a graph is that the data associated with such graph cannot be processed. To do this, it is necessary to have a **representation that can be handled by a computer program**. So we must find a suitable data structure to store it, one that is memory-efficient and that allows algorithms to exploit it quickly.

The graph density is often the decisive criterion for choosing between the matrix or the list. The density of a graph is the ratio of the number of edges to the total number of possible edges.

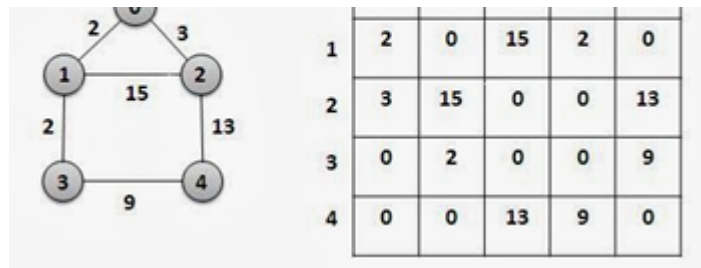
The **adjacency list** is the most common way to store a graph in memory, as this list matches the intuitive representation that we make of a graph. The adjacency list of a vertex is the list of its neighbours (or the list of edges that connects a vertex to its neighbours). This is a fairly compact representation when there are only a few edges (**sparse graph**) since the overall list contains $2 \times m$ elements, where m is the number of edges. In the example below, the adjacency list is shown in yellow for each vertex.



An adjacency matrix is a matrix of size $n \times n$ whose non-diagonal element a_{ij} is the number of edges linking vertex i to vertex j . The diagonal element a_{ii} is the number of loops at vertex i . While this representation is expensive for a sparse graph (the adjacency list is preferred for this type of graph), it proves to be very profitable for a dense graph. An example is provided below.



If the graph is **weighted**, you can replace the Booleans with numbers corresponding to the weighting of each edge. And then you set a special value to indicate the absence of an edge (-1 , 0 or ∞ for example, depending on the case). An example is provided below.



- Euler's theorem**

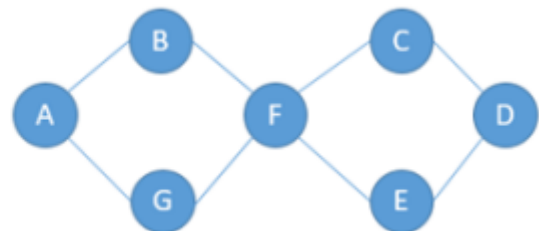
An **Eulerian path** (resp. a **directed Eulerian path**) is a path (resp. a trail) that contains all the edges of an undirected (resp. directed) graph exactly once. When the starting point is also the ending point of the path (resp. of the trail), this is known as an **Eulerian cycle** (resp. an **Eulerian circuit**). When an undirected (resp. directed) graph has an Eulerian path (resp. a directed Eulerian path), this is known as an **Eulerian graph**. An Eulerian graph has a cycle that passes exactly once through all the edges of a graph.

There is an Eulerian cycle (resp. an **Eulerian circuit**) in an undirected (resp. directed) graph if, and only if, the graph is connected and there is no vertex of odd degree in the graph.

There is an Eulerian path (resp. a **directed Eulerian path**) between i and j if in an undirected (resp. directed) graph if, and only if, the graph is connected and i and j are the only vertices of odd degree in the graph.

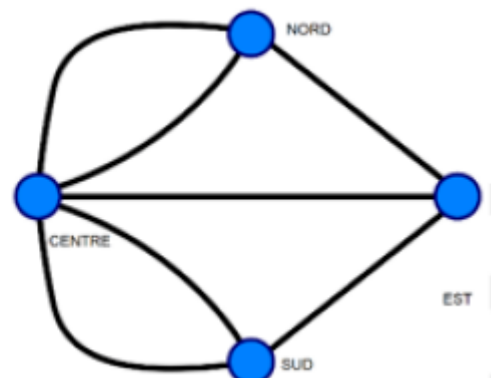
For example, the graph below is connected since all vertices are mutually reachable. And thanks to the following table and Euler's theorem, we can deduce that this graph is also Eulerian since all vertices are of even degree, so there is indeed an Eulerian cycle:

Vertex	A	B	C	D	E	F	G
Degree	2	2	2	2	2	4	2



Here's another example using the **7 bridges of Königsberg** (reminder of the graph below). Thanks to the following table and Euler's theorem, we can deduce **that there is no Eulerian cycle** (there is actually at least one vertex of odd degree) and therefore this problem has no solution:

Vertex	North	Center	East	South
Degree	3	5	3	3



- Chinese Postman Problem**

The Chinese Postman Problem is called like this because it is a problem that all postmen know, but which was mathematically studied by a Chinese, Mei-Ko Kwan, in 1962. The aim is to **find the shortest cycle (circuit) that allows a postman to cover all the streets in his area at least once**. Evidently, if there is an Eulerian cycle allowing to pass through each street exactly once, this cycle (circuit) would be at the same time the solution of the Chinese postman. But when this is not the case, there is no other way but to cover certain streets several times.

1. We return an Eulerian cycle if there is 1.
2. Otherwise, we double enough edges in the graph to make it as Eulerian as possible.

- **Hamiltonian Cycle, Hamiltonian Path and Hamiltonian Graph**

Although this is a **false lead** for solving the prosit issue, it is important that students fully understand the difference between Hamiltonian cycle and Eulerian cycle. The definitions are identical to those of Eulerian cycle, path and graph, except here we are looking for a way to **pass through all the vertices of the graph exactly once** (and not the edges as is the case with the 'Eulerian' definitions).

Surprisingly, even though we know a simple characterisation of graphs containing an Eulerian cycle - Euler's theorem -, **this is not the case for Hamiltonian cycles!**

Here we do not discuss the difference in terms of complexity, as this will be addressed in the next prosit.

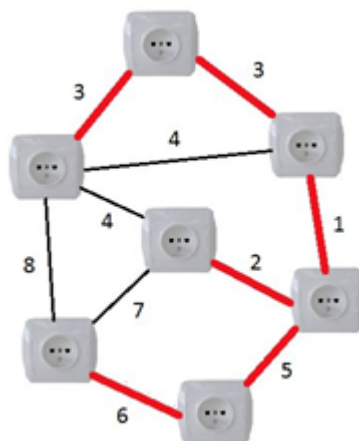
- **Minimum Spanning Tree (MST) Problem**

Although the MST is also a **false lead** for solving the prosit issue, it seems important to explain this problem, so that students fully understand what the MST Problem is and how to solve it.

A **tree** is a connected graph without a cycle. For example, an elementary (or simple) path is a tree. **The spanning tree** of a graph G is a partial connected graph of G without any cycles (it is a graph with less edges that still connects all vertices together).

The **Minimum Weight Spanning Tree (MST)** Problem consists in determining, for a simple connected graph,

a spanning tree whose edge weight is minimum. When the edges are not weighted, it is simply a matter of finding a spanning tree. In the following example, shown above, the red edges form the MST of the basic graph (red edges + black edges).



There are several algorithms for **solving the MST Problem** (Kruska, Prim and Borůvka).

- **Complexity of an algorithm**

Number of elementary operations performed by the algorithm. The complexity of an algorithm is expressed, not as an absolute value, but as a function with the amount of data that the algorithm processes (the instance length of the problem). An algorithm for finding a value in a list will take longer if there is more data to search through. When we study the speed at which an algorithm converges, this must be done according to the amount of data it has to process.

Advanced Algorithms Preparation Project 1 - PBL Loop 2 - PBL Loop 3 - PBL Loop 4 - PBL Loop

To simplify the study of the complexity of an algorithm, we consider the worst case (the one that is particularly unfavourable to the algorithm). There are also medium complexity demonstrations, but it is extremely hard to demonstrate. In the best-case scenario, complexities are not so interesting.

Moreover, we do not consider the exact complexity to the nearest instruction, we reason in order of magnitude to make reasoning easier. This translates into:

- Eliminating constants: $O(2n)$ is replaced by $O(n)$, and $O(n(n-1))$ by $O(n^2)$
- Considering only the dominant term (the one that has the most impact on the computation time growth rate according to the amount of data): $O(n^2+n)$ is replaced by $O(n^2)$, and $O(n+3)$ by $O(n)$

The formal definition consists in considering a function that asymptotically bounds the complexity to the nearest constant: The function $O(f(n))$ is asymptotically bounded by the function $f(n)$ multiplied by an arbitrary constant C . More specifically, this translates into considering the method shown above.

For example, an algorithm processing a square matrix of length n :

```
For i from 1 to n do
```

```
  For j from 1 to n do
```

```
    Matrix[i][j] = random number
```

will have a complexity in $O(n^2)$, since we do $n \times n$ processing tasks, where each processing task is done in constant time.

By following the same method, it is also possible to analyse the amount of memory space it uses. This is known as space (or spatial) complexity, as opposed to the time (or temporal) complexity we have discussed so far. This notion is not included in the goals, but it is often mentioned in the resources provided.

- **Asymptotic notation, Landau notation:**

Notation used to measure and compare the order of functions. It is used here on the algorithmic complexity functions of the algorithms studied. Formally, a function f is in $O(g)$ (pronounced as 'big O of g' or 'big omicron of g') if it is bounded above by the product of g and a constant from a certain rank N :

Intuitively, any function f in $O(g)$ is bounded by the function $g(n)$:

Asymptotically, i.e. for any sufficiently large n (the $\forall n \geq n_0$ part in the formula above). To the nearest constant c

We then derive that f and g are of the same order of magnitude (again, this notion of amount of data will be formalised further below). For the complexity measurement, we look for the 'smallest and simplest' g such that f is in $O(g)$.

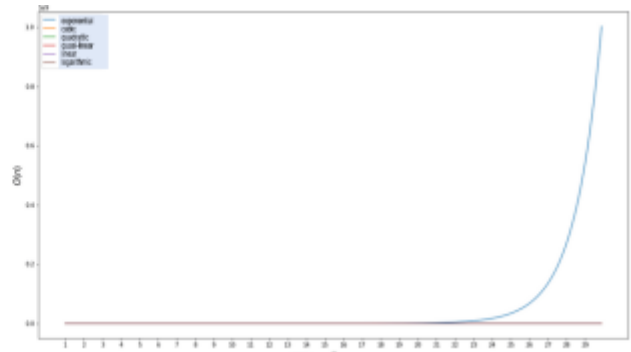
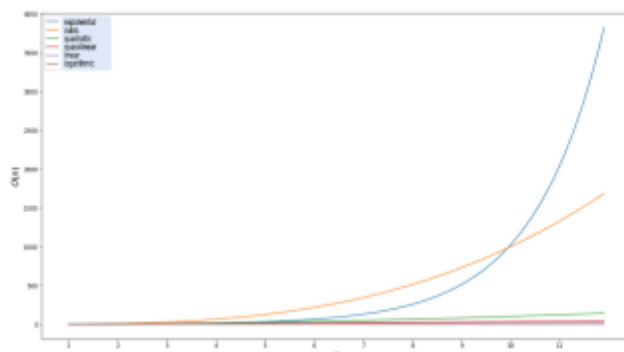
However, it is worth noting that here this notion refers to a worst-case complexity. When demonstrating the complexity of an algorithm, this implies imagining the worst situation in which the algorithm might end up (for example, for bubble sort, it is a list where the smallest element is at the end).

- **Complexity class:**

Advanced Algorithms Preparation Project 1 - PBL Loop 2 - PBL Loop 3 - PBL Loop 4 - PBL Loop

The main complexity classes, by ascending order of computation time, are:

O	Complexity class	
$O(1)$	Constant	considered reasonable
$O(\log(n))$	Logarithmic	considered reasonable
$O(n)$	Linear	considered reasonable
$O(n \log(n))$	Quasi-linear	considered reasonable
$O(n^2)$	Quadratic	considered reasonable
$O(n^3)$	Cubic	considered reasonable
$O(n^k)$	Polynomial ($k > 3$)	considered reasonable
$O(2^n)$	Exponential	
$O(n!)$	Factorial	



In general, the term polynomial complexity is used to refer to complexities that are, at most, polynomial.

An exponential complexity problem will be less likely to be processed efficiently (the notion of efficiency depends on the type of problem, decision or optimization), except for very small instances, **and** provided a substantial computing capacity is available. Quasi-linear and linear complexities are very efficient. Quadratic and cubic complexities represent problems that one may expect to process, but which tend to use up processing power (especially if the asymptotic constant is large). Typically, these complexities may or may not be accepted, depending on the context. Obviously, a cubic complexity algorithm could not be used in a real-time environment (with highly demanding reactivity needs), especially with an embedded architecture, which is usually characterised by its poor computing capabilities. For example, this is why GPS software in smartphones often uses a server connection for route calculation, which is a quadratic problem nevertheless (Dijkstra's algorithm). For polynomials of order greater than 3, even if from a theoretical point of view, we remain in the P class (see further below), because this is actually a complexity that is deemed incompatible with an implementation.

Validation questions

- What is the difference between an Eulerian Cycle and a Hamiltonian Cycle?

Euler -> we try to find a cycle that passes exactly once through all the edges of the graph. Hamilton -> same as above but through all the vertices.

- Which examples of specific problems are based on the search for an Eulerian/Hamiltonian cycle?

Euler -> postman problem, fibre optic installation teams when installing cables in the streets of a city...

Hamilton -> The travelling salesman who has to pass exactly once through all the cities he needs to visit and then return to his starting point...

- Can we solve the 7 bridges of Königsberg problem by adding one or more bridges?

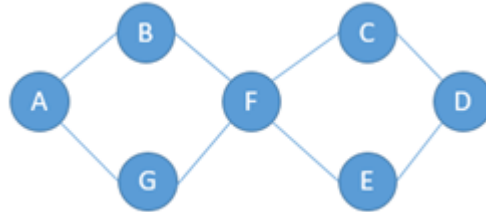
Yes, we add 2 bridges - one between North and Centre and another between South and East, for example.

See the graph in the problem solution.

- Adding bridges... doesn't that remind you of something?

- Is it possible for a graph to have an Eulerian cycle but not a Hamiltonian cycle?

Yes:



- In which cases should we choose an adjacency list or an adjacency matrix?

Dense graph → matrix.

Sparse graph → list.

- What is the complexity of an algorithm that searches if a value is present in an array?

$O(n)$, since the value is not present in the worst-case scenario, and so the algorithm must browse the entire array

- What is the complexity of an algorithm that searches if a value is present in an array with an index based on a B-TREE (indexing method used by Oracle and discussed in the previous course)?

$O(\log(n))$, since the value is not present in the worst-case scenario. However, the B-TREE index is a self-balancing binary search tree. In this worst case, the algorithm will have to work its way down from the root to a leaf. However, the depth of a balanced binary tree with n vertices is $\log(n)$ (intuitively, we understand this by considering that the logarithm with base 2 is the inverse function of the power of 2, and that for a tree of depth d , we have 2^d vertices)

- If an algorithm has a complexity in $O(2^n)$, does that mean that only very small instances can be processed?

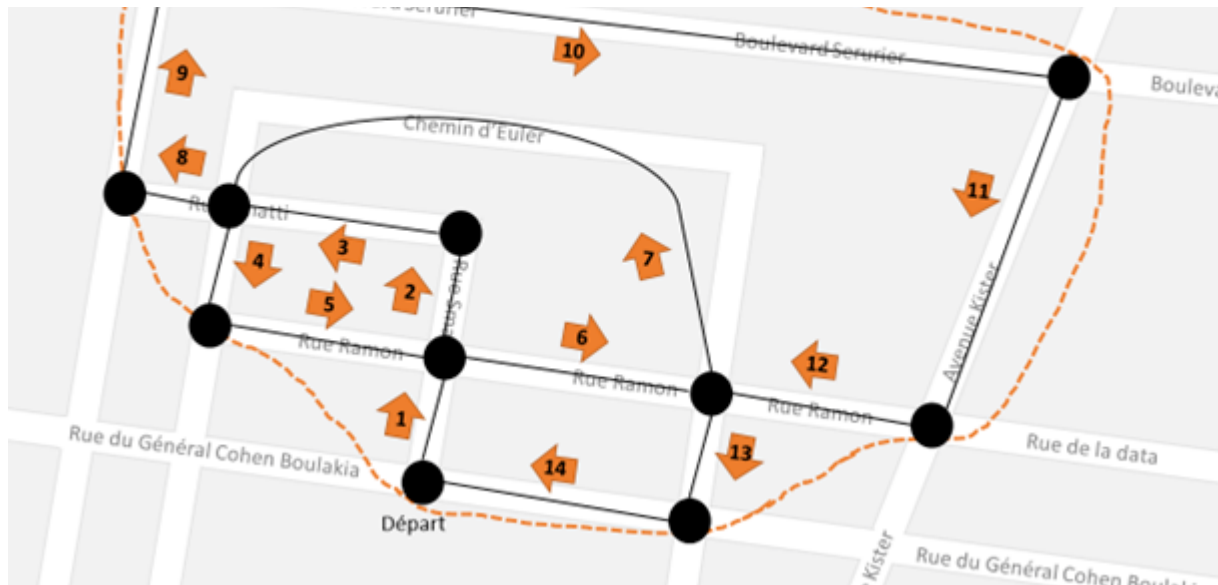
No, it is a worst-case complexity, which means that among all the instances of length n , there is at least one for which the convergence time is in $O(2^n)$. The average-case complexity may be much better.

- If we play with the Jupyter Notebook file 'Complexités asymptotiques.ipynb' provided as a prosit resource, what can we observe regarding the different complexities? (see the screenshots under Complexity class)

As soon as n becomes large enough, all complexity classes flatten out, and become insignificant due to the exponential complexity.

Possible solutions

- Agathe's Map turned into a graph**



Each crossroad is a vertex of the graph and each street in the intervention area is an edge of the graph.

We observe in this graph that all vertices are of even degree. **Therefore, there is an Eulerian cycle going from the starting vertex!** The arrows on the diagram show a possible itinerary.

- **Overall operation of the program:**

This program, which was developed during the Workshop, is used to find an Eulerian cycle when the graph is Eulerian. We rely on an Eulerian cycle search algorithm based on the **Backtracking** technique (in-depth browsing of the graph). The asymptotic complexity of this algorithm is $O(|V| \times |E|)$. As for the algorithm checking whether a graph is Eulerian, it is of complexity $O(|V|)$. Therefore, the entire program (cycle verification and calculation) has a complexity $O(|V| \times |E|) + O(|V|) = O(2|V| \times |E|)$, i.e. $O(|V| \times |E|)$ because here we are calculating the asymptotic complexity, which implies that constants are eliminated.

If the graph is not Eulerian, the program does not propose any itinerary. An upgrade would be to implement a Chinese postman algorithm in the program (i.e. an initial step that determines whether the graph is Eulerian, and if not, completes it. See definition of the Chinese postman algorithm).

- **Solution of the 7 bridges of Königsberg problem**

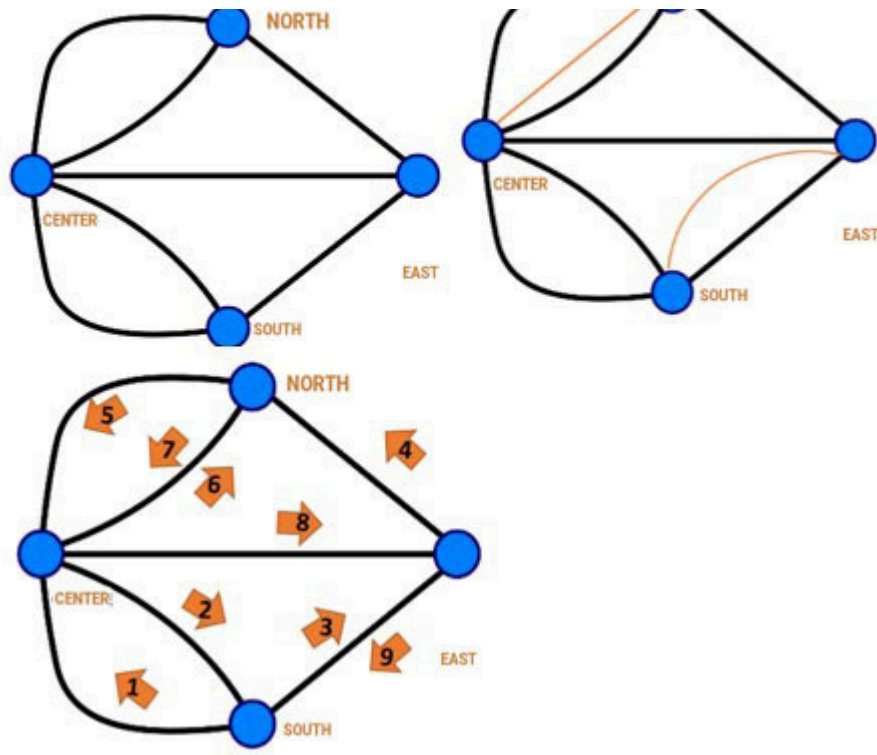
Euler's theorem shows, on the graph modeling the problem (below, on the left), that it is **impossible to find an Eulerian cycle** and therefore a walking route that would make us pass through all the bridges exactly once. **The Chinese postman algorithm** will therefore allow us to find a walking route that passes through all the bridges at least once and through some of them twice (minimum). The graph below on the right proposes a possible solution:

- We add one bridge between the North and Center areas and another one between the East and South areas (graph below, in the middle).

- All vertices are now of even degree, so there is an Eulerian cycle.

This is equivalent to saying (this is how the Chinese postman algorithm works) that there is an itinerary (see graph below, on the right) that would make me start from an area, and then makes me cross all bridges once, except for the bridge between the North and Center areas, which I cross twice (the same goes for the bridge between the East and South areas).

Advanced Algorithms Preparation Project 1 - PBL Loop 2 - PBL Loop 3 - PBL Loop 4 - PBL Loop



WORKSHOP

TOPIC

Instructions

QUIZ

Workshop Statement [ipynb] ↕

Note: The file must be unzipped and then opened using Jupyter

EDUCATIONAL GUIDE

Standard corrections

Tutor version workshop [ipynb] ↕

Note: The file must be unzipped and then opened using Jupyter

