A Town, a Network, the Moon, a Train and a Mouse

1 Connectivity (Connexité)

Exercise 1.1 (Algernon and the Labyrinth)

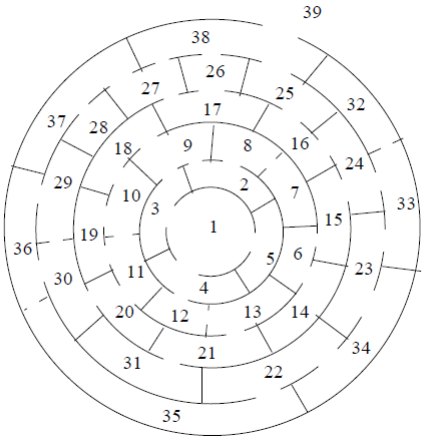


Figure 1: One of Algernon's labyrinth

Algernon, little white laboratory mouse, learns how to find his way out of labyrinths.

1. Algernon is placed at the center of the labyrinth. Will he be able to go out? Does he have several solutions?
2. For the next tries, Algernon is placed at a new place each time. Will he be able to go out every time?
3. Algernon is smart. He managed to find a path out from the center of a new labyrinth and remembers it. Researchers close all the doors the mouse goes through. Will he be able to find his way out again?

Exercise 1.2 (Network and Routers)

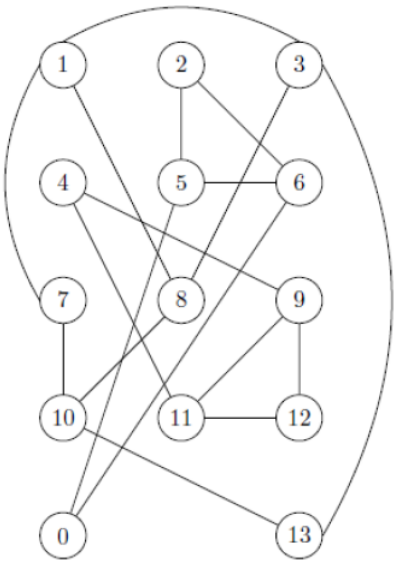


Figure 2: Router network

The Internet Service Provider (ISP) *KrisNet* has to set its router network up. Each router is in an interconnection point in order to have access to the other ISP routers. The ISP wishes to establish its own network (traffic passing through the network of another ISP being charged) with point to point connections. *KrisNet* management gets the solution given in figure 2.

1. The network architect has to verify that the proposed connections actually allow all routers to be connected to each other.
   (a) Which property of graph theory must this network respect?
   (b) Recall all definitions related to this property (chain, elementary chain...)
2. Connected graph (Graphe connexe):
   (a) How to verify that a graph has the property in question 1.
   (b) Apply to *KrisNet* network. Does this network allow all routers to be connected to each others?
3. Connected components (Composantes connexes):
   (a) Recall the definition of a connected component.
   (b) Modify the principle of the previous algorithm in order to calculate the number of connected components of a graph.

Exercise 1.3 (Union is strength)

KrisNet does not know the graph: the only informations he have are the routers (numbered) and the list of the existing connections.

1. How to know if two routers are interconnected?
2. Write a function that lists the connections to add to allow all routers to be connected to each other (without building a graph).

Exercise 1.4 (Journey To The Moon – *Midterm 2017*)

The member states of the UN are planning to send 2 people to the Moon. In line with their principles of global unity, they want to pair astronauts of different countries.

There are N trained astronauts numbered from 0 to $N-1$. But those in charge of the mission did not receive information about the citizenship of each astronaut. The only information they have is that some particular pairs of astronauts belong to the same country.

Your task is to compute in how many ways they can pick a pair of astronauts belonging to different countries. Assume that you are provided enough pairs to let you identify the groups of astronauts even though you might not know their country directly. For instance, if 1, 2, 3 are astronauts from the same country; it is sufficient to mention that (1, 2) and (2, 3) are pairs of astronauts from the same country without providing information about a third pair (1, 3).

L is a list of pairs (A, B) such that A and B, both in $[0, N-1]$, are astronauts from the same country.

Write the function `Moon(`N`, `L`)` that computes in how many different pairs of astronauts can be picked.

Exercise 1.5 (Union-Find – *Midterm 2017*)

Let G be the graph $< S, A >$, n its number of vertices and L its list of edges (a list of vertex pairs). Write the function `CCFromEdges(`n`, `L`)` that returns the pair (k, cc): k is the number of connected components of G, and cc its connected component vector (a list in Python).

Exercise 1.6 (Warshall – *Midterm 2017*)

Reminder:

- We call *transitive closure* of a graph G defined by the couple $< S, A >$, the graph G^* defined by the couple $< S, A* >$ such that for all pairs of vertices $x, y \in S$, there exists an edge $\{x, y\}$ in G^* if-and-only-if there exists a chain from x to y in G.
- Warshall algorithm computes the adjacency matrix of the transitive closure of a graph.

Given M, the result matrix of Warshall applied to the graph G, write the function `CCFromWarshall(`M`)` that returns the pair (k, cc): k is the number of connected components of G, and cc its connected component vector (a list in Python).

Exercise 1.7 (Minimize the Connections)

Renting point to point connections is expensive. The accountant asks the network architect to find a solution that ensures that all routers are connected, but using the minimal possible connections.

1. How can one satisfy the accountant: how to get a network where all the routers are connected using a minimum of links?
2. Such a graph is a *tree*.
   (a) What happens when any edge is added to a *tree*?
   (b) What if any edge is removed?
   (c) Give the three properties of a graph that is a *tree*?
3. Algorithm:
   (a) Is it necessary to test the three properties of the previous question to determine whether a graph is a *tree*?
   (b) Deduce the different methods to test if a network is connected with a minimum of links?

Exercise 1.8 (I want to be a tree – *Midterm 2016*)

The aim of this exercise is to turn any graph (represented by an adjacency matrix) into a *tree* with a minimum of modifications.

Short version: Write a function that

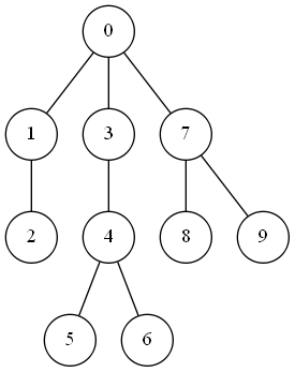
- builds the connected component vector *cc* of the original graph ;
- adds the edges to the graph to make it connected ;
- removes "useless" edges from the graph (without increasing the connected component number).

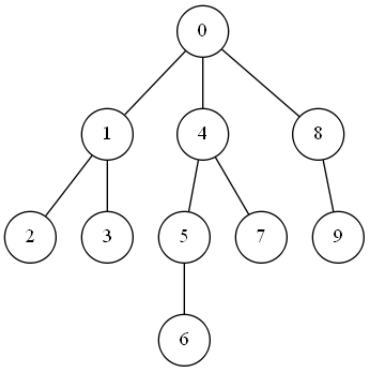
Exercise 1.9 (Even Tree – *Midterm 2017*)

You are given a *tree* (a simple connected graph with no cycles). The aim here is to find the **maximum** number of edges you can remove from the tree to get a forest such that each connected component of the forest contains an **even** number of vertices.

Note: The given graph is such that it can always be decomposed into components containing an even number of vertices.

Wich edges can be removed in the following trees?





In both cases, the result is a forest with connected components of even size.

Note: it works whatever the starting vertex is.

Write the function `evenTree(`G`)` that computes the maximum number of edges that can be removed from G according to the previous specifications.

---

2 Strong Connectivity (Forte connexité)

Exercise 2.1 (Γ^+, Γ^-)

Consider the following algorithm:

```
mark ⊕ and ⊖ some vertex x of the digraph
while feasible do
    mark ⊕ every successor of a vertex marked ⊕
    mark ⊖ every predecessor of a vertex marked ⊖
end while
```

1. What is the complexity of this algorithm?

2. What does the set of vertices marked both $\oplus$ and $\ominus$ represent?

3. How this algorithm can be used to determine if a digraph is strongly connected? Or, if it is not, to determine its strongly connected components?

Exercise 2.2 (One-way Traffic)

Mister Road has the job of installing one-ways in a whole neighborhood so that traffic is more fluid (the streets are too narrow, parking and traffic are problematic). Below is the map with the traffic direction of each road. Mr. Road wonders if we can really move around the whole area without leaving it.

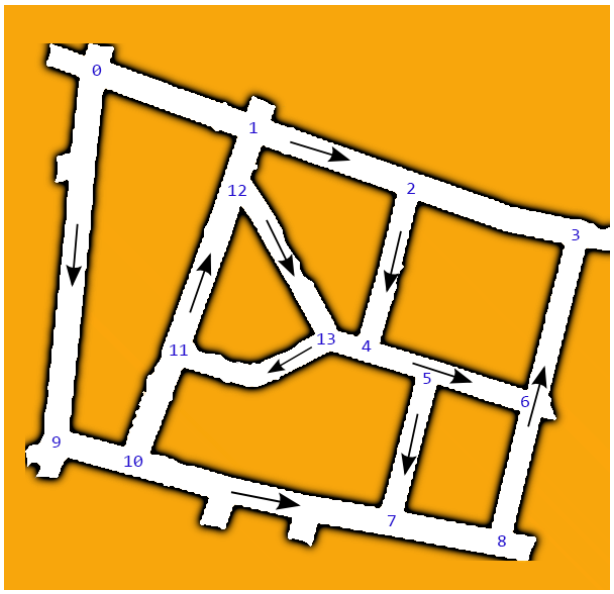


Figure 3: Neighborhood Map

1. Strongly connected graph (Graphe fortement connexe)
   (a) Represent the neighborhood map as a graph.
   (b) Is the map of mister Road strongly connected?
   (c) Which roads have to become two-ways?

2. **First method:** double traversal
   (a) What do we observe when we consider the inverse digraph and the property that the initial digraph should respect?
   (b) Give the principle of building the strongly connected components using two traversals and write the corresponding function.
   (c) What if we only want to test the strong connectivity of the digraph?

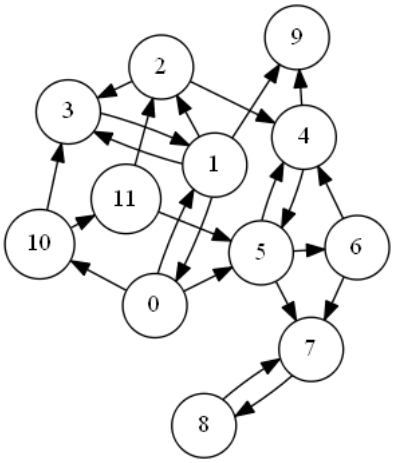


Figure 4: Another map

3. **Second Method:** Tarjan
   (a) Apply the Tarjan's algorithm to the graph in figure 4 to find its components.
   (b) Write the function that builds the strongly connected components of a digraph.
   (c) What if we only want to test the strong connectivity of the graph?

Exercise 2.3 (Toy Train)

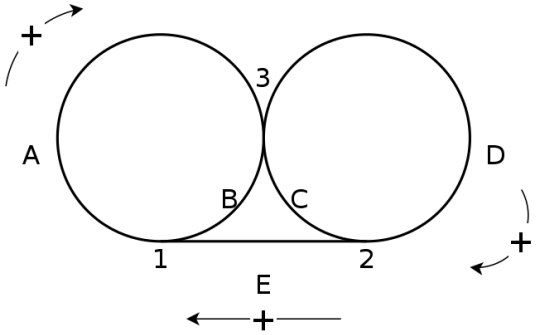


Figure 5: Toy Train Railway

The figure 5 represents a toy train railway. The train can move either forward or backward. The circuit is divided into 5 sections (A, B, C, D and E) that can be taken in both ways (+ and −). It also has 3 points (switches).

Show that if the direction of the train is constant then, from a certain time, the train can no longer take the section E.

Exercise 2.4 (Condensation – *Final 2017*)

Let G be a digraph with k strongly connected components: C_0, C_1, ..., C_{k-1}. The *condensation* of G is the digraph $G_R = < S_R, A_R >$ defined by:

- $S_R = \{C_0, C_1, \cdots, C_{k-1}\}$
- $C_i \rightarrow C_j \in A_R \Leftrightarrow$ There exists at least an edge in G with its head in the strongly connected component C_i and its tail in the strongly connected component C_j.

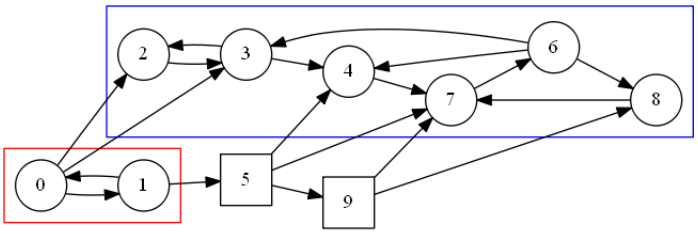


Figure 6: Digraph G_1

The aim here is to build the condensation of a graph G. We have the list of the strongly connected components of G (each component is a list of the vertices it contains).

For instance, the following list is the component list of the graph in figure 2.4:

```
scc = [[2, 3, 4, 6, 7, 8], [9], [5], [0, 1]]
```

Write the function `condensation(`G`, scc)` that builds the condensation G_R of a digraph G, with *scc* its component list. The function returns G_r and the vector of components: a vector that give for each vertex, the number of its component (the vertex in G_R).

For instance, with G_1 the graph in figure 2.4:

```
>>> (Gr, comp) = condensation(G1, scc)
>>> comp
[3, 3, 0, 0, 0, 2, 0, 0, 0, 1]
```

3 Finally

Exercise 3.1 (Failure resistance)

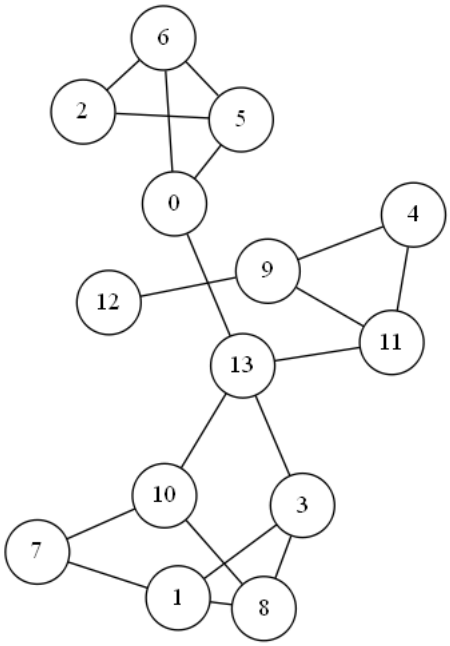


Figure 7: Network improvement

Even though it is important to minimize the number of connections, our ISP must have a network that resists failures. Indeed, if one router is cut off from another, it will have to take an alternative route via the external connections, and the traffic on these connections (transit traffic) has a relatively high cost. We are therefore interested in the resistance to failures, these can be of two kinds: failure of a router and failure of a connection.

1. **Router failure:**

   (a) What issue occurs with a network that uses a minimum of links when some routers break down? What are those routers?

   (b) In the graph that represents the network, what are those vertices called?

   (c) Show that if one of these vertices is root of a spanning tree, it has at least 2 children.

   (d) Let v be one one these vertices, that is not root of the spanning tree. Show that v has at least one child s such that there is no back edge starting from s or from a descendant of s that ends to an ancestor of v..

   (e) What information has to be associated with every vertices to represent the property of the previous question? How this information is computed?

   (f) Use the answer of the three last questions to give the principle of an algorithm that detects these vertices.

   (g) Apply it to the graph in figure 7.

   (h) Write the corresponding function.

2. **Link failure:**

   (a) What issue occurs with a network that uses a minimum of links when some links break down? In graph theory, how are those links called?

   (b) Show that these kind of links do not belong to a cycle in the graph?

   (c) How to identify these kind of links?

   (d) Do such links exist in the graph in figure 7?

   (e) Add the detection of these edges to the principle of the question 1f (and to the function).

3. **Secure sub-network:**

   (a) How are the subgraphs without vertices studied in question 1b called?

   (b) How to identify these subgraphs using the results of the previous questions?

   (c) Give the subgraphs with the property of question 3a in the graph in figure 7.

   (d) Modify the principle of question 2e so that it determines these subgraphs.

Exercise 3.2 (Algernon and the labyrinth (rest))

The new Algernon's labyrinth has now one-way doors.

1. The labyrinth is large and has circuits. How to know what are the different impossible paths, without traverse the whole graph each time? How to detect the "dead-ends"?

2. Algernon begins to weaken. Researchers take the initial labyrinth (without doors) to install new doors. But they want to make sure that Algernon can go everywhere without being blocked.
   - How to know which doors have to open in both ways?
   - How to know in which way should the remaining doors open?