Graphs and Networks Đồ thị và Mạng lưới

Giới thiệu

> id: intro-0
> section: introduction
> description: Discover the mathematical principles that connect our world – from shaking hands to travel and navigation, colouring maps and social networks.
Cùng khám phá các nguyên lý toán học kết nối thế giới của chúng ta - từ những cái bắt tay đến những cuộc du ngoạn, khám phá và mạng lưới xã hội.
> color: "#A7208A"
> level: Intermediate
> next: probability

Every day we are surrounded by countless connections and networks: roads and rail tracks, phone lines, the internet, electronic circuits and even molecular bonds. There are even _social networks_between friends and families. Can you think of any other examples?
Mỗi ngày, chúng ta đều được bao quanh bởi vô số những kết nối và mạng lưới: phố sá, đường ray, đường dây điện thoại, mạng Internet, mạch điện tử và cả những liên kết hóa học. Thậm chí có cả _mạng xã hội_ để kết nối với bạn bè và gia đình. Bạn còn nghĩ ra được ví dụ nào nữa không?

::: column(width=220 parent="padded-thin")

x-img(src="images/network1.jpg" width=220 height=220 lightbox)

{.caption} Road and Rail Networks
Mạng lưới đường sắt và đường bộ

::: column(width=220)

x-img(src="images/network6.jpg" width=220 height=220 lightbox)

{.caption} Computer Chips
Vi mạch máy tính

::: column(width=220)

x-img(src="images/network3.jpg" width=220 height=220 lightbox)

{.caption} Supply Chains
Chuỗi cung ứng

::: column(width=220)

x-img(src="images/network2.jpg" width=220 height=220 lightbox)

{.caption} Friendships
Tình bạn

::: column(width=220)

x-img(src="images/network7.jpg" width=220 height=220 lightbox)

{.caption} Neural Connections
Mạch nơron thần kinh

::: column(width=220)

x-img(src="images/network4.jpg" width=220 height=220 lightbox)

{.caption} The Internet
Mạng Internet

:::

> id: intro

::: column.grow

In mathematics, all these examples can be represented as [__graphs__](gloss:graph) (not to be confused with the _graph_ of a function). A graph consists of certain _points_ called [[vertices|circles|crossings]], some of which are connected by [[edges|boundaries|pairs]].
Trong toán học, tất cả những ví dụ này đều có thể được miêu tả bằng [__đồ thị__](chú giải:đồ thị] (tránh nhầm lẫn với đồ thị hàm số). Một đồ thị bao gồm những _điểm_ nhất định được gọi là [[đỉnh|vòng tròn|giao điểm]], một số được kết nối với nhau bởi [[cạnh|đường biên|cặp ghép đôi]].

__Graph theory__ is the study of graphs and their properties. It is one of the most exciting and visual areas of mathematics, and has countless important applications.
__Lý thuyết đồ thị__ là môn học nghiên cứu đồ thị và tính chất của chúng. Đây là một trong những lĩnh vực thú vị, trực quan nhất trong toán học, và có vô số ứng dụng quan trọng.

::: column(width=180)

svg#graph0.graph.novertices.noedges(width=180 height=180)

:::

> id: intro-1

We can draw the layout of simple graphs using circles and lines. The position of the vertices and the length of the edges is irrelevant – we only care about _how they are connected_ to each other. The edges can even cross each other, and don't have to be straight.
Chúng ta có thể vẽ ra bố cục của những đồ thị đơn giản với các đường tròn và đường thẳng. Vị trí của các đỉnh và độ dài của các cạnh không quan trọng - chúng ta chỉ cần để ý _cách chúng kết nối với nhau_. Các cạnh thậm chí có thể giao nhau, và không nhất thiết phải thẳng.

::: column(width=200)

    svg.graph(height=120 width=200 style="margin: 0 auto .8em")

{.caption} In some graphs, the edges only go one way. These are called [__directed graphs__](gloss:directed-graph).
Trong một số đồ thị, các cạnh chỉ có một hướng. Chúng được gọi là [__đồ thị có hướng__](chú giải: đồ thị có hướng)

::: column(width=200)

    svg.graph(height=120 width=200 style="margin: 0 auto .8em")

{.caption} Some graphs consist of multiple groups of vertices which are not connected with each other by edges. These graphs are __disconnected__.
Một số đồ thị bao gồm nhiều nhóm các đỉnh không kết nối với nhau bởi các cạnh. Những đồ thị này __không liên thông__.

::: column(width=200)

    svg.graph(height=120 width=200 style="margin: 0 auto .8em")

{.caption} Other graphs may contain multiple edges between the same pairs of vertices, or vertices which are connected to themselves (loops).
Những đồ thị khác có thể chứa nhiều cạnh giữa một cặp đỉnh nhất định, hoặc là các đỉnh này nối với chính nó (vòng lặp)

:::

    // TODO maybe include examples of graphs with edges crossing, curved edges, etc.
    // could include an "is this a graph?" quiz

> id: intro-2

We can create new graphs from an existing graph by removing some of the vertices and edges. The result is called a [__subgraph__](gloss:subgraph). Here you can see a few more examples of graphs, with coloured edges and vertices indicating a possible subgraph:
Chúng ta có thể tạo ra những đồ thị mới từ một đồ thị cho trước bằng cách bỏ đi một số cạnh và đỉnh. Kết quả tạo ra một [__đồ thị con__](chú giải:đồ thị con). Ở đây bạn có thể thấy một vài ví dụ của đồ thị, với những cạnh và đỉnh có màu là biểu thị cho một đồ thị con.

::: column(width=212 parent="padded-thin")

    svg.graph(height=100 width=100 style='float: left; margin-right: 12px')
    svg.graph(height=100 width=100 style='float: left')

::: column(width=212)

    svg.graph(height=100 width=100 style='float: left; margin-right: 12px')
    svg.graph(height=100 width=100 style='float: left')

::: column(width=212)

    svg.graph(height=100 width=100 style='float: left; margin-right: 12px')
    svg.graph(height=100 width=100 style='float: left')

:::

> id: intro-3

We say that the [__order__](gloss:graph-order) of a graph is the number of vertices it has. The [__degree__](gloss:graph-degree) of a vertex is the number of edges which meet at that vertex.
[__Cấp__](chú giải:Cấp-đồ thị) của một đồ thị là số đỉnh mà đồ thị đó có. [__Bậc__](chú giải:Bậc-đồ thị) của một đỉnh là số cạnh kết nối với nó.

::: column(width=130)

    svg.graph(height=120 width=120 style='margin: 0 auto .8em')

{.text-center} Order: [[5]]

::: column(width=130)

    svg.graph(height=120 width=120 style='margin: 0 auto .8em')

{.text-center} Order: [[8]]

::: column(width=130)

  svg.graph(height=120 width=120 style='margin: 0 auto .8em')

{.text-center} Degree: [[3]]

::: column(width=130)

  svg.graph(height=120 width=120 style='margin: 0 auto .8em')

{.text-center} Degree: [[6]]

:::

> id: intro-4

Graphs that consist of a single loop of vertices are called [__cycles__](gloss:graph-cycle). All cycles have [[the same number of edges and vertices|more edges than vertices|fewer edges than vertices]].
Một đồ thị mà bao gồm một vòng lặp các đỉnh được gọi là [__Đồ thị chu trình__](chú giải:Đồ thị chu trình). Tất cả đồ thị chu trình đều có [[số lượng đỉnh và cạnh bằng nhau|số cạnh nhiều hơn số đỉnh|số cạnh ít hơn số đỉnh]].

  .row
    svg.graph(style='width: 120px; height: 120px;')
    svg.graph(style='width: 120px; height: 120px;')
    svg.graph(style='width: 120px; height: 120px;')

{.reveal(when="blank-0")} Equipped with these new definitions, let's explore some of the fascinating properties and applications of graphs.
Giờ thì bạn đã hiểu về những khái niệm này rồi, hãy khám phá thêm những tính chất và ứng dụng của đồ thị nào.

> id: bridges-0
> title: The Bridges of Königsberg Bài toán bảy cây cầu Königsberg
> section: bridges

The Bridges of Königsberg

::: column.grow

One of the first mathematicians to think about graphs and networks was [Leonhard Euler](bio:euler). Euler was intrigued by an old problem regarding the town of Königsberg near the Baltic Sea.
Một trong những nhà toán học đầu tiên nghĩ đến đồ thị và mạng lưới là [Leonhard Euler](tiểu sử:euler). Euler đã được thúc đẩy bởi một vấn đề liên quan đến thành phố Königsberg gần biển Baltic

The river Pregel divides Königsberg into four separate parts, which are connected by seven bridges. Is it possible to walk around the city crossing all of the bridges exactly once – but not more than once? (You can start and finish anywhere, not necessarily in the same place.)
Con sông Pregel chia Königsberg thành 4 khu vực tách biệt và chúng được nối với nhau bởi bảy cây cầu. Liệu chúng ta có thể đi vòng quanh thành phố, qua cả bảy cây cầu, mỗi cây chỉ được đi qua duy nhất một lần hay không? (Bạn có thể xuất phát và kết thúc ở bất cứ đâu, không nhất thiết phải là cùng một địa điểm).

Try to find a valid route by drawing on these maps:
Hãy tìm một tuyến đường thỏa mãn các điều kiện trên bản đồ dưới đây:

::: column(width=250)

  img.shifted(src="images/konigsberg1.jpg" width=250 height=350)

:::

> id: bridges
> goals: bridge-0 bridge-1 bridge-2 bridge-3
> title: The Bridges of Königsberg

  x-tabbox.full-width
    .tab
      h3 Map 1#[span.check.incorrect(when="bridge-0")]
      x-solved
      include svg/bridges-1.svg
      button.btn Clear
      button.btn.right Skip
    .tab
      h3 Map 2#[span.check(when="bridge-1")]
      x-solved
      include svg/bridges-2.svg

```
    button.btn Clear
    button.btn.right Skip
  .tab
    h3 Map 3#[span.check(when="bridge-2")]
    x-solved
    include svg/bridges-3.svg
    button.btn Clear
    button.btn.right Skip
  .tab
    h3 Map 4 #[span.check.incorrect(when="bridge-3")]
    x-solved
    include svg/bridges-4.svg
    button.btn Clear
    button.btn.right Skip
```

> id: bridges-1

In the case of Königsberg it seems to be impossible to find a valid route, but some of the other cities do work. Euler managed to find a simple rule that can be applied to any city, without having to try lots of possibilities – using graph theory.
Trong trường hợp thành phố Königsberg, tìm ra một tuyến đường phù hợp có vẻ như là bất khả thi, nhưng ở một số thành phố còn lại thì dễ dàng hơn. Euler đã tìm ra một quy tắc đơn giản có thể áp dụng lên bất cứ thành phố nào mà không cần phải thử đi thử lại nhiều lần - đó là sử dụng lý thuyết đồ thị.

::: column.grow

First, we need to convert the city maps into graphs with edges and vertices. Every island or region of land is represented by [[a vertex|an edge|an area]] and every bridge connecting two regions is represented by a corresponding [[edge|vertex|street]].
Đầu tiên, chúng ta cần chuyển những bản đồ thành phố này thành các đồ thị với các cạnh và đỉnh. Coi mỗi hòn đảo hay khu vực là một [[đỉnh|cạnh|diện tích]] và mỗi cây cầu nối giữa 2 vùng được đặt là một [[cạnh|đỉnh|đường]] tương ứng.

{.reveal(when="blank-0 blank-1")} Now the problem of “touring a city while crossing every bridge exactly once” has become a problem of “drawing a graph with one continuous stroke while tracing every edge exactly once”.
Bây giờ bài toán “đi quanh thành phố và băng qua mỗi cây cầu một lần” đã trở thành “vẽ một đồ thị bằng một nét liên tục đi qua tất cả các cạnh, mỗi cạnh một lần”.
::: column(width=200)

include svg/konigsberg.svg

:::

> id: bridges-2

On paper, come up with a few different graphs and then try to work out which ones can be drawn with a single, continuous stroke.
Trên một tờ giấy, hãy nghĩ ra một vài đồ thị khác nhau và tìm xem đồ thị nào có thể được vẽ bằng một nét liên tục.

// p Try drawing these graphs with one continuous stroke: Vẽ những đồ thị sau đây bằng một nét liên tục
// p.todo Interactive coming soon…

> id: bridges-3
> goals: size prime eo

Just like for the city maps before, we find that some graphs are possible while others are not. To help us understand why, let us label every vertex with its [degree](gloss:graph-degree). Then we can colour the vertices in different ways, and try to reveal a pattern:
Giống như những bản đồ thành phố ở trên, chúng ta có thể thấy một số trường hợp là khả thi hay không. Để giúp hiểu lý do tại sao, chúng ta hãy kí hiệu trên mỗi đỉnh [bậc](chú giải:bậc) của nó. Sau đó tô màu các đỉnh theo những cách khác nhau và hãy cố nghĩ ra một con đường phù hợp.

figure
  x-select.var.tabs(:bind="colour")
    div(value="val") Value
    div(value="size") Size
    div(value="prime") Prime Numbers
    div(value="eo") Even and Odd
  .box
    p.no-voice(style="margin: 0"): strong These graphs are possible:
    include svg/vertex-orders-1.svg
    p.no-voice(style="margin: 1em 0 0"): strong These graphs are not possible:
    include svg/vertex-orders-2.svg

> id: bridges-4

Comparing these numbers for graphs which are possible, and those which are not possible, it seems that a graph can be drawn if it [[has no more than two “odd” vertices|only has “even” vertices|has no vertices with an order larger than 4|has an odd number of vertices|has no vertices of order 3]]. This condition can be explained if we look at just a single vertex in the graph:
So sánh những con số này giữa các dạng đồ thị khả thi, và các dạng không khả thi, dường như đồ thị có thể vẽ được bằng một nét thỏa mãn các điều kiện nếu nó [[có không quá hai đỉnh ”lẻ”|chỉ có đỉnh “chẵn”|không có đỉnh với cấp lớn hơn 4|số đỉnh là một số chẵn|không có đỉnh cấp 3]]. Điều kiện này có thể giải thích bằng cách chỉ nhìn vào một đỉnh đơn trong đồ thị:

::: x-slideshow

.stage(slot="stage"): include svg/konigsberg-proof.svg

Here you can see a single, magnified vertex in a graph.
Bạn có thể thấy đây là một đỉnh phóng to của một đồ thị.

If we draw the graph, we will eventually have an edge leading towards this vertex, and then another one leading away. This makes two edges meeting at the vertex.
Nếu vẽ đồ thị, chúng ta sẽ có một cạnh hướng đến đỉnh này, và một cạnh hướng ra khỏi đỉnh. Điều này có nghĩa là hai cạnh gặp nhau tại đỉnh.

Maybe the vertex is a crossing rather than a corner. In that case there will be another edge leading towards the vertex, and another edge leading away. Now we have four edges.
Đỉnh có thể được coi là một giao điểm thay vì là một góc. Trong trường hợp này, sẽ có một cạnh khác hướng tới đỉnh, và một cạnh nữa hướng ra. Bây giờ chúng ta đã có 4 cạnh.

And in some graphs, there may even be a third pair of edges leading towards and away from the vertex. Now there are six edges.
Và trong một số đồ thị, có thể có một cặp cạnh thứ 3 nữa đi qua đỉnh. Khi đó chúng ta sẽ được 6 cạnh.

Notice that, either way, there always is an even number of edges meeting at the vertex.
Chú ý rằng bằng cách nào đi nữa, số cạnh gặp nhau tại đỉnh luôn là một số chẵn.

The only two exceptions are the vertices where the path starts, and where it ends – these two may have an odd number of edges. If the start and end point are the same, all vertices in the graph are even.
Chỉ có 2 trường hợp ngoại lệ là các đỉnh nơi đường đi bắt đầu, và kết thúc - hai đỉnh này có thể có số lẻ các cạnh. Nếu bắt đầu và kết thúc tại cùng một điểm, tất cả các đỉnh trong đồ thị đều có số chẵn.

:::

> id: bridges-5

::: column.grow(parent="right")

If you scroll back to the map of Königsberg, you will find that there are more than two islands with an odd number of bridges. Therefore, a route that crosses every bridge exactly once is indeed impossible – and this is what Leonard Euler discovered.
Nếu bạn quay lại bản đồ của thành phố Königsberg ở trên, bạn sẽ thấy có nhiều hơn 2 khu vực có số lẻ các cây cầu. Vì vậy, một tuyến đường băng qua mỗi cây cầu một lần duy nhất là bất khả thi - và đây cũng là điều Leonard Euler đã khám phá ra.

Euler's discovery may not seem particularly useful in real life, but graphs are at the foundation of many other geographic problems, such as finding directions between two locations. We will discover more of these applications later.
Phát hiện của Euler trông có vẻ như là không quan trọng trong đời thực, nhưng đồ thị chính là nền tảng của rất nhiều bài toán địa lý khác, ví dụ như tìm đường đi giữa hai địa điểm. Chúng ta sẽ dần dần khám phá những ứng dụng này.

::: column(width=240)

    x-img(lightbox width=240 height=260 src="images/prague.jpg")

:::

> id: handshakes-1
> section: handshakes

Handshakes and Dating Bắt tay và hẹn hò

::: column.grow

You have been invited to a wonderful birthday party with your friends. Including yourself and the host, there are ${hnd}{hnd|5|3,15,1} people present.
Bạn được mời đến tham dự một bữa tiệc sinh nhật hoành tráng với bạn bè. Tính cả bạn và người chủ trì bữa tiệc, có tất cả ${hnd}{hnd|5|3,15,1} người ở đó.

In the evening, as the guests get ready to leave, everyone shakes hands with everyone else. How many handshakes are there in total?
Vào buổi tối, khi bắt đầu ra về, tất cả mọi người đều bắt tay nhau. Có tất cả bao nhiêu cái bắt tay?

We can represent the handshakes using a graph: every person is [[a vertex|an edge]], and every handshake is [[an edge|a vertex]].
Bạn có thể diễn tả những cái bắt tay đó bằng một đồ thị: mỗi người là một [[đỉnh|cạnh]], và mỗi một cái bắt tay là một [[cạnh|đỉnh]].

{.reveal(when='blank-0 blank-1')} Now it is easy to count the number of edges in the graph. We find that with ${hnd} people, there are ${hnd*(hnd-1)/2} handshakes.
Bây giờ, việc đếm các cạnh trong đồ thị sẽ dễ dàng hơn. Chúng ta thấy được với ${hnd} người thì có tổng cộng ${hnd*(hnd-1)/2} cái bắt tay.

::: column.s-hide(width=240)

    img.shifted(src="images/party.jpg" width=240 height152)
    svg.graph(style='width: 240px; height: 240px;')

:::

> id: handshakes-2

Rather than counting all the edges in large graphs, we could also try to find a simple formula that tells us the result for _any_ number of guests.
Thay vì đếm tất cả các cạnh trong một đồ thị lớn, chúng ta cũng có thể tìm một công thức tính được kết quả với _bất kỳ_ số lượng khách nào.

Each of the ${n}{n|5|2,8,1} people at the party shakes hands with ${n-1} others. That makes ${n} × ${n-1} = ${n×(n-1)} handshakes in total. For _n_ people, the number of handshakes would be [[`n×(n–1)`|`n×(n+1)`|`n^2`]].
Cứ ${n}{n|5|2,8,1} người ở bữa tiệc sẽ bắt tay với ${n-1} người khác. Như vậy, có tổng cộng ${n} × ${n-1} = ${n×(n-1)} cái bắt tay. Với _n_ người, số cái bắt tay sẽ là [[`n×(n–1)`|`n×(n+1)`|`n^2`]] cái.

    p.var(:html="handshakeTable(n)")
    x-gesture(target="#handshakes-2 x-var" slide="100,0")

> id: handshakes-2a

Unfortunately this answer is not quite right. Notice how [the first two entries on the top row](->.handshakes_tr:first-child_td:first-child,_.handshakes_tr:first-child_td:nth-child(2)) are actually the same, just flipped around.
Tuy nhiên, câu trả lời này là chưa đúng. Chú ý rằng [hai cặp đầu ở hàng đầu tiên](->.handshakes_tr:first-child_td:first-child,_.handshakes_tr:first-child_td:nth-child(2)) thực chất giống nhau, chỉ là chúng đảo lại vị trí.

In fact, we have counted every handshake [[twice|once|three times]],
_{span.reveal(when="blank-0")}
once for each of the two people involved. This means that the correct number of handshakes for ${n}{n|5|2,25,1} guests is `(var("n") × var("n-1"))/2 = var("n*(n-1)/2")`._
Thực tế, chúng ta đã đếm tất cả những cái bắt tay [[hai lần|một lần|ba lần]],
{span.reveal(when="blank-0")} với mỗi một cặp người. Điều này có nghĩa là số cái bắt tay chính xác với ${n}{n|5|2,25,1} khách tham gia là `(var("n") × var("n-1"))/2 = var("n*(n-1)/2")`.

> id: handshakes-3

The handshake graphs are special because every vertex is connected to every other vertex. Graphs with this property are called __complete graphs__. The complete graph with 4 vertices is often abbreviated as `K_4`, the complete graph with 5 vertices is known as `K_5`, and so on.
Đồ thị biểu diễn những cái bắt tay rất đặc biệt vì mỗi đỉnh được nối với tất cả những đỉnh còn lại. Những đồ thị có đặc điểm này được gọi là __đồ thị đầy đủ__. Đồ thị đầy đủ có 4 đỉnh được viết tắt là `K_4`, đồ thị đầy đủ có 5 đỉnh thì được gọi là `K_5`, vân vân.

We have just shown that a complete graph with `n` vertices, `K_n`, has `(n × (n-1))/2` edges.
Suy ra, một đồ thị đầy đủ có `n` đỉnh, `K_n`, có `(n × (n-1))/2` cạnh.

```
.row
  svg.graph(style="width: 90px; height: 90px")
  svg.graph(style="width: 90px; height: 90px")
  svg.graph(style="width: 90px; height: 90px")
  svg.graph(style="width: 90px; height: 90px")
```

> id: handshakes-4

```
figure: img(src="images/flags.jpg" width=855 height=100)
```

On a different day, you are invited to a speed dating event for ${m}{m|5|2,8,1} boys and ${f}{f|4|2,8,1} girls. There are many small tables and every boy spends 5 minutes with each of the girls. How many individual “dates” are there in total?
Vào một ngày khác, bạn được mời đến một buổi hẹn hò tốc độ bao gồm ${m}{m|5|2,8,1} nam và ${f}{f|4|2,8,1} nữ. Có nhiều chiếc bàn nhỏ và mỗi bạn nam sẽ nói chuyện với một bạn nữ trong 5 phút. Có tổng cộng bao nhiêu “cuộc hẹn nhỏ” như vậy?

::: column.grow

In this case, the corresponding graph consists of two separate sets of vertices. Every vertex is connected to all the vertices in [[the opposite|its own]] set, but none of the vertices in [[its own|the opposite]] set. Graphs which have this layout are called __bipartite graphs__.
Trong trường hợp này, đồ thị tương ứng bao gồm 2 tập hợp các đỉnh riêng biệt. Mỗi đỉnh nối với tất cả các đỉnh thuộc [[tập hợp đối diện|chính tập hợp của mình]], nhưng không nối với đỉnh nào thuộc [[chính tập hợp của mình|tập hợp đối diện]]. Những đồ thị có bố cục như này được gọi là __đồ thị hai phía__.

::: column(width=300)

    svg.graph(style="width: 300px; height: 140px;")

:::

{.reveal(when="blank-0 blank-1")} The bipartite graph with two sets of size _x_ and _y_ is often written as `K_"x,y"`. It has [[`x×y`|`x+y`|`2x–y`]] edges, _{span.reveal(when="blank-2")} which means that in the above example there are ${m} × ${f} = ${m×f} dates._
Đồ thị hai phía với 2 tập hợp _x_ và _y_ thường được viết là `K_"x,y"`. Nó có [[`x×y`|`x+y`|`2x–y`]] cạnh, _{span.reveal(when="blank-2")} tức là ví dụ trên có tất cả ${m} × ${f} = ${m×f} cuộc hẹn nhỏ.

> id: utilities
> goals: try-three-times
> section: planar-graphs

Planar Graphs Đồ thị phẳng

::: column.grow

Here is another puzzle that is related to graph theory.
Dưới đây là một câu đố khác liên quan đến lý thuyết đồ thị.

In a small village there are three houses and three utility plants that produce water, electricity and gas. We have to connect each of the houses to each of the utility plants, but due to the layout of the village, the different pipes and cables are not allowed to cross.
Ở một ngôi làng nhỏ, có 3 ngôi nhà và 3 nhà máy tiện ích cung cấp nước, điện và gas. Chúng ta phải nối mỗi ngôi nhà với từng nhà máy, nhưng do bố cục ngôi làng, những loại ống và cáp khác nhau sẽ không được phép giao nhau.

::: column(width=300)

x-img(width=300 height=200 src="images/power-plant.jpg")

:::

Try to connect each of the houses to each of the utility companies below, without any of your lines intersecting:
Hãy nối mỗi ngôi nhà với các nhà máy dưới đây mà những đường đó không được giao nhau.

.box.no-padding
include svg/utilities.svg
button.btn Clear

> id: utilities-1

Just like the Königsberg bridges before, you quickly discover that this problem is also impossible. It seems that some graphs can be drawn without overlapping edges – these are called __planar graphs__ – but others cannot.
Giống như bài toán những cây cầu ở Königsberg, bạn có thể nhanh chóng thấy rằng bài toán này cũng không thể giải được. Dường như có một số đồ thị có thể được vẽ mà các cạnh không cắt nhau - chúng được gọi là __đồ thị phẳng__ - nhưng có những đồ thị không thể làm được như vậy.

::: column(width=200)

svg.graph(width=200 height=200 style="margin-bottom: .4em")

{.text-center} `K_3` is planar. `K_3` là một đồ thị phẳng.

::: column(width=200)

svg.graph#planar-2(width=200 height=200 style="margin-bottom: .4em")

{.text-center} `K_4` [[is planar|is not planar]]. `K_4` [[là một đồ thị phẳng|không là một đồ thị phẳng]].

::: column(width=200)

svg.graph#planar-3(width=200 height=200 style="margin-bottom: .4em;")

{.text-center} `K_5` [[is not planar|is planar]]. `K_5` [[là một đồ thị phẳng|không là một đồ thị

phẳng]].

:::

> id: utilities-2

The [complete graph](gloss:complete-graph) `K_5` is the smallest graph that is not planar. Any other graph that contains `K_5` as a subgraph in some way is also not planar. This includes `K_6`, `K_7`, and all larger complete graphs.
[Đồ thị đầy đủ](chú giải:đồ thị đầy đủ `K_5` là đồ thị nhỏ nhất không phải là một đồ thị phẳng. Bất cứ đồ thị nào chứa một đồ thị con `K_5` đều không phải là đồ thị phẳng. Điều này bao gồm `K_6`, `K_7`, và tất cả đồ thị đầy đủ lớn hơn.

The graph in the three utilities puzzle is the [bipartite graph](gloss:bipartite-graph) `K_"3,3"`. It turns out that any non-planar graph must either contain a `K_5` or a `K_"3,3"` (or a [subdivision](gloss:subdivision) of these two graphs) as a subgraph. This is called _Kuratowski's theorem_.
Đồ thị ở câu đố trên là [đồ thị hai phía](chú giải:đồ thị hai phía) `K_"3,3"`. Như vậy, bất cứ đồ thị không phẳng nào sẽ bao gồm một đồ thị con `K_5` hoặc `K_"3,3" (hoặc một [đồng phôi](chú giải:đồng phôi) của hai đồ thị này). Đây được gọi là _định lý Kuratowski_.

    // TODO Add bio of Kazimierz Kuratowski

> id: planarity
> goals: planarity

::: .box.blue

Planarity

    x-solved
    svg#planarity(viewBox="0 0 720 360")

This is a planar graph, but the ${n}{n|7|5,20,1} vertices have been scrambled up. Rearrange the vertices so that none of the edges overlap.
Đây là một đồ thị phẳng, nhưng ${n}{n|7|5,20,1} đỉnh này đã bị rối. Hãy sắp xếp lại các đỉnh sao cho không có cạnh nào đè lên nhau.

    p.btn-row: button.btn New Random Graph
    // TODO Maybe mention that the restriction to straight line edges in the Planarity puzzle isn't
    // a restriction that matters (Fáry's Theorem).

:::

> id: euler

Euler's Formula Công thức Euler

All planar graphs divide the plane they are drawn on into a number of areas,
called __faces__.
Tất cả đồ thị phẳng chia mặt phẳng của chúng thành các ô được gọi là __mặt__.

::: column(width=200)

  include svg/euler-2.svg

{.text-center} [[6]] Vertices
[[5]] Faces
[[10]] Edges
{span.euler-sum} 11 Vertices + Faces

::: column(width=200)

  include svg/euler-1.svg

{.text-center} [[8]] Vertices
[[7]] Faces
[[14]] Edges
{span.euler-sum} 15 Vertices + Faces

::: column(width=200)

  include svg/euler-3.svg

{.text-center} [[12]] Vertices
[[13]] Faces
[[24]] Edges
{span.euler-sum} 25 Vertices + Faces

:::

> id: euler-1

When comparing these numbers, you will notice that the number of edges is always [[one less|bigger|the same]] than the number of faces plus the number of vertices. In other words, _{.b.m-blue}F_ + _{.b.m-green}V_ = _{.b.m-red}E_ + 1. This result is called __Euler's equation__ and is named after the same [mathematician](bio:euler) who solved the Königsberg Bridges problem.
Khi so sánh những con số này, bạn sẽ thấy số cạnh luôn luôn [[ít hơn 1|lớn hơn|bằng]] số mặt cộng số đỉnh. Nói cách khác, _{.b.m-blue}F_ + _{.b.m-green}V_ = _{.b.m-red}E_ + 1. Kết quả này được gọi là __Phương trình Euler__, được đặt tên theo [nhà toán học](tiểu sử: euler) đã giải bài toán Bảy cây cầu Königsberg.

Unfortunately, there are infinitely many graphs, and we can't check every one to see if Euler's equation works. Instead, we can try to find a simple [proof](gloss:proof) that works for any graph…
Tuy nhiên, có vô số đồ thị và chúng ta không thể kiểm tra tất cả để xem Phương trình Euler có đúng hoàn toàn không. Thay vào đó, chúng ta có thể tìm ra một cách [chứng minh](chú giải:chứng minh) đơn giản có thể áp dụng cho bất kỳ đồ thị nào...

> id: euler-2

::: x-slideshow

```
.stage(slot="stage")
  svg(viewBox="0 0 640 200")
    line.link(style="stroke-width: 3px; display: none" x1=270 y1=30  x2=150 y2=100)
    line.link(style="stroke-width: 3px; display: none" x1=150 y1=100 x2=270 y2=170)
    line.link(style="stroke-width: 3px; display: none" x1=270 y1=170 x2=390 y2=100)
    line.link(style="stroke-width: 3px" x1="390" y1="100" x2="270" y2="30")
    circle.node(cx=270 cy=30  r=7)
    circle.node(cx=150 cy=100 r=7 style="display: none")
    circle.node(cx=270 cy=170 r=7 style="display: none")
    circle.node(cx=390 cy=100 r=7 style="display: none")

  .euler-table
    table.grid.table-small
      tr
        td: strong.m-blue.i F
        td: strong.m-green.i V
        td: strong.m-red.i E
      tr
        td.xf 0
        td.xv 1
        td.xe 0
    p.no-voice #[strong.m-blue.xf 0] + #[strong.m-green.xv 1]  =  #[strong.m-
```

red.xe 0] + 1

The simplest graph consists of a single vertex. We can easily check that Euler's equation works.
Đồ thị đơn giản nhất bao gồm duy nhất một đỉnh. Chúng ta có thể dễ dàng chứng minh Phương trình Euler.

Let us add a new vertex to our graph. We also have to add an edge, and Euler's equation still works.
Chúng ta thêm một đỉnh và cả một cạnh vào đồ thị, phương trình Euler vẫn đúng.

If we want to add a third vertex to the graph we have two possibilities. We could create a small triangle: this adds one vertex, one face and two edges, so Euler's equation still works.
Nếu chúng ta muốn thêm một đỉnh thứ ba vào đồ thị, hai khả năng sẽ xảy ra. Chúng ta có thể tạo ra một hình tam giác nhỏ: tức là thêm một đỉnh, một mặt và hai cạnh, như vậy phương trình Euler vẫn đúng.

Instead we could simply extend the line by one: this adds one vertex and one edge, and Euler's equation works.
Thay vào đó, chúng ta có thể đơn giản là kéo dài đường thẳng: tức là thêm một đỉnh và một cạnh, phương trình Euler đúng.

Let's keep going: if we now create a quadrilateral we add one vertex, two edges and one face. Euler's equation still works.
Hãy cứ tiếp tục: nếu bây giờ chúng ta muốn tạo ra một hình tứ giác, chúng ta sẽ thêm một đỉnh, hai cạnh và một mặt. Phương trình Euler tiếp tục đúng.

:::

> id: euler-3

Any (finite) graph can be constructed by starting with one vertex and adding more vertices one by one. We have shown that, whichever way we add new vertices, Euler's equation is valid. Therefore, it is valid for all graphs.
Bất cứ đồ thị (hữu hạn) nào cũng có thể được dựng bằng cách bắt đầu với một đỉnh và thêm vào đó từng đỉnh một. Chúng ta đã chứng minh rằng dù có thêm đỉnh bằng cách này hay cách khác, phương trình Euler vẫn luôn chính xác. Như vậy, nó đúng với tất cả đồ thị.

The process we have used is called __mathematical induction__. It is a very useful technique for proving results in infinitely many cases, simply by starting with the simplest case, and showing that the result holds at every step when constructing more complex cases.
Kỹ thuật chúng ta vừa dùng được gọi là __quy nạp toán học__. Đây là một phương pháp rất hữu dụng để chứng minh kết quả cho vô số trường hợp, chỉ đơn giản bằng cách bắt đầu từ ví

dụ đơn giản nhất, và cho thấy kết quả đều cố định khi áp dụng vào các ví dụ phức tạp hơn.

.svg-block: include svg/dominoes.svg

> id: euler-4

Many planar graphs look very similar to the nets of [polyhedra](gloss:polyhedron), three-dimensional shapes with [polygonal](gloss:polygon) faces. If we think of polyhedra as made of elastic bands, we can imagine stretching them out until they become flat, planar graphs:
Nhiều đồ thị phẳng nhìn rất giống các [hình đa diện](chú giải:đa diện), hình 3D với các mặt [đa giác](chú giải:đa giác). Nếu chúng ta coi hình đa diện là một sợi dây đàn hồi, hãy tưởng tượng kéo dãn nó ra cho đến khi trở thành một đồ thị phẳng:

::: column(width=300)

img.img-sequence(src="images/cube/cube0.png" width=300 height=300)
x-slider(steps=31)

::: column(width=300)

img.img-sequence(src="images/dodecahedron/dodeca0.png" width=300 height=300)
x-slider(steps=31)

:::

> id: euler-5

This means that we can use Euler's formula not only for planar graphs but also for all polyhedra – with one small difference. When transforming the polyhedra into graphs, one of the faces disappears: the topmost face of the polyhedra becomes the "outside"; of the graphs.
Điều này có nghĩa là chúng ta có thể sử dụng công thức Euler không chỉ cho đồ thị phẳng mà còn cho hình đa diện - với một chút khác biệt. Khi chuyển hình đa diện thành đồ thị, một trong số các mặt sẽ biến mất: mặt trên cùng của đa diện trở thành phần ngoài cùng của đồ thị.

In other words, if you count the number of __{.m-red}edges__, __{.m-blue}faces__ and __{.m-green}vertices__ of _any_ polyhedron, you will find that _{.b.m-blue}F_ + _{.b.m-green}V_ = _{.b.m-red}E_ + [[2]].
Nói cách khác, nếu bạn đếm số __{.m-red}cạnh__, __{.m-blue}mặt__ và __{.m-green}đỉnh__

của _bất cứ_ hình đa diện nào, bạn sẽ thấy _{.b.m-blue}F_ +
{.b.m-green}V = _{.b.m-red}E_ + [[2]].

::: column(width=200)

x-video(width=200 height=200 src="images/icosahedron.mp4" hover loop)

{.caption} __Icosahedron__
__{.m-blue}20__ Faces
__{.m-green}12__ Vertices
__{.m-red}30__ Edges

::: column(width=200)

x-video(width=200 height=200 src="images/rhombi.mp4" hover loop)

{.caption} __Rhombicosidodecahedron__
__{.m-blue}62__ Faces
__{.m-green}60__ Vertices
__{.m-red}120__ Edges

::: column(width=200)

x-video(width=200 height=200 src="images/football.mp4" hover loop)

{.caption} __Truncated Icosahedron__
__{.m-blue}32__ Faces (12 black, 20 white)
__{.m-green}60__ Vertices
__{.m-red}90__ Edges

:::

> id: maps
> section: map-colouring

Map Colouring Tô màu bản đồ

::: column.grow

We have already used graph theory with certain maps. As we zoom out, individual roads and bridges disappear and instead we see the outline of entire countries.
Chúng ta đã áp dụng lý thuyết đồ thị với một số bản đồ rồi. Nếu chúng ta thu nhỏ bản đồ lại,

những con đường, cây cầu sẽ biến mất và thay vào đó, chúng ta sẽ nhìn thấy được tổng quan của cả một đất nước.

When colouring a map – or any other drawing consisting of distinct regions – adjacent countries cannot have the same colour. We might also want to use as few different colours as possible.
Khi tô màu một bản đồ - hoặc bất kỳ hình vẽ nào khác bao gồm các vùng riêng biệt - những quốc gia liền kề nhau không thể được tô cùng một màu. Chúng ta cũng cần phải sử dụng ít màu sắc nhất có thể.

Some simple “maps”, like a chessboard, only need two colours (black and white), but most complex maps need more.
Một số loại “bản đồ” đơn giản, như bàn cờ vua, chỉ cần 2 màu duy nhất (đen và trắng), nhưng hầu hết các bản đồ phức tạp cần nhiều màu sắc hơn

```
::: column(width=240 style="margin-top: -10px")

  x-img(src="images/globe.jpg" width=240 height=320)

:::
```

> id: maps-1
> goals: map-0 map-1 map-2 map-3
> title: Colouring Maps

When colouring the map of US states, 50 colours are obviously enough, but far fewer are necessary. Try colouring the maps below with as few colours as possible:
Khi tô màu bản đồ nước Mỹ, bạn đương nhiên có thể dùng 50 màu, nhưng điều đó hoàn toàn không cần thiết. Bạn hãy thử tô bản đồ dưới đây với ít màu sắc nhất có thể:

```
  .four-colour-icons
    for i in [1, 2, 3, 4, 5, 6, 7]
      .four-colour-icon(tabindex=0)

  x-tabbox.four-colours.full-width
    .tab
      h3 United States #[span.check(when="map-0")]
      x-solved
      .colour-count(style="margin-bottom: -32px") #[span 0] colours used
      include svg/colours-1.svg
      button.btn.clear Clear
      // Note that states or countries which only share a corner are allowed to have the same
```

colour. Chú ý rằng những bang hoặc quốc gia chỉ có chung một góc được phép tô màu giống nhau.

// Alaska and Hawaii are isolated from all of the other states and can have any colour.
Alaska và Hawaii là 2 địa điểm tách biệt với các bang còn lại nên có thể được tô bằng bất kỳ màu nào.

```
.tab
  h3 South America #[span.check(when="map-1")]
  x-solved
  .colour-count #[span 0] colours used
  include svg/colours-2.svg
  button.btn.clear Clear
.tab
  h3 Germany #[span.check(when="map-2")]
  x-solved
  .colour-count #[span 0] colours used
  include svg/colours-3.svg
  button.btn.clear Clear
.tab
  h3 England #[span.check(when="map-3")]
  x-solved
  .colour-count #[span 0] colours used
  include svg/colours-4.svg
  button.btn.clear Clear
```

> id: maps-2

> title: The Four Colour Theorem Định lý bốn màu

::: column.grow

All of these maps can be coloured with only four different colours, but it is not hard to imagine that other, very complicated maps might need many more colours. In fact, some maps need __at least__ four colours, whenever they contain four countries all connected to each other.

Tất cả những bản đồ này đều có thể được tô chỉ với bốn màu khác nhau, nhưng không khó để tưởng tượng ra những bản đồ vô cùng phức tạp và cần nhiều màu sắc hơn. Thực tế, một số bản đồ cần __ít nhất__ bốn màu khi chúng có 4 khu vực đều liên kết với nhau.

::: column(width=200)

```
img(src="images/four-colours.png" width=200 height=120)
```

:::

Like before, we can convert a map with countries and borders into a planar graph: every country becomes [[a vertex|an edge|a face]], and countries which [[share a border|have the same colour]] get connected by an edge:
Tương tự, chúng ta có thể chuyển một bản đồ có các nước và biên giới thành một đồ thị phẳng: mỗi nước là một [[đỉnh|cạnh|mặt]], và những nước [[chung biên giới|có cùng màu]] được nối với nhau bằng một cạnh:

.svg-block: include svg/colour-graph.svg

{.reveal(when="blank-0 blank-1")} Now we want to colour the vertices of a graph, and two vertices must have a different colour if they are connected by an edge.
Bây giờ chúng ta sẽ tô màu các đỉnh của đồ thị, và hai đỉnh nào được kết nối với nhau bởi một cạnh thì không được tô cùng màu.

> id: maps-3

::: column(width=240 parent="right")

x-img(lightbox width=240 height=320 src="images/england-counties.jpg")

::: column.grow

In 1852, the botany student [Francis Guthrie](bio:guthrie) had to colour a map of counties in England. He observed that four colours seemed to suffice for any map he tried, but he was not able to find a proof that worked for _all_ maps. This turned out to be an extremely difficult problem, and became known as the __four colour theorem__.
Vào năm 1852, một sinh viên thực vật học [Francis Guthrie](tiểu sử:guthrie) phải tô màu một bản đồ các thị trấn nước Anh. Guthrie thấy rằng 4 màu dường như đủ để tô bất cứ bản đồ nào mà cậu ấy thử, nhưng cậu ấy không thể chứng minh rằng nó đúng với _tất cả_ bản đồ được. Điều này đã trở thành một bài toán khó, và được biết đến với tên gọi __định lý bốn màu__.

During the following 100 years, many mathematicians published “proofs” to the four colour theorem, only for mistakes to be found later. Some of these invalid proofs were so convincing that it took more than 10 years to discover errors.
Trong cả một thế kỷ sau đó, nhiều nhà toán học đã tung ra những “bằng chứng” cho định lý bốn màu, nhưng sau đó đều bị phát hiện có khuyết điểm. Một số bằng chứng lỗi này thuyết phục tới mức phải mất hơn 10 năm để phát hiện ra các lỗ hổng.

For a long time, mathematicians were unable to either prove that four colours are enough, or to find a map that needed more than four colours.
Trong một khoảng thời gian dài, các nhà toán học đã không thể chứng minh được định lý bốn màu cũng như tìm ra được một bản đồ cần nhiều hơn bốn màu.

:::

> id: maps-4

Little progress was made on the four colour problem until 1976, when [Wolfgang Haken](bio:haken) and [Kenneth Appel](bio:appel) used a computer to finally solve it. They reduced infinitely many possible maps to 1936 special cases, which were each checked by a computer taking over 1000 hours in total.
Tình trạng này không có tiến triển gì nhiều cho đến năm 1976, khi [Wolfgang Haken](tiểu sử:haken) và [Kenneth Appel](tiểu sử:appel) dùng một máy tính để giải quyết vấn đề này. Họ giảm tối đa số bản đồ có thể xuống còn 1936 trường hợp đặc biệt, chúng được kiểm tra bởi một chiếc máy tính trong hơn 1000 giờ đồng hồ.

x-parallax.full-width(background="images/ibm-360.jpg")

> id: maps-5

The __four colour theorem__ is the first well-known mathematical theorem to be proven using a computer, something that has become much more common and less controversial since. Faster computers and a more efficient algorithm mean that today you can prove the four colour theorem on a laptop in just a few hours.
__Định lý bốn màu__ là một trong những định lý toán học nổi tiếng đầu tiên được chứng minh bằng cách sử dụng máy tính, điều mà đã trở nên phổ biến hơn và ít gây tranh cãi hơn từ khi đó. Ngày nay, với máy tính nhanh hơn và thuật toán hiệu quả hơn cho phép chúng ta chứng minh định lý bốn màu trên một chiếc máy tính xách tay chỉ trong vài giờ.

figure
x-img(src="images/suffice.jpg" width=320 height=80 credit="http://www.math.illinois.edu/History/postmarks.pdf")
p.caption Postmark for the Department of Mathematics at the University of
Illinois Urbana-Champaign, where Haken and Appel worked.

> id: maps-6

::: column.grow

The four colour theorem only works for maps on a flat plane or a sphere, and where all countries consist of a single area.
Định lý bốn màu chỉ đúng khi áp dụng trên mặt phẳng hoặc mặt cầu, và nơi mà các đất nước chỉ bao gồm một khu vực duy nhất.

However, mathematicians have also looked at maps of _empires_, where countries can consist of multiple disconnected components, and at maps on differently-shaped planets, such as a torus (doughnut shape). In these cases you may need more than four colours, and the proofs become even more difficult.
Tuy nhiên, các nhà toán học cũng đã xem xét bản đồ của các_đế chế_, nơi mà một đất nước có thể bao gồm nhiều lục địa tách biệt nhau, và cả bản đồ những hành tinh có hình dạng khác biệt, ví dụ như hình xuyến (hình bánh donut). Trong những trường hợp này, bạn có thể cần nhiều hơn bốn màu, và để chứng minh điều đó thì còn phức tạp hơn.

::: column(width=300)

    x-video(width=300 height=220 src="images/torus.mp4" hover loop)
    p.caption This map on a torus requires seven colours. Bản đồ hình xuyến này cần tới 7 màu sắc.

:::

> id: salesman
> section: travelling-salesman

The Travelling Salesman Problem Bài toán người giao hàng

::: column.grow(parent="right")

Let us think, once more, about networks and maps. Imagine that a delivery service has to visit ${tsn}{tsn|8|2,50,1} different cities to distribute parcels. We can think of these cities as the vertices in a graph. If all the cities are connected by roads, this is a [[complete graph|cycle|bipartite graph]], so there are `(var("tsn") × (var("tsn") – 1)) / 2 = var("tsn*(tsn-1)/2")` edges in total.
Hãy suy nghĩ một lần nữa về mạng lưới và bản đồ. Hãy tưởng tượng một dịch vụ giao hàng phải đi tới ${tsn}{tsn|8|2,50,1} thành phố khác nhau để giao bưu kiện. Chúng ta có thể coi những thành phố này là các đỉnh trong một đồ thị. Nếu tất cả các thành phố được nối với nhau bởi những con đường thì đây là một đồ thị [[đầy đủ|chu trình|hai phía]], vì vậy có tổng cộng `(var("tsn") × (var("tsn") – 1)) / 2 = var("tsn*(tsn-1)/2")` cạnh.

The delivery truck has to visit all cities, in any order. In the Königsberg bridges problem we wanted to find paths which travel along _every edge_ exactly once. Now we want to find paths which visit _every vertex_ exactly once. These paths are called __Hamiltonian cycles__.
Xe tải chở hàng phải tới tất cả các thành phố theo bất kỳ thứ tự nào. Ở bài toán Bảy cây cầu Königsberg, chúng ta cần tìm một tuyến đường đi qua _hết các cạnh_ duy nhất một lần. Bây giờ thì chúng ta cần tìm đường đi qua _mỗi đỉnh_ duy nhất một lần. Những con đường này

được gọi là __chu trình Hamiltonian__.

::: column(width=260)

x-img(src="images/truck.jpg" width=260 height=280)

:::

> id: salesman-1

There are countless different possibilities for Hamiltonian cycles in complete graphs. In fact, we can pick any vertex as starting vertex and then pick any of the remaining cities in any order:
Có vô số trường hợp khác nhau để hình thành chu trình Hamiltonian trên đồ thị đầy đủ. Thực tế, chúng ta có thể chọn bất kỳ đỉnh nào làm điểm bắt đầu và chọn bất kỳ thành phố nào làm địa điểm tiếp theo.

.row
  .grow: p.todo Diagram coming soon…
  .grow: p.todo Diagram Coming Soon…

> id: salesman-2

In a graph with ${tsn1}{tsn1|4|2,10,1} cities, every Hamiltonian cycle must also contain ${tsn1} cities. Now,
Trong đồ thị với ${tsn1}{tsn1|4|2,10,1} thành phố, mỗi chu trình Hamiltonian cũng phải bao gồm ${tsn1} thành phố. Bây giờ,

ul.var(:html="tsmString(tsn1)")

This means that, in total, there are ${tsnPaths(tsn1)} possible paths. A shorthand for this product is ${tsn1}! or ${tsn1} __Factorial__.
Điều này có nghĩa là có tổng cộng ${tsnPaths(tsn1)} con đường khả thi. Cách viết ngắn gọn cho kết quả này là ${tsn1}! hoặc ${tsn1} __Giai thừa__.

You could imagine that it might not be possible to travel directly between two cities - without going via another city. In that case we no longer have a complete graph, and finding the number of Hamiltonian cycles, if they exist at all, becomes much more difficult.
Bạn có thể thấy rằng chưa chắc chúng ta có thể đến thẳng một thành phố mà không đi qua một thành phố khác. Trong trường hợp đó, chúng ta sẽ không còn có một đồ thị đầy đủ nữa, và việc tìm ra chu trình Hamiltonian, nếu có, sẽ trở nên khó khăn hơn nhiều.

> id: salesman-3

::: column.grow(parent="right")

So far we have ignored the fact that some cities might be further apart than others. In real life, however, this is a very important consideration: we don't just want to find _any_ path but we want to find the shortest one. This is called the __Travelling Salesman Problem__. It has to be solved not only in transportation and logistics, but also when positioning transistors on microchips, to make faster computers, or when analysing the structure of [DNA](gloss:dna).
Cho tới thời điểm này chúng ta đã bỏ qua việc một số thành phố có thể ở xa hơn những thành phố còn lại. Trên thực tế, đây là một yếu tố rất quan trọng: chúng ta không chỉ tìm ra _bất cứ_ tuyến đường nào mà phải tìm ra tuyến ngắn nhất. Đây được gọi là __Bài toán người giao hàng__. Chúng không chỉ giải bài toán về vận chuyển và logistics mà còn để định vị điện trở trong vi mạch để làm máy chạy nhanh hơn, hoặc khi phân tích cấu trúc [ADN](chú giải:ADN)

One simple method would be to try all possible paths, finding the length of each, and then picking the shortest one. However we have just shown that, even with just ${tsn2}{tsn2|10|2,20,1} cities there are ${tsn2}! = ${factorial(tsn2)} possible paths. Once you have hundreds or thousands of vertices, trying all possible paths becomes impossible, even using powerful computers.
Một phương pháp đơn giản là thử qua tất cả các tuyến đường, tìm độ dài của chúng và chọn tuyến ngắn nhất. Tuy nhiên, chúng ta đã thấy là chỉ với ${tsn2}{tsn2|10|2,20,1} thành phố mà đã có ${tsn2}! = ${factorial(tsn2)} con đường khả thi. Một khi bạn có hàng trăm hoặc hàng nghìn đỉnh thì thử hết tất cả các khả năng là một việc không thể thực hiện được ngay cả khi bạn dùng siêu máy tính.

::: column(width=220)

    x-img(lightbox src="images/microchip.jpg" width=210 height=365)

:::

> id: salesman-4
> goals: move

Unfortunately, there is no more efficient algorithm to solve the travelling salesman problem. Instead, mathematicians and computer scientists have developed various algorithms that find _good_ solutions, even if they may not be the very best one. These algorithms, which only give approximate solutions, are called

__Heuristics__.
Tuy nhiên không may rằng, không có một thuật toán nào hiệu quả hơn để giải quyết bài toán người giao hàng. Thay vào đó, các nhà toán học và các nhà khoa học máy tính đã phát triển rất nhiều các thuật toán có thể tìm ra lời giải _tốt_, mặc dù nó không phải là phương pháp hiệu quả nhất Những thuật toán mà tìm ra những giải pháp tương đối này được gọi là __Heuristics__.

Try rearranging the cities on this map, and watch how the shortest path between them changes. You can remove cities by tapping them, and you can add cities by clicking anywhere on the map (up to 8):
Thử sắp xếp lại các thành phố trên bản đồ và quan sát xem con đường ngắn nhất giữa chúng thay đổi như thế nào. Bạn có thể loại bỏ thành phố bằng cách nhấp vào chúng, và thêm thành phố bằng cách nhấn vào bất kỳ vị trí nào trên bản đồ (tối đa 8):

```
figure: .tsm
  svg(width=760 height=480 viewBox="0 0 760 480")
```

> id: salesman-5

::: column.grow

The __Greedy Algorithm__ (or Nearest Neighbour Algorithm) is very simple: you start in a random city and consecutively move to the closest city you haven't visited before. Once you have visited all cities, you stop.
__Giải thuật tham lam__ (hoặc Thuật toán láng giềng gần nhất) rất đơn giản: bạn bắt đầu ở một thành phố bất kỳ và liên tiếp đi đến thành phố gần nhất mà bạn chưa từng đi qua. Khi bạn ghé hết tất cả thành phố thì dừng lại.

::: column(width=300)

{.todo} Animation coming soon…

:::

You can show that, on average, paths found using the greedy algorithm are 25% longer than the shortest possible path.
Trung bình, con đường được tìm ra khi sử dụng giải thuật tham lam dài hơn 25% so với con đường ngắn nhất.

> id: salesman-6

::: column.grow

The __2-Opt Algorithm__ starts with a random possible path. Then you repeatedly pick two edges and swap them around if that would reduce the length of the path. You stop when you can't reduce the length further by swapping any pairs of edges.
__Thuật toán 2-Opt__ bắt đầu với một đường đi bất kỳ. Sau đó bạn liên tục chọn hai cạnh và tráo đổi vị trí của chúng nếu nó giúp giảm độ dài đường đi. Quá trình này dừng lại khi bạn không thể làm con đường ngắn đi bằng cách đổi vị trí bất cứ cặp cạnh nào nữa.

::: column(width=300)

{.todo} Animation coming soon…

:::

> id: ants

It turns out that, long before computers even existed, Nature had found a clever way to find optimal paths between different locations: in ant colonies.
Hóa ra là từ lâu trước cả khi máy tính ra đời, Mẹ thiên nhiên đã tìm ra một cách thông minh để tìm ra tuyến đường tối ưu để đi lại giữa các địa điểm khác nhau, điều đó được thể hiện ở các đàn kiến.

x-parallax.full-width(background="images/ants.jpg")

Ants want to find the shortest possible routes between their nest and possible food sources. They can communicate with each other through chemicals which they leave along their trail, and which other ants can follow.
Những con kiến muốn tìm ra con đường ngắn nhất có thể giữa tổ và các nguồn thức ăn. Chúng có thể giao tiếp với nhau qua những chất hóa học được để lại trên đường đi và giúp những con kiến khác có thể đi theo.

> id: ants-1

::: column.grow

* The ant colony sends out many scouts which initially travel in random directions. Once they find food, they return, leaving behind a trail of pheromone.
Đàn kiến gửi đi những “trinh sát” để đi khám phá những địa điểm bất kỳ. Một khi chúng tìm thấy đồ ăn và quay lại tổ, chúng sẽ để lại pheromone dọc đường đi.
* Other ants tend to follow a trail when they find one, which leads them to food. On their return journey they deposit more pheromone, thus reinforcing

the trail.
Những con kiến khác thường cứ đi theo con đường mòn dẫn đến thức ăn đó nếu chúng tìm thấy. Trên đường về thì chúng lại tiết ra thêm pheromone, như vậy con đường sẽ càng được củng cố.
* Over time, pheromone evaporates. The longer a path is, the more time it takes ants to travel along it, and so the pheromone has more time to evaporate. Short paths, on the other hand, can get reinforced more quickly, so their strength increases faster.
Pheromone sẽ dần dần bay hơi đi. Con đường càng dài thì càng cần nhiều thời gian di chuyển, và vì thế pheromone càng có nhiều thời gian để bốc hơi. Những con đường ngắn thì lại khác, chúng được củng cố nhanh hơn và dễ dàng hơn.

::: column(width=240)

{.todo} Diagram coming soon…

:::

> id: ants-2

::: column(width=220 parent="right")

x-img(style="margin-top: 5px" src="images/ant.jpg" width=220 height=220)

::: column.grow

Ant Colony System (ACS) algorithms try to replicate this behaviour on computers, using many "virtual" ants. They can quickly find very good solutions for the travelling salesman problem.
Thuật toán tối ưu hóa đàn kiến (ACS) bắt chước hành vi này của kiến vào máy tính, sử dụng những con kiến ảo. Chúng có thể nhanh chóng tìm ra những giải pháp hiệu quả cho bài toán người giao hàng.

One particularly useful property of ACS algorithms is that they can run continuously and adapt in real time to changes to the graph. These changes could be caused by car accidents and road closures on street networks, or by traffic spikes to web servers on computer networks.
Một tính chất hữu dụng của thuật toán tối ưu hóa đàn kiến là chúng có thể chạy liên tục và thích ứng với thay đổi thực tế của đồ thị. Một số thay đổi có thể là tai nạn ôtô, đường bị chặn hay giao thông tăng đột biến trên mạng lưới máy chủ trên máy tính.

:::

> id: ants-3

::: column(width=140)

    img(src="images/binary.jpg" width=140 height=320)

::: column.grow

The Travelling Salesman problem is [NP-hard](gloss:np), which means that it is very difficult to be solved by computers (at least for large numbers of cities).
Bài toán người giao hàng [NP-khó](chú giải:np), tức là rất khó để giải bằng máy tính (nhất là với số lượng lớn các thành phố).

Finding a fast and exact algorithm would have serious implications in the field of computer science: it would mean that there are fast algorithms for _all_ NP-hard problems. It would also render most of Internet security useless, which relies on the fact that certain problems are believed to be very difficult for computers.
Tìm ra được một thuật toán nhanh và chính xác sẽ kéo theo rất nhiều tác động trong lĩnh vực khoa học máy tính: nó có nghĩa là sẽ có thuật toán nhanh cho _tất cả_ các bài toán NP-khó. Nó cũng sẽ có thể làm cho hầu hết các bảo mật mạng Internet trở nên vô dụng vì chúng dựa trên những bài toán khó mà máy tính không giải được.

Finding a fast algorithm to solve the Travelling Salesman problem would also solve one of the most famous open problems in mathematics and computer science, the __P vs NP__ problem. It is one of the seven [Millennium Prize Problems](gloss:millennium-prize), each carrying a \$1m prize.
Tìm ra được một thuật toán để giải bài toán người giao hàng cũng sẽ giúp giải được một trong những bài toán khó nhất trong toán học và khoa học máy tính, bài toán __P và NP__. Đây là một trong bảy [Giải thưởng Những bài toán của Thiên niên kỷ](chú giải:Giải thưởng Những bài toán của Thiên niên kỷ], mỗi giải có phần thưởng là một triệu đô.

:::

> section: scheduling
> sectionStatus: dev

Scheduling Problems Bài toán thời gian biểu

{.todo} Coming Soon

> id: applications
> section: applications

Graphs in Everyday Life Đồ thị trong cuộc sống hằng ngày

We have seen many different applications of graph theory in the previous chapters, although some of them were a bit contrived. However, it turns out that graphs are at the very foundation of many objects, concepts and processes in everyday life.
Chúng ta đã bắt gặp nhiều ứng dụng khác nhau của lý thuyết đồ thị trong các chương trước, mặc dù vài trường hợp trong số chúng có hơi không thực tế. Tuy nhiên, đồ thị lại là nền tảng để tạo ra nhiều đồ vật, khái niệm và tiến trình trong cuộc sống hằng ngày.

::: column.grow

The Internet, for example, is a vast, virtual graph. Every vertex is an individual webpage, and every edge means that there is a hyperlink between two pages. Note that links only go one way, so this graph is [[directed|multi-line|connected]], and that this graph is _very, very, large_.
Ví dụ như mạng Internet là một đồ thị ảo bao la rộng lớn. Mỗi đỉnh là một trang web và mỗi cạnh tượng trưng cho một liên kết giữa hai trang. Hãy chú ý rằng mỗi liên kết chỉ đi một chiều, nên đồ thị này là một đồ thị [[có hướng|nhiều đường|liên thông]], và đây là một đồ thị _rất rất lớn_.

    // * "can be viewed as" instead of "is a vast, virtual graph". "Every
    // vertex represens an individual webpage and every edge a hyperlink
    // from one page to another".

Some websites, like Wikipedia or Facebook, have lots of incoming links, while many smaller websites may have very few incoming links. This is the underlying concept which Google uses to sort search results.
Một số trang web như Wikipedia hay Facebook có rất nhiều liên kết tới trong khi những trang web nhỏ hơn có rất ít. Đây là phương pháp cơ bản mà Google dùng để phân loại kết quả tìm kiếm.

::: column(width=240)

    img(credit="© Various" src="images/websites.png" width=240 height=240)

:::

> id: applications-1

Websites with more incoming links tend to be of higher quality and should be shown at the top of the search results. For example, when searching for “London”, official tourist information sites are shown before small shops in London, or blogs of people who live in London. This simple idea from graph theory, the __Page Rank Algorithm__, made Google much better than other early search engines.
Những trang web có nhiều liên kết tới thường có chất lượng cao hơn và thường hiện lên ở đầu danh sách kết quả tìm kiếm. Ví dụ, nếu bạn tra “London” , những trang cung cấp thông tin du lịch chính thức sẽ hiện lên trước các cửa hàng nhỏ ở London hoặc blog của những người sống ở London. Sáng kiến đơn giản này từ lý thuyết đồ thị, __Thuật toán xếp trang các trang web__, đã cải tiến Google tốt hơn nhưng công cụ tìm kiếm trước đó.

> id: applications-2

The Internet is the largest network ever created by mankind. This image shows a very small proportion of all the servers connected to the Internet:
Mạng Internet là mạng lưới lớn nhất từng được tạo ra bởi con người. Bức ảnh này cho thấy một phần rất nhỏ các máy chủ được kết nối với mạng Internet.

```
x-parallax.full-width(background="images/internet.jpg")
  .credit © LyonLabs, LLC and Barrett Lyon, 2014
```

> id: applications-3

While websites and hyperlinks form a _virtual_ graph, there is also the _physical_ network of computers, servers, routers, phone lines and cables.
Trong khi các trang web và liên kết tạo nên một đồ thị _ảo_ thì có cả những mạng lưới _hữu hình_ của các máy tính, máy chủ, bộ định tuyến, dây điện thoại và cáp.

::: column.grow(parent="right")

Every time you make a phone call or load a website, network operators have to find a way to connect sender and receiver, without exceeding the capacity of any individual cable or connection. Graph theory and probability make it possible to guarantee a reliable service, for example by finding diversions when a particular connection is busy.
Mỗi khi bạn gọi một cuộc điện thoại hay tải một trang web, tổ chức mạng lưới phải tìm một con đường để kết nối với người gửi và người nhận mà không vượt quá khả năng của bất cứ cáp hay kết nối nào. Lý thuyết đồ thị và xác suất đã cho phép chúng ta đảm bảo một dịch vụ uy tín, ví dụ như chủ động chuyển hướng khi có kết nối bận.

::: column(width=220)

x-img(lightbox src="images/phone.jpg" width=220 height=166)

:::

> id: applications-4

Graphs also play an important role in transportation and navigation. All flight, train and subway networks form graphs, which can be used when creating efficient schedules. One of the most recognisable graphs is the London Underground map:
Đồ thị cũng có vai trò quan trọng trong giao thông vận tải và điều hướng. Tất cả các mạng lưới chuyến bay, tàu hỏa và tàu điện đều tạo nên các đồ thị, cho phép việc lên lịch trình một cách hiệu quả. Một trong những đồ thị dễ nhận biết nhất là bản đồ tàu điện ngầm London:

figure: x-img(lightbox src="images/tube-map.png" width=720 height=480 credit="© Transport for London")

> id: applications-5

::: column.grow

All roads and motorways also form a large network, which is used by navigation services like Google Maps when working out the shortest route between two given points.
Tất cả đường và đường cao tốc đều tạo nên một mạng lưới lớn và được các dịch vụ chỉ đường sử dụng, như Google Bản đồ để tìm ra con đường ngắn nhất giữa hai địa điểm.

::: column(width=60)

x-img(credit="© Google" src="images/google-maps.jpg" width=70 height=70)

:::
::: column(width=280)

x-img(lightbox src="images/congestion.jpg" width=280 height=170)

::: column.grow

In the future, __Intelligent Transportation Systems__ will reduce congestion and accidents by routing cars more efficiently, using location data collected from smartphones and self-driving cars. This could save millions of hours lost on the road every year, significantly reduce pollution, and allow emergency services to

travel faster.
Trong tương lai, __Hệ thống giao thông thông minh__ sẽ làm giảm tắc nghẽn và tai nạn bằng cách chỉ dẫn ô tô hiệu quả hơn, sử dụng dữ liệu vị trí thu thập được từ điện thoại thông minh và ô tô tự lái. Điều này sẽ giúp tiết kiệm hàng triệu giờ đồng hồ trên đường mỗi năm, giảm mạnh ô nhiễm và cho phép các dịch vụ khẩn cấp di chuyển nhanh hơn.

:::

> id: applications-6

This image shows the network of commercial airline flights across northern Europe.
Bức ảnh này cho chúng ta thấy mạng lưới các chuyến bay thương mại trên khắp châu Âu.

    x-parallax.full-width(background="images/flights.jpg")

> id: applications-7

There are countless other graphs in science, engineering or everyday life:
Có vô số các đồ thị khác trong khoa học, kỹ thuật và đời sống hàng ngày.

::: column(width=200)

    x-img(lightbox src="images/molecules.jpg" width=200 height=200)

{.caption} The links between atoms in __molecules__ and crystal grids form a graph.
Liên kết giữa các nguyên tử trong các __phân tử__ và lưới tinh thể tạo nên một đồ thị.

::: column(width=200)

    x-img(lightbox src="images/epidemic.jpg" width=200 height=200)

{.caption} The __spread of diseases__ and epidemics can be modelled using a network.
__Sự lan truyền bệnh__ và bệnh dịch có thể được mô hình hóa bằng cách sử dụng mạng lưới.

::: column(width=200)

    x-img(lightbox src="images/evolution.jpg" width=200 height=200)

{.caption} In Biology, the __evolutionary trees__ that show the ancestry of species form a graph.
Trong Sinh học, __cây tiến hóa__ chỉ ra tổ tiên của các loài cũng là một đồ thị.

::: column(width=200)

x-img(lightbox src="images/network6.jpg" width=200 height=200)

{.caption} The different components of __electric circuits__ and computer chips form a network.
Các thành phần khác nhau của __mạch điện__ và các vi mạch tạo nên một đồ thị.

::: column(width=200)

x-img(lightbox src="images/letters.jpg" width=200 height=200)

{.caption} The grammatical structure of __languages__ can be modelled using graphs, for example to create translation algorithms.
Những cấu trúc ngữ pháp của __các ngôn ngữ__ có thể được mô hình hóa bằng cách sử dụng đồ thị, vídụ như tạo các thuật toán phiên dịch

::: column(width=200)

x-img(lightbox src="images/finance.jpg" width=200 height=200)

{.caption} Graphs also have many applications in __probability__, __game theory__ and __financial mathematics__.
Đồ thị cũng có nhiều ứng dụng trong __xác suất__, __lý thuyết trò chơi__ và __toán tài chính__.

:::

> id: social

Social Networks Mạng lưới xã hội

Finally, let us think about one particularly good example of graphs which exist in everyday life: social media. Here, vertices represent [[people|friends|networks]] and edges represent friendships, likes, subscriptions or followers.
Cuối cùng, một ví dụ thú vị về đồ thị hiện diện trong đời sống hàng ngày của chúng ta: Mạng xã hội. Ở đây, các đỉnh đại diện cho [[mọi người|bạn bè|mạng lưới]] và các cạnh đại diện cho kết bạn, lượt yêu thích, đăng ký hoặc theo dõi.

When we draw social media graphs, we might see certain __clusters__ of mutual friends, who may have gone to the same school or live in the same city. We can also determine people's __centrality__, which depends on how well-connected a vertex is, and which may be a measure of a person's popularity on social media.

Khi chúng ta vẽ đồ thị mạng xã hội, chúng ta sẽ thấy một __cụm__ bạn bè qua lại nhất định, họ là những người học cùng trường hoặc sống trong cùng một thành phố. Chúng ta cũng có thể chọn __tập trung__ một số người, điều đó phụ thuộc vào một đỉnh được kết nối nhiều như thế nào, và đâu là cách đo mức độ phổ biến của một người trên mạng xã hội.

figure: x-img(lightbox src="images/social-network.png" width=720 height=500)

> id: social-1

::: column.grow

In 2014, Facebook had 1.4 billion active users and a total of more than 200 billion friendships. Half of all Facebook users have more than 200 friends, and since most of our friends have a similar number of friends, we could easily have tens of thousands of _friends of friends_.
Vào năm 2014, Facebook có 1,4 tỷ người dùng tích cực và có tổng cộng hơn 200 tỷ “tình bạn”. Một nửa số người dùng Facebook có hơn 200 bạn và vì phần lớn bạn của chúng ta có số lượng bạn tương tự, nên về cơ bản chúng ta có hàng nghìn _bạn của bạn_.

An exciting question would now be: if you pick any two random Facebook users, how many “friendship edges” would you need to follow to get from one to the other? For example, the distance between friends is [[1]], the distance between friends of friends is [[2]], and so on.
Bây giờ chúng ta có một câu hỏi thú vị: nếu bạn chọn bất kỳ 2 người dùng Facebook nào đó, có bao nhiêu “cạnh tình bạn” bạn phải đi theo để có thể đi từ người này tới người kia? Ví dụ, khoảng cách giữa hai người bạn là [[1]], khoảng cách giữa bạn của bạn và bạn là [[2]] và vân vân.

::: column(width=200)

x-img(src="images/facebook-like.png" width=200 height=200)

:::

> id: social-2

In 2016, Facebook conducted [a study](https://research.facebook.com/blog/three-and-a-half-degrees-of-separation/)
to determine how its users are connected to each other. They found that, on average, you are connected to _anyone else_ on Facebook through at most 3.57 other people. And this includes celebrities, politicians or even royalty!
Vào năm 2016, Facebook thực hiện một [nghiên cứu](https://research.facebook.com/blog/three-

and-a-half-degrees-of-separation/) để xem người dùng được kết nối với nhau như thế nào. Họ thấy rằng trung bình bạn được kết nối với _một người bất kỳ_ trên Facebook qua nhiều nhất 3,57 người khác, kể cả khi người bất kỳ đó là người nổi tiếng, chính trị gia hay thậm chí người trong hoàng gia!

In other words, if you pick any one of the billions of Facebook users all around the world, they will probably have a friend of a friend who knows a friend of one of your friends. We say there are 3.57 __degrees of separation__.
Nói cách khác, nếu bạn chọn bất kỳ một trong số hàng tỷ người sử dụng Facebook trên thế giới, họ có thể sẽ có một người bạn của bạn họ mà người đó biết một người bạn của bạn của bạn. Chúng ta gọi đó là có 3,57 __mức độ tách biệt__.

```
figure
  x-img(lightbox src="images/facebook.jpg" width=720 height=360 credit="© Facebook")
  p.caption Geographic visualisation of all Facebook friendships in 2010.
```

> id: social-3

::: column(width=200)

```
x-img(credit="© Metro-Goldwyn-Mayer" src="images/six-degrees.jpg" width=200 height=265 style="border: 1px solid #ccc")
```

::: column.grow

In 1929, when the Hungarian author [Frigyes Karinthy](bio:karinthy) first proposed the idea of “six degrees of Separation”, there was no Internet or social media, but the world had already started to become more interconnected.
Vào năm 1929, khi nhà văn người Hungary [Frigyes Karinthy](tiểu sử:karinthy) lần đầu đề xuất ý tưởng “Sáu mức độ tách biệt” khi chưa hề có mạng Internet hay mạng xã hội nhưng thế giới đã bắt đầu kết nối với nhau.

In 1967, [Stanley Milgram](bio:milgram) conducted a first empirical experiment, where 296 participants living in Nebraska and Kansas were asked to deliver a letter to a particular person living in Boston, Massachusetts. They all had to choose a friend to send the letter to, who then picked another friend. At every step, the letter moved closer to Boston. Milgram found that there were, on average, only 5.2 intermediate friends – 5.2 degrees of separation.
Vào năm 1967, [Stanley Milgram](tiểu sử:milgram) đã cho ra một thí nghiệm thực nghiệm, với 296 người tham gia sống ở Nebraska và Kansas được yêu cầu đưa một bức thư cho một người sống ở Boston, Massachusetts. Họ đều phải chọn một người quen biết để đưa bức thư, sau đó người đó lại chọn một người quen khác. Dần dần, bức thư sẽ tới được Boston. Milgram

thấy rằng trung bình có 5,2 người trung gian – 5,2 mức độ tách biệt.

:::

Today, every one of us is part of countless invisible graphs, which underlie our social interactions, travel, Internet and technology, science, and so much more.
Ngày nay, mỗi người trong chúng ta đều là một phần của vô số các đồ thị vô hình dựa trên những mối quan hệ xã hội, đi lại, mạng Internet và công nghệ, khoa học và hơn thế nữa.