



# Đồ thị - Cây tối đại ngắn nhất

## Nhóm 6

Lê Khánh Duy – 18520661  
Hoàng Văn Hùng – 1852079  
4  
Tô Nhật Huy – 18520855





## Các mục tìm hiểu

- 01** Đồ thị và biểu diễn đồ thị trong chương trình máy tính.
- 02** Cây tối đai ngắn nhất
- 03** Thuật toán Prim
- 04** Thuật toán Kruskal

# Bản phân công công việc

Tên	Nội dung	Hoàn thành
Hoàng Văn Hùng	Tìm tài liệu nghiên cứu về Kruskal+ Lập trình Web	100%
Tô Nhật Huy	Lập trình Web	100%
Lê Khánh Duy	Tìm tài liệu nghiên cứu về Prim + làm slice báo cáo	100%

# I. Đồ thị

Các cách biểu diễn đồ thị trong  
chương trình máy tính

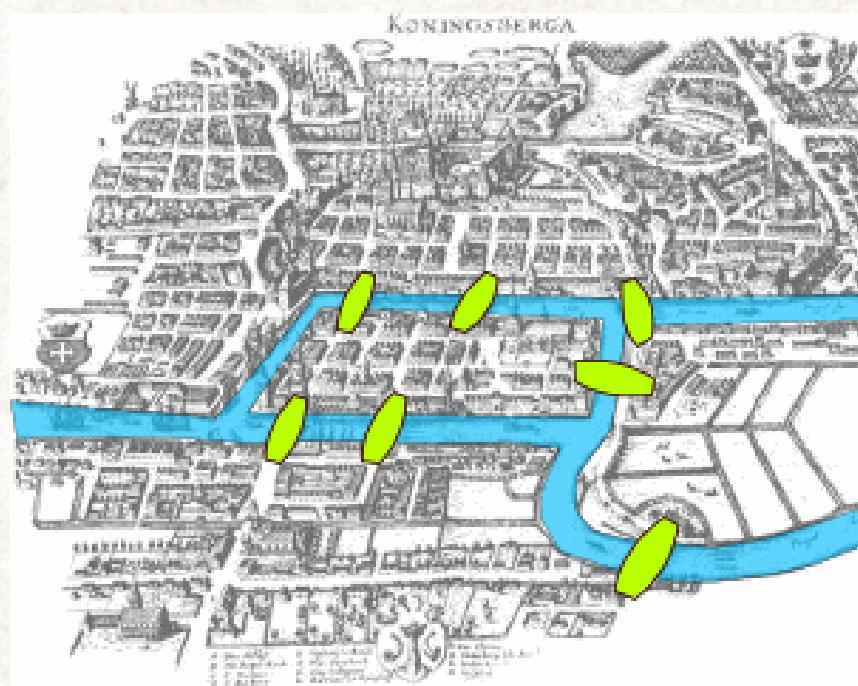


# Đồ Thị

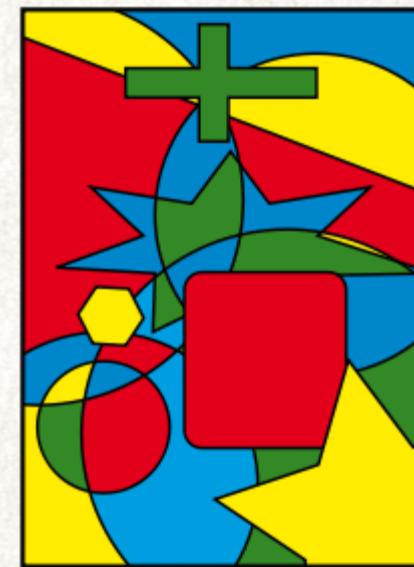
## Lịch sử

- Một trong những kết quả đầu tiên trong lý thuyết đồ thị xuất hiện trong bài báo của Leonhard Euler về Bảy cây cầu ở Königsberg, xuất bản năm 1736.
- Năm 1845, Gustav Kirchhoff đưa ra Định luật Kirchhoff cho mạch điện để tính điện thế và cường độ dòng điện trong mạch điện.
- Năm 1852 Francis Guthrie đưa ra bài toán bốn màu về vấn đề liệu chỉ với bốn màu có thể tô màu một bản đồ bất kì sao cho không có hai nước nào cùng biên giới được tô cùng màu. Bài toán này được xem như đã khai sinh ra lý thuyết đồ thị, và chỉ được giải sau một thế kỉ vào năm 1976 bởi Kenneth Appel và Wolfgang Haken. Trong khi cố gắng giải quyết bài toán này, các nhà toán học đã phát minh ra nhiều thuật ngữ và khái niệm nền tảng cho lý thuyết đồ thị.

# Minh họa



Bảy cây cầu ở Königsberg



bài toán bốn màu



# Đồ Thị

## Khái niệm

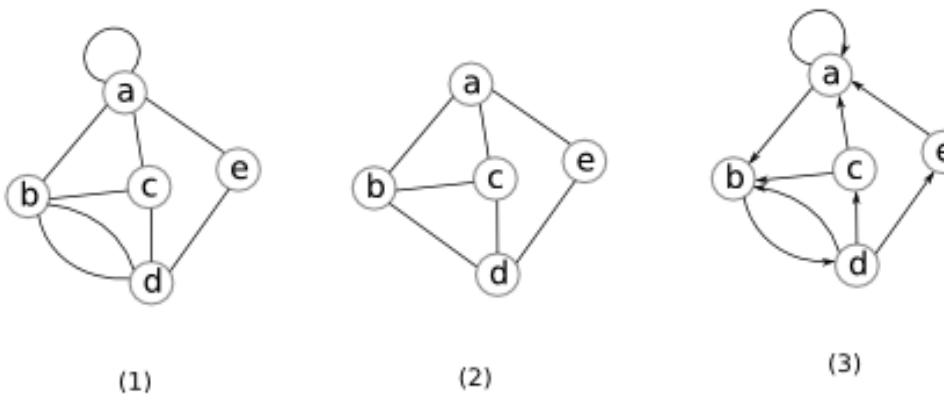
- Đồ thị là một cấu trúc rời rạc gồm các đỉnh và các cạnh nối các đỉnh đó. Được mô tả hình thức:

$$G = (V, E)$$

với V gọi là tập đỉnh (Vertices) còn E gọi là tập cạnh (Edges). Có thể gọi E là tập các cặp  $(u, v)$  với  $u, v$  là 2 đỉnh của V

- Đồ thị gọi là đồ thị có hướng nếu các cạnh trong E có định hướng (3) còn nếu cạnh không định hướng gọi là đồ thị vô hướng (1,2)

- Đồ thị gọi là đa đồ thị nếu giữa 2 đỉnh có nhiều hơn 1 cạnh (1,3) còn 2 đỉnh có nhiều nhất 1 cạnh là đơn đồ thị (2)



# Đồ Thị

## Ứng dụng

- Lý thuyết đồ thị được ứng dụng nhiều trong phân tích lưới. Có hai kiểu phân tích lưới. Kiểu thứ nhất là phân tích để tìm các tính chất về cấu trúc của một lưới, chẳng hạn nó là một scale-free network hay là một small-world network. Kiểu thứ hai, phân tích để đo đạc, chẳng hạn mức độ lưu thông xe cộ trong một phần của mạng lưới giao thông (transportation network).
- Lý thuyết đồ thị còn được dùng trong nghiên cứu phân tử. Trong vật lý vật chất ngưng tụ, cấu trúc ba chiều phức tạp của các hệ nguyên tử có thể được nghiên cứu một cách định lượng bằng cách thu thập thống kê về các tính chất lý thuyết đồ thị có liên quan đến cấu trúc tô pô của các nguyên tử. Ví dụ, các vành đường đi ngắn nhất Franzblau (Franzblau's shortest-path rings).
- Ví dụ như mạng internet là một đồ thị ảo cực lớn. Mỗi đỉnh là một trang web, và mỗi cạnh là đường kết nối giữa hai trang.
- Một số trang web, như Wikipedia hay Facebook, có rất nhiều đường dẫn, trong khi các trang web khác có ít đường dẫn hơn. Đây là nguyên tắc chính để Google sắp xếp các kết quả tìm kiếm.

Do có nhiều ứng dụng trong thực tế và đặc biệt có nhiều ứng dụng trong công nghệ thông tin nên đồ thị cần được biểu diễn trên máy tính để có thể xử lý các bài toán liên quan đến đồ thị. Vậy để biểu diễn được đồ thị thì chúng ta làm thế nào?



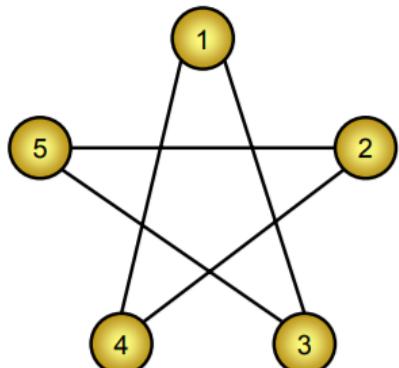
# Các cách biểu diễn đồ thị

## 1. Ma trận kề

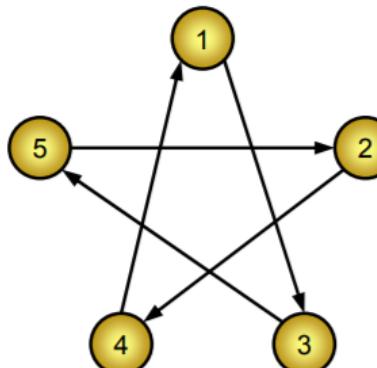
Giả sử  $G = (V, E)$  là một đơn đồ thị có số đỉnh là  $n$  ( $|V|$ ), không mất tí nh tổng quát có thể coi các đỉnh được đánh số từ  $1, 2, \dots, n$ .

Khi đó ta có thể biểu diễn đồ thị bằng một ma trận vuông  $A$  cấp  $n$  trong đó  $a[i][j]$  bằng 1 nếu cạnh có tồn tại và bằng 0 nếu cạnh không tồn tại.

Đối với đa đồ thị thì  $(i, j)$  là ghi số cạnh nối giữa đỉnh  $i$  và  $j$ .



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



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

# Các cách biểu diễn đồ thị

## 1. Ma trận kề

### **Ưu điểm:**

Ma trận kề mô tả rõ ràng đồ thị  
Dễ dàng cài đặt  
Dễ kiểm tra sự liền kề của 2 đỉnh bất kì

### **Nhược điểm:**

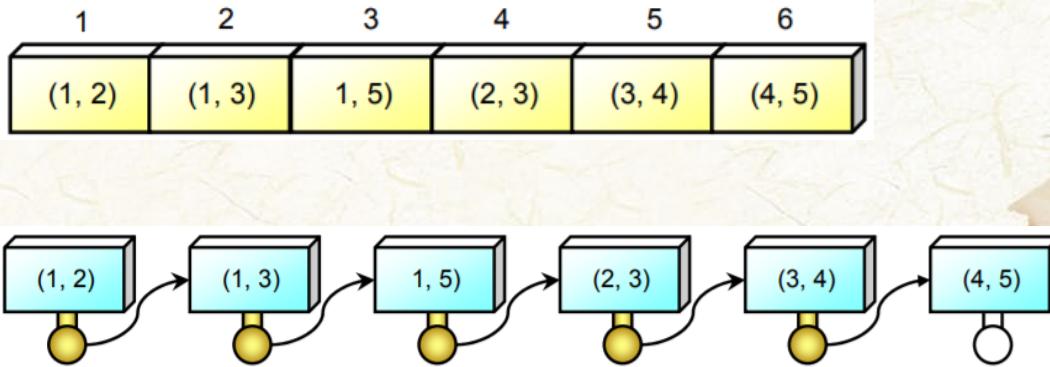
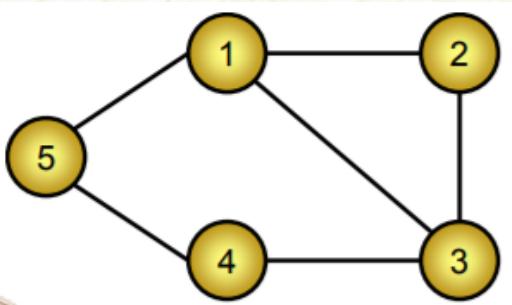
Chi phí lưu trữ cố định là  $n^2$   
Chi phí duyệt các cạnh kề của đỉnh i luôn là n



# Các cách biểu diễn đồ thị

## 2. Danh sách cạnh

Trong trường hợp đồ thị có  $n$  đỉnh và  $m$  cạnh, ta có thể biểu diễn đồ thị dưới dạng danh sách cạnh, trong cách biểu diễn này, người ta liệt kê tất cả các cạnh của đồ thị trong một danh sách, mỗi phần tử là một cặp  $(u, v)$  tương ứng với một cạnh của đồ thị và được lưu dưới dạng mảng hoặc danh sách liên kết



# Các cách biểu diễn đồ thị

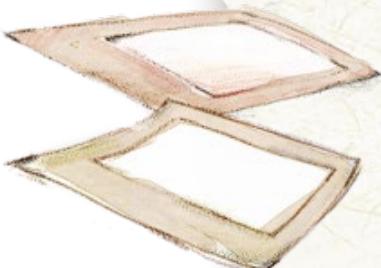
## 2. Danh sách cạnh

### **Ưu điểm:**

Trong trường hợp đồ thị (thừa số cạnh ít), biểu diễn bằng cách này sẽ tiết kiệm không gian lưu trữ, bởi chỉ cần  $2m$  ô nhớ để lưu danh sách cạnh.

### **Nhược điểm:**

Khi ta cần duyệt tất cả các đỉnh kề với đỉnh  $v$  nào đó của đồ thị thì ta phải duyệt tất cả các cạnh, lọc ra những cạnh có chứa đỉnh  $v$  là m tốn rất nhiều thời gian.



# Các cách biểu diễn đồ thị

## 3. Danh sách kề



Để khắc phục nhược điểm của các phương pháp ma trận kề và danh sách cạnh, người ta đề xuất phương pháp biểu diễn đồ thị bằng danh sách kề. Trong cách này, với mỗi đỉnh  $v$  của đồ thị ta cho tương ứng với nó một danh sách các đỉnh kề với  $v$ .

Với đồ thị  $G = (V, E)$   $V$  gồm  $n$  đỉnh và  $E$  gồm  $m$  cạnh có 2 cách cài đặt danh sách kề phổ biến.

Có 2 cách để cài đặt danh sách kề:

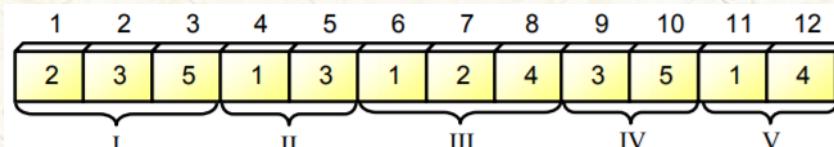
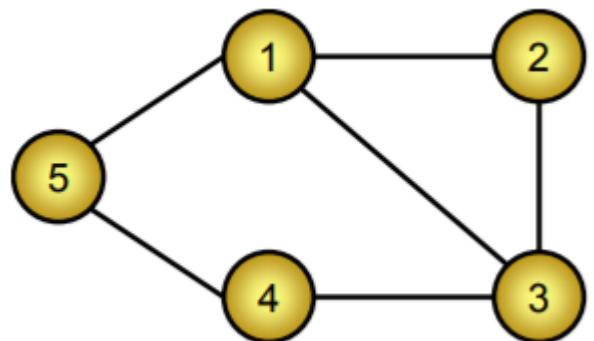


# Các cách biểu diễn đồ thị



## 3. Danh sách kề

Dùng một mảng các đỉnh, mảng đó chia làm n đoạn, đoạn thứ i trong mảng lưu danh sách các đỉnh kề với đỉnh i. Để biết được một đoạn từ vị trí nào đến vị trí nào ta có một mảng lưu vị trí riêng gọi là mảng Head, cấu trúc này gọi là Forward Star



Với đồ thị trên thì mảng vị trí là  $(0, 3, 5, 8, 10, 12)$ . Như vậy đoạn từ vị trí  $\text{Head}[i]+1$  đến  $\text{Head}[i+1]$  trong mảng sẽ chứa các đỉnh kề với  $i$ .

Lưu ý với đồ thị có hướng gồm m cung thì cấu trúc này phải đủ chứa m phần tử còn với đồ thị vô hướng thì là  $2m$ .

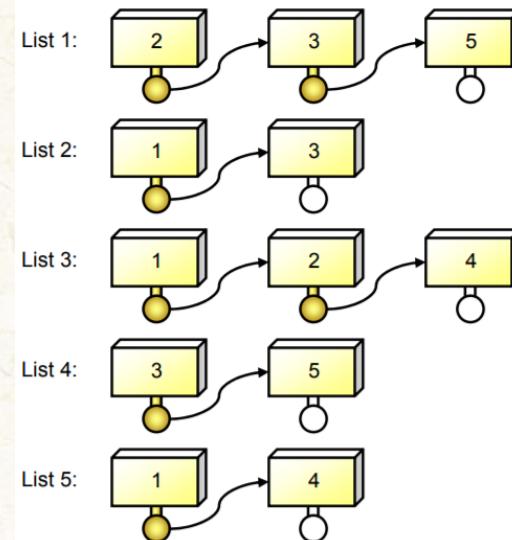
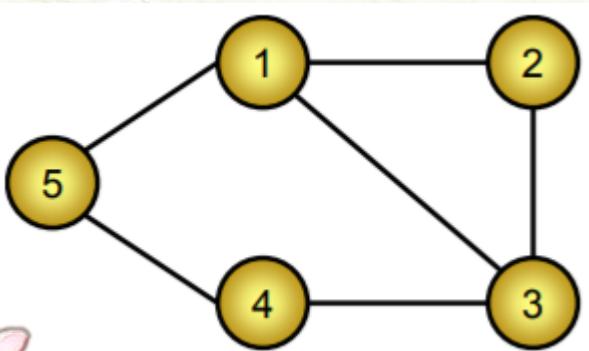


# Các cách biểu diễn đồ thị



## 3. Danh sách kề

Dùng các danh sách liên kết: Với mỗi đỉnh  $i$  của đồ thị, ta cho tương ứng với nó một danh sách móc nối các đỉnh kề với  $i$ , có nghĩa là tương ứng với một đỉnh  $i$ , ta phải lưu lại  $\text{List}[i]$  là chốt của một danh sách liên kết.



# Các cách biểu diễn đồ thị

## 3. Danh sách kề



### Ưu điểm:

Đối với danh sách kề việc duyệt tất cả các đỉnh kề với một đỉnh v là rất dễ dàng. Và việc duyệt tất cả cạnh cũng đơn giản.

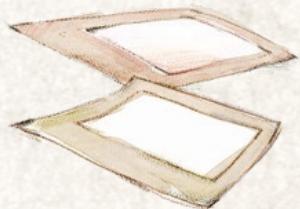
### Nhược điểm:

Về lý thuyết so với 2 phương pháp trên, danh sách kề tốt hơn. Nhưng trong trường hợp cụ thể mà ma trận kề hay danh sách cạnh cũng không thể hiện nhược điểm thì ta nên dùng ma trận kề vì danh sách kề cài đặt rất mất thời gian.



# Code mẫu của việc cài đặt danh sách kè bằng Python:

```
n=int(input("nhap so dinh \n"))
lst = []
for i in range(n):
    temp=[]
    print("nhap so dinh ke cua dinh ",i+1)
    m=int(input())
    for j in range(m):
        temp.append(int(input()))
    lst.append(temp)
print(lst)
```



## **II. Cây tối đai ngắn nhất**

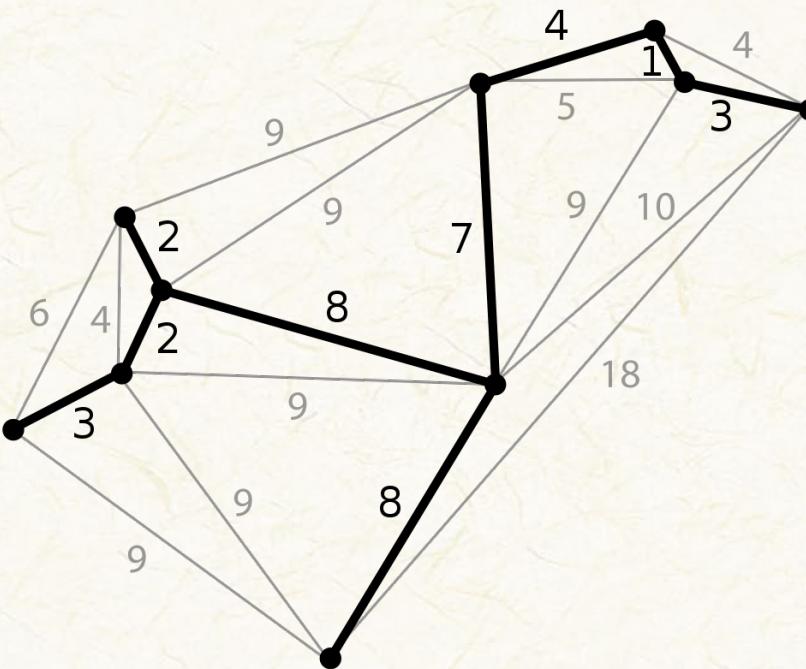
**Khái niệm và ứng dụng**



# Khái niệm

Cho đồ thị vô hướng, cây khung (spanning tree) của đồ thị là một cây con chứa tất cả các đỉnh của đồ thị. Nói cách khác, cây khung là một tập hợp các cạnh của đồ thị, không chứa chu trình và kết nối tất cả các đỉnh của đồ thị.

Trong đồ thị có trọng số, cây khung nhỏ nhất là cây khung có tổng trọng số các cạnh nhỏ nhất. Định nghĩa tương tự với cây khung lớn nhất.



# Ứng dụng

Cây tối đai ngắn nhất được ứng dụng rất nhiều trong việc thiết kế các mạng như mạng máy tính, mạng cáp quang, mạng điện, mạng lưới đường sắt, ... ngoài ra còn một số ứng dụng như:

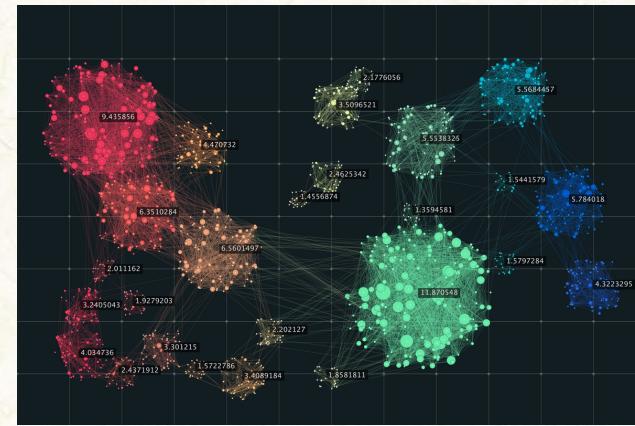
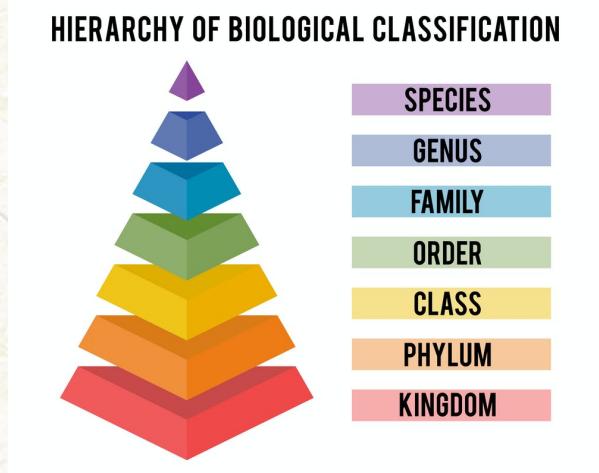
- Phân loại học
- Cluster analysis
- Xây dựng các trạm phát sóng
- Xây dựng trạm quan sát mạng lưới điện
- So sánh dữ liệu của chất độc sinh thái
- Minimax process control
- Nhận biết ký tự viết tay của những biểu thức toán học
- Thiết kế mạch điện
- Nhóm các khu vực địa lý thành các vùng
- Miêu tả đặc điểm của thị trường tài chính để tính toán và tìm ra sự liên hệ giữa các loại hàng hóa.

# Ứng dụng

# Taxonomy

Ứng dụng vào Taxonomy trở thành một cách thức để nhóm nội dung với nhau cho phép tạo một nhóm có cùng tính chất nào đó.

Ứng dụng trong sinh học để phân loại các ngành, loài, họ, ...



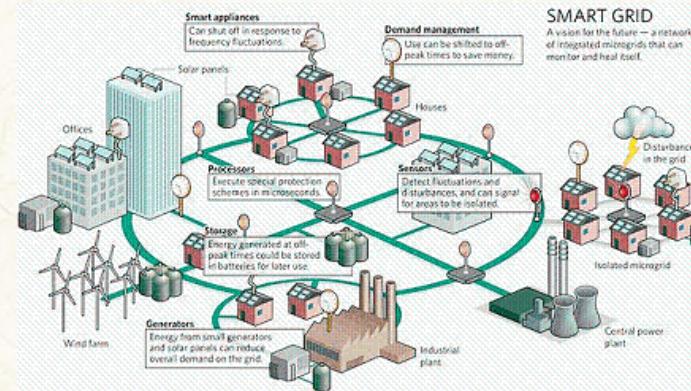
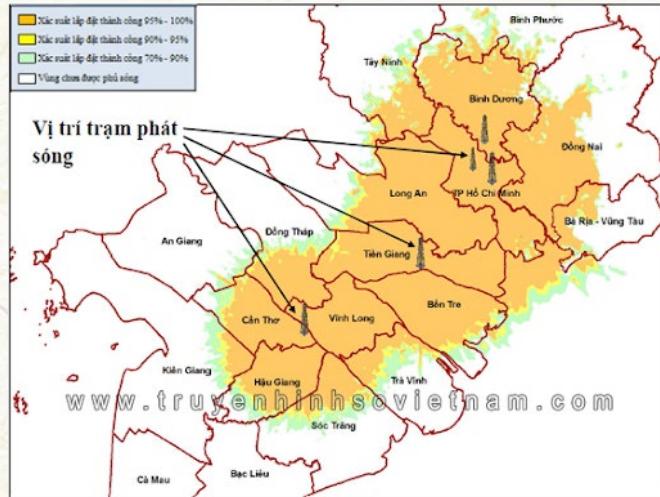
# Cluster analysis

Cây tối đại nhỏ nhất được ứng dụng trong các phương pháp gom nhóm như single-linkage clustering, graph-theoretic clustering, clustering gene expression data.

# Ứng dụng

## Xây dựng trạm phát sóng

Ứng dụng cây tối đai ngắn nhất để có thể đặt các trạm phát sóng sau cho có thể phủ sóng toàn bộ khu vực.



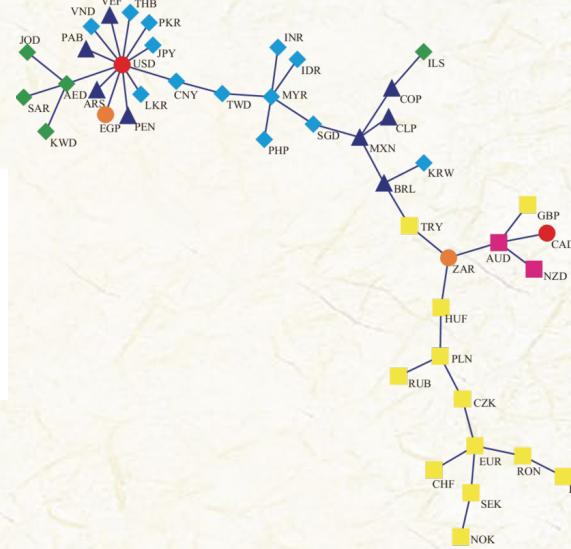
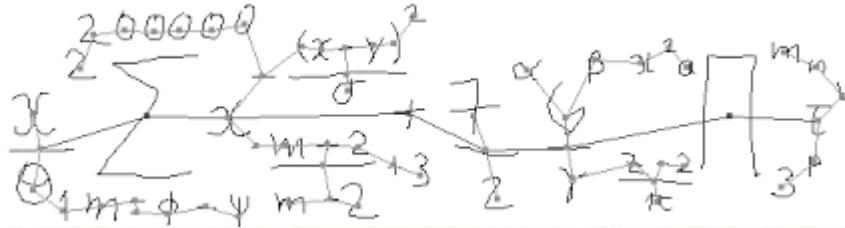
## Xây dựng mạng lưới điện

Cây tối đai nhỏ nhất được ứng dụng để xây dựng mạng lưới điện ít tốn chi phí nhất nhưng vẫn cung cấp điện đủ cho khu vực

# Ứng dụng

## Nhận biết ký tự viết tay của những biểu thức toán học

Dùng cây tối đại ngắn nhất để phân tích bối cục của biểu thức và nhận dạng chúng.



## Ứng dụng trong thị trường tài chính

Ứng dụng để chỉ ra mối tương quan của hàng hóa hay tiền tệ

Do có nhiều ứng dụng nên việc tìm ra  
cây tối đai ngắn nhất là rất cần thiết.  
Vậy để tìm ra cây tối đai ngắn nhất thì  
phải làm thế nào?

### **III. Thuật toán Prims**

**Ý tưởng và ứng dụng**

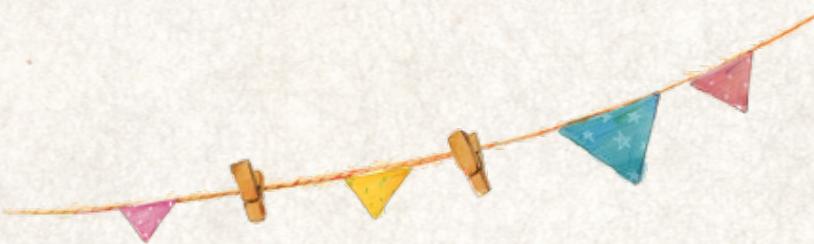


# Giới Thiệu

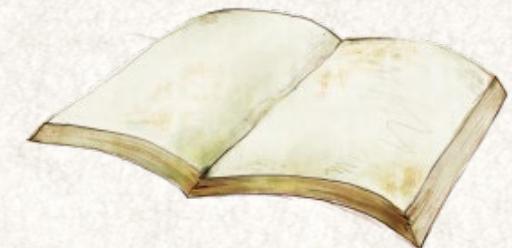


Thuật toán Prim hay còn gọi là thuật toán Jarnik được phát triển vào năm 1930 bởi nhà toán học người Czech Vojtěch Jarník sau đó được tái khám phá và công bố lại bởi nhà khoa học máy tính Robert C. Prim vào 1957.





# Khái niệm



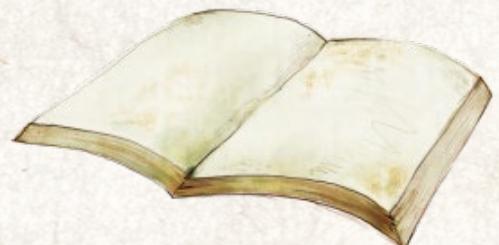
Trong khoa học máy tính, thuật toán Prim là một thuật toán tham lam để tìm cây bao trùm nhỏ nhất của một đồ thị vô hướng có trọng số liên thông. Nghĩa là nó tìm một tập hợp các cạnh của đồ thị tạo thành một cây chứa tất cả các đỉnh, sao cho tổng trọng số các cạnh của cây là nhỏ nhất.



# Ý tưởng

Thuật toán xuất phát từ một cây chỉ chứa đúng một đỉnh và mở rộng từng bước một, mỗi bước thêm một cạnh mới vào cây, cho tới khi bao trùm được tất cả các đỉnh của đồ thị.

- **Dữ liệu vào:** Một đồ thị có trọng số liên thông với tập hợp đỉnh  $V$  và tập hợp cạnh  $E$  (trọng số có thể âm). Đồng thời cũng dùng  $V$  và  $E$  để ký hiệu số đỉnh và số cạnh của đồ thị.
- **Khởi tạo:**  $V_{\text{mới}} = \{x\}$ , trong đó  $x$  là một đỉnh bất kì (đỉnh bắt đầu) trong  $V$ ,  $E_{\text{mới}} = \{\}$
- **Lặp lại** cho tới khi  $V_{\text{mới}} = V$ :
  - Chọn cạnh  $(u, v)$  có trọng số nhỏ nhất thỏa mãn  $u$  thuộc  $V_{\text{mới}}$  và  $v$  không thuộc  $V_{\text{mới}}$  (nếu có nhiều cạnh như vậy thì chọn một cạnh bất kì trong chúng)
  - Thêm  $v$  vào  $V_{\text{mới}}$ , và thêm cạnh  $(u, v)$  vào  $E_{\text{mới}}$
- **Dữ liệu ra:**  $V_{\text{mới}}$  và  $E_{\text{mới}}$  là tập hợp đỉnh và tập hợp cạnh của một cây bao trùm nhỏ nhất

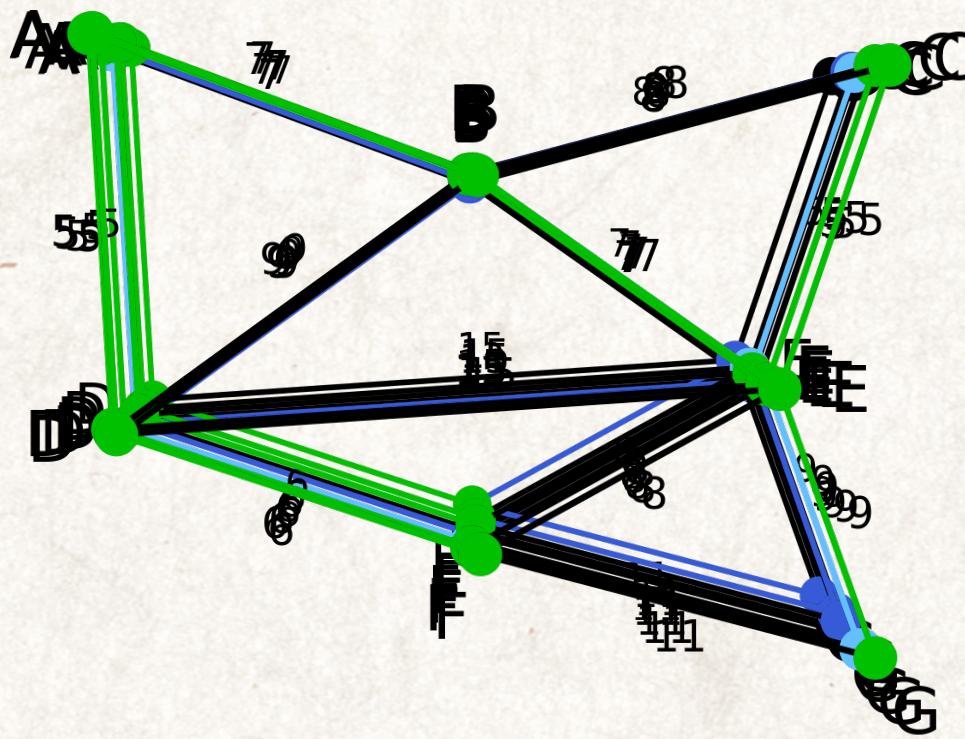


# Độ phức tạp

Một cách lập trình đơn giản sử dụng ma trận kề và tìm kiếm toàn bộ mảng để tìm cạnh có trọng số nhỏ nhất có thời gian chạy  $O(V^2)$ . Bằng cách sử dụng cấu trúc dữ liệu đống nhị phân và danh sách kề, có thể giảm thời gian chạy xuống  $O(E \log V)$ . Bằng cách sử dụng cấu trúc dữ liệu đống Fibonacci phức tạp hơn, có thể giảm thời gian chạy xuống  $O(E + V \log V)$ , nhanh hơn thuật toán trước khi đồ thị có số cạnh  $E = \omega(V)$ .



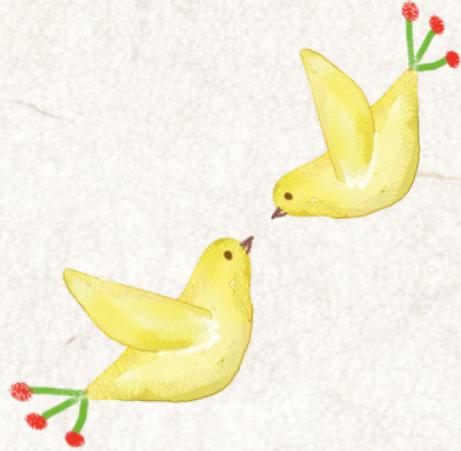
# Minh họa



# **Ưu và nhược điểm**

## **Ưu điểm**

Đối với đồ thị có nhiều cạnh thì thuật toán Prims chiếm ưu thế về mặt tốc độ



## **Nhược điểm**

Nhưng đối với một đồ thị không liên thông thì thuật toán này chịu thua

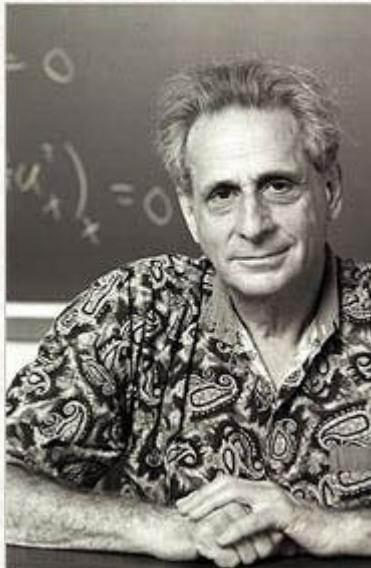
## IV. Thuật toán Kruskal

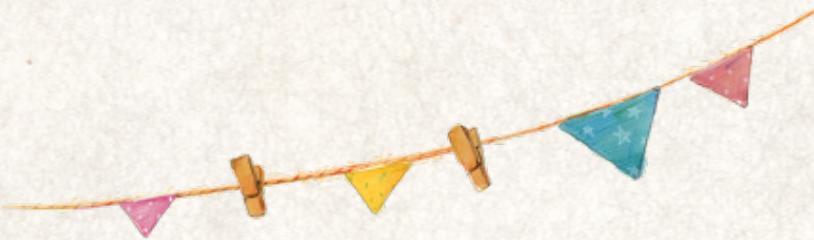
Ý tưởng và ứng dụng



# Giới Thiệu

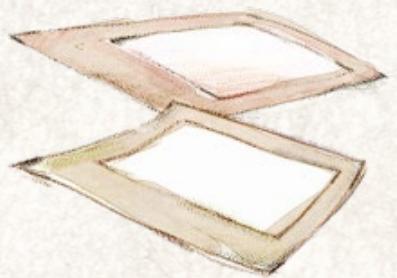
Thuật toán này xuất bản lần đầu tiên năm 1956 trong cuốn **Proceedings of the American Mathematical Society**, bởi Joseph Kruskal





## Khái niệm

Thuật toán Kruskal là một thuật toán trong lý thuyết đồ thị để tìm cây bao trùm nhỏ nhất của một đồ thị liên thông có trọng số. Nói cách khác, nó tìm một tập hợp các cạnh tạo thành một cây chứa tất cả các đỉnh của đồ thị và có tổng trọng số các cạnh là nhỏ nhất. Thuật toán Kruskal là một ví dụ của thuật toán tham lam.



# Ý tưởng

Thuật toán Kruskal dựa trên mô hình xây dựng cây khung nhỏ nhất bằng t  
huật toán hợp nhất.

Thuật toán không xét các cạnh với thứ tự tuỳ ý.

Thuật toán xét các cạnh theo thứ tự đã sắp xếp theo trọng số.

Để xây dựng tập  $n-1$  cạnh của cây khung nhỏ nhất – tạm gọi là tập K, Kruska  
l đề nghị cách kết nạp lần lượt các cạnh vào tập đó theo nguyên tắc như sau:

- Ưu tiên các cạnh có trọng số nhỏ hơn.
  - Kết nạp cạnh khi nó không tạo chu trình với tập cạnh đã kết nạp trước đó.
- Đó là một nguyên tắc chính xác và đúng đắn, đảm bảo tập K nếu thu đủ  $n - 1$  cạnh sẽ là cây khung nhỏ nhất.



# Ý tưởng

Giả sử ta cần tìm cây bao trùm nhỏ nhất của đồ thị  $G$ . Thuật toán bao gồm các bước sau.

- Khởi tạo rừng  $F$  (tập hợp các cây), trong đó mỗi đỉnh của  $G$  tạo thành một cây riêng biệt
- Khởi tạo tập  $S$  chứa tất cả các cạnh của  $G$
- Chừng nào  $S$  còn khác rỗng và  $F$  gồm hơn một cây
  - Xóa cạnh nhỏ nhất trong  $S$
  - Nếu cạnh đó nối hai cây khác nhau trong  $F$ , thì thêm nó vào  $F$  và hợp hai cây kề với nó làm một
    - Nếu không thì loại bỏ cạnh đó.

Khi thuật toán kết thúc, rừng chỉ gồm đúng một cây và đó là một cây bao trùm nhỏ nhất của đồ thị  $G$ .



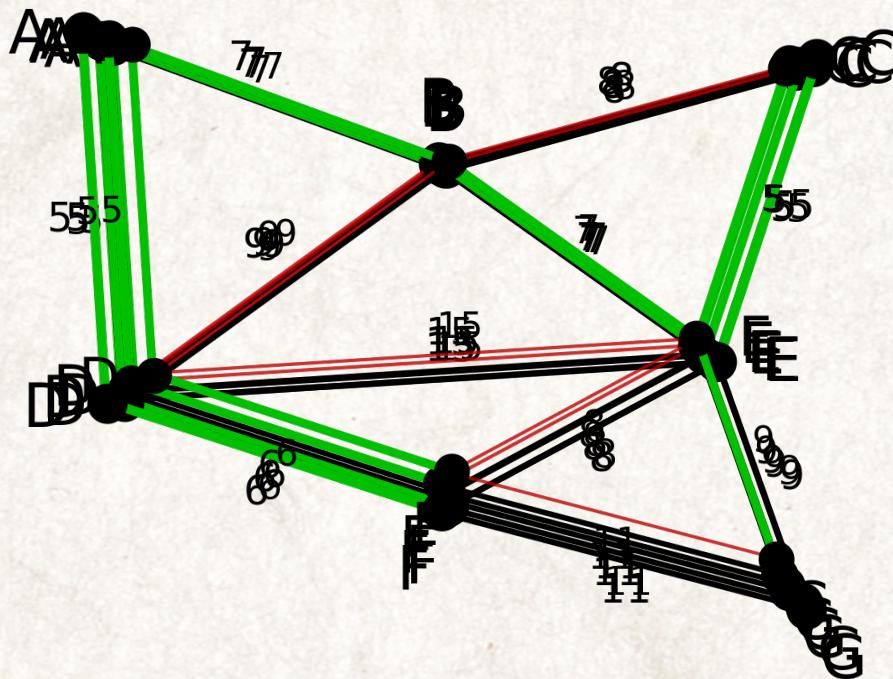
# Độ phức tạp

Nếu  $E$  là số cạnh và  $V$  là số đỉnh của đồ thị thì thuật toán Kruskal chạy trong thời gian  $O(E \log V)$ .

Có thể đạt được thời gian này bằng phương pháp sau: sắp xếp tất cả các cạnh theo trọng số trong thời gian  $O(E \log E)$ . Điều này cho phép thực hiện bước "xóa cạnh nhỏ nhất trong  $S$ " trong thời gian hằng số. Sau đó sử dụng cấu trúc dữ liệu cho các tập hợp không giao nhau để lưu trữ thông tin đỉnh nào nằm ở cây nào trong  $F$ . Ta cần thực hiện  $O(E)$  thao tác, hai thao tác 'tìm' và không quá một thao tác 'hợp' cho mỗi cạnh. Ngay cả những thuật toán đơn giản cho bài toán này, chẳng hạn hợp bằng trọng số cũng có thể thực hiện  $O(E)$  thao tác trong thời gian  $O(E \log V)$ . Vì vậy tổng thời gian là  $O(E \log E) = O(E \log V)$ .



# Minh họa



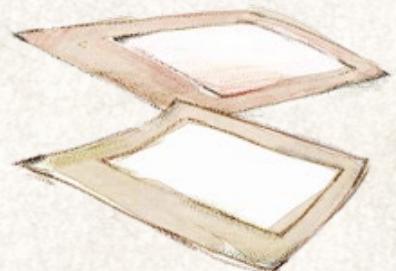
# **Ưu và nhược điểm**

## **Ưu điểm**

Đối với đồ thị có nhiều đỉnh thì thuật toán Kruskal chiếm ưu thế về tốc độ  
Giải được cả khi đồ thị không liên thông

## **Nhược điểm**

Cài đặt khó khăn do phải kiểm tra xem cạnh có nằm trong chu trình hay không



# Nguồn tài liệu tham khảo

<https://www.slideshare.net/TieuCom8800/l-40456453>

[https://vi.wikipedia.org/wiki/L%C3%BD\\_thuy%E1%BA%BFt\\_%C4%91%E1%BB%93\\_th%E1%BB%8B](https://vi.wikipedia.org/wiki/L%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)

<https://vi.mathigon.org/course/graph-theory/applications>

[https://vi.wikipedia.org/wiki/C%C3%A2y\\_t%E1%BB%91i\\_%C4%91%E1%BA%A1i](https://vi.wikipedia.org/wiki/C%C3%A2y_t%E1%BB%91i_%C4%91%E1%BA%A1i)

<https://vietcodes.github.io/algo/mst>

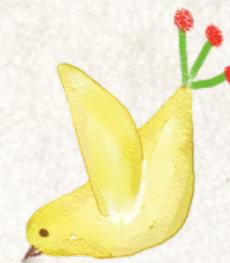
<https://www.baeldung.com/cs/kruskals-vs-prims-algorithm#:~:text=The%20advantage%20of%20Prim's%20algorithm,with%20the%20same%20weight%20occur.>



**Q & A**



# Demo





# **THANK YOU**

Cảm ơn thầy cô và các bạn đã lắng nghe

