COPYRIGHT NOTICE
THÔNG BÁO BẢN QUYỀN

COPYRIGHT (English):

This document is not up to date and may contain several errors or outdated information.

Last revision date: 2024-04-13

BẢN QUYỀN (Tiếng Việt):

Tài liệu này không được cập nhật và có thể chứa nhiều lỗi hoặc thông tin cũ.

Ngày sửa đổi cuối cùng: 2024-04-13

VNU-HUS MAT3500: Toán rời rạc

Lý thuyết đồ thị III
Cây

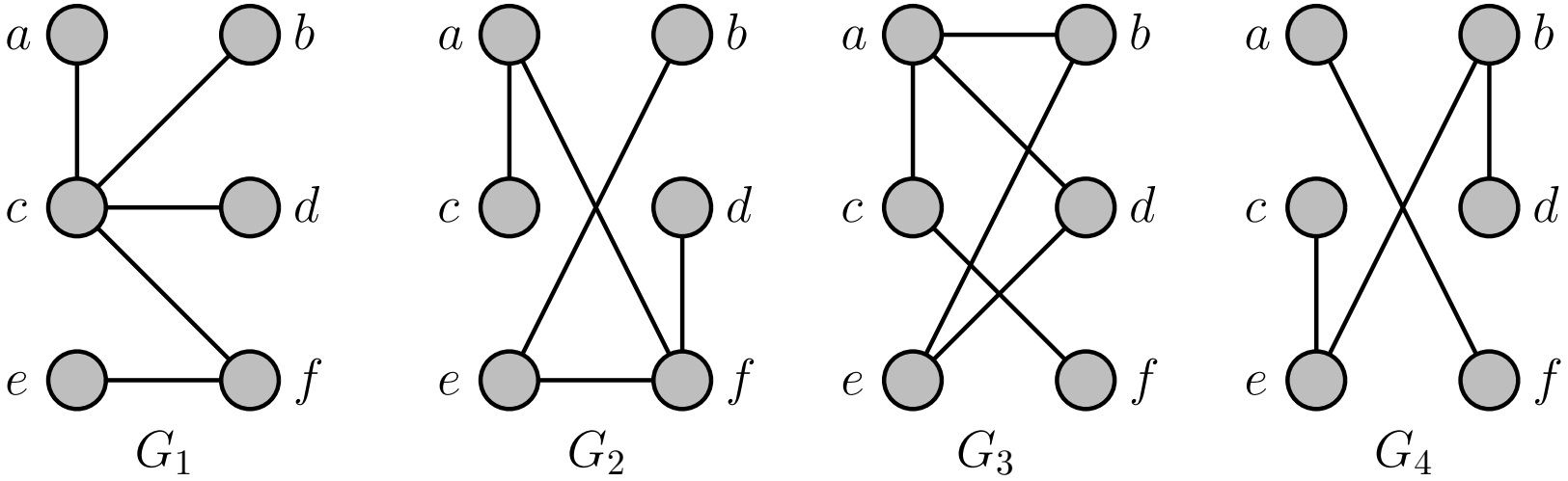
Hoàng Anh Đức

Bộ môn Tin học, Khoa Toán-Cơ-Tin học
Đại học KHTN, ĐHQG Hà Nội
hoanganhduc@hus.edu.vn

Nội dung

Cây
Giới thiệu

- Đơn đồ thị vô hướng $T = (V, E)$ được gọi là một *cây (tree)* nếu T là đồ thị liên thông và không có chu trình
- Đơn đồ thị vô hướng $T = (V, E)$ được gọi là một *rừng (forest)* nếu T là đồ thị không có chu trình



Hình: G_1, G_2 là cây. G_3, G_4 không là cây. G_4 là rừng. G_3 không là rừng

Định lý 1

Một đồ thị vô hướng G là một cây khi và chỉ khi giữa hai đỉnh bất kỳ của G tồn tại một đường đi đơn duy nhất

Chứng minh.

($\Rightarrow$) Giả sử G là cây

- Do G liên thông, với hai đỉnh bất kỳ $u, v \in V$, *tồn tại* một đường đi đơn giữa u và v
- Ta chứng minh đường đi này là *duy nhất*. Giả sử phản chứng rằng có ít nhất hai đường đi đơn $P = v_0, v_1, \ldots, v_k$ và $Q = w_0, w_1, \ldots, w_\ell$ khác nhau giữa hai đỉnh u và v, với $v_0 = w_0 = u$ và $v_k = w_\ell = v$
  - Do $v_0 = w_0$, tồn tại i thỏa mãn $v_j = w_j$ với mọi $0 \leq j \leq i$ và $v_{i+1} \neq w_{i+1}$
  - Do $v_k = w_\ell$, tồn tại $p \geq i + 1$ nhỏ nhất thỏa mãn $v_p \in \{w_i, \ldots, w_\ell\}$ hoặc $w_p \in \{v_i, \ldots, v_k\}$. Giả sử $v_p = w_q$ với $q \geq i$ nhỏ nhất có thể
  - $v_i, v_{i+1}, \ldots, v_p$ và $w_q, \ldots, w_{i+1}, w_i$ tạo thành một chu trình trong G, mâu thuẫn với giả thiết G là cây

Chứng minh (tiếp).

($\Leftarrow$) Giả sử với mọi cặp đỉnh $u, v \in V$, tồn tại một đường đi đơn duy nhất giữa u và v trong G

- Theo định nghĩa, G là *liên thông*
- Ta chứng minh G *không có chu trình*. Giả sử phản chứng rằng tồn tại một chu trình C trong G. Do đó, với hai đỉnh u, v bất kỳ thuộc C, có hai đường đi đơn khác nhau nối u và v, mâu thuẫn với giả thiết tồn tại duy nhất một đường đi đơn giữa hai đỉnh bất kỳ trong G

□

Định lý 2

Mọi cây $T = (V, E)$ với $|V| \geq 2$ có ít nhất hai đỉnh bậc 1

Chứng minh.

Lấy một đường đi đơn *dài nhất* $v_0, v_1, \ldots, v_m$ trong T. Bậc của v_0 và v_m đều bằng 1, nếu không ta có thể tìm được một đường đi dài hơn □

Định lý 3

Mọi cây gồm n đỉnh có chính xác $n - 1$ cạnh

Chứng minh.

Quy nạp theo n

- **Bước cơ sở:** Với $n = 1$, cây gồm 1 đỉnh có 0 cạnh
- **Bước quy nạp:** Giả sử mọi cây gồm k đỉnh có chính xác $k - 1$ cạnh, với $k \geq 1$. Ta chứng minh mọi cây T gồm $k + 1$ đỉnh có chính xác k cạnh. Thật vậy, do T là một cây, T có một đỉnh u bậc 1 nào đó. $T - u$ là một cây gồm k đỉnh và theo giả thiết quy nạp có $k - 1$ cạnh. Do đó, T có $(k - 1) + 1 = k$ cạnh

□

Bài tập 1

Chứng minh rằng một đồ thị $G = (V, E)$ là một cây khi và chỉ khi G không có chu trình và với mọi cặp đỉnh $u, v \in V$ thỏa mãn $uv \notin E$, đồ thị $G + uv$ có chính xác một chu trình

Bài tập 2

Giả sử một cây $T = (V, E)$ có số các cạnh là một số chẵn. Chứng minh rằng tồn tại một đỉnh trong T có bậc chẵn

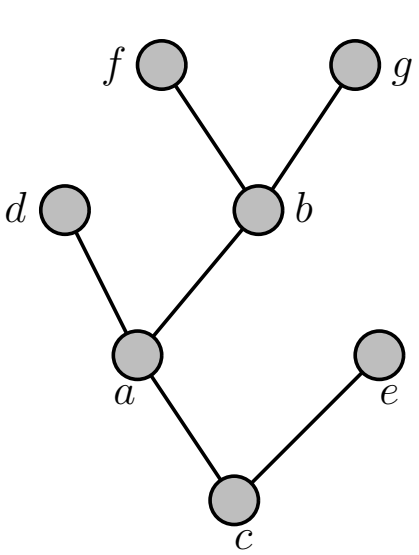
Bài tập 3

Cho $T = (V, E)$ là một cây. Gọi Δ là bậc lớn nhất của một đỉnh trong T, nghĩa là, $\Delta = \max_{v \in V} \deg_T(v)$. Chứng minh rằng T có ít nhất Δ đỉnh bậc 1

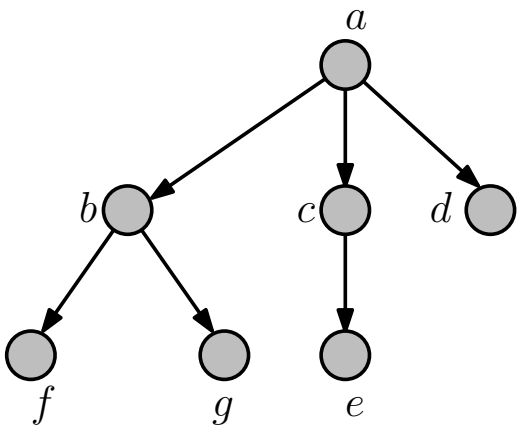
Bài tập 4

Cho $T = (V, E)$ là một cây với $|V| > 1$. Giả sử nếu v là một đỉnh liền kề với một đỉnh bậc 1 trong T thì $\deg_T(v) \geq 3$. Chứng minh rằng tồn tại hai đỉnh bậc 1 trong T có chung một đỉnh liền kề với hai đỉnh đó

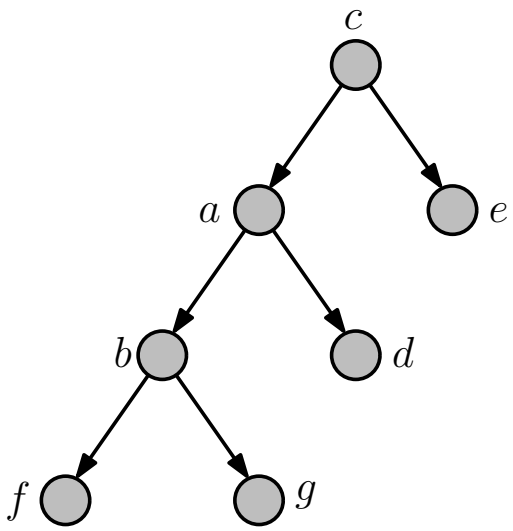
Một *cây có gốc (rooted tree)* là một cây trong đó một đỉnh được coi là *đỉnh gốc (root)* và mọi cạnh được định hướng xa khỏi đỉnh gốc. Ta ký hiệu một cây T với gốc r bằng cặp (T, r)



Cây T



Cây T với gốc a



Cây T với gốc c

Thông thường, một cây có gốc (T, r) được vẽ với *đỉnh gốc ở trên đỉnh của đồ thị*. Khi đó, ta có thể *bỏ qua các hướng của các cạnh*, do việc lựa chọn đỉnh gốc đã xác định các hướng này

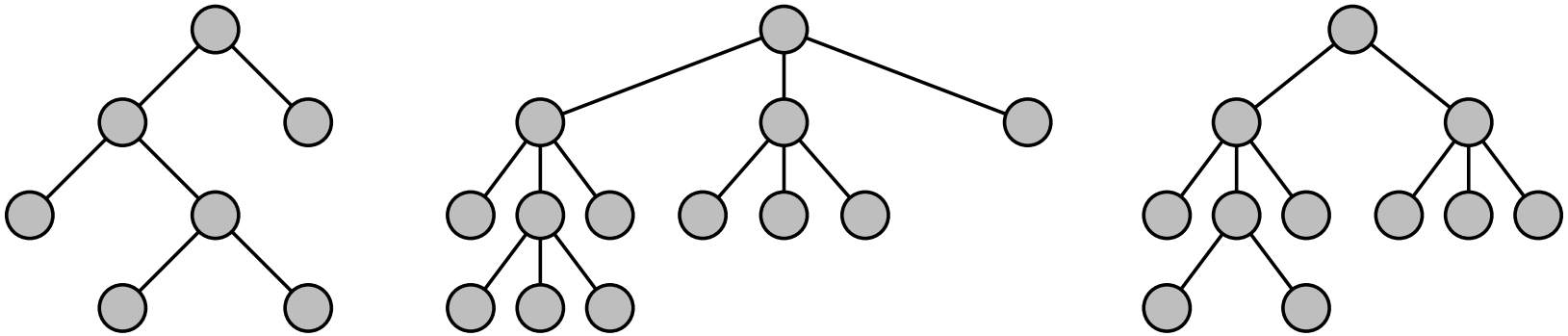
Cây
Cây có gốc

- Với một đỉnh v khác đỉnh gốc của một cây có gốc T
  - *Đỉnh cha (parent)* của v là đỉnh u duy nhất sao cho có một cạnh có hướng từ u đến v. Ta cũng gọi v là *đỉnh con (child)* của u.
  - Đỉnh v được gọi là *đỉnh lá (leaf)* nếu v không có đỉnh con. Các đỉnh có đỉnh con được gọi là *các đỉnh trong (internal vertices)*
  - Các đỉnh có chung một đỉnh cha được gọi là *các đỉnh anh em (siblings)*
  - *Các đỉnh tổ tiên (ancestors)* của v là các đỉnh trên đường đi từ gốc tới đỉnh đó. *Các đỉnh con cháu (descendants)* của v là các đỉnh có v là một đỉnh tổ tiên
  - *Cây con (subtree) với gốc v* là đồ thị con của T cảm sinh bởi v và các đỉnh con cháu của nó
- Đỉnh gốc là đỉnh duy nhất trong cây *không có đỉnh cha* và là *tổ tiên của mọi đỉnh còn lại*. Đỉnh gốc luôn là một đỉnh trong, trừ trường hợp cây chỉ có một đỉnh duy nhất là đỉnh gốc thì đỉnh gốc là một đỉnh lá

- Một cây có gốc T được gọi là một *cây m-phân (m-ary tree)* nếu tất cả các đỉnh trong của T có tối đa là m đỉnh con. Nếu như mỗi đỉnh trong của T có chính xác m đỉnh con, ta gọi T là một *cây m-phân đầy đủ (full m-ary tree)*
- Cây m-phân với $m = 2$ được gọi là *cây nhị phân (binary tree)*

Bài tập 5

Trong số các cây sau, với m là số nguyên dương nào đó, cây nào là cây m-phân? Cây nào là cây m-phân đầy đủ?



Cây
Cây có gốc

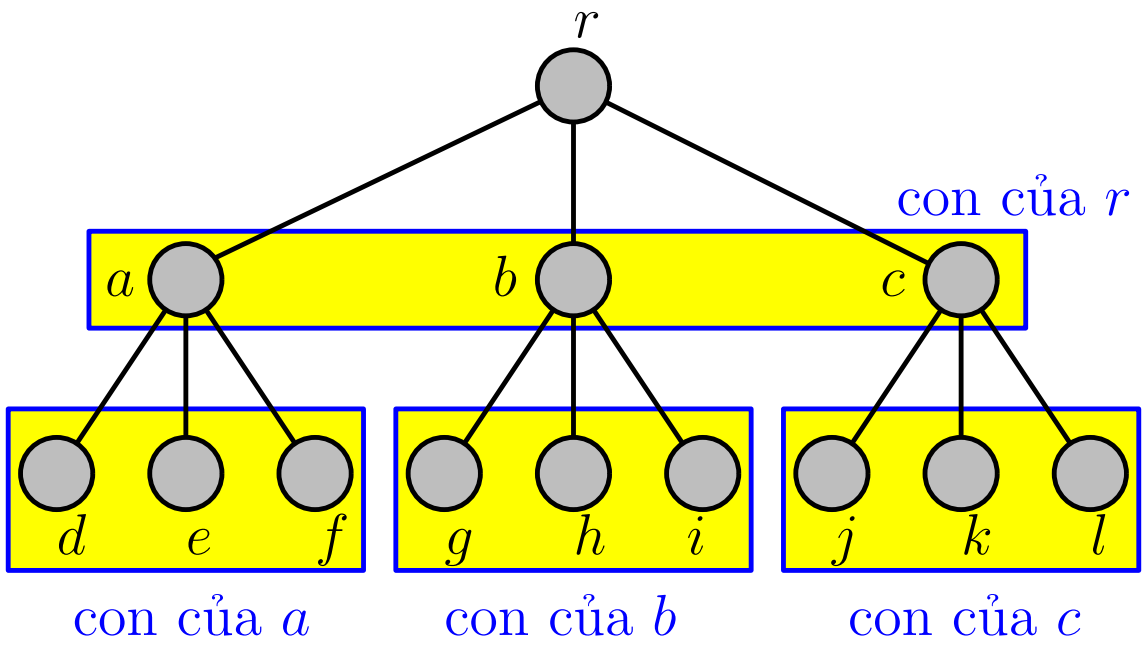
- Một *cây có gốc được sắp thứ tự (ordered rooted tree)* là một cây có gốc trong đó các đỉnh con của mỗi đỉnh được sắp xếp theo một thứ tự nào đó
- Ta thường vẽ cây có gốc được sắp thứ tự theo cách thông thường với hiểu ngầm rằng *các đỉnh con của mỗi đỉnh được sắp xếp theo thứ tự từ trái sang phải*
- Với một cây nhị phân được sắp thứ tự, nếu một đỉnh trong u có hai đỉnh con thì đỉnh thứ nhất được gọi là *đỉnh con trái (left child)*, đỉnh thứ hai được gọi là *đỉnh con phải (right child)*. Tương tự, cây con có gốc là đỉnh con trái được gọi là *cây con trái (left subtree)* và cây có gốc là đỉnh con phải được gọi là *cây con phải (right subtree)* của u
- Các cây có gốc sắp thứ tự được ứng dụng rộng rãi trong khoa học máy tính, ví dụ như các cây phân tích cú pháp (parse trees), cấu trúc cây trong các tài liệu XML (XML documents), cấu trúc cây mô tả các thư mục và tệp (directories and file system), v.v...

Định lý 4

Với mọi $m \geq 1$, mọi cây m-phân đầy đủ với i đỉnh trong có chính xác $n = m \cdot i + 1$ đỉnh

Chứng minh.

Mọi đỉnh khác đỉnh gốc là con của một đỉnh trong nào đó. Do đó có $m \cdot i$ đỉnh con (của các đỉnh trong) và 1 đỉnh gốc



- *Mức (level)* của một đỉnh trong một cây có gốc là độ dài của đường đi duy nhất từ gốc đến đỉnh đó. Mức của đỉnh gốc là 0
- *Độ cao (height)* h của một cây có gốc là độ dài lớn nhất của một đường đi từ gốc đến đỉnh lá. Nói cách khác, độ cao của một cây là mức lớn nhất của một đỉnh trong cây đó
- Một cây m-phân có độ cao h được gọi là *cân đối (balanced)* nếu tất cả các đỉnh lá có mức $h - 1$ hoặc h

Định lý 5

Với mọi $m \geq 1$, một cây m-phân đầy đủ với

(i) *n đỉnh có $i = (n-1)/m$ đỉnh trong và $l = ((m-1)n+1)/m$ đỉnh lá;*

(ii) *i đỉnh trong có $n = mi+1$ đỉnh và $l = (m-1)i+1$ đỉnh lá;*

(iii) *l đỉnh lá có $n = (ml-1)/(m-1)$ đỉnh và $i = (l-1)/(m-1)$ đỉnh trong*

Bài tập 6

Sử dụng Định lý 4, hãy chứng minh Định lý 5

Bài tập 7

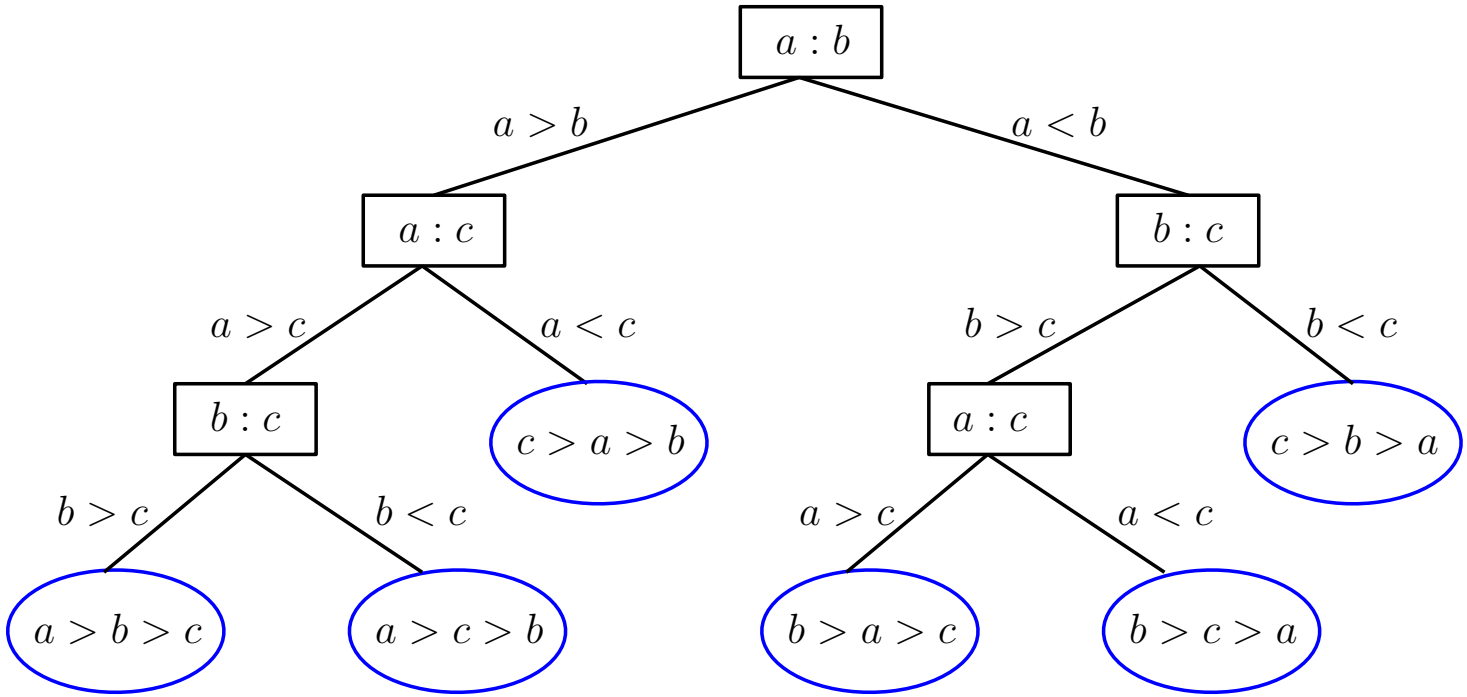
*Chứng minh rằng với mọi $m \geq 1$, có nhiều nhất m^h đỉnh lá trong một cây m-phân có độ cao h. (**Gợi ý:** Quy nạp theo h)*

Một ứng dụng của cây nhị phân được thể hiện trong định lý sau

Định lý 6

Mọi thuật toán sắp xếp n phần tử nào đó dựa trên các phép so sánh hai phần tử bất kỳ cần sử dụng ít nhất $\lceil \log_2(n!) \rceil$ phép so sánh trong trường hợp xấu nhất

- Do $\log_2(n!) = \Theta(n \log n)$, *một thuật toán sắp xếp n phần tử trong trường hợp xấu nhất cần $\Theta(n \log n)$ phép so sánh là một thuật toán tối ưu*, theo nghĩa là không tồn tại thuật toán sắp xếp nào khác có thời gian chạy trong thời gian xấu nhất tốt hơn
- **Ý tưởng:** Mô hình các thuật toán bằng cây nhị phân đầy đủ với *các đỉnh trong là các phép so sánh* và *các đỉnh lá là các kết quả thu được* khi đi từ gốc (gọi là *cây quyết định (decision tree)*). *Số phép so sánh* sử dụng để ra một kết quả chính là *độ dài đường đi từ gốc đến nút lá tương ứng*

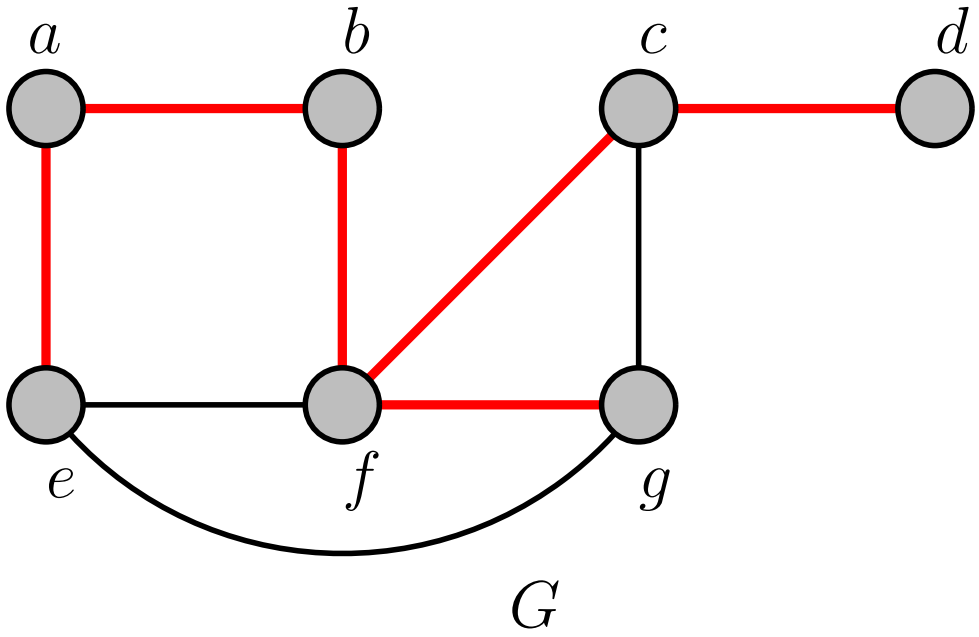


Hình: Mô hình thuật toán sắp xếp dãy ba phần tử a, b, c bằng cây quyết định

Bài tập 8

Với mọi $m \geq 1$, chứng minh rằng trong một cây m-phân có l đỉnh lá và có độ cao h, ta có $h \geq \lceil \log_m l \rceil$

Cho $G = (V, E)$ là một đơn đồ thị. Một *cây bao trùm (spanning tree)* của G là một đồ thị con T của G thỏa mãn điều kiện T là một cây và T chứa tất cả các đỉnh của G



G

Định lý 7

Mọi đơn đồ thị liên thông G có một cây bao trùm

Chứng minh.

Ta xây dựng một cây bao trùm của G như sau

- Lấy một chu trình trong G nếu có
- Xóa một cạnh từ chu trình đó. Đồ thị mới thu được vẫn là liên thông
- Lặp lại các bước trên cho đến khi không có chu trình trong G

Do G có hữu hạn số đỉnh, cuối cùng ta thu được một đồ thị liên thông không có chu trình chứa tất cả các đỉnh của G—một cây bao trùm của G □

Cho $G = (V, E)$ là một đơn đồ thị vô hướng liên thông. Ta trình bày thuật toán *duyệt theo chiều sâu (depth-first search, DFS)* để tìm một cây bao trùm của G

Thuật toán 1: Duyệt theo chiều sâu

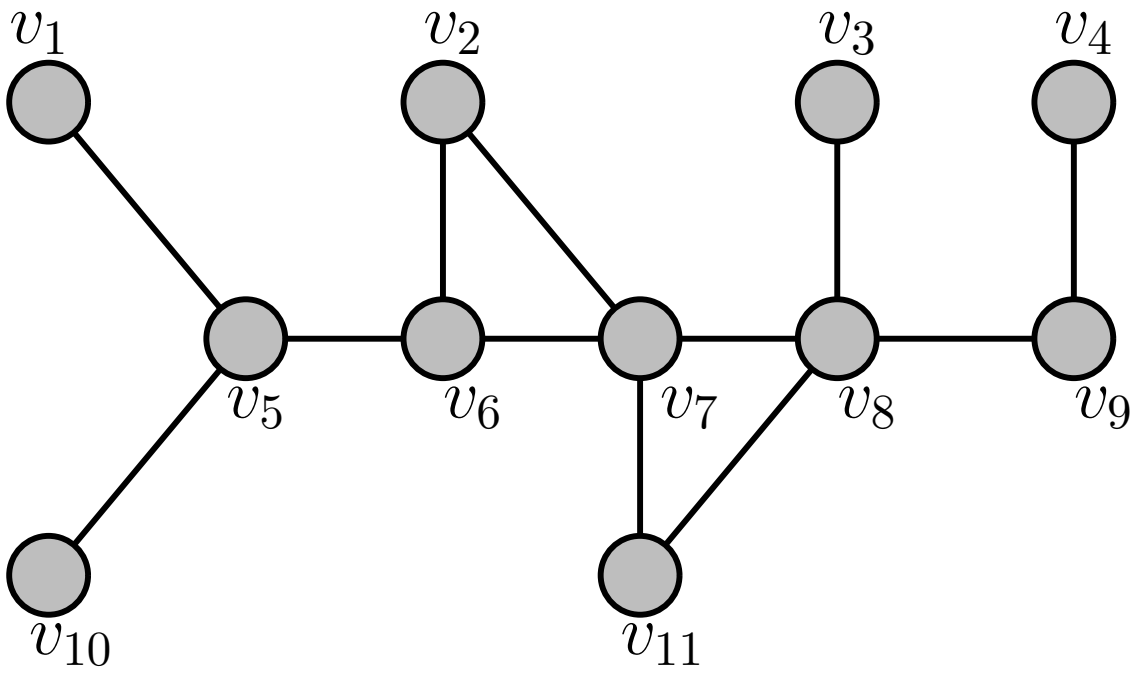
Input: G: đơn đồ thị vô hướng liên thông với các đỉnh $v_1, v_2, \ldots, v_n$

Output: Cây bao trùm T

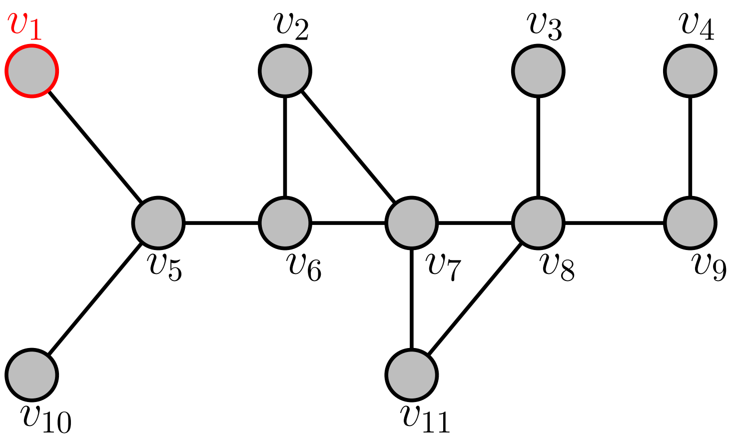
```
1 procedure dfs(G):
2     T := cây có chính xác một đỉnh v_1
3     visit(v_1)
4 procedure visit(v: một đỉnh của G):
5     for mỗi đỉnh w liền kề với v và chưa thuộc T do
6         Thêm w và cạnh vw vào T
7         visit(w)
8 return T
```

Ví dụ 1

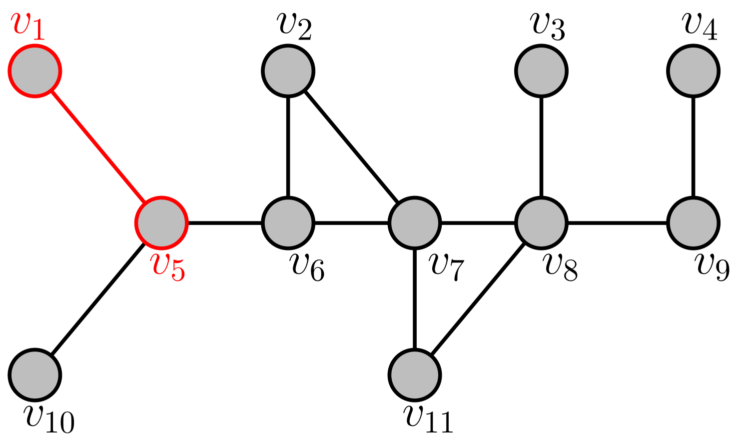
Tìm một cây bao trùm T của đồ thị $G = (V, E)$ sau bằng thuật toán duyệt theo chiều sâu



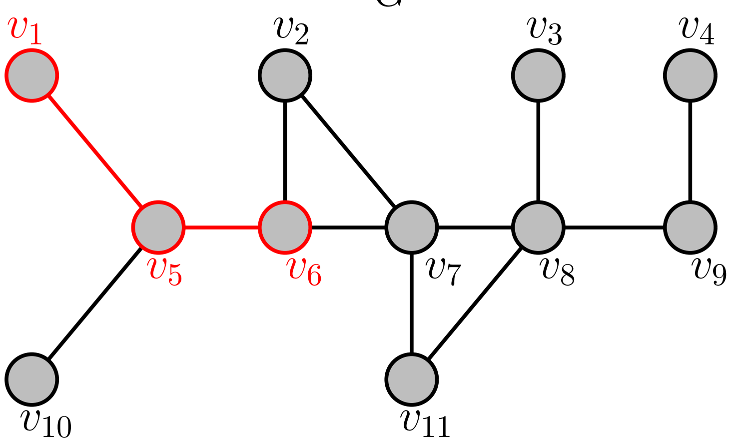
G



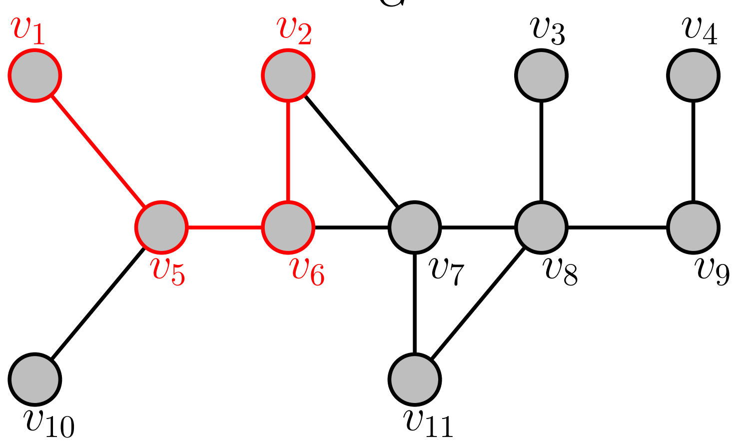
G



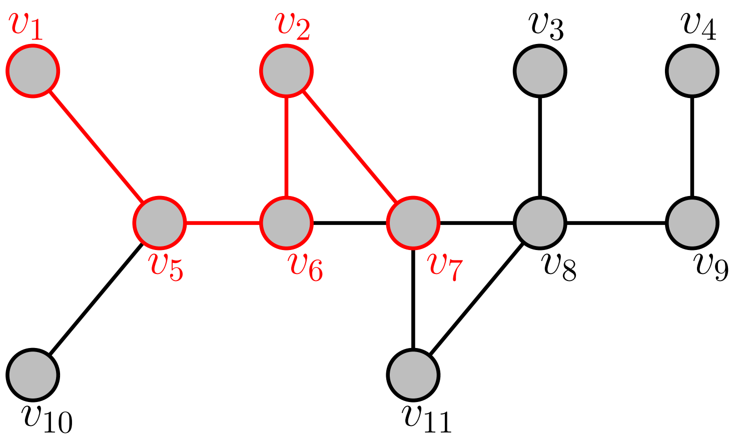
G



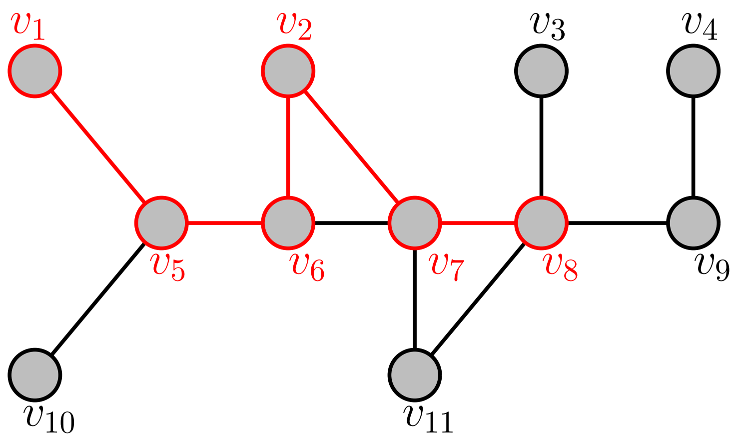
G



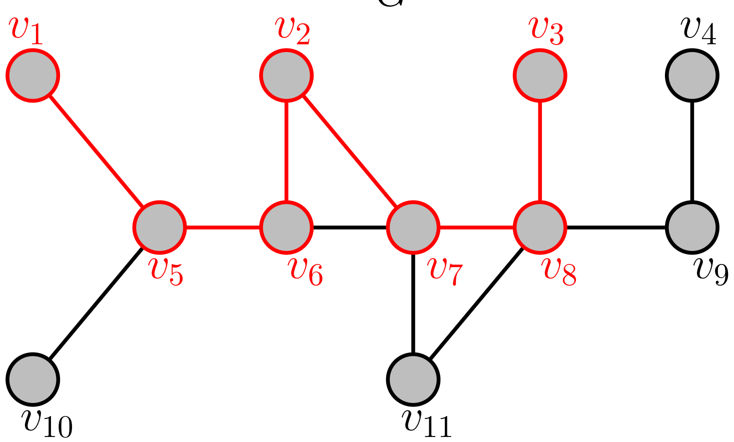
G



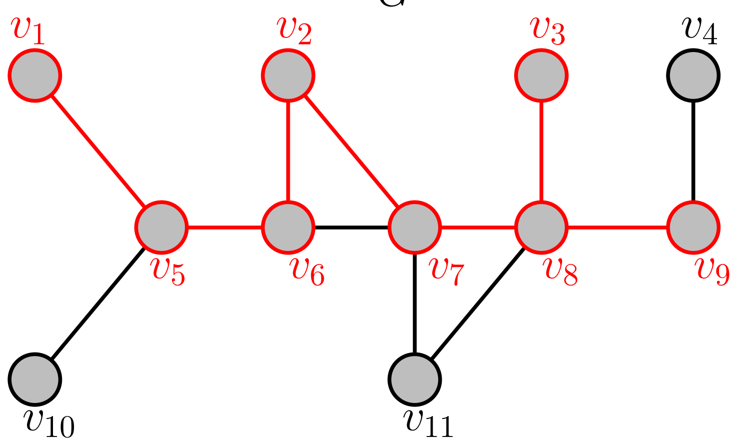
G



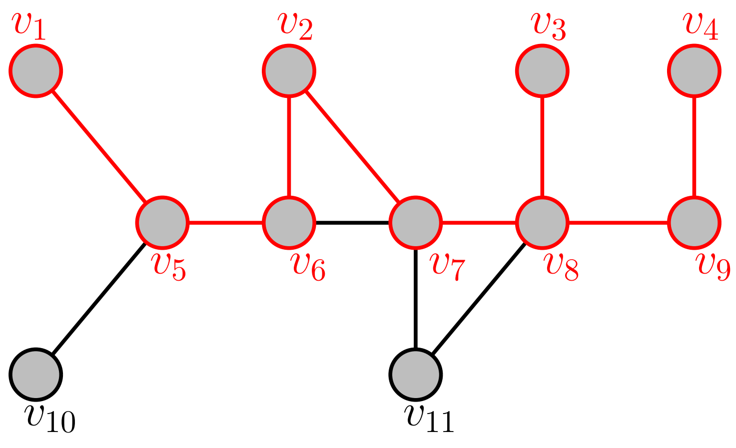
G



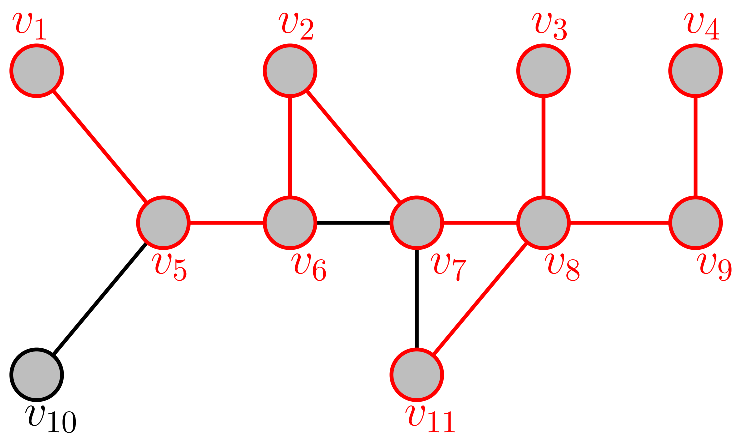
G



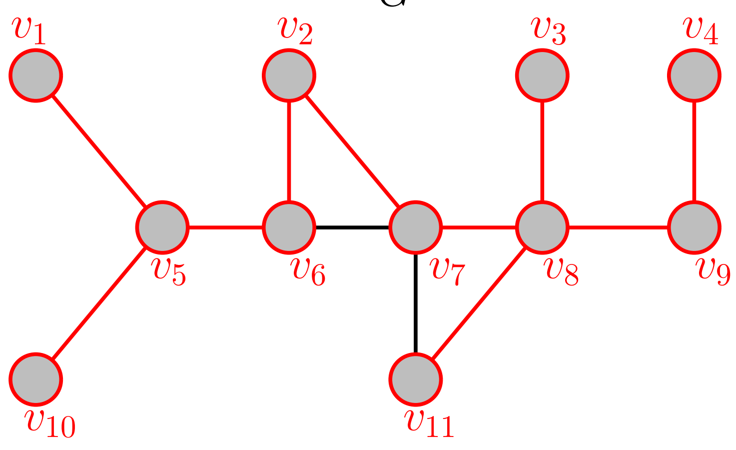
G



G



G



G

Thứ tự các đỉnh được thêm vào T (hay thứ tự các đỉnh thực hiện thủ tục `visit()`) lần lượt là: v_1, v_5, v_6, v_2, v_7, v_8, v_3, v_9, v_4, v_{11}, v_{10}

- Trước mỗi lần lặp trong thủ tục `visit()`, T là cây chứa toàn bộ các đỉnh đã được gọi bởi thủ tục `visit()`
- Thuật toán DFS chạy trong thời gian $O(n^2)$ với đơn đồ thị vô hướng liên thông $G = (V, E)$ bất kỳ, trong đó $n = |V|$
  - `visit()` được gọi cho mỗi đỉnh chính xác một lần
  - Một cạnh e trong G được xét tối đa hai lần để xác định xem e và một đầu mút của e có thể được thêm vào cây T hay không
  - Do đó, thuật toán DFS chạy trong thời gian $O(|E|)$. Chú ý rằng $|E| \leq n(n-1)/2 = O(n^2)$
- Thuật toán DFS có thể được sử dụng như là cơ sở để giải quyết nhiều bài toán khác, ví dụ như bài toán tìm các đường đi và chu trình trong đồ thị, bài toán xác định các thành phần liên thông, bài toán tìm đỉnh cắt, v.v... DFS cũng là cơ sở cho *kỹ thuật quay lui (backtracking technique)* được sử dụng để thiết kế các giải thuật cho nhiều bài toán khó

Cho $G = (V, E)$ là một đơn đồ thị vô hướng liên thông. Ta trình bày thuật toán *duyệt theo chiều rộng (breadth-first search, BFS)* để tìm một cây bao trùm của G

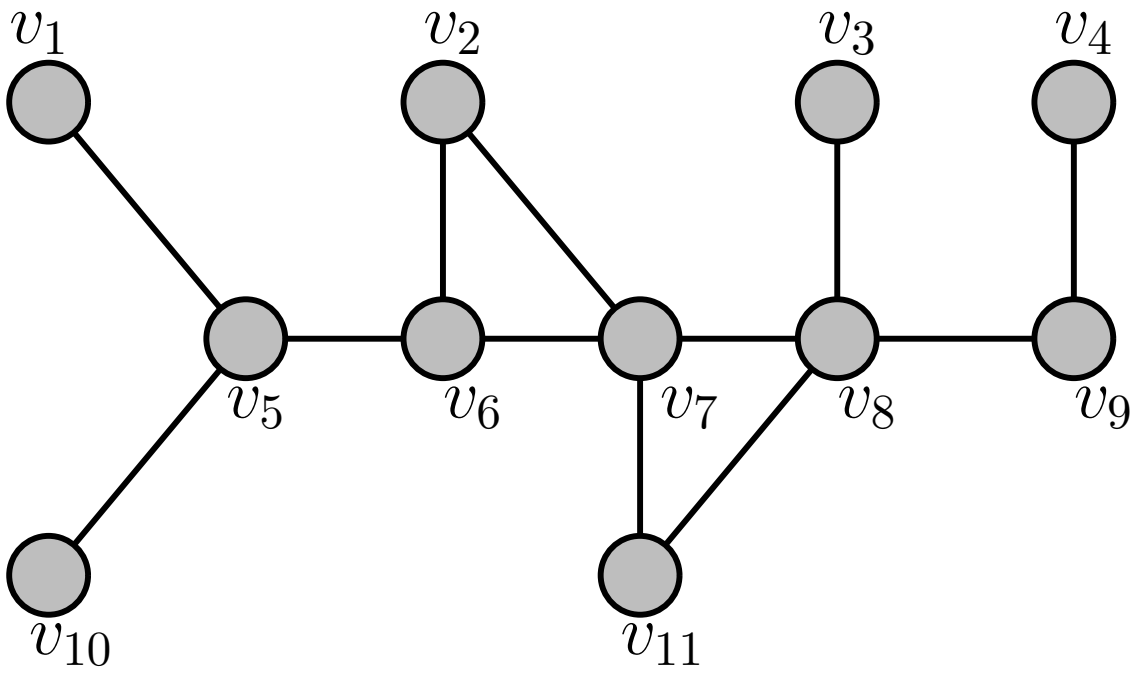
Thuật toán 2: Duyệt theo chiều rộng

Input: G: đơn đồ thị vô hướng liên thông với các đỉnh $v_1, v_2, \ldots, v_n$
Output: Cây bao trùm T

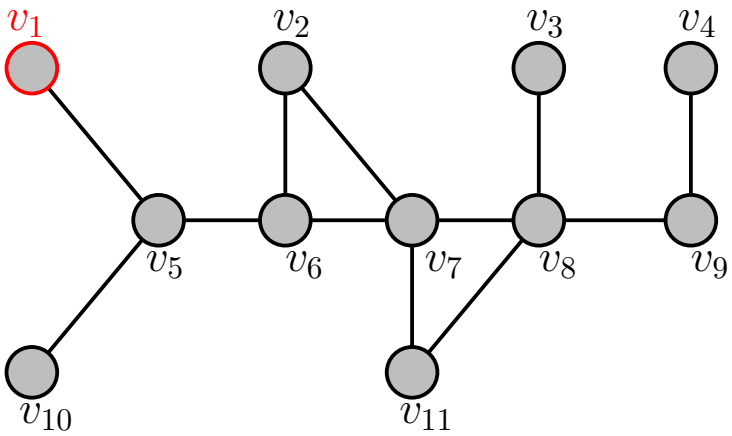
```
1  T := cây chỉ chứa duy nhất một đỉnh v_1
2  L := danh sách rỗng
3  Thêm v_1 vào danh sách L các đỉnh chưa xét
4  while L khác rỗng do
5      Bỏ đi đỉnh thứ nhất v từ L
6      for mỗi đỉnh w liền kề với v do
7          if w không thuộc L và w không thuộc T then
8              Thêm w vào cuối danh sách L
9              Thêm w và cạnh vw vào T
10 return T
```

Ví dụ 2

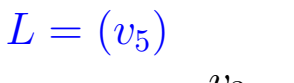
Tìm một cây bao trùm T của đồ thị $G = (V, E)$ sau bằng thuật toán duyệt theo chiều rộng

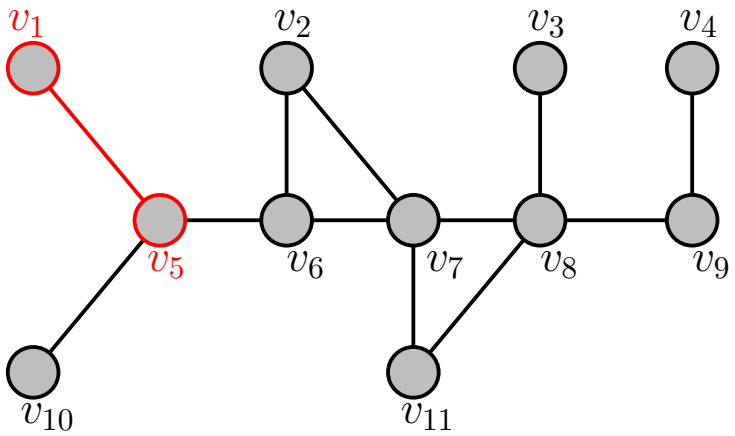


G

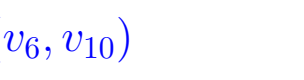


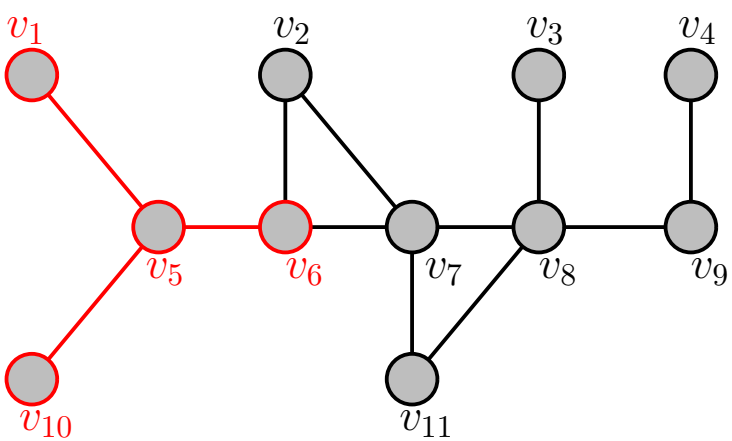
G



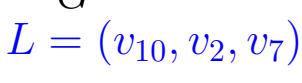


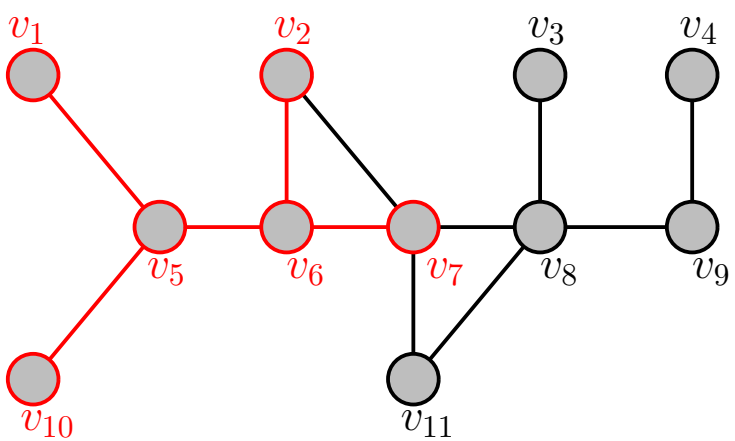
G





G

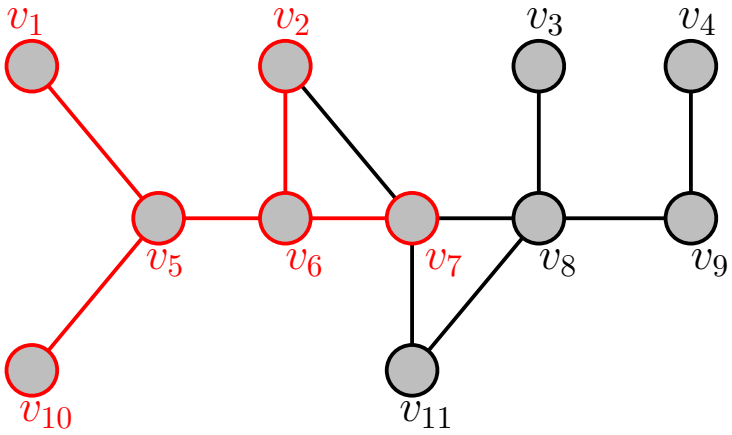




G

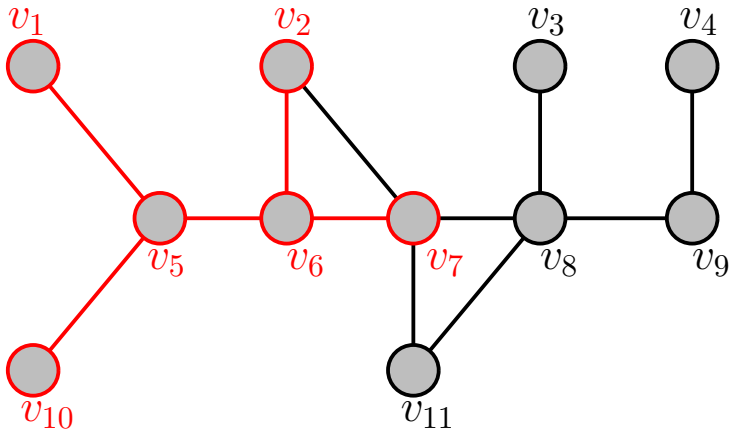
Cây
Cây bao trùm

$L = (v_2, v_7)$



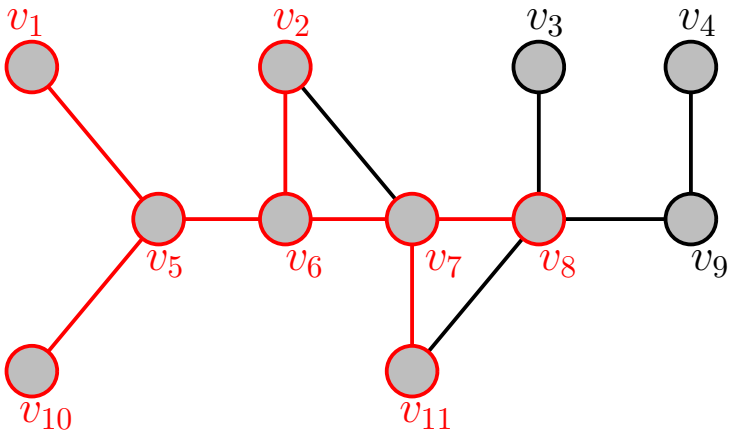
G

$L = (v_7)$



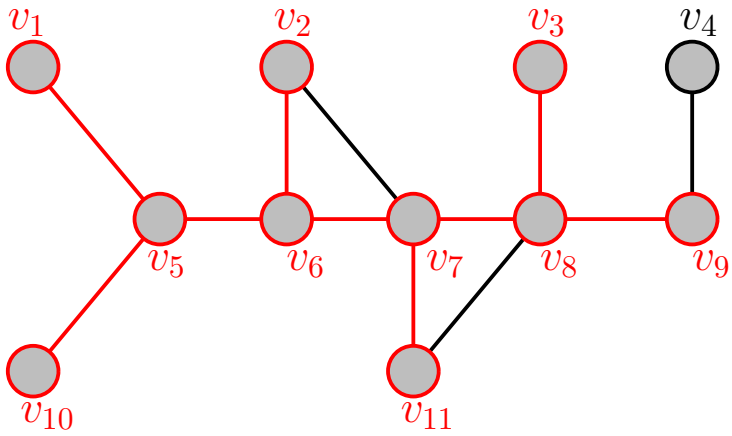
G

$L = (v_8, v_{11})$



G

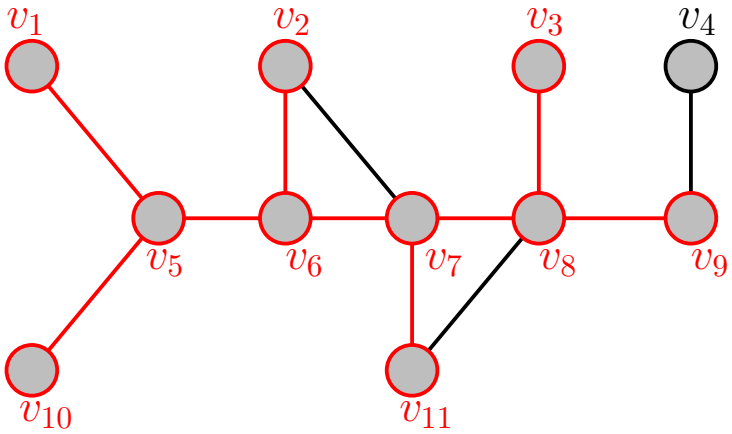
$L = (v_{11}, v_3, v_9)$



G

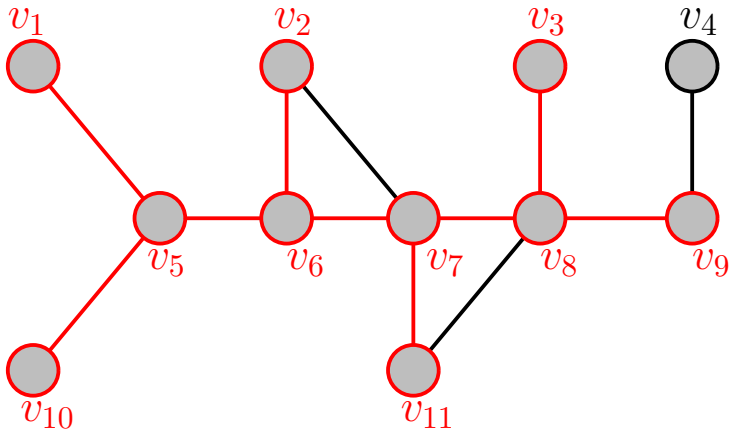
Cây
Cây bao trùm

$L = (v_3, v_9)$



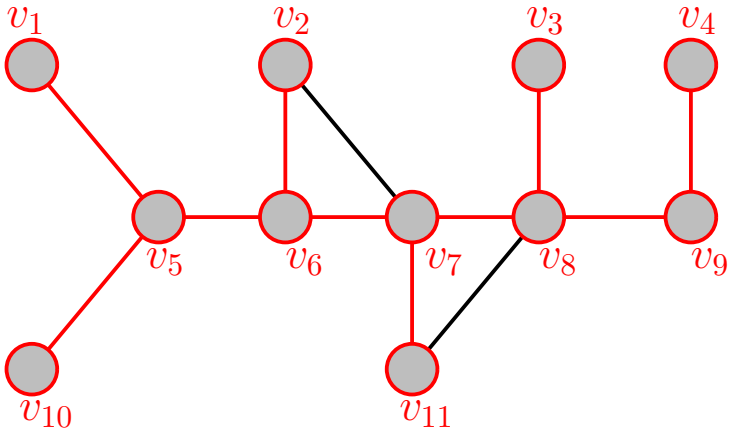
G

$L = (v_9)$



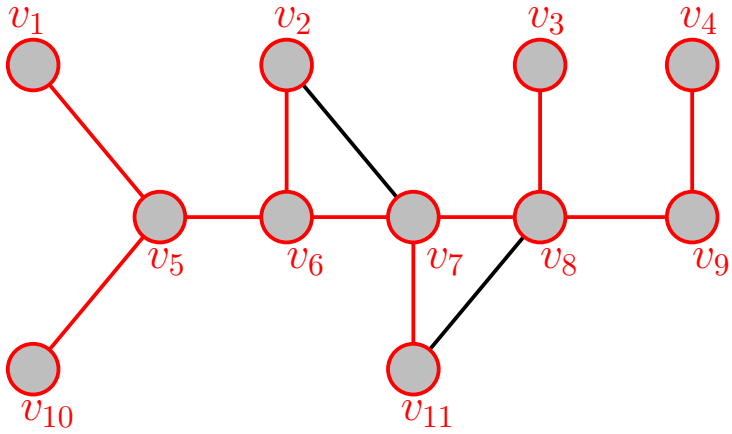
G

$L = (v_4)$



G

$L = ()$



G

- Trước mỗi lần lặp vòng **while**, T là cây chứa các đỉnh đã xét (= các đỉnh đã bỏ đi từ danh sách L)
- Thuật toán BFS chạy trong chạy trong thời gian $O(n^2)$ với đơn đồ thị vô hướng liên thông $G = (V, E)$ bất kỳ, trong đó $n = |V|$
  - Với mỗi đỉnh v, thuật toán xét các đỉnh liền kề với v và thêm các đỉnh chưa xét vào cuối danh sách L
  - Mỗi cạnh được xét nhiều nhất hai lần để xác định xem có cần thêm cạnh đó và một đỉnh đầu mút vào cây T hay không
  - Do đó, thuật toán BFS chạy trong thời gian $O(|E|)$, hay nói cách khác $O(n^2)$
- Thuật toán BFS là cơ sở để thiết kế các thuật toán giải nhiều bài toán khác nhau, ví dụ như bài toán tìm các thành phần liên thông, bài toán xác định xem một đồ thị có phải đồ thị hai phần hay không, bài toán tìm đường đi ngắn nhất giữa hai đỉnh, v.v...

Cho $G = (V, E, w)$ là một đồ thị liên thông vô hướng có trọng số. Một *cây bao trùm nhỏ nhất (minimum spanning tree)* của G là một cây bao trùm của G có tổng trọng số các cạnh của cây là nhỏ nhất có thể

- **Input:** Đồ thị liên thông vô hướng có trọng số $G = (V, E, w)$
- **Output:** Một cây bao trùm nhỏ nhất T của G

- Thuật toán Prim
- Thuật toán Kruskal

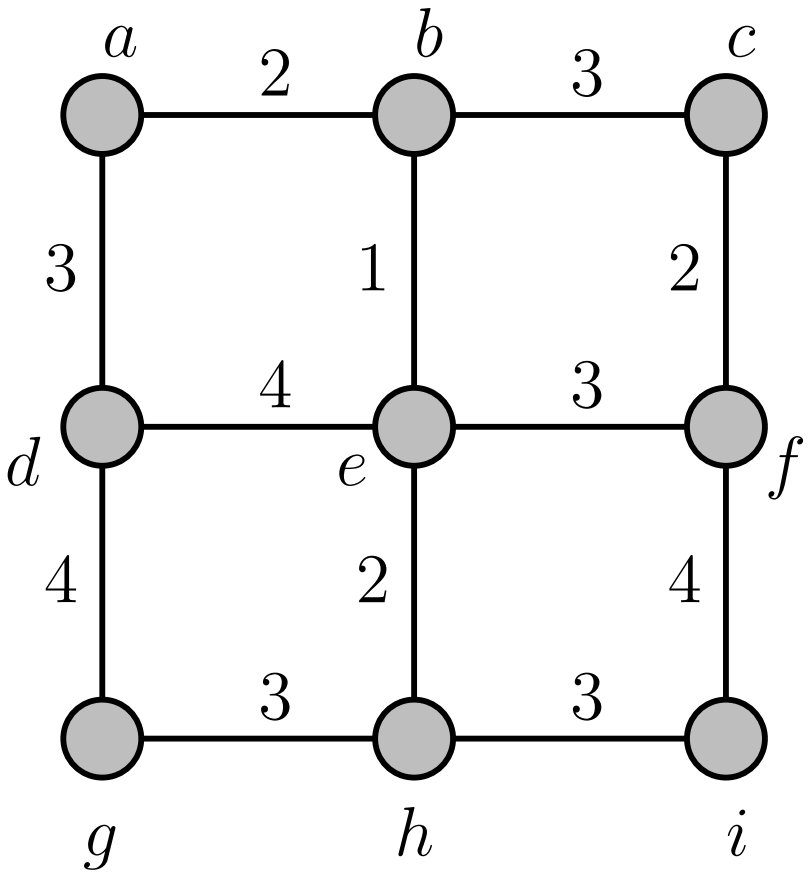
Thuật toán 3: Thuật toán Prim

Input: $G = (V, E, w)$: đồ thị liên thông vô hướng có trọng số
Output: Một cây bao trùm nhỏ nhất T của G

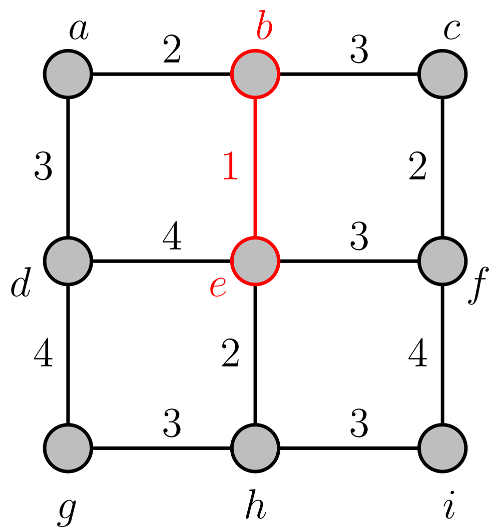
1 $T :=$ một cạnh có trọng số nhỏ nhất
2 **for** $i := 1$ **to** $n - 2$ **do**
3 $\quad e :=$ một cạnh có trọng số nhỏ nhất liên thuộc với một đỉnh của T thỏa mãn $T + e$ không có chu trình
4 $\quad T := T + e$
5 **return** T

Ví dụ 3

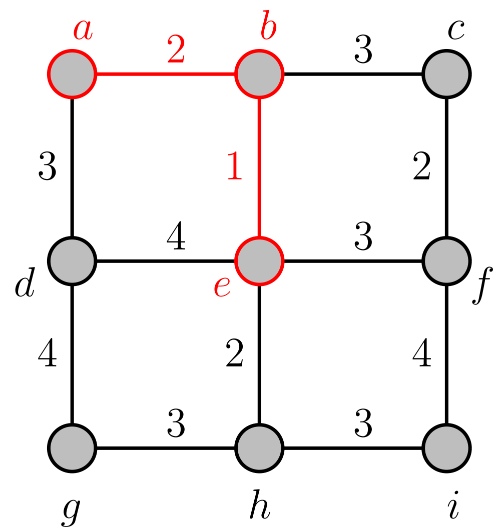
Tìm một cây bao trùm nhỏ nhất T của đồ thị $G = (V, E, w)$ sau bằng thuật toán Prim



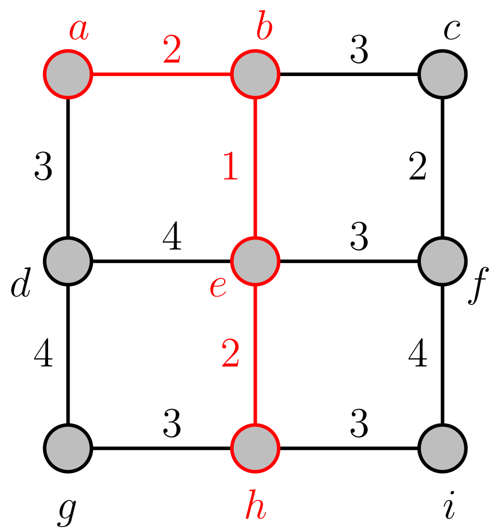
G



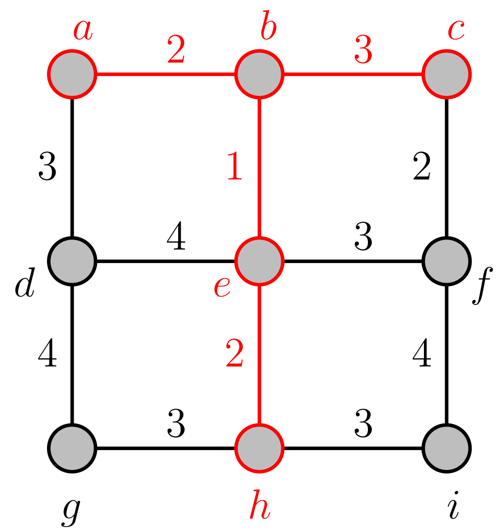
G



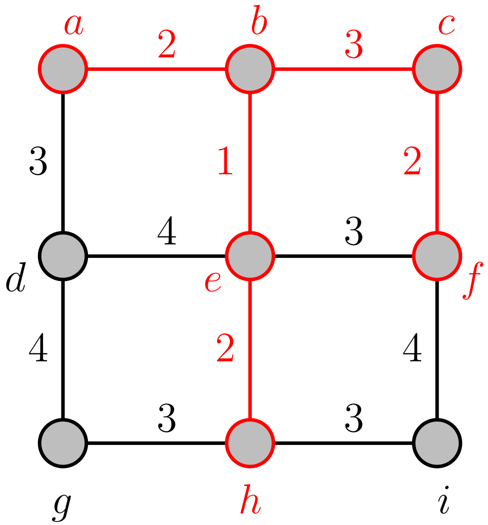
G



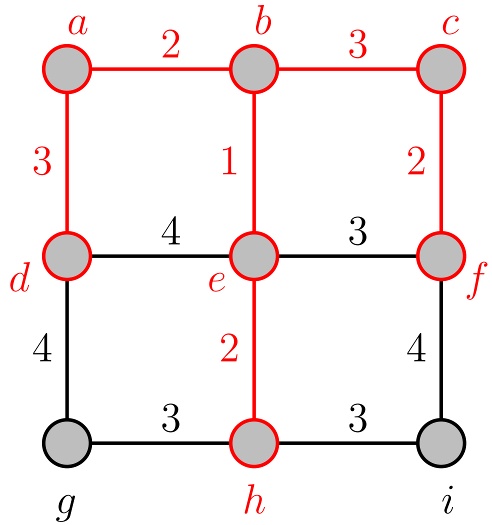
G



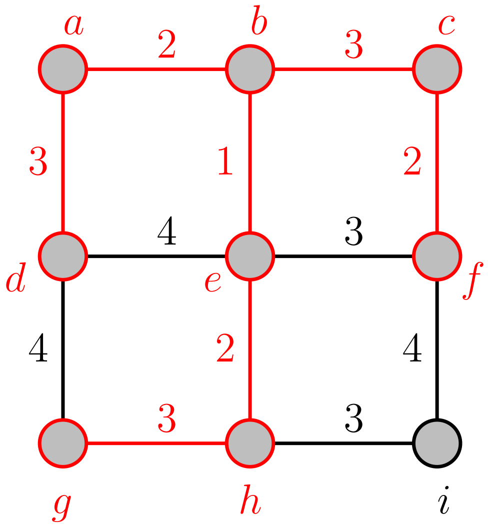
G


a
2
b
3
c
3
1
2
4
3
d
e
f
4
2
4
3
3
g
h
i

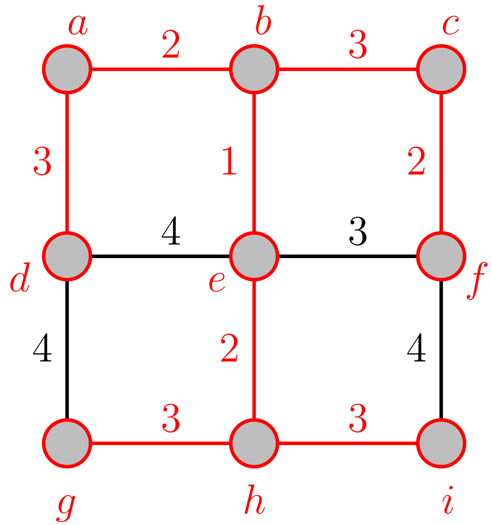
G


a
2
b
3
c
3
1
2
4
3
d
e
f
4
2
4
3
3
g
h
i

G


a
2
b
3
c
3
1
2
4
3
d
e
f
4
2
4
3
3
g
h
i

G


a
2
b
3
c
3
1
2
4
3
d
e
f
4
2
4
3
3
g
h
i

G

Định lý 8

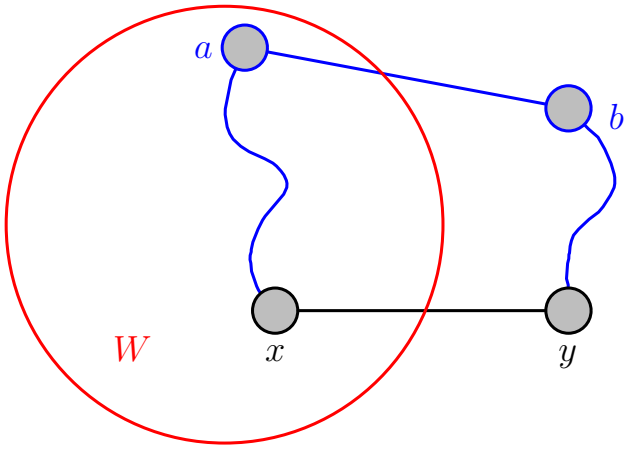
Nếu T là một cây bao trùm xuất ra bởi thuật toán Prim với đồ thị đầu vào $G = (V, E, w)$ thì T là một cây bao trùm nhỏ nhất của G

Chứng minh.

- Thuật toán Prim luôn xuất ra một cây bao trùm
  - Ở các bước trung gian, T luôn là cây
  - Mỗi bước trung gian, thuật toán thêm một cạnh và một đỉnh mới vào T
- Ta chứng minh bằng phản chứng. Giả sử T không có tổng trọng số nhỏ nhất
- Gọi $ET = (e_1, e_2, \ldots, e_m)$ là dãy các cạnh được chọn theo thứ tự thực hiện của thuật toán Prim
- Gọi U là một cây bao trùm nhỏ nhất thỏa mãn điều kiện U có chứa k cạnh đầu tiên của ET với k lớn nhất có thể, nghĩa là $e_1, e_2, \ldots, e_k$ thuộc U và $e_{k+1} = xy$ không thuộc U

Chứng minh (tiếp).

- Gọi W là tập các đỉnh của T tính đến thời điểm *ngay trước khi* $e_{k+1} = xy$ *được chọn*
- Gọi P là một đường đi giữa x và y trong U (chú ý rằng xy không là một cạnh của U) và gọi ab *là cạnh đầu tiên của* P *có một đầu mút* (a) *thuộc* W *và một đầu mút* (b) *không thuộc* W
- Xét cây bao trùm $S = U - ab + xy$
  - **Nếu** $w(a, b) > w(x, y)$**:** Tổng trọng số của S nhỏ hơn của U, mâu thuẫn với giả thiết U là cây bao trùm nhỏ nhất
  - **Nếu** $w(a, b) = w(x, y)$**:** S là một cây bao trùm nhỏ nhất và S có chứa nhiều cạnh của ET hơn U, mâu thuẫn với định nghĩa của U
  - **Nếu** $w(a, b) < w(x, y)$**:** Thuật toán Prim sẽ chọn cạnh ab thay vì cạnh xy, mâu thuẫn với giả thiết ban đầu là xy là cạnh được chọn



□

- Thuật toán Prim là một thuật toán tham lam
  - Ở mỗi bước xây dựng cây bao trùm nhỏ nhất T, thuật toán Prim chỉ thêm các cạnh có trọng số nhỏ nhất liền kề với một đỉnh của cây T tính đến hiện tại mà không tạo thành chu trình mới
- Thuật toán Prim có thể được lập trình để chạy trong thời gian $O(m \log n)$ với đồ thị $G = (V, E, w)$ thỏa mãn $m = |E|$ và $n = |V|$
- Thuật toán Prim tương tự như thuật toán Dijkstra tìm đường đi ngắn nhất giữa hai đỉnh đã giới thiệu trước đó

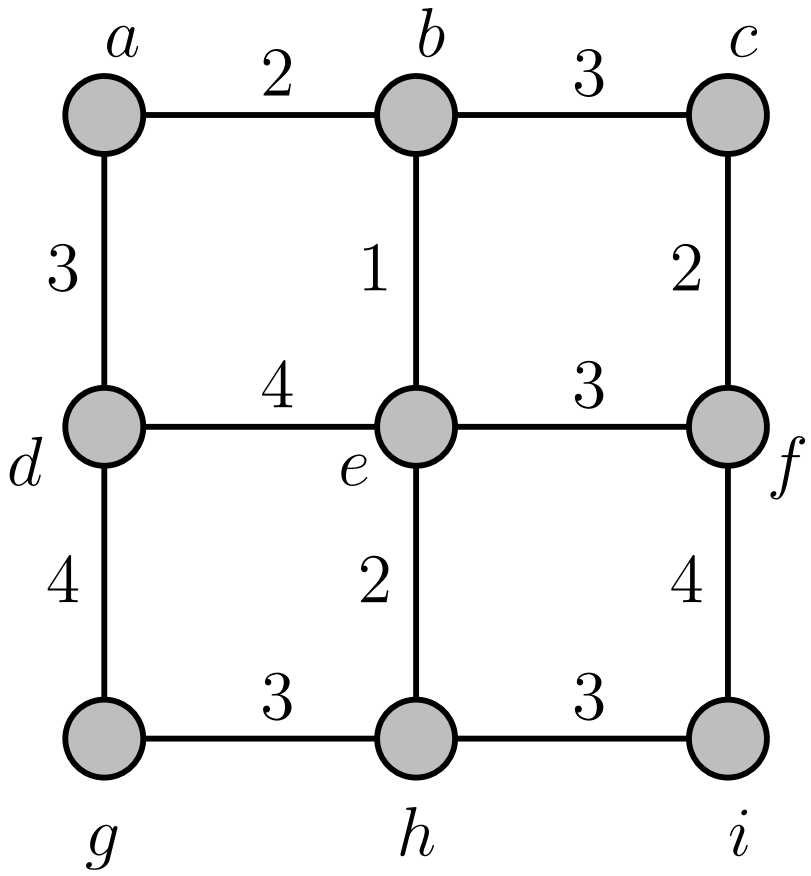
Thuật toán 4: Thuật toán Kruskal

Input: $G = (V, E, w)$: đồ thị liên thông vô hướng có trọng số
Output: Một cây bao trùm nhỏ nhất T của G

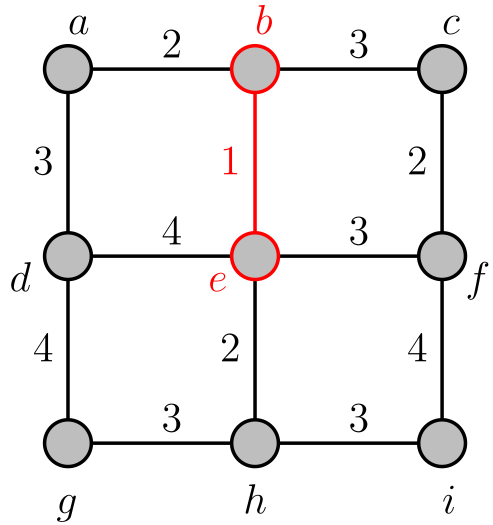
1 $T :=$ đồ thị rỗng (không có đỉnh và cạnh)
2 **for** $i := 1$ **to** $n - 1$ **do**
3 $\quad e :=$ một cạnh có trọng số nhỏ nhất thỏa mãn $T + e$ không có chu trình
4 $\quad T := T + e$
5 **return** T

Ví dụ 4

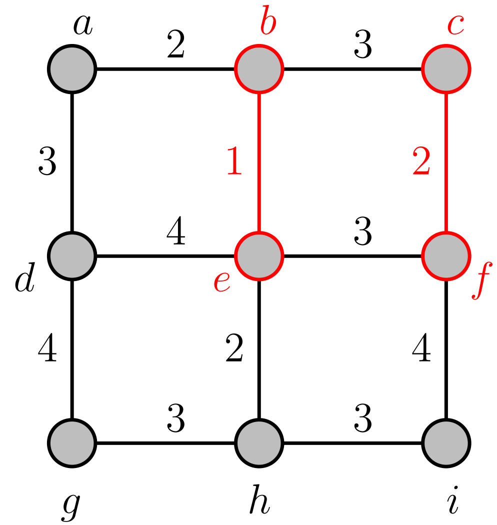
Tìm một cây bao trùm nhỏ nhất T của đồ thị $G = (V, E, w)$ sau bằng thuật toán Kruskal



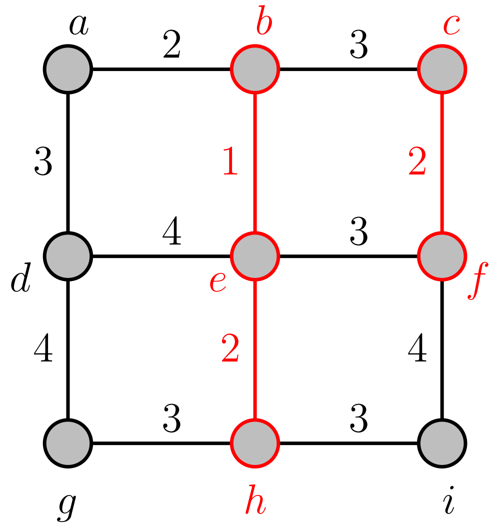
G



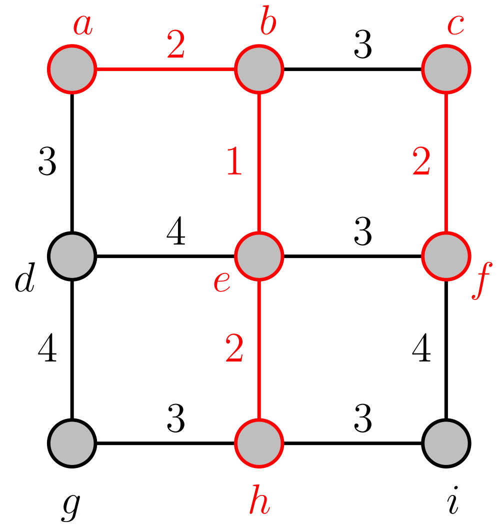
G



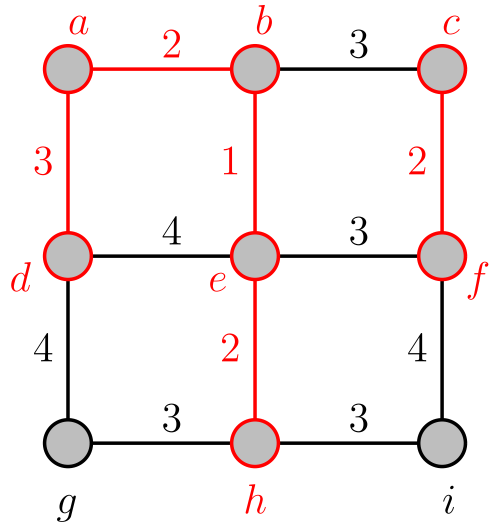
G



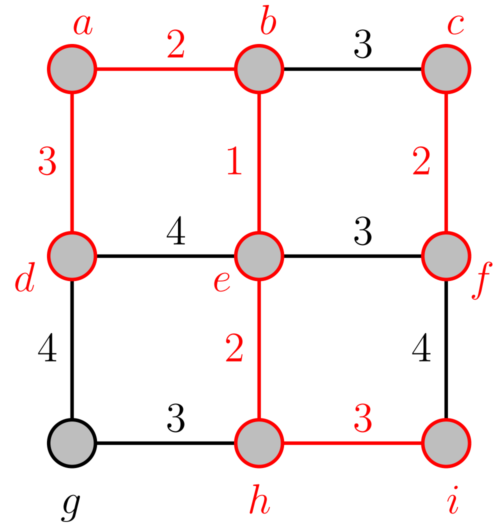
G



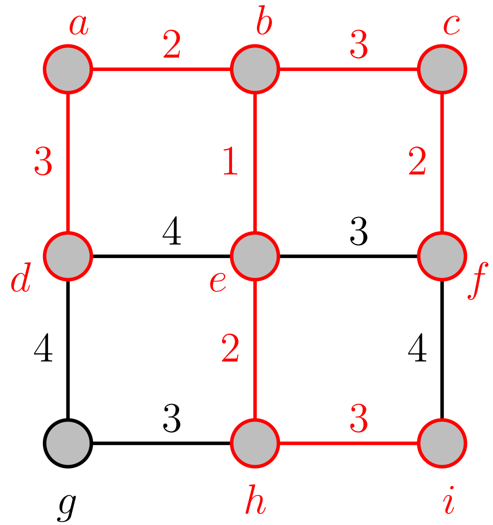
G



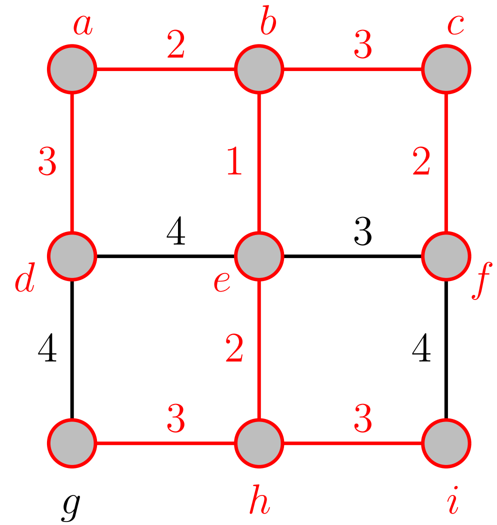
G



G



G



G

Bài tập 9

*Chứng minh rằng nếu T là một cây bao trùm của $G = (V, E, w)$ xuất ra bởi thuật toán Kruskal thì T là cây bao trùm nhỏ nhất (**Gợi ý:** xem lại chứng minh tính đúng đắn của thuật toán Prim)*

- Thuật toán Kruskal là một thuật toán tham lam
  - Ở mỗi bước xây dựng cây bao trùm nhỏ nhất T, thuật toán Kruskal chỉ thêm các cạnh có trọng số nhỏ nhất mà không tạo thành chu trình mới
  - Khác với thuật toán Prim, *ở các bước trung gian của thuật toán Kruskal, T có thể không là cây*
- Thuật toán Kruskal có thể được lập trình để chạy trong thời gian $O(m \log m)$ với đồ thị $G = (V, E, w)$ thỏa mãn $m = |E|$