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This document is not up to date and may contain several errors or outdated information.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: 2023-05-15

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: 2023-05-15

VNU-HUS MAT3500: Toán rời rạc

Lý thuyết đồ thị IV

Hoàng Anh Đức

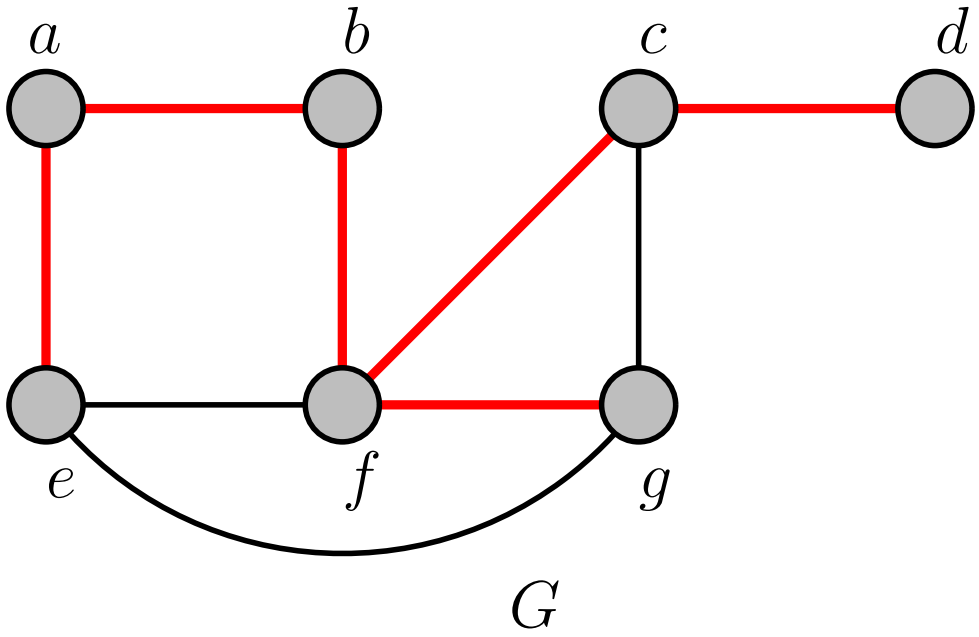
Bộ môn Tin học, Khoa Toán-Cơ-Tin học
Đại học KHTN, ĐHQG Hà Nội
hoanganhduc@hus.edu.vn

Nội dung

Cây

- Cây bao trùm
- Cây bao trùm nhỏ nhất

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 1

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

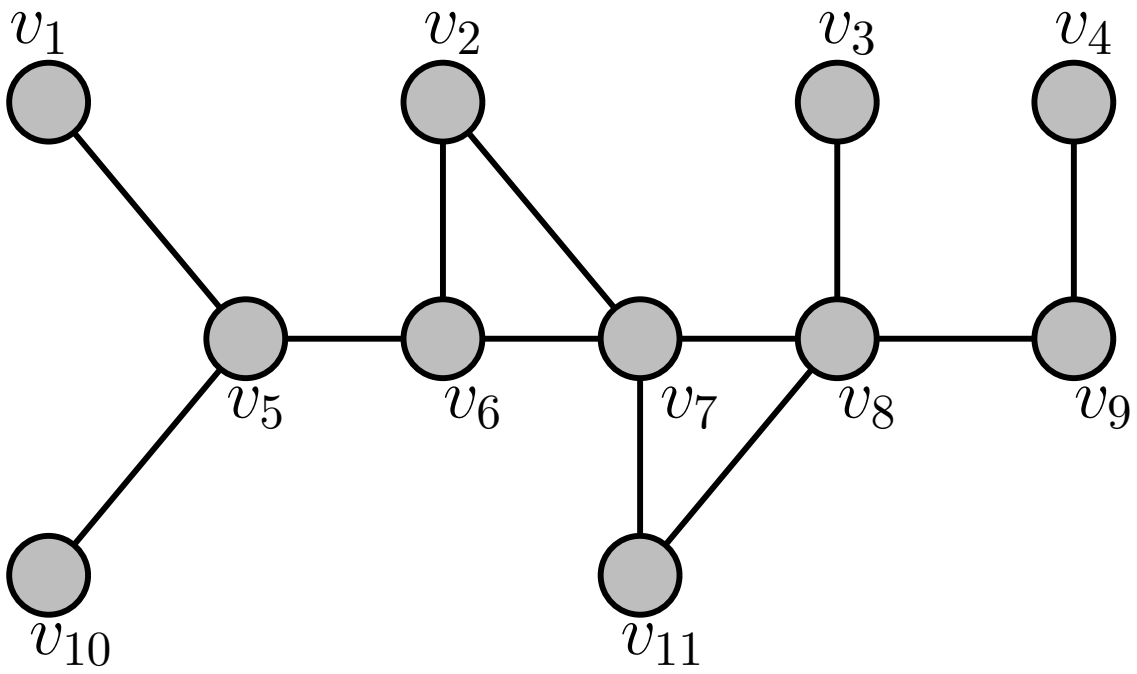
Input: G: đơn đồ thị vô hướng liên thông với các đỉnh $v_1, v_2, \ldots, v_n$

Output: Cây bao trùm T

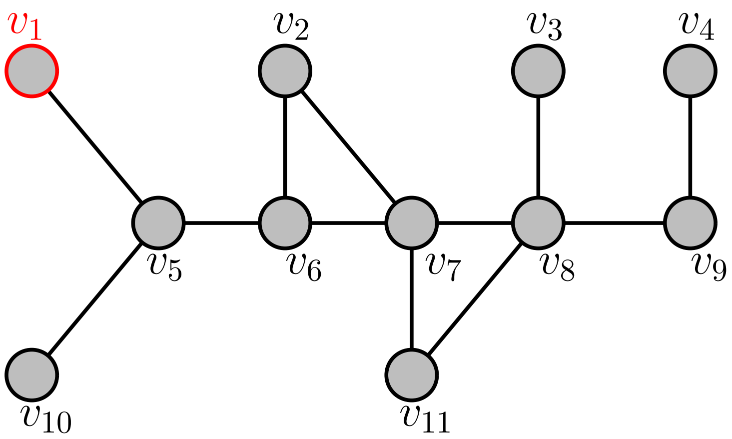
```
1 procedure dfs(G):
2     T := cây có chính xác một đỉnh v_1
3     visit(v_1)
4 procedure visit(v: một đỉnh của G):
5     for mỗi đỉnh w liền kề với v và chưa thuộc T do
6         Thêm w và cạnh vw vào T
7         visit(w)
8 return T
```

Ví dụ 1

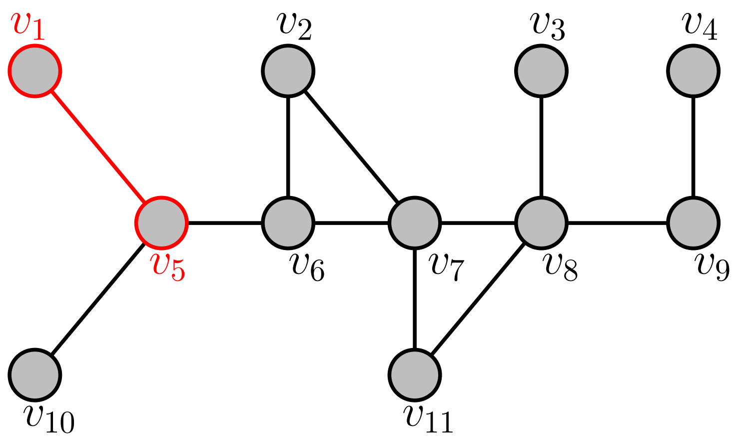
Tìm một cây bao trùm T của đồ thị $G = (V, E)$ sau bằng thuật toán duyệt theo chiều sâu



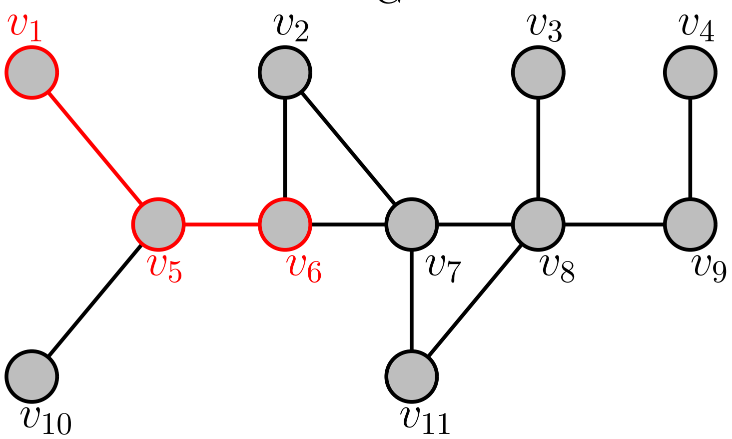
G


v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

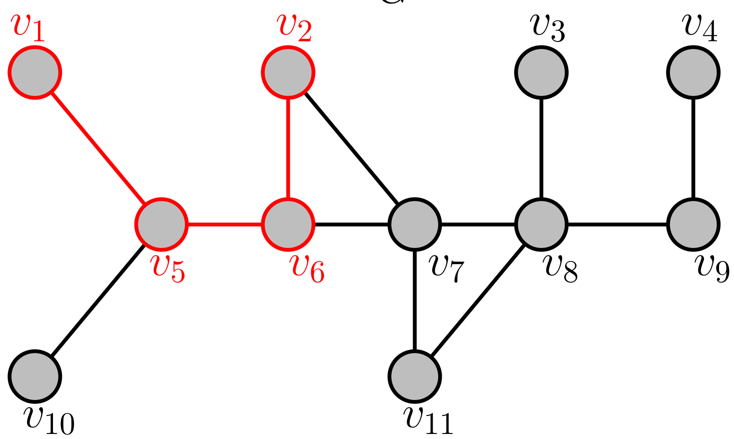
G


v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

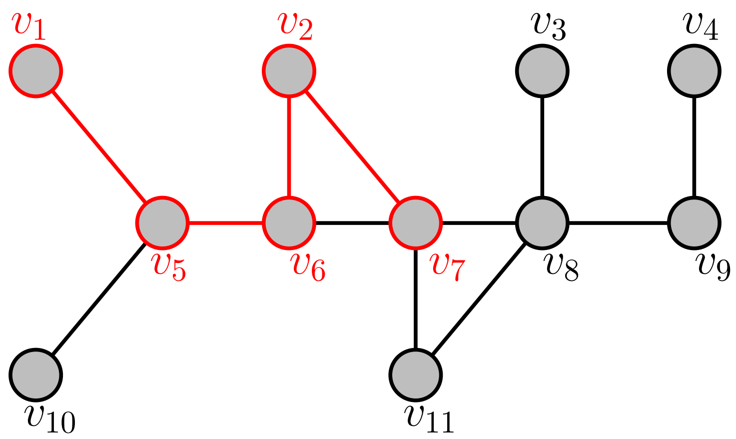
G


v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

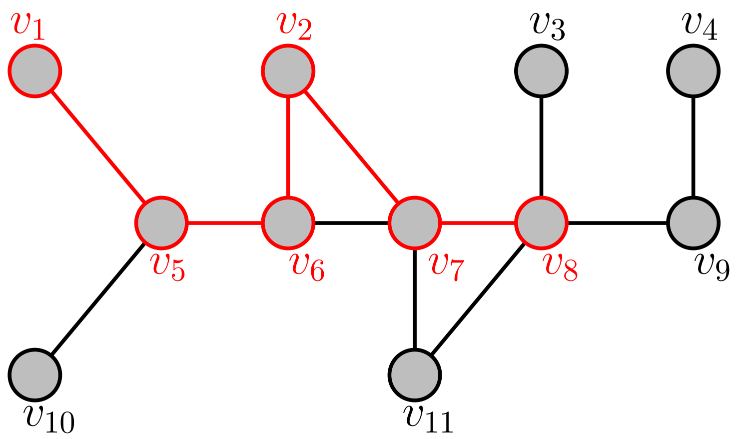
G


v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

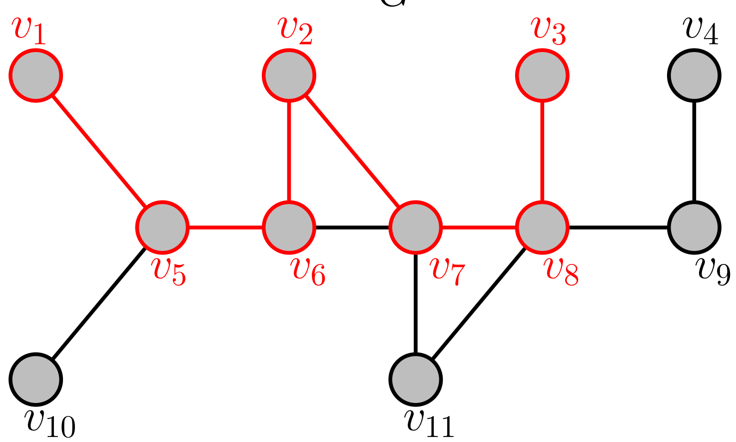
G


v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

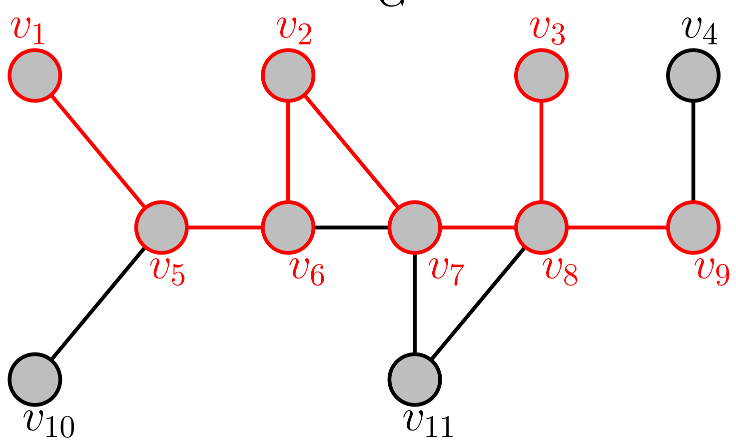
G


v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

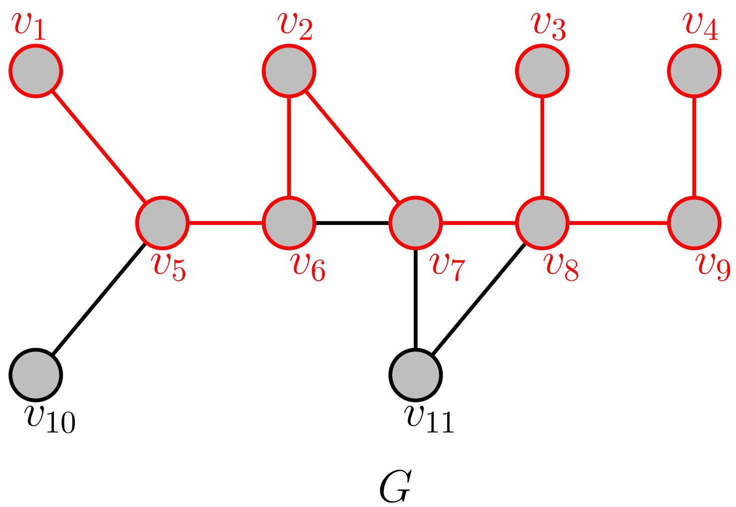
G

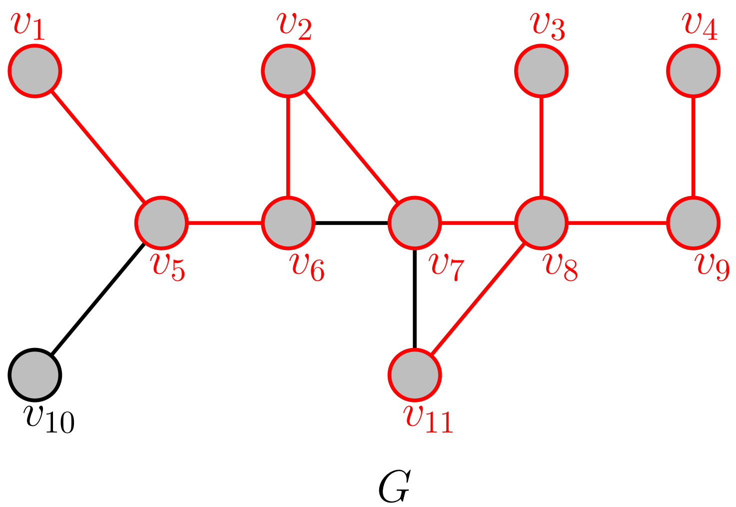

v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

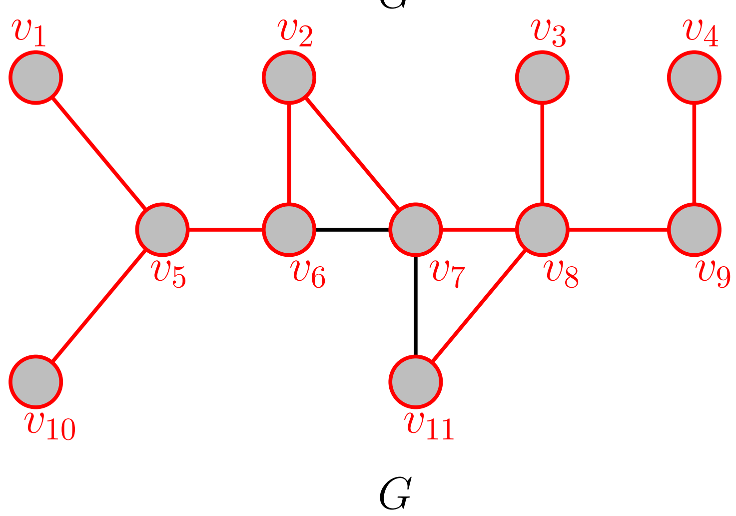
G


v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}

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

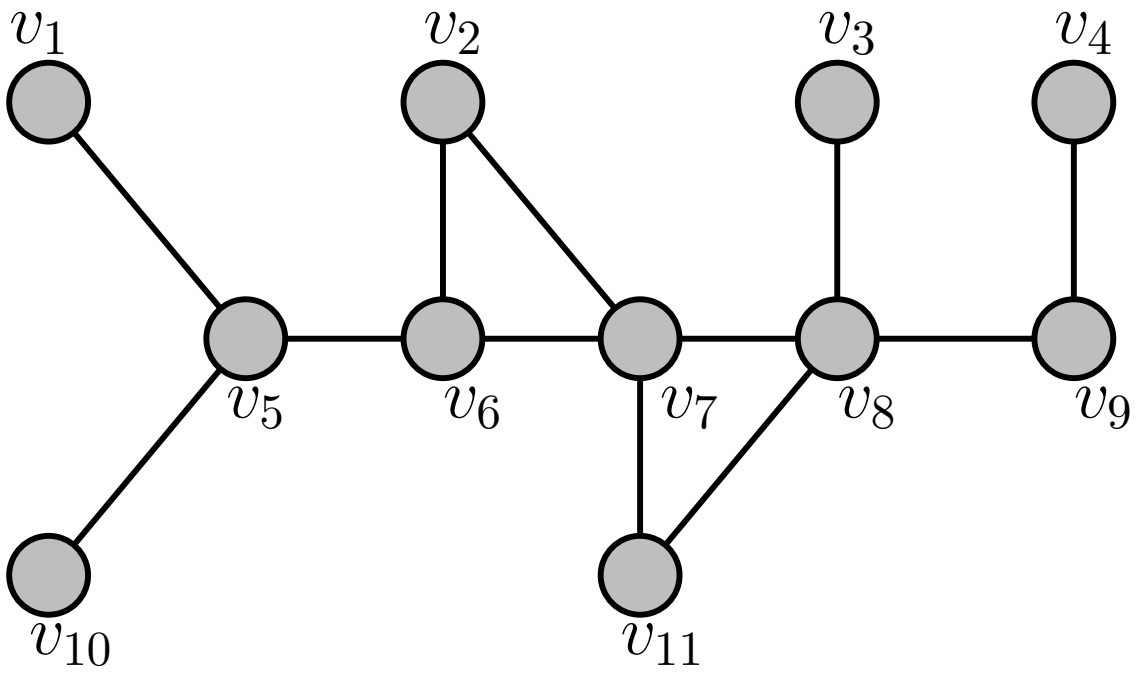
Thuật toán 2: Duyệt theo chiều rộng

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

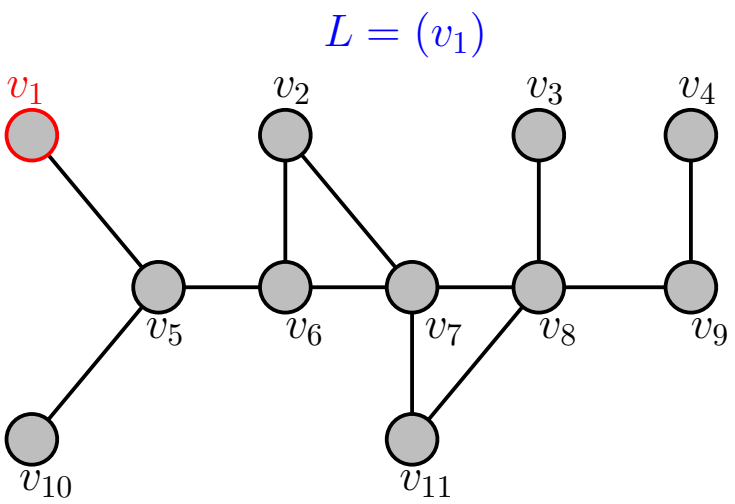
```
1  T := cây chỉ chứa duy nhất một đỉnh v_1
2  L := danh sách rỗng
3  Thêm v_1 vào danh sách L các đỉnh chưa xét
4  while L khác rỗng do
5      Bỏ đi đỉnh thứ nhất v từ L
6      for mỗi đỉnh w liền kề với v do
7          if w không thuộc L và w không thuộc T then
8              Thêm w vào cuối danh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

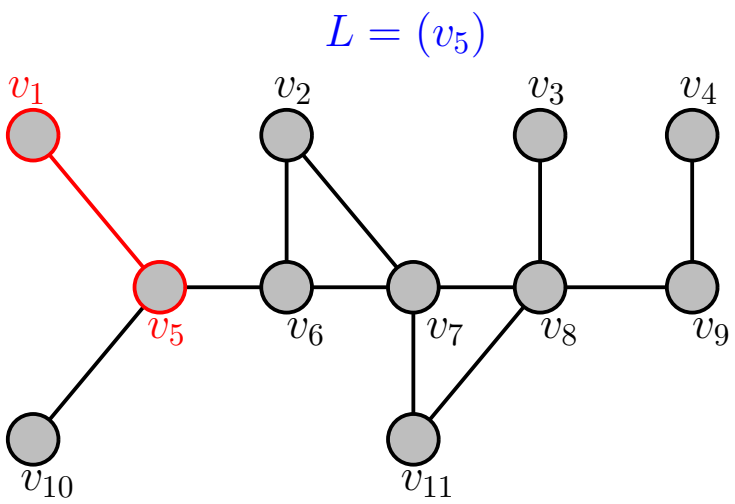
Ví dụ 2

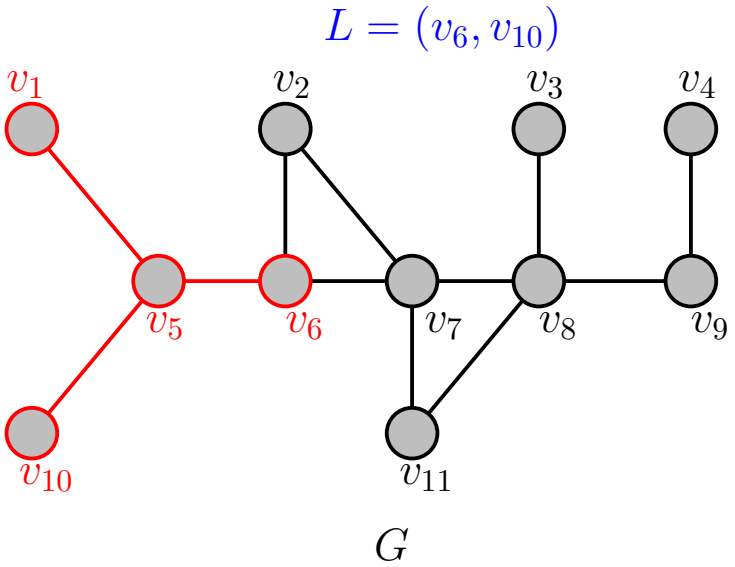
Tìm một cây bao trùm T của đồ thị $G = (V, E)$ sau bằng thuật toán duyệt theo chiều rộng

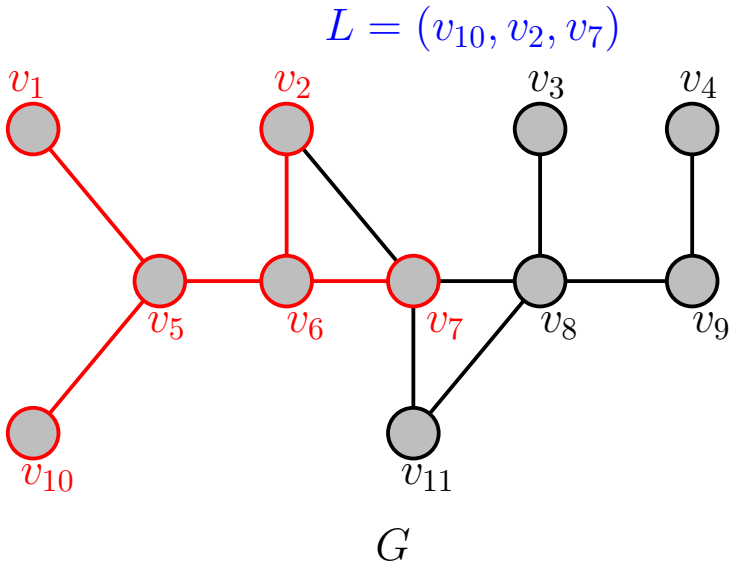


G

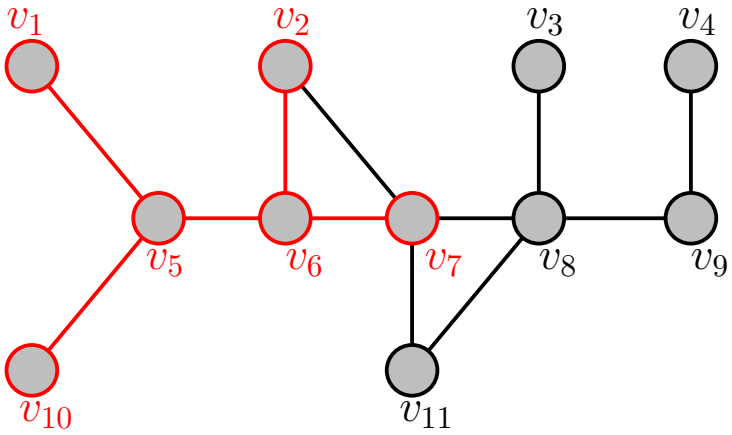

$L = (v_1)$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
G


$L = (v_5)$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
G


$L = (v_6, v_{10})$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
G

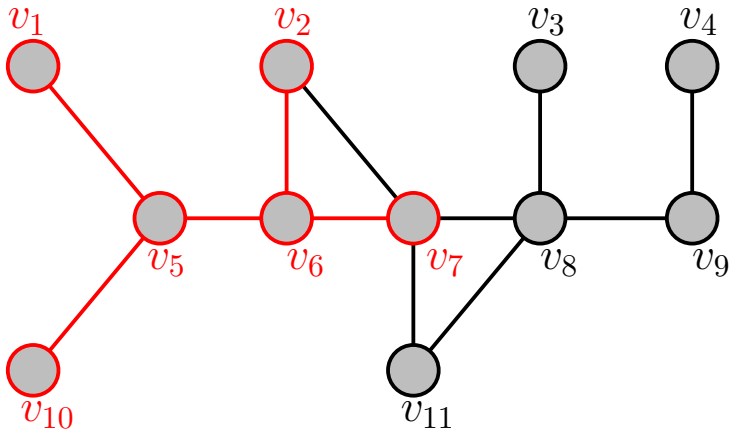

$L = (v_{10}, v_2, v_7)$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
G

$L = (v_2, v_7)$



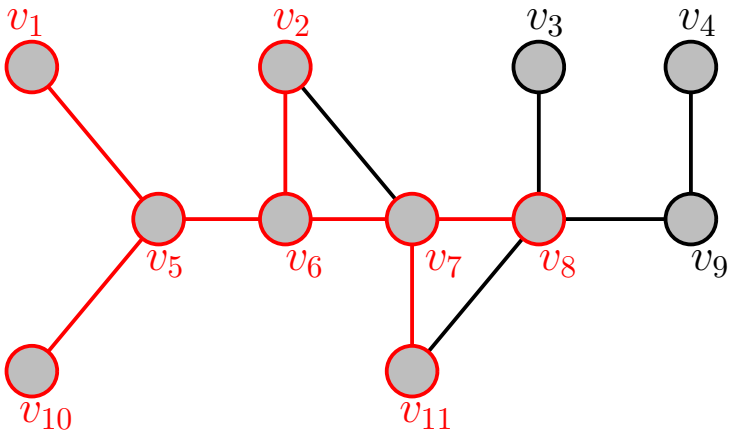
G

$L = (v_7)$



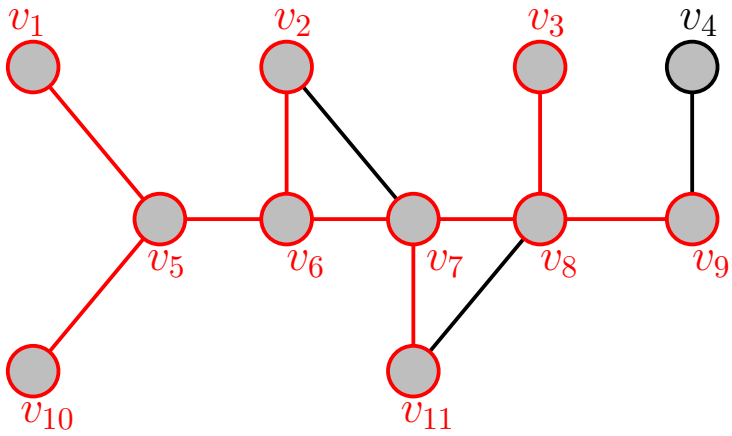
G

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

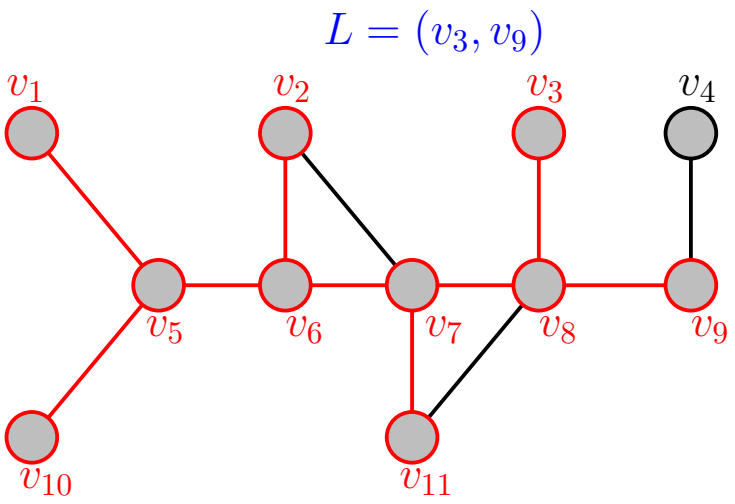


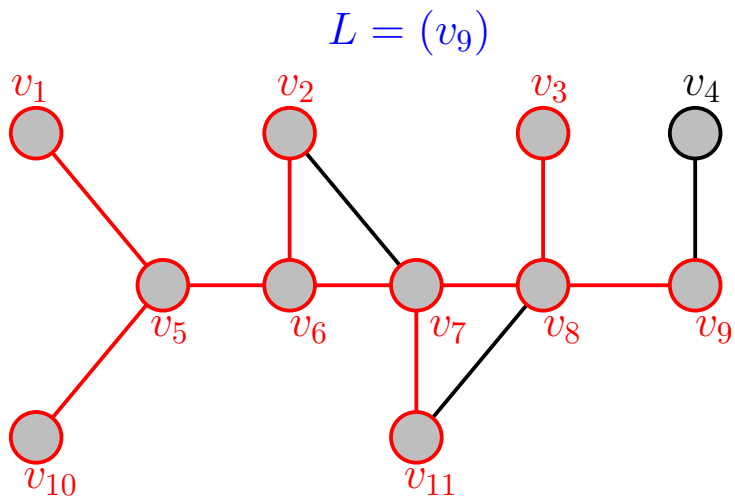
G

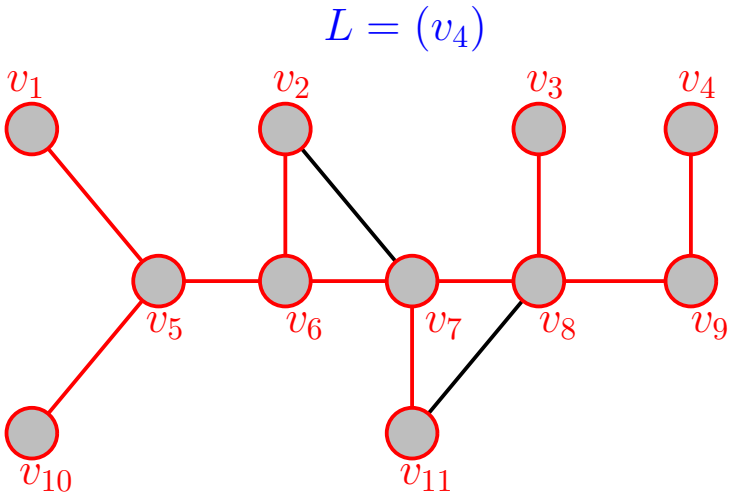
$L = (v_{11}, v_3, v_9)$

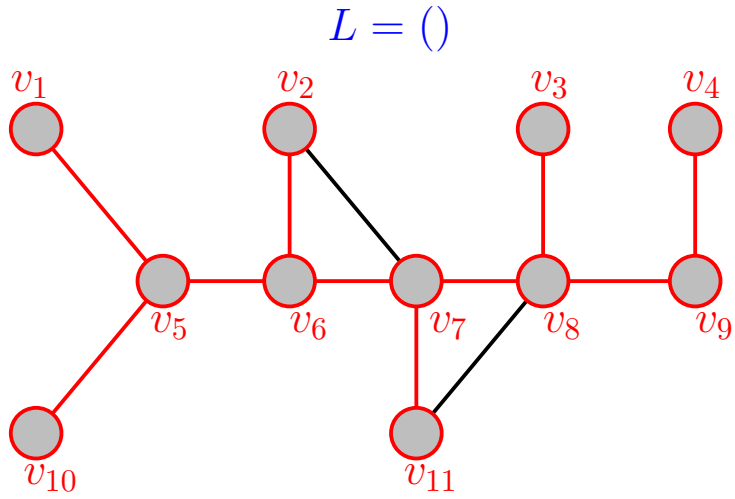


G


$L = (v_3, v_9)$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
G


$L = (v_9)$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
G


$L = (v_4)$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
G


$L = ()$
v_1 v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9
v_{10} v_{11}
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

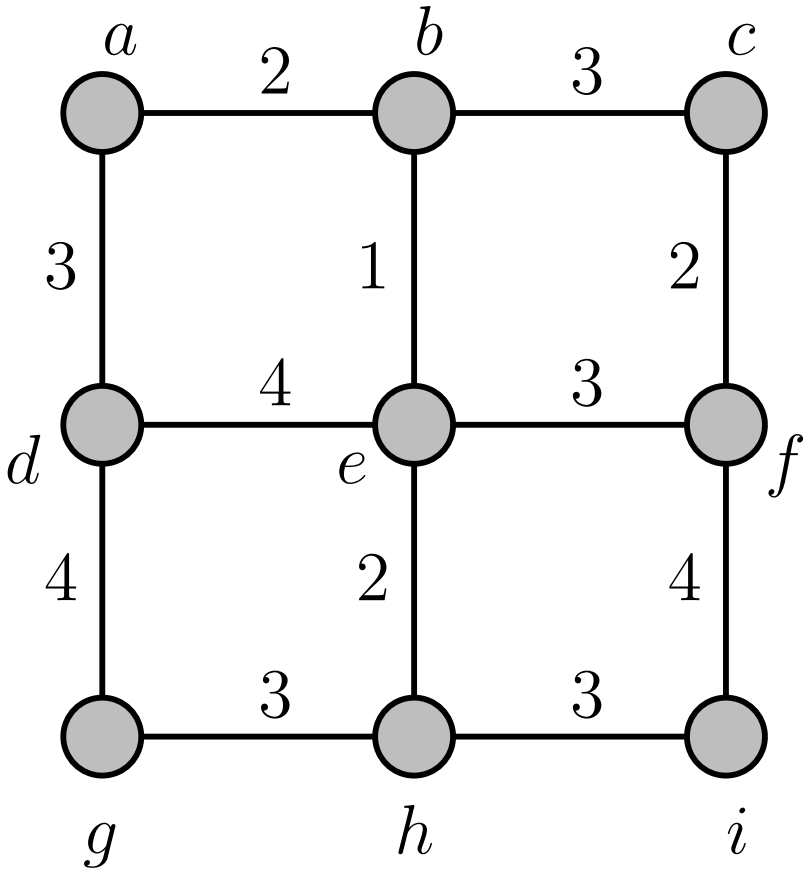
Thuật toán 3: Thuật toán Prim

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

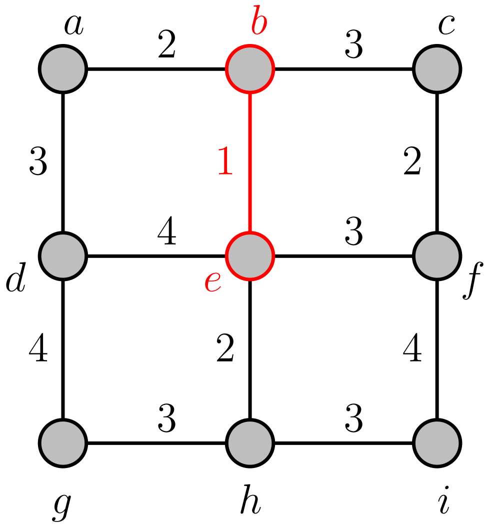
Tìm một cây bao trùm nhỏ nhất T của đồ thị $G = (V, E, w)$ sau bằng thuật toán Prim



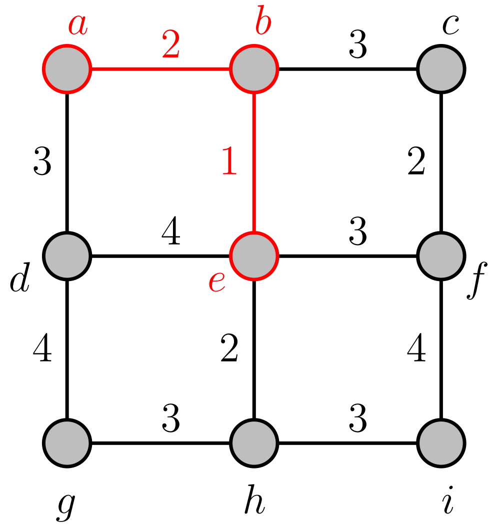
G

Cây

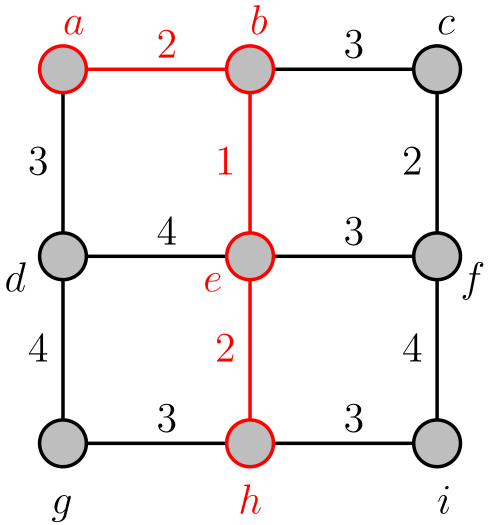
Cây bao trùm nhỏ nhất



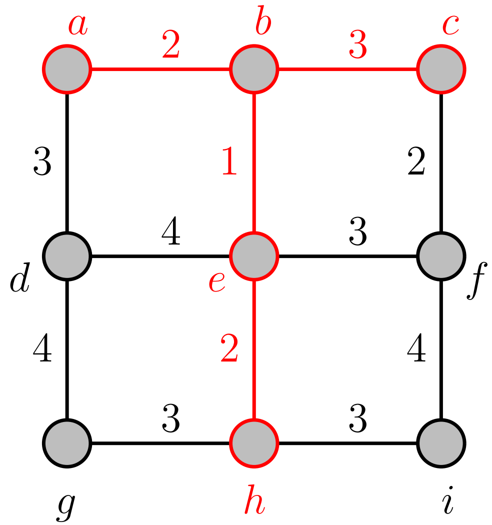
G



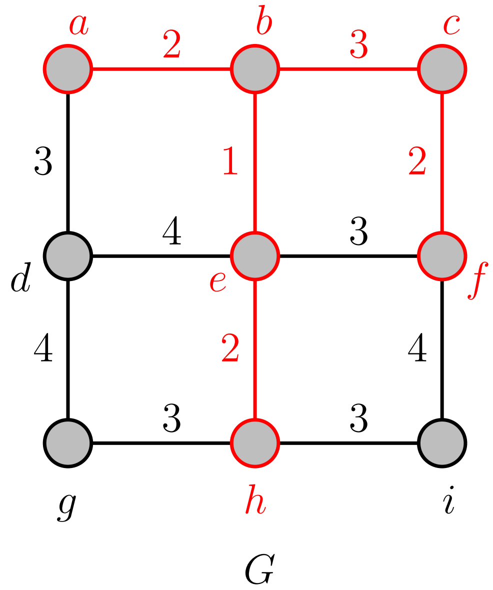
G

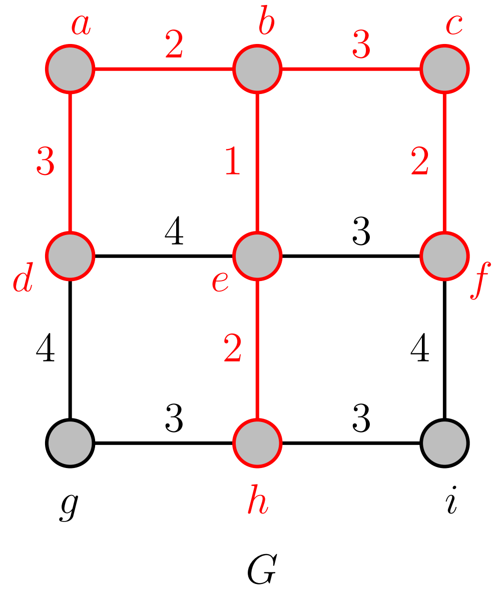


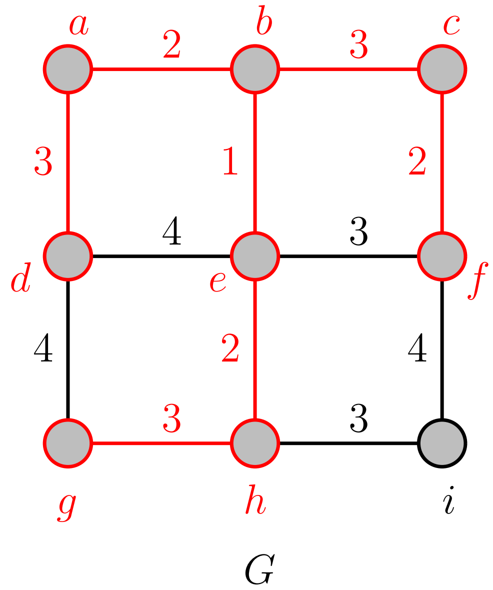
G

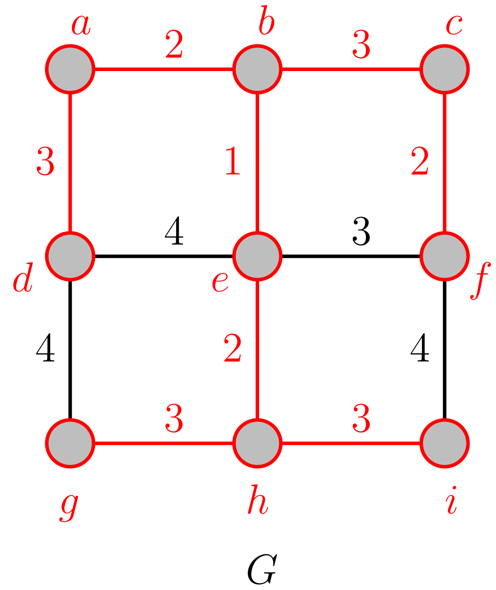


G


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


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


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


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

Định lý 2

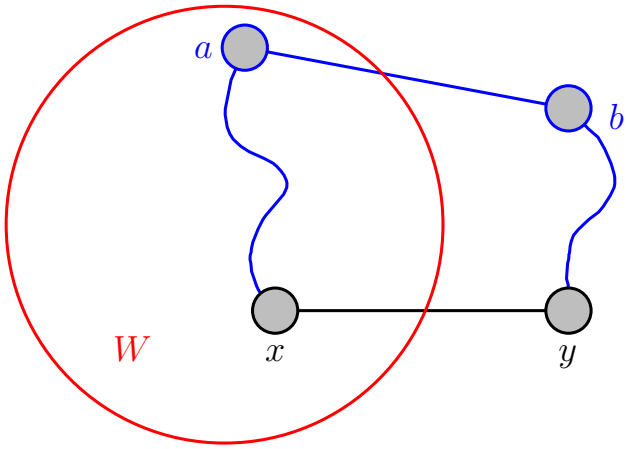
Nếu T là một cây bao trùm xuất ra bởi thuật toán Prim với đồ thị đầu vào $G = (V, E, w)$ thì T là một cây bao trùm nhỏ nhất của G

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

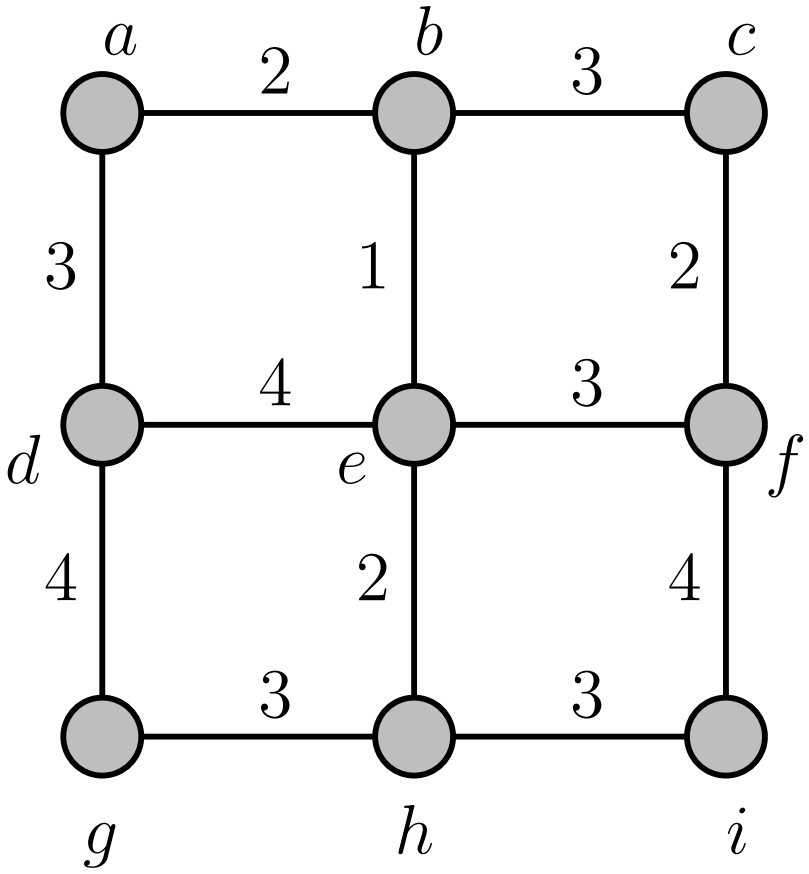
Thuật toán 4: Thuật toán Kruskal

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

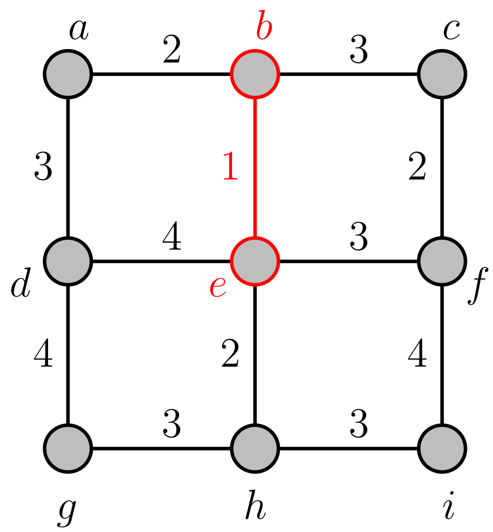
Ví dụ 4

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

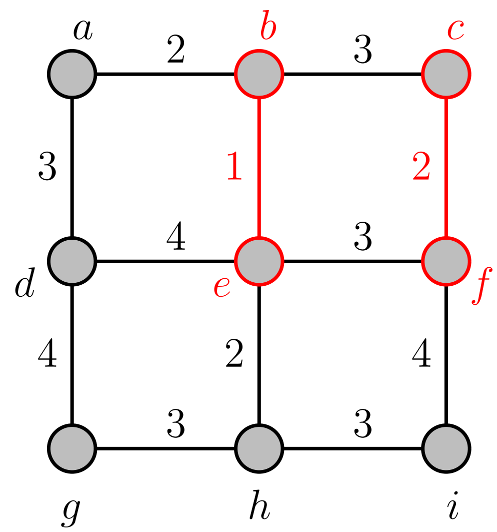


G

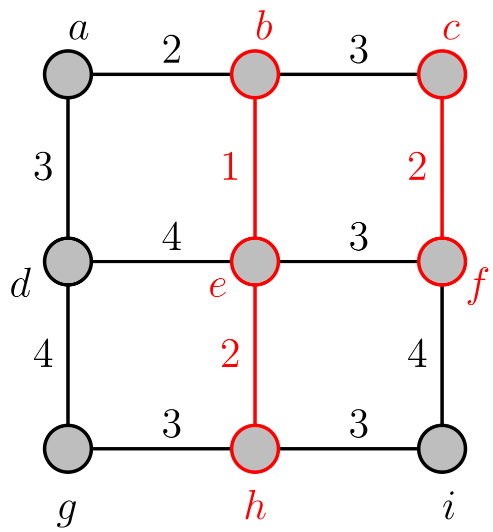
Cây
Cây bao trùm nhỏ nhất



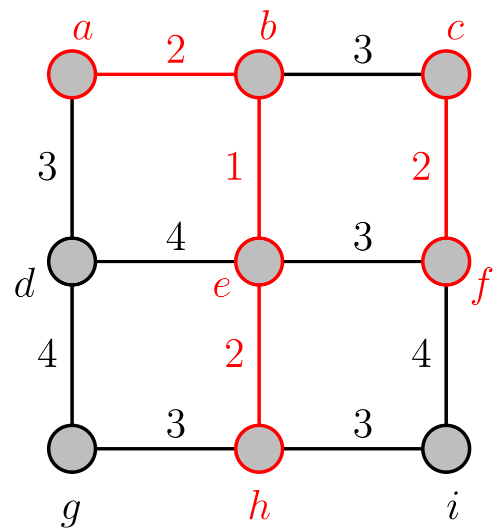
G



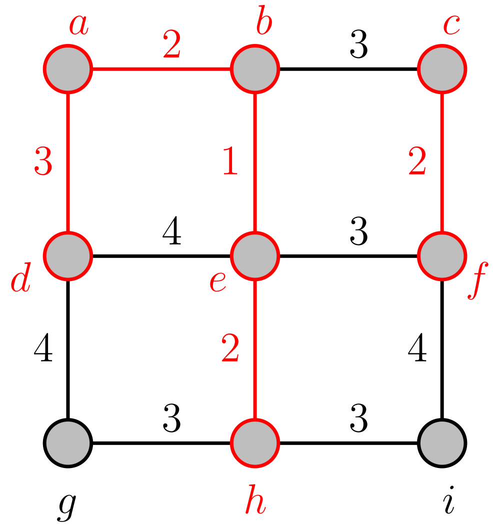
G



G

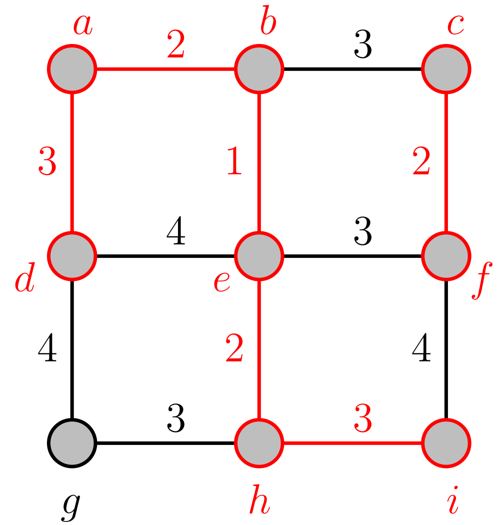


G



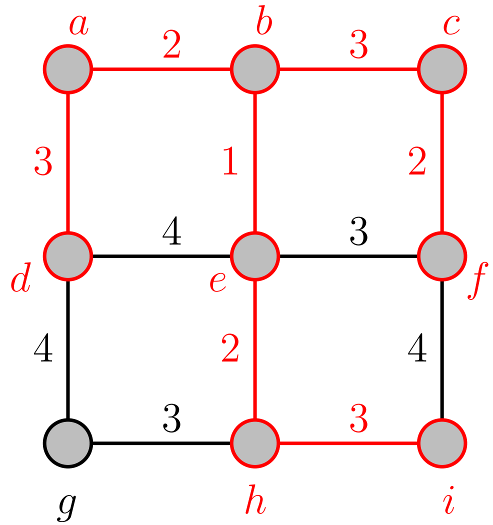
a b c d e f g h i
2 3 3 1 2 4 3 4 2 4 3 3

G



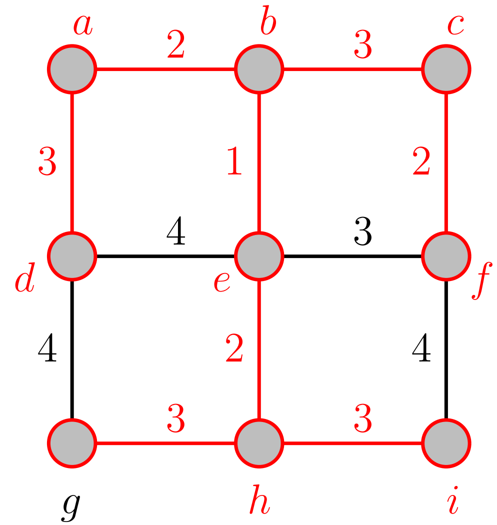
a b c d e f g h i
2 3 3 1 2 4 3 4 2 4 3 3

G



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

G



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

G

Bài tập 1

*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 $m = |E|$