Chương 4

CÂY

4.1 Mở đầu

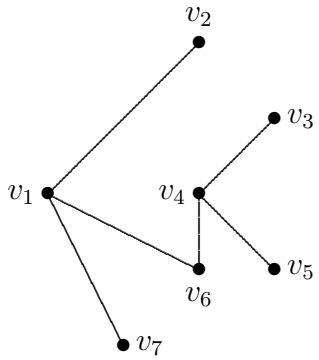
Cây là một trong những khái niệm quan trọng nhất của lý thuyết đồ thị, và thường xuất hiện trong những lãnh vực ít có liên quan đến đồ thị.

Trong chương này, trước hết sẽ nghiên cứu *cây Huffman* và những ứng dụng của nó trong việc nén dữ liệu. Kế tiếp chúng ta xét trình bày các thuật toán tìm cây bao trùm, cây bao trùm có trọng lượng nhỏ nhất khi các cạnh của đồ thị được gắn với các chi phí (trọng lượng). Cây bao trùm nhỏ nhất của đồ thị có nhiều ứng dụng trong những trường hợp các đường dẫn (ống dẫn ga, dây dẫn trong mạng điện, v.v) được sử dụng để nối n điểm với nhau theo cách tốt nhất: tổng khoảng cách của các đường dẫn là nhỏ nhất. Nếu n điểm được nối với nhau trên một mặt phẳng, ta có thể biểu diễn bởi một đồ thị đầy đủ trong đó các chi phí cạnh là khoảng cách giữa hai điểm tương ứng. Khi đó cây bao trùm với trọng lượng nhỏ nhất sẽ cho mạng giao thông với chi phí ít nhất. Nếu có thể nối thêm ngoài n điểm cho phép, ta có thể thậm chí xây dựng được mạng với chí phí rẻ hơn và xác định nó chính là giải quyết bài toán Steiner. Bài toán sau này sẽ được đề cập ở phần cuối chương.

Định nghĩa 4.1.1 Các định nghĩa sau của *cây* (vô hướng) là tương đương:

1. Đồ thị liên thông có n đỉnh và $(n-1)$ cạnh.
2. Đồ thị liên thông không có chu trình.
3. Đồ thị mà mọi cặp đỉnh được nối với nhau bởi một và chỉ một dây chuyền sơ cấp.
4. Đồ thị liên thông và khi bớt một cạnh bất kỳ thì mất tính liên thông.

Hình 4.1 minh họa cây có bảy đỉnh và sáu cạnh.



Hình 4.1: Một ví dụ về cây.

Khái niệm về cây như một thực thể của toán học được đưa ra lần đầu tiên bởi Kirchhoff [37] khi liên hệ với định nghĩa các mạch cơ bản được sử dụng trong phân tích các mạng điện. Khoảng 10 năm sau đó, một cách độc lập, Cayley [11] đã phát hiện lại các cây và những tích chất của nó khi nghiên cứu các tính chất hoá học của các chất đồng phân của hydrocarbon.

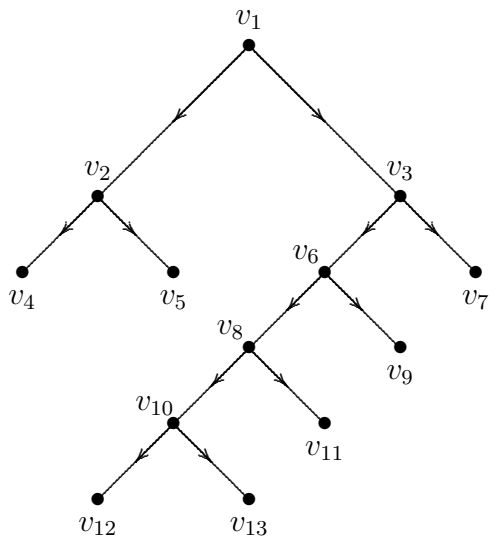
Cây có gốc (còn gọi là *cây gia phả*) được định nghĩa tương tự như sau:

Định nghĩa 4.1.2 *Cây có gốc T* là đồ thị có hướng không mạch mà mọi đỉnh, ngoại trừ một đỉnh (chẳng hạn v_1), có bậc trong bằng một: bậc trong của đỉnh v_1 (gọi là *gốc* của cây) bằng không; nói cách khác, mọi đỉnh $v \in T$ tồn tại duy nhất một đường đi từ gốc đến v.

Hình 4.2 minh họa một đồ thị là cây có gốc với đỉnh v_1 là gốc. Từ định nghĩa suy ra rằng cây có gốc n đỉnh có $(n-1)$ cung và là đồ thị liên thông (khi bỏ qua hướng trên các cung).

Cần chú ý rằng, có thể định hướng trên một cây (vô hướng) sao cho đồ thị thu được là cây có gốc: Ta chỉ cần chọn một đỉnh tuỳ ý, chẳng hạn v_1, làm gốc và định hướng các cung theo dây chuyền từ v_1 đến các đỉnh treo. Ngược lại, nếu bỏ qua các hướng trên cây có gốc ta thu được một cây.

Cây gia phả mà trong đó mỗi người đàn ông biểu thị một đỉnh và các cung được vẽ từ các cha đến các con của họ là một ví dụ quen thuộc của cây có gốc, gốc của cây là người đầu tiên trong dòng họ mà có thể xác định được.



Hình 4.2: Một ví dụ về cây có gốc.

4.2 Cây Huffman

Tiến trình gán dãy các bit cho các ký hiệu gọi là *mã hoá*. Trong phần này chúng ta một tả một thuật toán mã hoá rất quen thuộc-*thuật toán mã hoá Huffman*.

4.2.1 Các bộ mã "tốt"

Khi ta nói về *mã hoá* có nghĩa là gán dãy các bit cho các phần tử của một bảng chữ cái. Tập các chuỗi nhị phân gọi là *bộ mã* và các phần tử của chúng gọi là *từ mã*. Một bảng chữ cái là một tập hợp các ký hiệu, gọi là các *ký tự*. Chẳng hạn, bảng chữ cái sử dụng trong hầu hết các sách (tiếng Anh) gồm 26 ký tự thường, 26 ký tự hoa, và các dấu ngắt câu (như dấu phẩy). Mã ASCII (viết tắt của các chữ cái đầu tiên của chuỗi American Standard Code for Information Interchange của ký tự A là 01000001, của ký tự a là 01100001 và ký tự , là 0011010. Chú ý rằng trong mã ASCII số các bit sử dụng để biểu diễn các ký tự là bằng nhau. Mã như vậy gọi là *mã có độ dài cố định*. Nếu ta muốn giảm số các bit đòi hỏi để biểu diễn các thông báo khác nhau, ta cần các chuỗi bit biểu diễn ký tự có độ dài (nói chung) không bằng nhau. Nếu biểu diễn các bit dài hơn cho các ký tự thường xuyên xuất hiện và ngược lại, chúng ta có thể, về trung bình, giảm số bit biểu diễn các ký hiệu. Chẳng hạn, mã Morse [31] sử dụng các từ mã ngắn hơn cho các ký tự xuất hiện thường xuyên: mã của của a là $\cdot-$, của e là $\cdot$, trong khi của z là $--\cdot\cdot$, q là $--\cdot-$, j là $\cdot---$.

Tuy nhiên độ dài trung bình của mã không phải là tiêu chuẩn quan trọng khi thiết kế

một bộ mã "tốt". Xét ví dụ sau. Giả sử bảng chữ cái gồm bốn ký tự a_1, a_2, a_3, a_4 với các xác suất xuất hiện tương ứng là $P(a_1) = \frac{1}{2}, P(a_2) = \frac{1}{4}, P(a_3) = \frac{1}{8}, P(a_4) = \frac{1}{8}$. Entropy của mã nguồn này là 1.75 bit/ký hiệu (xem [31] để có khái niệm về entropy). Xét các bộ mã trong nguồn này cho bởi bảng sau

Các ký tự	Mã C_1	Mã C_2	Mã C_3	Mã C_4
a_1	1	1	1	1
a_2	1	0	01	10
a_3	0	11	001	100
a_4	01	00	000	1000
Độ dài trung bình	1.125	1.125	1.75	1.875

Độ dài trung bình l của mỗi mã xác định bởi

$$l = \sum_{i=1}^{4} P(a_i)n(a_i) \quad \text{(bit/ký hiệu)},$$

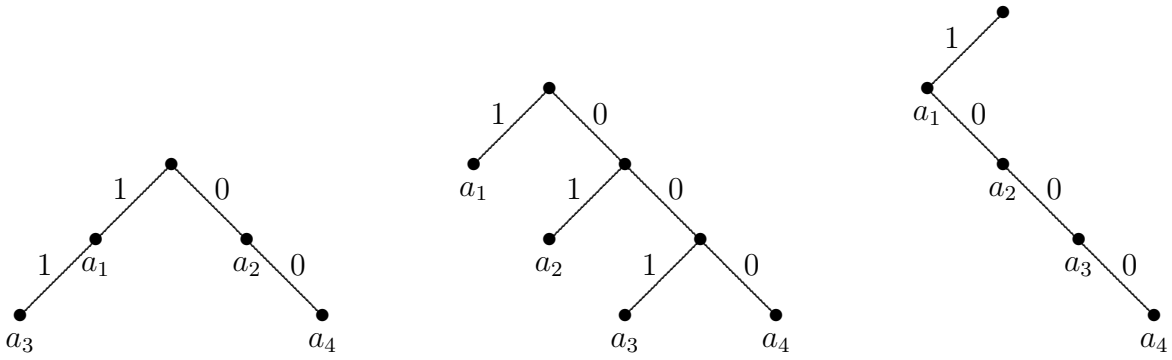
trong đó $n(a_i)$ là số các bit của từ mã a_i.

Dựa trên độ dài trung bình, mã C_1 là tốt nhất. Tuy nhiên, các mã cần phải có khả năng truyền thông tin sao cho người nhận có thể hiểu được rõ ràng. Hiển nhiên mã C_1 không có tính chất này: Vì cả hai ký hiệu a_1 và a_2 đều được gán là 1 nên khi nhận được thông tin là 1, người nhận không thể phân biệt đó là ký hiệu a_1 hay a_2. Chúng ta muốn mỗi ký hiệu được gán duy nhất một từ mã.

Mã C_2 có các ký tự được gán các chuỗi bit khác nhau cho các ký tự. Tuy nhiên, giả sử mã hoá dãy $a_2a_1a_1$ được 011 và gửi đi chuỗi bit này. Người nhận chuỗi 011 có một số cách giải mã: $a_2a_1a_1$ hoặc a_2a_3. Điều này có nghĩa là một dãy được mã hoá với bộ mã C_2 thì dãy này có thể được giải mã không trùng với thông báo ban đầu. Nói chung, đây không phải là điều chúng ta mong muốn khi thiết kế các bộ mã. Chúng ta muốn có tính chất *giải mã duy nhất*; tức là mọi dãy các từ mã chỉ có duy nhất một cách giải mã. Có thể chứng minh các mã C_3 và C_4 thoả mãn tính chất này.

Mặc dù việc kiểm tra tính duy nhất khi giải mã là khó, ta có thể chứng minh tính chất này cho bộ mã C_3 dựa trên thuộc tính của bộ mã: không có từ mã nào trong mã C_3 là "tiền tố" của từ mã khác. Bộ mã như vậy gọi là *mã tiền tố.* Một cách đơn giản để kiểm tra một bộ mã có phải là tiền tố hay không ta vẽ *cây nhị phân* (mỗi đỉnh có bậc ≤ 2) tương ứng với bộ mã. Cây được vẽ xuất phát từ một nút đơn (*nút gốc*) và mỗi nút trong có bậc ngoài nhỏ hơn hoặc bằng hai. Một trong hai nhánh con tương ứng bit 1 và nhánh còn lại

tương ứng bit 0. Để thuật tiện, ta quy ước nhánh con bên phải tương ứng 0 và nhánh con bên trái tương ứng 1. Theo cách này, các cây nhị phân tương ứng các mã C_2, C_3 và C_4 cho trong Hình 4.3. (Để đơn giản, ta sẽ không vẽ các mũi tên trong các cây nhị phân).



Hình 4.3:

Chú ý rằng ngoài nút gốc, cây nhị phân có hai loại nút: các *nút lá* có bậc ngoài bằng không; và các *nút trong* có bậc ngoài khác không. Trong bộ mã tiền tố, các từ mã chỉ tương ứng các nút lá. Mã C_4 không phải là mã tiền tố do tồn tại từ mã tương ứng nút trong. Duyệt cây từ gốc đến các nút lá cho ta biểu diễn chuỗi bit tương ứng ký hiệu. Mỗi nhánh đóng góp một bit vào từ mã của nó: bit 1 cho nhánh trái và bit 0 cho nhánh phải.

Mã tiền tố luôn luôn được giải mã duy nhất nhưng ngược lại không đúng (chẳng hạn mã C_4). Tuy nhiên có thể chứng minh rằng bộ mã có thể giải mã duy nhất tương đương với mã tiền tố theo nghĩa: số trung bình các bit biểu diễn các ký hiệu bằng nhau.

4.2.2 Mã Huffman

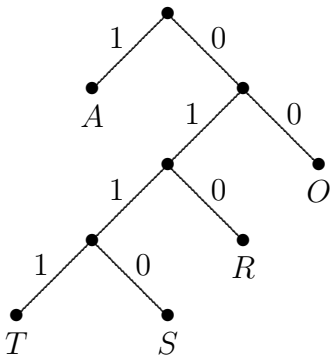
Mã Huffman là mã tiền tố và tối ưu với các xác xuất cho trước. Phương pháp xây dựng mã Huffman dựa trên hai quan sát sau:

1. Trong một bộ mã tốt ưu, các ký hiệu xuất hiện thường xuyên (có xác suất hay tần số xuất hiện lớn) sẽ có các từ mã ngắn hơn các ký hiệu ít xuất hiện.
2. Trong một bộ mã tốt ưu, hai ký hiệu xuất hiện ít nhất sẽ có các từ mã cùng độ dài.

Để xây dựng mã Huffman, chúng ta có thể biểu diễn qua cây nhị phân mà các nút lá tương ứng các ký hiệu. Duyệt cây nhị phân sẽ cho ta các từ mã của bộ mã: xuất phát từ

nút gốc và đi đến các nút lá, thêm bit 1 vào từ mã mỗi lần qua nhánh trái và bit 0 mỗi lần qua nhánh phải. Với cây trong Hình 4.4, ta có biểu diễn các ký tự qua các từ mã như sau:

Ký tự	Mã hoá
A	1
O	00
R	010
S	0110
T	0111



Hình 4.4:

Để giải mã một chuỗi bit, chúng ta bắt đầu từ gốc và di chuyển dọc theo cây cho đến khi gặp ký tự: đi theo nhánh trái nếu đó là bit 1, ngược lại đi theo nhánh phải. Chẳng hạn, chuỗi bit

01010111

tương ứng từ RAT. Với một cây xác định mã Huffman như Hình 4.4, chuỗi bit bất kỳ được giải mã duy nhất mặc dù các ký tự tương ứng với những chuỗi bit có độ dài thay đổi.

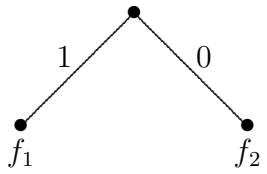
Huffman đã chỉ ra thuật toán xây dựng mã Huffman từ bảng các tần số xuất hiện của các ký tự như sau:

Thuật toán xây dựng mã Huffman

Xét chuỗi cần mã hoá s từ n ký tự với $n \geq 2$.

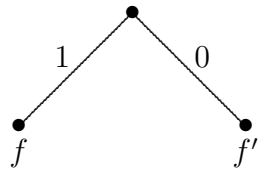
1. Xây dựng dãy tần số $f_i, i = 1, 2, \ldots, n$, xuất hiện của các ký tự trong chuỗi s.

2. Nếu $n = 2$ (giả sử $f_1 \leq f_2$), xuất cây như trong Hình 4.5 và dừng.



Hình 4.5:

3. Giả sử f và f' là hai tần số nhỏ nhất và $f \leq f'$. Tạo một danh sách tần số mới bằng cách thay f và f' bởi $f + f'$. Gọi thuật toán này sử dụng danh sách tần số mới để tạo cây T'. Thay đỉnh được gán nhãn $f + f'$ để nhận được cây T trong Hình 4.6. Xuất T.



Hình 4.6:

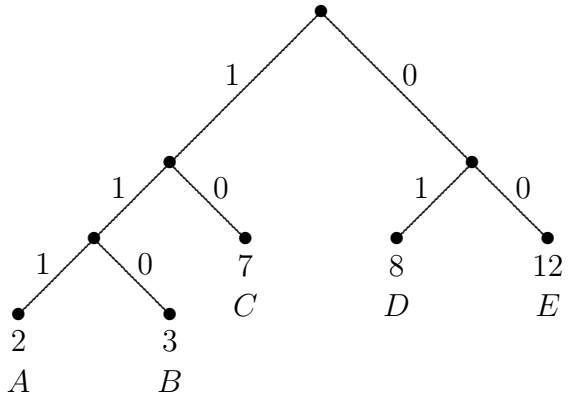
Ví dụ 4.2.1 Cho bảng tần số

Ký tự	tần số
A	2
B	3
C	7
D	8
E	12

Khi đó cây Huffman tương ứng cho trong Hình 4.7.

4.3 Cây bao trùm

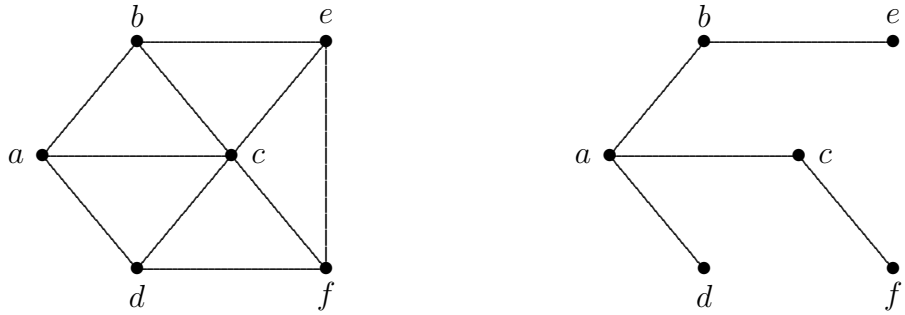
Chúng ta đã nghiên cứu riêng biệt các tính chất của một cây, trong mục này chúng ta sẽ nghiên cứu cây khi gắn nó như một đồ thị con của một đồ thị khác. Chúng ta biết rằng cho đồ thị có m cạnh, có thể xây dựng được 2^m đồ thị con khác nhau; rõ ràng là trong số



Hình 4.7:

đó có một vài đồ thị con là một cây. Chúng ta quan tâm đến một loại cây đặc biệt: "cây bao trùm". Khái niệm cây bao trùm lần đầu tiên được sử dụng và phát triển lý thuyết về cây bởi nhà vật lý người Đức Kirchoff năm 1847. Kirchoff đã sử dụng cây bao trùm nhằm giải hệ các phương trình tuyến tính để xác định cường độ dòng điện trong mỗi nhánh và xung quanh mạch của một mạng điện.

Ví dụ 4.3.1 Đồ thị trong Hình 4.8(a) có cây bao trùm trong Hình 4.8(b).



(a) (b)

Hình 4.8:

Định nghĩa 4.3.2 Cây T được gọi là *cây bao trùm* của đồ thị liên thông G nếu T là đồ thị con của G và T chứa tất cả các đỉnh của G.

Định lý 4.3.3 *Đồ thị $G = (V, E)$ có đồ thị bộ phận là một cây nếu và chỉ nếu G liên thông. Nói cách khác, cho trước một đồ thị liên thông và có n đỉnh, bao giờ ta cũng có thể bỏ đi một số cạnh của G để được một cây chứa tất cả các đỉnh của G (cây có n đỉnh).*

Chứng minh. Điều kiện cần. Nếu G liên thông thì ta thử tìm xem có cạnh nào mà khi xóa đi không làm cho đồ thị mất tính liên thông không. Nếu không có một cạnh nào như vậy thì G là một cây; nếu có một cạnh như vậy thì xóa nó đi, và ta lại đi tìm một cạnh mới để xóa... Cho tới khi không thể xóa một cạnh nào được nữa thì ta có một cây mà tập hợp các đỉnh của nó đúng bằng V.

Điều kiện đủ. Giả sử a, b là hai đỉnh trong G và do đó thuộc cây bao trùm T của G. Khi đó tồn tại dây chuyền μ trong T từ a đến b. Suy ra μ cũng thuộc G. Vậy G liên thông. $\triangleleft$

Chúng ta sẽ sử dụng *thuật toán tìm kiếm theo chiều rộng* để xây dựng cây bao trùm của đồ thị liên thông.

4.3.1 Thuật toán tìm kiếm theo chiều rộng xác định cây bao trùm

Trong thuật toán này, S ký hiệu là một dãy.

Nhập: Đồ thị liên thông $G := (V, E)$ với các đỉnh được đánh số thứ tự

$$v_1, v_2, \ldots, v_n.$$

Xuất: Cây bao trùm T.

1. [Khởi tạo] Đặt $S := [v_1]$ và T là đồ thị gồm đỉnh v_1 và không có cạnh. Ký hiệu v_1 là đỉnh gốc.
2. [Thêm cạnh] Với mỗi $x \in S$, theo thứ tự, thêm cạnh $(x, y) \in E$ và đỉnh y (theo thứ tự) vào T nếu $T \cup (x, y)$ không tạo thành chu trình. Nếu không có cạnh như vậy, dừng. T là cây bao trùm.
3. [Cập nhật S] Thay S bởi con (trong T) của S theo thứ tự. Chuyển sang Bước 2.

Để tìm cây bao trùm của đồ thị liên thông ta còn có thể dùng thuật toán *tìm kiếm theo chiều sâu* (còn gọi là *quay lui*) như sau:

4.3.2 Thuật toán tìm kiếm theo chiều sâu xác định cây bao trùm

Nhập: Đồ thị liên thông $G := (V, E)$ với các đỉnh được đánh số thứ tự

$$v_1, v_2, \ldots, v_n.$$

Xuất: Cây bao trùm T.

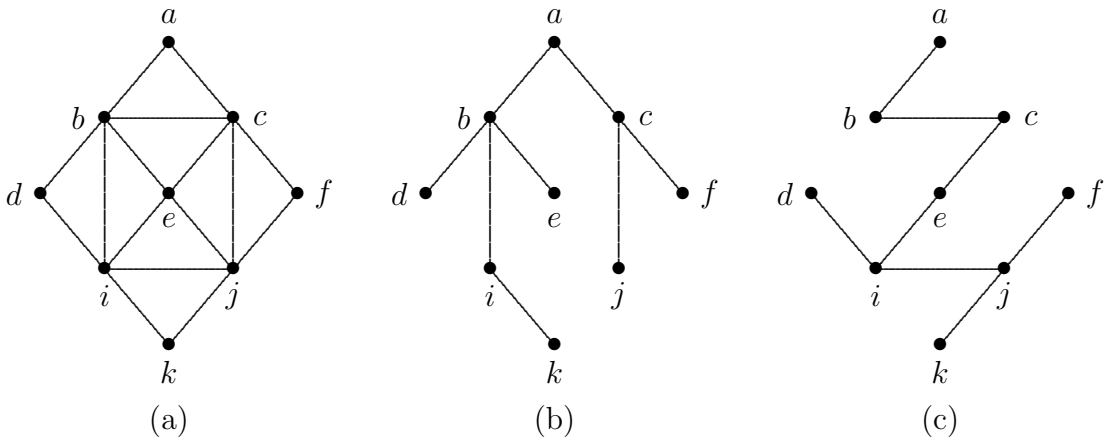
1. [Khởi tạo] Đặt $w := v_1$ và T là đồ thị gồm đỉnh v_1 và không có cạnh. Ký hiệu v_1 là đỉnh gốc.

2. [Thêm cạnh] Chọn cạnh (w, v_k) với chỉ số k nhỏ nhất sao cho việc thêm cạnh này vào T không tạo ra chu trình. Nếu không tồn tại, chuyển sang Bước 3. Ngược lại, thêm cạnh (w, v_k) và đỉnh v_k vào T; đặt $w := v_k$ và chuyển sang Bước 2.

3. [Kết thúc?] Nếu $w = v_1$, thuật toán dừng, T là cây bao trùm.

4. [Quay lui] Đặt x là cha của w (trong T); gán $w := x$ và chuyển sang Bước 2.

Ví dụ 4.3.4 Đồ thị trong Hình 4.9(a) có các cây bao trùm, Hình 4.9(b) và 4.9(c), được xây dựng theo các thuật toán tìm kiếm theo chiều rộng và chiều sâu tương ứng.



Hình 4.9: (a) Đồ thị G. (b) Cây bao trùm sinh bởi thuật toán tìm kiếm theo chiều rộng. (c) Cây bao trùm sinh bởi thuật toán tìm kiếm theo chiều sâu.

4.3.3 Tìm cây bao trùm dựa trên hai mảng tuyến tính

Để cài đặt các thuật toán tìm kiếm theo chiều rộng và chiều sâu trên đồ thị liên thông G tìm cây bao trùm T ta có thể dùng cấu trúc dữ liệu ma trận kề hay cải biên, mảng các danh sách kề $V_{out}[]$. Tuy nhiên, trong trường hợp đồ thị được biểu diễn bởi hai mảng tuyến tính α và β thì cách tiếp cận sau sẽ hiệu quả hơn. Ngoài ra, phương pháp sau này cũng cho ta một "rừng" (tập các cây bao trùm) chứa $(n - p)$ cạnh trong trường hợp đồ thị có $p > 1$ thành phần liên thông. Hiển nhiên, với những thuật toán xây dựng cây bao trùm ta có thể kiểm tra đồ thị có liên thông hay không, và nếu nó không liên thông thì có thể xác định các thành phần liên thông. Nếu mặt khác, đồ thị có trọng số thì chúng ta có thể tìm cây bao trùm có tổng trọng lượng nhỏ nhất (xem Phần 4.4). Hơn nữa chúng ta cũng có thể xây dựng hệ các chu trình độc lập dựa trên cây bao trùm của đồ thị như trong Phần 4.3.4.

Xét đồ thị vô hướng G không có khuyên n đỉnh và m cạnh. Các đỉnh được gán nhãn $v_1, v_2, \ldots, v_n$, và đồ thị xác định bởi hai mảng tuyến tính α, β, trong đó α_i và $\beta_i, i = 1, 2, \ldots, m$, là các đỉnh được liên thuộc bởi cạnh e_i.

Mỗi bước lặp của thuật toán, một cạnh mới được đưa vào kiểm tra để xác định các đỉnh của cạnh đó có xuất hiện trong cây nào đó (đã được thiết lập ở bước trước; bước đầu tiên chúng ta chưa có cây bao trùm nào). Ở bước thứ $i, 1 \leq i \leq m$, khi kiểm tra cạnh (α_i, β_i) có năm trường hợp xảy ra:

1. Nếu cả hai đỉnh không nằm trong bất cứ một cây nào đã được xây dựng ở $(i-1)$ bước trước, khi đó các đỉnh α_i, β_i được gán số thành phần liên thông là số c, sau đó tăng c lên một đơn vị.

2. Nếu α_i thuộc cây T_j còn β_i thuộc cây T_k $(j, k = 1, \ldots, c$ và $j < k)$, cạnh thứ i được sử dụng để nối hai cây này; do đó, mọi đỉnh trong cây T_k được gán là số thành phần liên thông của T_j. Giá trị c giảm một đơn vị.

3. Nếu cả hai đỉnh cùng nằm trong một cây, thì cạnh (α_i, β_i) cùng với một số cạnh khác của cây sẽ tạo thành một chu trình và do đó không xử lý trường hợp này.

4. Nếu đỉnh α_i thuộc cây T_j còn β_i không thuộc cây nào, khi đó cạnh (α_i, β_i) được thêm vào cây T_j bằng cách gán số thành phần liên thông của T_j cho đỉnh β_i.

5. Nếu đỉnh β_i thuộc cây T_k còn α_i không thuộc cây nào, khi đó cạnh (α_i, β_i) được thêm vào cây T_k bằng cách gán số thành phần liên thông của T_k cho đỉnh α_i.

Để thực hiện việc kiểm tra các đỉnh cuối của một cạnh được khảo sát có xuất hiện trong cây nào không, chúng ta xây dựng một mảng tuyến tính n phần tử Vertex[]. Khi một cạnh (v_i, v_j) nằm trong cây thứ c thì các phần tử thứ i và j của mảng này được đặt là c.

Trong các quá trình xử lý kế sau, khi một cạnh khác (α_i, β_i) được đưa vào kiểm tra, chúng ta chỉ cần kiểm tra các phần tử thứ α_i và β_i trong mảng Vertex[] có khác 0 không. Phần tử thứ q trong mảng Vertex[] bằng 0 chỉ ra rằng đỉnh thứ q này không nằm trong bất cứ cây nào. Kết thúc chương trình, mảng Vertex[] cho chúng ta biết các thành phần liên thông của đồ thị G.

Nhận xét rằng, cây không chỉ được mô tả bởi tập các đỉnh. Bởi vậy, chúng ta cần có một mảng các cạnh để xuất dữ liệu. Đặt mảng này là Edge[]. Nếu cạnh thứ i nằm trong cây thứ c, ta có Edge$[k] = c$; ngược lại, nó được đặt bằng 0. Tất cả các phần tử 0 trong mảng này tương ứng với các khuyên cô lập (tức là các cạnh không nằm trong bất cứ cây bao trùm hay rừng nào). Mảng này cùng với các mảng α và β, xác định duy nhất cây bao trùm (hoặc rừng) được sinh bởi thuật toán này.

Trong thuật toán này, vòng lặp chính thực hiện m lần. Thời gian đòi hỏi để kiểm tra các đỉnh có xuất hiện trong cây hay không là hằng số-không phụ thuộc vào n và m. Do đó thời gian thực hiện thuật toán tỉ lệ với m[1]. Trong trường hợp $m \gg n$, ta có thể giảm thời gian thực hiện bằng cách lưu trữ một biến đếm số các cạnh được đặt vào cây. Khi biến này đạt giá trị $(n-1)$ chương trình sẽ kết thúc (nếu đồ thị liên thông; trái lại ta cần kiểm tra mọi cạnh).

Thủ tục sau minh họa thuật toán tìm cây bao trùm dựa trên hai mảng tuyến tính $\alpha[]$ và $\beta[]$:

```
void SpaningTree()
{
  byte i, j, Tempt, Count = 0;
  byte Vertex[MaxVertices], Edge[MaxEdges];

  for (j = 1; j <= NumVertices; j++) Vertex[j] = 0;
  for (i = 1; i <= NumEdges; i++) Edge[i] = 0;

  for (i = 1; i <= NumEdges; i++)
  {
    if (Vertex[alpha[i]] == 0)
    {
      if (Vertex[beta[i]] == 0)
      {
        Count++;
        Edge[i] = Count;
        Vertex[alpha[i]] = Count;
        Vertex[beta[i]] = Count;
      }
      else
      {
        Edge[i] = Vertex[beta[i]];
        Vertex[alpha[i]] = Vertex[beta[i]];
      }
    }
    else
    {
      if (Vertex[beta[i]] == 0)
      {
```

[1]Thời gian đòi hỏi để trộn hai cây T_i và T_j được thực hiện trong ngôn ngữ C không phụ thuộc vào n. Tuy nhiên, có những thuật toán trộn rất hiệu quả cho phép thực hiện tiến trình này.


```
        Edge[i] = Vertex[alpha[i]];
        Vertex[beta[i]] = Vertex[alpha[i]];
      }
      else if (Vertex[alpha[i]] != Vertex[beta[i]])
      {
        Edge[i] = Vertex[alpha[i]];
        Tempt = Vertex[beta[i]];
        for (j = 1; j <= NumEdges; j++)
        {
          if (Vertex[beta[j]] == Tempt)
          {
            Vertex[alpha[j]] = Vertex[alpha[i]];
            Vertex[beta[j]] = Vertex[alpha[i]];
          }

          if (Vertex[beta[j]] == Count)
          {
            Vertex[alpha[j]] = Tempt;
            Vertex[beta[j]] = Tempt;
          }
        }
        Edge[Count] = Tempt;
        Count--;
      }
    }
  }

  printf("So thanh phan lien thong la %d ", Count);

  for (j = 1; j <= Count; j++)
  {
    printf(" \n Cay bao trum thu %d gom cac canh ", j);

    for (i = 1; i <= NumEdges; i++)
    if (Edge[i] == j) printf(" %d %d ", alpha[i], beta[i]);
  }
}
```


4.3.4 Thuật toán tìm tất cả các cây bao trùm

Việc phân tích các mạch điện về cơ bản có thể đưa về bài toán tìm tất cả các cây bao trùm của đồ thị (xem [19]). Do tầm quan trọng của nó, có nhiều thuật toán khác nhau giải quyết bài toán này. Một trong những phương pháp là hoán đổi các chu trình như sau: Xuất phát từ một cây bao trùm T nào đó. Với mỗi cạnh không nằm trên cây T, khi thêm vào cây này sẽ tạo ra duy nhất một chu trình. Xóa bỏ một cạnh bất kỳ trong chu trình này sẽ cho ta một cây bao trùm mới.

Do số các cây bao trùm là rất lớn thậm chí cả với những đồ thị nhỏ, tính hiệu quả của những thuật toán giải quyết bài toán rất quan trọng (xem [14]). Một tổng quan của các phương pháp này là một luận án tiến sĩ của Chase [12]. Chase đã chỉ ra rằng thuật toán đưa ra bởi Minty có hiệu quả nhất: liên tiếp thu gọn đồ thị bằng các phép toán xoá một cạnh và hợp nhất các đỉnh đầu cuối của nó. Từ các cây bao trùm của các đồ thị thu gọn (mà nhỏ hơn rất nhiều) ta có thể dựng được tất cả các cây bao trùm của đồ thị ban đầu. Để bảo đảm thuật toán kết thúc, các đồ thị có kích thước dưới một ngưỡng cho trước sẽ không được thu gọn thêm; thay vào đó là các cây bao trùm của chúng được suy ra.

4.3.5 Hệ cơ sở của các chu trình độc lập

Nhắc lại rằng chu số của đồ thị vô hướng G có n đỉnh, m cạnh, p thành phần liên thông bằng $c(G) = m - n + p$−chính là số cực đại các chu trình độc lập. Phần dưới đây ta xây dựng thuật toán tìm hệ cơ sở của các chu trình độc lập dựa trên cây bao trùm của đồ thị. Không giảm tổng quát, có thể giả thiết đồ thị G liên thông, vì trong trường hợp trái lại, thì ta xét từng thành phần liên thông. Ý tưởng ở đây là

1. Xây dựng cây bao trùm $T := (V, E_T)$.

2. Giả sử $e_1, e_2, \ldots, e_{m-n+1}$ là các cạnh của E mà không thuộc cây T. Khi thêm một cạnh bất kỳ trong các cạnh này, chẳng hạn cạnh e_k, vào cây T ta được một và chỉ một chu trình μ_k. Các chu trình $\mu_1, \mu_2, \ldots, \mu_{m-n+1}$ là độc lập vì mỗi chu trình chứa một cạnh không thuộc các chu trình kia; mặt khác ta có $(m - n + 1) = c(G)$ chu trình như vậy, nên ta đã xác định được hệ cơ sở của các chu trình độc lập.

Như vậy bài toán đưa về tìm các chu trình μ_k khi thêm cạnh e_k vào cây T. Trong khi kiểm tra cạnh $e_k = (\alpha_k, \beta_k)$ trong thuật toán 4.3.3, nếu điều kiện 3 đúng (tức là cả hai đỉnh α_k và β_k xuất hiện trong cùng cây T_i) thì thay cho việc loại bỏ cạnh này chúng ta cần tìm các cạnh trong T_i mà tạo thành dây chuyền giữa hai đỉnh α_k và β_k. Dây chuyền này cùng với cạnh (α_k, β_k) tạo thành một chu trình cơ bản. Vấn đề chính ở đây là tìm dây chuyền

này. Có nhiều phương pháp hiệu quả giải quyết bài toán, trong số đó thuật toán của Paton [50] là hiệu quả nhất.

Thuật toán Paton tìm hệ cơ sở của các chu trình độc lập

Chúng ta cũng sẽ kiểm tra mỗi cạnh có tạo thành chu trình với những cạnh trong cây được xây dựng trong quá trình thực hiện thuật toán, nhưng thay việc lấy các cạnh theo thứ tự tuỳ ý (như trong Thuật toán 4.3.3), ta chọn một đỉnh z và kiểm tra các cạnh liên thuộc với nó. (Đỉnh z là đỉnh vừa được thêm vào cây). Để đơn giản, ta sẽ sử dụng ma trận kề A biểu diễn đồ thị. Ký hiệu T là tập hiện hành các đỉnh trong cây được xây dựng ở bước nào đó và W là tập các đỉnh chưa được kiểm tra (tức là những đỉnh thuộc T hoặc không nhưng có ít nhất một cạnh liên thuộc với nó chưa được kiểm tra).

Chúng ta khởi đầu thuật toán bằng cách đặt $T = \{v_1\}$ và $W = V$. Đỉnh v_1 sẽ là gốc của cây. Sau quá trình khởi tạo, thực hiện các bước sau:

1. Nếu $T \cap W = \emptyset$ thuật toán dừng.
2. Nếu $T \cap W \neq \emptyset$ chọn đỉnh $z \in T \cap W$.
3. Kiểm tra z bằng cách xét mỗi cạnh liên thuộc với nó. Nếu không có cạnh nào, khử z từ W và chuyển sang Bước 1.
4. Nếu tồn tại cạnh $(z, p) \in E$ kiểm tra z có thuộc T không.
5. Nếu $p \in T$ tìm chu trình cơ bản gồm cạnh (z, p) và một dây chuyền (duy nhất) từ z đến p trong cây đang được xây dựng. Xoá cạnh (z, p) khỏi đồ thị và chuyển sang Bước 3.
6. Nếu $p \notin T$ thêm cạnh (z, p) vào cây và đỉnh p vào T. Xoá cạnh (z, p) khỏi đồ thị và chuyển sang Bước 3.

Như đã đề cập, vấn đề cơ bản là Bước 5. Chúng ta phải tìm dây chuyền từ z đến p như thế nào? Thủ tục sau sẽ trả lời câu hỏi này.

Chúng ta sử dụng một ngân xếp (stack) $TW = T \cap W$ để lưu trữ các đỉnh trong cây nhưng chưa được kiểm tra. Đỉnh được thêm gần đây nhất vào cây sẽ được chèn vào đầu ngân xếp. Mỗi lần một đỉnh được kiểm tra được lấy ra khỏi từ đỉnh của ngân xếp. Ta sử dụng thêm hai mảng tuyến tính độ dài n : mảng $l[j]$ là khoảng cách từ gốc v_1 đến đỉnh v_j trong cây bao trùm; và Pred$[j]$ là đỉnh v_i sao cho cạnh (v_i, v_j) là một cạnh trong cây với v_i gần gốc hơn v_j. Nói cách khác, Pred$[j]$ là đỉnh liền trước đỉnh v_j trong dây chuyền từ

v_1 đến v_J. Nhãn $l[J] = -1$ nếu và chỉ nếu đỉnh v_j không thuộc tập T. Khởi tạo $l[1] = 0$ và $l[j] = -1$ với mọi $j = 2, 3, \ldots, n$.

Trong Bước 5, khi xét đỉnh z và cạnh (z, p) được tìm thấy với $p \in T$. Để xác định chu trình cơ bản sinh bởi cạnh (z, p) chúng ta lần theo dây chuyền từ z đến p trong cây bằng cách tìm liên tiếp các "tiền bối" Pred[z], Pred[Prede[z]], ... cho đến khi chúng ta bắt gặp Pred[p]-tiền bối của p. Nói cách khác, như chỉ ra trong Hình **??**, chu trình cơ bản được tạo ra là

$$z, \text{Pred}[z], \text{Pred}[\text{Pred}[z]], \ldots, \text{Pred}[p], p, z.$$

Cần chú ý rằng tiền bối Pred[k] của mọi đỉnh $k \in T$ là một đỉnh mà hoặc đã được kiểm tra hoặc đang được kiểm tra. Tức là, nếu $k \in T \cap W$ thì

$$\text{Pred}[k] \notin W \quad \text{nhưng} \quad \text{Pred}[k] \in T.$$

Thủ tục FundamentalCircuits() minh họa các bước đã trình bày ở trên.

Thời gian thực hiện của thuật toán bị chặn trên bởi n^ν trong đó giá trị thực $\nu \in [2, 3]$ phụ thuộc vào cấu trúc của đồ thị cũng như cách đánh nhãn các đỉnh [50].

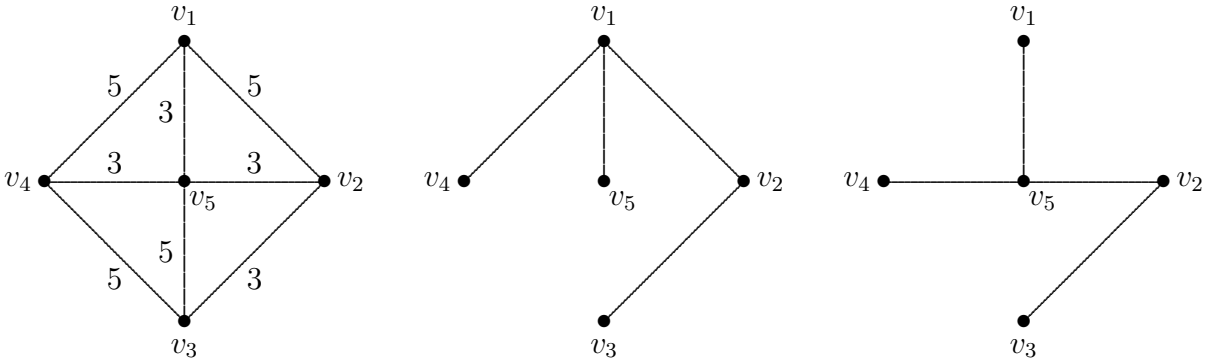
Mặc dù để đơn giản chúng ta đã giả sử G liên thông, tuy nhiên thuật toán có thể dễ dàng cải biên cho trường hợp tổng quát. Đầu tiên, thuật toán sẽ tạo ra tất cả các chu trình cơ bản trong thành phần liên thông chứa đỉnh xuất phát v_1. Sau đó ta chọn đỉnh y mà $l[y] = -1$ và bắt đầu với y là gốc của cây bao trùm thứ hai. Thuật toán lặp lại cho đến khi tất cả các đỉnh được gán nhãn l khác -1.

4.4 Cây bao trùm tối thiểu

Định nghĩa 4.4.1 Giả sử G là đồ thị có trọng số. *Cây bao trùm tối thiểu* của G là một cây bao trùm của G với trọng lượng nhỏ nhất.

Bài toán này rất hay gặp trong thông tin liên lạc và trong các ngành khác. Chẳng hạn ta đặt câu hỏi như sau: độ dài dây điện ngắn nhất cần thiết để nối n thành phố đã định là bao nhiêu? Khi đó coi các thành phố là các đỉnh của đồ thị và $w(a, b)$ là khoảng cách tính bằng km giữa các thành phố a và b. Mạng dây điện cần phải liên thông, và vì độ dài dây điện cần ngắn nhất nên nó không có chu trình: vậy mạng đó là một cây. Ở đây ta cần tìm một cây "tối thiểu" có thể được và là một đồ thị bộ phận của đồ thị đầy đủ có n đỉnh. Một ứng dụng gián tiếp: bài toán tìm cây bao trùm tối thiểu là bước trung gian trong việc tìm lời giải của bài toán người du lịch-một bài toán thường xuất hiện trong thực tế.

Cần chú ý rằng, cây bao trùm tối thiểu của đồ thị không liên quan đến cây sinh bởi tất cả các dây chuyền ngắn nhất từ một đỉnh cho trước. Do đó, đồ thị trong Hình 4.10(a), với các số trên các cạnh tương ứng các trọng lượng cạnh, cây sinh ra bởi đường đi ngắn nhất từ đỉnh cho trước, chẳng hạn v_1, trong Hình 4.10(b); ngược lại, cây bao trùm tối thiểu cho trong Hình 4.10(c).



(a) (b) (c)

Hình 4.10: (a) Đồ thị G. (b) Cây sinh bởi đường đi ngắn nhất xuất phát từ v_1. (c) Cây bao trùm nhỏ nhất.

Tìm cây bao trùm tối thiểu là một trong những bài toán của lý thuyết đồ thị có thể giải quyết triệt để. Do đó, đặt T_i và T_j là hai cây con được tạo ra trong quá trình xây dựng cây bao trùm tối thiểu. Nếu T_i được sử dụng để biểu diễn tập các đỉnh của cây con thì Δ_{ij} có thể định nghĩa là khoảng cách ngắn nhất từ một đỉnh trong T_i đến một đỉnh trong T_j; tức là

$$\Delta_{ij} := \min_{v_i \in T_i}[\min_{v_j \in T_j}\{w(v_i, v_j)\}], \qquad i \neq j. \tag{4.1}$$

Khi đó, dễ dàng chứng minh rằng lặp lại phép toán sau sẽ cho ta cây bao trùm tối thiểu:

Phép toán I. Với cây con T_s nào đó, tìm cây con T_{j^*} sao cho $\Delta_{sj^*} = \min_{T_j}[\Delta_{sj}]$, và đặt (v_s, v_{j^*}) là cạnh tương ứng giá trị Δ_{sj^*} trong (4.1) đạt được. Khi đó (v_s, v_{j^*}) thuộc cây bao trùm tối thiểu và có thể được thêm vào với các cạnh khác trong quá trình lặp để tạo ra cây bao trùm tối thiểu.

Thật vậy, giả sử các cạnh trong các cây con ở bước k nào đó thuộc cây bao trùm tối thiểu T_m ở bước cuối cùng. Giả sử cạnh (v_s, v_{j^*}) được chọn như trên không thuộc cây bao

trùm tối thiểu T_m. Theo định nghĩa, cây con T_s ở bước cuối cùng được nối với cây con khác nào đó bằng cạnh (v_i, v_j) trong đó $v_i \in T_s$ và $v_j \notin T_s$ và ngoài ra T_s phải chứa trong cây bao trùm tối thiểu T_m. Xoá cạnh (v_i, v_j) khỏi cây T_m sẽ cho ta hai thành phần liên thông và thay nó bởi cạnh (v_s, v_{j^*}) sẽ tạo một cây mới có trọng lượng nhỏ hơn trọng lượng cây T_m, mâu thuẫn. Vậy, với những giả thiết trên, ta có thể thêm cạnh (v_s, v_{j^*}) để tạo thành cây (chứa trong cây bao trùm tối thiểu) ở bước k và tiếp tục với bước $(k+1)$. Cũng cần chú ý rằng, phương pháp trên không phụ thuộc vào cây T_s được chọn. Hơn nữa, do bước khởi tạo (tức là trước khi bất cứ cạnh nào được chọn) giả thiết là không tồn tại (và do đó đúng) nên lặp lại thuật toán trên cuối cùng sẽ tạo ra cây bao trùm tối thiểu.

Hầu hết các phương pháp tìm cây bao trùm tối thiểu đều dựa trên những trường hợp đặc biệt của thủ tục trên. Đầu tiên, trong số đó là phương pháp của Kruskal [39] như sau.

4.4.1 Thuật toán Kruskal

Ý tưởng của thuật toán Kruskal tìm cây bao trùm trong đồ thị liên thông có trọng số G như sau: Khởi tạo với đồ thị T gồm tất cả các đỉnh của G và không có cạnh. Tại mỗi bước lặp chúng ta thêm một cạnh có trọng lượng nhỏ nhất lên cây T mà không tạo thành chu trình trong T. Thuật toán dừng khi T có $(n-1)$ cạnh.

1. [Khởi tạo] Giả sử T là đồ thị gồm n đỉnh và không có cạnh.
2. [Sắp xếp] Sắp xếp thứ tự các cạnh của đồ thị G theo thứ tự trọng lượng tăng dần.
3. [Thêm cạnh] Thêm cạnh (bắt đầu từ cạnh đầu tiên) trong danh sách các cạnh được sắp xếp vào cây T sao cho không tạo thành chu trình trong T khi thêm. (Cạnh được thêm vào gọi là *chấp nhận được).*
4. [Kết thúc] Nếu T có $(n-1)$ cạnh thì thuật toán dừng; T là cây bao trùm tối thiểu.

Thuật toán này thêm vào cây T những cạnh có trọng lượng nhỏ nhất chấp nhận được hơn là thêm cạnh giữa một cây con của T, chẳng hạn T_s, và một cây con nào khác (như chỉ ra trong Phép toán I). Hiển nhiên cạnh được chọn có trọng lượng nhỏ nhất giữa một cây con *nào đó* và bất kỳ cây con khác, nên nguyên tắc chọn của thuật toán Kruskal là một trường hợp đặc biệt của Phép toán I. Tuy nhiên có thể xảy ra trường hợp cạnh e được kiểm tra để thêm vào liên thuộc giữa hai đỉnh của cùng một cây con và do đó nó sẽ tạo ra một chu trình nếu thêm vào; tức là cạnh e là không chấp nhận được. Vì vậy, trong Bước 3, các cạnh cần được kiểm tra tính chấp nhận của nó trước khi thêm vào T. Tiến trình kiểm tra này có thể thực hiện hiệu quả bằng cách sử dụng thuật toán xây dựng cây bao trùm dựa trên hai mảng tuyến tính như đã trình bày trong Phần 4.3.3.

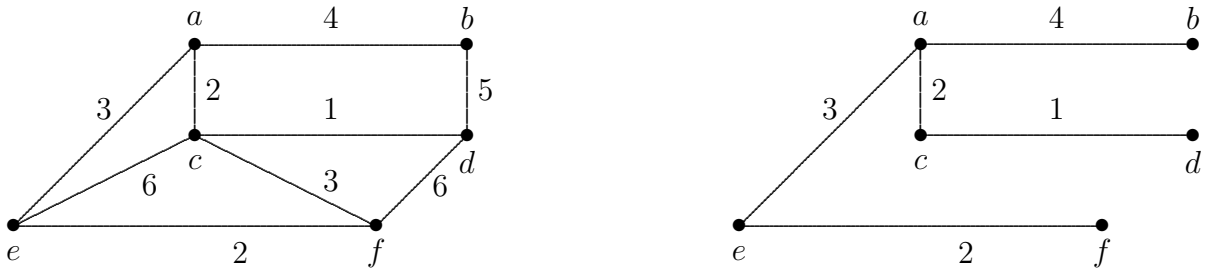
Ví dụ 4.4.2 Xét đồ thị trong Hình 4.11(a). Sắp xếp các cạnh theo trọng lượng tăng dần (sử dụng hai mảng tuyến tính α và β) ta được

k	1	2	3	4	5	6	7	8	9
α	c	a	e	a	c	a	b	c	d
β	d	c	f	e	f	b	d	e	f
Trọng lượng	1	2	2	3	3	4	5	6	6

Các cạnh (không tạo thành chu trình) được thêm vào cây T theo thứ tự là

$$(c,d),(a,c),(e,f),(a,e),(a,b).$$

T là cây bao trùm nhỏ nhất có trọng lượng 12 (Hình 4.11(b)).



(a) (b)

Hình 4.11:

Bổ đề 4.4.3 *Nếu $K_n = (V, E)$ là đồ thị đầy đủ, và nếu tất cả các các trọng lượng của các cạnh khác nhau thì tồn tại duy nhất một cây bao trùm tối thiểu $T = (V, E_T)$.*

Chứng minh. Ký hiệu $E_k := \{e_1, e_2, \ldots, e_k\}$ là tập các cạnh được thêm vào cây T trong Thuật toán 4.4.1 ở bước lặp thứ $k, 1 \leq k \leq n-1$. Hiển nhiên theo cách xây dựng, T là đồ thị có $(n-1)$ cạnh và không có chu trình nên T là cây bao trùm của K_n.

Giả sử $T' = (V, E_{T'})$ là cây bao trùm tối thiểu, ta chứng minh $E_{T'} = E_{n-1}$. Thật vậy, giả sử ngược lại tồn tại chỉ số k nhỏ nhất sao cho cạnh e_k không thuộc $E_{T'}$. Khi đó theo tính chất của cây, tồn tại tồn tại một và chỉ một chu trình μ trong $T' \cup \{e_k\}$. Trên chu trình này có một cạnh e_0 mà $e_0 \notin E_{n-1}$, vì nếu ngược lại tồn tại một chu trình, là μ, trong cây

$T-$mâu thuẫn. Nếu đặt $E_{T''} := (E_{T'} \cup \{e_k\}) \setminus \{e_0\})$ thì đồ thị $T'' := (V, E_{T''})$ không có chu trình và có $(n-1)$ cạnh nên nó là một cây.

Mặt khác $E_{k-1} \cup \{e_0\} \subset E_{T''}$ nên $E_{k-1} \cup \{e_0\}$ không chứa chu trình. Suy ra

$$w(e_0) > w(e_k).$$

Nhưng cây T'' nhận được từ cây T' bằng cách thay cạnh e_0 thành cạnh e_k nên $W(T'') < W(T')$. Mâu thuẫn vì T' là cây bao trùm tối thiểu. ◁

Định lý 4.4.4 *Thuật toán Kruskal là đúng; tức là, kết thúc thuật toán T là cây bao trùm tối thiểu.*

Chứng minh. Thật vậy trước hết ta thu xếp để mọi cạnh đều có độ dài khác nhau; chẳng hạn nếu $w(e_1) = w(e_2) = \cdots = w(e_s)$ thì thực hiện phép biến đổi:

$$\begin{aligned} w(e_1) &= w(e_1) + \epsilon, \\ w(e_2) &= w(e_2) + \epsilon^2, \\ &\vdots \\ w(e_s) &= w(e_s) + \epsilon^s, \end{aligned}$$

trong đó ϵ là số dương đủ bé sao cho không làm đảo lộn thứ tự về quan hệ giữa trọng lượng của các cạnh.

Cũng thế, ta cũng có thể thêm các cạnh f với trọng lượng đủ lớn $w(f) > \sum_{e \in E} w(e)$ và khác nhau sao cho đồ thị nhận được $K_n = (V, E')$ là đầy đủ.

Theo Bổ đề 4.4.3 tồn tại duy nhất một cây bao trùm tối thiểu T trong đồ thị K_n. Mặt khác, mọi cây bao trùm của đồ thị G có trọng lượng không vượt quá $\sum_{e \in E} w(e)$ và mọi cây bao trùm của G cũng là cây bao trùm của K_n. Suy ra T là cây bao trùm tối thiểu của G. ◁

Nhận xét rằng, có thể dùng phương pháp đối ngẫu: loại dần các cạnh có trọng lượng lớn nhất của đồ thị mà không làm mất tính liên thông của nó cho đến khi không thể loại cạnh được nữa.

Độ phức tạp tính toán của thuật toán Kruskal phụ thuộc vào Bước 2: đồ thị có m cạnh cần $m \log_2 m$ phép toán để thực hiện sắp xếp mảng theo trọng lượng tăng dần. Tuy nhiên, nói chung ta không cần duyệt toàn bộ mảng vì cây bao trùm tối thiểu gồm $(n-1)$ cạnh chấp nhận được nên chỉ cần kiểm tra $r < m$ phần tử đầu tiên của mảng. Do đó ta có thể cải tiến thuật toán này để tăng tốc độ thực hiện (xem [14], [36]).

Mặc dù có những cải tiến, nhưng thuật toán Kruskal chỉ thích hợp với những đồ thị tương đối thưa. Với những đồ thị khác, chẳng hạn đồ thị đầy đủ có số cạnh $m = n(n-1)/2$,

Prim [49] và Dijkstra [20] đã đưa ra những thuật toán khác dựa trên việc đặc biệt hoá hiệu quả hơn Phép toán I.

4.4.2 Thuật toán Prim

Thuật toán này xây dựng cây T bằng cách liên tiếp thêm các cạnh có trọng lượng nhỏ nhất liên thuộc một đỉnh trong cây đang được hình thành và một đỉnh không thuộc cây này cho đến khi nhận được cây bao trùm tối thiểu. Trong mỗi bước lặp, quá trình thêm cạnh này sẽ bảo đảm không tạo thành chu trình trong T.

Chúng ta minh họa thuật toán bằng cách sử dụng "ma trận trọng lượng" $W = (w_{ij})_{n\times n}$, trong đó

$$w_{ij} = \begin{cases} 0 & \text{nếu } i = j, \\ +\infty & \text{nếu không tồn tại cạnh } (v_i, v_j), \\ w(v_i, v_j) & \text{nếu tồn tại cạnh } (v_i, v_j). \end{cases}$$

Khởi đầu từ đỉnh v_1 và nối nó đến một đỉnh kề gần nhất (tức là đỉnh mà có phần tử nhỏ nhất trong hàng thứ nhất trong ma trận W), giả sử là v_k. Bây giờ khảo sát v_1 và v_k như một đồ thị con, và nối đồ thị con này với một đỉnh kề với nó và gần nhất (tức là đỉnh khác v_1 và v_k mà có phần tử nhỏ nhất trong tất cả các phần tử của hàng thứ nhất và thứ k trong W). Giả sử đỉnh mới là v_i. Kế tiếp, xem cây với các đỉnh v_1, v_k, v_i như một đồ thị con, và tiếp tục xử lý cho đến khi tất cả n đỉnh được nối bởi $(n-1)$ cạnh.

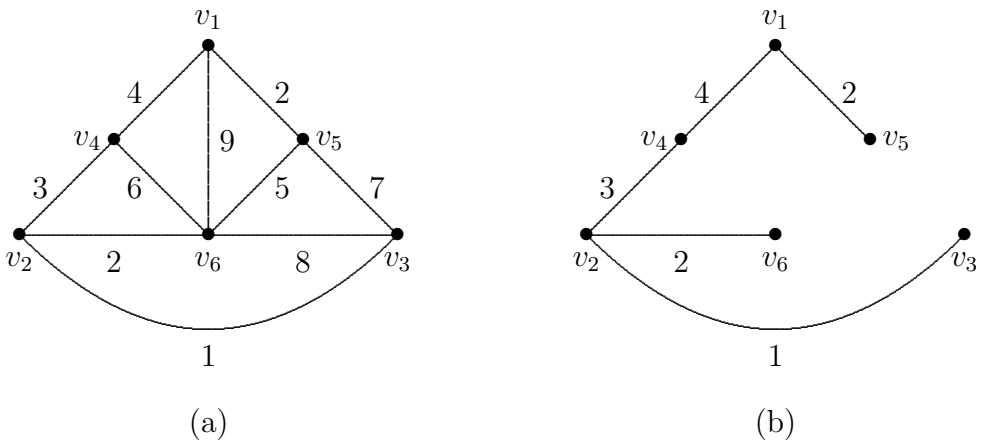
Tổng quát ta có thuật toán được trình bày như sau:

1. [Khởi tạo] Giả sử T là đồ thị chỉ có một đỉnh v_1 và không có cạnh.
2. [Kết thúc] Nếu T có $(n-1)$ cạnh thì thuật toán dừng; T là cây bao trùm tối thiểu.
3. [Thêm cạnh] Trong số tất cả các cạnh không thuộc cây T mà liên thuộc với một đỉnh trong T và không tạo thành chu trình khi thêm vào T, chọn cạnh có trọng lượng nhỏ nhất và thêm nó và đỉnh nó liên thuộc vào cây T. Chuyển sang Bước 2.

Ví dụ 4.4.5 Áp dụng Thuật toán Prim cho đồ thị trong Hình 4.12(a) với đỉnh xuất phát là v_1 ta có các cạnh được lần lượt thêm vào cây T theo thứ tự là

$$(v_1, v_5), (v_1, v_4), (v_4, v_2), (v_2, v_3), (v_2, v_6).$$

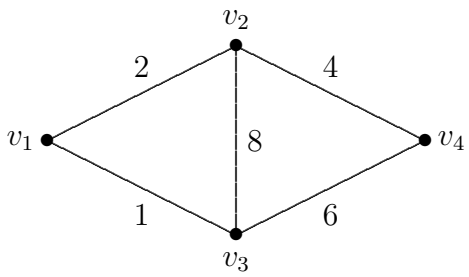
Cây bao trùm nhỏ nhất có trọng lượng 12 (Hình 4.12(b)).



Hình 4.12: (a) Đồ thị G. (b) Cây bao trùm nhỏ nhất của G sinh bởi Thuật toán Prim.

Thuật toán Prim là một ví dụ của *thuật toán tham lam* (greedy algorithm). Thuật toán tham lam là thuật toán chọn lựa tối ưu tại mỗi bước lặp mà không quan tâm đến sự chọn lựa ở bước trước. Có thể nói nguyên lý này là "thực hiện tốt nhất về mặt địa phương". Trong thuật toán Prim, vì chúng ta muốn tìm cây bao trùm tối thiểu nên tại mỗi bước lặp chúng ta thêm một cạnh có trọng lượng nhỏ nhất (nếu việc thêm không tạo thành chu trình).

Tối ưu hoá tại mỗi bước lặp không nhất thiết cho lời giải tối ưu của bài toán gốc. Tính đúng đắn của thuật toán Prim sẽ được chứng minh trong Định lý 4.4.6. Một ví dụ của thuật toán tham lam nhưng không đưa đến lời giải tối ưu như sau: xét "thuật toán tìm đường đi ngắn nhất" trong đó, mỗi bước lặp chúng ta chọn cạnh có trọng lượng nhỏ nhất (mà không tạo thành chu trình khi thêm vào) liên thuộc với đỉnh được thêm vào gần đây nhất. Nếu ta áp dụng thuật toán này với đồ thị có hướng trong Hình 4.13 để tìm đường đi ngắn nhất từ v_1 đến v_4, chúng ta sẽ chọn cung (v_1, v_3) và sau đó cung (v_3, v_4). Tuy nhiên, đây không phải là đường đi ngắn nhất từ v_1 đến v_4.



Hình 4.13:

Định lý sau chứng minh tính đúng đắn của thuật toán Prim.

Định lý 4.4.6 *Thuật toán Prim là đúng; tức là, kết thúc thuật toán T là cây bao trùm tối thiểu.*

Chứng minh. Trong chứng minh này chúng ta sẽ đặc trưng cây bởi mảng các cạnh của nó.

Theo cách xây dựng, kết thúc Thuật toán Prim, T là đồ thị con liên thông không chu trình của đồ thị G và chứa tất cả các đỉnh của G; do đó T là cây bao trùm của G.

Để chỉ ra T là cây bao trùm tối thiểu, chúng ta chứng minh bằng quy nạp rằng ở bước thứ k của Thuật toán Prim, T được chứa trong một cây bao trùm tối thiểu. Và do đó, kết thúc thuật toán T là cây bao trùm tối thiểu. Ký hiệu T_k là cây được tạo ra ở bước lặp thứ k.

Nếu $k = 1$ thì T_1 gồm một đỉnh và do đó được chứa trong mọi cây bao trùm tối thiểu. Khẳng định đúng.

Giả sử rằng ở bước thứ $(k-1)$ của Thuật toán Prim, cây T_{k-1} chứa trong cây bao trùm tối thiểu T'. Ký hiệu V_{k-1} là tập các đỉnh của T_{k-1}. Theo Thuật toán Prim, cạnh $e = (v_i, v_j)$ có trọng lượng nhỏ nhất với $v_i \in V_{k-1}$ và $v_j \notin V_{k-1}$ được thêm vào T_{k-1} để tạo ra T_k. Nếu $e \in T'$ thì T_k chứa trong cây bao trùm tối thiểu T'. Nếu $e \notin T'$ thì $T' \cup \{e\}$ chứa một chu trình μ. Chọn cạnh $(v_x, v_y) \neq e$ trên chu trình μ sao cho $v_x \in V_{k-1}$ và $v_y \notin V_{k-1}$. Ta có

$$w(x, y) \geq w(i, j).$$

Suy ra đồ thị $T" := [T' \cup \{e\}] - \{(v_x, v_y)\}$ có trọng lượng nhỏ hơn hoặc bằng trọng lượng của T'. Vì $T"$ là cây bao trùm nên $T"$ là cây bao trùm tối thiểu. Do đó ta có điều cần chứng minh. ◁

Thuật toán Prim cần kiểm tra $O(n^3)$ cạnh trong trường hợp xấu nhất (bài tập) để xác định cây bao trùm tối thiểu trong đồ thị n đỉnh. Chúng ta có thể chỉ ra Thuật toán 4.4.3 dưới đây chỉ cần kiểm tra $O(n^2)$ cạnh trong trường hợp xấu nhất. Vì K_n có n^2 cạnh nên thuật toán sau hiệu quả hơn.

4.4.3 Thuật toán Dijkstra-Kevin-Whitney

Thuật toán dưới đây tìm cây bao trùm tối thiểu trong đồ thị liên thông có trọng số hiệu quả hơn phương pháp của Prim. Phương pháp sau này đưa ra bởi Dijkstra [20], và bởi Kevin và Whitney [41].

Ý tưởng của thuật toán là gán mỗi đỉnh $v_j \notin T_s$ một nhãn (α_j, β_j), trong đó ở bước nào đó α_j là đỉnh thuộc T_s gần với đỉnh v_j nhất và β_j là độ dài cạnh (α_j, v_j). Tại mỗi bước

trong suốt quá trình thực hiện thuật toán, một đỉnh mới, chẳng hạn v_{j^*}, với giá trị β_j nhỏ nhất được thêm cùng với cạnh (α_{j^*}, v_{j^*}) vào cây T_s. Vì cây T_s được thêm đỉnh v_{j^*} nên các nhãn (α_j, β_j) của các đỉnh $v_j \notin T_s$ cần được cập nhật lại (nếu, chẳng hạn, $w(v_j, v_{j^*})$ nhỏ hơn nhãn trước đó β_j), và tiến trình được xử lý tiếp tục. Thủ tục gán nhãn này tương tự với thủ tục gán nhãn của thuật toán Dijkstra tìm đường đi ngắn nhất.

Thuật toán như sau:

1. [Khởi tạo] Đặt $T_s := \{s\}$, trong đó s là đỉnh được chọn tuỳ ý, và $A_s = \emptyset$. (A_s là tập các cạnh của cây bao trùm nhỏ nhất được tạo ra trong quá trình lặp).

2. [Gán nhãn] Với mọi đỉnh $v_j \notin T_s$ tìm đỉnh $\alpha_j \in T_s$ sao cho

$$w(\alpha_j, v_j) = \min\{w(v_i, v_j) \mid v_i \in T_s\} = \beta_j$$

   và gán nhãn của đỉnh v_j là (α_j, β_j).

   Nếu không tồn tại đỉnh α_j, tức là $\Gamma(v_j) \cap T_s = \emptyset$ thì đặt nhãn của v_j là $(0, +\infty)$.

3. [Thêm cạnh] Chọn đỉnh v_{j^*} sao cho

$$\beta_{j^*} = \min\{\beta_j \mid v_j \notin T_s\}.$$

   Cập nhật $T_s := T_s \cup \{v_{j^*}\}, A_s := A_s \cup \{(\alpha_{j^*}, v_{j^*})\}$.

   Nếu $\#T_s = n$ thuật toán dừng; các cạnh trong A_s cho cây bao trùm nhỏ nhất. Ngược lại, chuyển sang Bước 4.

4. [Cập nhật nhãn] Với mọi $v_j \notin T_s$ và $v_j \in \Gamma(v_{j^*})$ cập nhật các nhãn như sau:

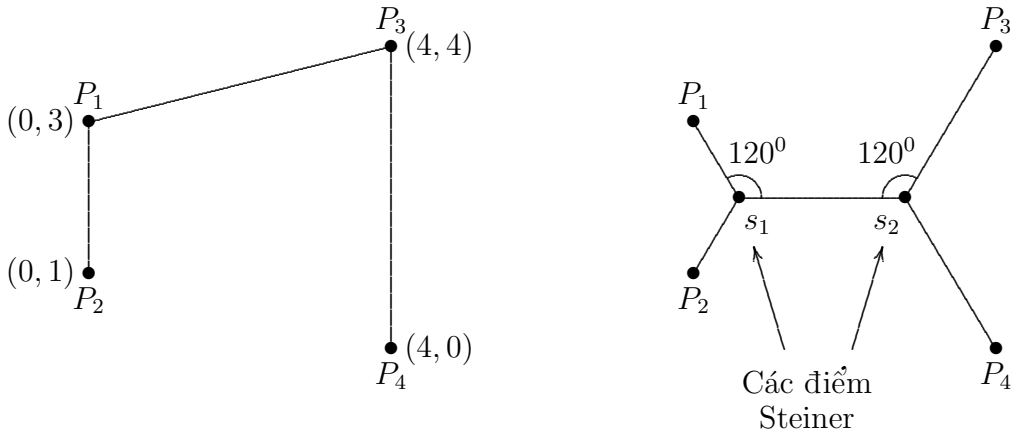
   Nếu $\beta_j > w(v_{j^*}, v_j)$ thì đặt $\beta_j = w(v_{j^*}, v_j), \alpha_j = v_{j^*}$. Chuyển sang Bước 3.

4.5 Bài toán Steiner

Trong phần trước chúng ta đã xét bài toán tìm cây bao trùm nhỏ nhất. Một bài toán có liên quan với tìm cây bao trùm nhỏ nhất nhưng khó hơn nhiều là "*bài toán Steiner trong các đồ thị*" [21]. Trong bài toán Steiner, cây bao trùm nhỏ nhất T đòi hỏi đi qua một tập con $P \subset V$ cho trước của đồ thị G. Các đỉnh khác (thuộc $V \setminus P$) có thể thuộc hoặc không thuộc cây với điều kiện cực tiểu trọng lượng của cây T. Do đó, bài toán Steiner trên đồ thị tương đương với tìm cây bao trùm nhỏ nhất trong đồ thị con $G' = (V', \Gamma')$ của G với $P \subseteq V' \subseteq V$.

Thật ra, *bài toán Steiner* được phát biểu ở dạng nguyên thuỷ là một bài toán hình học [15], trong đó tập P các điểm trên mặt phẳng được nối với nhau bằng các đoạn thẳng

sao cho tổng các đoạn thẳng được vẽ là ít nhất. Nếu giả thiết hai đoạn thẳng chỉ có thể cắt nhau tại các đỉnh của P thì bài toán trở thành tìm cây bao trùm nhỏ nhất của đồ thị tương đương có $\#P$ đỉnh với ma trận trong lượng xác định bởi ma trận khoảng cách giữa các điểm thuộc tập P. Tuy nhiên, khi các "đỉnh nhân tạo" khác (gọi là các *điểm Steiner*) được tạo thêm trên mặt phẳng thì trọng lượng của cây bao trùm nhỏ nhất tương ứng tập đỉnh mới $P' \supset P$ sẽ có thể giảm đi. Chẳng hạn, xét bốn điểm trong Hình 4.14(a) với cây bao trùm nhỏ nhất T; tuy nhiên khi thêm hai điểm mới s_1 và s_2 ta được cây bao trùm nhỏ nhất mới T' qua sáu đỉnh có trọng lượng nhỏ hơn cây T (xem Hình 4.14(b)). Do đó trong bài toán Steiner gốc, nhiều điểm Steiner được thêm vào trong mặt phẳng để tạo ra cây bao trùm nhỏ nhất qua những đỉnh cho trước P. Kết quả được đồ thị gọi là *cây Steiner nhỏ nhất*.



Hình 4.14: (a) Cây bao trùm nhỏ nhất có trọng lượng bằng 10.123. (b) Cây Steiner nhỏ nhất có trọng lượng 9.196.

Có rất nhiều công trình nghiên cứu xung quanh bài toán Steiner trên mặt phẳng Euclide, và nhiều tính chất của cây Steiner nhỏ nhất được phát hiện. Trong số đó, các tính chất sau là quan trọng nhất (chứng minh các tính chất này có thể xem [28]):

1. Đối với điểm Steiner s_i có bậc $d(s_i) = 3$. Bằng hình học dễ dàng chỉ ra rằng góc liên thuộc giữa các điểm Steiner bậc ba bằng 120^0, và có đúng ba cạnh liên thuộc với điểm Steiner s_i. Do đó điểm này là "tâm" (tâm Steiner) của "tam giác ảo" mà ba đỉnh của tam giác khác nhau được nối với đỉnh s_i trong cây bao Steiner nhỏ nhất.

   Một số điểm tạo thành các đỉnh của tam giác này có thể là các điểm Steiner khác. Chẳng hạn, trong Hình 4.14, điểm Steiner s_2 là tâm Steiner của tam giác ảo với các đỉnh là P_3, P_4 và s_1.

2. Với đỉnh $P_i \in P$ nào đó mà $d(P_i) \leq 3$. Nếu $d(P_i) = 3$ thì góc giữa hai cạnh bất kỳ

trong ba cạnh liên thuộc với P_i bằng 120^0, và $d(P_i) = 2$ thì góc giữa hai cạnh liên thuộc P_i lớn hơn hoặc bằng 120^0.

3. Số k các điểm Steiner trong cây bao trùm nhỏ nhất Steiner thoả ràng buộc $0 \leq k \leq n-2$, trong đó $n = \#P$.

Mặc dù có nhiều cố gắng trong việc giải quyết bài toán Steiner trên mặt phẳng Euclide, nhưng chúng ta chỉ mới thu được những kết quả rất hạn chế (trong trường hợp $\#P \leq 10$). Với những bài toán có kích thước lớn ta cần dùng phương pháp heuristics.

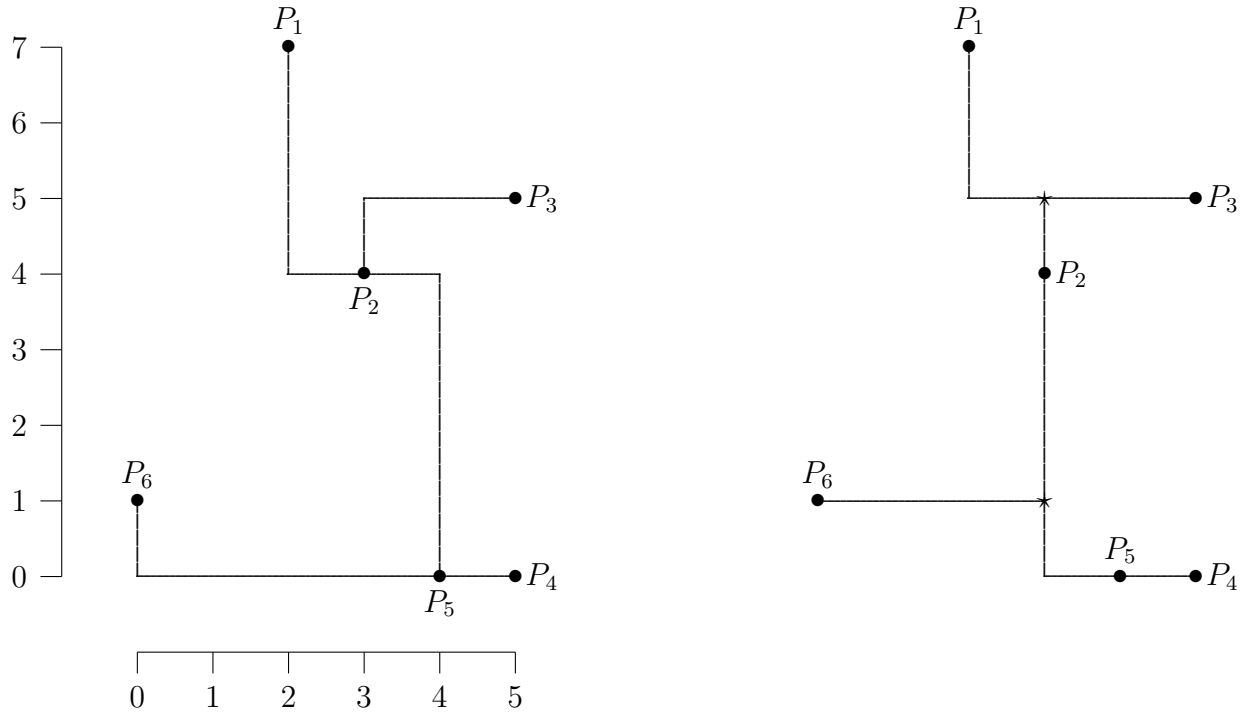
Một dạng khác của bài toán Steiner sử dụng khoảng cách *bàn cờ* thay cho khoảng cách Euclide. Bài toán này được xét lần đầu tiên bởi Hanan [32] và có liên quan với các đường dẫn được in trên các bảng mạch điện tử. Nhắc lại rằng, khoảng cách bàn cờ của hai điểm (x_1, y_1) và (x_2, y_2) trong mặt phẳng $\mathbf{R}^2$ xác định bởi

$$d_{1,2} := |x_1 - x_2| + |y_1 - y_2|.$$

Với các điều kiện này, có thể chứng minh rằng nếu kẻ một lưới các đường thẳng nằm ngang và đứng đi qua các điểm của tập P thì lời giải của bài toán Steiner có thể xác định bằng cách xem vị trí các điểm Steiner thuộc vào các giao điểm của lưới này.

Do đó, nếu đồ thị G được xác định bởi tập đỉnh V là các giao điểm của lưới và một cạnh liên thuộc hai đỉnh tương ứng đoạn thẳng thuộc lưới nối hai giao điểm. Khi đó bài toán Steiner trên mặt phẳng với khoảng cách bàn cờ trở thành bài toán Steiner trên đồ thị hữu hạn G. Hình 4.15(b) minh họa ví dụ của cây Steiner nhỏ nhất trong bài toán Steiner 6 điểm với khoảng cách bàn cờ và Hình 4.15(a) là cây bao trùm nhỏ nhất của đồ thị xác định bởi 6 điểm (để đơn giản lưới không được vẽ ra).

Bài toán Steiner trên đồ thị vô hướng tổng quát được nghiên cứu bởi nhiều tác giả và đã có những thuật toán (không hiệu quả về mặt tính toán) giải quyết chúng. Tuy nhiên (như trong trường hợp bài toán Steiner với khoảng cách Euclide), kích thước lớn nhất của một bài toán Steiner trên đồ thị có thể giải trên máy tính điện tử với thời gian cho phép vẫn không vượt quá 10 đỉnh (trong P). Nói cách khác, bài toán Steiner trên đồ thị cũng có thể xem là bài toán chưa có lời giải.



Hình 4.15: (a) Cây bao trùm nhỏ nhất có trọng lượng bằng 18. (b) Cây Steiner nhỏ nhất có trọng lượng bằng 15 ($\star$ = các điểm Steiner).