ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

Phương pháp quy hoạch động trên phân rã cây và ứng dụng trong bài toán tìm tập thống trị cực tiểu

NGUYỄN XUÂN NGHĨA
nghia.nx184166@sis.hust.edu.vn

Ngành: Công nghệ thông tin

Giảng viên hướng dẫn: TS. Trần Vĩnh Đức _____

Chữ kí GVHD

Khoa: Khoa học máy tính

Trường: Công nghệ Thông tin và Truyền thông

HÀ NỘI, 07/2023

LỜI CẢM ƠN

Trong những tháng ngày nghiên cứu và hoàn thành đồ án tốt nghiệp này, tôi xin gửi lời cảm ơn chân thành đến những người đã hỗ trợ và đóng góp ý kiến quý báu để giúp tôi hoàn thành công việc này.

Đầu tiên, tôi xin gửi lời cảm ơn sâu sắc tới tiến sĩ Trần Vĩnh Đức, người đã tận tâm hướng dẫn, truyền đạt kiến thức và động viên tôi trong suốt quá trình thực hiện đồ án. Những kiến thức và kỹ năng mà thầy đã chia sẻ đã giúp tôi hiểu sâu hơn về lĩnh vực nghiên cứu này.

Ngoài ra, tôi muốn gửi lời cảm ơn tới gia đình và bạn bè đã luôn ở bên, động viên và hỗ trợ tinh thần trong suốt hành trình học tập và nghiên cứu.

Cuối cùng, tôi xin gửi lời cảm ơn chân thành tới những người đã giúp đỡ tôi dưới bất kỳ hình thức nào trong quá trình thực hiện đồ án này.

Tôi biết ơn những đóng góp của mọi người đã giúp tôi hoàn thành đồ án này một cách thành công và xin chân thành cảm ơn!

TÓM TẮT NỘI DUNG ĐỒ ÁN

Bài toán tìm tập thống trị cực tiểu của đồ thị G được phát biểu như sau: tìm một tập hợp đỉnh D với lực lượng cực tiểu thỏa mãn mỗi đỉnh của G hoặc thuộc D hoặc kề với ít nhất một đỉnh thuộc D.

Do tính tự nhiên của nó, bài toán có rất nhiều ứng dụng thực tế. Ví dụ, ứng dụng trong việc xác định các vị trí hợp lý để xây dựng các cơ sở dịch vụ trong lĩnh vực quy hoạch; hay ứng dụng trong việc xác định những người tầm sức ảnh hưởng lớn trong một mạng xã hội (để nâng cao hiệu quả của các chiến dịch tuyên truyền, marketing). Tuy nhiên, bài toán tìm tập thống trị cực tiểu đã được chứng minh là thuộc lớp NP-khó. Do đó, người ta không hy vọng có thuật toán hiệu quả để giải một cách chính xác. Có hai cách tiếp cận để vượt qua khó khăn nay:

1. Tiếp cận giải gần đúng, tức là tìm một tập thống trị có lực lượng đủ nhỏ (không cần cực tiểu);
2. Tiếp cận giải chính xác, nhưng chỉ hạn chế trên một số lớp đồ thị phổ biến trong thực tế.

Đồ án này nghiên cứu phương pháp giải bài toán tập thống trị cực tiểu và chọn tiếp cận thứ hai. Cụ thể, đồ án tập trung vào việc phân rã một đồ thị thành dạng cây và phát triển thuật toán quy hoạch động dựa trên phân rã cây này. Mặc dù thuật toán này chậm trên các đồ thị tổng quát, nhưng lại có hiệu suất cao trên đồ thị có độ rộng cây nhỏ. May mắn là các bài toán thực tế thường thuộc vào lớp đồ thị có độ rộng cây nhỏ này.

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI 1

1.1 Đặt vấn đề 1

1.2 Mục tiêu 1

1.3 Hướng tiếp cận 1

1.4 Bố cục đồ án 1

CHƯƠNG 2. PHÂN RÃ CÂY CỦA ĐỒ THỊ 3

2.1 Giới thiệu 3

2.2 Định nghĩa 3

2.3 Tính chất 5

2.4 Phân rã cây tốt - Nice tree decomposition 6

2.5 Xây dựng phân rã cây 7

2.6 Một ví dụ chi tiết 10

2.7 Xây dựng thứ tự loại trừ 18

2.8 Xây dựng phân rã cây tốt 22

2.8.1 Khối chứa 1 khối con 24

2.8.2 Khối chứa nhiều hơn 1 khối con 28

2.9 Cài đặt và thử nghiệm 30

CHƯƠNG 3. TẬP THỐNG TRỊ 34

3.1 Định nghĩa 34

3.2 Phỏng đoán Vizing 35

3.3 Tìm tập thống trị tối thiểu thông qua phương pháp quy hoạch động 37

3.3.1 Giới thiệu 37

3.3.2 Quy hoạch động dựa trên tính đơn điệu 38

3.3.3 Chi tiết thuật toán 39

3.4 Mở rộng thuật toán 42
3.4.1 Tập thống trị thỏa mãn tính chất cho trước 42
3.4.2 Tập thống trị có trọng số - Weighted Versions of Dominating Set ... 43
3.4.3 Tập thống trị đỏ-xanh - Red-Blue Dominating Set 43
3.5 Cài đặt và thử nghiệm 43
3.5.1 Cài đặt 43
3.5.2 Thử nghiệm và đánh giá 48
CHƯƠNG 4. ỨNG DỤNG 51
4.1 Giới thiệu 51
4.2 Bài toán Xác định vị trí thiết lập các đơn vị, cơ sở 52
4.2.1 Giới thiệu 52
4.2.2 Phương pháp áp dụng 52
4.2.3 Thử nghiệm 52
4.2.4 Kết luận 57
CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 59
5.1 Kết luận 59
5.1.1 Phân rã cây - Tree Decomposition 59
5.1.2 Tập thống trị cực tiểu - Minimum Dominating Set 60
5.2 Hướng phát triển 60
TÀI LIỆU THAM KHẢO 61

DANH MỤC HÌNH VẼ

Hình 2.1 Ví dụ về một đồ thị và phân rã cây của nó 4
Hình 2.2 Tối ưu hóa phân rã cây 5
Hình 2.3 Phân rã cây và phân rã cây tốt 7
Hình 2.4 Đồ thị G 10
Hình 2.5 Đồ thị G sau khi loại đỉnh v_0 và cất v_0 vào ngăn xếp 11
Hình 2.6 Đồ thị G sau khi loại đỉnh v_3 và cất v_3 vào ngăn xếp 11
Hình 2.7 Đồ thị G sau khi loại đỉnh v_2 và cất v_2 vào ngăn xếp 12
Hình 2.8 Đồ thị G sau khi loại đỉnh v_5 và cất v_5 vào ngăn xếp 12
Hình 2.9 Đồ thị G sau khi loại đỉnh v_1 và cất v_1 vào ngăn xếp 13
Hình 2.10 Đồ thị G sau khi loại đỉnh v_4 và cất v_4 vào ngăn xếp 13
Hình 2.11 Đồ thị G sau khi loại đỉnh v_6 và cất v_6 vào ngăn xếp 14
Hình 2.12 Lấy v_6 từ ngăn xếp trả về đồ thị 14
Hình 2.13 Trả về khối đầu tiên của phân rã cây 14
Hình 2.14 Lấy v_4 từ ngăn xếp trả về đồ thị 15
Hình 2.15 Trả về khối thứ 2 của phân rã cây 15
Hình 2.16 Lấy v_1 từ ngăn xếp trả về đồ thị 15
Hình 2.17 Trả về khối thứ 3 của phân rã cây 15
Hình 2.18 Lấy v_5 từ ngăn xếp trả về đồ thị 16
Hình 2.19 Trả về khối thứ 4 của phân rã cây 16
Hình 2.20 Lấy v_2 từ ngăn xếp trả về đồ thị 16
Hình 2.21 Trả về khối thứ 5 của phân rã cây 16
Hình 2.22 Lấy v_3 từ ngăn xếp trả về đồ thị 17
Hình 2.23 Trả về khối thứ 6 của phân rã cây 17
Hình 2.24 Lấy v_0 từ ngăn xếp trả về đồ thị 17
Hình 2.25 Trả về khối thứ 7 của phân rã cây 17
Hình 2.26 Trả về khối thứ 7 của phân rã cây 18
Hình 2.27 Đồ thị G với chỉ số fill-in của từng đỉnh 20
Hình 2.28 $\pi(v_3) = 2$, loại trừ v_3 khỏi đồ thị 20
Hình 2.29 $\pi(v_2) = 3$, loại trừ v_2 khỏi đồ thị 20
Hình 2.30 $\pi(v_5) = 4$, loại trừ v_5 khỏi đồ thị 21
Hình 2.31 $\pi(v_1) = 5$, loại trừ v_1 khỏi đồ thị 21
Hình 2.32 $\pi(v_4) = 6$, loại trừ v_4 khỏi đồ thị 21
Hình 2.33 $\pi(v_6) = 7$, loại trừ v_6 khỏi đồ thị 21
Hình 2.34 $\pi(v_7) = 8$ 22
Hình 2.35 Phân rã cây tốt 23

Hình 2.36 Phân rã cây ban đầu 25
Hình 2.37 Tạo khối chứa đỉnh chung 25
Hình 2.38 Lần loại trừ đầu tiên 26
Hình 2.39 Lần loại trừ tiếp theo 27
Hình 2.40 Đồ thị con và bậc của các đỉnh 28
Hình 2.41 Phân rã cây ban đầu 29
Hình 2.42 Tạo khối con lần 1 29
Hình 2.43 Tạo khối con lần 2 30
Hình 2.44 Tạo khối con lần 3 30
Hình 2.45 Mối quan hệ giữa kích thước phân rã cây với thời gian thực thi 32

Hình 3.1 Đồ thị Petersen 34
Hình 3.2 0-regular 34
Hình 3.3 1-regular 34
Hình 3.4 2-regular 34
Hình 3.5 3-regular 34
Hình 3.6 Connected Dominating Set 35
Hình 3.7 Independent Dominating Set 35
Hình 3.8 Total Dominating Set 35
Hình 3.9 Đồ thị G với tập các đỉnh thống trị chưa được xác định 46
Hình 3.10 Phân rã cây của đồ thị G 46
Hình 3.11 Đồ thị G với tập các đỉnh thống trị đã được xác định 47

Hình 4.1 Thành phố Hà Nội dưới dạng một đồ thị với 12 đỉnh và 25 cạnh 53
Hình 4.2 Khối 1: Bắc Từ Liêm - Cầu Giấy - Nam Từ Liêm - Tây Hồ - Thanh Xuân Khối 2: Ba Đình - Cầu Giấy - Đống Đa - Tây Hồ - Thanh Xuân 55
Hình 4.3 Khối 1: Ba Đình - Đống Đa - Hai Bà Trưng - Long Biên - Thanh Xuân Khối 4: Ba Đình - Đống Đa - Hai Bà Trưng - Hoàn Kiếm - Long Biên 56
Hình 4.4 Khối 5: Đống Đa - Hoàng Mai - Long Biên - Thanh Xuân Khối 6: Hà Đông - Nam Từ Liêm - Thanh Xuân 56
Hình 4.5 3 quận Hoàn Kiếm - Tây Hồ - Thanh Xuân cùng với liên kết của 3 quận này với những quận khác được biểu diễn bằng màu đỏ . 57

DANH MỤC BẢNG BIỂU

Bảng 2.1 Các thuật toán tham lam xây dựng thứ tự loại trừ 19
Bảng 2.2 Thời gian chạy của chương trình với một số đồ thị 32

Bảng 3.1 Bảng so sánh thời gian chạy thuật toán của Telle và Proskurowski với Jochen Alber và Rolf Nieredermierer. Giả định n = 1000 và máy tính xử lí được 10^9 phép tính một giây 37
Bảng 3.2 Ví dụ đầu vào của một đồ thị có 10 đỉnh và 16 cạnh 45
Bảng 3.3 Ví dụ đầu ra là một phân rã cây có kích thước bằng 4 với 6 khối 45
Bảng 3.4 Ví dụ đầu ra của thuật toán tìm tập thống trị cực tiểu cho đồ thị G ở ví dụ trên 47
Bảng 3.5 Kết quả thu được khi chạy thuật toán sử dụng phân rã cây trên một số đồ thị 48
Bảng 3.6 So sánh kết quả kích thước tập thống trị tối thiểu thu được của các thuật toán khác nhau trên từng đồ thị 49
Bảng 3.7 So sánh thời gian thực thi của 3 thuật toán 50

Bảng 4.1 Số thứ tự của các quận 54
Bảng 4.2 Dữ liệu đầu vào cho bài toán được lấy từ hình 4.1 bảng 4.1 . 54
Bảng 4.3 Phân rã cây có 6 khối với độ rộng bằng 4 55
Bảng 4.4 Kết quả thu được cho bài toán thực tế 57

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Như đã đề cập ở trên, bài toán tìm tập thống trị cực tiểu đã được chứng minh là thuộc một trong những bài toán NP-complete. Do đó sẽ không hy vọng có thuật toán hiệu quả để tìm kết quả chính xác hoàn toàn. Với những bài toán dạng này, ta có 2 cách tiếp cận chính:

- Thuật giải gần đúng - heuristic;
- Giải bài toán hạn chế cho một số lớp đồ thị.

Hai hướng tiếp cận trên có những ưu nhược điểm riêng của chúng. Với những thuật giải gần đúng, chúng có điểm mạnh về thời gian thực thi hay kết quả có thể chấp nhận được khi mà dữ liệu đầu vào trở nên phức tạp, v.v. Tuy nhiên khi chúng ta yêu cầu một kết quả chính xác thì các thuật giải gần đúng không thể đáp ứng được. Ở điểm này, những thuật giải chính xác có ưu thế hơn, tuy nhiên chúng lại có điểm yếu là thời gian thực thi lâu, cùng với đó là khó có thể mở rộng cho những bài toán lớn hơn. Do đó, khi tiếp cận theo hướng thứ 2, ta thường chỉ xem xét chúng trên một số lớp bài toán nhất định.

1.2 Mục tiêu

Mục tiêu của đồ án này là giới thiệu về một thuật toán tìm tập thống trị cực tiểu trên đồ thị. Cùng với đó là một cách ứng dụng bài toán này vào một bài toán cụ thể là bài toán quy hoạch để chứng minh tính hiệu quả của bài toán này trong việc giải quyết các vấn đề thực tế.

1.3 Hướng tiếp cận

Đồ án này đi theo hướng tiếp cận giải bài toán trên một số lớp đồ thị nhất định. Cụ thể là những đồ thị có độ rộng cây (treewidth) nhỏ. Bằng phương pháp quy hoạch động trên phân rã cây của đồ thị, đồ án cài đặt một thuật toán chính xác để đi tìm tập thống trị cực tiểu của đồ thị. Hướng tiếp cận cụ thể sẽ như sau:

1. Xây dựng và tối ưu phân rã cây của một đồ thị.
2. Dùng quy hoạch động trên phân rã cây vừa tìm được để tìm tập thống trị cực tiểu.

1.4 Bố cục đồ án

Phần còn lại của đồ án tốt nghiệp sẽ được tổ chức như sau:

- Chương 2 trình bày những lý thuyết cơ bản của phân rã cây, bao gồm định

nghĩa, tính chất và phương pháp xây dựng một phân rã cây từ đồ thị.

- Chương 3 trình bày những lý thuyết liên quan đến tập thống trị cực tiểu trên đồ thị cũng như cách áp dụng quy hoạch động lên phân rã cây để đi tìm tập thống trị cực tiểu.
- Chương 4 tìm hiểu các ứng dụng của tập thống trị cực tiểu trong các bài toán thực tế và đi vào tìm hiểu chi tiết một ứng dụng.
- Chương 5 kết luận và hướng phát triển trong tương lai

CHƯƠNG 2. PHÂN RÃ CÂY CỦA ĐỒ THỊ

2.1 Giới thiệu

Trong đa số các trường hợp khi mà việc đưa ra một lời giải thích hợp là một điều rất khó khăn thì việc đầu vào của bài toán tuân theo một cấu trúc cụ thể sẽ giúp ích rất nhiều trong việc đưa ra lời giải.

Một số thuật toán được thiết kế để xét những đỉnh có bậc nằm trong một ngưỡng được đưa vào từ đầu vào của đồ thị. Do đó, nếu chúng ta đặt giới hạn ở đầu vào là một hằng số thì thuật toán sẽ có thể thực thi trong thời gian đa thức. Có một thuật toán hoạt động dựa trên cách thức trên được gọi là *Fixed-parameter tractable algorithms* (FPT), thuật toán này sẽ chạy hiệu quả với những tham số đầu vào có giá trị nhỏ.

Một số thuật toán khác hoạt động dựa trên dữ liệu đầu vào cần phải ở dạng thích hợp thì thuật toán đó mới có thể thực thi hiệu quả. Ví dụ như bài toán tìm đường dài nhất trong một đồ thị thông thường là một bài toán thuộc lớp các bài toán NP-hard, nên sẽ không tồn tại một thuật toán hiệu quả cho ra kết quả trong thời gian đa thức. Tuy nhiên, vẫn cùng là bài toán đó nhưng đầu vào là một đồ thị có hướng không chu trình thì chỉ đơn giản duyệt theo chiều sâu (Depth First Search) là có thể giải quyết bài toán một cách nhanh chóng. Rất nhiều thuật toán trên đồ thị có thể thực thi dễ dàng nếu đầu vào của thuật toán là một đồ thị có dạng cây. Ví dụ, trong bài toán tìm tập độc lập cực đại, nếu đầu vào của bài toán là một đồ thị dạng cây thì thuật toán sẽ có thời gian thực thi tuyến tính.

Câu hỏi ở đây là liệu có cách nào để có thể tận dụng những ưu điểm của những thuật toán dựa trên đầu vào là những dữ liệu "gần" tốt (những đồ thị có dạng gần giống cây). Việc "gần" giống cây này của dữ liệu thường được sử dụng dựa trên khái niệm *treewidth*. Nếu *treewidth* của đồ thị nhỏ, thì đồ thị đó có thể được gọi là gần giống cây. Đặc biệt, một cây có *treewidth* bằng 1. *Treewidth* được định nghĩa dựa trên khái niệm của *phân rã cây* (tree decomposition).

2.2 Định nghĩa

Một phân rã cây của đồ thị $G = (V, E)$ bao gồm một cây T và một tập con $V_t \subseteq V$ với mỗi $t \in T$. T và $\{T_t : t \in T\}$ phải thỏa mãn:

- (Node coverage) Mọi đỉnh của đồ thị G phải tồn tại trong ít nhất một đỉnh của cây T.
- (Edge coverage) Với mọi cạnh e của đồ thị G, tồn tại ít nhất một đỉnh của cây T chứa cả hai đầu mút của cạnh e.

- (Coherence) Với t_1, t_2 và t_3 là 3 đỉnh của cây T mà t_2 nằm trên đường đi từ t_1 đến t_3. Nếu một đỉnh v của đồ thị G tồn tại trong cả hai V_{t_1}, V_{t_3}, đỉnh v cũng sẽ tồn tại trong V_{t_2}.

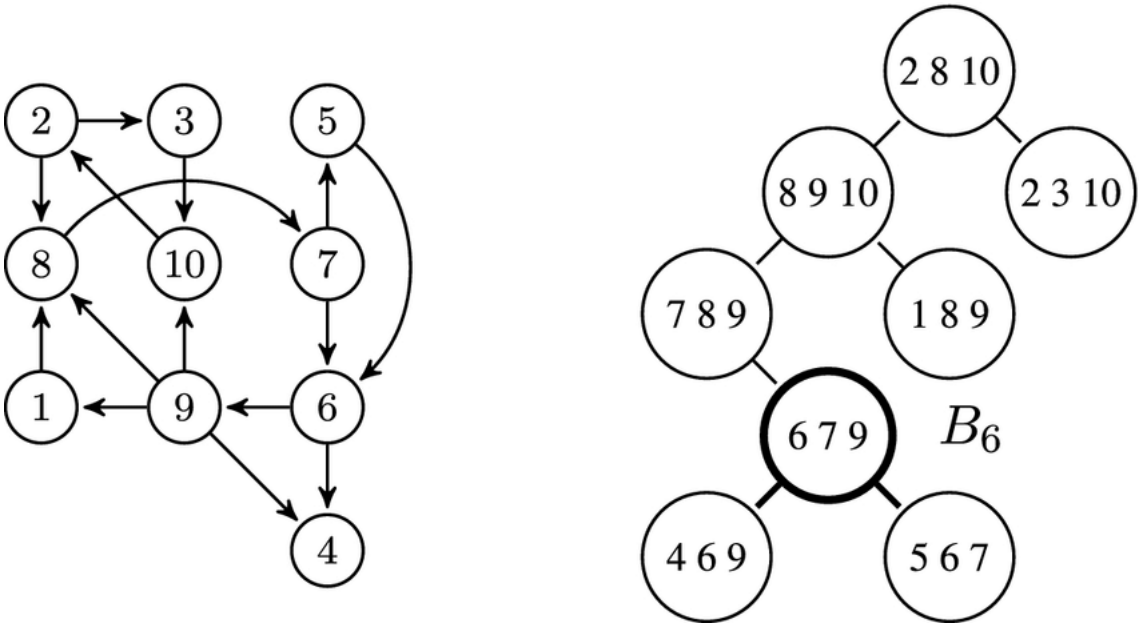
Định nghĩa 2.2.1. *Một phân rã cây của đồ thị $G = (V, E)$ là một cặp $(T = (I, F), \{V_t | t \in I\})$ với $\{V_t | t \in I\}$ là các tập con của V gọi là các khối hoặc túi.*

Ta định nghĩa độ rộng của phân rã cây $(T, \{V_t\})$ là kích thước lớn nhất của tất cả các khối V_t trừ đi 1, có nghĩa là

$$width(T, \{V_t\}) = max_{t \in T} |V_t| - 1$$

Định nghĩa 2.2.2. *Độ rộng cây của đồ thị G là độ rộng nhỏ nhất của tất cả các phân rã cây của G. Kí hiệu tw(G).*

Ví dụ 2.2.1. *Hình 3.1 dưới đây là một đồ thị và phân rã cây của nó. Phân rã cây này có độ rộng bằng 2 do kích thước lớn nhất của phân rã cây này là 3.*



Hình 2.1: Ví dụ về một đồ thị và phân rã cây của nó

2.3 Tính chất

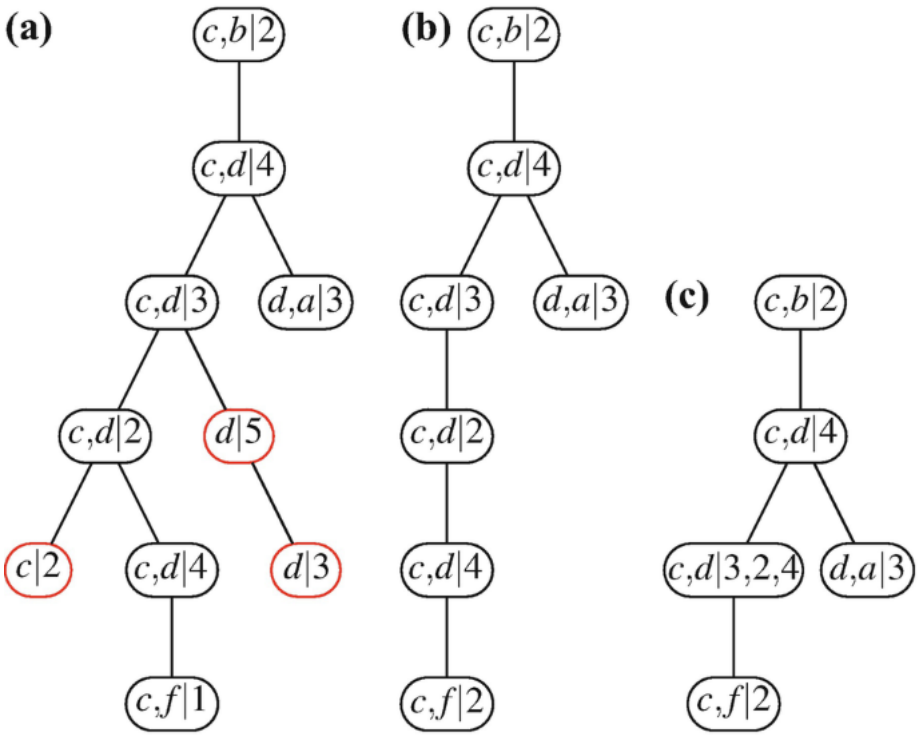
Giả sử ta xóa một đỉnh $t \in T$.

Định lý 2.3.1. *Nếu $T-t$ gồm các thành phần $T_1, ...T_d$, thì đồ thị con $G_{T_1} - V_t, G_{T_2} - V_t, ..., G_{T_d} - T_t$ không có đỉnh chung cũng như không có cạnh nối giữa chúng.*

Định lý 2.3.2. *Cho X và Y là hai thành phần liên thông của T hình thành sau khi xóa cạnh $\{x, y\}$. Nếu ta xóa tập $V_x \cap V_y$ từ V thì G tách thành hai đồ thị con $G_x - V_x \cap V_y$ và $G_y - V_x \cap V_y$. Chính xác hơn, hai đồ thị con này không chia sẻ bất kì nút nào, và không có cạnh nào nối hai đỉnh nằm trên hai đồ thị con.*

Định lý 2.3.3. *Một đồ thị liên thông G có độ rộng cây 1 khi mà chỉ khi nó là một cây.*

Khi chúng ta tìm được một phân rã cây, ta không muốn phân rã cây này có quá nhiều khối. Có một cách đơn giản để tối ưu một phân rã cây $(T, \{V_t\})$ của đồ thị G như sau. Nếu ta thấy một cạnh $\{x, y\}$ của T sao cho $V_x \subseteq V_y$, thì chúng ta có thể co cạnh $\{x, y\}$ lại và có được một phân rã cây của G dựa trên một cây có kích thước nhỏ hơn. Lặp đi lặp lại quá trình này, chúng ta thu được một phân rã cây không dư thừa: Không có cạnh (x, y) của cây sao cho $V_x \subseteq V_y$.



Hình 2.2: Tối ưu hóa phân rã cây

Định lý 2.3.4. *Mọi phân rã cây không dư thừa nào của một đồ thị n nút đều có nhiều nhất là n khối.*

Thông thường khi chúng ta tìm phân rã cây, kết quả chúng ta nhân được ban đầu sẽ là một phân rã cây dư thừa. Khi đó, chúng ta có thể sử dụng tính chất này để tối ưu phân rã cây thu được.

2.4 Phân rã cây tốt - Nice tree decomposition

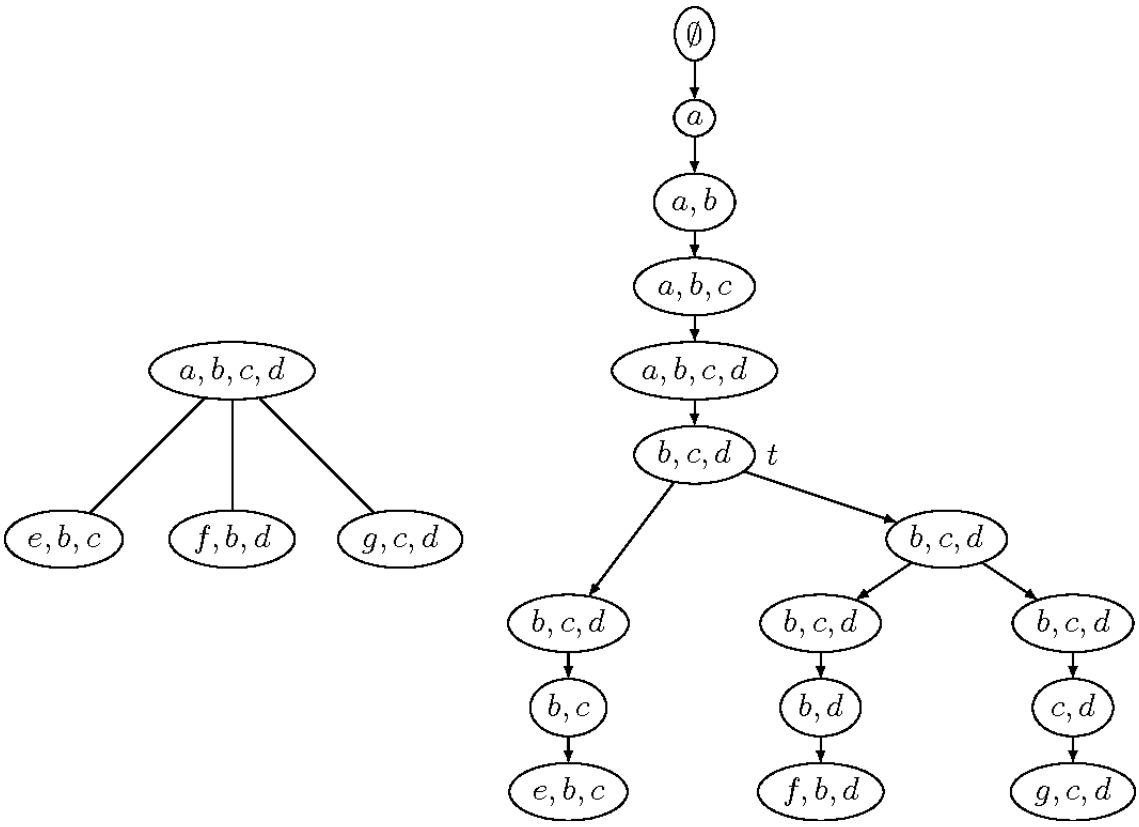
Định nghĩa 2.4.1. *[1]Một phân rã cây $\langle\{X_i|i \in I\}, T\rangle$ được gọi là một phân rã cây tốt khi mà chỉ khi cả ba điều kiện dưới đây được thỏa mãn:*

1. *Mọi khối của cây T đều chỉ có nhiều nhất là 2 khối con;*
2. *Nếu khối i có 2 khối con là j và k, thì $X_i = X_j = X_k$*
3. *Nếu khối i có một đỉnh con j, thì $|X_i| = |X_j|+1$ và $X_j \subset X_i$ hoặc $|X_j| = |X_i|+1$ và $X_i \subset X_j$.*

Định nghĩa 2.4.2. *Trong một phân rã cây tốt $(\{X_i|i \in T\}, T = (I, F))$, mọi khối sẽ thuộc một trong các trường hợp dưới đây:*

1. ***START NODE**: chỉ những khối lá của phân rã cây;*
2. ***JOIN NODE**: chỉ những khối có 2 khối con;*
3. ***FORGET NODE**: chỉ những khối cha i có 1 khối con j sao cho $|X_i| < |X_j|$;*
4. ***INTRODUCE NODE**: chỉ những khối cha i có 1 khối con j sao cho $|X_i| > |X_j|$;*

Mệnh đề 2.4.1. *Với mọi đồ thị G với treewidth bằng k sẽ có một phân rã cây tốt có treewidth bằng k. Hơn nữa, nếu n là số đỉnh của đồ thị G thì tồn tại một phân rã cây tốt với nhiều nhất là 4n khối.*



Hình 2.3: Phân rã cây và phân rã cây tốt

Cho sẵn một phân rã cây với độ rộng k và n đỉnh của một đồ thị G, ta có thể tìm ra một *nice tree decomposition* trong thời gian tuyến tính.

2.5 Xây dựng phân rã cây

Định nghĩa 2.5.1. *1. Trong đồ thị tam giác $G = (V, E)$, với mỗi một chu trình có ít nhất 4 đỉnh sẽ có ít nhất một cạnh chéo.*

2. *Một thứ tự loại trừ (Elimination order) của đồ thị $G = (V, E)$ là một song ánh $f : V \to \{1, 2, ..., n\}$. Một thứ tự loại trừ là hoàn hảo nếu với mọi $v \in V$, tập hợp những đỉnh hàng xóm có số thứ tự cao hơn v $\{w|\{v, w\} \in E \wedge f(w) > f(v)\}$ tạo thành một clique.*

3. *Đồ thị $G = (V, E)$ là một đồ thị giao điểm của các cây con của một cây, khi mà chỉ khi tồn tại một cây $T = (I, F)$, và với mỗi $v \in V$, cây con của T, $T_v = (I_v, F_v)$, sao cho với tất cả $v, w \in V$, $v \neq w$, chúng ta có $\{v, w\} \in E$ nếu và chỉ nếu T_v và T_w có ít nhất một đỉnh chung.*

Định lý 2.5.1. *Với đồ thị $G = (V, E)$, những điều sau đây là tương đương.*

1. *G là một đồ thị tam giác.*
2. *G sở hữu một thứ tự loại trừ hoàn hảo.*
3. *G là một đồ thị giao điểm của các cây con của một cây.*

4. *Tồn tại một phân rã cây* $(\{X_i|i \in I\}), T = (I, F)$ *của đồ thị G, sao cho với mỗi* $i \in I$, X_i *là tạo thành một clique của đồ thị G.*

Định nghĩa 2.5.2. *Đồ thị* $H = (V_H, E_H)$ *là một tam giác hóa của đồ thị* $G = (V_G, E_G)$*, nếu H là một đồ thị tam giác thu được bằng việc thêm 0 hoặc ít nhất 1 cạnh vào* $G = (V_G = V_H, E_G \subseteq E_H)$*. Một đồ thị tam giác tối thiểu* $H = (V, E_H)$ *của* $G = (V, E_G)$ *nếu không tồn tại một tam giác hóa của G là một đồ thị con của H.*

Ta sẽ cần một phương pháp thêm cạnh vào đồ thị để biến nó thành một đồ thị tam giác bằng việc sử dụng thứ tự loại trừ. Ta có thể cân nhắc sử dụng thuật toán Fill-in dưới đây. Thuật toán Fill-in có đầu vào là một đồ thị và thứ tự π. Thông qua từng đỉnh trong thứ tự π, thuật toán sẽ bổ sung các cạnh còn thiếu vào những đỉnh hàng xóm để tạo thành clique.

Algorithm 1 Fill-in ($G = (V, E)$, π)

```
1: H ← G;
2: for i = 1 to |V(H)| do
3:     v ← π^{-1}(i);
4:     for {(w, x) ∈ V(G) | {w, x} ⊆ N_G[v], w ≠ x, π(w) > π(v), π(x) > π(v)} do
5:         if w ∉ N_H[x] then
6:             E(H) ← E(H) ∪ {w, x};
7:         end if
8:     end for
9: end for
10: return H;
```

Thuật toán trên hoạt động như sau:

- Ta tạo một đồ thị mới H từ G.
- Ta xét từng đỉnh thuộc đồ thị H thông qua thứ tự loại trừ π. $\pi^{-1}(i)$ được hiểu là đỉnh thứ i trong thứ tự loại trừ.
- Xét từng cặp đỉnh hàng xóm của đỉnh đang xét trong đồ thị G, sao cho thứ tự loại trừ của 2 đỉnh này là lớn hơn thứ tự của đỉnh đang xét. Nếu trong đồ thị H, hai đỉnh này không kề nhau thì ta bổ sung thêm cạnh mới với hai đầu mút là hai đỉnh này vào H.
- Đầu ra của thuật toán là đồ thị đã được tam giác hóa bằng phương pháp Fill-in H.

Định lý 2.5.2. *Cho đồ thị* $G = (V, E)$*, số dương* $k \leqslant n$*. Những điều sau đây là tương đương*

1. *G có treewidth lớn nhất là k.*

2. *G có một tam giác hóa H sao cho kích thước tối đa của một clique trong H là k + 1.*
3. *Tồn tại một thứ tự loại trừ π sao cho kích thước tối đa của một clique trong đồ thị fill-in của G theo π lớn nhất là k.*
4. *Tồn tại một thứ tự loại trừ π, sao cho không đỉnh $v \in V$ nào có nhiều hơn k hàng xóm có thứ tự trong π cao hơn v trong đồ thị fill-in của G dựa trên π.*

Định lý 2.5.3. *Cho đồ thị $G = (V, E)$ và đỉnh $v \in V$. Việc loại bỏ đỉnh v bao gồm các thao tác thêm vào các cạnh nối 2 đỉnh hàng xóm không kề nhau của v rồi sau đó loại bỏ v.*

Mệnh đề 2.5.1. *Cho đồ thị $G = (V, E)$ và thứ tự loại trừ của đồ thị G π. Cho $H = (V, E_H)$ là đồ thị fill-in của đồ thị G. Giả sử $V = \{v_1, v_2, ..., v_n\}$ và với mọi $v_i \in V$, $\pi(v_i) = i$, thuật toán PermutationToTreeDecomposition dưới đây có đầu ra là một phân rã cây $(\{X_v| \in V\}, T = (V, F))$ với đầu vào là đồ thị G và thứ tự các đỉnh $(v_1, v_2, ..., v_n)$, sao cho:*

1. *Với mọi $v_i \in V$, X_{v_i} là tập hợp bao gồm v_i và các đỉnh hàng xóm có thứ tự cao hơn v_i trong H.*
2. *Kích thước của phân rã cây thu được bằng kích thước tối đa của một clique trong H trừ đi 1.*

Algorithm 2 PermutationToTreeDecomposition($G = (V, E)$, $V = (v_1, v_2, ..., v_n)$)

if $n = 1$ **then**
    return $X_{v_1} = \{v_1\}$;
end if
$V' \leftarrow V \setminus \{v_1\}$;
$E' \leftarrow E \setminus \{(v_1, v_i)|i \in V \wedge i \neq v_1\}$;
$(\{X_w|w \in V'\}, T' = (V', F')) \leftarrow PermutationToTreeDecomposition(G' = (V', E'), (v_2, v_3, ..., v_n))$;
$V_n \leftarrow N_G[v_1]$;
$v_j \leftarrow min_{v_t \in V_n} \pi^{-1}(t)$;
$X_{v_1} \leftarrow N_G[v_1]$;
return $(\{X_v|v \in V\}, T = (V, F))$ with $F = F' \cup \{v_1, v_j\}$;

Thuật toán trên hoạt động như sau:

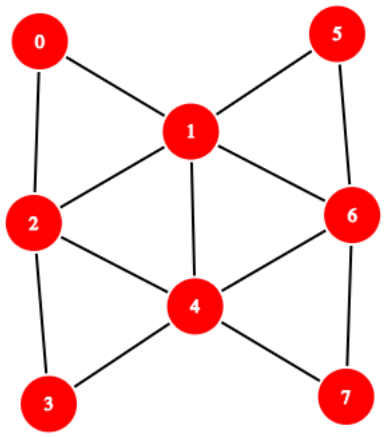
- Với đồ thị bao gồm duy nhất một đỉnh, thuật toán trả về kết quả là một khối duy nhất chứa đỉnh v_1.
- Ta tạo một đồ thị mới $G' = (V', E')$, với V' và E' là tập đỉnh và cạnh của G sau

khi đã loại bỏ đỉnh v_1. Gọi đệ quy thuật toán PermutationToTreeDecomposition với đầu vào là đồ thị G' và danh sách đỉnh $V' = (v_2, v_3, ..., v_n)$. Đầu ra là một phân rã cây T'.

- v_j là đỉnh hàng xóm của v_1 có chỉ số fill-in thấp nhất.
- Ta tạo khối mới X_{v_1} từ những đỉnh hàng xóm của v_1.
- Đầu ra của thuật toán là một phân rã cây $(\{X_v | v \in V\}, T = (V, F))$ với $F = F' \cup \{v_1, v_j\}$.

2.6 Một ví dụ chi tiết

Xét đồ thị G

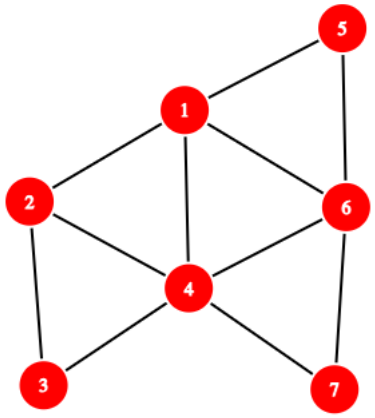


Hình 2.4: Đồ thị G

Ta có thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7)$.

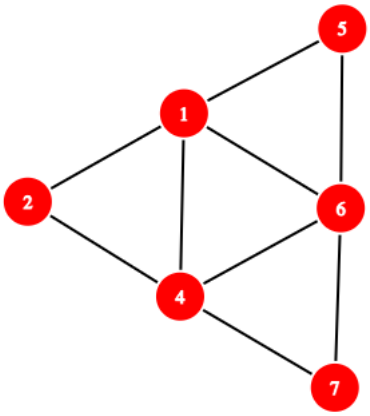
Ta thực hiện thuật toán PermutationToTreeDecomposition cho đồ thị G với thứ tự loại trừ trên, tạo đồ thị G' bằng việc lần lượt loại trừ các đỉnh dựa trên thứ tự loại trừ

- Loại trừ đỉnh 0, cất 0 vào ngăn xếp rồi gọi lại thuật toán với thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7) \setminus \{0\}$.



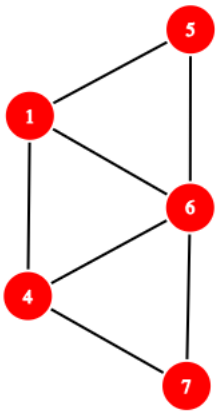
Hình 2.5: Đồ thị G sau khi loại đỉnh v_0 và cất v_0 vào ngăn xếp

- Loại trừ đỉnh 3, cất 3 vào ngăn xếp rồi gọi lại thuật toán với thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7) \setminus \{0, 3\}$.



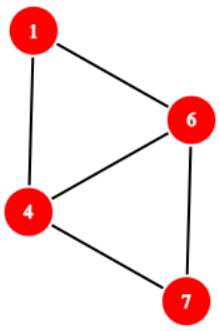
Hình 2.6: Đồ thị G sau khi loại đỉnh v_3 và cất v_3 vào ngăn xếp

- Loại trừ đỉnh 2, cất 2 vào ngăn xếp rồi gọi lại thuật toán với thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7) \setminus \{0, 3, 2\}$.



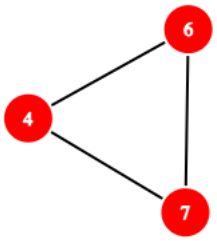
Hình 2.7: Đồ thị G sau khi loại đỉnh v_2 và cất v_2 vào ngăn xếp

- Loại trừ đỉnh 5, cất 5 vào ngăn xếp rồi gọi lại thuật toán với thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7) \setminus \{0, 3, 2, 5\}$.



Hình 2.8: Đồ thị G sau khi loại đỉnh v_5 và cất v_5 vào ngăn xếp

- Loại trừ đỉnh 1, cất 1 vào ngăn xếp rồi gọi lại thuật toán với thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7) \setminus \{0, 3, 2, 5, 1\}$.



Hình 2.9: Đồ thị G sau khi loại đỉnh v_1 và cất v_1 vào ngăn xếp

- Loại trừ đỉnh 4, cất 4 vào ngăn xếp rồi gọi lại thuật toán với thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7) \setminus \{0, 3, 2, 5, 1, 4\}$.

Hình 2.10: Đồ thị G sau khi loại đỉnh v_4 và cất v_4 vào ngăn xếp

- Loại trừ đỉnh 6, cất 6 vào ngăn xếp rồi gọi lại thuật toán với thứ tự loại trừ $\pi = (0, 3, 2, 5, 1, 4, 6, 7) \setminus \{0, 3, 2, 5, 1, 4, 6\}$.

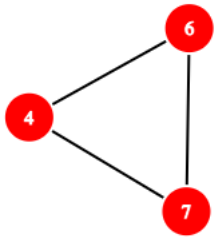
Hình 2.11: Đồ thị G sau khi loại đỉnh v_6 và cất v_6 vào ngăn xếp

Sau khi G' chỉ còn duy nhất một đỉnh, trả về phân rã cây đầu tiên gồm đỉnh cuối cùng là v_7, $X_7 = N_G[v_7] = \{v_7\}$. Sau đó lấy thông tin từ ngăn xếp ra:

Hình 2.12: Lấy v_6 từ ngăn xếp trả về đồ thị

Hình 2.13: Trả về khối đầu tiên của phân rã cây

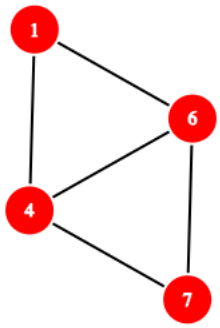
Đỉnh v_6 có hàng xóm duy nhất là v_7, do đó đỉnh hàng xóm có chỉ số nhỏ nhất trên thứ tự loại trừ của v_6 là v_7. Từ đó ta có $X_6 = N_G[v_6] = \{v_6, v_7\}$ và cạnh (X_6, X_7). Tiếp tục lấy thông tin từ ngăn xếp:



Hình 2.14: Lấy v_4 từ ngăn xếp trả về đồ thị

Hình 2.15: Trả về khối thứ 2 của phân rã cây

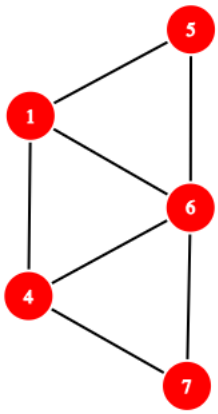
Đỉnh v_4 có hai đỉnh hàng xóm là v_6, v_7, trong đó $\pi(v_7) = 8 > \pi(v_6) = 7$, do đó phân rã cây có thêm đỉnh $X_4 = N_G[v_4] = \{v_4, v_6, v_7\}$ và cạnh (X_4, X_6). Tiếp tục lấy thông tin từ ngăn xếp ra:



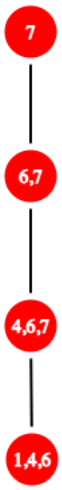
Hình 2.16: Lấy v_1 từ ngăn xếp trả về đồ thị

Hình 2.17: Trả về khối thứ 3 của phân rã cây

Đỉnh v_1 có hai đỉnh hàng xóm là v_4, v_6, trong đó $\pi(v_4) = 6 < \pi(v_6) = 7$, do đó phân rã cây thu được thêm một đỉnh $X_1 = N_G[v_1] = \{v_1, v_4, v_6\}$ và cạnh (X_1, X_4). Tiếp tục lấy thông tin từ ngăn xếp ra:

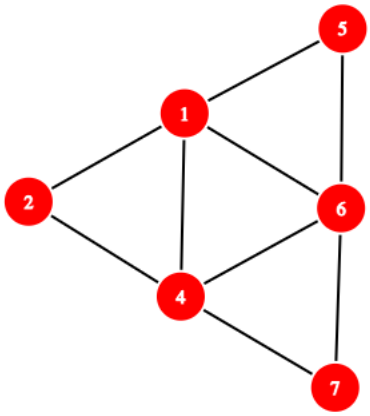


Hình 2.18: Lấy v_5 từ ngăn xếp trả về đồ thị

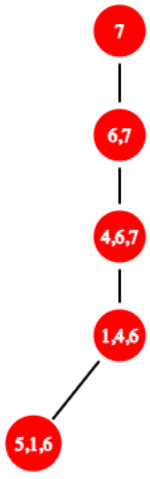


Hình 2.19: Trả về khối thứ 4 của phân rã cây

Đỉnh v_5 có hai đỉnh hàng xóm là v_1, v_6, trong đó $\pi(v_1) = 5 < \pi(v_6) = 7$, do đó phân rã cây có thêm đỉnh $X_5 = N_G[v_5] = \{v_5, v_1, v_6\}$ và cạnh (X_5, X_1). Tiếp tục lấy thông tin từ ngăn xếp:

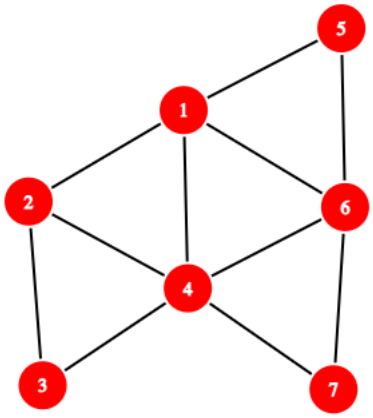


Hình 2.20: Lấy v_2 từ ngăn xếp trả về đồ thị

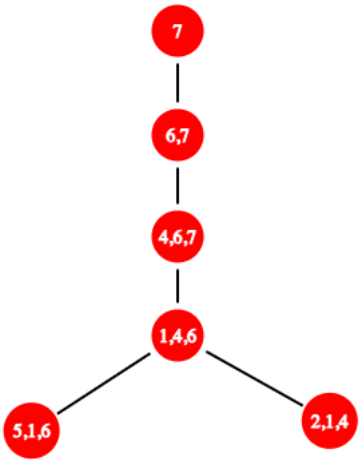


Hình 2.21: Trả về khối thứ 5 của phân rã cây

Đỉnh v_2 có hai hàng xóm là v_1, v_4, trong đó $\pi(v_1) = 5 < \pi(v_4) = 6$, do đó phân rã cây có thêm đỉnh $X_2 = N_G[v_2] = \{v_2, v_1, v_4\}$. Tiếp tục lấy thông tin từ ngăn xếp:

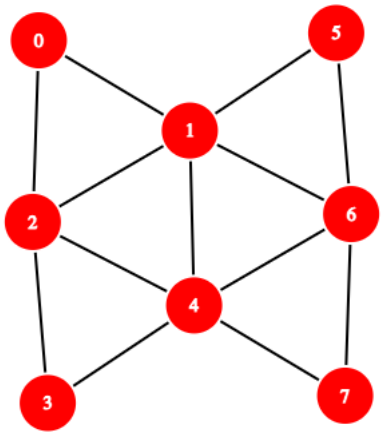


Hình 2.22: Lấy v_3 từ ngăn xếp trả về đồ thị

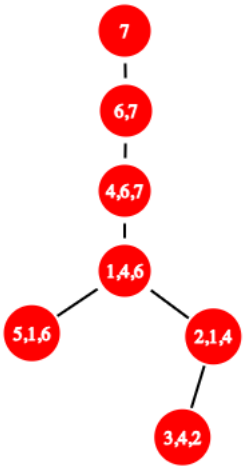


Hình 2.23: Trả về khối thứ 6 của phân rã cây

Đỉnh v_3 có hai hàng xóm là v_2, v_4, trong đó $\pi(v_2) = 3 < \pi(v_4) = 6$, do đó phân rã cây có thêm đỉnh $X_3 = N_G[v_3] = \{v_3, v_2, v_4\}$. Tiếp tục lấy thông tin từ ngăn xếp:

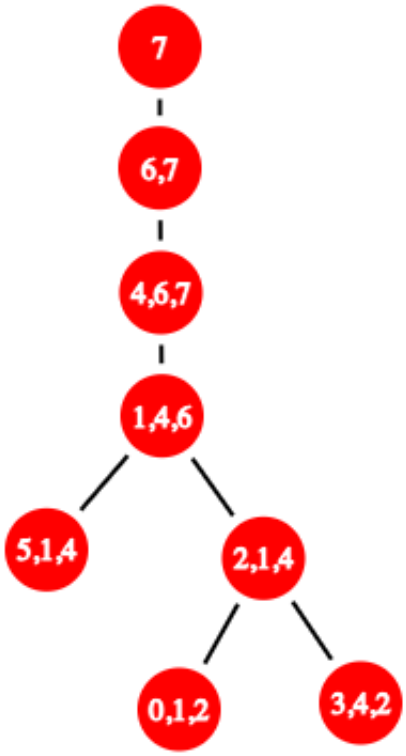


Hình 2.24: Lấy v_0 từ ngăn xếp trả về đồ thị



Hình 2.25: Trả về khối thứ 7 của phân rã cây

Đỉnh v_0 có hai đỉnh hàng xóm là v_1, v_2, trong đó $\pi(v_1) = 5 > \pi(v_2) = 3$, do đó phân rã cây có thêm đỉnh $X_0 = N_G[v_0] = \{v_0, v_1, v_2\}$ và cạnh (X_0, X_2):



Hình 2.26: Trả về khối thứ 7 của phân rã cây

2.7 Xây dựng thứ tự loại trừ

Ta có thể thấy ở 2 thuật toán đã đề cập phía trên, việc xác định được thứ tự loại trừ rất quan trong trong việc xây dựng một phân rã cây tối ưu. Do đó ta cần một phương pháp để có thể thu được một thứ tự loại trừ phù hợp với những tiêu chí mà ta muốn. Ta có thể tham khảo thuật toán GreedyX dưới đây.

Algorithm 3 GreedyX($G = (V, E)$)

1: $H \leftarrow G$;
2: **for** $i = 1$ to n **do**
3: $v \leftarrow getByX(H)$;
4: $\pi(i) \leftarrow v$;
5: $V_n \leftarrow N_H[v]$;
6: $V \leftarrow V \setminus \{v\}$;
7: Add edges into H in order to form a clique with V_n;
8: **end for**
9: **return** π;

Thuật toán trên hoạt động như sau:

- Ta xem xét tất cả các đỉnh có trong đồ thị H. Tại mỗi giá trị i, ta tìm đỉnh v phù hợp với tiêu chí X. Gán phần tử thứ i của thứ tự loại trừ π là đỉnh v.
- Ta loại bỏ đỉnh v khỏi H. Sau đó thêm cạnh vào đồ thị H sao cho các đỉnh hàng xóm của v tạo thành 1 clique.
- Thuật toán trả về thứ tự loại trừ π.

Với mỗi tiêu chí X khác nhau, ta có thể thu được một thuật toán khác nhau để xây dựng thứ tự loại trừ π. Chẳng hạn như một tiêu chí thường được sử dụng là chọn đỉnh có bậc nhỏ nhất, ta có thể gọi là GreedyDegree. Chậm hơn một chút, nhưng cho ta kết quả có phần tốt hơn là GreedyFillIn, chọn đỉnh có số lượng cạnh fill-in nhỏ nhất.

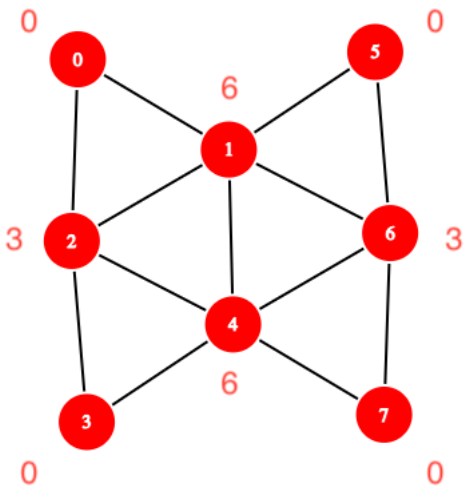
GreedyDegree và GreedyFillIn là những Heuristic khá đơn giản, cho thấy được hiệu quả rất tốt khi ứng dụng cho những nhiều bài toán điển hình. Hai cách tiếp cận kể trên cùng với những cách tiếp cận khác sẽ được thống kê trong bảng dưới đây:

Thuật toán	Đỉnh được chọn
GreedyDegree	$v = argmin_u \delta_H(u)$
GreedyFillIn	$v = argmin_u \phi_H(u)$
GreedyDegree+FillIn	$v = argmin_u \delta_H(u) + \phi_H(u)$
GreedySparsestSubgraph	$v = argmin_u \phi_H(u) - \delta_H(u)$
GreedyFillInDegree	$v = argmin_u \delta_H(u) + \frac{1}{n^2}\phi_H(u)$
GreedyDegreeFillIn	$v = argmin_u \phi_H(u) + \frac{1}{n}\delta_H(u)$

Bảng 2.1: Các thuật toán tham lam xây dựng thứ tự loại trừ

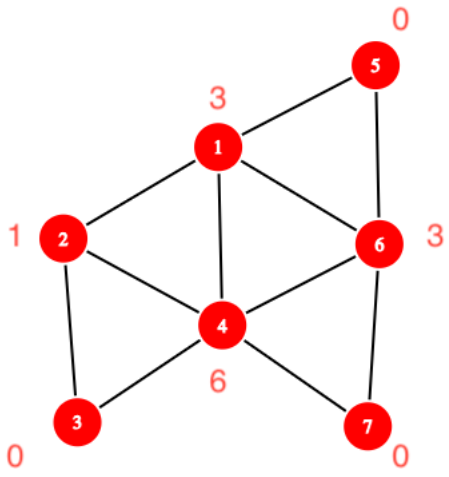
Giả sử ta sử dụng tiêu chí chọn đỉnh có số fill-in nhỏ nhất cho đồ thị G trong ví

dụ ở mục 2.6, ta có số fill-in của từng đỉnh như sau:

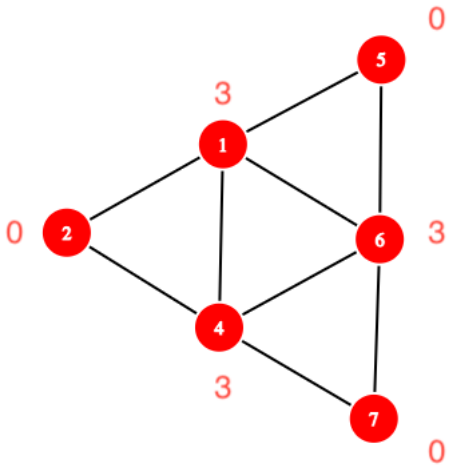


Hình 2.27: Đồ thị G với chỉ số fill-in của từng đỉnh

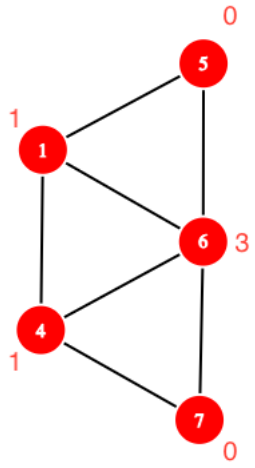
Ta duyệt các đỉnh của đồ thị theo thứ tự từ điển và chọn đỉnh v_0 dựa theo tiêu chí đỉnh có chỉ số fill-in nhỏ nhất. Ta được $\pi(v_0) = 1$. Sau đó ta loại trừ v_0 ra khỏi đồ thị rồi tính toán lại chỉ số fill-in của các đỉnh liên quan, ta được:



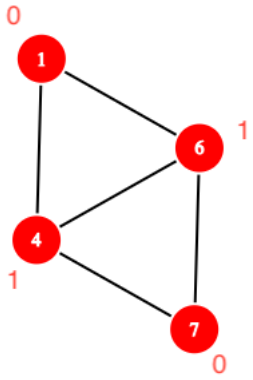
Hình 2.28: $\pi(v_3) = 2$, loại trừ v_3 khỏi đồ thị



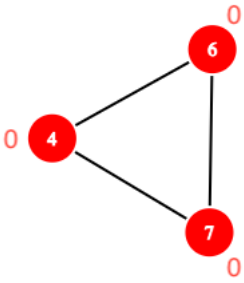
Hình 2.29: $\pi(v_2) = 3$, loại trừ v_2 khỏi đồ thị



Hình 2.30: $\pi(v_5) = 4$, loại trừ v_5 khỏi đồ thị



Hình 2.31: $\pi(v_1) = 5$, loại trừ v_1 khỏi đồ thị



Hình 2.32: $\pi(v_4) = 6$, loại trừ v_4 khỏi đồ thị

Hình 2.33: $\pi(v_6) = 7$, loại trừ v_6 khỏi đồ thị

Hình 2.34: $\pi(v_7) = 8$

Vậy thứ tự loại trừ của đồ thị G theo tiêu chí fill-in là $\pi = (v_0, v_3, v_2, v_5, v_1, v_4, v_6, v_7)$.

Định nghĩa 2.7.1. *Đỉnh $v \in V$ là đơn hình (simplicial) trong $G = (V, E)$, nếu mà chỉ nếu tập các đỉnh hàng xóm của v tạo thành một clique trong G. Một đỉnh $v \in V$ gần đơn hình, nếu tồn tại một đỉnh hàng xóm w, sao cho tập đỉnh hàng xóm của v loại trừ w, $N_G[v] - \{w\}$, tạo thành 1 clique trong G.*

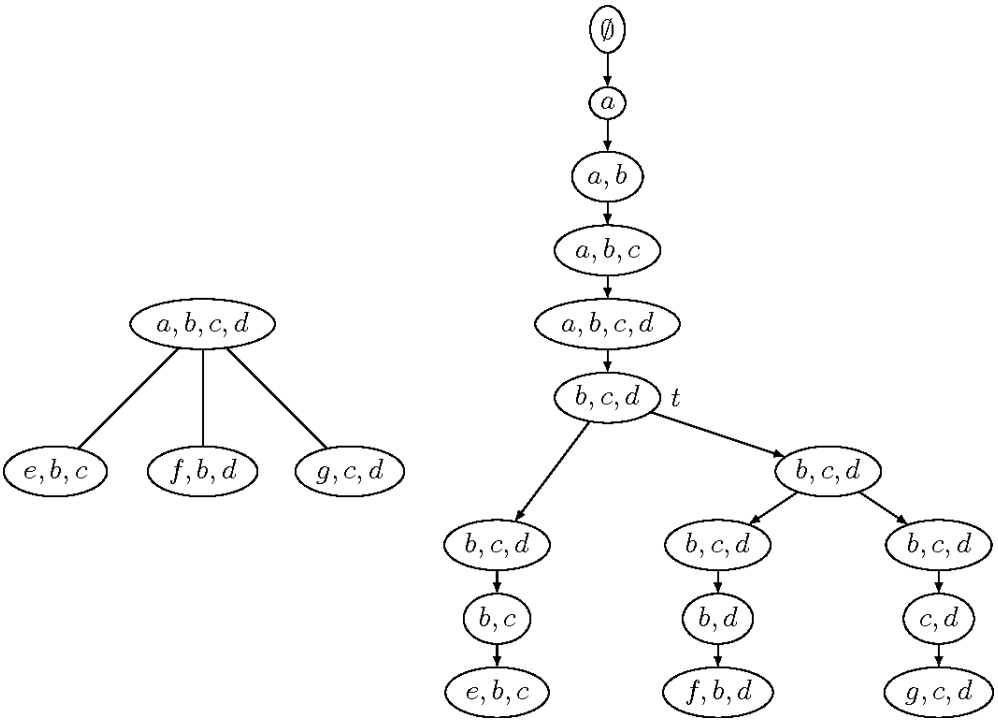
2.8 Xây dựng phân rã cây tốt

Ta nhắc lại định nghĩa của một phân rã cây tốt (*nice decomposition tree*), một phân rã cây tốt là một phân rã cây thỏa mãn:

1. Mỗi khối trong một phân rã cây tốt chỉ chứa nhiều nhất là 2 khối con.
2. Với phân rã cây (T, E), $t_i \in T$ có chứa 2 khối con $t_j, t_k \in T$, ta có X là tập đỉnh chứa trong các khối t_i, t_j, t_k thì $X_i = X_j = X_k$.
3. Với phân rã cây (T, E), $t_i \in T$ có 1 khối con $t_j \in T$, ta có X là tập đỉnh chứa trong các khối t_i, t_j, một trong các điều kiện dưới đây cần được thỏa mãn:
   - $|X_i| = |X_j| + 1$ và $X_j \subset X_i$;
   - $|X_j| = |X_i| + 1$ và $X_i \subset X_j$;

Hình dưới đây mô tả một ví dụ của một phân rã cây và phiên bản phân rã cây tốt của nó. Ta có thể thấy số lượng khối trong phân rã cây đã tăng lên nhiều lần,

điều này xảy ra là vì sự khác nhau của các đỉnh có trong mỗi khối kề nhau. Sự khác nhau này càng lớn thì lượng đỉnh cần được thêm vào giữa chúng càng lớn và ngược lại.



Hình 2.35: Phân rã cây tốt

Bản chất của việc chuyển đổi từ một phân rã cây bình thường thành một phân rã cây tốt chính là việc thêm vào giữa hai khối ban đầu những khối mới nhằm thỏa mãn những điều kiện cần có của một phân rã cây tốt.

- Đối với những khối có 1 khối con, ta xác định những đỉnh chung có ở cả hai khối rồi thêm một khối mới chứa những đỉnh chung vào giữa chúng. Sau đó ta kết hợp đồng thời thao tác loại bỏ dần những đỉnh riêng của hai khối ban đầu rồi tạo ra khối mới với những đỉnh còn lại sau khi loại bỏ. Quá trình này sẽ được thực hiện ở cả hai phía, từ khối cha loại bỏ dần đỉnh cho đến khi chỉ còn những đỉnh chung. Ta cũng sẽ làm thế với khối con. Quá trình này có thể được mô tả bằng thuật **SingleChildHandle**.
- Đối với những khối có 2 khối con trở lên, ta sẽ tạo một cây nhị phân có gốc là khối cha và khối con là những khối có chứa đỉnh giống hệt khối cha. Quá trình tạo cây nhị phân sẽ dừng lại cho đến khi số lượng nút lá trong cây nhị phân này đủ cho mỗi khối con ban đầu. Quá trình này sẽ được mô tả trong thuật toán **MultiChildHandle**.

2.8.1 Khối chứa 1 khối con

Algorithm 4 SingleChildHandle ($t_i \in T$)

if $|childs(t_i)| = 1$ **then**
  $t_j \leftarrow childs(t_i)$;
  $\phi \leftarrow X_i \cap X_j$;
  $t_{common} \leftarrow newBag(\phi, t_j)$;
  $childs(t_i) \leftarrow t_{common}$;
  $X'_i \leftarrow X_i$;
  $t_{prev} \leftarrow t_i$;
  while $|X'_i \setminus \phi| \neq 0$ **do**
    $\phi' \leftarrow \{X'_i \setminus \{x\} | x \in X'_i \setminus \phi\}$;
    $t'_{up} \leftarrow newBag(\phi', t_{common})$;
    $childs(t_{prev}) \leftarrow t'_{up}$;
    $t_{prev} \leftarrow t'_{up}$;
    $X'_i \leftarrow X'_i \setminus \{x\}$;
  end while
  $X'_j \leftarrow X_j$;
  $t_{prev} \leftarrow t_j$;
  while $|X'_j \setminus \phi| \neq 0$ **do**
    $\phi' \leftarrow \{X'_j \setminus \{x\} | x \in X'_j \setminus \phi\}$;
    $t'_{down} \leftarrow newBag(\phi', t_{prev})$;
    $childs(t_{common}) \leftarrow t'_{down}$;
    $t_{prev} \leftarrow t'_{down}$;
    $X'_j \leftarrow X'_j \setminus \{x\}$;
  end while
else if $|childs(t_i)| > 1$ **then**
  $MultiChildHandle(t_i)$;
end if

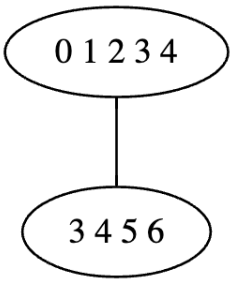
Thuật toán SingleChildHandle hoạt động như sau:

- Thuật toán chỉ xét đến những khối có duy nhất 1 khối con, trong trường hợp nhiều hơn 1 thuật toán sẽ gọi MultiChildHandle để xử lí. Chúng ta sẽ không xem xét những khối là khối lá.
- Ta xác định những đỉnh chung có trong cả 2 khối cha t_i và khối con t_j, rồi tạo một khối chung t_{common}. Khối chung t_{common} sẽ thay t_i làm khối cha của t_j, và thay t_j làm khối con của t_i.
- Ta xem xét từ khối chung t_{common} lên đến khối cha t_i trước tiên. Bằng một thứ tự đã được xác định sẵn, ta loại bỏ dần những đỉnh không thuộc những đỉnh chung có trong t_i. Với mỗi lần loại bỏ một đỉnh, ta tạo thêm khối mới t'_{up} chưa

những đỉnh chưa bị loại bỏ. Ta lần lượt bổ sung khối mới này vào vị trí phía trên khối chung cho đến khi không còn đỉnh nào không thuộc tập đỉnh chung ϕ trong t_i nữa.

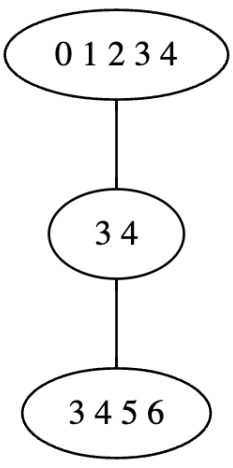
- Ta thực hiện tương tự bước đó với nửa dưới tính từ khối chung t_{common} xuống khối t_j. Khối mới t'_{down} được tạo sẽ thay nhau làm khối con của khối chung cho đến khi không còn đỉnh nào không thuộc tập đỉnh chung ϕ trong t_j nữa.

Quá trình thêm khối trong trường hợp khối cha có một khối con được minh họa trong hình dưới đây:



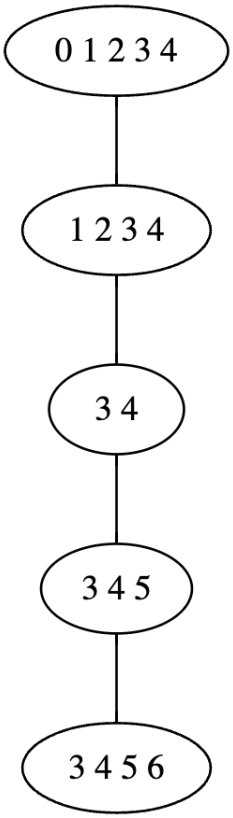
Hình 2.36: Phân rã cây ban đầu

Ở hình 2.36, ta có thể thấy 2 khối có tập đỉnh chung $\phi = \{4, 5, 6\}$, vì vậy ở bước đầu tiên, ta tạo một khối mới có chứa những đỉnh chung này rồi chèn vào hai khối ban đầu:

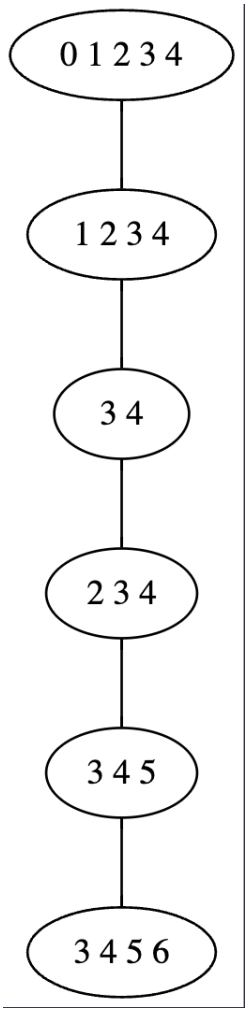


Hình 2.37: Tạo khối chứa đỉnh chung

Những đỉnh còn lại ở 2 khối lần lượt là $X_i \setminus \phi = \{0, 1, 2, 3\}$, $X_j \setminus \phi = \{7, 8, 9\}$. Ta loại bỏ dần đỉnh từ 2 tập này rồi tạo khối mới kề với khối chứa đỉnh chung:



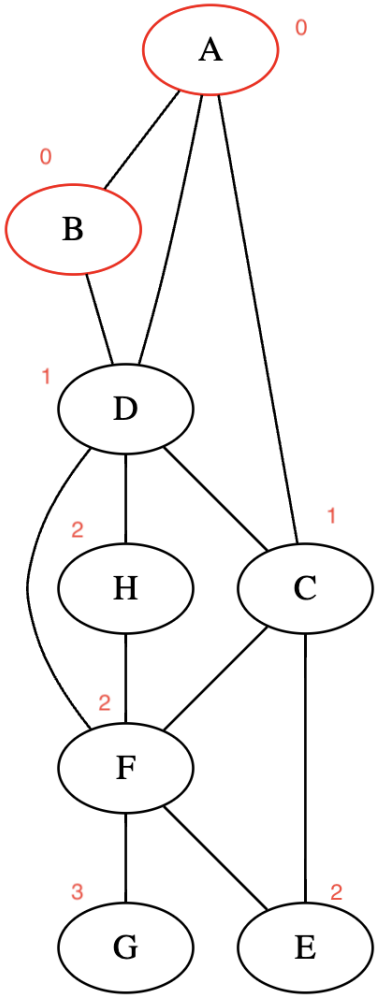
Hình 2.38: Lần loại trừ đầu tiên



Hình 2.39: Lần loại trừ tiếp theo

Trong thuật toán **SingleChildHandle** trên, ta có thể nhận thấy thao tác loại bỏ dần các đỉnh không thuộc tập đỉnh chung để tạo khối mới. Do đồ thị con được tạo bởi các đỉnh trong một khối phải là một đồ thị liên thông, nên việc lựa chọn đỉnh để loại bỏ cần được chú ý để đảm bảo tính liên thông trong một khối. Để loại bỏ các đỉnh mà không gây ảnh hưởng tới sự liên thông của đồ thị, ta cần xác định được đỉnh có liên kết chặt chẽ với các đỉnh khác và đỉnh nào không. Điều này được thể hiện thông qua liên kết của đỉnh đó với đỉnh có bậc thấp hơn trong đồ thị con. Trong một khối, ta xét những đỉnh không thuộc tập đỉnh chung, những đỉnh có bậc cao hơn sẽ được ưu tiên chọn trước, trong trường hợp có nhiều hơn 1 đỉnh cùng bậc, việc lựa chọn sẽ dựa vào số liên kết của chúng với đỉnh bậc thấp hơn.

Ví dụ 2.8.1. *Giả sử một đồ thị con tạo nên bởi các đỉnh trong một khối được biểu diễn ở hình bên dưới, với những đỉnh chung được tô màu đỏ và có bậc là 0:*



Hình 2.40: Đồ thị con và bậc của các đỉnh

Từ hình 2.40 trên, ta có thể xác định được thứ tự loại bỏ đỉnh là (G, E, F, H, C, D).

2.8.2 Khối chứa nhiều hơn 1 khối con

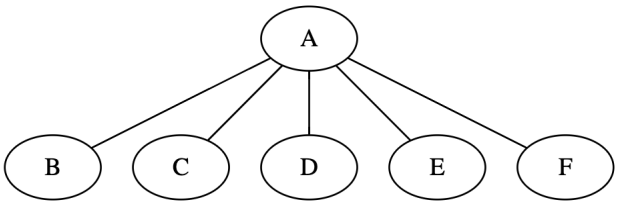
Algorithm 5 MultiChildHandle ($t_i \in T$)

if t_i has more than 1 child bag **then**
    $c \leftarrow childs(t_i)$;
    $c_j \leftarrow [c_0, c_1, ..., c_{|c|/2}]$;
    $c_k \leftarrow [c_{|c|/2+1}, c_{|c|/2+2}, ..., c_{|c|-1}]$;
    $t_j \leftarrow newBag(t_{x_i}, c_j)$;
    $t_k \leftarrow newBag(t_{x_i}, c_k)$;
    $childs(t_i) \leftarrow [t_j, t_k]$;
    $MutltiChildHandle(t_j)$;
    $MutltiChildHandle(t_k)$;
else
    $SingleChildHandle(t_i)$;
end if

Thuật toán MultiChildHandle hoạt động như sau:

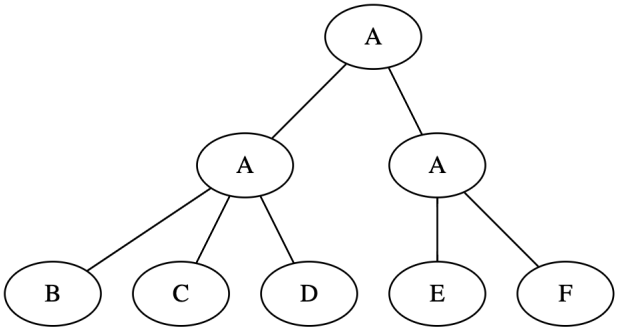
- Thuật toán chỉ thực thi với những khối có nhiều hơn 1 khối con, trong trường hợp ít hơn sẽ gọi thuật toán SingleChildHandle để xử lí.
- Tại khối t_i, ta tạo thêm 2 khối mới t_j và t_k có chứa các đỉnh bên trong giống với khối t_i. Các khối con của t_i được phân chia vào 2 khối mới tạo này.
- 2 khối mới t_j và t_k thay thế các khối ban đầu trở thành khối con của t_i.
- Ta tiếp tục xem xét với 2 khối t_j và t_k.

Quá trình thêm khối trong trường hợp khối có nhiều khối con được mình họa trong hình dưới đây:



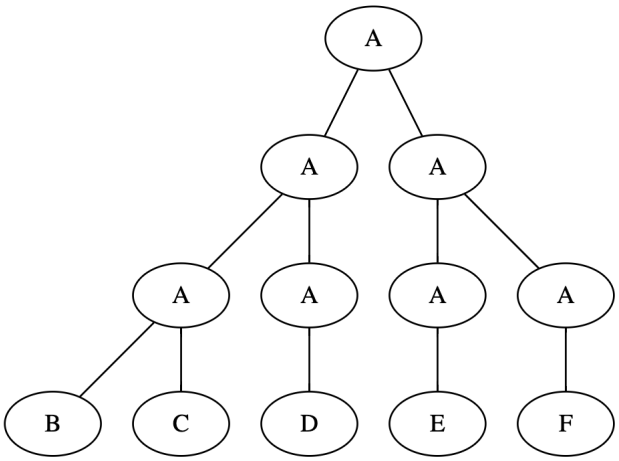
Hình 2.41: Phân rã cây ban đầu

Ta thêm 2 khối con A để tạo cây nhị phân với gốc A, các khối con ban đầu sẽ được chia đều cho 2 khối mới được thêm vào.

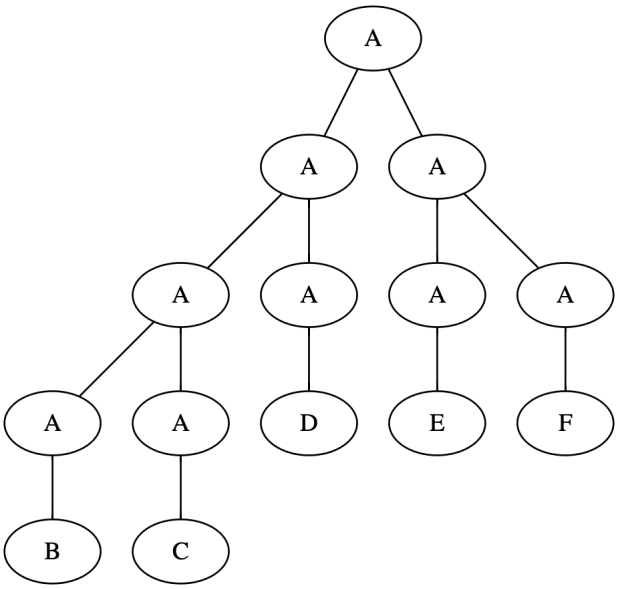


Hình 2.42: Tạo khối con lần 1

Ta liên tục lặp lại bước mở rộng cây nhị phân, đối với khối đã thỏa mãn điều kiện, ta sẽ không chạy thuật toán trên khối đó nữa mà tiếp tục ở khối khác.



Hình 2.43: Tạo khối con lần 2



Hình 2.44: Tạo khối con lần 3

Cuối cùng ta thu được một phân rã cây mới thỏa mãn điều kiện của một phân rã cây tốt đối với khối chứa nhiều hơn 2 khối con trong hình 2.44

2.9 Cài đặt và thử nghiệm

Trong phần này, ta sẽ đi cài đặt thuật toán tìm phân rã cây và thử nghiệm thuật toán trên một số đồ thị có sẵn. Chương trình được viết bằng ngôn ngữ C++ bao gồm một số lớp như sau:

- Lớp Graph: chứa thông tin của đồ thị, thứ tự loại trừ, chỉ số fill-in của từng đỉnh.
- Lớp dTree: chứa thông tin của phân rã cây, bao gồm cái khối, độ rộng cây.

- Lớp dNode: chứa thông tin của từng khối trong phân rã cây, bao gồm thông tin về các đỉnh, các khối con.

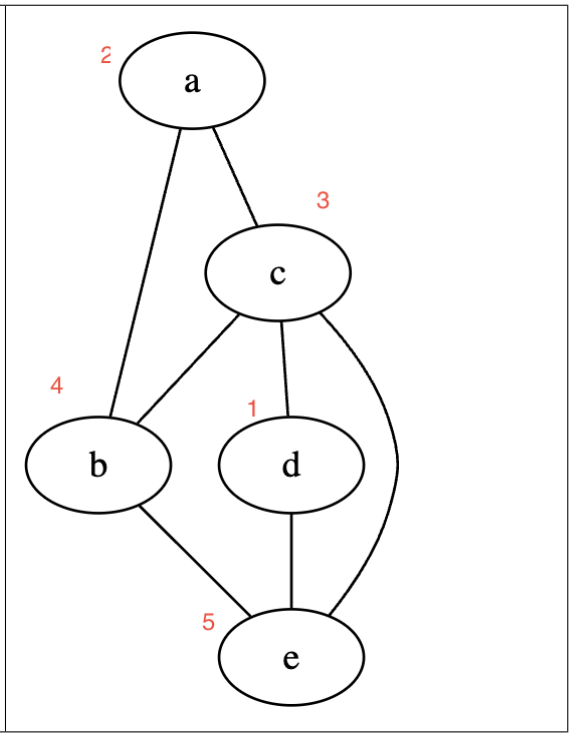
Mã nguồn của chương trình được lưu trong link sau đây: github

Dữ liệu thử nghiệm được lấy tại Dataset. Dữ liệu được viết tuân theo chuẩn DIMACS, một trung tâm nghiên cứu toán học rời rạc và lý thuyết khoa học máy tính tại Mỹ đề xướng. Được thành lập vào năm 1989, đây là nơi cung cấp dữ liệu thử nghiệm cho rất nhiều bài toán khó trong lĩnh vực khoa học máy tính, điển hình như Tập độc lập, Tô màu,....

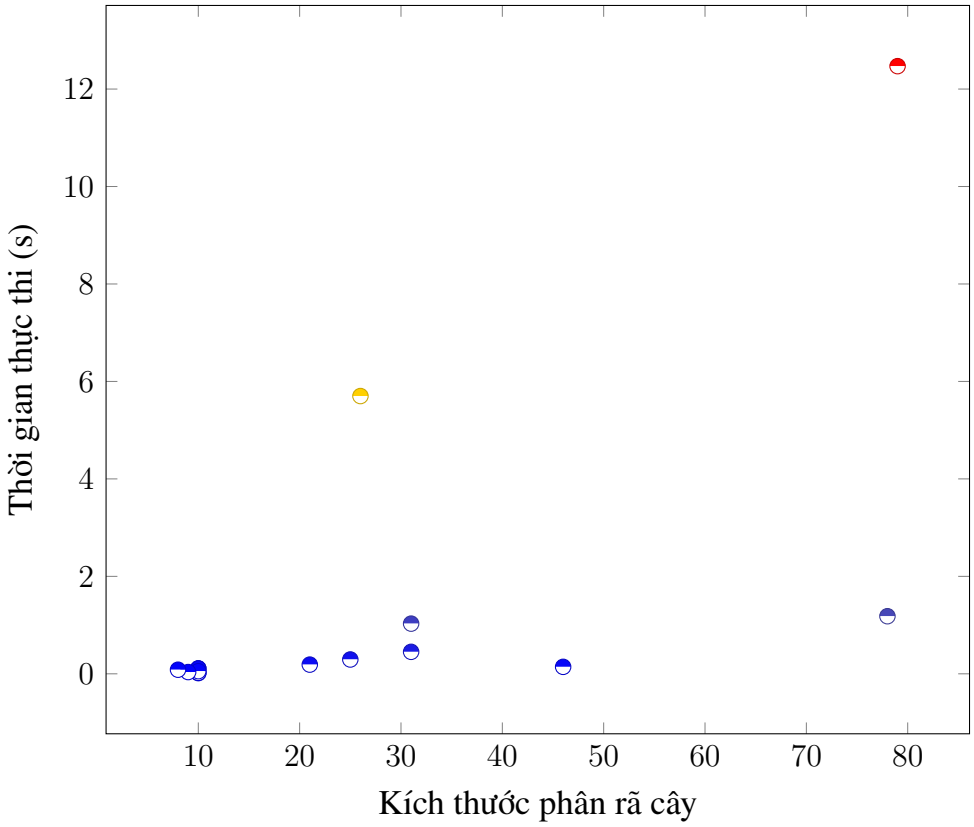
Định dạng của DIMACS như sau:

- Bắt đầu với ký tự c mô tả chú thích "c <chú thích>".
- Dòng bắt đầu với ký tự p mô tả thông tin đồ thị "p edge <số đỉnh> <số cạnh>".
- Mỗi cạnh của đồ thị mô tả dưới dạng "e <đỉnh đầu> <đỉnh cuối>".
- Trong trường hợp đồ thị có trọng số, trọng số của đồ thị được mô tả dưới dạng "n <tên đỉnh> <trọng số>".

Hình dưới đây mô tả một đồ thị được viết dưới chuẩn của DIMACS.

c comment p edge 5 7 e a b e a c a b c e b e e c d e c e e d e n a 2 n b 4 n c 3 n d 1 n e 5	

Tiến hành chạy thử trên một số loại đồ thị có trong tập dữ liệu:



Hình 2.45: Mối quan hệ giữa kích thước phân rã cây với thời gian thực thi

Tên đồ thị	Số đỉnh	Số cạnh	tree-width	Thời gian (s)
anna	138	193	10	0.112
anna-pp	22	148	10	0.015
david-pp	29	191	10	0.02
games120	120	1276	10	0.109
huck	74	301	10	0.054
jean	77	254	9	0.036
miles-250	125	387	8	0.084
miles-500	128	1170	21	0.189
miles-1500	128	5198	78	1.182
le450-15b	450	8170	26	5.7
mulsol.i.5-pp	119	2556	31	0.451
queen8-12	96	2736	25	0.294
school1-nsh-pp	327	14512	79	12.47
zeroin.i.1-pp	54	1267	46	0.143
zeroin.i.3	157	3540	31	1.03

Bảng 2.2: Thời gian chạy của chương trình với một số đồ thị

Từ bảng 2.2 và biểu đồ 2.45 trên, ta có thể thấy được mối liên hệ giữa kích thước của đồ thị với kích thước của phân rã cây và thời gian cần thiết để chương

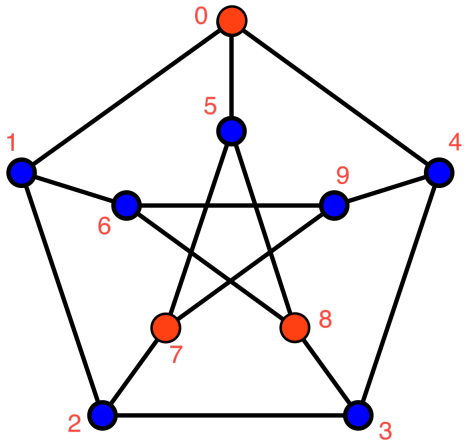
trình xây dựng được phân rã cây đó. Kích thước của đồ thị càng lớn, càng phức tạp, phân rã cây tạo ra có độ rộng càng lớn, kéo theo thời gian thực thi lớn.

CHƯƠNG 3. TẬP THỐNG TRỊ

3.1 Định nghĩa

Trong đồ thị $G = (V, E)$, một tập những đỉnh $S \subseteq V$ là một tập thống trị nếu mọi đỉnh $v \in V$ là một đỉnh trong tập S hoặc là hàng xóm với ít nhất một đỉnh thuộc tập S. Một tập thống trị là nhỏ nhất nếu mà chỉ nếu không tồn tại một tập thống trị $S' \subseteq S$. Chỉ số **dominating number** $\gamma(G)$ dùng để chỉ kích thước của S trong G.

Ví dụ 3.1.1. *Xét đồ thị Petersen như trong hình dưới đây*



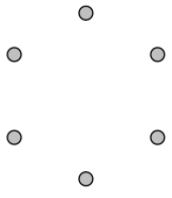
Hình 3.1: Đồ thị Petersen

Ta có thể thấy, tập thống trị tối thiểu của đồ thị trên có kích thước bằng 3. Mỗi đỉnh của đồ thị thống trị chính nó cùng với 3 đỉnh khác, do đó cần ít nhất 3 đỉnh để tạo thành một tập thống trị.

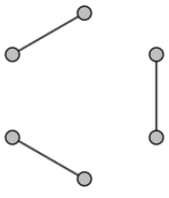
Cho đồ thị $G = (V, E)$ và tập thống trị S, $S \subseteq V$. Đỉnh $v \in S$ bị *bao vây* trong S khi $N_G[v] \subseteq S$. Với $S \subseteq V$, đỉnh $v \in S$ bị cô lập trong S nếu $N_G[v] \subseteq V - S$.

Định lý 3.1.1. *Một đỉnh bậc k thống trị chính nó và k đỉnh khác. Do đó tập thống trị trong một đồ thị k-regular G có kích thước nhỏ nhất là* $n(G)/(k+1)$.

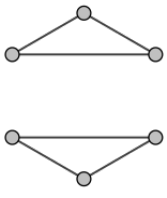
Ví dụ 3.1.2. *Một ví dụ của đồ thị k-regular với k lần lượt bằng 0, 1, 2, 3*



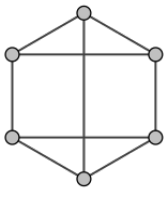
Hình 3.2: 0-regular



Hình 3.3: 1-regular



Hình 3.4: 2-regular



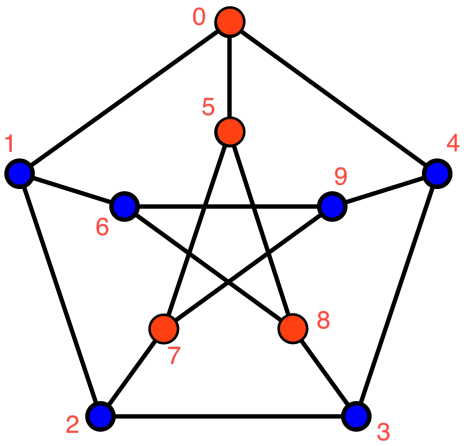
Hình 3.5: 3-regular

Có thể thấy kích thước nhỏ nhất tương ứng với từng đồ thị k-regular trên là $6/(0+1) = 6$, $6/(1+1) = 3$, $6/(2+1) = 2$ *và* $6/(3+1) = [1.5] = 2$

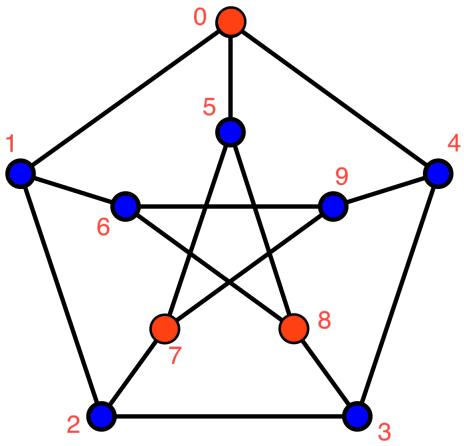
Định lý 3.1.2. *Tập các đỉnh của đồ thị G chính là một tập thống trị đó, do đó* $\gamma(G) \leqslant n(G)$.

Định lý 3.1.3. *Với mọi đồ thị n-đỉnh với bậc nhỏ nhất là k có tập thống trị tối thiểu bằng* $n\frac{1+ln(k+1)}{k+1}.\gamma(G) \leqslant n(G)$.

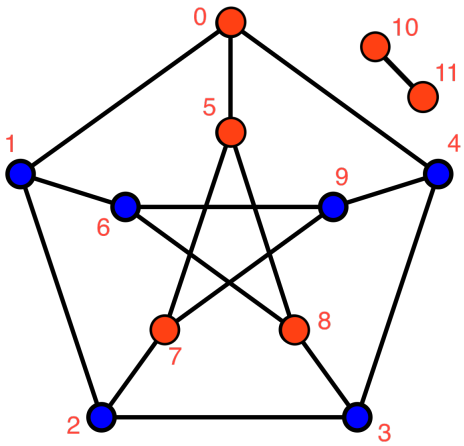
Định lý 3.1.4. *Một tập thống trị S của đồ thị G là tập thống trị liên thông (connected dominating set) nếu* $G[S]$ *liên thông, tập thống trị độc lập (independent dominating set) nếu* $G[S]$ *độc lập và là tập thống trị hoàn toàn (total dominating set) nếu* $G[S]$ *không tồn tại đỉnh bị cô lập.*



Hình 3.6: Connected Dominating Set



Hình 3.7: Independent Dominating Set



Hình 3.8: Total Dominating Set

3.2 Phỏng đoán Vizing

Năm 1968, nhà toán học người Liên Xô Vadim G.Vizing (xem [2]) đã đưa ra một phỏng đoán, phỏng đoán của ông nói về sự liên quan giữa số thống trị với tích

Đề cát của đồ thị. Phỏng đoán của Vizing như sau: nếu $\gamma(G)$ đại diện cho số lượng đỉnh nhỏ nhất của một tập thống trị của đồ thị G, thì $\gamma(G\square H) \geqslant \gamma(G)\gamma(H)$.

Phỏng đoán của Vizing có thể được coi là bài toán lớn nhất mà vẫn chưa được chứng minh khi nói đến sự thống trị trên đồ thị. Từ khi được đề xuất bởi Vizing, đã có rất nhiều những nghiên cứu được thực hiện nhằm chứng minh tính đúng đắn của phỏng đoán trên một số đồ thị thỏa mãn những tính chất nhất định. Do phỏng đoán vẫn chưa được chứng minh một cách tổng quát, những nhà nghiên cứu đã hướng đến những bài toán tương tự cho những dạng tích đồ thị (graph product) khác hay dạng thống trị khác. Một trong số những bài toán tương tự này vẫn còn là phỏng đoán, một số khác đã được chứng minh. Trong phần này, ta sẽ đi phân tích những dạng đồ thị thỏa mãn phỏng đoán của Vizing.

Ta nói một đồ thị G thỏa mãn phỏng đoán của Vizing nếu với mọi đồ thị H, phỏng đoán được giả định là đúng. Đã có nhiều nghiên cứu được thực hiện cho thấy phỏng đoán của Vizing đúng với những đồ thị thỏa mãn những tính chất nhất định. Chứng minh của hai nhà nghiên cứu Barcalkin và German cho thấy phỏng đoán của Vizing đối với những đồ thị có thể phân rã đúng với những đồ thị có $\gamma(G) \leqslant 2$. Vào năm 2004, nghiên cứu của Liang Sun cho thấy phỏng đoán đúng với mọi đồ thị có $\gamma(G) \leqslant 3$.

Mệnh đề 3.2.1. *Nếu G thỏa mãn phỏng đoán Vizing và K là một đồ thị spanning từ G (spanning subgraph) sao cho $\gamma(G) = \gamma(K)$ thì K thỏa mãn phỏng đoán của Vizing.*

Định lý 3.2.1. *Cho đồ thị G và $x \in V(G)$ sao cho $\gamma(G - x) < \gamma(G)$. Nếu G thỏa mãn phỏng đoán của Vizing thì G - x cũng thỏa mãn.*

Mệnh đề 3.2.2. *Với mọi đồ thị G và H, $\gamma(G\square H) \geqslant min\{|V(G)|, |V(H)|\}$.*

Kết quả dưới đây cung cấp cho chúng ta cận dưới của $\gamma(G\square H)$, được chứng minh bởi Jacobson và Kinch.

Định lý 3.2.2. *Với mọi đồ thị G và H, $\gamma(G\square H) \geqslant \frac{|H|}{\Delta(H)+1}\gamma(G)$.*

Định lý 3.2.3. *Với mọi đồ thị G và H, ta có $\gamma(G\square H) \leqslant min\{\gamma(G)|V(H)|, |V(G)|\gamma(H)\}$.*

Định lý 3.2.4. *Với G là một đồ thị có thể phân rã (decomposable graph) và K là một spanning của G với $\gamma(G) = \gamma(K)$. K thỏa mãn phỏng đoán của Vizing.*

Phỏng đoán của Vizing vẫn đang là một câu hỏi mở chưa được chứng minh tính đúng đắn, mặc dù đã có nhiều công trình nghiên cứu bởi những nhà toán học trên khắp thế giới. Tuy nhiên nó vẫn cho chúng ta một góc nhìn khác thông qua sự liên hệ giữa số thống trị với tích Đề các của 2 đồ thị. Mặc dù vẫn còn rất nhiều kết quả

được chứng minh, nhưng trong phạm vi đồ án này, vấn đề này sẽ không được nhắc tới.

3.3 Tìm tập thống trị tối thiểu thông qua phương pháp quy hoạch động

3.3.1 Giới thiệu

Bài toán đi tìm tập thống trị của đồ thị đã được chứng minh là một bài toán thuộc lớp NP-complete. Do đó sẽ không tồn tại một phương pháp đủ hiệu quả để đi tìm tập thống trị tối thiểu cho mọi đồ thị. Tuy nhiên, đối với một số dạng đồ thị nhất định, vẫn có một số thuật toán có thể đưa ra kết quả mà ta có thể chấp nhận được. Bằng việc sử dụng một cấu trúc gọi là phân rã cây như đã đề cập ở phần trước, ta có thể thực hiện phương pháp quy hoạch động trên một phân rã cây có độ rộng k để đi tìm tập thống trị tối thiểu.

Tóm lại chúng ta sẽ cần thực hiện 2 công việc để giải quyết bài toán tìm tập thống trị tối thiểu:

1. Xây dựng phân rã cây từ đồ thị ban đầu.
2. Sử dụng quy hoạch động trên phân rã cây mới tìm được đi tìm tập thống trị tối thiểu.

Trong quá khứ, đã có những nghiên cứu được thực hiện bằng phương pháp quy hoạch động trên phân rã cây, kết quả tốt nhất được hai nhà nghiên cứu Telle và Proskurowski vào năm 1997 với thời gian chạy là $O(9^k n)$ cho phân rã cây có độ rộng k và n khối. Trước đó 10 năm, vào năm 1987, Corneil và Keil giới thiệu một thuật toán có thời gian chạy $O(4^k n^k + 2)$ cho cây có độ rộng k.

Cho tới năm 2002, trong một nghiên cứu có tên là **Improved Tree Decomposition Based Algorithm for Domination-like Problems** của Jochen Alber và Rolf Niedermierer tại Đức, đã giới thiệu một hướng tiếp cận mới có thời gian chạy $O(4^k)$ thông qua một khái niệm gọi là tính đơn điệu (monotonicity). Bằng việc sử dụng tính chất này, thuật toán mới của hai nhà nghiên cứu đã cải thiện hầu hết thời gian chạy của thuật toán cũ do Telle và Proskurowski giới thiệu. Kết quả so sánh thời gian chạy được mô tả bên dưới với một số giá trị của k và $n = 1000$. Giả định máy tính có thể xử lí 10^9 phép tính trong một giây.

	$k = 5$	$k = 10$	$k = 15$	$k = 20$
$9^k n$	0.05 giây	1 giờ	6.5 năm	$3.9 \cdot 10^5$ năm
$4^k n$	0.001 giây	1 giây	18 phút	13 ngày

Bảng 3.1: Bảng so sánh thời gian chạy thuật toán của Telle và Proskurowski với Jochen Alber và Rolf Nieredermierer. Giả định n = 1000 và máy tính xử lí được 10^9 phép tính một giây

3.3.2 Quy hoạch động dựa trên tính đơn điệu

Trong phần này, chúng ta sẽ thảo luận cơ sở lý thuyết của thuật toán.

Định lý 3.3.1. *Đối với một phân rã cây có độ rộng k, một tập thống trị cực tiểu có thể xác định được trong thời gian $O(4^k)$, với n là số lượng khối trong phân rã cây.*

Từ bây giờ ta sẽ giả định một phân rã cây của đồ thị $G = (V, E)$ là $X = \langle\{X_i | i \in I\}, T\rangle$. Ta có thể coi X ở đây là một phân rã cây tốt (*nice decomposition*).

Giả sử rằng $V = \{x_1, ..., x_n\}$. Ta giả sử thứ tự các đỉnh trong một khối như sau $X_i = \{x_{i_1}, ..., x_{i_{n_i}}\}$ với $i_1 \leqslant ... \leqslant i_{n_i}$ cho mọi $i \in I$.

Tiếp theo, ta sử dụng 3 màu khác nhau để gán cho từng đỉnh:

- **đen** (biểu diễn bởi giá trị 1, có nghĩa là đỉnh đó thuộc tập thống trị)
- **trắng** (biểu diễn bởi giá trị 0, có nghĩa là đỉnh đó đã bị thống trị ở bước hiện tại của thuật toán)
- **xám** (biểu diễn bởi giá trị $\hat{0}$, có nghĩa là đỉnh đó vẫn chưa thể xác định được thuộc hay không thuộc tập thống trị tại bước hiện tại của thuật toán)

Một vector $c = (c_1, ..., c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ được gọi là dãy màu của khối $X_i = \{x_{i_1}, ..., x_{i_{n_i}}\}$ trong phân rã cây. Trong đó giá trị màu được gán cho x_{i_t} đỉnh thuộc khối X_i thông qua vector c tương ứng với phần tử thứ t của c c_t.

Với mỗi khối X_i với $|X_i| = n_i$, ta sử dụng ánh xạ:

$$A_i : \{0, \hat{0}, 1\}^{n_i} \longrightarrow \mathbb{N} \cup \{+\infty\}. \tag{3.1}$$

Với một giá trị dãy màu $c = \{c_1, ..., c_{n_i}\} \in \{0, \hat{0}, 1\}^{n_i}$, giá trị $A_i(c)$ cho thấy có bao nhiêu đỉnh cần thiết để tạo thành một tập thống trị cực tiểu xét cho đến bước hiện tại của thuật toán.

Một dãy màu $c = \{c_1, ..., c_{n_i}\} \in \{0, \hat{0}, 1\}^{n_i}$ được coi là không hợp lệ trong khối X_i nếu:

$$(\exists s \in \{1, ..., n_i\} : c_s = 0) \wedge (\nexists t \in \{1, ..., n_i\} : (x_{i_t} \in N(x_{i_s}) \wedge c_t = 1)). \tag{3.2}$$

Nói một cách dễ hiểu, một dãy màu không hợp lệ trong một khối là trường hợp khi tồn tại một đỉnh x_{i_s} trong khối được gán màu trắng (đỉnh đang bị thống trị), tuy nhiên những đỉnh hàng xóm của x_{i_s} không tồn tại đỉnh nào được gán màu đen (đỉnh thống trị) theo như vector c.

Với một dãy màu $c = (c_1, ..., c_m) \in \{0, \hat{0}, 1\}^m$ và một giá trị màu $d \in \{0, \hat{0}, 1\}$, ta có

$$\#_d = |\{t \in \{1, ..., m\} : c_t = d\}|. \tag{3.3}$$

Trong tập màu $\{0, \hat{0}, 1\}$, $\prec$ là một sắp thứ tự một phần được quy định bởi $\hat{0} \prec 0$ và $d \prec d$ cho mọi $d \in \{0, \hat{0}, 1\}$. Sắp thứ tự này mở rộng cho tập các dãy màu: với $c = (c_1, ..., c_m)$, $c' = (c'_1, ..., c'_m) \in \{0, \hat{0}, 1\}$, ta nói $c \prec c'$ nếu và chỉ nếu $c_t \prec c'_t$ với mọi $t = 1, .., m$.

Ta nói một ánh xạ

$$A_i : \{0, \hat{0}, 1\}^{n_i} \longrightarrow \mathbb{N} \cup \{+\infty\} \tag{3.4}$$

là đơn điệu từ tập sắp thứ tự một phần $(\{0, \hat{0}, 1\}^{n_i}, \prec)$ đến $(\mathbb{N} \cup \{+\infty\})$ nếu với $c, c' \in \{0, \hat{0}, 1\}^{n_i}$, $c \prec c$" thì $A(c) \leqslant A(c')$.

3.3.3 Chi tiết thuật toán

Phần này sẽ trình bày chi tiết thuật toán sử dụng quy hoạch động sử dụng ánh xạ đã nói tới ở trên.

Bước 1: Ở bước đầu tiên của thuật toán, tại mỗi khối lá i của phân rã cây, ta khởi tạo ánh xạ A_i:

Với mọi c $\in \{0, 0, 1\}^{n_i}$:

$$A_i(c) \longleftarrow \begin{cases} +\infty & \text{nếu c không thỏa mãn trong khối } X_i \\ \#_1(c) & \text{trong trường hợp khác} \end{cases} \tag{3.5}$$

Thông qua bước khởi tạo này, ta có thể đảm bảo được việc chỉ những dãy màu thỏa mãn trong khối X_i mới được xét đến.

Mệnh đề 3.3.1. *1. Công thức 3.5 có thời gian thực thi là $O(3^{n_i} n_i)$.*

2. Ánh xạ A_i là đơn điệu.

Bước 2: Sau khi kết thúc quá trình khởi tạo, thuật toán sẽ đi dần từ những khối lá lên đến khối gốc, công thức đánh giá sẽ dựa vào loại khối mà thuật toán đang xét đến:

FORGET NODES: Giả sử i là một FORGET NODE với khối con j và $X_i = (x_{i_1}, ..., x_{i_{n_i}}), X_j = (x_{j_1}, ..., x_{j_{n_j}})$. Công thức tính $A_i(c)$ như sau:

Với mọi $c \in \{0, \hat{0}, 1\}^{n_i}$:

$$A_i(c) \longleftarrow \min_{d \in \{0,1\}} A_j(c \times \{d\}) \tag{3.6}$$

Một dãy màu $c \times \{\hat{0}\}$ trong khối X_j có nghĩa là đỉnh x được gán giá trị $\hat{0}$, tức là chưa được xác định có thuộc hay không thuộc tập thống trị. Từ điều kiện số 3 trong định nghĩa 2.2, với mọi $i, j, k \in I$, nếu j nằm trên đường đi từ i đến k thì $X_i \cap X_k \subseteq X_j$, đỉnh x sẽ không còn xuất hiện trong những khối khác ở giai đoạn sau của thuật toán nữa, do đó dãy màu $c \times \{\hat{0}\}$ sẽ không được xử lí và sẽ không thể hình thành tập thống trị. Do đó công thức 3.6 chỉ xét những trường hợp đỉnh x đã xác định được trạng thái của nó.

Mệnh đề 3.3.2. *1. Công thức 3.6 có thời gian thực thi là $O(3^{n_i})$.*

2. Nếu ánh xạ A_j là đơn điệu thì A_i cũng đơn điệu.

INTRODUCE NODES: Giả sử rằng i là một INTRODUCE NODE với khối con là j và $X_j = (x_{j_1}, ..., x_{j_{n_j}}), X_i = (x_{j_1}, ..., x_{j_{n_j}}, x)$. $N(x) \cap X_j = \{x_{j_{p_1}}, ..., x_{j_{p_s}}\}$ là những đỉnh hàng xóm với đỉnh x có trong khối X_i. Ta có hàm $\phi : \{0, \hat{0}, 1\}^{n_j} \longrightarrow \{0, \hat{0}, 1\}^{n_j}$ trên tập các dãy màu của khối X_j. Với $c = (c_1, ..., c_{n_j}) \in \{0, \hat{0}, 1\}$, $\phi(c) = (c'_1, ..., c'_{n_j})$ sao cho:

$$c'_t = \begin{cases} \hat{0} & \text{nếu } t \in \{p_1, ..., p_s\} \text{ và } c_t = 0, \\ c_t & \text{trong những trường hợp khác} \end{cases} \tag{3.7}$$

Từ đó, công thức cho ánh xạ A_i đối với khối X_i như sau:

Với mọi $c = (c_1, ..., c_{n_j}) \in \{0, \hat{0}, 1\}^{n_j}$:

$$A_j(c \times \{0\}) \leftarrow \begin{cases} A_j(c) & \text{nếu tồn tại một đỉnh hàng xóm } x_{j_q} \text{ trong } X_i \text{ với } c_q = 1, \\ +\infty & \text{trong những trường hợp khác} \end{cases} \tag{3.8}$$

$$A_i(c \times \{1\}) \leftarrow A_j(\phi(c)) + 1 \tag{3.9}$$

$$A_i(c \times \{\hat{0}\}) \leftarrow A_j(c) \tag{3.10}$$

Ta kiểm tra tính đúng đắn của công thức 3.8 và 3.9 như sau:

Đối với 3.8, nếu chúng ta gán giá trị 0 cho đỉnh x, tức là coi đỉnh x là một đỉnh bi thống trị tại giai đoạn hiện tại của thuật toán, ta phải một lần nữa kiểm tra sự thỏa mãn của dãy màu giống như ở bước khởi tạo. Tuy nhiên ở bước này, ta thêm một điều kiện mới là phải tồn tại ít nhất một đỉnh là hàng xóm của x mang giá trị là 1, tức là đỉnh thuộc tập thống trị.

Trong trường hợp chúng ta gán giá trị 1 cho đỉnh x, điều này có nghĩa là đỉnh x thuộc tập thống trị. Những đỉnh bị thống trị bởi x trong khối này là $\{x_{j_{p_1}}, ..., x_{j_{ps}}\}$. Ta muốn tìm giá trị của $A_i(c \times \{1\})$, giả sử một số đỉnh trong số chúng mang giá trị 0 thông qua c. Do việc gán giá trị 1 cho x đã thỏa mãn điều kiện đối với những đỉnh mang giá trị 0 này, ta có thể xác định được giá trị của $A_i(c \times \{1\})$ thông qua $A_j(c')$, với $c' = \phi(c)$.

Mệnh đề 3.3.3. *1. Công thức 3.8, 3.9 và 3.10 có thời gian thực thi là $O(3^{n_i} n_i)$.*

2. Nếu ánh xạ A_j là đơn ánh thì A_i cũng là đơn ánh.

JOIN NODES: Giả sử i là một JOIN NODE với 2 khối con là j và k với $X_i = X_j = X_k = (x_1, ..., x_{i_{n_i}})$. $c = (c_1, ..., c_{n_i}) \in \{0, \hat{0}, 1\}^{n_i}$ là dãy màu của khối X_i. Ta nói rằng c bị phân chia bởi c' và c'' khi:

1. $(c_t \in \{1, \hat{0}\}) \Rightarrow c'_t = c''_t = c_t$ và
2. $(c_t = 0 \Rightarrow [(c'_t, c''_t \in \{0, \hat{0}\}) \wedge (c'_t = 0 \vee c''_t = 0)])$

Công thức tính cho $A_i(c)$ sẽ như sau:

Với mọi $c \in \{0, \hat{0}, 1\}^{n_i}$:

$$A_i(c) \leftarrow \min\{A_j(c') + A_k(c'') - \#_1(c) \mid \text{c bị phân chia bởi } c' \text{ và } c''\} \tag{3.11}$$

Nói cách khác, để có thể xác định được giá trị $A_i(c)$, ta đi tìm dãy màu tương ứng với c ở cây con trái và cây con phải của i. Ta lấy tổng của $A_j(c')$ và $A_j(c'')$ rồi trừ đi số lượng đỉnh được gán giá trị 1 (đỉnh thuộc tập thống trị) của khối i, do trong quá trình cộng $A_j(c')$ và $A_k(c'')$, số lượng đỉnh được gán giá trị 1 đã tăng gấp 2 lần.

Khi gán giá trị 1 hoặc $\hat{0}$ cho đỉnh x trong khối X_i, ta cần đảm bảo đỉnh đó trong 2 khối con của i cũng được gán giá trị tương tự thông qua c' và c''. Tuy nhiên khi gán giá trị 0 cho x, cần phải đảm bảo rằng ít nhất một trong 2 đỉnh tương ứng của 2 khối con có cùng giá trị giống x. Do A_j và A_k có tính đơn điệu, nên ta có thể thu được giá trị cực tiểu tương tự ở công thức 3.11 bằng cách thay thế điều kiện 2 của 3.3.3 như sau:

$$(c_t = 0 \Rightarrow (c'_t, c''_t \in \{0, \hat{0}\} \wedge c'_t \neq c''_t)) \tag{3.12}$$

Mệnh đề 3.3.4. *1. Công thức 3.11 có thời gian thực thi là $O(4^{n_i})$.*

2. Nếu A_i và A_k đơn điệu thì A_i đơn điệu.

Chứng minh: Thời gian chạy của được xác định thông qua:

$$\sum_{c\in\{0,\hat{0},1\}^{n_i}} |\{(c', c''), \text{c bị phân chia bởi } c' \text{ và } c''\}| \tag{3.13}$$

Với dãy màu $c \in \{0, \hat{0}, 1\}^{n_i}$, $z = \#_0(c)$, ta có 2^z cặp (c', c'') có thể phân chia c. Tồn tại $2^{n_i-z} \cdot C_{n_i}^z$ dãy màu c có $\#_0(c) = z$. Từ đó 3.13 tương đương:

$$\begin{aligned}
&\sum_{z=0}^{n} 2^{n_i-z} \cdot C_{n_i}^z \cdot 2^z \\
=&2^{n_i} \cdot \sum_{z=0}^{n} 2^{n_i-z} C_{n_i}^z \\
=&2^{n_i} \cdot 2^{n_i} \\
=&4^{n_i}
\end{aligned} \tag{3.14}$$

Bước 3: Giả sử khối gốc của phân rã cây T được ký hiệu là r. Để tìm $\gamma(G)$, ta có công thức:

$$\gamma(G) = min\{A_r(c)|c \in \{0, 1\}^{n_r}\}. \tag{3.15}$$

Ở công thức 3.15 cuối cùng, ta chỉ xét những trường hợp dãy màu chỉ bao gồm giá trị 0 và 1. Do ta sẽ không thể thu được kết quả chính xác nếu vẫn còn đỉnh chưa thể xác định được trạng thái thống trị của nó.

3.4 Mở rộng thuật toán

Thuật toán đã nêu trên có thể được mở rộng để giải quyết những bài toán liên quan đến tập thống trị, điển hình như:

3.4.1 Tập thống trị thỏa mãn tính chất cho trước

Tập thống trị với tính chất P là bài toán đi tìm tập thống trị cực tiểu D sao cho tập thống trị cực tiểu thỏa mãn tính chất P được quy định từ trước, hay nói cách khác $P(D) = true$.

Một số ví dụ của bài toán này bao gồm:

• Tập thống trị độc lập (the Independent Dominating Set): tính chất P của bài

toán này là tập thống trị cực tiểu phải là tập độc lập.

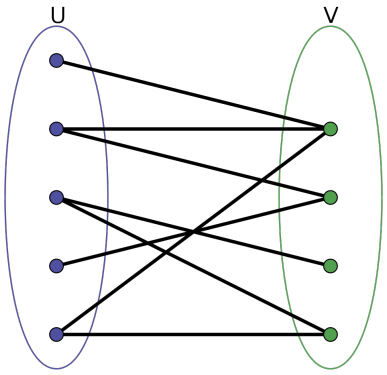
- Tập thống trị hoàn toàn (the Total Dominating Set): tính chất P của bài toán này là mỗi đỉnh trong D đều là hàng xóm với ít nhất một đỉnh thuộc D.
- Tập thống trị hoàn hảo (the Perfect Dominating Set): tính chất P của bài toán này là mỗi đỉnh không thuộc tập D chỉ có duy nhất một hàng xóm thuộc tập D.
- Tập thống trị độc lập hoàn hảo (the Perfect Independent Dominating Set): tập thống trị D sở hữu hai tính chất độc lập và hoàn hảo.
- Tập thống trị hoàn hảo toàn phần (the Total Perfect Dominating Set): tập thống trị D sở hữu hai tính chất hoàn hảo và toàn phần.

3.4.2 Tập thống trị có trọng số - Weighted Versions of Dominating Set

Với một đồ thị $G = (V, E)$ tồn tại một hàm trọng số dương: $w : V \to \mathbb{N}$. Trọng số của tập $D \subseteq V$ được định nghĩa là $w(D) = \Sigma_{v \in D} w(v)$. Bài toán Tập thống trị có trọng số đi tìm một tập thống trị trên đồ thị $G = (V, E)$ sao cho có tổng trọng số nhỏ nhất.

3.4.3 Tập thống trị đỏ-xanh - Red-Blue Dominating Set

Trong một đồ thị hai phía (bipartitle graph) $G = (V, E)$, với $V = V_{red} \cup V_{blue}$. Câu hỏi được đặt ra là làm thế nào để xác định một tập $V' \subseteq V_{red}$ có kích thước nhỏ nhất sao cho mọi đỉnh thuộc V_{blue} kề với ít nhất một đỉnh thuộc V'.



3.5 Cài đặt và thử nghiệm

3.5.1 Cài đặt

Trong phần này, tôi sẽ tiến hành cài đặt thuật toán đã được đề cập ở phần trước, rồi sau đó sẽ thử nghiệm trên một số đồ thị. Chương trình được viết bằng ngôn ngữ C. Mặc dù thuật toán xây dựng phân rã cây đã được nói đến và cài đặt thành công ở chương trước. Tuy nhiên do sự chính xác của thuật toán tìm tập thống trị tối thiểu phụ thuộc vào chất lượng của phân rã cây thu được, do đó trong phần này tôi sẽ sử dụng kết quả trong cuộc thi PACE 2016. Đây là một cuộc thi được khởi xướng từ năm 2015 với mục đích kết nối giữa việc nghiên cứu, phân tích thuật toán với việc ứng dụng những thuật toán đó vào những bài toán thực tế. Ở đây tôi sẽ sử dụng thuật toán của Hisao Tamaki, người đoạt giải nhất với việc giải được 199/200

mẫu thử. Chương trình được viết bằng ngôn ngữ C dựa trên một thuật toán được điều chỉnh từ hướng tiếp cận brute-force đã được đề cập trong bài báo có tựa đề Complexity of Finding Embeddings in a k-Tree[3].

Toàn bộ mã nguồn của chương trình được lưu trữ trong Github

Đầu vào của chương trình được quy định như sau:

```
p tw <số đỉnh> <số cạnh>
<cạnh 1>
<cạnh 2>
...
<cạnh n>
```

Trong số các cạnh được viết dưới dạng: <đỉnh đầu> <đỉnh cuối>.

Đầu ra của chương trình sẽ có dạng như sau:

```
<tree-width>
<id khối> <số đỉnh trong khối> <đỉnh 1> <đỉnh 2> ... <đỉnh n>
...
-1
<id khối> <id khối>
...
```

Thông số đầu tiên của đầu ra sẽ là tree-width của đồ thị đầu vào. Tiếp sau đó là thông tin của mỗi khối trong phân rã cây vừa được xây dựng, với mỗi khối được viết trên một dòng. Sau khi kết thúc việc định nghĩa các khối trong phân rã cây, kí tự -1 được thêm vào để đánh dấu việc bắt đầu cho phần xác định sự liên kết giữa các khối đã nói ở trên.

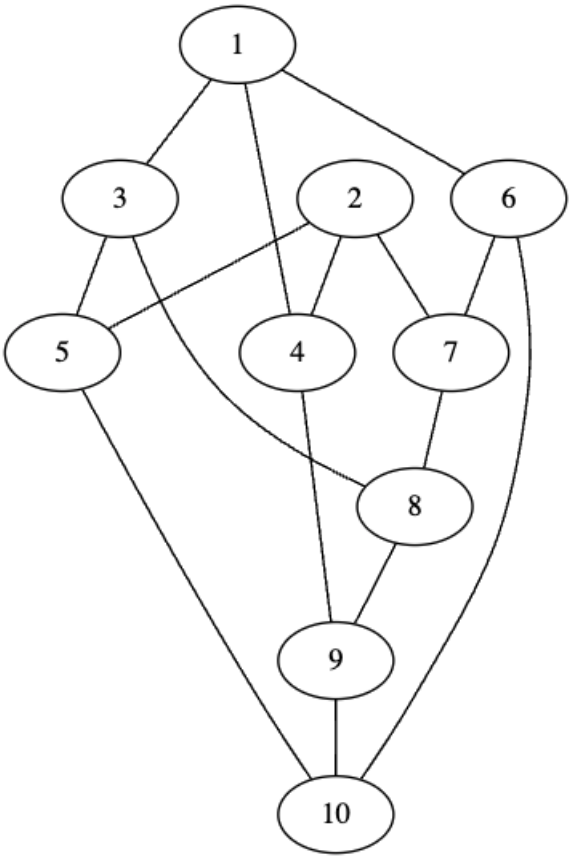
```
p tw 10 16
1 5
1 9
3 6
1 2
2 3
2 8
2 9
3 4
5 10
3 7
1 6
6 10
3 8
4 5
5 9
7 10
```

Bảng 3.2: Ví dụ đầu vào của một đồ thị có 10 đỉnh và 16 cạnh

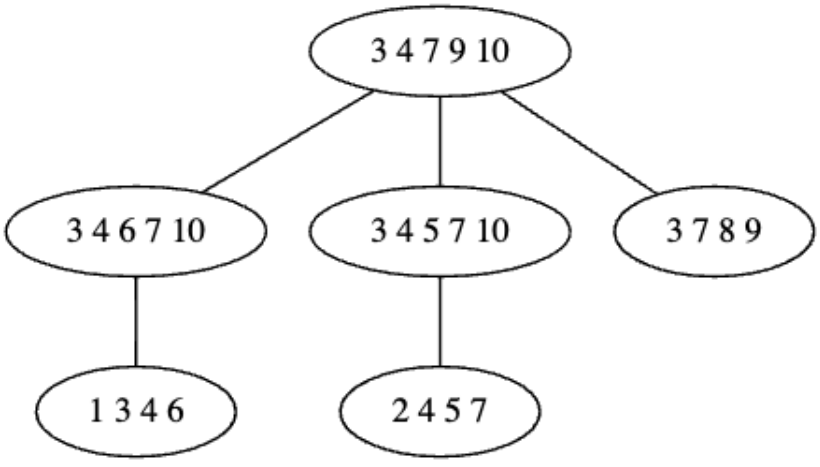
```
4
1 5 3 4 7 9 10
2 5 3 4 6 7 10
3 4 1 3 4 6
4 5 3 4 5 7 10
5 4 2 4 5 7
6 4 3 7 8 9
-1
2 3
1 2
4 5
1 4
1 6
```

Bảng 3.3: Ví dụ đầu ra là một phân rã cây có kích thước bằng 4 với 6 khối

Theo như ví dụ đầu ra trên thì từ đồ thị G bao gồm 10 đỉnh và 16 cạnh, ta có thể xây dựng một phân rã cây có 6 khối. Trong đó khối 1 bao gồm 5 đỉnh 3, 4, 7, 9, 10; khối 2 bao gồm 5 đỉnh 3, 4, 6, 7, 10;...



Hình 3.9: Đồ thị G với tập các đỉnh thống trị chưa được xác định



Hình 3.10: Phân rã cây của đồ thị G

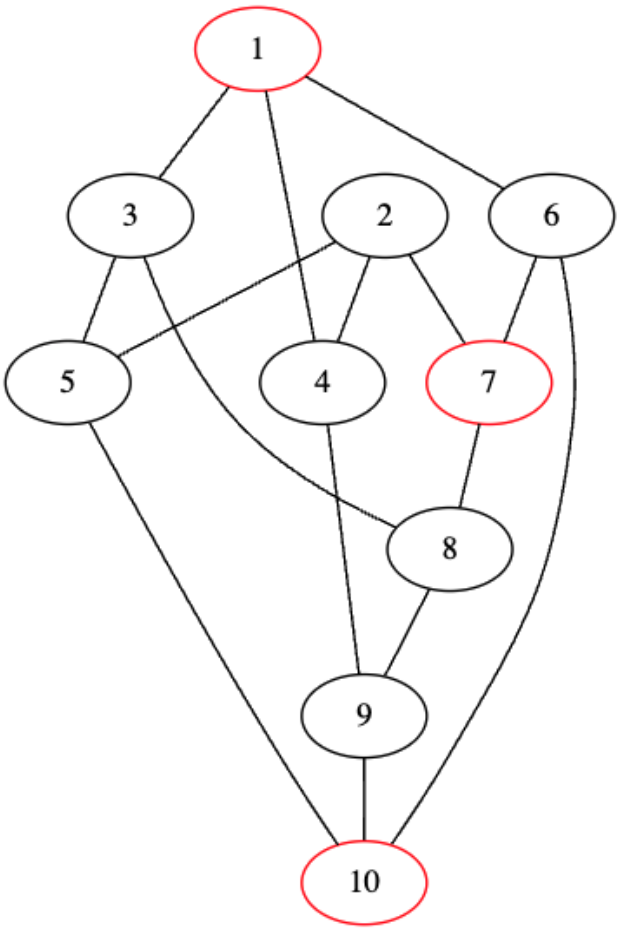
Sau khi thu được dữ liệu của phân rã cây, chúng sẽ được sử dụng để chạy thuật toán tìm tập thống trị cực tiểu đã được đề cập ở chương trước. Kết quả thu được sẽ có dạng:

<$\gamma(G)$> <đỉnh 1> <đỉnh 2> ... <đỉnh n>

3 1 7 10

Bảng 3.4: Ví dụ đầu ra của thuật toán tìm tập thống trị cực tiểu cho đồ thị G ở ví dụ trên

Một đồ thị có thể tồn tại nhiều hơn 1 tập các đỉnh thống trị, do đó mặc dù kích thước của tập thống trị có thể đúng tuy nhiên các đỉnh trong tập thống trị có thể khác nhau tùy theo cách cài đặt chương trình.



Hình 3.11: Đồ thị G với tập các đỉnh thống trị đã được xác định

3.5.2 Thử nghiệm và đánh giá

Trong phần này, chúng ta sẽ tiến hành đánh giá độ chính xác của thuật toán vừa cài đặt. Do không tìm thấy bộ dữ liệu cho kết quả chính xác về tập thống trị tối thiểu, thuật toán sẽ được đánh giá thông qua việc so sánh kết quả thu được với những thuật toán đã được nghiên cứu từ trước đến nay.

Tập dữ liệu về đồ thị được lấy từ PACE 2016, sau khi xây dựng được phân rã cây cho từng đồ thị, ta sử dụng những phân rã cây đó đưa vào các thuật toán để xác định tập thống trị tối thiểu. Thuật toán sử dụng để so sánh được tham khảo từ Github. Tác giả đã sử dụng ngôn ngữ Java để cài đặt một số thuật toán phổ biến để xác định tập thống trị tối thiểu trên đồ thị. Có tổng cộng là 9 thuật toán đã được tác giả cài đặt, trong đó có những thuật toán như Greedy, GreedyRev, TrivialSetCover, GreedyRandom, ... Tuy nhiên để tiết kiệm thời gian, ta sẽ tiến hành so sánh với 2 thuật toán Arbitrary và Greedy.

Dữ liệu của bài toán được lưu trong PACE-treewidth-testbed. Thử nghiệm được thực hiện trên máy tính Macbook 2020, chip M1, RAM 16Gb.

	Đỉnh	**Cạnh**	**Width**	$\gamma(G)$	**Thời gian**
ChvatalGraph	12	24	6	4	0.04
ClebschGraph	16	40	8	4	1.25
BlanusaSecondSnarkGraph	18	27	4	5	0.018
DesarguesGraph	20	30	6	6	0.07
BrinkmannGraph	21	42	8	5	2.05
DoubleStarSnark	30	45	6	8	0.15
DyckGraph	32	48	7	8	0.41
4x12_torusGrid	48	192	8	12	13
CycleGraph_100	100	100	2	34	0.0009
FibonacciTree_10	143	142	1	60	0.001
Balance_tree_3,5	364	363	1	84	0.005
DorogovtsevGoltsevMendesGraph	3282	6561	2	366	0.6

Bảng 3.5: Kết quả thu được khi chạy thuật toán sử dụng phân rã cây trên một số đồ thị

Theo như bảng trên, ta có thể thấy ngoài yếu tố kích thước của đồ thị như số đỉnh hay số cạnh có ảnh hưởng tới thời gian thực thi, mà chất lượng của phân rã cây được xây dựng cũng góp phần không nhỏ trong việc giảm thiểu thời gian thực thi của thuật toán.

Đầu tiên, hãy tìm hiểu 2 thuật toán được sử dụng nhằm so sánh: Arbitrary và Greedy.

Greedy: Ý tưởng của thuật toán này là gán cho mỗi đỉnh của đồ thị một trọng số, trọng số này có thể là bậc của đỉnh đó. Thuật toán liên tục lọc những đỉnh có trọng số lớn nhất để bổ sung vào tập thống trị. Mỗi lần lọc đỉnh đó, ta sẽ cập nhật lại trọng số cho các đỉnh. Quá trình sẽ dừng lại cho đến khi chỉ còn các đỉnh có trọng số là 0.

Arbitrary[4]: thuật toán dựa trên một mệnh đề cho rằng: Với mỗi đồ thị có n đỉnh bậc nhỏ nhất là 3 thì kích thước của tập thống trị nhỏ nhất sẽ nhỏ hơn hoặc bằng $3n/8$. Ý tưởng của thuật toán này sẽ là loại bỏ các đỉnh có bậc bằng 1 hoặc 2 cho đến khi ta thu được một đồ thị chỉ bao gồm những đỉnh có bậc lớn hơn hoặc bằng 3 hay những đỉnh có bậc 0. Gọi V' là tập các đỉnh có bậc ít nhật bằng 3. Với $t = |V'|$, theo mệnh đề trên, ta có thể xác định rằng kích thước của tập thống trị tối thiểu của đồ thị đã cho có kích thước lớn nhất là $3t/8$.

Tiếp theo ta sẽ tiến hành so sánh kết quả thu được từ phương pháp sử dụng phân rã cây với 2 thuật toán được nhắc tới trên. Kết quả của thử nghiệm được thể hiện ở trong bảng dưới đây.

	Đỉnh	**Cạnh**	$\gamma(G)$		
			Treewidth	**Greedy**	**Arbitrary**
ChvatalGraph	12	24	4	4	4
ClebschGraph	16	40	4	4	4
BlanusaSecondSnarkGraph	18	27	5	5	5
DesarguesGraph	20	30	6	6	6
BrinkmannGraph	21	42	5	6	5
DoubleStarSnark	30	45	8	10	8
DyckGraph	32	48	8	10	8
4x12_torusGrid	48	192	12	12	N/A
CycleGraph_100	100	100	34	38	38
FibonacciTree_10	143	142	60	64	N/A
Balance_tree_3,5	364	363	84	88	N/A
DorogovtsevGoltsevMendesGraph	3282	6561	366	366	N/A

Bảng 3.6: So sánh kết quả kích thước tập thống trị tối thiểu thu được của các thuật toán khác nhau trên từng đồ thị

	Thời gian thực thi		
	Treewidth	**Greedy**	**Arbitrary**
ChvatalGraph	0.04	0.031	0.051
ClebschGraph	1.25	0.03	0.054
BlanusaSecondSnarkGraph	0.018	0.034	0.08
DesarguesGraph	0.07	0.03	0.139
BrinkmannGraph	2.05	0.032	0.15
DoubleStarSnark	0.15	0.032	5.173
Dyck	0.41	0.034	5.71
4x12_torusGrid	13	0.031	N/A
CycleGraph_100	0.0009	0.036	0.232
FibonacciTree_10	0.001	0.048	N/A
Balance_tree_3,5	0.005	0.047	N/A
DorogovtsevGoltsevMendesGraph	0.6	0.11	N/A

Bảng 3.7: So sánh thời gian thực thi của 3 thuật toán

Ta có thể nhận thấy một số điểm đặc biệt từ bảng 3.6, thứ nhất là có sự không thống nhất giữa 2 kết quả của 2 thuật toán được đưa vào để so sánh. Trong khi thuật toán Greedy có xu hướng đưa ra một tập đỉnh thống trị có kích thước lớn hơn khi kích thước đồ thị tăng lên, còn thuật toán Arbitrary thì cho ra kết quả trùng với thuật toán sử dụng phân rã cây khá nhiều. Điều thứ hai là thuật toán Arbitrary khi sử dụng với một số đồ thị có kích thước phức tạp hơn thì tồn tại trường hợp thuật toán mất nhiều thời gian nhưng vẫn chưa đưa ra được kết quả. Nguyên nhân của 2 điều này là như sau:

1. Việc 2 thuật toán đưa ra 2 kết quả khác nhau là do thuật toán Greedy bản chất là một thuật toán Heuristic. Điều đó dẫn đến kết quả mà thuật toán đưa ra chưa phải là một kết quả mang tính chính xác nhất hay tối ưu nhất. Trong khi đó, phương pháp được đề cập trong đồ án này tiếp cận bài toán theo hướng chính xác, do đó sẽ có sự khác biệt khi bài toán trở nên phức tạp. Thuật toán Arbitrary cũng tiếp cận bài toán theo hướng chính xác nên ta có thể thấy sự khác biệt là không nhiều.
2. Việc đưa ra một kết quả chính xác của những thuật toán theo hướng chính xác như Arbitrary đòi hỏi thời gian lâu hơn những thuật toán Heuristic khác. Trong một số trường hợp, việc đưa ra kết quả tốn quá nhiều thời gian giống như những thử nghiệm bên trên.

CHƯƠNG 4. ỨNG DỤNG

4.1 Giới thiệu

Tập đỉnh thống trị cực tiểu là một bài toán có thể được ứng dụng rộng rãi trong rất nhiều lĩnh vực của đời sống thực tế. Một số lĩnh vực có thể kể đến như:

1. Xác định vị trí thiết lập các trạm Radar: bài toán đặt ra là có một số những vị trí quan trọng cần được giám sát bởi radar. Câu hỏi được đặt ra là làm thế nào để có thể giám sát toàn bộ những vị trí này mà số lượng trạm radar cần được thiết lập là cực tiểu. [5]

2. Xác định vị trí thiết lập các đơn vị, cơ sở: mục tiêu của bài toán này thường là xác định vị trí những đơn vị, cơ sở công cộng sao cho chúng thỏa mãn những yêu cầu nhất định, tùy thuộc vào những tình huống khác nhau. Điển hình là cực tiểu hóa chi phí vận chuyển, tối đa hóa lượng khách hàng có thể tiếp cận những dịch vụ cần thiết, v..v.. [5]

3. Mạng xã hội: nếu ta xem mạng xã hội dựa trên cấu trúc của một đồ thị, những người dùng mạng xã hội sẽ đóng vai trò như là một đỉnh của đồ thị, còn mối liên kết giữa những người dùng với nhau có thể được coi là những cạnh. Sẽ có những trường hợp mà do sự giới hạn của kinh phí, người ta sẽ chỉ đi tìm những người có sức ảnh hưởng lớn nhất để có thể tạo sự ảnh hưởng đến những người xung quanh. Phương pháp này có thể sử dụng trong việc marketing sản phẩm, tuyên truyền, v..v.. [5]

4. Xây dựng tuyến xe buýt đưa đón học sinh: khác với xe buýt thông thường, một số xe buýt đưa đón học sinh thường tuân thủ một số quy tắc nhất định. Chẳng hạn đảm bảo cho không học sinh nào phải đi bộ nhiều hơn một khoảng cách nhất định, hay một xe chỉ được chở một số lượng học sinh nhất định, v..v.. Do đó, việc xây dựng tuyến xe buýt đảm bảo được quy tắc trên có thể được thực hiện thông qua tập thống trị với một số điều kiện kèm theo. [5]

5. Tóm tắt văn bản: vào năm 2016, hai nhà nghiên cứu đã có một bài báo nói về việc sử dụng một dạng khác của tập thống trị cực tiểu gọi là generalized minimum dominating set, trong đó mỗi đỉnh của đồ thị đại diện cho một câu hay một đoạn văn bản trong một văn bản lớn hơn, trọng số của các cạnh đồ thị là sự tương đồng của 2 đoạn văn bản. Kết hợp với những phương pháp khác, họ đã có thể tiến hành thử nghiệm việc tóm tắt nội dụng của văn bản [6].

6. Mạng cảm biến không dây: bằng cách sử dụng Connected Dominating Set như một xương sống cho hệ thống cảm biến, thông tin có thể truyền đi thông

qua xương sống này từ nguồn cho đến đích mà không khiến cho các cảm biến không liên quan phải tiếp nhận những thông tin thừa, qua đó tiết kiệm được năng lượng cũng như bộ nhớ cho cảm biến [7].

4.2 Bài toán Xác định vị trí thiết lập các đơn vị, cơ sở

4.2.1 Giới thiệu

Bài toán xác định vị trí hay còn gọi là Quy hoạch hướng đến việc tối ưu hóa các vị trí của các cơ sở cung cấp dịch vụ cùng với đó là sự ràng buộc về chi phí vận chuyển cũng như khoảng cách. Những cơ sở này có thể kể đến như trường học, trạm cứu hỏa, bệnh viện, v..v.. Mục tiêu của những cơ sở này là nhằm đảm bảo mọi người có thể tiếp cận với khoảng cách vận chuyển cũng như chi phí phải bỏ ra là nhỏ nhất có thể.

Những cơ sở này có thể được chia thành 2 loại: cơ sở "mong muốn" và cơ sở "không mong muốn". Những cơ sở "mong muốn" là những nơi cung cấp những dịch vụ cần thiết cho người dân như trường học, bệnh viện, trạm cứu hỏa, sở cảnh sát, v..v.. Do đó những nơi này cần được đặt ở những vị trí thuận tiện, ít cản trở nhất có thể đảm bảo việc tiếp cận của người dân luôn luôn dễ dàng. Trong khi đó, những cơ sở "không mong muốn" thì ngược lại, là những nơi như nhà máy, xí nghiệp, khu chất thải - những nơi cần được đặt ở vị trí càng xa khu dân cư càng tốt.

Mặc dù đã có nhiều những giải pháp cho bài toán này, nhưng do đây là một bài toán thuộc lớp NP-complete nên việc đưa ra một giải pháp tối ưu là rất khó. Do đó, Tập thống trị cực tiểu sẽ là một hướng tiếp cận thích hợp. Ta hãy cùng tìm hiểu cách áp dụng của nó thông qua một ví dụ đơn giản.

4.2.2 Phương pháp áp dụng

Ta sẽ xét phạm vi áp dụng là trên một thành phố. Ta coi cả một thành phố là một đồ thị vô hướng với mỗi khu vực cụ thể được coi là một đỉnh trong đồ thị, mối liên kết giữa hai khu vực sẽ là cạnh đồ thị. Cách quy định khu vực ở đây phụ thuộc vào từng hoàn cảnh khác nhau, có thể coi một quận hay một huyện của một thành phố là một đỉnh hay ta có thể nhóm những cụm dân cư có mật độ cao thành một đỉnh. Tương tự với cạnh đồ thị, sự liên kết ở đây có thể khác nhau, điển hình như khu vực giáp nhau hay giữa hai khu vực đó có tuyến đường đi lại thuận tiện, v..v..

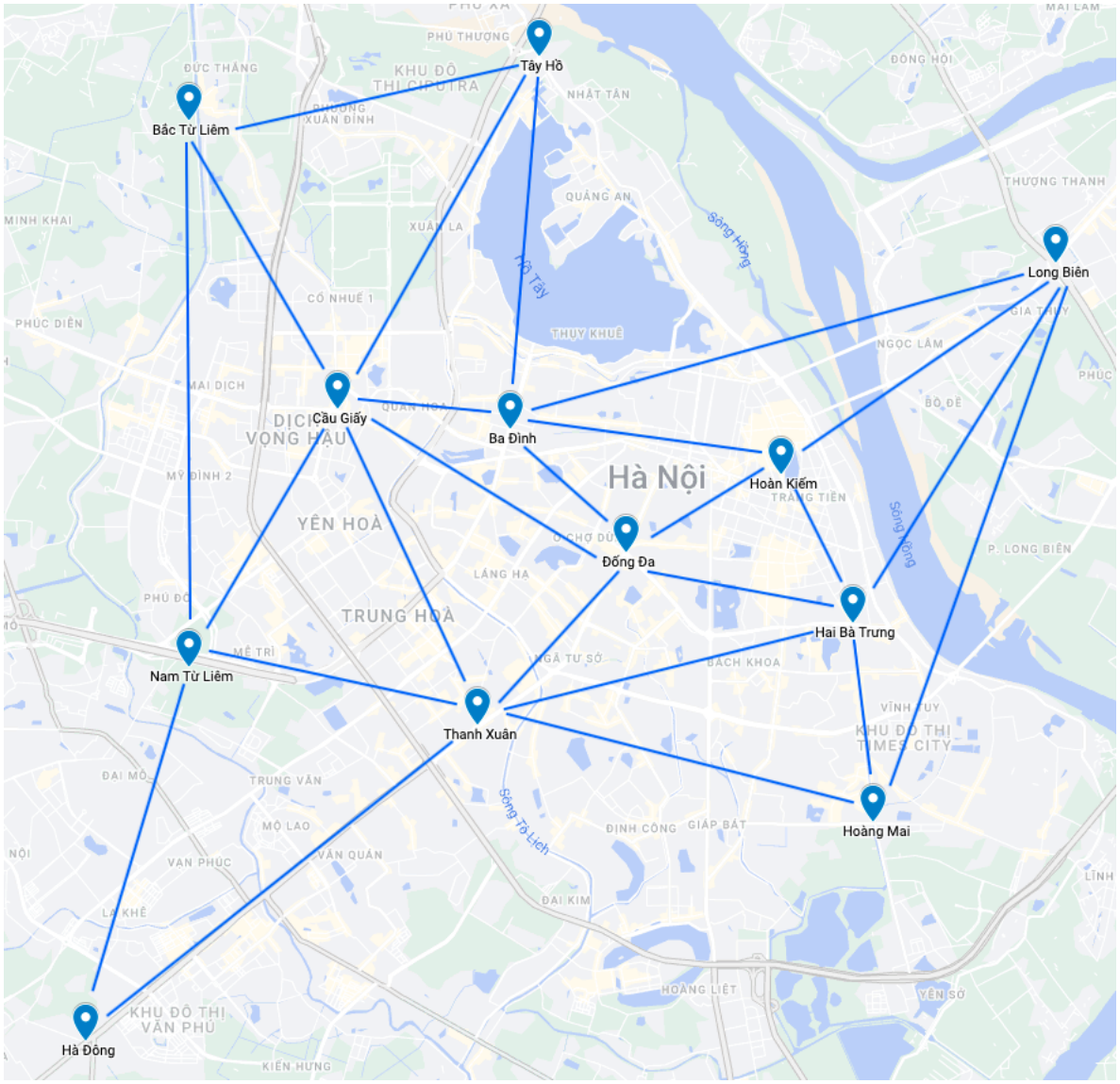
4.2.3 Thử nghiệm

Trong phần này, thành phố Hà Nội sẽ được coi như một đồ thị vô hướng. Trong đó, 12 quận nội thành sẽ được coi là một đỉnh trong đồ thị và cạnh giữa hai đỉnh biểu diễn việc giữa hai quận đó có giáp nhau hay không. Giả sử một tập đoàn bán lẻ có kế hoạch xây dựng các trung tâm thương mại trong nội thành Hà Nội, sao cho

người dân chỉ cần di chuyển giới hạn trong quận mình sống hoặc các quận giáp với quận của mình là có thể tiếp cận được với trung tâm thương mại.

Nói cách khác, mục tiêu cần phải đạt được là xác định đâu là những quận mà có thể tiếp cận được với những quận lân cận mà không tốn quá nhiều chí phí vận chuyển hay thời gian. Đồng thời cần phải tối ưu hóa số lượng quận sao cho nhỏ nhất có thể để có thể tiết kiệm được chi phí xây dựng.

Mối liên hệ giữa 12 quận nội thành của thành phố Hà Nội có thể được biểu diễn dưới dạng đồ thị như sau:



Hình 4.1: Thành phố Hà Nội dưới dạng một đồ thị với 12 đỉnh và 25 cạnh

Ta đánh số cho mỗi quận từ 1 đến 12 theo thứ tự Alphabet:

Tên quận	**Id**
Ba Đình	1
Bắc Từ Liêm	2
Cầu Giấy	3
Đống Đa	4
Hà Đông	5
Hai Bà Trưng	6
Hoàn Kiếm	7
Hoàng Mai	8
Long Biên	9
Nam Từ Liêm	10
Tây Hồ	11
Thanh Xuân	12

Bảng 4.1: Số thứ tự của các quận

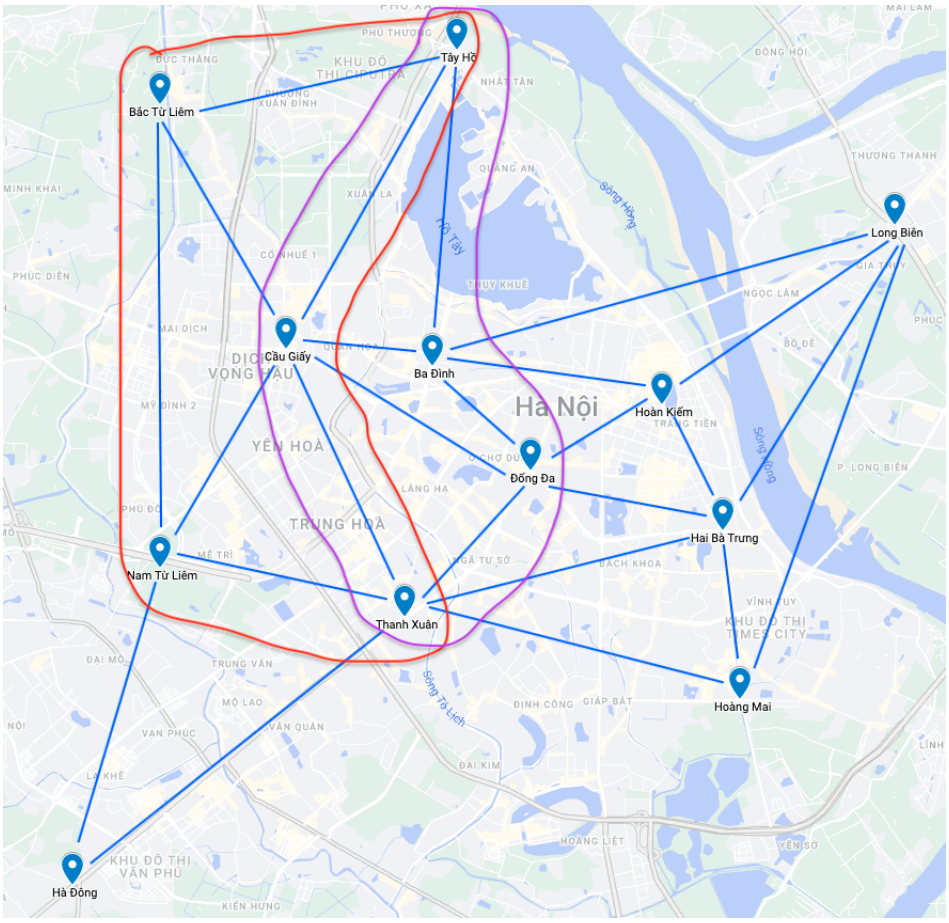
```
p tw 12 25
1 3
1 4
1 7
1 9
1 11
2 3
2 10
2 11
3 4
3 10
3 11
3 12
4 6
4 7
4 12
5 10
5 12
6 7
6 8
6 9
6 12
7 9
8 9
8 12
10 12
```

Bảng 4.2: Dữ liệu đầu vào cho bài toán được lấy từ hình 4.1 bảng 4.1

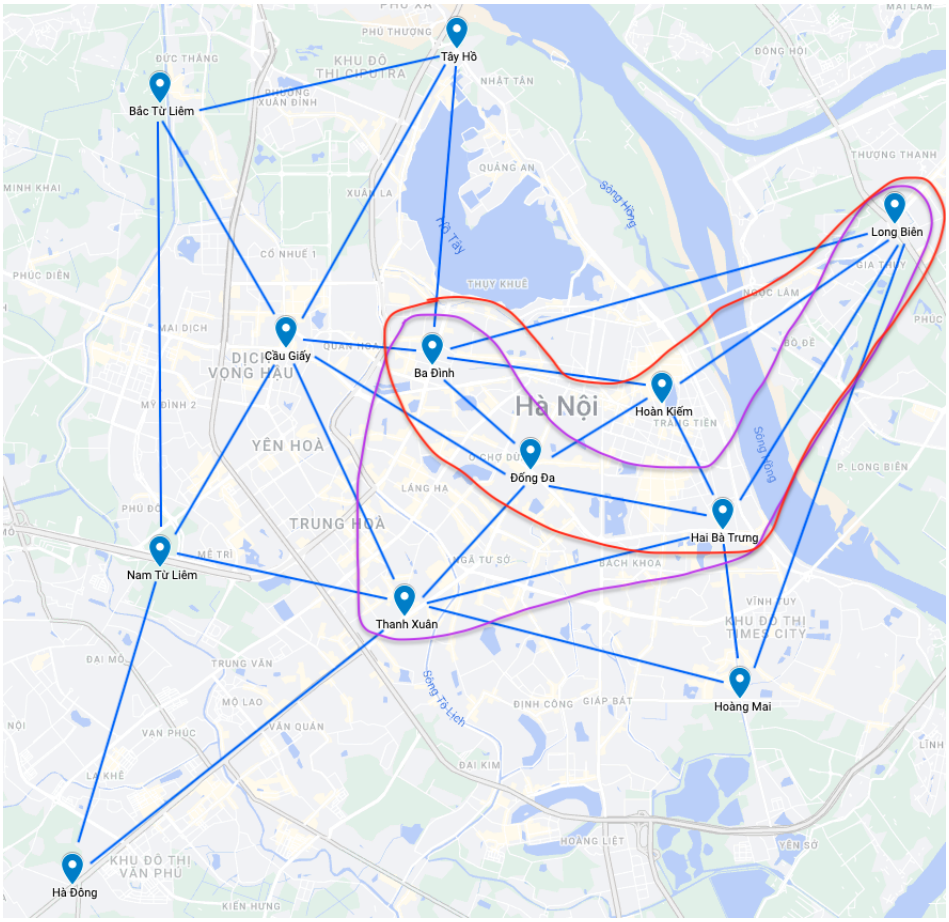
Từ dữ liệu đầu vào trên ta thu được kết quả như sau:

```
4
1 5 2 3 10 11 12
2 5 1 3 4 11 12
3 5 1 4 6 9 12
4 5 1 4 6 7 9
5 4 6 8 9 12
6 3 5 10 12
-1
3 4
3 5
2 3
1 2
1 6
```

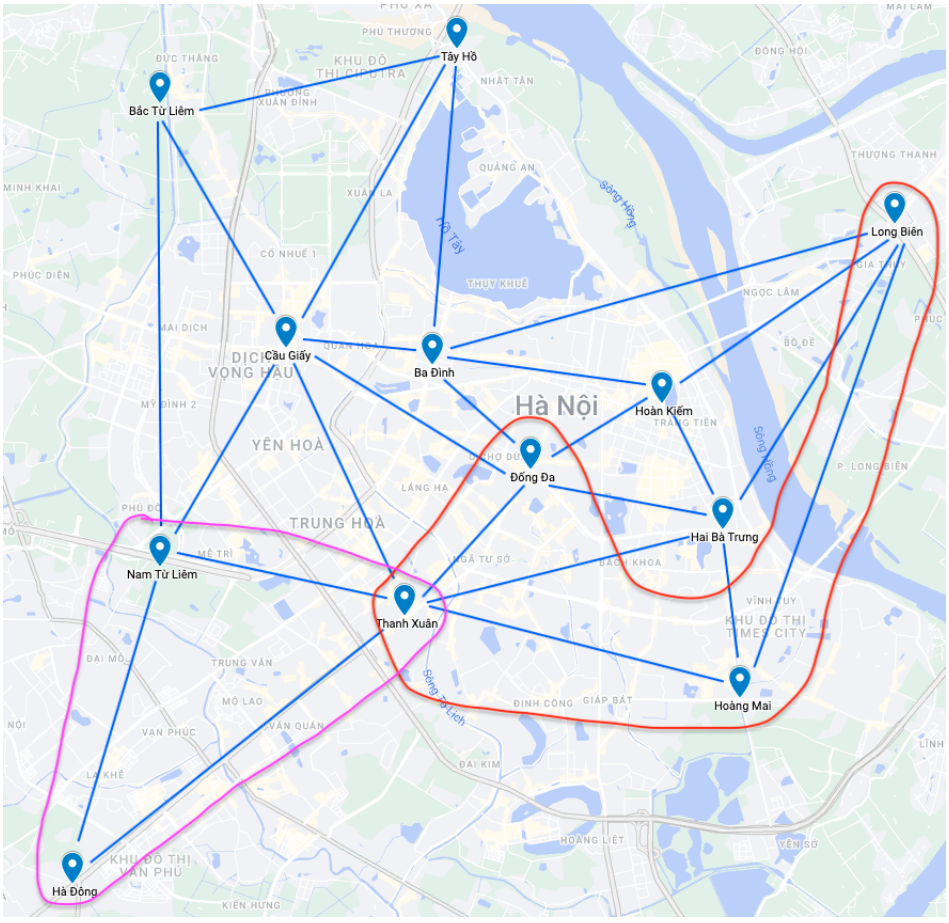
Bảng 4.3: Phân rã cây có 6 khối với độ rộng bằng 4



Hình 4.2: Khối 1: Bắc Từ Liêm - Cầu Giấy - Nam Từ Liêm - Tây Hồ - Thanh Xuân
Khối 2: Ba Đình - Cầu Giấy - Đống Đa - Tây Hồ - Thanh Xuân



Hình 4.3: Khối 1: Ba Đình - Đống Đa - Hai Bà Trưng - Long Biên - Thanh Xuân
Khối 4: Ba Đình - Đống Đa - Hai Bà Trưng - Hoàn Kiếm - Long Biên



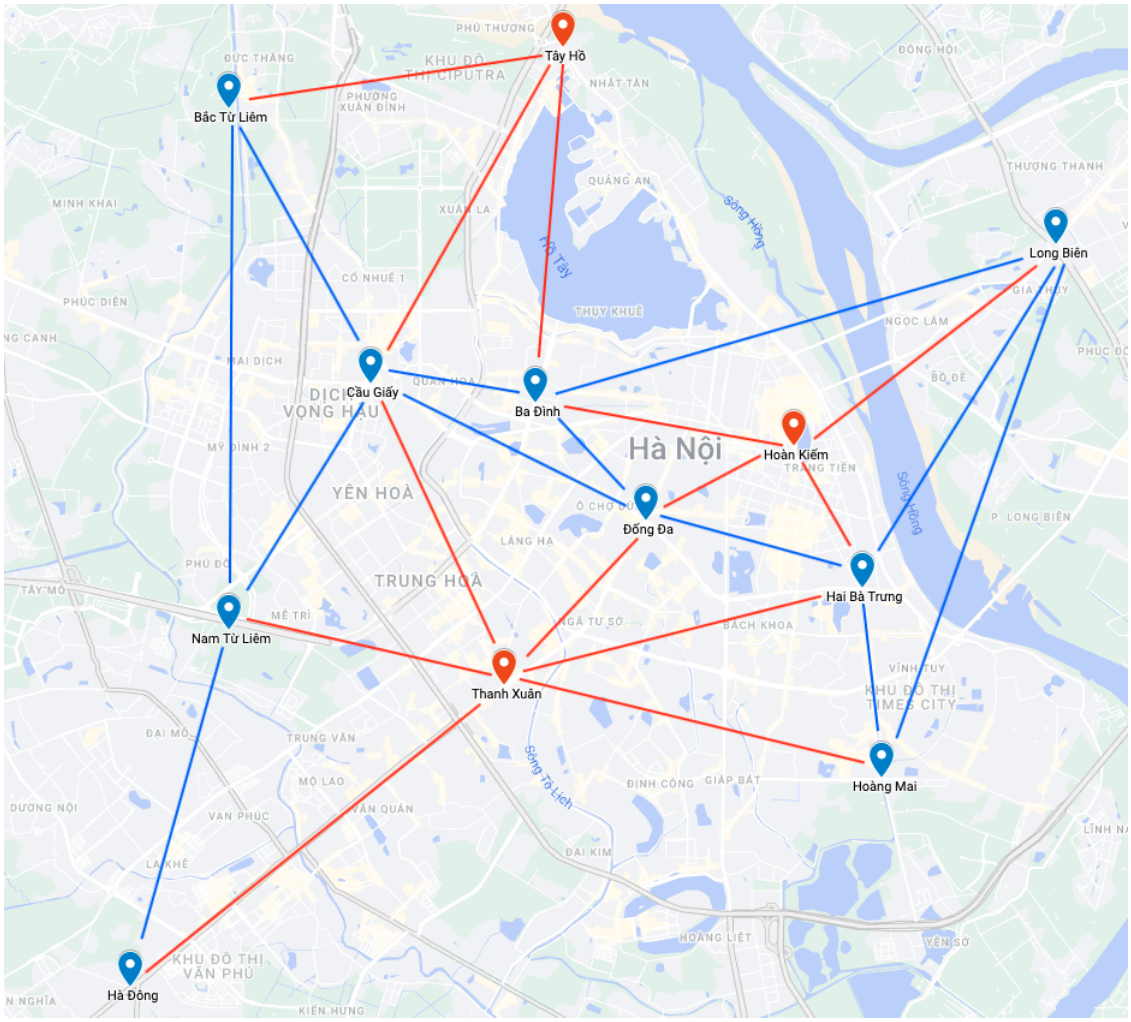
Hình 4.4: Khối 5: Đống Đa - Hoàng Mai - Long Biên - Thanh Xuân
Khối 6: Hà Đông - Nam Từ Liêm - Thanh Xuân

Tiếp tục sử dụng kết quả này để đi tìm tập thống trị cực tiểu, kết quả ra được:

```
Minimum dominating set: 3
7 11 12
time: 0.05
```

Bảng 4.4: Kết quả thu được cho bài toán thực tế

Kết quả trên tương đương với 3 quận: Hoàn Kiếm - Tây Hồ - Thanh Xuân.



Hình 4.5: 3 quận Hoàn Kiếm - Tây Hồ - Thanh Xuân cùng với liên kết của 3 quận này với những quận khác được biểu diễn bằng màu đỏ

4.2.4 Kết luận

Phương pháp được sử dụng ở phần trên chỉ là một hướng tiếp cận đơn giản nhất nhằm minh họa cách mà tập thống trị cực tiểu trong lý thuyết đồ thị có thể được

áp dụng vào bài toán thực tế, ở đây là bài toán xác định vị trí thiết lập các cơ sở dịch vụ. Ngoài ra, phương pháp này có thể được sử dụng với những cơ sở khác như trường học, bệnh viện, hay khu công nghiệp.

Do trong thực tế còn tồn tại rất nhiều biến số mà chưa thể nào biểu diễn một cách đầy đủ trong ví dụ trên, như khoảng cách, mật độ dân số, kinh tế - xã hội từng khu vực, v..v..

Ví dụ trong trường hợp các khu công nghiệp hay khu chứa chất thải, ta không thể quy định mỗi quận của thành phố là một đỉnh của đồ thị do mật độ dân số ở mỗi quận là khác nhau, thay vào đó bằng những thuật toán phân cụm, ta có thể nhóm những khu vực có mật độ dân số đủ để trở thành một đỉnh, từ đó kết hợp với điều kiện các vị trí đặt các cơ sở này cần có một khoảng cách lớn hơn một ngưỡng nhất định, ta có thể xác định được vị trí thiết lập một cách chính xác hơn.

Ngoài ra, do bài toán này có thể tồn tại đồng thời nhiều kết quả thỏa mãn, nên thông qua việc phân tích nhiều kết quả khác nhau, ta có thể đưa ra một phương án khả thi nhất đồng thời đảm bảo việc tiết kiệm chi phí một cách tối đa. Qua đó giúp ích cho các chuyên gia quy hoạch trong việc xây dựng các thành phố thông minh từ đó cải thiện chất lượng cuộc sống cho người dân.

CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Trong đồ án này, tôi đã tiến hành 2 công việc:

1. Tìm hiểu khái niệm, tính chất của phân rã cây; cài đặt thuật toán xây dựng phân rã cây và thử nghiệm trên một số đồ thị cho sẵn.
2. Tìm hiểu khái niệm, tính chất của Tập thống trị cực tiểu trên đồ thị và cài đặt thuật toán sử dụng kỹ thuật quy hoạch động trên phân rã cây để xác định tập thống trị cực tiểu của đồ thị.

5.1.1 Phân rã cây - Tree Decomposition

Phân rã cây của đồ thị $G = (V, E)$ là một cặp $(\{X_i | i \in I\}, T = (I, F))$, với $\{X_i | i \in I\}$ là các tập con của V, sao cho T thỏa mãn:

- Mọi đỉnh của đồ thị G phải tồn tại trong ít nhất một đỉnh của cây T.
- Với mọi cạnh $e \in E$, tồn tại ít nhất một đỉnh của cây T chứa cả hai đầu mút của e.
- Với $t_1, t_2, t_3 \in T$, sao cho t_2 nằm trên đường đi từ t_1 đến t_3. Nếu tồn tại một đỉnh $v \in V$ mà ở đó $v \in V_{t_1}$ và $v \in V_{t_2}$ thì $v \in V_{t_2}$.

Kích thước của phân rã cây $(\{X_i | i \in I\}, T = (I, F))$ được định nghĩa là $\max_{i \in I} |X_i| - 1$. Ta có định nghĩa *treewidth* của đồ thị G $(tw(G))$ là bằng kích thước nhỏ nhất trong tất cả các phân rã cây của đồ thị G.

Có nhiều phương pháp để xây dựng một phân rã cây như:

- Sử dụng thứ tự loại trừ - Elimination Ordering.
- Sử dụng điểm phân chia - Separators.
- Sử dụng quy hoạch động.

Trong đồ án này, tôi đã tiến hành cài đặt chương trình xây dựng phân rã cây sử dụng thứ tự loại trừ và có một số nhận xét như sau:

- Độ chính xác của thuật toán phụ thuộc vào điều kiện mà người lập trình chọn để xác định thứ tự loại trừ. Những điều kiện đó có thể kể đến như: số cạnh fill-in của một đỉnh, bậc của một đỉnh, v..v.. Do đó với mỗi đồ thị, việc xác định tiêu chí để xây dựng thứ tự loại trừ là rất quan trọng.
- Trong trường hợp dữ liệu đầu vào là một phân rã cây tốt, thuật toán đưa ra kết quả với độ chính xác cao và thời gian thực thi hiệu quả hơn đáng kể những

thuật toán khác.

5.1.2 Tập thống trị cực tiểu - Minimum Dominating Set

Tập thống trị D của đồ thị G là tập những đỉnh thỏa mãn: Với mọi đỉnh v của đồ thị G, nếu v không thuộc D thì v sẽ có ít nhất một đỉnh hàng xóm thuộc D. $\gamma(G)$ được định nghĩa là kích thước nhỏ nhất của tập đỉnh thống trị của đồ thị G.

Bài toán đi tìm tập thống trị cực tiểu của đồ thị G là một bài toán thuộc lớp NP-complete. Do đó, không tồn tại một phương pháp chính xác để cho ra một lời giải thỏa mãn cho mọi trường hợp. Tuy nhiên, trong một số trường hợp đặc biệt, ta có thể sử dụng phương pháp quy hoạch động trên phân rã cây để xác định tập thống trị cực tiểu này.

Thuật toán này phụ thuộc nhiều vào chất lượng của phân rã cây đầu vào, kích thước của phân rã cây càng lớn, thời gian thực thi của thuật toán càng lâu.

Tập thống trị cực tiểu có thể ứng dụng trong nhiều lĩnh vực thực tế, như quy hoạch, mạng xã hội, v..v.. Mặc dù chưa thể đưa ra một giải pháp hoàn hảo cho từng lĩnh vực trên, tuy nhiên bằng cách kết hợp với những phương pháp khác, nó có thể cho ta một giải pháp tối ưu ở một số trường hợp nhất định.

5.2 Hướng phát triển

Trong quá trình tìm hiểu, do bị giới hạn về mặt thời gian, kiến thức cũng như kinh nghiệm nên đồ án có thể tồn tại những sai sót. Dưới đây là hướng phát triển trong tương lai:

- Chỉnh sửa chương trình xây dựng phân rã cây nhằm đưa ra phân rã cây có độ rộng tốt hơn cho nhiều đồ thị hơn. Qua đó tăng hiệu suất của thuật toán xác định tập thống trị cực tiểu.
- Tìm hiểu những thuật toán xác định tập thống trị cực tiểu khác từ đó biết được thuật toán nên sử dụng cho mỗi đồ thị.

Trên đây là những điều mà tôi đã đạt được cũng như chưa đạt được trong quá trình thực hiện đồ án. Do kiến thức còn hạn chế, đồ án sẽ không tránh khỏi những thiếu sót. Tôi rất mong nhận được những góp ý từ thầy cô và các bạn để đồ án được hoàn thiện hơn.

TÀI LIỆU THAM KHẢO

[1] T. Kloks, *Computations and Approximations*. Springer Berlin, Heidelberg, 1994. DOI: https://doi.org/10.1007/BFb0045375.

[2] J. M. Tarr, "Domination in graphs," *USF Tampa Graduate Theses and Dissertations*, 2010.

[3] S. Arnborg, D. G. Corneil **and** A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM Journal on Algebraic Discrete Methods*, **journvol** 8, **number** 2, **pages** 277–284, 1987. DOI: 10.1137/0608024.

[4] F. V. Fomin, D. Kratsch **and** G. J. Woeginger, "Exact (exponential) algorithms for the dominating set problem," **in***Graph-Theoretic Concepts in Computer Science* Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, **pages** 245–256.

[5] J. Rani **and** P. Mor, "Domination in graph and some of its applications in various fields," *International Journal of Emerging Technologies and Innovative Research*, **journvol** 6, **pages** 322–326, 4 **april** 2019. **url**: http://www.jetir.org/papers/JETIR1904141.pdf.

[6] Y.-Z. Xu **and** H.-J. Zhou, "Generalized minimum dominating set and application in automatic text summarization," *Journal of Physics*, **journvol** 699, **number** 1, **page** 12 014, 2016. DOI: 10.1088/1742-6596/699/1/012014.

[7] A. Hassani Karbasi **and** R. Ebrahimi Atani, "Application of dominating sets in wireless sensor networks," *International Journal of Security and its Applications*, **journvol** 7, **pages** 185–202, **january** 2013.