**MỤC LỤC**

[MỤC LỤC 1](#_Toc361059276)

[1. Danh mục các bảng 4](#_Toc361059277)

[2. Danh mục các hình 4](#_Toc361059278)

[Chương 1 6](#_Toc361059279)

[MỞ ĐẦU 6](#_Toc361059280)

[1.1. Phạm vi và mục tiêu 6](#_Toc361059281)

[1.2. Những đóng góp chính của khóa luận 7](#_Toc361059282)

[1.3. Cấu trúc khóa luận 8](#_Toc361059283)

[1.4. Qui ước ký hiệu và viết tắt 8](#_Toc361059284)

[Chương 2 10](#_Toc361059285)

[TỔNG QUAN VỀ GIẢI THUẬT 10](#_Toc361059286)

[VÀ ĐỘ PHỨC TẠP CỦA GIẢI THUẬT 10](#_Toc361059287)

[2.1. Giới thiệu 10](#_Toc361059293)

[2.2. Giải thuật và các tính chất của giải thuật 10](#_Toc361059294)

[2.3. Độ phức tạp của giải thuật 13](#_Toc361059295)

[2.4. Một số kỹ thuật cơ bản để thiết kế giải thuật 16](#_Toc361059296)

[2.5. Cơ sở toán học hỗ trợ tính độ phức tạp của giải thuật 22](#_Toc361059299)

[2.6. Kết luận 28](#_Toc361059300)

[Chương 3 29](#_Toc361059301)

[ỨNG DỤNG 29](#_Toc361059302)

[3.1. Giới thiệu 29](#_Toc361059305)

[3.2. Định nghĩa và ví dụ giải thuật ngẫu nhiên 29](#_Toc361059306)

[3.3. Phân loại các giải thuật ngẫu nhiên 31](#_Toc361059307)

[3.4. Các lĩnh vực áp dụng của giải thuật ngẫu nhiên 32](#_Toc361059308)

[3.5. Kết luận 33](#_Toc361059309)

[Chương 4 34](#_Toc361059310)

[GIẢI THUẬT NGẪU NHIÊN CHO MỘT SỐ BÀI TOÁN 34](#_Toc361059311)

[4.1. Giới thiệu 34](#_Toc361059313)

[4.2. Bài toán sắp xếp 34](#_Toc361059314)

[4.3. Bài toán xác định số nguyên tố 39](#_Toc361059315)

[4.4. Bài toán tìm cặp điểm gần nhất 42](#_Toc361059316)

[4.5. Bài toán kiểm tra phép nhân ma trận 47](#_Toc361059317)

[4.6. Bài toán tìm cây bao trùm nhỏ nhất của đồ thị 49](#_Toc361059318)

[4.7. Kết luận 64](#_Toc361059319)

[Chương 5 65](#_Toc361059320)

[HIỆN THỰC CÁC ỨNG DỤNG 65](#_Toc361059321)

[GIẢI THUẬT NGẪU NHIÊN 65](#_Toc361059322)

[5.1. Giới thiệu 65](#_Toc361059324)

[5.2. Bài toán sắp xếp 65](#_Toc361059325)

[5.3. Bài toán xác định số nguyên tố 65](#_Toc361059326)

[5.4. Bài toán tìm cặp điểm gần nhất 65](#_Toc361059327)

[5.5. Bài toán so trùng mẩu 65](#_Toc361059328)

[5.6. Giải thuật ngẫu nhiên tìm cây bao trùm nhỏ nhất của đồ thị 65](#_Toc361059329)

[5.7. Kết luận 82](#_Toc361059330)

[Chương 6 83](#_Toc361059331)

[TỔNG KẾT VÀ ĐỀ NGHỊ 83](#_Toc361059332)

[3. Tổng kết 83](#_Toc361059334)

[4. Đề nghị 83](#_Toc361059335)

[TÀI LIỆU THAM KHẢO 83](#_Toc361059336)

## Danh mục các bảng

## Danh mục các hình

[Hình 4.4.1 43](#_Toc361059337)

[Hình 4.4.2 43](#_Toc361059338)

[Hình 4.4.3 Các dạng mở rộng khi nhân đôi *δ* 44](#_Toc361059339)

[Hình 4.4.4 Minh họa ví dụ 4.4.1 46](#_Toc361059340)

[**Hình 4.6.1 Đồ thị ban đầu** 51](#_Toc361059341)

[Hình 4.6.2 Các cạnh được chọn trong Boruka step 52](#_Toc361059342)

[Hình 4.6.3 Đồ thị sau khi xây dựng 52](#_Toc361059343)

[Hình 4.6.4 Kết quả sau khi áp dụng Boruvka step lần thứ nhất 53](#_Toc361059344)

[Hình 4.6.5 Các cạnh được chọn trong Boruvka step lần 2 54](#_Toc361059345)

[Hình 4.6.6 Đồ thị được xây dựng lại từ Boruvka step lần 2 54](#_Toc361059346)

[Hình 4.6.7 Đồ thị kết quả thu được từ Boruvka step lần 2 55](#_Toc361059347)

[Hình 4.6.8 Cây bao trùm nhỏ nhất thu được 55](#_Toc361059348)

[Hình 4.6.9 57](#_Toc361059349)

[Hình 4.6.10 58](#_Toc361059350)

[Hình 4.6.11 59](#_Toc361059351)

[Hình 4.6.12 60](#_Toc361059352)

[Hình 5.6.1 Giao diện chính của giải thuật tìm cây bao trùm nhỏ nhất 66](#_Toc361059353)

[Hình 5.6.2 Nhập trọng số 66](#_Toc361059354)

[Hình 5.6.3 Chọn vị trí lưu 67](#_Toc361059355)

[Hình 5.6.4 Chọn một tập tin để mở. 67](#_Toc361059356)

[Hình 5.6.5 Kết quả thu được khi thực hiện giải thuật tìm cây bao trùm nhỏ nhất 69](#_Toc361059357)

[Hình 5.6.6 Đồ thị thời gian chạy. 69](#_Toc361059358)

[Hình 5.6.7 71](#_Toc361059359)

[Hình 5.6.8 73](#_Toc361059360)

[Hình 5.6.9 74](#_Toc361059361)

[Hình 5.6.10 75](#_Toc361059362)

[Hình 5.6.11 76](#_Toc361059363)

[Hình 5.6.12 77](#_Toc361059364)

[Hình 5.6.13 78](#_Toc361059365)

[Hình 5.6.14 79](#_Toc361059366)

[Hình 5.6.15 80](#_Toc361059367)

[Hình 5.6.16 Biểu đồ thời gian chạy của giải thuật Prim 81](#_Toc361059368)

[Hình 5.6.17 Biểu đồ thời gian chạy của giải thuật ngẫu nhiên 82](#_Toc361059369)

# Chương 1

# MỞ ĐẦU

## Phạm vi và mục tiêu

Các nhà khoa học vào thế kỉ trước đã thấy rằng tính chất ngẫu nhiên là thành phần thiết yếu trong việc mô hình hóa và phân tích tự nhiên. Ngày nay, sự ngẫu nhiên đóng vai trò quan trọng trong hầu hết mọi lĩnh vực của khoa học từ di truyền học và tiến hóa trong sinh học đến mô hình biến động giá của nền kinh tế thị trường tự do. Khoa học máy tính cũng không ngoại lệ, phương pháp ngẫu nhiên và xác xuất đóng vai trò cốt lỗi trong khoa học máy tính hiện đại. Trong hai thập kỉ qua chúng ta đã chứng kiến sự phát triển to lớn của việc sử dụng lý thuyết xác xuất trong khoa học máy tính. Thêm vào đó nhiều kỹ thuật xác suất hiện đại và cải tiến tạo điều kiện cho việc phát triển các giải thuật ngẫu nhiên.

Khái niệm giải thuật ngẫu nhiên là một khái niệm tương đối mới. Trong mọi giải thuật được giới thiệu từ xưa đến nay, mỗi bước trong giải thuật đều được xác định. Với giải thuật ngẫu nhiên thì chúng ta không xác định được, trong quá trình thực hiện một giải thuật ngẫu nhiên nó sẽ đưa ra một chọn lựa tùy ý. Một số hành động được thực hiện một cách ngẫu nhiên. Ngoài đầu vào, giải thuật sử dụng một nguồn các con số ngẫu nhiên. Trong quá trình tính toán, nó sử dụng những lựa chọn ngẫu nhiên phụ thuộc vào những số ngẫu nhiên. Output có thể khác nhau nếu giải thuật chạy nhiều lần trên cùng một input.

Hơn nữa, giải thuật ngẫu nhiên thường đơn giản và dễ thực hiện. Các giải thuật này chạy nhanh hơn giải thuật cổ điển giải cùng một bài toán. Yêu cầu về không gian của một giải thuật ngẫu nhiên có thể nhỏ hơn của giải thuật nhất định mà chúng ta biết để giải cùng một bài toán . Trong vài trường hợp, giải thuật ngẫu nhiên là cách duy nhất hoặc tốt nhất để giải quyết vấn đề.(**Error! Reference source not found.**,**Error! Reference source not found.**,**Error! Reference source not found.**)

Giải thuật ngẫu nhiên và xác suất đóng vai trò quan trọng trong khoa học máy tính, với những ứng dụng từ tối ưu tổ hợp và máy tính học cách giao tiếp với mạng và giao thức bảo mật.

Hiện nay, các giải thuật được áp dụng trong lĩnh vực khoa học máy tính đã được tối ưu rất nhiều, các giải thuật này đã chạy rất nhanh. Tuy nhiên, thời gian rất quý giá, một giây cũng có thể thay đổi rất nhiều thứ. Với giải thuật ngẫu nhiên chúng ta sẽ tiết kiệm được rất nhiều thời gian so với giải thuật truyền thống cùng giải một bài toán. Thêm vào đó, giải thuật ngẫu nhiên có thể sử dụng ít tài nguyên hệ thống hơn. Với những ưu điểm mà giải thuật ngẫu nhiên mang lại, thì việc nghiên cứu các giải thuật ngẫu nhiên là hết sức có ý nghĩa. Vì vậy, chúng em đã chọn đề tài “*Nghiên cứu và ứng dụng các giải thuật ngẫu nhiên*” để làm khóa luận. Trong khóa luận này chúng em đã đầu tư nghiên cứu các giải thuật ngẫu nhiên, so sánh giải thuật ngẫu nhiên với các giải thuật cổ điển đã biết.

## Những đóng góp chính của khóa luận

Dưới đây là những đóng góp chính của khóa luận đối với lĩnh vực khoa học máy tính:

1. Trình bày khái niệm giải thuật và cách tính toán độ phực tạp của giải thuật
2. Trình bày khái niệm giải thuật ngẫu nhiên
3. Trình bày và áp dụng lý thuyết xác suất để tính toán độ phức tạp, cũng như khả năng cho ra kết quả đúng của giải thuật ngẫn nhiên.
4. Hiện thực một số giải thuật ngẫu nhiên cho các bài toán thuộc các lĩnh vực toán học, tìm kiếm, lý thuyết đồ thị.

## Cấu trúc khóa luận

Khóa luận bao gồm 6 chương. Chương 1 trình bày phạm vi, mục tiêu và ý nghĩa về lý thuyết cũng như ứng dụng của đề tài khóa luận, giới thiệu cấu trúc, các qui ước ký hiệu và viết tắt trong khóa luận. Mỗi chương tiếp theo, từ Chương 2 đến Chương 6 có một phần giới thiệu và một phần kết luận.

Chương 2 giới thiệu tổng quan về giải thuật và độ phức tạp của giải thuật. Đây là một số kiến thức cần thiết sử dụng cho việc thiết kế giải thuật. Chương này còn giới thiệu các công cụ toán học hỗ trợ phân tích, đánh giá độ phức tạp của giải thuật

Chương 3 trình bày tổng quát về giải thuật ngẫu nhiên và phân loại giải thuật ngẫu nhiên.

Chương 4 trình bày một số giải thuật ngẫu nhiên cho các bày toán thuộc nhiều lĩnh vực khác nhau như: sắp xếp, tìm kiếm, toán học, lý thuyết đồ thị….

Chương 5 là phần hiện thực các giải thuật ngẫu nhiên được nêu ra ở chương 4.

Chương 6 là phần tổng kết và đề nghị các hướng nghiên cứu trong tương lai liên quan đến giải thuật ngẫu nhiên.

## Qui ước ký hiệu và viết tắt

Trong sách này có một số qui ước và kí hiệu viết tắt như sau:

: Với mọi

: Thuộc

: Tồn tại

: -lớn

: Omega

: Theta

: Phép toán giao tập hợp

: Phép toán hợp tập hợp

BT: Bài toán.

In : Input

Out : Output

# Chương 2

# 

# TỔNG QUAN VỀ GIẢI THUẬT

# VÀ ĐỘ PHỨC TẠP CỦA GIẢI THUẬT



## Giới thiệu

Chương này giới thiệu một cách khái quát về giải thuật, độ phức tạp thời gian của giải thuật và trình bày một số công cụ toán học hỗ trợ việc phân tích và thiết kế giải thuật.

Đầu tiên Phần 2.2 trình bày khái niệm giải thuật, tính chất và ngôn ngữ biểu diễn giải thuật. Kế đến Phần 2.3 nêu khái niệm độ phức tạp của giải thuật, cách biểu diễn độ phức tạp thời gian của giải thuật và một số ví dụ minh họa. Tiếp theo, Phần 2.4 là trình bày một số chiến lược thiết kế cơ bản của giải thuật là cơ sở cho việc phát triển các giải thuật truyền thống cũng như giải thuật ngẩu nhiên. Phần 2.5 trình bày một số công cụ toán học làm cơ sở cho việc tính toán độ phức tạp của giải thuật. Cuối cùng, Phần 2.6, là các kết luận đáng lưu ý của Chương này.

## Giải thuật và các tính chất của giải thuật

Một cách không hình thức, giải thuật là cách thức để đạt được lời giải của bài toán dựa trên một số hữu hạn các bước cần phải thao tác, tính toán. Giải thuật được định nghĩa một cách hình thức ([4],[8]) như sau:

**Định nghĩa 2.2.1** *Giải thuật* là một thủ tục xác định bao gồm một dãy hữu hạn các bước cần thực hiện để thu được lời giải bài toán.

Xét về bản chất toán học, mỗi giải thuật tương ứng với một *ánh xạ* (mapping). Nghĩa là, ánh xạ là mô hình toán học của giải thuật. Mỗi giải thuật có một tên, một tập dữ liệu *đầu vào* (input) và một tập dữ liệu *đầu ra* (output) tương ứng với yêu cầu và lời giải bài toán. Một giải thuật có một số đặc tính cơ bản sau.

1. Tính *chính xác* (precision), đảm bảo kết quả tính toán hay các thao tác giải thuật được thực hiện là chính xác.
2. Tính *hữu hạn* (finiteness), nghĩa là thuật giải phải dừng sau một số hữu hạn bước thực hiện.
3. Tính *phổ dụng* (generality), nghĩa là giải thuật có thể áp dụng cho một lớp các bài toán có cùng kiểu dữ liệu đầu vào với kích thước khác nhau.

Để có thể mô tả và biểu diễn giải thuật và từ đó có thể viết thành các chương trình máy tính, chúng ta có thể dùng *ngôn ngữ tự nhiên* (natural language), *mã giả* (pseudocode) hay *ngôn ngữ lập trình cấp cao* (high programming language) như Pascal, C/C++, v.v. Trong phạm vi khóa luận này, các giải thuật sẽ được trình bày bằng mã giả.

Ví dụ 2.2.1 Giải thuật tìm số *k* trong dãy số *a*1, *a*2, ...., *an*, trong đó n là một số tự nhiên, có thể được viết như sau.

SequentialSearch(*a*[1..*n*], *k*)

1 **for** *i* ← 1 **to** *n*

2 **do if** *a*[*i*] =*k*

3 **then** **return** true

4 **return** false

Ở đây, đầu vào là một số *k* và một dãy *n* số *a*[1], *a*[2],…, *a*[*n*], đầu ra là một giá trị logic biểu diễn có hay không số *k* trong dãy đã cho. Đây cũng là một giải thuật có mặt hầu như trong bất kỳ hệ thống thông tin nào. Chẳng hạn, trong CSDL bài toán tìm kiếm một đối tượng theo một khóa dựa trên thuật giải này.

Trong lý thuyết độ phức tạp tính toán, giải thuật được chia làm hai loại. Giải thuật *đơn định* (deterministic)và *không* *đơn định* (nondeterministic) như các định nghĩa sau.

**Định nghĩa 2.2.2** Một giải thuật được gọi là *đơn định* nếu kết quả của mỗi thao tác, tính toán luôn luôn được xác định một cách duy nhất.

Giải thuật đã nêu trong Ví dụ 2.2.1 là một giải thuật đơn định. Vì rõ ràng kết quả của mỗi tính toán trong giải thuật này là duy nhất.

**Định nghĩa 2.2.3** Một giải thuật được gọi là *không đơn định* nếu tồn tại các thao tác, tính toán mà kết quả thuộc về một tập nhưng không biết chính xác kết quả nào trong tập đó.

Ví dụ 2.2.2 Giải thuật sắp xếp sau đây là một giải thuật không đơn định (có độ phức tạp O(n))

NSort(*A*[1..*n*]) // sắp xếp *n* số nguyên dương

1 **for** *i* ← 1 **to** *n* **do** *B*[*i*] ← 0

2 **for** *i* ← 1 **to** *n*

3 **do**  *j* := **choice**(1:*n*) //thực hiện đồng thời nhiều bản sao (không đơn định)

4 **if** *B*[*j*] <> 0 **then failure**

5 **else** *B*[*j*] ← *A*[*i*]

6 **for** *i* ← 1 **to** *n*-1

7 **do if** *B*[*i*] > *B*[*i*+1] **then failure**

8 **return** *B* //*B*[1..*n*] là mảng được sắp các số trong *A*[1..*n*]

Lớp các giải thuật ngẫu nhiên được trình bày trong các chương sau là các giải thuật không đơn định.

## Độ phức tạp của giải thuật

*Độ phức tạp của giải thuậ*t (complexity of algorithm) là độ do về tính hiệu quả của giải thuật. Độ phức tạp là phương tiện, công cụ để đánh giá giải thuật tốt hay xấu, giải thuật dễ hay khó. Độ phức tạp giải thuật là một khái niệm cơ bản, trung tâm của lý thuyết phân tích và thiết kế giải thuật.

**Định nghĩa 2.3.1** *Độ phức tạp của giải thuật* là chi phí về tài nguyên của hệ thống (chủ yếu là thời gian, bộ nhớ, bộ xử lí và đường truyền) cần thiết để thực hiện giải thuật ([4] **Error! Reference source not found.**).

Để tính toán độ phức tạp của giải thuật, cần thiết phải phân tích giải thuật. Đó là kỹ thuật đánh giá tài nguyên chi phí cho giải thực hiện như định nghĩa sau.

**Định nghĩa 2.3.2** Phân tích giải thuật (analyzing of algorithm) là quá trình tìm ra những đánh giá về tài nguyên cần thiết để thực hiện giải thuật.

Trong khóa luận này, chỉ độ phức về thời gian của giải thuật được quan tâm, vì độ phức tạp thời gian của giải thuật phụ thuộc chủ yếu vào kỹ thuật thiết kế và độc lập với phần cứng của hệ thống máy tính.

Một cách cụ thể hơn, thời gian chi phí cho một giải thuật phụ thuộc vào số thao tác (lệnh) mà giải thuật thực hiện, nghĩa là phụ thuộc vào kích thước đầu vào của giải thuật. Nói các khác thời gian thực hiện của một giải thuật là một hàm *T*(*n*) của đối số n nguyên dương biểu diễn kích thức của dữ liệu bài toán mà giải thuật được thiết kế để giải nó. Trong quá trình phân tích giải thuật, thời gian được tính toán có thể được chia thành ba trường hợp như định nghĩa sau.

**Định nghĩa 2.3.3** Thời gian tối thiểu, nhiều nhất, trung bình để thực hiện giải thuật với kích thước đầu vào n gọi là thời gian chạy *tốt nhất* (best-case), *xấu nhất* (worst-case), *trung bình* (average-case) của giải thuật.

Ví dụ 2.3.1 Xét giải thuật trong Ví dụ 2.2.1, gọi α, β và γ là thời gian thực hiện của phép gán, phép so sánh và trả về của giải thuật, thì

1. Trường hợp tốt nhất xảy ra nếu *a*[1] = *k*, khi đó *T*(*n*) = α+β + γ.
2. Trường hợp xấu nhất xảy ra nếu *k* ∉ {*a*[1], *a*[2], ...., *a*[*n*]} khi đó *T*(*n*) = (*n*+1)α + *n*β + γ.
3. Trường hợp trung bình được tính toán dựa trên kỳ vọng xác suất. Gọi xác suất tìm kiếm thành công là *p* (xác không tìm thấy là 1-*p*) khi đó

*T*(*n*) = Σ*i*=1,..*n*(*i*α + *i*β + γ)*p*/*n* + ((*n*+1)α + *n*β + γ))(1-*p*)

= (α + β)(*n*+1)*p*/2 +γ*p* + ((*n*+1)α + *n*β + γ))(1-p)

Lưu ý rằng khi *p*=1 (luân luôn tìm thấy), thì *T*(*n*)= (α + β)(*n*+1)/2 +γ.

Để biểu diễn độ phức tạp giải thuật một cách đơn giản, gần đúng, chúng ta dùng khái niệm tốc độ tăng của hàm như định nghĩa sau ([4][8]).

**Định nghĩa 2.3.3** Giả sử là hàm không âm đối số nguyên dương *n*, khi đó

1. có tốc độ tăng (bậc) không quá và viết nếu có hằng số và số nguyên dương N sao cho ,
2. có tốc độ tăng ít nhất là và viết nếu có hằng số và số nguyên dương N sao cho ,
3. có tốc độ tăng là và viết nếu và

Các ký hiệu O, và giọi là ký hiệu tiệm cận (asymptotic notation). Chúng ta dễ dàng chứng minh được định lý sau.

**Định lý 2.3.1** Giả sử và *c* là một hằng số thì

Định lý có thể được phát biểu tương tự cho các ký hiệu và Bây giờ thời gian chạy của một giải thuật có thể được định nghĩa theo các ký hiệu tiệm cận như sau.

**Định nghĩa 2.3.4** Nếu giải thuật có thời gian chạy tốt nhất (trung bình, xấu nhất) là và thì ta nói thời gian chạy tốt nhất (trung bình, xấu nhất) của giải thuật có bậc (tốc độ tăng) không quá hay thời gian chạy tốt nhất (trung bình, xấu nhất) của giải thuật là .

Chúng tôi lưu ý rằng, Định nghĩa 3.2.4 có thể được phát biểu cho các ký hiệu tiệm cận. Tốc độ tăng của thời gian chạy càng lớn thì giải thuật càng chậm (chẳng hạn, giải thuật có thời gian chạy hiệu quả hơn giải thuật có thời gian chạy

**Ví dụ 2.3.2** Phân tích giải thuật tính gần đúng

Exp(x, n)

1 s ←1

2 **for** i ←1 to n

3 **do** p ←1

4 **for** j ←1 to i

5 **do** p ←p\*x/j

6 s ←s + p

7 **return** s

Gọi lần lượt là thời gian thực hiện lệnh gán, trả về, thì

Tốc độ tăng của độ phức tạp giải thuật biểu diễn theo O, , phụ thuộc vào số lần thực hiện của thao tác (lệnh) được thực thi nhiều lần nhất và có có chi phí lớn nhất trong giải thuật. Các thao tác được thực thi nhiều lần nhất và có có chi phí lớn nhất trong giải thuật gọi là *thao tác cơ bản* (basic operation). Vì vậy, có thể tính độ phức tạp đơn giản bằng cách chỉ xác định và đếm số thao tác cơ bản trong giải thuật ([4],[9]).

**Ví dụ 2.3.3** Phân tích độ phức tạp của giải thuật sau (một cải tiến của giải thuật trong Ví dụ 2.3.2)

Exp(, n)

1 s ←1

2 p ←1

3 **for** i 🡨1 to *n*

4 **do** p ← // phép toán cơ bản là phép chia

5 s ←s + p

6 **return** s

Rõ ràng lệnh cơ bản là *x*/*i*, nên , trong đó là thời gian thực hiện của thao tác *x*.

## Một số kỹ thuật cơ bản để thiết kế giải thuật



Để có thể phát triển các giải thuật hiệu quả cho các bài toán ứng dụng thực tế, cần có các phương pháp, cách thức và *chiến lược thiết kế* (design strategy) một cách khoa học. Dưới đây là một số chiến lược thiết kế cơ bản nhất, thường được ứng dụng nhiều nhất trong quá xây dựng các giải thuật.

Chiến lược trực tiếp



Chiến lược *trực tiếp* (brute force) áp dụng để thiết kế giải thuật cho một bài toán dựa trên định nghĩa và các khái niệm liên quan trực tiếp đến bài toán đó ([4][4]). Đây là chiến lược dễ dàng áp dụng và được lựa chọn đầu tiên trong quá trình phân tích thiêt kế giải thuật cho bài toán . Chiến lược trực tiếp được áp dụng cho một lớp rất rộng các bài toán. Chi phí thiết kế rẻ, nên chiến lược trực tiếp thích hợp cho các bài toán kích thước nhỏ. Tuy nhiên, chiến lược trực tiếp có thể sinh ra một số giải thuật có độ phức tạp khá lớn hoặc rất lớn). Mặc dù vậy, chiến lược trực tiếp là cơ sở để đề xuất các giải thuật mới.

**Ví dụ 2.4.1** Xét bài toán tính tổng *S*=1+2+…+ *n*, với *n* là một số nguyên dương. Giải thuật dựa trên chiến lược trực tiếp để tính tổng này như sau.

Sum(*n*)

1 *S* ←0

2 **for** *i* ←1 to *n*

3 **do** *S* ←*S*+*i*

4 **return** *S*

Rõ ràng đây là một giải thuật được thiết kế trực tiếp và tự nhiên nhất dựa trên biểu diễn trực tiếp của tổng *S*. Kỹ thuật đơn giản là cộng dồn (như trong giải thuật), hoặc đệ qui đều có thể tính *S* một cách trực tiếp. Dễ thấy độ phức tạp của giải thuật là O(*n*). Tuy nhiên nếu dùng phép biến đổi *S*=*n*(*n*+1)/2 thì sẽ có giải thuật O(1), hiệu quả hơn rất nhiều.

Chiến lược chia để trị

Chiến lược chia để trị (Divide-and-Conquer) là chiến lược thiết kế được áp dụng rất rộng rãi trong giải toán nói chung và thiết kế giải thuật nói riêng ([4]). Giải thuật giải một bài toán dựa trên chiến lược chia để trị được phát triển theo ba bước:

1. Chia bài toán thành các bài toán con, thường là cùng kiểu.
2. Giải các bài toán con (thường dùng kỹ thuật đệ qui).
3. Kết hợp lời giải các bài toán con để có lời giải bài toán ban đầu .

**Ví dụ 2.4.2** Cho một mảng các số được sắp theo thứ tự tăng và một số , tìm trong mảng số có giá trị bằng

BinarySearch(*A*[*p*..*r*], *K*)

1. **if** *p*≤*r*
2. **then** *m* ← ⎣(*p*+*r*)/2⎦
3. **if** *A[m]=K*
4. **then return** true
5. **else if** *A*[*m*]>*K*
6. **then return** BinarySearch(*A*[*p*..*m*-1], *K*)
7. **else return** BinarySearch(*A*[*m+1*..*r*], *K*)
8. **else return** false

Để tính độ phức tạp của giải thuật này, đặt *n*=*r*+*p*-1 (số phần phần tử của mảng). Khi đó *T*(*n*)= O(1), nếu *p*>*r* và *T*(*n*)=*T*(⎣*n*/2⎦)+O(1), nếu *p*≤*r*. Không mất tính tổng quát giả sử *m* là số nguyên sao cho *n*=2*m*. Khi đó *T*(*n*)≤*T*(*n*/2)+1=*T*(*n*/22)+2=…= *T*(*n*/2*m*)+*m*. Vậy *T*(*n*) ≤T(1)+*m*= O(1)+log2*n*=O(log2*n*).

Chiến lược quy hoạch động



Chiến lược qui hoạch động (dymamic programming) giải một bài toán bằng cách giải các bài toán con kích thước nhỏ hơn([4]). Nghiệm của bài toán có được bằng cách kết hợp nghiệm của các bài toán con. Nghiệm (lời giải) bài toán ban đầu và các bài toán con được biểu diễn bởi một hệ thức truy hồi. Sử dụng chiến lược qui hoạch động để giải một bài toán được chia làm hai bước:

1. Mô hình hóa lời giải bài toán bằng một hệ thức truy hồi
2. Giải hệ thức truy hồi bằng một giải thuật hiệu quả (từ dưới lên)

Chiến lược qui hoạch động phát triển giải thuật giải bài toán bằng cách giải các bài toán con từ *dưới lên* (bottom up). Kết quả các bài toán con được lưu trữ trong một bảng và được sử dụng để giải bài toán đã cho. Vì kết quả các bài toán con được lưu trữ trong một bảng và được sử dụng để giải bài toán ban đầu nên chiến lược qui hoạch động rất hiệu quả. Qui hoạch động thường được ứng dụng để giải các bài toán tối ưu.

Ví dụ 2.4.3 Cho một dãy n đồng xu có giá trị tương ứng là c1, c2, . . . , c*n*, hãy nhặt một số các đồng xu sao cho tổng giá trị lớn nhất và không có 2 đồng xu nào kề nhau.

Trước hết cần xây dựng hệ thức truy hồi biểu diễn tổng giá trị lớn nhất các đồng xu nhặt được F(*n*) xét hai trường hợp. Nếu đồng xu cuối cùng được chọn thì giá trị lớn nhất là *cn*+F(*n*-2). Nếu đồng xu cuối cùng không được chọn thì giá trị lớn nhất là F(*n-*1). Suy ra hệ thức truy hồi là

F(*n*)=*max*{*cn*+ F(*n* − 2), F(*n* − 1)} với *n* > 1,

F(0)=0, F(1)=*c*1

Giải thuật qui hoạch động được thiết kế như sau.

CoinRow(C[1..n])

1 F[0]←0; F[1]←C[1]

2 **for** *i* ←2 **to** *n*

3 **do**  F[*i*]←*max*(C[*i*]+ F[*i* − 2], F[*i* − 1])

4 **return** F[*n*]

Dễ thấy độ phức tạp của giải thuật là O(*n*). Chúng tôi lưu ý rằng, nếu dùng chiến lược trực tiếp để giải bài toán này sẽ dẫn đến giải thuật có độ phức tạp lũy thừa (rất lớn).

Chiến lược tham ăn



Giải thuật được thiết kế bằng chiến lược *tham ăn* (greedy) giải các bài toán con trước khi giải bài toán gốc. Quá trình tìm lời giải được thực hiện thông qua một dãy các bước để tìm lời giải (nghiệm) tối ưu cục bộ (lời giải các bài toán con) cho đến khi tìm thấy lời giải bài toán gốc. Mỗi bước tìm nghiệm cục bộ thỏa mãn ba tính chất sau.

1. Phải thỏa mãn ràng buộc bài toán.
2. *Tối ưu cục bộ* (locally optimal)
3. Không thay đổi nghiệm cục bộ trong các bước kế tiếp.

Chiến lược greedy luôn thực hiện một “lựa chọn” tốt nhất hiện tại khi tìm kiếm lời giải bài toán con mà không quan tâm đến bước tiếp theo. Chiến lược gready không giải tất cả các bài toán con (tối ưu) như qui hoạch động mà mỗi bước chỉ tìm lời giải tối ưu trong một tập các bài toán con ([4][4])

**Ví dụ 2.4.4** Tìm cây bao trùm nhỏ nhất của đồ thị liên thông vô hướng có trọng số.

Bài toán này được giải bằng chiến lược tham lam theo Kruskal như sau.

Kruskal(*G*, *w*) // Đồ thị *G* = (*V*, *E*) có trọng số mỗi cạnh *e* là *w*(*e*)

1 *F* ← ∅; *Q* ←*E*[*G*]; *N* ← *V*[*G*]

3 **while** **⏐***F***⏐**<**⏐***N***⏐**-1 and *Q*≠ ∅

4 **do** *e* ← ExtractMin(*Q*) // e có trọng số bé nhất

5 **if** *F*∪{*e*} not contain cycle **then** *F* ← *F*∪{*e*}

6 **if** ⏐*F*⏐<**⏐***N***⏐**-1

7 **then** *G* is not connected

8 **else** **return** *T* // T= (V, F)

Dòng 4 chọn cạnh có trọng số nhỏ nhất bằng thủ tục ExtractMin(*Q*) tại một lần lặp thể hiện chiến lượng tham lam trong giải thuật. Thời gian chạy của ExtractMin(*Q*) là O(log2 *E*). từ đó thời gian chạy của giải thuật này được xác định là O(*V* log2 *E*).

Chiến lược biến đổi để trị



Chiến lược biến đổi để trị (transform-and-conquer) gồm hai tầng (giai đoạn) ([4]):

1. Biến đổi thể hiện (instance-input) của bài toán để có thể dễ giải (hoặc giải hiệu quả) hơn.
2. Giải bài toán trên thể hiện (input) đã được biến đổi.

**Ví dụ 2.4.5** Xét bài toán tính tổng *S*=1+2+…+ *n*, với *n* là một số nguyên dương (trong Ví dụ 2.4.1). Áp dụng chiến lược biến đổi để trị biến đổi tổng *S*=1+2+…+ *n=n*(*n+*1)/2, ta có giải thuật.

Sum(*n*)

1 **return** *n*\*(*n*+1)/2

Rõ ràng độ phức tạp giải thuật là O(1), tốt hơn rất nhiều độ phức tạp O(n) của giải thuật trực tiếp trong Ví dụ 2.4.1).

Chiến lược quay lui



Giải thuật được thiết kế bằng chiến lược quay lui (backtraking) biểu diễn nghiệm bài toán như một vector với được chọn trong tập nào đó. Quá trình tìm được thực hiện bằng cách tính toán mỗi thành phần. tại một thời điểm. Tại bước thứ *i*, vector con tìm được gọi là một nghiệm bộ phận của bài toán. Để tìm thành phần thứ *k* của nghiệm (thỏa một số ràng buộcnào đó), khi đã chọn được thành phần của , giải thuật chọn (tính) trong số các khả năng có thể có của nó trong . Với mỗi khả năng j, giải thuật kiểm tra xem có chấp nhận đươc không. Nếu chấp nhận thì xác định theo j, khi thì có một lời giải, ngược thì tiếp tục xác định (thường bằng đệ qui). Nếu thử tất cả các khả năng mà không có khả năng nào được chấp nhận thì quay lui lại bước trước để xác định lại ([4]). Lược đồ tổng quát cho giải thuật được thiết kế theo chiến lược quay lui.

BackTracking(*x*[1..*k*], *n*) // xác định *x*[*k*], *k* nguyên

1 **for** *j* ←1 **to** *nk* // xét khả năng *j* trong *nk* khả năng

2 **do** **if** accepting *j*

3 **then** <computing *x*[*k*] in *Ak* that subjects to *j*>

4 **if** *k*=*n*

5 **then** < recording 1 solution >

6 **else** BackTracking(*x*[1..*k*+1], *n*)

Chiến lược nhánh cận



Chiến lược quay lui cho phép tìm được nghiệm đúng của bài toán, nhưng do phải thử mọi khả năng nên khi kích thước bài toán lớn sẽ rất kém hiệu quả. Chiến lược nhánh cận (Branch-and-Bound) khắc phục được hạn chế này bằng cách xác định nhánh cận với mục tiêu tại mỗi bước tìm kiếm, vì vậy loại bỏ được hầu hết các hướng tìm kiếm không cần thiết (trong cây không gian tìm kiếm). Chiến lược nhánh cận thường được áp dụng để giải các bài toán tối ưu. Chiến lược nhánh cận dựa trên chiến lược quay lui và một *hàm lượng giá* (estimate function) mục tiêu hướng đến các nhánh cận mục tiêu để tìm lời giải nhanh nhất có thể ([4]). Lược đồ tổng quát cho giải thuật được thiết kế theo chiến lược nhánh cận.

BranchBound(*x*[1..*k*], *n*) // developing local solution (*a*1, *a*2, …., *ak*-1)

1 **for** *ak* ∈ *Ak*

2 **do** **if** accepting *ak*

3 **then** *xk* ← *ak*

4 **if** *k* = *n*

5 **then** < update record>

6 **else if** *g*(*a*1, *a*2, …., *ak*) ≤ *f*\*

7 **then** BranchBound(*x*[1..*k*+1], *n*)

Trong lược đồ này *g*(*x*) là cận dưới (lượng giá) của hàm mục tiêu *f*(*x*) và *f*\* là giá trị nhỏ nhất của hàm mục tiêu hiện tại (gọi là một kỷ lục).

## Cơ sở toán học hỗ trợ tính độ phức tạp của giải thuật

Để tính toán độ phức tạp của giải thuật, cần có một số công cụ toán cần thiết được giới thiệu sau đây

Phần nguyên và một số hệ thức cơ bản

Khái niệm phần nguyên của một số thực được định nghĩa sau đây, thường được áp dụng trong thiết kế và phân tích giải thuật.

**Định nghĩa 2.5.1** Phần nguyên *sàn* (floor) của một số thực x, ký hiệu , là số nguyên lớn nhất nhỏ hơn *x*. Phần nguyên trần (ceiling) của số thực , ký hiệu , là các số nguyên nhỏ nhất lớn hơn *x*.

Một số hệ thức toán học cơ bản được ứng dụng để tính toán độ phức tạp của giải thuật:

1. Giả sử *x* là một số thực, thì
2. Nếu n là số tự nhiên và a, b, c là các số thực dương thì
3. Với mọi thì

( gọi là số điều hòa thứ n - Harmonic Number)

Hệ thức truy hồi

Hệ thức truy hồi **(**recurrence Relation) là cơ sở toán học để tính toán độ phức tạp của giải thuật đệ qui. Phần này giới thiệu khái niệm và một định lý cơ bản về hệ thức truy hồi.

**Định nghĩa 2.5.2** Hệ thức truy hồi đối với dãy là một phương trình có dạng

Ví dụ 2.5.1 Hệ thức truy hồi

Dãy được gọi là nghiệm của hệ thức truy hồi nếu các số hạng của nó thỏa mãn hệ thức truy hồi này. Ví dụ dãy {a*n*} với a*n* =3*n* là nghiệm của hệ thức trong Ví dụ 2.5.1.

**Định nghĩa 2.5.3** Một *hệ thức truy hồi tuyến tính thuần nhất* (linear homogeneous recurrence relation) bậc *k*>0 là một hệ thức truy hồi ([2]) có dạng

Với là các hằng số thực,

Định lý 2.5.1 Giả sử là các hằng số, phương trình có k nghiệm phân biệt. Khi đó dãy {} là nghiệm của hệ thức truy hồi nếu và chỉ nếu với n =0,1,2,… Trong đó là các hằng số.

**Định lý 2.5.2** Giả sử *c*1, *c*2, ..., *ck* là các hằng số, phương trình *rk* - *c*1*rk*-1 - ... -*ck* = 0 có *t* nghiệm *r*1, *r*2, …, *rt* bội tương ứng *m*1, *m*2,,..., *m*t. Khi đó dãy {an} là nghiệm của hệ thức truy hồi *an*= *c*1*an*-1 + *c*2*an*-2 + ... + *ckan*-*k* nếu và chỉ nếu *an*= (α1, 0+ *α*1, 1*n*+…+ *α*1, *m*1-1*nm*1-1)*r*1*n* + (*α*2, 0+ *α*2,1*n* +…+ *α*2, *m*2-1n*m*2-1)*r*2*n* +...+ (*αt*, 0+ *αt*, 1*n*+…+ *αt*, *mt*-1n*mt*-1)*rtn* với *n*= 0, 1, 2, ... Trong đó *αi*, *j*, *i*=0,..*t*, *j*=0…, *mt* -1 là các hằng số.

**Định nghĩa 2.5.4** Một *hệ thức truy hồi tuyến tính không thuần nhất* (linear nonhomogeneous recurrence relation) bậc *k*>0 là một hệ thức truy hồi ([2]) có dạng

với là các hằng số thực, , là một hàm chỉ phụ thuộc vào *n*

Ví dụ 2.5.2 Hệ thức là một hệ thức truy hồi tuyến tính không thuần nhất bậc 1.

Định lý 2.5.4 Nếu là một nghiệm của hệ thức truy hồi tuyến tính không thuần nhất , thì mọi nghiệm của nó có dạng , trong đó là nghiệm của hệ thức truy hồi tuyến tính .

Định lý 2.5.5 Giả sử thỏa mãn hệ thức truy hồi tuyến tính không thuần nhất , với , t là một số nguyên không âm. Khi đó

1. Nếu *s* không phải là nghiệm của phương trình thì (1) có một nghiệm dạng
2. Nếu *s* là một nghiệm bội *m* của phương trình thì (1) có một nghiệm dạng

**Định nghĩa 2.5.5** Hệ thức truy hồi dạng , trong đó a, b là các nguyên dương, được gọi là hệ thức truy hồi chia để trị (divide-and-conquer recurrence relation).

Ví dụ 2.5.3 là một hệ thức truy hồi chia để trị.

Định lý 2.5.6 Giả sử là một hàm tăng thỏa mãn hệ thức truy hồi khi n là số nguyên lớn hơn 1, c là một số thực dương , thì

Định lý 2.5.7 Giả sử là một hàm tăng thỏa mãn hệ thức truy hồi . khi n= là số nguyên lớn hơn 1, c và d là một số thực dương, thì

Các Định lý 2.5.6 và 2.5.7 được áp dụng để tính độ phức tạp của các giải thuật chia để trị.



Lý thuyết xác suất

Phần này giới thiệu một số khái niệm và hệ thức cơ bản nhất về xác suất được ứng dụng để thiết kế và phân tích giải thuật ngẩu nhiên trong các chương sau.



**Định nghĩa 2.5.6** Nếu *S* là một tập khác rỗng (gọi là không gian mẩu) của các *kết cục đồng khả năng* (equally likely outcome) và *E* là một sự kiện (một tập con của *S*), thì xác suất của *E* là *P*(*E*)*=* .

Chúng tôi lưu ý rằng một tập con *E* của *S* gọi là một sự kiện, một phần tử *s*∈*S*, gọi là một sự kiện cơ bản.

**Định nghĩa 2.5.3** Một phân bố xác suất (probability distribution) là một ánh xạ từ không gian mẩu đến tập các số thực *R*, Pr: *S*→ *R*, thỏa các tiên đề sau:

1. Pr(*A*)≥0, ∀*A*⊂*S*.
2. Pr(*S*)=1.
3. Pr(*A*∪*B*)=Pr(*A*)+Pr(*B*) với hai sự kiện bất kỳ *A*, *B* xung khắc, loại trừ lẫn nhau (mutually exclusive).

Từ các tiên đề ta suy ra:

1. Pr(∅)=0
2. Pr

**Định nghĩa 2.5.4** Một phân bố xác suất được gọi là *phân bố rời rạc* (discrete distribution) nếu nó được định nghĩa trên không gian mẩu rời rạc. Nếu *S* là không gian mẩu, thì Pr(*A*)=Σ*s*∈*A* Pr({*s*}), ∀*A*⊂*S*. Nếu Pr({*s*})=1/|*S*| thì ta có *phân bố đều* (uniform probability distribution) trên *S*

Để đơn giản khi sử dụng ký hiệu Pr({*s*}) thường được viết Pr(*s*).

**Ví dụ 2.5.4** Tung một đồng xu như nhau *n* lần, xác suất để đồng xu sấp (S) hoặc ngữa (N) mỗi lần tung là ½. Khi đó chúng ta có một phân bố đều trên không gian *S*={S, N}*n* với 2*n* phần tử. Mỗi sự kiện xác suất là một xâu *n* ký hiệu trong {S, N}. Mỗi xâu xuất hiện với xác suất là 1/2*n*. Xem sự kiện *A* là biến có (xâu) có đúng *k* mặt sấp xuất hiện. Khi đó |*A*|=, nên Pr(*A*)= /|*S*|

**Định nghĩa 2.5.4** Xác suất có điều kiện của sự kiện A khi có sự kiện B được định nghĩa là

Từ định nghĩa trên ta có các hệ thức sau:

1. , nếu A và B độc lập

Một hệ các sự kiện được gọi là đầy đủ nếu chúng đôi một xung khắc và =S.

**Định lý 2.5.8** Nếu các biến cố là một hệ đầy đủ các biến cố. H là một biến cố trong không gian mẩu. Khi đó

Ta có:

**Định lý 2.5.8** (Định lý Bayes)Giả sử A và B là 2 sự kiện trong không gian mẩu, ta có

**Định nghĩa 2.5.5** Một *biến ngẩu nhiên* (random variable) là một hàm *X*: *S* → *R* từ không gian mẩu *S* đến tập các số thực *R*.

Với mỗi biến ngẩu nhiên *X* và số thực *x*, sự kiện *X*=*x* được định nghĩa là {*s*∈*S*: *X*(*s*)=*x*}. Khi đó Pr(*X*=*x*)=Σ*s*∈*S*: *X*(*s*)=*x* Pr(*s*).

**Ví dụ 2.5.5** Nếu ta tung 2 con xúc sắc 6 mặt thì không gian xác suất *S* gồm 36 sự kiện cơ bản. Giả sử phân bố xác suất trên *S* là đều, biến ngẩu nhiên *X* được định nghĩa là gía trị lớn nhất của số trên mặt của 2 con xúc sắc xuất hiện khi tung chúng. Từ định nghĩa ta có Pr(*X*=3) =5/36. Sự kiện *X*=3 tương ứng với {(1; 3), (2; 3), (3; 3), (3; 2), (3; 1)}.

**Định nghĩa 2.5.5** *Giá trị kỳ vọng* (expected value) của biến ngẩu nhiên X được định nghĩa là

*E*(*X*)=Σ*x*∈*R* *x*.Pr(*X*=*x*)

Từ định nghĩa chúng ta suy ra rằng *E*(*X*)=Σ*s*∈*S* Pr(*s*).*X*(*s*). Ngoài ra, chúng ta cũng nhận xét rằng giá trị kỳ vọng của biến *X* cũng là giá trị trung bình của nó. Vì vậy, kỳ vọng là công cụ toán học giúp tính toán độ phức tạp trung bình của giải thuật.

**Ví dụ 2.5.6** Giả sử *X* là biến ngẩu nhiên nhận giá trị là số trên mặt của con xúc sắc xuất hiện khi chúng ta tung nó. Tính kỳ vọng của *X*.

Có 6 khả năng xảy ra, biến *X* nhận giá trị tương ứng là 1, 2,…, 6 với xác suất là 1/6 cho mỗi khả năng. Vì vậy *E*(*X*)=1/6(1+2+…+6)= 21/6 là số trung bình trên mặt con xúc sắc xuất hiện khi tung nó.

## Kết luận

Trong chương 2 này, các khái niệm cơ bản nhất về giải thuật, độ phức tạp, cũng như các chiến lược thiết kế giải thuật thông dụng nhất đã được trình bày làm cơ sở để xây dựng và phát triển các giải thuật hiệu quả. Một số công cụ toán học như hệ thức truy hồi làm phương tiện để tính độ phức tạp các giải thuật đệ qui. Các công cụ về lý thuyết xác suất là cơ sở để xây dựng và phân tích các giải thuật ngẩu nhiên trong chương 3 tiếp theo.

# Chương 3

# ỨNG DỤNG

**GIẢI THUẬT NGẪU NHIÊN**



## Giới thiệu

Chương này trình về giải thuật ngẫu nhiên, phân loại và các lĩnh vực áp dụng của giải thuật ngẫu nhiên.

Đầu tiên, Phần 3.2 trình bày định nghĩa của giải thuật ngẫu nhiên. Kế đến, Phần 3.3 trình bày về hai loại của giải thuật ngẫu nhiên đó là giải thuật Monte Carlo và giải thuật Las Vegas. Tiếp đó, Phần 3.4 trình bày các lĩnh vực áp dụng của giải thuật ngẫu nhiên như lý thuyết số, cấu trúc dữ liệu,…Cuối cùng, Phần 3.5 là kết luận của Chương này.

## Định nghĩa và ví dụ giải thuật ngẫu nhiên

Trong lịch sử, giải thuật ngẫu nhiên đầu tiên là một phương pháp được phát triển bởi Rabin năm ? [] được dùng cho bài toán tìm cặp điểm gần nhất trong hình học. Việc nghiên cứu các giải thuật ngẫu nhiên được thúc đẩy bởi việc khám phá kiểm tra số nguyên tố ngẫu nhiên vào năm 1977 (tức là xác định xem số đó là số nguyên tố hay hợp số) bởi Solovay and Strassen năm ? [] . Ngay sau đó, Rabin năm [] đã chứng minh rằng việc kiểm tra số nguyên tố của Miller năm 1976 có thể được chuyển thành một giải thuật ngẫu nhiên. ([9])

Trong các giải thuật truyền thống, mỗi bước trong giải thuật đều được xác định. Với giải thuật ngẫu nhiên thì chúng ta có thể không xác định được bước tiếp theo, vì trong quá trình thực hiện một giải thuật ngẫu nhiên nó sẽ đưa ra một chọn lựa tùy ý. Một số hành động được thực hiện một cách ngẫu nhiên. Hình 3.2.1 và 3.2.2 dưới đây là sự mô tả sự khác biệt giữa giải thuật xác định và giải thuật ngẫu nhiên.

**INPUT**

**ALGORITHM**

**OUTPUT**

**Hình 3.2.1 Sơ đồ giải thuật xác định**

**INPUT**

**ALGORITHM**

**OUTPUT**

**RANDOM NUMBERS**

**Hình 3.2.2 Sơ đồ giải thuật ngẫu nhiên**

Ngoài dữ liệu đầu vào, giải thuật còn dùng một nguồn số ngẫu nhiên và thực hiện các chọn lựa ngẫu nhiên trong quá trình thực hiện. Kết quả có thể khác nhau ngay cả khi thực hiện trên một đầu vào nhất định.

Định nghĩa 3.2.1 *Giải thuật ngẫu nhiên* (randomized algorithm) là một giải thuật mà ngoài dữ liệu đầu vào nó còn nhận một dãy số ngẩu nhiên nhằm mục đích đưa ra các lựa chọn ngẫu nhiên.

Ngay cả đối với đầu vào là cố định, những lần chạy khác nhau của giải thuật ngẫu nhiên thì đưa ra các kết quả khác nhau. Ví dụ, với đầu vào dữ liệu cố định, thời gian thực hiện của một giải thuật ngẫu nhiên có thể là một giá trị ngẫu nhiên ([1]).

Ví dụ 3.2.1 Bài toán tìm ‘a’ trong mảng. Cho một dãy phần tử, trong đó một nửa là *a* và nửa còn lại là *b*. Tìm *a* trong mảng.

Có thể phát triển hai giải thuật ngẫu nhiên để giải bài toán này như dưới đây ([9]).

Findinga\_lv(Array *a*,*n,k*)

1. **for** i=1 **to** k **do**
2. Randomly select one element out of *n* elements
3. **if** ‘a’ is found
4. **then return**

Nếu *a* được tìm thấy, giải thuật thành công, không thì giải thuật thất bại. Sau khi lặp lại *k* lần, xác suất tìm thấy *a* là:

Giải thuật này không bảo đảm thành công nhưng thời gian chạy là cố định. Sự chọn lựa thì được thực hiện chính xác *k* lần, do dó thời gian chạy là

Findinga\_lv(Array *a*,*n*)

* 1. **while** (‘a’ is not found)
  2. Randomly select one element out of *n* elements.

Giải thuật này thành công với xác suất là 1. Thời gian chạy là ngẫu nhiên (có thể lớn tùy ý) nhưng kỳ vọng (thời gian chạy trung bình) của nó bị chặn trên bởi .

## Phân loại các giải thuật ngẫu nhiên

Như đã nêu trong Ví dụ 3.2.1, chúng ta có thể giải một bài toán bằng hai giải thuật ngẩu nhiên. Một giải thuật có thời chạy luôn luôn cố định nhưng chỉ thành công với một xác suất nào đó. Giải thuật còn lại chắc chắn thành công nhưng thời gian chạy không xác định. Các giải thuật này tương ứng thuộc về lớp giải thuật Monte Carlo và Las Vegas. Một giải thuật ngẩu nhiên thuộc về một về một trong hai lớp này. Các lớp giải thuật Monte Carlo và Las Vegas lần lượt được định nghĩa như sau.

Định nghĩa 3.3.1 Một Giải thuật Monte Carlo là một giải thuật ngẫu nhiên có thời gian chạy luôn luôn xác định nhưng kết quả có thể sai với một xác suất nào đó.

Trong lĩnh vực tính toán, một giải thuật Monte Carlo là một giải thuật ngẫu nhiên mà có độ phức tạp xác định, nhưng đầu ra của nó có thể không chính xác với xác suất nhất định (thường là nhỏ). Đối với các bài toán đưa ra quyết định, một giải thuật có thể được phân loại *lệch về false* (false-biased) hoặc *lệch về true* (true-biased). Một giải thuật Monte Carlo lệch về *false* thì luôn đưa ra kết quả chính xác khi nó trả về giá trị *false*, một giải thuật lệch về *true* thì luôn chính xác khi nó trả về *true,* và được gọi chung là có lỗi một chiều. Những giải thuật khác không có thiên hướng bị lệch, chúng được gọi là giải thuật có các lỗi hai chiều. Câu trả lời chúng đưa ra (cả *true* và *false*) sẽ là không chính xác, hoặc chính xác với một xác suất có giới hạn.

Định nghĩa 3.2.2 Một giải thuật Las Vegas là một giải thuật ngẫu nhiên luôn đưa ra kết quả đúng, nhưng thời gian chạy là một giá trị ngẩu nhiên ở mỗi lần thực hiện khác nhau.

Trong lĩnh vực tính toán, một giải thuật Las Vegas là một giải thuật ngẫu nhiên, mà nó luôn đưa ra các kết quả chính xác. Nghĩa là, luôn sinh ra kết quả chính xác hoặc nó thông báo cho biết thất bại. Nói cách khác giải thuật Las Vegas không đánh cược với tính chân thực của kết quả, nó đánh cược với các tài nguyên (thời gian, bộ nhớ, bộ xử lí…) được dùng để tính toán. Một ví dụ đơn giản là giải thuật *randomized quicksort*, có chốt được chọn ngẫu nhiên, nhưng kết quả là dãy luôn là được sắp xếp.

Một giải thuật Monte Carlo có thể được chuyển thành giải thuật Las Vegas khi mà tồn tại một thủ tục để xác định output được sinh ra bởi giải thuật chắc chắn là chính xác.

Giải thuật Las Vegas được giới thiệu bởi Babai năm 1979, trong bài toán đẳng cấu đồ thị. Giải thuật Las Vegas có thể được dùng trong các tình huống ở đó số lượng các lời giải khả thi tương đối hạn chế, việc kiểm tra tính đúng đắn của các lời giải được đề cử tương đối dễ dàng trong khi việc tính toán các lời giải lại thực sự phức tạp.

## Các lĩnh vực áp dụng của giải thuật ngẫu nhiên

Giải thuật ngẩu nhiên được ứng dụng trong hầu hết các lĩnh vực của khoa học tính toán. Giải thuật ngẫu nhiên và xác suất đóng vai trò quan trọng trong khoa học máy tính, với những ứng dụng từ tối ưu tổ hợp, lý thuyết số, hình học, lý thuyết đồ thị,.., cho đến máy học, mạng máy tính và các giao thức bảo mật. Cụ thể về một số bài toán có thể giải bằng các giải thuật ngẩu nhiên trong một số lĩnh vực như sau ([6]):

* Lý thuyết số: bài toán kiểm tra số nguyên tố, tìm nghiệm phương trình đồng dư.
* Đại số: bài toán xác định tính đồng nhất của các ma trận và đa thức, hệ thống chứng minh tương tác.
* Lập trình toán: bài toán lập trình tuyến tính, làm tròn số.
* Tìm kiếm và sắp xếp: bài toán kiểm tra tính bằng nhau của hai xâu ký tự có độ dài lớn, so trùng mẩu, bài toán sắp xếp một dãy số (Quicksort).
* Lý thuyết đồ thị: bài toán tìm cây bao trùm nhỏ nhất, đường đi ngắn nhất, lát cắt nhỏ nhất.
* Hình học: bài toán tìm cặp điểm gần nhau nhất.
* Tối ưu tổ hợp: bài toán liệt kê tổ hợp, tính số phần tử của hợp các tập hợp, bài toán gép cặp cực đại.

## Kết luận

Trong Chương 3 này, các khái niệm cơ bản về giải thuật ngẩu nhiên và sự phân loại của chúng đã được giới thiệu. Đó là các giải thuật có thể không xác định kết quả duy nhất tại mỗi bước thực thi và kết quả tính toán mà giải thuật trả về cũng có thể không đúng với một xác suất nào đó. Tuy nhiên, giải thuật ngẩu nhiên thường có độ phức tạp về thời gian nhỏ hơn giải thuật truyền thống, đặc biệt là độ phức tạp trung bình. Vì vậy, giải thuật ngẩu nhiên được ứng dụng nhiều trong thực tế để cải thiện hiệu suất tính toán. Chương 4 kế tiếp, một số giải thuật ngẩu nhiên tiêu biểu sẽ được giới thiệu cho thấy khả năng ứng dụng hiệu quả của chúng để giải các bài toán thực tế.

# Chương 4

# GIẢI THUẬT NGẪU NHIÊN CHO MỘT SỐ BÀI TOÁN



## Giới thiệu

Chương này trình việc ứng dụng các giải thuật ngẫu nhiên cho một số bài toán cụ thể.

Đầu tiên, Phần 4.2 trình bày bài toán sắp xếp (quicksort). Kế đến, Phần 4.3 trình bày bài toán xác định số nguyên tố. Tiếp theo, Phần 4.4 trình bày bài toán tìm cặp điểm gần nhất. Sau đó, Phần 4.5 trình bày bài toán kiểm tra phép nhân ma trận. Phần 4.6 trình bày bài toán tìm cây bao trùm nhỏ nhất của đồ thị. Cuối cùng, Phần 4.7 là kết luận của Chương này.

## Bài toán sắp xếp

Giải thuật *quicksort* áp dụng chiến lược chia để trị để sắp xếp và có độ phức tạp thuật toán theo thời gian trong trường hợp xấu nhất là , với *n* là số phần tử. Mặc dù độ phức tạp về thời gian trong trường hợp xấu nhất của giải thuật chậm nhưng *quicksort* vẫn thường là sự chọn lựa thực tế nhất cho việc sắp xếp bởi hiệu quả đáng kể ở trường hợp trung bình: độ phức tạp về thời gian được mong chờ của giải thuật là .

Trong phiên bản giải thuật ngẫu nhiên của *quicksort*, chúng ta sẽ chọn ngẫu nhiên một phần tử trong mảng *A*[*p…r*] để làm phần tử chốt thay vì dùng *A*[*r*] như ở *quicksort* cổ điển. Bằng cách chọn giá trị phần tử chốt ngẫu nhiên, chúng ta có thể mong chờ sự phân chia của mảng input được cân bằng khá tốt trên mức trung bình.

Hàm Quicksort và hàm Partition chỉ có một chút thay đổi nhỏ, đó là chúng ta sẽ hoán đổi giá trị của phần tử được chọn ngẫu nhiên và giá trị của *A*[*r*] trước khi phân vùng.

Mã giả

Randomized-Partition (*A,p,r*)

1. *i*= Random (*p,r*)
2. exchange *A*[*r*] with *A*[*i*]
3. **return** Partition(*A,p,r*)

Hàm Randomized-Partition được thay cho hàm Partition trong giải thuật *quicksort*

Random-Quicksort(*A,p,r*)

1. **if** *p*<*r*
2. *q*= Randomized-Partition (*A,p,r*)
3. Randomized-Quicksort(*A,p,q-1*)
4. Randomized-Quicksort(*A,q+1,r*)

Phân tích độ phức tạp

Thời gian chạy mong chờ của hàm Random-Quicksort là và ở đây chúng ta giả định các phần tử được sắp xếp là riêng biệt vì với đầu vào như thế sẽ không tạo nên trường hợp xấu nhất. Nếu, trong mỗi cấp đệ quy, sự phân chia được thực hiện bởi hàm Randomized-Partition đưa phần không đổi bất kỳ trong các phần tử về một phía của mảnh, thì cây đệ quy có độ sâu là , và được thực hiện ở mỗi cấp. Ngay cả khi chúng ta thêm một vài cấp mới với sự phân chia không cân bằng nhất giữa các cấp thì tổng thời gian vẫn duy trì ở .

Hàm Quicksort và Randomized-Quicksort chỉ khác nhau ở cách chọn phần tử chốt, còn ở tất cả các khía cạnh khác thì đều giống nhau. Do đó, chúng tôi sẽ trình bày phân tích của chúng tôi về Randomized-Quicksort thông qua việc trình bày về hai tiến trình Quicksort và Partition, nhưng với giả định phần tử chốt được chọn ngẫu nhiên từ mảng con thông qua hàm Randomized-Partition.

Quicksort(*A,p,r*)

1. **if** *p<r*
2. q=Partition(*A,p,r*)
3. Quicksort(*A,p,q-1*)
4. Quicksort(*A,q+1,r*)

Partition(*A,p,r*)

1. **for** **to**
2. **if**
3. exchange *A*[*i*] with *A*[*j*]
4. exchange *A*[*i+1*] with *A*[*r*]
5. **return** *i+1*

Thời gian chạy của hàm Quicksort bị chi phối bởi thời gian chạy trong hàm Partition. Mỗi lần hàm Partition được gọi, nó chọn một phần tử chốt, và phần tử này sẽ không có trong các lần gọi đệ quy hàm Quicksort và Partition sau đó. Do đó, có thể có nhiều nhất *n* lần gọi hàm Partition trong quá trình chạy của giải thuật *quicksort*. Một lần gọi hàm Partition có cộng với giá trị thời gian tỉ lệ thuận với vòng lặp **for** được thực hiện từ dòng 3 đến dòng 6. Phép toán cơ bản của vòng lặp **for** ở đâylà phép so sánh ở dòng 4, so sánh phần tử chốt và các phần tử khác của mảng A. Do đó, nếu chúng ta có thể đếm được tổng số lần thực hiện dòng 4, chúng ta có thể ràng buộc được thời gian thực hiện vòng lặp trong suốt thời gian thực hiện hàm Quicksort.

Bổ đề 4.2.1: Đặt X là số phép so sánh thực hiện ở dòng 4 của hàm Partition trong suốt thời gian thực hiện hàm Quicksort trên một mảng *n* phần tử. Thì thời gian chạy của Quicksort là .

**Chứng minh:** Bằng mô tả trên, giải thuật gần như gọi hàm Partition *n* lần, trong mỗi lần gọi đều thực hiện số công việc không đổi và sau đó thực hiện vòng lặp **for** vài lần. Trong mỗi lần lặp của vòng lặp **for** đều thực hiện câu lệnh ở dòng 4.

Do đó, mục tiêu của chúng tôi là tính *X*, tổng số lần phép so sánh được gọi trong hàm Partition. Chúng tôi sẽ không cố gắng phân tích có bao nhiêu phép so sánh được thực hiện trong mỗi lần gọi hàm Partition. Thay vào đó, chúng tôi sẽ lấy một ràng buộc tổng thể trên số lượng phép so sánh. Để làm như vậy, chúng ta phải biết rằng khi nào giải thuật so sánh hai phần tử của mảng và khi nào thì không. Để dễ phân tích, chúng tôi đổi tên các phần tử của mảng A thành *z1,z2,…,zn* với *zi*  là phần tử nhỏ nhất thứ *i*. Chúng tôi cũng đặt tập *Zij=*{*zi, zi+1,…,zj*} là tập phần tử giữa *zi* và *zj* và gồm cả *zi, zj*.

Khi nào giải thuật so sánh *zi* và *zj*? Để trả lời câu hỏi này, trước hết chúng ta hãy quan sát mỗi cặp phần tử chỉ so sánh một lần. Tại sao? Các phần tử chỉ được so sánh với phần tử chốt và sau khi hàm Partition được hoàn thành, phần tử chốt trong lần gọi vừa rồi sẽ không được so sánh với bất cứ phần tử khác ở các lần đệ qui sau đó.

Phân tích của chúng tôi sử dụng chỉ số biến ngẫu nhiên. Đặt *Xij=I*{*zi* so sánh với *zj*}, ở đây chúng ta đang xem xét liệu có phải phép so sánh xảy ra bất cứ lúc nào trong suốt quá trình thực hiện giải thuật mà không phải chỉ trong một vòng lặp hoặc một lần gọi hàm Partition. Bởi vì mỗi cặp phần tử chỉ được so sánh một lần, chúng ta có thể mô tả tổng số phép so sánh được thực hiện bởi giải thuật là: .

Đầu tiên, chúng ta hãy xem xét khi nào hai phần tử không so sánh với nhau. Giả sử input là mảng số từ 1 đến 10 (sắp tùy ý), và phần tử đầu tiên chọn làm chốt là 7. Thì sau lần gọi hàm Partition đầu tiên, dãy số được phân thành hai tập {1, 2, 3, 4, 5, 6} và {8, 9, 10}. Trong quá trình thực hiện, số 7 được so sánh với các phần tử còn lại nhưng không có phần tử nào ở tập thứ nhất (chẳng hạn số 2) được hoặc sẽ được so sánh với bất kỳ phần tử nào ở tập thứ hai (chẳng hạn số 9).

Nói chung, bởi chúng ta giả định giá trị các phần tử là phân biệt, một phần tử chốt *x* được chọn với , chúng ta biết rằng *xi* và *xj* không thể được so sánh ở các lần kế tiếp. Nếu *zi* được chọn làm chốt trước bất kì phần tử nào trong *Zij*, thì *zi* sẽ được so sánh với từng phần tử trong *Zij*, trừ chính nó, và cũng tương tự với *zj*. Trong ví dụ của chúng ta, giá trị 7 và 9 được so sánh với nhau vì số 7 là phần tử đầu tiên được chọn từ *Z7,9*. Ngược lại, 2 và 9 sẽ không được so sánh với nhau bởi vì phần tử đầu tiên được làm chốt từ *Z2,9* là 7. Do đó, *zi* và *zj* được so sánh với nhau khi và chỉ khi phần tử đầu tiên được chọn làm chốt từ *Zi,j* là *zi* và *zj*.

Bây giờ, chúng ta tính xác suất để điều này xảy ra. Trước hết, bất cứ phần tử nào trong *Zij* cũng có thể là phần tử đầu tiên được chọn làm chốt. Vì vậy, tập *Zij* có phần tử, và bởi vì chốt được chọn ngẫu nhiên và độc lập, xác suất để một phần tử trở thành phần tử đầu tiên được chọn làm chốt là . Do đó, chúng ta có:

Ta có: (vì )

Bằng cách thay biến () và sử dụng Harmonic Number (xem phần 2.5), chúng ta có thể đánh giá tổng trên như sau:

Do đó, chúng ta có thể kết luận rằng, khi sử dụng Randomized-Partition, thời gian chạy mong chờ của *quicksort* là trong trường hợp giá trị của các phần tử là phân biệt.[8]

## Bài toán xác định số nguyên tố

Bài toán này trình bày vấn đề xác định một số có phải là số nguyên tố hay không (kiểm tra tính nguyên tố).( Số nguyên tố là số chỉ chia hết cho 1 và chính nó). Với một số nguyên nhỏ n thì dễ dàng để xác định tính nguyên tố của nó bằng cách kiểm tra xem nó có chia hết cho các số từ 2 đến hay không? Với một số lớn thì phương pháp này không khả thi. Thay vào đó chúng ta chỉ có thể kiểm tra tính nguyên tố theo một xác suất đúng nhất định. Số nào là được kết luận là số nguyên tố theo cách kiểm tra này được gọi là số *giả nguyên tố* (pseudoprime)- nghĩa là số này có thể không phải là số nguyên tố, mà số này là số nguyên tố với một xác suất nhất định

Ở đây chúng tôi sẽ giới thiệu một giải thuật ngẫu nhiên giải quyết vấn đề này dựa vào định lý Fermat.

Ý tưởng

Giải thuật này thực hiện *m* lần kiểm tra. Nếu có bất kì lần kiểm tra nào thành công, chúng ta kết luận rằng số này là hợp số, và chúng ta tin chắc rằng kết luận này là hoàn toàn đúng. Mặt khác, nếu toàn bộ *m* lần kiểm tra đều thất bại thì chúng ta kết luận số này là số nguyên tố. Tuy nhiên, với trường hợp này chúng ta không thể tin chắc đây là một kết luận đúng. Kết luận này đúng với xác suất . Vì vậy, nếu *m* lớn, thì sẽ tiến gần về 1 hơn, khi đó giải thuật cho ta một kết quả có độ tin cậy cao.

Các lần kiểm tra bên trên dựa vào định lý Fermat dưới đây.

Định lý 1 : Định lý Fermat Error! Reference source not found.

Nếu *n* là một số nguyên tố, thì với mọi phải thỏa mãn ( tương đương )

Như vậy, nếu một số không thỏa mãn định lý Fermat thì số này chắc chắn là hợp số, nhưng một số thỏa mãn định lý Fermat không chắc là số nguyên tố vì tồn tại số Carmichael **Error! Reference source not found.** - là những hợp số thỏa mãn tính chất của định lý Fermat

( số Carmichael là những hợp số n, mà với mọi , )

Mã giả

**Input**: Một số nguyên *n* >1, và một tham số *m*, với .

**Output**: *n* là số nguyên tố hay hợp số, với xác suất đúng là

KiemTraSoNguyenTo(*n,m*)

1. **if** *n* **mod** 2 = 0
2. **then** **return** **false**
3. **else**
4. *k*=(*n*-1)/2
5. **for** *i*=0 **to** *m* **do**
6. *a*= randomly choose a numbers *ai*, 1<*ai*<*n*
7. *kq*= Pow(*a*,*n*) mod *n*
8. **if** *kq*!=*a*
9. **then return** **false**
10. **return true**

Hàm Pow là hàm tính *a* mũ *n* (

Khi thủ tục trên trả về false thì số *n* là hợp số, ngược lại nếu return true thì n là số nguyên tố.

Ví dụ 4.3.1

1. n=12, m=3,

Lần kiểm tra thứ nhất a1=9, tiếp tục kiểm tra lần thứ 2

Lần kiểm tra thứ hai a2=2, return false, vậy 12 là hợp số

1. n=13, m=3

Lần kiểm tra thứ nhất *a*1=9,  tiếp tục kiểm tra lần thứ 2

Lần kiểm tra thứ hai *a*1=10,  tiếp tục kiểm tra lần thứ 3

Lần kiểm tra thứ ba *a*1=8,

Cả ba lần kiểm tra đều thất bại nến ta kết luận 13 là số nguyên tố.

Phân tích độ phức tạp

Gọi *t* là độ phức tạp của hàm Pow(*a,n*) thì ta có độ phức tạp của toàn bộ giải thuật là O(*k.t*).

Hàm Pow(*a,n*) được viết theo phương pháp chia để trị như sau:

Pow(*a,n*)

1. **if** *n* =0
2. **then return** 1;
3. *u* = Pow(a, );
4. **if** *n* mod 2 = 1
5. **then** **return** *u \* u \* a*;
6. **else**
7. **return** *u* *\* u*;

Độ phức tạp của hàm Pow là

Áp dụng định lý 2.5.6 ta có

Như vậy độ phức tạp của toàn bộ giải thuật là , so với độ phức tạp của giải thuật cổ điển kiểm tra số nguyên tố là . Với *k* =10 thì giải thuật ngẫu nhiên cho kết quả đúng với xác xuất cao, và có độ phức tạp thấp hơn nhiều so với giải thuật cổ điển khi *n* lớn.

## Bài toán tìm cặp điểm gần nhất

Bài toán tìm cặp điểm gần nhất có thể giải quyết bằng giải thuật chia để trị với độ phức tạp là . Trong phần này, chúng tôi sẽ chỉ ra một giải thuật ngẫu nhiên và độ phức tạp của giải thuật để giải quyết bài toán tìm cặp điểm gần nhất là .

Gọi *x1,x2,….xn* là *n* điểm trên mặt phẳng. Yêu cầu của bài toán là tìm cặp điểm *xi* và *xj* mà khoảng cách giữa *xi* và *xj*là nhỏ nhất trong toàn bộ các cặp điểm. Với phương pháp tiếp cận trực tiếp dùng để giải bài thì theo tính toán có tất cả *n(n-1)/2* khoảng cách sau đó tìm con số nhỏ nhất trong tất cả khoảng cách này.

Ý tưởng chính của giải thuật ngẫu nhiên là dựa trên sự quan sát sau: Nếu hai điểm *xi* và *x­j* có khoảng cách giữa chúng, và khoảng cách giữa chúng có vẻ như không là ngắn nhất và do đó tốt hơn hết là bỏ qua. Với ý tưởng này, giải thuật ngẫu nhiên trước hết là phân vùng các điểm thành một vài cụm theo một cách nào đó mà ở mỗi cụm có các điểm gần nhau. Sau đó chúng ta tính toán khoảng cách giữa các điểm trong cùng một cụm.

Xét các điểm trong Hình 4.4.1:

Hình 4.4.1

Nếu chúng ta chia vùng cho sáu điểm ở Hình 4.4.1 thành ba cụm *S1={x1,x2}, S2={x3,x4}, S1={x5,x6}*, sau đó chúng ta chỉ tính toán ba khoảng cách đặt tên là *d(x1,x2)*, *d(x2,x4), d(x5,x6)*, sau cùng chúng ta tìm giá trị nhỏ nhất trong ba khoảng cách đó. Ngược lại, nếu chúng ta không chia các điểm thành các cụm, số khoảng cách phải tính là khoảng cách.

Dĩ nhiên, mô tả này thì hơi không thực tế bởi không có gì đảm bảo giải thuật này hoạt động. Trong thực tế, giải thuật này có vẻ giống giải thuật chia để trị. Nhưng nó có một quá trình phân chia nhưng không có quá trình hợp nhất. Xét Hình 4.4.2:

Hình 4.4.2

Chúng ta có thể thấy cặp điểm gần nhất trong Hình 4.4.2 là *{x1,x3}*. Nhưng, chúng ta đã chia hai điểm này vào hai cụm khác nhau. Vì thế, nếu chúng ta chia không gian này thành các hình vuông với độ dài cạnh *δ* không nhỏ hơn khoảng cách ngắn nhất, thì sau khi chúng ta tính hết các khoảng cách ở từng vùng, chúng ta phải nhân đôi độ dài cạnh và tạo ra các hình vuông lớn hơn. Các trường hợp mở rộng điển hình được minh họa ở Hình 4.4.3

Hình 4.4.3 Các dạng mở rộng khi nhân đôi *δ*

Cuối cùng, câu hỏi quan trọng là tìm ra mạng lưới kích thước *δ* thích hợp. Nếu *δ* quá lớn, hình vuông ban đầu sẽ quá lớn và số lượng khoảng cách cần tính cũng lớn. Trong thực tế, nếu *δ* quá lớn, hầu như không có việc chia ra và bài toán của chúng ta trở về bài toán ban đầu. Mặt khác, *δ* cũng không thể quá nhỏ bởi vì nó không thể nhỏ hơn khoảng cách ngắn nhất. Trong giải thuật ngẫu nhiên, chúng ta chọn lựa ngẫu nhiên một tập con các điểm và tìm khoảng cách ngắn nhắn trong tập đó. Khoảng cách ngắn nhất này sẽ trở thành *δ* của chúng ta.

Giải thuật

**Input:** Một tập *S* gồm *n* điểm *x1, x2,…, xn* với .

**Output:** Cặp điểm gần nhất trong tập R

Bước 1: Chọn ngẫu nhiên một tập điểm *S1={xi1, xi2,…, xim}* với . Tìm cặp điểm gần nhất trong tập *S1* và gán giá trị khoảng cách giữa cặp điểm đó cho *δ*.

Bước 2: Xây dựng tập hình vuông *T* có độ dài cạnh là *δ*.

Bước 3: Xây dựng bốn tập hình vuông *T1, T2, T3 và T­­­4* bắt nguồn từ *T* bằng cách nhân đôi kích thước cạnh hình vuông thành 2*δ*.

Bước 4: Đối với mỗi *Ti*, xác định các tập điểm là giao giữa tập điểm S và các hình vuông của , sao cho

Bước 5: Đối với mỗi *x­p,xq Sj(i)*, tính *d(xp, xq)*. Đặt *xa* và *xb* là cặp điểm có khoảng cách ngắn nhất trong tất cả các cặp điểm. Trả về *xa* và *xb* là cặp điểm gần nhất.

Ví dụ 4.4.1

Cho tập điểm S có 27 điểm, được mô tả trong Hình 4.4.4. Bước 1, chúng ta chọn ngẫu nhiên điểm, được đánh dấu là *x1, x2,…, x9*. Nhìn vào *Hình 4.4.4*, ta có thể thấy rằng cặp điểm gần nhau nhất là *(x1, x2)*. Gán khoảng cách giữa *x1* và *x2* cho *δ* và xây dựng tập hình vuông *T*, ở ví dụ này sẽ gồm 36 hình vuông, theo yêu cầu của bước 2. Sau đó xây dựng các tập hình vuông *T1, T2, T3, T4* như yêu cầu ở bước 3:

.

.

.

.

Hình 4.4.4 Minh họa ví dụ 4.4.1

Tổng số khoảng cách giữa các cặp điểm cần phải tính lúc này là:

.

.

.

.

Trong cặp điểm trên, cặp điểm gần nhất được tìm thấy tại ô vuông .

Phân tích độ phức tạp giải thuật

Trong bài toán tìm cặp điểm gần nhất ngẫu nhiên được trình bày ở trên, bước đầu tiên là tìm cặp điểm gần nhất trong điểm. Khoảng các ngắn nhất này có thể được tìm bằng cách đệ quy giải thuật ngẫu nhiên này một lần. Nghĩa là, chọn ngẫu nhiên từ điểm, sau đó dùng chiến lược trực tiếp để tìm cặp điểm gần nhất trong điểm, với số phép tính khoảng cách cần thực hiện là .

Bước 2 và bước 3 có thể hoàn thành với *O(n)*. Ở bước 4, ta có thể sử dụng công nghệ *hash* (hashing technique- kỹ thuật băm là một phương pháp hiệu quả để tìm chính xác mục dữ liệu trong thời gian ngắn).Sử dụng kỹ thuật này chúng ta có thể xác định xem điểm nào đang ở trong hình vuông nào. Điều này có nghĩa là bước 4 có thể thực hiện với độ phức tạp *O(n)*.

Số phép tính khoảng cách ở bước 5 không dễ dàng để xác định. Thực sự là chúng ta không có một công thức nào để tính số phép tính này. Thay vào đó chúng tôi sẽ đưa ra một xác suất để tính số khoảng cách này là . Như vậy xác suất để bước 5 có độ phức tạp *O(n)*  là rất cao.

Tại sao chúng ta có thể đưa ra kết luận nói trên? Chúng ta hãy xem xét trường hợp giải thuật ngẫu nhiên này với ô vuông có độ dài cạnh là . Gọi việc phân chia các ô vuông là *T* và số phép tính khoảng các cho việc phân chia này là *N*(*T*). Ngoài ra tồn tại một phép phân chia đặc biệt ( chúng tôi sẽ đề cập sau), chúng ta đặt là , các ô vuông được phân chia có kích thước là với 2 tính chất như sau:

1. Xác suất để là là rất cao

Giả sử, chúng ta gấp 4 kích thước ô vuông từ thành thì sẽ gồm 16 bộ ô vuông. Chúng ta gọi các ô vuông này là *T*i (*T*1, *T*2,… *T*16)

Từ đây xác suất là , xác suất để mỗi ô trong *T* trùng với ít nhất một ô trong *T*i là . Gọi số phép tính khoảng cách cho việc phân chia *T*i là *N*(*Ti*). Và xác suất để là . Bây giờ chúng ta tính *N*(*T*). Chúng ta sẽ bắt đầu từ *N*(*Ti*). Mỗi ô vuông trong *T*i lớn hơn 16 lần một ô vuông *T*0. Giả sử ô vuông có số phần tử nhiều nhất trong *T*0 có *k* phần tử. Giả sử ô vuông trong Sij có số phần tử lớn nhất là *kij* phần tử. Tổng số phép tính khoảng cách trong *T0* lớn hơn. Đó là, . Và, số phép tính khoảng cách trong ô vuông thứ *j* của tập Ti là nhỏ hơn Vì thế, , với *Ci* là hằng số. Vì vậy, với xác suất . Khi *n* lớn, với xác suất bằng 1, sẽ nhanh chóng tiến về 0.

Chúng ta sẽ giải thích tại sao chúng ta nên kết luận rằng tồn tại một vùng có hai tính chất trên. Lý do khá phức tạp. Trước khi đưa ra lí do chính chúng ta sẽ xác định một số thuật ngữ

Gọi *D* là một vùng trong tập điểm. Đó là, và nếu . Nếu là một sự lựa chọn của *m* điểm thì ta gọi *T* là một vùng thành công trên *D* nếu có ít nhất 2 điểm của *T* được chọn từ cùng một phần *Si* của vùng đối với một vài *i*. Nếu *D’* là một vùng khác của *S*, chúng ta nói rằng *D* bao phủ *D’* nếu đối với mỗi điểm *m*, xác suất thành công trên *D* với sự chọn lựa của *m* điểm thì lớn hơn hoặc bằng sự thành công trên *D’* với sự chọn lựa của *m* điểm.

Dựa vào sự định nghĩa trên, có thể dễ dàng thấy rằng các mệnh đề sau là đúng:

1. Vùng (2,2,2) bao phủ (3,1,1,1) . Điều này có nghĩa là bất kì sự phân chia của một tập có 6 điểm vào ba cặp bao phủ bất kì phân vùng của một tập tương tự thành một bộ ba hoặc ba điểm đơn lẻ. Tại sao nó bao phủ? Lý do thì khá dễ hiểu đối với vùng (2,2,2) có một xác suất để 2 điểm được vẽ từ cùng hình vuông thì lớn hơn vùng (3,1,1,1) bởi vì chỉ có một hình vuông với số điểm lớn hơn 1.
2. Vùng (3,3) bao phủ (4,1,1)
3. Vùng (4,4) bao phủ (5,1,1,1)
4. Vùng (p,q), , bao phủ (l,1,1…,1) với số lượng số 1 là .

Có thể thấy rằng nếu vùng *D* bao phủ vùng *D’*, thì số lượng phép tính khoảng cách trong *D’* bé hơn hoặc bằng số lượng có trong *D*.

Gọi *N(D)* là số lượng phép tính khoảng cách được yêu cầu trong *D*. Theo những mệnh đề trước và sự thảo luận trên, ta có thể thấy rằng mọi vùng *D* của bất kì tập điểm *S* hữu hạn điểm tồn tại một vùng *D’* khác cũng thuộc tập điểm này mà với *D* bao phủ *D’* và toàn bộ tập *D’* với một ngoại lệ là duy nhất với là một số dương bé hơn hoặc bằng 1.

Chúng ta hãy giả sử *D* là một vùng của tập *S*, và . Đặt *D’* là vùng mà , bị bao phủ bởi *D* mà , với và Điều này ám chỉ rằng . Do đó chúng ta có . Đặt , chúng ta có .

Với mỗi sự chọn lựa giữa các điểm, xác suất một điểm không được vẽ từ *H1*là . Giả sử điểm được chọn ngẫu nhiên từ *S*. Xác suất để tất cả chúng không được vẽ từ *H1*thì bé hơn:

Khi chúng ta chọn ngẫu nhiên điểm từ *S*, xác suất để ít nhất hai trong số chúng được vẽ từ *H1*thì lớn hơn . Bởi vì *D* bao phủ *D’* chúng ta có thể kết luận rằng nếu các điểm được chọn ngẫu nhiên từ *S*, thì xác suất để ít nhất 2 điểm được vẽ từ cùng tập *D* là ít nhất với *c* là hằng số.

Chúng ta vẫn còn một vấn đề: có phải có một vùng *T*0 mà với *c0*là hằng số ? Vùng này có thể được tìm thấy bằng giải thuật sau

Input: Tập *S* gồm *n* điểm

Output: Một vùng *T*­0 mà .

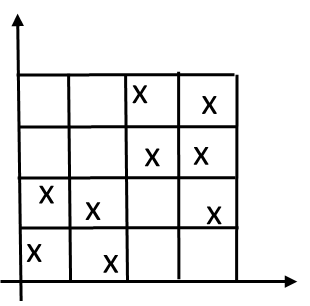
Bước 1: Tìm một vùng *T* với kích thước đủ tốt mà để từng ô vuông trong *T* chứa ít nhất một điểm của *S* và không có điểm nào của *S* nằm ở đường kẻ. Từ đó

Bước 2: Trong khi , gấp đôi kích thước của *T*

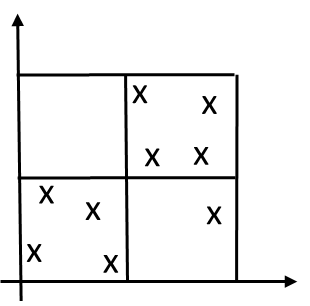
Bây giờ chúng ta sẽ giải thích ý nghĩa của giải thuật này.

Bước 1 của giải thuật phân vùng các điểm chính của tập vào các ô vuông mà mỗi ô vuông chứa ít nhất một điểm. Dĩ nhiên trong trường hợp này, *N(T)* sẽ bằng 0 và không có ít gì với chúng ta. Do đó chúng ta phải gấp đôi kích thước của mỗi ô vuông dần dần cho đến khi chúng ta đạt được vùng đầu tiên mà .

Xem xét Hình 4.4.5 với . Hình 4.4.5(a) biểu diễn vùng ban đầu là kết quả của bước 1 của giải thuật. Giả sử chúng ta gấp đôi kích thước cạnh. Chúng ta thu được các vùng được biểu diễn trong hình Hình 4.4.5(b) . Bây giờ tổng số phép tính khoảng cách sẽ là . Do đó sự phân vùng đặc biệt này sẽ là thỏa mãn tính chất của *T0*. Lưu ý rằng trong trường hợp này .



**(a)**



**(b)**

Hình 4.4.5 Ví dụ minh họa giải thuật

Gọi là kích thước cạnh của *T0*. Đối với sự chọn lựa ngẫu nhiên bất kì tập *Sa* gồm điểm nếu tồn tại 2 điểm của *Sa* mà chúng nằm trong cùng một ô vuông của *T0*, thì khoảng cách ngăn nhất trong *Sa­* bé hơn hoặc bằng và bất đẳng thức này cố định với xác suất lớn hơn . Chúng ta đã chứng mình rằng tồn tại một *T0*then chốt có kích thước là với 2 tính chất sau:

1. Xác suất để là là rất cao, trong đó là khoảng cách ngắn nhất trong tập điểm được chọn ngẫu nhiên.

Do đó chúng ta có thể giải thích rằng trong giải thuật ngẫu nhiên tìm cặp điểm gần nhất , bước 5 cần *O(n)* bước. Lưu ý rằng bước 1 chỉ là bước đệ quy. Và bởi bước 5 là một bước có ảnh hưởng lớn, chúng ta có thể kết luận rằng giải thuật ngẫu nhiên tìm cặp điểm gần nhất là một giải thuật *O(n)*.

## Bài toán kiểm tra phép nhân ma trận

Giải thuật Freivalds (đặt tên theo Rusins Freivalds) là một giải thuật xác suất ngẫu nhiên được sử dụng kiểm tra phép nhân ma trận. Cho ba ma trận *A*, *B* và *C* kích thước , kiểm tra liệu . Một giải thuật thông thường có thể tính sau đó so sánh kết quả với *C*. Tuy nhiên, ngay cả khi dùng một giải thuật nhân ma trận tốt nhất, biên tốt nhất có thể đạt được với cách tiếp cận này là theo thời gian. Giải thuật Freivalds sử dụng sự ngẫu nhiên để giảm biên thời gian này xuống với một xác suất cao. Trong độ phức tạp thời gian của thuật toán có thể xác minh ma trận kết quả với xác suất sai bé hơn .

Mô tả thuật toán

**Input:** Ma trận *A*, *B* và *C* kích thước

**Output:** Đúng, nếu ; Sai, ngược lại.

Giải thuật

1. Tạo một ma trận ngẫu nhiên (chỉ gồm giá trị 0 và 1) kích thước , được gọi là vectơ .
2. Tính .
3. Trả về “Đúng” nếu ; Ngược lại, trả về “Sai”.

**Lỗi**

Nếu , thì giải thuật luôn trả lời “Đúng”. Nếu , thì xác suất giải thuật cũng trả lời “Đúng” thì bé hơn hoặc bằng . Điều này được gọi là *lỗi một chiều* (one-sided error). Bằng cách thực hiện giải thuật *k* lần và chỉ trả lời “Đúng” khi tất cả các lần lặp đều trả lời “Đúng”. Khi đó, ta đạt được độ phức tạp về thời gian là và xác suất lỗi .

Ví dụ 4.5.1

Giả sử có một yêu cầu xác minh:

.

Một vectơ ngẫu nhiên có bộ số gồm hai phần tử với giá trị nhập bằng *0* hoặc bằng *1*-được gọi là và dùng để tính

Ta được kết quả là vectơ *0*. Tuy nhiên, nếu trong lần thử thứ hai, được chọn, kết quả sẽ như sau:

Kết quả không bằng *0*, chứng minh thực tế .

Có tất cả bốn vectơ gồm 2 phần tử *0/1*, và hai trong số đó cho kết quả vectơ *0* (), vì vậy khả năng xuất hiện của sự chọn lựa ngẫu nhiên này trong hai lần thử (và giả sử kết luận rằng ) là hoặc . Trong trường hợp thông thường, tỷ lệ *r* đưa ra kết quả là vectơ *0* thì có thể nhỏ hơn , và với số lượng phép thử lớn (chẳng hạn hai mươi lần), giúp cho xác suất lỗi là rất nhỏ.

Phân tích lỗi

Đặt *p* là xác suất lỗi. Cho rằng nếu , thì *p*=0, và nếu , thì *p*.

Nếu , thì , và vì vậy , chúng ta không cần quan tâm vectơ là gì.

Đặt , do đó . Do , chúng ta có , vì vậy một vài phần tử của *D* khác 0.

Giả sử rằng phần tử . Bằng định nghĩa của phép nhân ma trận, chúng ta có . Sử dụng định lý *Bayer* chúng ta lại có

.

Chúng ta cũng cần lưu ý rằng:

Khi thêm cái này vào hàm trên, chúng ta có:

Do đó,

## Bài toán tìm cây bao trùm nhỏ nhất của đồ thị

Trong phần này, chúng tôi sẽ giới thiệu một giải thuật ngẫu nhiên tìm cây bao trùm nhỏ nhất với độ phức tạp là Giải thuật này tốt hơn rất nhiều so với các giải thuật truyền thống tìm cây bao trùm nhỏ nhất (giải thuật Kruskal, độ phức tạp hay giải thuật Prim, độ phức tạp ).

Giải thuật này dựa trên khái niệm Boruvka step, được đề xuất bởi Boruvka vào năm 1926. Bổ đề sau minh họa ý tưởng bên trong Boruvka step:

Bổ đề 4.6.1 Gọi và là tập hợp những đỉnh khác rỗng khi và  
 và gọi cạnh (v,u) là cạnh có trọng số nhỏ nhất với một điểm kết thúc trong và một điểm kết thúc trong . Thì, (u,v) là một trong các cạnh trong cây bao trùm nhỏ nhất của G.

Bổ đề 4.6.1 có thể phát biểu dưới dạng khác như sau: trong một đồ thị G, với mỗi đỉnh u, trong số các cạnh liên thuộc u, nếu cạnh (u,v) có trọng số nhỏ nhất thì (u,v) phải là cạnh nằm trong cây bao trùm nhỏ nhất của G. Xem



**Hình 4.6.1**. Với đỉnh 2, trong số tất cả cạnh liên thuộc với 2, Cạnh (2,4) có trọng số nhỏ nhất. Vì vậy, cạnh (2,4) phải nằm trong cây bao trùm nhỏ nhất của G. Tương tự, dễ dàng chứng minh cạnh (5,6) cũng phải nằm trong cây bao trùm nhỏ nhất .



**Hình 4.6.1 Đồ thị ban đầu**

Bây giờ chúng ta hãy chọn tất cả cạnh phải nằm trong cây bao trùm nhỏ nhất, của đồ thị



**Hình 4.6.1** dựa vào Bổ đề 4.6.1. Kết quả những thành phần liên thông được thể hiện ở hình 4.6.2. Trong đồ thị Hình 4.6.2, tất cả những đường đứt nét thì kết nối với các cạnh thành phần. bây giờ chúng ta hãy thu nhỏ tất cả các đỉnh liên thông với nhau thành một đỉnh. Bây giờ ta có 5 đỉnh, thể hiện trong Hình 4.6.3. Sau khi chúng ta loại trừ những cạnh bội và khuyên được kết quả ở Hình 4.6.4.

Khi đồ thị kết quả chứa nhiều hơn một đỉnh, Chúng ta áp dụng Bổ đề 4.6.1 một lần nữa. Kết quả thể hiện ở Hình 4.6.5 và cạnh được chọn là (0,3), (2,11) và (6,7).

Sau khi thu nhỏ các nút ở mỗi thành phần liên thông, chúng ta có 2 đỉnh. (Hình 4.6.6) Sau khi chúng ta loại trừ những cạnh bội và khuyên được kết quả ở Hình 4.6.7.



Hình 4.6.2 Các cạnh được chọn trong Boruka step



Hình 4.6.3 Đồ thị sau khi xây dựng



Hình 4.6.4 Kết quả sau khi áp dụng Boruvka step lần thứ nhất



Hình 4.6.5 Các cạnh được chọn trong Boruvka step lần 2



Hình 4.6.6 Đồ thị được xây dựng lại từ Boruvka step lần 2



Hình 4.6.7 Đồ thị kết quả thu được từ Boruvka step lần 2



Hình 4.6.8 Cây bao trùm nhỏ nhất thu được

Một lần nữa, sau khi chúng ta loại bỏ cạnh bội và chọn (8,7), chúng ta có thể thu nhỏ tất cả đỉnh thành 1 đỉnh. Tiến trình hoàn thành. Những cạnh được chọn tạo thành một cây bao trùm nhỏ nhất (Hình 4.6.8)

Giải thuật Boruvka’s để tìm cây bao trùm nhỏ nhất áp dụng Boruvka step một cách đệ quy cho đến khi đồ thị kết quả chỉ còn một cạnh. Gọi đầu vào là đồ thị G(V,E) và đầu ra là đồ thị G’(V’,E’). Boruvka step mô tả tiếp theo.

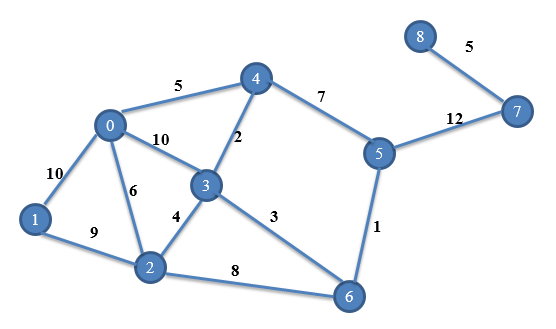
**Boruvka step:**

1. Với mỗi đỉnh u, Tìm cạnh (u,v) với trọng số nhỏ nhất kết nối với chúng. Tìm tất cả cách thành phần liên thông bằng cánh đánh dấu cạnh.
2. Thu nhỏ mỗi thành phần liên thông xác định bởi cạnh đánh dấu thành một đỉnh duy nhất. Gọi đồ thị kết quả là G’(V’,E’). Loại bỏ cạnh bội và khuyên.

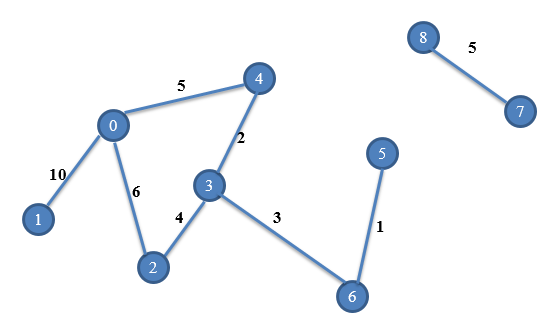
Độ phức tạp của một Boruvka step là O(n+m) với n là số cạnh, m là số đỉnh. Để G liên thông thì m>n. Do đó, O(n+m)=O(m). Để mỗi thành phần liên thông xác định bởi một cạnh đánh dấu chứa ích nhất 2 đỉnh, sau khi thi hành mỗi Boruvka step , số cạnh của đồ thị thì nhỏ hơn một nữa so với ban đầu, do đó, tổng các lần Boruvka là O(log n). độ phức tạp của giải thuật Boruvka là O(m log n).

Để sử dụng Boruvka step hiệu quả, chúng ta phải sử dụng một khái niệm mới. Xem Hình 4.6.9, Hình 4.6.10, Hình 4.6.11, Hình 4.6.12. Trong Hình 4.6.10, đồ thị G­s là một đồ thị con của G ở Hình 4.6.9. Trong Hình 4.6.12,rừng bao trùm nhỏ nhất F được gắn vào đồ thị G ban đầu . Tất cả những cạnh không thuộc F là những cạnh gạch đứt. chúng ta hãy xét cạnh (4,5). Trọng số của (4,5) là 7. Vâng, có một đường nối giữa đỉnh 4 và đỉnh 5 trong rừng F, đó là (4,3)-> (3,6)->(6,5). Trọng số của (4,5) lớn hơn trọng số lớn nhất của con đường này. Theo một bổ đề được phát biểu sao đây, cạnh (4,5) không thể là cạnh nằm trong cây bao trùm nhỏ nhất của G. Trước khi phát biểu bổ đề, chúng ta hãy mô tả một cụm từ gọi là F-heavy.

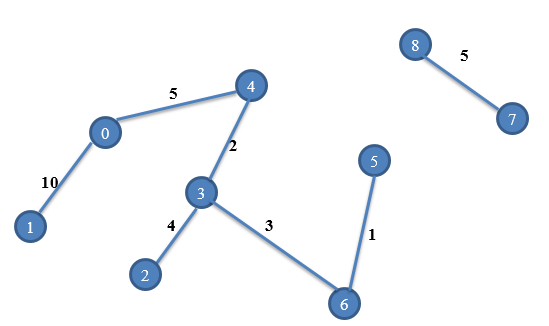
Gọi *w*(*x*, *y*) thể hiện trọng số của cạnh (*x*, *y*) trong *G*. Gọi *Gs* là một đồ thị con của *G*. *F* là rừng bao trùm nhỏ nhất của *Gs*. *wF*(*x*, *y*) thể hiện trọng số lớn nhất của một cạnh trong một đường liên thông *x* và *y* trong *F*. Nếu *x* và *y* là 2 đỉnh không liên thông trong *F*, đặt . Chúng ta nói rằng cạnh (*x*,*y*) là *F*-heavy(*F*-light) với *F* nếu .



Hình 4.6.9



Hình 4.6.10



Hình 4.6.11



Hình 4.6.12

7

Xem xét Hình 4.6.12. Chúng ta có thể thấy rằng cạnh (4,5), (0,3) và (2, 6) là tất cả những F-heavy trong mối quan hệ với F. Với khái niệm mới này thì chúng ta có Bổ đề 4.6.2 khá quan trọng.

Bổ đề 4.6.2 Gọi Gs là một đồ thị con của G. F là một rừng bao trùm nhỏ nhất của Gs . cạnh F-heavy trong G với mối liên hệ với F không thể là cạnh nằm trong cây bao trùm nhỏ nhất.

Chúng tôi cũng không chứng minh Bổ đề 4.6.2, với bổ đề này, chúng ta biết cạnh (4,5), (0, 3),(2, 6) không thể là cạnh cây bao trùm nhỏ nhất.

Chúng ta cần một bổ đề khác để dùng Boruvka step. Đó là Bổ đề 4.6.3

Bổ đề 4.6.3 Gọi H là đồ thị con của G bằng cánh bao gồm mỗi cạnh độc lập với xác xuất p, và gọi F là cây bao trùm nhỏ nhất của H. Con số mong đợi của cạnh F-light trong G lớn nhất là n/p, với n là số đỉnh của G.

**Giải thuật**

**Input:** Một đồ thị liên thông có trọng số G

**Output**: cây bao trùm nhỏ nhất của G

Bước 1: Thực hiện Boruvka step 3 lần. Gọi kết quả là đồ thị G1(V1, E1). Nếu G1 chứa một đỉnh, trả về bộ các cạnh đánh dấu ở bước 1 và thoát.

Bước 2: Phát sinh một đồ thị con H của G1 bằng cách chọn mỗi cạnh phụ thuộc với xác xuất ½. Sử dụng giải thuật đệ quy cho H để tạo một rừng bao trùm nhỏ nhất F của H. nhận được đồ thị G2(V2, E2) bằng cách xóa toàn bộ cạnh F-heavy trong G1 với mối quan hệ với F.

Bước 3: Sử dụng giải thuật đệ quy với G2

Độ phức tạp của giải thuật

Đặt T(|V|,|E|) là giá trị kỳ vọng của thời gian thời gian của giải thuật cho đồ thị

Mỗi lần chạy bước 1 tốn . Sau khi thực hiện bước 1, ta có

và . Với bước 2, thời gian để tính *H* là . Thời gian để tính *F* là T. Thời gian cần thiết để loại bỏ những cạnh *F*-heavy là . Sử dụng Bổ đề 4.6.3, ta có giá trị kỳ vọng cho lớn nhất là . Từ đó, ta có kỳ vọng của thời gian cần thiết để thực hiện bước 3 là

T. Đặt và . Chúng ta có hệ thức truy hồi sau:

Với *c* là hằng số. Ta tìm được:

Như vậy, kỳ vọng thời gian chạy của giải thuật là

Mã giả

1. Randomized-MST (G)
2. G1 = BoruvkaStep(G); // Boruvka Step lần thứ I
3. **if** G1 has edges
4. **if** G1 has one edg //nếu G1 chỉ có một cạnh thì thêm vào đồ thị kết quả (solvedGraph)
5. Add this edg to solvedGraph
6. **else**
7. G2 = BoruvkaStep(G1); // Boruvka Step lần thứ II
8. **if** G2 has edges
9. **if** G2 has one edg //nếu G2 chỉ có một cạnh thì thêm vào đồ thị kết quả (solvedGraph)
10. Add this edg to solvedGraph
11. **else**
12. G3 = BoruvkaStep(G2); // Boruvka Step lần thứ III
13. **if** G3 has edges
14. **if** G3 has one edg //nếu G3 chỉ có một cạnh thì thêm vào đồ thị kết quả
15. Add this edg to solvedGraph // kết thúc bước 1
16. **else**

// phát sinh một đồ thị con H bằng cách chọn các cạnh từ đồ thị G3 với xác xuất 1/2

1. Obtain a subgraph H of G3 by selecting each edge with probability ½
2. F=Randomized-MST(H)
3. **for** each edg in G3 and not in F
4. **if** this edg is not a F-heavy edges
5. Add this edg to G4 // kết thúc bước 2
6. Randomized-MST(G4); // bước 3
7. **return** solvedGraph

## Kết luận

Trong chương này, chúng tôi đã giới thiệu và phân tích độ phức tạp của một số giải thuật ngẫu nhiên cho các bài toán thuộc các lĩnh vực khác nhau: cấu trúc dữ liệu (bài toán sắp xếp), lý thuyết số (bài toán kiểm tra số nguyên tố, bài toán nhân ma trận), hình học (bài toán tìm cập điểm gần nhất), lý thuyết đồ thị ( bài toán tìm cây bao trùm nhỏ nhất). Trong chương tiếp theo chúng tôi sẽ trình bày phần hiện thực của các giải thuật ngẫu nhiên được trình bày ở chương này.

# Chương 5

# HIỆN THỰC CÁC ỨNG DỤNG

# GIẢI THUẬT NGẪU NHIÊN



## Giới thiệu

## Bài toán sắp xếp

## Bài toán xác định số nguyên tố

## Bài toán tìm cặp điểm gần nhất

## Bài toán so trùng mẩu

## Giải thuật ngẫu nhiên tìm cây bao trùm nhỏ nhất của đồ thị

Cấu trúc dữ liệu

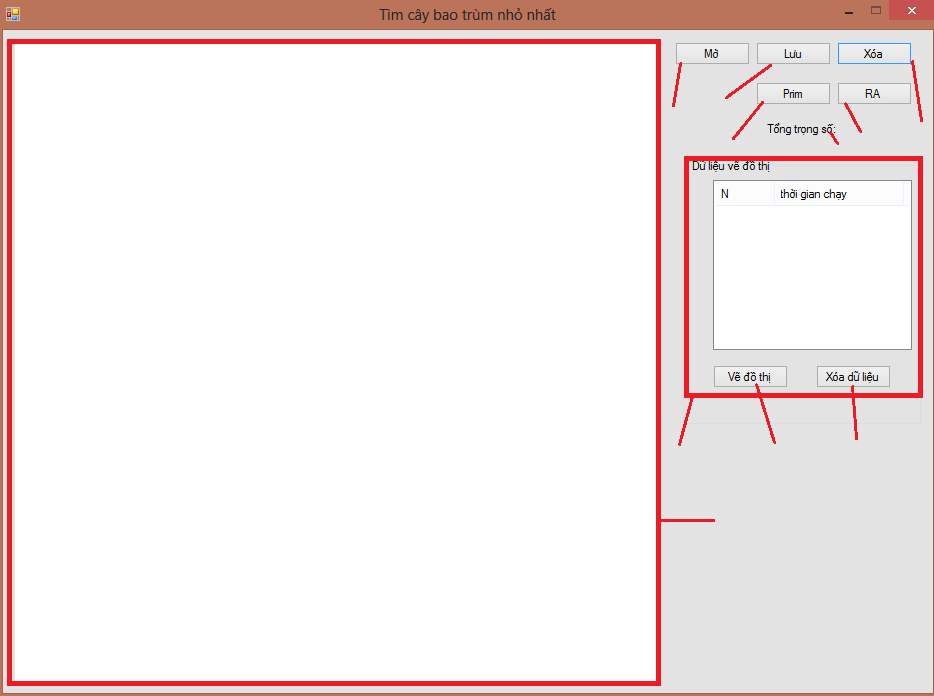
Chúng tôi mô tả một đồ thị là danh sách các cạnh, mỗi cạnh bao gồm 2 *đỉnh* (vertex), *trọng số của cạnh* (cost), vị trí hiển thị giá trị cost trên form (point).

Mỗi vertex bao gồm các thuộc tính *tên đỉnh* (name), *vùng* (rank), *nút gốc*( root), *vị trí hiển thị đỉnh trên form* (point).

Giao diện và sử dụng

Giao diện chính của phần hiện thực cho giải thuật tìm cây bao trùm nhỏ nhất:

Hình 5.6.1 Giao diện chính của giải thuật tìm cây bao trùm nhỏ nhất



**1**

**8**

**9**

**10**

**7**

**6**

**4**

**5**

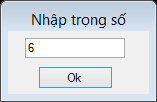
**3**

**2**

Vùng (1) đây là khu vực vẽ đồ thị mới, hiển thị đồ thị đã có lên đồng thời cũng là nơi hiển thị kết quả (cây bao trùm nhỏ nhất cần tìm)

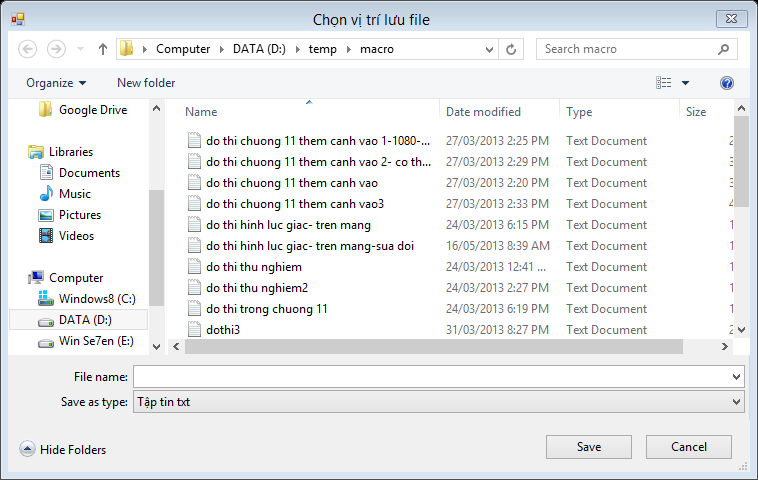
Khi muốn vẽ một đồ thị mới, ta tiến hành click chuột trên vùng 1, mỗi lần click ta sẽ nhận được một đỉnh của đồ thị tại vị trí click. Sau đó, giữ phím Ctrl click chọn lần lượt 2 đỉnh thuộc cạnh này. Một hộp thoại (Hình 5.6.2) hiện lên ta nhập trọng số của cạnh này vào.

Hình 5.6.2 Nhập trọng số



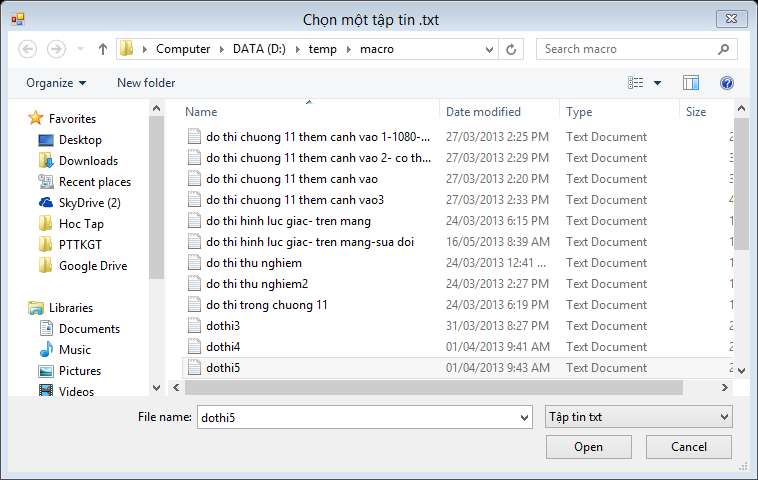
Sau khi vẽ xong đồ thị chúng ta có thể sử dụng nút *Lưu (3)* để lưu lại đồ thị đang có, để tiện sử dụng cho lần sau. Khi click *Lưu* thì một cửa số hiện ra (Hình 5.6.3) ta chọn vị trí lưu, và đặt tên file, click *Save.*

Hình 5.6.3 Chọn vị trí lưu

**

Chúng ta có thể mở đồ thị đã vẽ sẵn bằng cách click nút *Mở* (2)*.* Một của sổ hiện ra (Hình 5.6.4) , chọn file đã lưu trước đó, rồi click *Open.* Đồ thị đã lưu sẽ hiện lại lên vùng (1), chúng ta có thể sử dụng luôn hay vẽ thêm cạnh.

Hình 5.6.4 Chọn một tập tin để mở.

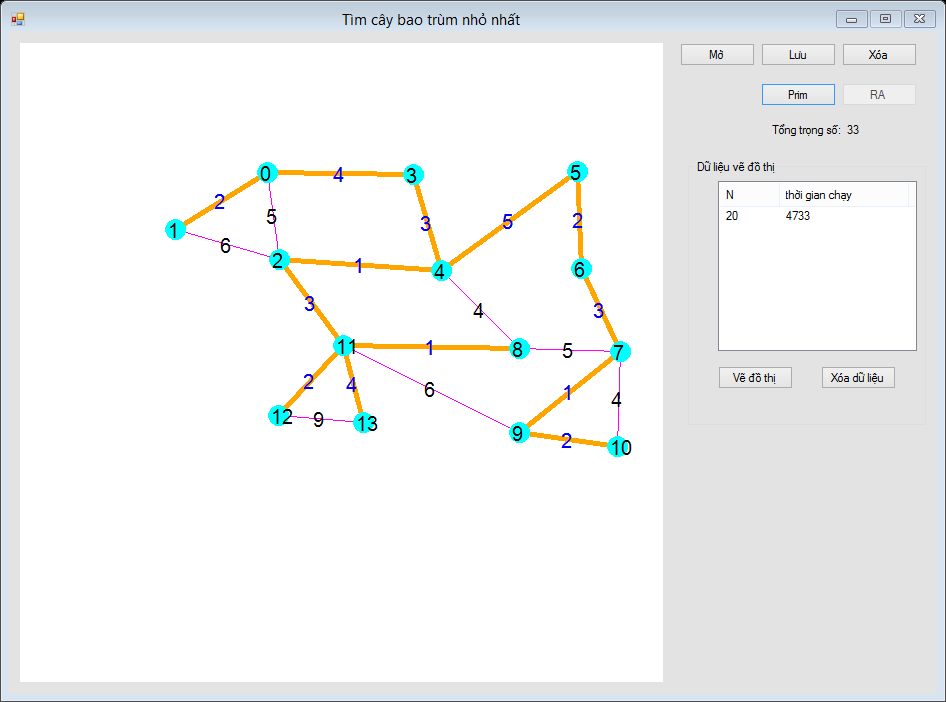


Nút *Xóa* (4) cho phép ta xóa đồ thị đang có trên vùng (1).

Nút *Prim* (5) và *RA* (6), đây là 2 nút quan trọng nhất , nút (5) cho phép ta tìm cây bao trùm nhỏ nhất theo giải thuật Prim, nút (6) cho phép ta tìm cây bao trùm nhỏ nhất theo giải thuật ngẫu nhiên đã trình bày ở chương 4.

*Tổng trọng số* (7), là tổng trọng số của cây bao trùm nhỏ nhất tìm được

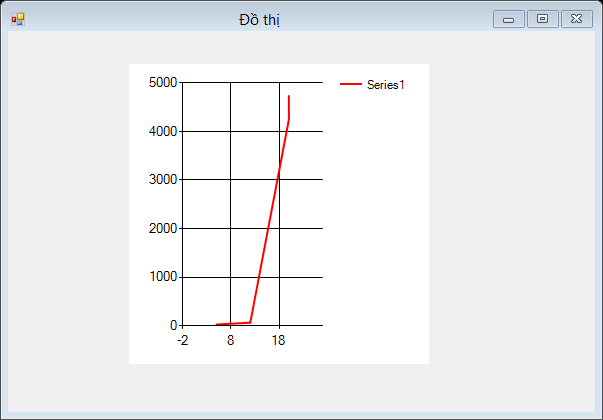
Hình 5.6.5 Kết quả thu được khi thực hiện giải thuật tìm cây bao trùm nhỏ nhất



Vùng (8) là khu vực dùng vẽ đồ thị thời gian chạy cho giải thuật

Mỗi lần thực hiện giải thuật, chương trình sẽ đếm số cạnh của đồ thị và tính thời gian chạy (đơn vị microsecond) đưa vào list. Từ đó ta có thể vẽ đồ thị thời gian (như Hình 5.6.6) chạy bằng nút *Vẽ đồ thị* (9), ta cũng có thể xóa hết dữ liệu trong list này bằng cách click nút *Xóa dữ liệu* (10).

Hình 5.6.6 Đồ thị thời gian chạy.

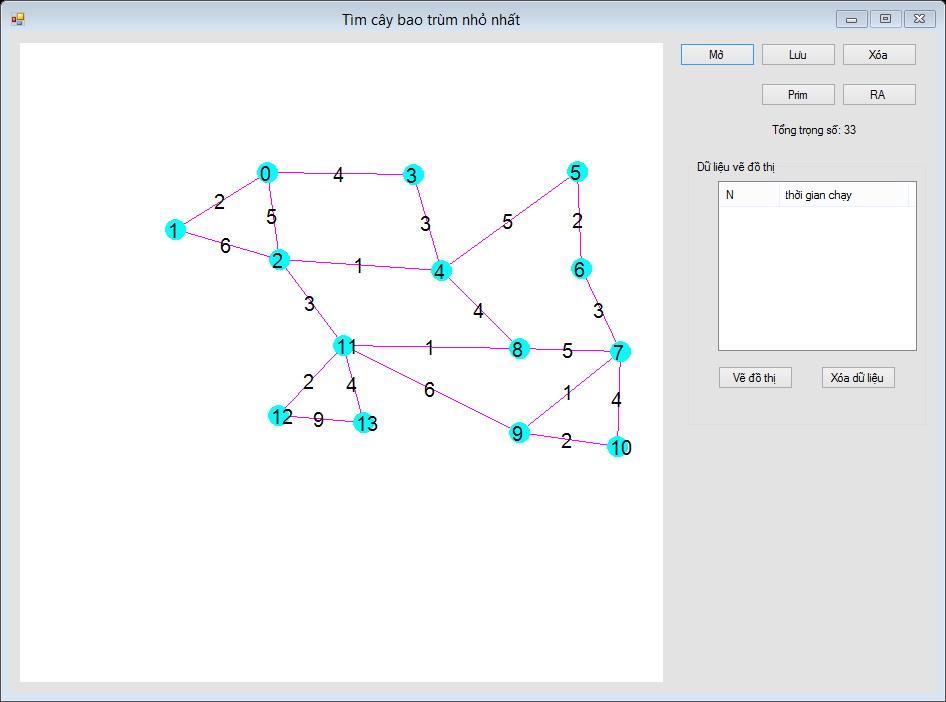


Dữ liệu thử

Đồ thị 1

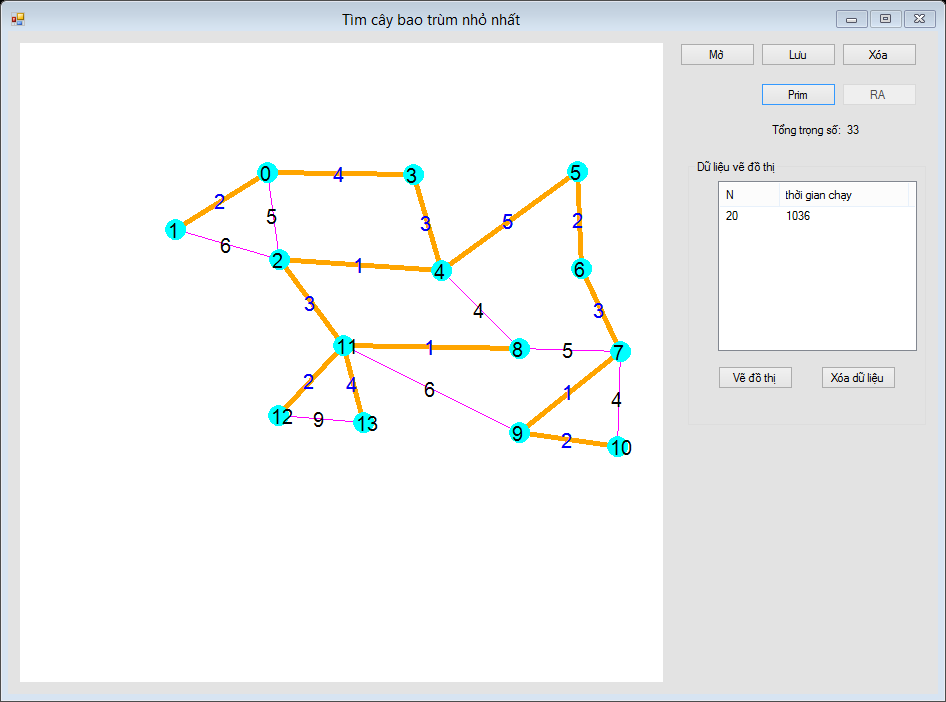
Đồ thị đưa vào (Hình 5.6.7)

Hình 5.6.7



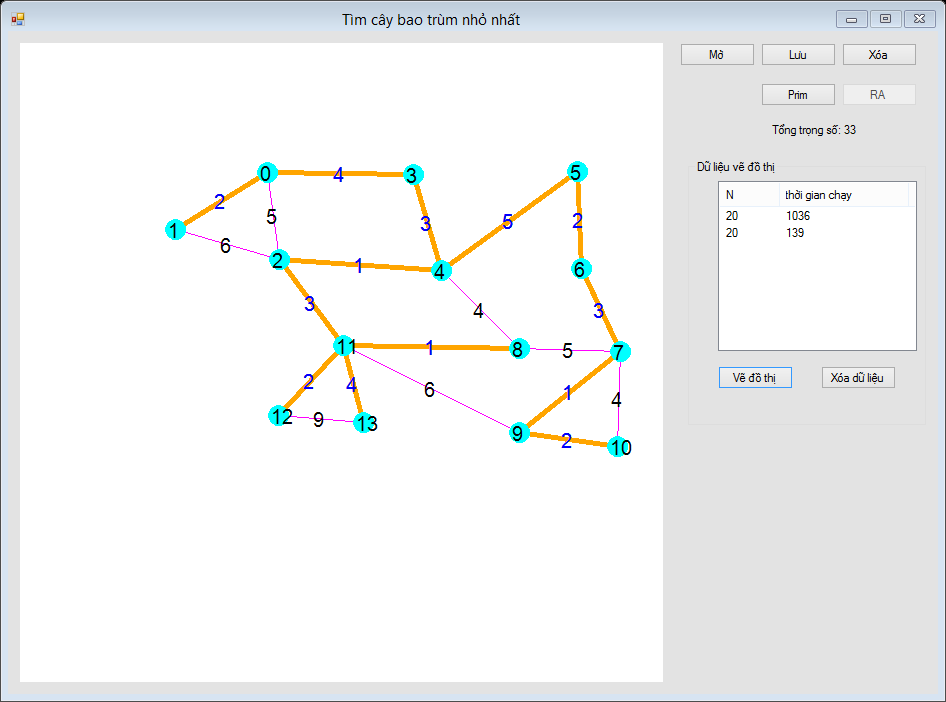
Kết quả khi chạy bằng giải thuật Prim (Hình 5.6.8)

Hình 5.6.8



Kết quả khi chạy bằng giải thuật ngẫu nhiên (Hình 5.6.9)

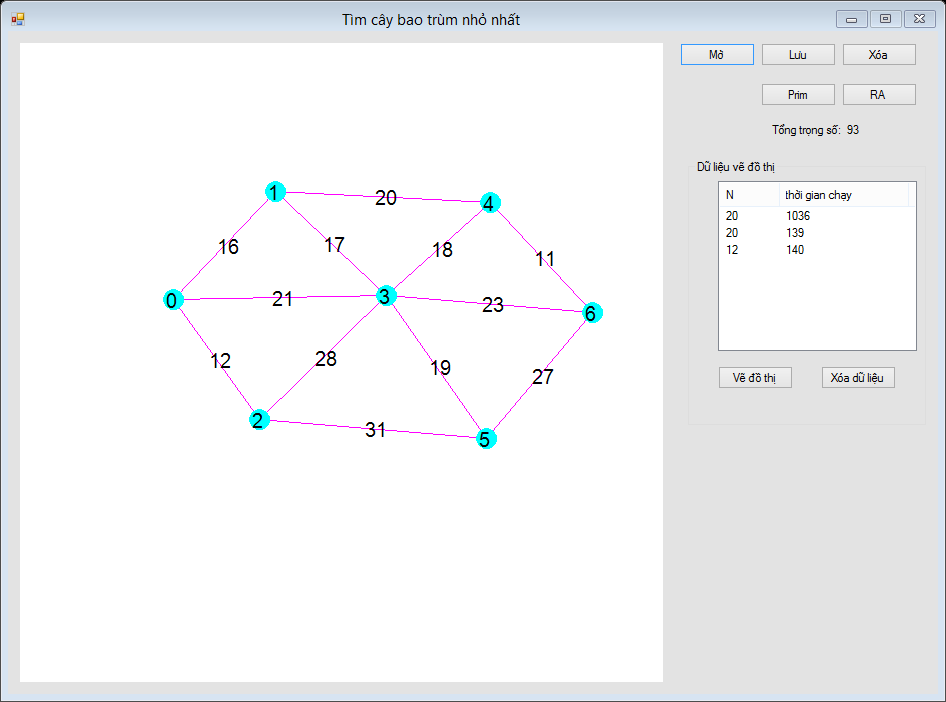
Hình 5.6.9



Đồ thị 2

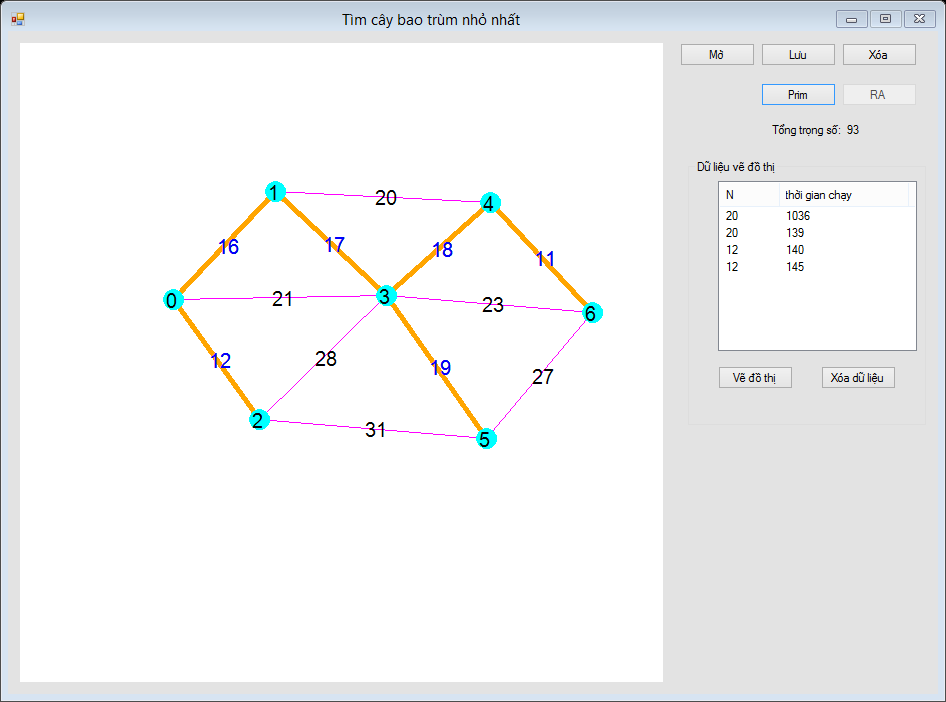
Đồ thị đưa vào (Hình 5.6.10)

Hình 5.6.10



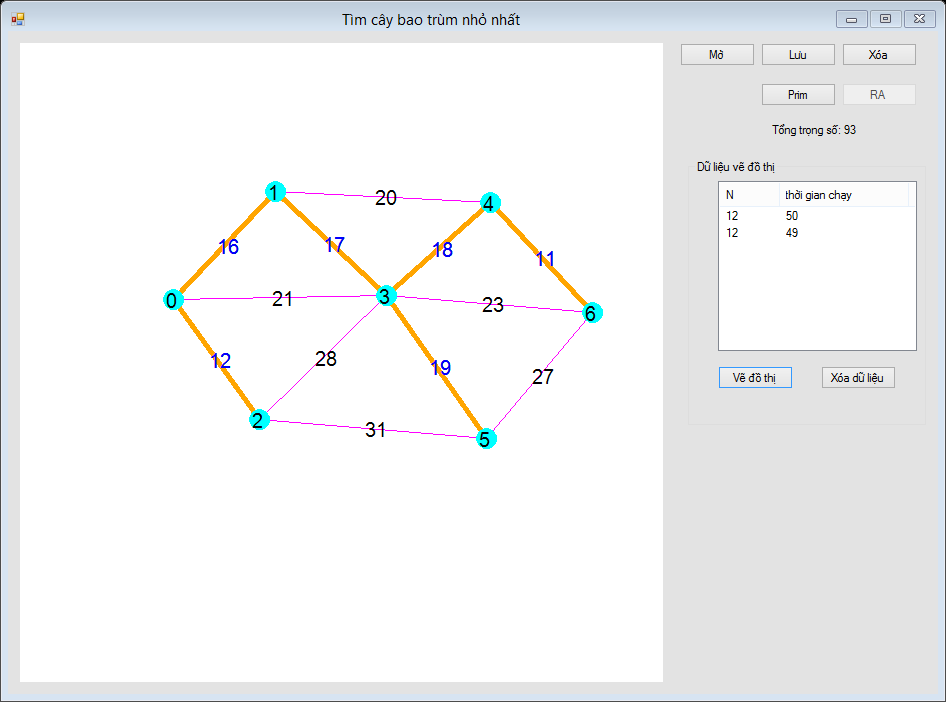
Kết quả khi chạy bằng giải thuật Prim (Hình 5.6.11)

Hình 5.6.11



Kết quả khi chạy bằng giải thuật ngẫu nhiên (Hình 5.6.12)

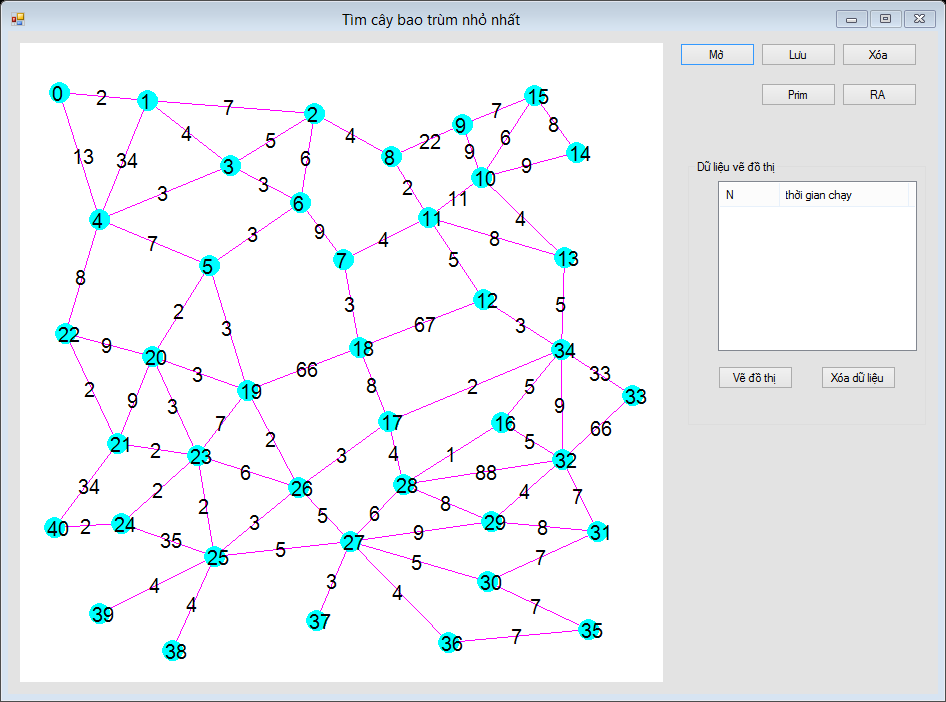
Hình 5.6.12



Đồ thị 3

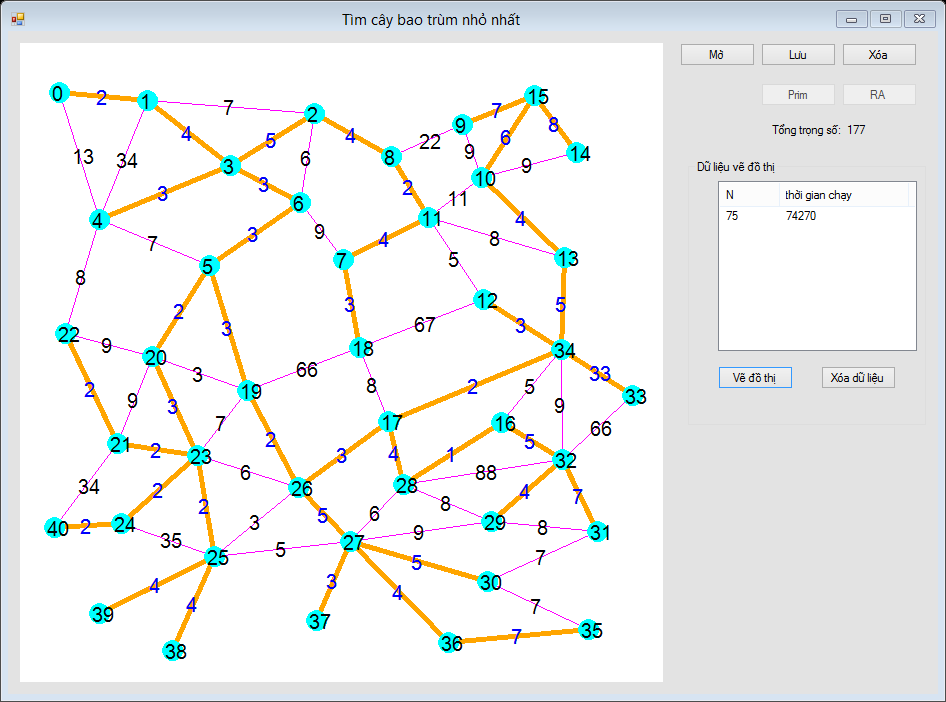
Đồ thị đưa vào (Hình 5.6.13)

Hình 5.6.13



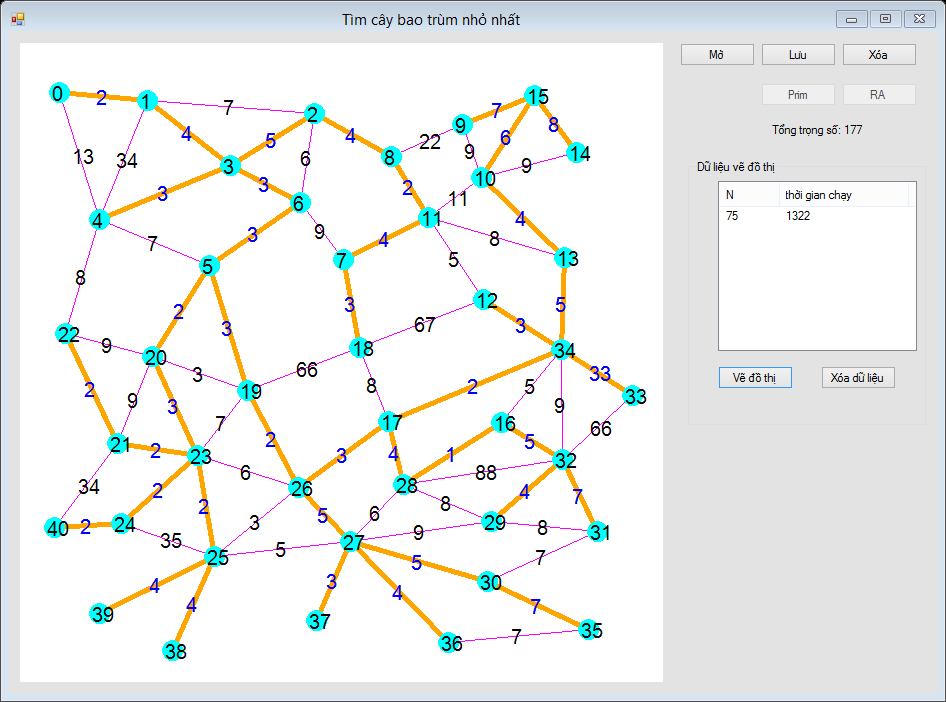
Kết quả khi chạy bằng giải thuật Prim (Hình 5.6.14)

Hình 5.6.14



Kết quả khi chạy bằng giải thuật ngẫu nhiên (Hình 5.6.15)

Hình 5.6.15



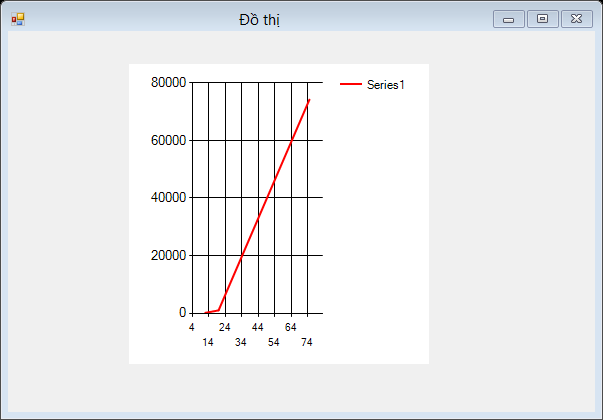
So sánh giữa giải thuật ngẫu nhiên và giải thuật Prim

Bảng 1 Bảng so sánh thời gian chạy (đơn vị tính: microsecond) của giải thuật ngẫu nhiên và giải thuật Prim

|  |  |  |
| --- | --- | --- |
| Đồ thị | Thời gian chạy của giải thuật Prim | Thời gian chạy của giải thuật ngẫu nhiên |
| Đồ thị 1 | 988 | 139 |
| Đồ thị 2 | 146 | 68 |
| Đồ thị 3 | 75158 | 1323 |

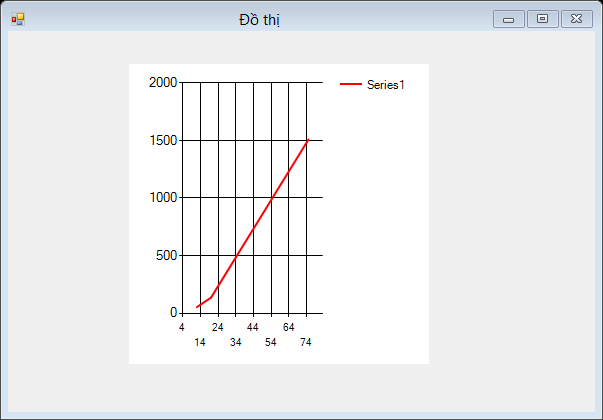
Biểu đồ thời gian chạy của giải thuật Prim

Hình 5.6.16 Biểu đồ thời gian chạy của giải thuật Prim



Biểu đồ thời gian chạy của giải thuật ngẫu nhiên

Hình 5.6.17 Biểu đồ thời gian chạy của giải thuật ngẫu nhiên



## Kết luận

# Chương 6

# TỔNG KẾT VÀ ĐỀ NGHỊ



## Tổng kết

## Đề nghị

# TÀI LIỆU THAM KHẢO

[1] Karp Richard M. *An introduction to randomized algorithms*. Discrete Applied

[2] Mathematics, 34, 1991, 165-201.

[3] Lee R.C.T, Tseng S.S, Chang R.C, Tsai Y.T. *Introduction to The design and Analysis of Algorithms-A Strategic Approach*. McGraw-Hill Education, 2005.

[4] Levitin A. *Introduction to The design and Analysis of Algorithms*. Addison-Wesley, 2012.

[5] Mitzenmacher M., Upfal E. *Probability and Computing Randomized Algorithms and Probabilistic Analysis*. Cambridge university press, 2005

[6] Motwani R., Prabhakar Raghavan P. *Randomized algorithms*. Cambridge university press, 1995.

[7] Rosen K.H. *Discrete Mathematics*, Prentice Hall Inc., 2012

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald D. Rivest, clifford Stein. *Introduction to Algorithms*. McGraw-Hill Book Company, 2009.

[9] en.wikipedia.org/

[10] vi.wikipedia.org/