ỦY BAN NHÂN DÂN TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

PHAN HOÀNG NHÂN

ĐINH NGỌC THANH MY

NGHIÊN CỨU VÀ ỨNG DỤNG CÁC GIẢI THUẬT NGẪU NHIÊN

KHÓA LUẬN TỐT NGHIỆP

NGÀNH: CÔNG NGHỆ THÔNG TIN

TRÌNH ĐỘ ĐÀO TẠO: ĐẠI HỌC

NGƯỜI HƯỚNG DẪN: TS. NGUYỄN HÒA

NGƯỜI PHẢN BIỆN:

TP. HỒ CHÍ MINH, THÁNG 08 NĂM 2013

LỜI CAM ĐOAN

Chúng tôi xin cam đoan nội dung của khóa luận này là kết quả nghiên cứu của chính chúng tôi. Các số liệu và kết quả nghiên cứu nêu trong khóa luận là trung thực. Tất cả những tham khảo từ các nghiên cứu liên quan đều được nêu rõ nguồn gốc tại danh mục tài liệu tham khảo ở phần sau của khóa luận. Những đóng góp trong khóa luận là kết quả nghiên cứu của chúng tôi và chưa được công bố trong bất kỳ công trình khoa học nào khác.

Ngày 20 tháng 08 năm 2013

Sinh viên

Phan Hoàng Nhân

Đinh Ngọc Thanh My

LỜI CẢM ƠN

Đầu tiên, chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy Nguyễn Hòa, thầy đã nhiệt tình hướng dẫn, góp ý, động viên và giúp đỡ chúng em hoàn thành tốt khóa luận này.

Chúng em cũng xin bày tỏ lòng biết ơn chân thành đến tất cả quý thầy cô trường Đại học Sài Gòn nói chung và quý thầy cô trong khoa Công nghệ thông tin nói riêng đã tạo điều kiện thuận lợi để chúng em thực hiện khóa luận này. Các kiến thức mà thầy cô đã cung cấp cũng hỗ trợ cho chúng em rất nhiều trong việc nghiên cứu này.

Sau cùng, chúng tôi cũng xin gửi lời cảm ơn đến bạn bè và gia đình, những người đã luôn quan tâm, giúp đỡ và ở bên chúng tôi trong suốt quá trình học tập cũng như quá trình thực hiện khóa luận này.

TP Hồ Chí Minh, tháng 08 năm 2013

Sinh viên

Phan Hoàng Nhân

Đinh Ngọc Thanh My

TÓM TẮT

Trong khóa luận này, chúng tôi đã trình bày tổng quan về giải thuật cũng như những nghiên cứu về giải thuật ngẫu nhiên: định nghĩa, ưu khuyết điểm, phân loại, ứng dụng của giải thuật ngẫu nhiên. Bên cạnh đó, chúng tôi cũng giới thiệu, phân tích độ phức tạp theo thời gian, hiện thực giải thuật ngẫu nhiên trong một số bài toán thuộc các lĩnh vực khác nhau, và so sánh giải thuật ngẫu nhiên với các giải thuật truyền thống được dùng để giải quyết những bài toán đó. Chương trình hiện thực các giải thuật được viết bằng Window Form (ngôn ngữ C#). Trong phần hiện thực này, bộ số ngẫu nhiên trong các giải thuật được sinh bằng lớp Random của hệ thống.

MỤC LỤC

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH

Chương 1

MỞ ĐẦU

1.1. Phạm vi và mục tiêu

Như chúng ta đã biết *giải thuật cổ điển* (classical algorithm) là cơ sở để xây dựng các hệ thống tính toán nói chung và phần mềm máy tính nói riêng [2, 3, 8]. Tính hiệu quả của các phần mềm máy tính phụ thuộc quyết định vào việc phát triển các giải thuật để tạo nên chúng. Các giải thuật càng tốt thì các phần mềm được xây dựng càng hiệu quả và khả năng đáp ứng nhu cầu thực tế càng cao. Chính vì vậy, lý thuyết thiết kế và phân tích giải thuật là một trong các lĩnh vực đã và đang được nghiên cứu mạnh mẽ, không chỉ để giải quyết các bài toán đang tồn tại mà còn đáp ứng nhu cầu giải các bài toán mới luôn luôn được đặt ra từ thực tiễn.

Theo tinh thần đó, đã có nhiều chiến lược thiết kế giải thuật kinh điển [3] được phát triển và ứng dụng như *chia để trị* (divide-and-conquer), *biến đổi đệ trị* (transform- and-conquer), *qui hoạch động* (dynamic programming), *tham ăn* (greedy), v.v. nhằm giảm độ phức tạp thời gian và nâng cao hiệu quả tính toán khi giải các bài toán. Tuy nhiên, có rất nhiều bài toán khó giải khi ứng dụng các kỹ thuật thiết kế giải thuật đã nêu ở trên [8]. Nói một cách khác, độ phức tạp giải thuật để giải các bài toán này là một hàm mũ theo kích thước của bài toán. Với độ phức tạp như vậy, những giải thuật này nói chung không sử dụng được trong thực tế, đặc biệt khi kích thước bài toán đủ lớn.

Một ví dụ điển hình cho lớp bài toán này là bài toán "tìm chu trình Hamilton ngắn nhất trong đồ thị có trọng số". Cho đến nay khoa học máy tính chưa tìm thấy bất kỳ giải thuật nào tính toán đúng chu trình Hamilton ngắn nhất trong thời gian đa thức [2, 7]. Những bài toán không có giải thuật với thời gian chạy đa thức gọi là bài toán NP-Complete [8]. Có rất nhiều bài toán như vậy trong khoa học và thực tiễn cần phải giải quyết. Ví dụ, bài toán xác định tính *luôn đúng* (valid) của một *công thức logic* (well formed logic formula) là bài toán NP-Complete [8]. Một trong những phương pháp giải các bài toán NP-Complete là dùng các *giải thuật xấp xỉ* (approximation algorithm) để tìm lời giải gần đúng của bài toán với độ phức tạp đa thức [8]. Ví dụ có thể giải bài toán tìm chu trình Hamilton ngắn nhất bằng giải thuật xấp xỉ với độ phức tạp thời gian bậc hai theo số đỉnh của đồ thị.

Mặc dù giải thuật xấp xỉ là một công cụ tốt để giải các bài toán khó, nhưng chúng không thể là phương tiện để cải thiện (giảm) độ phức tạp các giải thuật đa thức đã có nhằm nâng cao hiệu quả tổng thể của các ứng dụng thực tế. Để khắc phục hạn chế của các giải thuật cổ điển nói chung và giải thuật xấp xỉ nói riêng, các *giải thuật ngẫu nhiên* (randomized algorithm) đã được nghiên cứu và phát triển [1, 2, 5, 6, 12], nhằm giảm độ phức tạp về thời gian giải toán, như một giải pháp thay thế.

Giải thuật ngẫu nhiên là giải thuật kết hợp dữ liệu ngẫu nhiên trong tiến trình thực thi các dòng lệnh của nó [1]. Hệ quả là trong các giải thuật ngẫu nhiên, các lệnh có thể được chọn ngẫu nhiên để thực hiện, dẫn đến tính *không đơn định* (nondeterministic) của giải thuật và do đó làm giảm độ phức tạp của nó. Chẳng hạn, chúng ta có thể giải bài toán xác định cặp điểm gần nhau nhất trong không gian hai chiều bằng giải thuật ngẫu nhiên tuyến tính O(n) [2] trong khi độ phức tạp giải thuật cổ điển tốt nhất để giải bài toán này là O($n\log_2 n$) [3]. Hay chúng ta có thể giải bài toán tìm cây bao trùm nhỏ nhất của đồ thị có trọng số bằng giải thuật ngẫu nhiên với độ phức tạp O(n+m), trong đó n và m tương ứng là số đỉnh và cạnh của đồ thị. Với độ phức tạp O(n+m), rõ ràng giải thuật ngẫu nhiên hiệu quả hơn nhiều so với giải thuật cổ điển giải nó trong lý thuyết đồ thị mà ta đã biết [2].

Mặc dù hiệu quả hơn so với giải thuật cổ điển tương ứng và được nghiên cứu, áp dụng rộng rãi trên thế giới, nhưng trong nước nói chung và trong các trường đại học Việt Nam nói riêng, các giải thuật ngẫu lại chưa được quan tâm nhiều. Đó cũng là động lực để chúng tôi thực hiện đề tài khóa luận tốt nghiệp "*nghiên cứu và ứng dụng các giải thuật ngẫu nhiên*" như một đóng góp nhằm làm sáng tỏ hơn tính chất, khả năng của giải thuật ngẫu nhiên cũng như cách thức thiết kế, tính toán độ phức tạp, hiện thực các giải thuật ngẫu nhiên nhằm áp dụng chúng vào các bài toán cụ thể.

Để thực hiện mục tiêu này, chúng tôi đã nghiên cứu khái niệm, các đặc trưng cơ bản và cách thức phân loại của các giải thuật ngẫu nhiên. Có hai loại giải thuật ngẫu nhiên [5, 6], loại thứ nhất luôn luôn cho lời giải đúng nhưng độ phức tạp thay đổi với một xác suất nhỏ, loại thứ hai có thể sai với một xác suất đủ nhỏ nhưng độ phức tạp luôn luôn xác định. Chúng tôi xem xét, nghiên cứu cách tiếp cận, phương pháp để thiết kế các giải thuật ngẫu nhiên. Cuối cùng, chúng tôi ứng dụng giải thuật ngẫu nhiên để giải một số bài toán cụ thể (bao gồm cả giải thuật và hiện thực chương trình máy tính) như sắp xếp, kiểm tra số nguyên tố, kiểm tra phép nhân ma trận, tìm cặp điểm gần nhau nhất, tìm cây bao trùm nhỏ nhất.

1.2. Những đóng góp chính của khóa luận

Dưới đây là những đóng góp chính của khóa luận đối với lĩnh vực thiết kế, phân tích và hiện thực ứng dụng giải thuật.

1. Nghiên cứu và phân loại các giải thuật ngẫu nhiên.
2. Nghiên cứu và giới thiệu các giải thuật ngẫu nhiên (mã giả và tính độ phức tạp) để giải các bài toán sắp xếp (Quicksort), kiểm tra phép nhân ma trận, kiểm tra số nguyên tố, tìm cặp điểm gần nhau nhất và tìm cây bao trùm nhỏ nhất.
3. Hiện thực một số giải thuật ngẫu nhiên một cách hiệu quả như một hệ thống để giải các bài toán đã nêu trong điểm 2.

1.3. 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 trong khóa luận. Mỗi chương tiếp theo, từ Chương 2 đến Chương 5 có một phần giới thiệu và một phần kết luận.

Chương 2 giới thiệu tổng quan về giải thuật. Đây là những vấn đề cơ bản, cốt lõi của lý thuyết giải thuật cổ điển như khái niệm giải thuật, đặc trưng cơ bản, các phương pháp thiết kế giải thuật và các kỹ thuật tính toán độ phức tạp giải thuật.

Chương 3 trình bày giải thuật ngẫu nhiên. Đó là các vấn đề liên quan đến khái niệm, tính chất, cách phân loại giải thuật ngẫu nhiên và các lĩnh vực ứng dụng giải thuật ngẫu nhiên.

Chương 4 trình bày một số ứng dụng giải thuật ngẫu nhiên để giải một số bài toán cụ thể như sắp xếp, kiểm tra số nguyên tố, kiểm tra phép nhân ma trận, tìm cặp điểm gần nhau nhất, tìm cây bao trùm nhỏ nhất.

Chương 5 giới thiệu quá trình hiện thực các giải thuật ngẫu nhiên để giải các bài toán đã nêu trong Chương 4 (bao gồm cả đồ thị biểu diễn độ phức tạp và thời gian thực sự trên máy tính).

Chương 6 là phần tổng kết và đề nghị các hướng nghiên cứu trong tương lai liên quan đến các vấn đề của khóa luận.

1.4. Qui ước ký hiệu và viết tắt

Các ký hiệu và qui ước chung sau đây được sử dụng trong suốt khóa luận này:

$\forall$: lượng từ với mọi

$\in$: quan hệ liên thuộc của một phần tử đối với một tập hợp

$\exists$: lượng từ tồn tại

O: O-lớn

Ω: Omega

Θ: Theta

$\subseteq$: quan hệ tập con

$\cap$: phép toán giao tập hợp

$\cup$: phép toán hợp tập hợp

$\leq$: quan hệ nhỏ hơn hoặc bằng trên trường các số thực

$\geq$: quan hệ lớn hơn hoặc bằng trên trường các số thực

Pr: hàm tính xác suất của các sự kiện

min: hàm tính giá trị nhỏ nhất của một tập các số thực

max: hàm tính giá trị lớn nhất của một tập các số thực

Chương 2

TỔNG QUAN VỀ GIẢI THUẬT

2.1. 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 ngẫu 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. Mục tiêu chính của các định lý và công thức toán học được nêu trong chương này là để áp dụng nên các chứng minh định lý và công thức không được trình bày. Chúng ta có thể tìm thấy các chứng minh đó trong các tài liệu tham khảo. Cuối cùng, Phần 2.6, là các kết luận đáng lưu ý của Chương này.

2.2. 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 [3, 8] như sau.

Định nghĩa 2.2.1 *Giải thuật* (algorithm) 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). Nói cách khác, 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 $a_1, a_2, ..., a_n$, trong đó n là một số tự nhiên, có thể được viết như sau.

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

1 **for** $i \leftarrow 1$ **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 giải thuật 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) [8] 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** $i \leftarrow 1$ **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.

2.3. Độ phức tạp của giải thuật

*Độ phức tạp của giải thuậ*t (complexity of algorithm) là độ đo 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 [3, 8].

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.

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 thuật 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, kích thước đầu vào 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 thước 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 $T(n) = \alpha + \beta + \gamma$.
2. Trường hợp xấu nhất xảy ra nếu $k \notin \{a[1], a[2], ..., a[n]\}$ khi đó $T(n) = (n+1)\alpha + n\beta + \gamma$.
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

$$T(n) = \Sigma_{i=1,..n}\,(i\alpha + i\beta + \gamma)p/n + ((n+1)\alpha + n\beta + \gamma))(1-p)$$

$$= (\alpha + \beta)(n+1)p/2 + \gamma p + ((n+1)\alpha + n\beta + \gamma))(1-p)$$

Lưu ý rằng khi p=1 (luôn luôn tìm thấy), thì $T(n)= (\alpha + \beta)(n+1)/2 + \gamma$.

Để biểu diễn độ phức tạp giải thuật một cách đơn giản, gần đúng, chúng ta dùng khái niệm tốc độ tăng của hàm [2, 3, 8] như định nghĩa sau.

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

1. $f(n)$ có tốc độ tăng (bậc) không quá $g(n)$ và viết $f(n = O(g(n))$ nếu có hằng số $c > 0$ và số nguyên dương N sao cho $f(n) \leq cg(n), \forall n \geq N$,
2. $f(n)$ có tốc độ tăng ít nhất là $g(n)$và viết $f(n) = \Omega(g(n))$ nếu có hằng số $c > 0$ và số nguyên dương N sao cho $f(n) \geq cg(n), \forall n \geq N$,
3. $f(n)$ có tốc độ tăng là $g(n)$ và viết $f(n = \Theta(g(n))$ nếu $f(n = O(g(n))$ và $f(n) = \Omega(g(n))$

Các ký hiệu O, Ω và Θ giọi là ký hiệu tiệm cận (asymptotic notation). Chúng ta dễ dàng thấy các kết quả được phát biểu bởi định lý sau.

Định lý 2.3.1 Giả sử $T_1(n) = O(f(n)), T_2(n) = O(g(n)$ và c là một hằng số thì:

1. $T_1(n) + T_2(n) = O(max(f(n), g(n))$
2. $T_1(n).T_2(n) = O(f(n).g(n))$
3. $cO(f(n)) = O(f(n))$
4. $O(c) \equiv O(1)$

Định lý có thể được phát biểu tương tự cho các ký hiệu Ω và Θ. Bây giờ thời gian chạy của một giải thuật có thể được định nghĩa theo các ký hiệu tiệm cận [8] như sau.

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

Chúng tôi lưu ý rằng, Định nghĩa 2.3.4 có thể được phát biểu cho các ký hiệu tiệm cận khác. Tốc độ tăng của thời gian chạy càng lớn thì giải thuật càng chậm (chẳng hạn, giải thuật có thời gian chạy $T(n) = O(n^2)$ hiệu quả hơn giải thuật có thời gian chạy $T(n) = O(n^3)$.

Ví dụ 2.3.2 Phân tích giải thuật tính gần đúng $e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!}$

```
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 α, γ lần lượt là thời gian thực hiện lệnh gán, trả về, thì

$$T(n) = \alpha + n\alpha + (1 + 2 + .. + n)\alpha + n\alpha + \gamma$$

$$= (2n + 1)\alpha + \frac{n(n+1)\alpha}{2} + \gamma = O(n^2)$$

Tốc độ tăng của độ phức tạp giải thuật biểu diễn theo O, Ω, Θ phụ thuộc vào số lần thực hiện của thao tác (lệnh) được thực thi nhiều lần nhất và có chi phí lớn nhất trong giải thuật. Các thao tác được thực thi nhiều lần nhất và 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 [3].

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(x, n)

1 s ←1

2 p ←1

3 **for** i ←1 to n

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

5 s ←$s + p$

6 **return** s

Rõ ràng lệnh cơ bản là x/i, nên $T(n) = nc = O(n)$, trong đó c là thời gian thực hiện của thao tác x/i.

2.4. 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 [2, 3, 8].

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 [3]. Đây là chiến lược dễ dàng áp dụng và được lựa chọn đầu tiên trong quá trình phân tích thiết 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 $S=1+2+\ldots+n$, 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 với độ phức tạp 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 [3, 8]. 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 $A[p..r]$ các số được sắp theo thứ tự tăng và một số K, tìm trong mảng số có giá trị bằng K.

```
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 $p \geq r$ và $T(n)=T(\lfloor n/2 \rfloor)+O(1)$, nếu $p<r$. Không mất tính tổng quát giả sử m là số nguyên sao cho $n=2^m$. Khi đó $T(n) \leq T(n/2)+1=T(n/2^2)+2=\ldots=T(n/2^m)+m$. Vậy $T(n) \leq T(1)+m=$ O(1)+$\log_2 n$=O($\log_2 n$).

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 [3]. Nghiệm (lời giải) của bài toán có được bằng cách kết hợp nghiệm của các bài toán con. Nghiệm 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 $c_1, c_2, \ldots, c_n$, hãy nhặt một số các đồng xu sao cho tổng giá trị là 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). Chúng ta xét hai trường hợp. Nếu đồng xu cuối cùng được chọn thì giá trị lớn nhất là c_n+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

$$F(n)=max\{c_n+ F(n-2), F(n-1)\} \text{ với } n > 1,$$

$$F(0)=0, F(1)=c_1$$

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 [3]. 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 [3].

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 $F \leftarrow \varnothing$; $Q \leftarrow E[G]$; $N \leftarrow V[G]$

3 **while** $|F| < |N| - 1$ and $Q \neq \varnothing$

4 **do** $e \leftarrow$ ExtractMin(Q) // e có trọng số bé nhất

5 **if** $F \cup \{e\}$ not contain cycle **then** $F \leftarrow F \cup \{e\}$

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 mỗi lần lặp thể hiện chiến lược tham lam trong giải thuật. Thời gian chạy của ExtractMin(Q) là $O(\log_2 E)$. Từ đó thời gian chạy của giải thuật này được xác định là $O(V \log_2 E)$.

Chiến lược biến đổi để trị

Chiến lược *biến đổi để trị* (transform-and-conquer) gồm hai tầng (giai đoạn) [3]:

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 đã được biến đổi.

Ví dụ 2.4.5 Xét bài toán tính tổng $S=1+2+\ldots+n$, với n là một số nguyên dương (trong Ví dụ 2.4.1). Áp dụng chiến lược biến đổi để trị biến đổi tổng $S=1+2+\ldots+n=n(n+1)/2$, 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 $x = (x_1, x_2, \dots, x_n)$ với x_i được chọn trong tập A_i nào đó. Quá trình tìm $x = (x_1, x_2, \dots, x_n)$ được thực hiện bằng cách tính toán mỗi thành phần. $x_1, x_2, \dots, x_n$ tại một thời điểm. Tại bước thứ *i*, vector con $x_1, x_2, \dots, x_n$ tìm được gọi là một nghiệm bộ phận của bài toán. Để tìm thành phần thứ *k* của nghiệm (thỏa một số ràng buộc nào đó), khi đã chọn được $k-1$ thành phần $x_1, x_2, \dots, x_{k-1}$ của x, giải thuật chọn (tính) x_k trong số các khả năng có thể có của nó trong A_k. Với mỗi khả năng *j*, giải thuật kiểm tra xem có chấp nhận được x_k không. Nếu chấp nhận thì xác định x_k theo *j*, khi k = n thì có một lời giải, ngược lại thì tiếp tục xác định x_{k+1} (thường bằng đệ qui). Nếu thử tất cả các khả năng $x_k \in A_k$ mà không có khả năng nào được chấp nhận thì quay lui lại bước trước để xác định lại x_{k-1} [3]. 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 [2, 3]. Lược đồ tổng quát cho giải thuật được thiết kế theo chiến lược nhánh cận.

BoundBranch(x[1..k], n) // developing local solution (a_1, a_2, …., a_{k-1})

1 **for** $a_k \in A_k$

2 **do if** accepting a_k

3 **then** $x_k \leftarrow a_k$

4 **if** $k = n$

5 **then** < update record>

6 **else if** $g(a_1, a_2, \ldots, a_k) \le f^*$

7 **then** BoundBranch(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).

2.5. 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 [7, 8] 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 $\lfloor x \rfloor$, là số nguyên lớn nhất nhỏ hơn *x*. Phần nguyên trần (ceiling) của số thực x, ký hiệu $\lceil x \rceil$, là các số nguyên nhỏ nhất lớn hơn *x*.

Một số hệ thức toán học cơ bản [7, 8] được ứng dụng để tính toán độ phức tạp của giải thuật:

1. Giả sử *x* là một số thực, thì $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$
2. Nếu *n* là số tự nhiên và *a, b* là các số thực dương thì $a^{\log_b n} = n^{\log_b a}$
3. Với mọi $x \neq 1$ thì $\sum_{k=0,n} x^k = \frac{x^{n+1}-1}{x-1}$
4. $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1,n} \frac{1}{k} = \ln n + O(1)$

   (H_n 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 số định lý cơ bản về hệ thức truy hồi [7].

Định nghĩa 2.5.2 Hệ thức truy hồi đối với dãy $\{a_n\}$ là một phương trình có dạng

$$a_n = f(a_{n-1}, a_{n-2}, \ldots, a_{n-k}), \forall n \geq n_0 \geq 0.$$

Ví dụ 2.5.1 Hệ thức truy hồi $a_n = 2a_{n-1} - a_{n-2}, \forall n \geq 2$.

Dãy $\{a_n\}$ 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 $a_n = 3n$ 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 có dạng

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}, n \geq k \quad (1)$$

Với $c_1, c_2, \ldots, c_k$ là các hằng số thực, $c_k \neq 0$

Định lý 2.5.1 Giả sử $c_1, c_2, \ldots, c_k$ là các hằng số, phương trình $r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_k = 0$ có k nghiệm phân biệt. Khi đó dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ nếu và chỉ nếu $a_n = \alpha_1 {r_1}^n + \alpha_2 {r_2}^n + \cdots + \alpha_k {r_k}^n$ với n =0,1,2,… Trong đó $\alpha_1, \alpha_2, \ldots, \alpha_k$ là các hằng số.

Định lý 2.5.2 Giả sử $c_1, c_2, \ldots, c_k$ là các hằng số, phương trình $r^k - c_1 r^{k-1} - \ldots - c_k = 0$ có t nghiệm $r_1, r_2, \ldots, r_t$ bội tương ứng $m_1, m_2, \ldots, m_t$. Khi đó dãy $\{a_n\}$ là nghiệm của hệ thức truy hồi $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \ldots + c_k a_{n-k}$ nếu và chỉ nếu $a_n = (\alpha_{1,0} + \alpha_{1,1} n + \ldots + \alpha_{1,m_1-1} n^{m_1-1}) r_1^n + (\alpha_{2,0} + \alpha_{2,1} n + \ldots + \alpha_{2,m_2-1} n^{m_2-1}) r_2^n + \ldots + (\alpha_{t,0} + \alpha_{t,1} n + \ldots + \alpha_{t,m_t-1} n^{m_t-1}) r_t^n$ với n= 0, 1, 2, ... Trong đó $\alpha_{i,j}$, i=0,..t, j=0…,m_t -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 có dạng

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n), n \geq k$$

với $c_1, c_2, \ldots, c_k$ là các hằng số thực, $c_k \neq 0$, $F(n)$ là một hàm chỉ phụ thuộc vào n.

Ví dụ 2.5.2 Hệ thức $a_n = 3a_{n-1} + 2n$ là một hệ thức truy hồi tuyến tính không thuần nhất bậc 1.

Định lý 2.5.4 Nếu $\{{a_n}^{(p)}\}$ là một nghiệm của hệ thức truy hồi tuyến tính không thuần nhất $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n)$, thì mọi nghiệm của nó có dạng $\{{a_n}^{(p)} + {a_n}^{(h)}\}$, trong đó ${a_n}^{(h)}$ là nghiệm của hệ thức truy hồi tuyến tính $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$.

Định lý 2.5.5 Giả sử $\{a_n\}$ thỏa mãn hệ thức truy hồi tuyến tính không thuần nhất $a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k} + F(n)$ (1), với $F(n) = (b_tn^t + b_{t-1}n^{t-1} + \cdots + b_1n + b_0)s^n$, t là một số nguyên không âm. Khi đó

1. Nếu s không phải là nghiệm của phương trình $r^k - c_1r^{k-1} - c_2r^{k-2} - \cdots - c_k = 0$ thì (1) có một nghiệm dạng $(p_tn^t + p_{t-1}n^{t-1} + \cdots + p_1n + p_0)s^n$
2. Nếu s là một nghiệm bội m của phương trình $r^k - c_1r^{k-1} - c_2r^{k-2} - \cdots - c_k = 0$ thì (1) có một nghiệm dạng $n^m(p_tn^t + p_{t-1}n^{t-1} + \cdots + p_1n + p_0)s^n$

Định nghĩa 2.5.5 Hệ thức truy hồi dạng $f(n) = af(\frac{n}{b}) + g(n)$, 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 $f(n) = 2f(\frac{n}{2}) + n$ là một hệ thức truy hồi chia để trị.

Định lý 2.5.6 Giả sử $f(n)$ là một hàm tăng thỏa mãn hệ thức truy hồi $f(n) = af(\frac{n}{b}) + c$ khi $n : b, a \geq 1, b$ là số nguyên lớn hơn 1, c là một số thực dương , thì

$$f(n) = \begin{cases} O\left(n^{\log_b a}\right), a > 1 \\ O(\log_2 n), a = 1 \end{cases}$$

Định lý 2.5.7 Giả sử $f(n)$ là một hàm tăng thỏa mãn hệ thức truy hồi $f(n) = af(\frac{n}{b}) + cn^d$. khi $n=b^k, a \geq 1, b$ là số nguyên lớn hơn 1, c và d là một số thực dương, thì

$$f(n) = \begin{cases} O(n^d), a < b^d \\ O(n^d \log_2 n), a = b^d \\ O\left(n^{\log_b a}\right), a > b^d \end{cases}$$

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 [7, 8] được ứng dụng để thiết kế và phân tích giải thuật ngẫu nhiên trong các chương sau.

Định nghĩa 2.5.6 Nếu S là một tập khác rỗng (gọi là không gian mẫu) 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 $\Pr(E)=\frac{|E|}{|S|}$.

Chúng tôi lưu ý rằng một tập con E của S gọi là một sự kiện, một phần tử $s\in S$, 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 mẫu đến tập các số thực R, $\Pr: S\to R$, thỏa các tiên đề sau:

1. $\Pr(A)\geq 0$, $\forall A\subset S$.
2. $\Pr(S)=1$.
3. $\Pr(A\cup B)=\Pr(A)+\Pr(B)$ với hai sự kiện bất kỳ A, B xung khắc, loại trừ lẫn nhau (mutually exclusive).

Từ các tiên đề ta suy ra:

1. $\Pr(\varnothing)=0$
2. $\Pr(\bar{A}) = 1 - \Pr(A)$
3. $\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B)\leq \Pr(A) + \Pr(B)$.

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 mẫu rời rạc. Nếu S là không gian mẫu, thì $\Pr(A)=\Sigma_{s\in A}\Pr(\{s\})$, $\forall A\subset S$. 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 ngửa (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 cố (xâu) có đúng *k* mặt sấp xuất hiện. Khi đó $|A|=C_n^k$, nên $\Pr(A)= C_n^k/|S|$.

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

$$\Pr(A|B) = \frac{\Pr(A \cap B)}{\Pr(B)} ; \Pr(B) \neq 0$$

Từ định nghĩa trên ta có các hệ thức sau:

1. $\Pr(A \cap B) = \Pr(A).\Pr(B/\mathrm{A}) = \Pr(B).\Pr(A/B)$.
2. $\Pr(A \cap B) = \Pr(A).\mathrm{P}(B)$, nếu *A* và *B* độc lập.

Một hệ các sự kiện $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_\mathrm{n}$ được gọi là đầy đủ nếu chúng đôi một xung khắc và $\mathrm{A}_1 \cup \mathrm{A}_2 \cup \ldots \cup \mathrm{A}_\mathrm{n} = \mathrm{S}$.

Định lý 2.5.8 Nếu các biến cố $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_\mathrm{n}$ là một hệ đầy đủ các biến cố. *H* là một biến cố trong không gian mẫu. Khi đó

$$\Pr(H) = \Pr(A_1).\Pr(H/A_1) + \Pr(A_2).\Pr(H/A_2) + \cdots + \Pr(A_2).\Pr(H/A_n).$$

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

$$\Pr(A|B) = \frac{\Pr(A)\,.\Pr(B|A)}{\Pr(A)\,.\Pr(B|A) + \Pr(\bar{A})\,.\Pr(B|\bar{A})}$$

Định nghĩa 2.5.5 Một *biến ngẫu nhiên* (random variable) là một hàm $X: S \to R$ từ không gian mẫu *S* đến tập các số thực *R*.

Với mỗi biến ngẫu nhiên *X* và số thực *x*, sự kiện $X=x$ được định nghĩa là $\{s \in S: X(s)=x\}$. Khi đó $\Pr(X=x)=\Sigma_{s \in S:\, X(s)=x} \Pr(s)$.

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 ngẫu nhiên *X* được định nghĩa là giá 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 ngẫu nhiên *X* được định nghĩa là $E(X)=\Sigma_{x\in R}\, x.\Pr(X=x)$.

Từ định nghĩa này chúng ta suy ra rằng $E(X)=\Sigma_{s\in S}\Pr(s).X(s)$. 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 [8] của giải thuật.

Ví dụ 2.5.6 Giả sử *X* là biến ngẫu 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 $E(X)=1/6(1+2+\ldots+6)= 21/6$ là số trung bình trên mặt con xúc sắc xuất hiện khi tung nó.

2.6. 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 ngẫu nhiên trong Chương 3 tiếp theo.

Chương 3

GIẢI THUẬT NGẪU NHIÊN

3.1. Giới thiệu

Chương này trình bày 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.

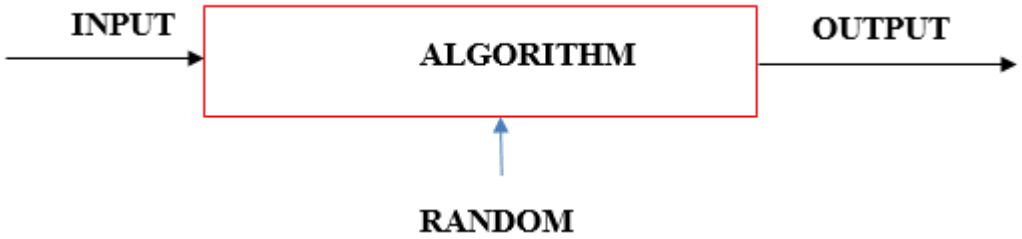
3.2. Định nghĩa và ví dụ giải thuật ngẫu nhiên

Giải thuật ngẫu nhiên đầu tiên được nghiên cứu và đề nghị bởi Michael O. Rabin năm 1976 khi giải bài toán tìm cặp điểm gần nhất với độ phức tạp tuyến tính [4]. Việc nghiên cứu các giải thuật ngẫu nhiên được thúc đẩy bằng việc kiểm tra số nguyên tố ngẫu nhiên vào năm 1977 bởi Solovay và Strassen [10]. Kể từ đó nhiều bài toán trong khoa học và thực tiễn đã được giải bằng các giải thuật ngẫu nhiên hiệu quả hơn giải thuật cổ điển.

Không như giải thuật truyền thống, mỗi bước luôn được xác định một cách chắc chắn, trong giải thuật ngẫu nhiên, mỗi bước có thể được lựa chọn ngẫu nhiên phụ thuộc vào dữ liệu có tính ngẫu nhiên mà giải thuật được cung cấp.

Hình 3.2.1 và Hình 3.2.2 dưới đây là sự mô tả về sự khác biệt giữa giải thuật cổ điển và giải thuật ngẫu nhiên.

Hình 3.2.1: Sơ đồ giải thuật xác định



Hình 3.2.2: Sơ đồ giải thuật ngẫu nhiên

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

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 ngẫu 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 có thể cho 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 một mảng. Cho một dãy $\boldsymbol{n \geq 2}$ 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 [10].

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: $\Pr[find\ a] = 1 - (\frac{1}{2})^k$

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 đó thời gian chạy là $O(k)$.

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 $O(1)$.

3.3. 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 ngẫu 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 ngẫu 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 [5, 10].

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 [10].

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 ngẫu 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* [8], 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 [5]. 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.

3.4. Các lĩnh vực áp dụng của giải thuật ngẫu nhiên

Giải thuật ngẫu 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 ngẫu 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 mẫu, 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 ghép cặp cực đại.

3.5. Kết luận

Trong Chương 3 này, các khái niệm cơ bản về giải thuật ngẫu 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 ngẫu 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 ngẫu nhiên được ứng dụng nhiều trong thực tế để cải thiện hiệu suất tính toán. Chương 4 kế tiếp, một số giải thuật ngẫu 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

ỨNG DỤNG GIẢI THUẬT NGẪU NHIÊN

4.1. Giới thiệu

Chương này trình bày một số giải thuật ngẫu nhiên được ứng dụng để giải một số bài toán tiêu biểu, thường gặp với độ phức tạp về thời gian nhỏ hơn các giải thuật cổ điển. Đầu tiên, Phần 4.2 là giải thuật ngẫu nhiên (Random-Quicksort) để giải bài toán sắp xếp. Kế đến, Phần 4.3 là một giải thuật giải quyết bài toán xác định số nguyên tố. Tiếp theo, Phần 4.4 là giải thuật ngẫu nhiên để giải bài toán tìm cặp điểm gần nhất trong không gian hai chiều. Phần 4.5 giới thiệu giải thuật ngẫu nhiên kiểm tra phép nhân ma trận. Phần 4.6 là giải thuật ngẫu nhiên tìm cây bao trùm nhỏ nhất của đồ thị liên thông có trọng số. Như trong Chương 2, để đơn giản, một số chứng minh bổ đề và định lý trong chương này không được trình bày. Chúng ta có thể tìm thấy các chứng minh đó trong các tài liệu tham khảo. Cuối cùng, Phần 4.7 là một số kết luận của chương này.

4.2. Sắp xếp dãy số

Giải thuật Quicksort (cổ điển) áp dụng chiến lược chia để trị để sắp xếp và có độ phức tạp trung bình là O($n\log_2 n$), tuy nhiên độ phức tạp trong trường hợp xấu nhất của giải thuật này là O(n^2). Hạn chế này được khắc phục bằng giải thuật Quicksort ngẫu nhiên [8].

Trước khi trình bày giải thuật Quicksort ngẫu nhiên, chúng tôi giới thiệu giải thuật Quicksort cổ điển [8] làm cơ sở để thiết kế giải thuật ngẫu nhiên này như sau.

```
Quicksort(A,p,r)
    1  if p<r then
    2      q=Partition(A,p,r)
    3      Quicksort(A,p,q-1)
    4      Quicksort(A,q+1,r)
Partition(A,p,r)
    1  x = A[r]
    2  i = p − 1
    3  for j = p to r − 1
    4      if A[j] <= x then
    5          i =i+1
    6          exchange A[i] with A[j]
    7  exchange A[i+1] with A[r]
    8  return i+1
```

Giải thuật Quicksort cổ điển luôn luôn chọn phần tử *chốt* (pivot) tại vị trí cố định (trong phiên bản này là phần tử cuối mảng). Như trong [8] đã phân tích, độ phức tạp xấu nhất là O(n^2), trường hợp này xảy ra khi trong tất cả các lời gọi đệ qui, mảng các phần tử cần sắp luôn luôn được phân hoạch thành hai phần có kích thước là n-1 và 0.

Trong phiên bản giải thuật ngẫu nhiên [8], chúng ta giả sử các phần tử của mảng là phân biệt và sẽ chọn ngẫu nhiên một phần tử trong $A[p...r]$ 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 kỳ vọng sự phân hoạch của mảng input được cân bằng khá tốt để hầu như không rơi vào trường hợp xấu nhất của giải thuật Quicksort cổ điển. Giải thuật Quicksort ngẫu nhiên được thiết kế bằng cách chọn *ngẫu nhiên* (random) phần tử chốt như sau [8].

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

Randomized-Quicksort(A,p,r)

1 **if** $p<r$ **then**
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

Để phân tích độ phức tạp của giải thuật Randomized-Quicksort, trước hết chúng ta chứng minh bổ đề về độ phức tạp của Quicksort [8] sau đây.

Bổ đề 4.2.1 Giả sử X là số phép so sánh thực hiện ở dòng 4 của hàm Partition trong quá trình thực thi của Quicksort trên một mảng n phần tử, thì thời gian chạy của Quicksort là $O(n+X)$.

Chứng minh: Hàm Partition được gọi nhiều nhất là n lần trong Quicksort, chi phí thời gian mỗi lần thực hiện là một hằng số cộng thêm chi phí thời gian của vòng lặp **for**, thao tác hoán vị dòng 7 và trả về dòng 8. Trong mỗi lần lặp của vòng lặp **for** câu

lệnh ở dòng 4 luôn luôn được thực hiện. Vì vậy, tổng thời gian của Quicksort là $c(n+X)$ trong đó c là một hằng số, X là tổng số lần so sánh trong dòng 4.

Để tiện lợi khi phân tích Randomized-Quicksort, chúng ta đổi tên các phần tử của mảng A thành $z_1, z_2, \ldots, z_n$ với z_i là phần tử nhỏ nhất thứ i. Gọi $Z_{ij}=\{z_i, z_{i+1}, \ldots, z_j\}$ là tập các phần tử giữa z_i và z_j và bao gồm cả z_i, z_j.

Giả sử $X_{ij}=\mathrm{I}\{z_i$ được so sánh với $z_j\}$, là *biến ngẫu nhiên chỉ định* (indicator random variable) kết hợp với sự kiện "z_i được so sánh với z_j". Lưu ý trong Partition mỗi cặp phần tử chỉ được so sánh một lần, khi đó tổng số lần so sánh X được xác định

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

và
$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E\left[X_{ij}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ được} c \text{ so sánh với } z_j\}$$

Do bất cứ phần tử nào trong Z_{ij} cũng có thể là phần tử đầu tiên được chọn làm chốt. Bởi vì tập Z_{ij} có $j-i+1$ phần tử, và phần tử 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 $1/(j-i+1)$. Do đó, chúng ta có:

$$\Pr\{z_i \text{ được so sánh với } z_j\} = \Pr\{z_i \text{ hoặc } z_j \text{ là chốt đầu tiên trong } Z_{ij}\}$$
$$= \Pr\{z_i \text{ là chốt đầu tiên trong } Z_{ij}\} + \Pr\{z_j \text{ là chốt đầu tiên trong } Z_{ij}\}$$
$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

Nên $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$ (vì $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ được so sánh với } z_j\}$)

Bằng cách thay biến ($k = j - i$) và sử dụng Harmonic Number (xem Phần 2.5), chúng ta có thể đánh giá tổng trên như sau:

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1} \mathrm{O}(\lg n) = \mathrm{O}(n \lg n)$$

Do đó, chúng ta có thể kết luận rằng, khi sử dụng Randomized-Partition, thời gian chạy kỳ vọng của Quicksort là $O(n \lg n)$.

Trong giải thuật Randomized-Quicksort, sử dụng Randomized-Partition, phần tử chốt được chọn ngẫu nhiên nên xác suất xảy ra trường hợp xấu nhất như trong Quicksort là rất nhỏ. Khi đó thời gian chạy của giải thuật Randomized-Quicksort là thời gian chạy kỳ vọng của giải thuật Quicksort như đã nêu ở trên. Vì vậy, chúng ta có định lý sau.

Định lý 4.2.1 Thời gian chạy của giải thuật Randomized-Quicksort là $O(n \lg n)$.

Giải thuật Randomized-Quicksort là loại giải thuật Las Vegas.

4.3. Xác định số nguyên tố

Bài toán xác định một nguyên dương *n* có là nguyên tố hay không là bài toán không có giải thuật đa thức cho đến năm 2004 [2]. Giải thuật kinh điển để giải bài toán này dựa trên việc kiểm tra *n* có chia hết cho các số từ 2 đến $\sqrt{n}$ hay không. Giải thuật ngẫu nhiên sẽ giảm thời gian tính toán bằng cách kiểm tra một cách ngẫu nhiên một số lần để xác định tính nguyên tố của *n* dựa trên định lý Fermat [1, 6] sau đây.

Định lý 4.3.1 Nếu *n* là một số nguyên tố, thì với mọi $a \in Z_n^*$ phải thỏa mãn $a^{n-1} \equiv 1(mod\ n)$. (tương đương $a^{n-1} mod\ n = 1 <=> a^n mod\ n = a$).

Trong giải thuật ngẫu nhiên sau đây, số lần kiểm tra tính nguyên tố của *n* được chọn là *m* sao cho $m \geq \left\lceil \log_2(\frac{1}{\varepsilon}) \right\rceil$ với ε đủ bé. Giải thuật sẽ cho kết quả đúng *n* là số nguyên tố với xác suất đủ lớn.

Mã giả

Input: Một số nguyên *n* >1, và một tham số *m*, với $m \geq \left\lceil \log_2(\frac{1}{\varepsilon}) \right\rceil$.

Output: *n* là số nguyên tố hay hợp số, với xác suất đúng là $1 - \varepsilon = 1 - 2^{-m}$.

```
PrimeNumberTest(n,m)
1   if n mod 2 = 0
2      then return false
3      else  k=(n-1)/2
4   for i=0 to m do
5        b= Random(2,n-1)
6        kq= Power(a,n) mod n
7        if kq≠a
8            then return false
9   return true
```

Trong đó Power(a, n) là hàm tính a^n. Khi thủ tục trên trả về false thì số *n* là hợp số, ngược lại nếu trả về true thì *n* là số nguyên tố.

Ví dụ 4.3.1

1) *n*=12, *m*=3, lần kiểm tra thứ nhất a_1=9, $a^n mod\ n = 9^{12} mod\ 12 = 9, 9 = a$, tiếp tục kiểm tra lần thứ 2. Lần kiểm tra thứ hai a_2=2, $a^n mod\ n = 2^{12} mod\ 12 = 4, 4 \neq a$ nên 12 là hợp số.
2) *n*=13, *m*=3, lần kiểm tra thứ nhất a_1=9, $a^n mod\ n = 9^{13} mod\ 13 = 9, 9 = a$ tiếp tục kiểm tra lần thứ 2. Lần kiểm tra thứ hai a_2=10, $a^n mod\ n = 10^{13} mod\ 13 = 10, 10 = a$, tiếp tục kiểm tra lần thứ 3. Lần kiểm tra thứ ba a_3=8, $a^n mod\ n = 8^{13} mod\ 13 = 8,8 = a$. 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 giải thuật Power(*a*,*n*) thì độ phức tạp của toàn bộ giải thuật là O(*k.t*). Giải thuật Power(*a*,*n*) được thiết kế theo chiến lược chia để trị kết hợp với qui hoạch động như sau.

Power(a,n)

1. **if** n =0
2. **then return** 1;
3. u = Power(a, $\lfloor n / 2 \rfloor$);
4. **if** n mod 2 = 1
5. **then return** $u * u * a$;
6. **else return** $u * u$;

Độ phức tạp của Power là $\begin{cases} O(1) = 1 \\ O(n) = O\left(\frac{n}{2}\right) + O(1) \end{cases}$

Áp dụng Định lý 2.5.6 ta có $O(n) = O(log_2 n)$

Như vậy độ phức tạp của toàn bộ giải thuật là $O(k.\log_2 n)$, tốt hơn so với độ phức tạp của giải thuật cổ điển kiểm tra số nguyên tố là $O(\sqrt{n})$. Với k =10 thì giải thuật ngẫu nhiên cho kết quả đúng với xác suất cao, và có độ phức tạp thấp hơn nhiều so với giải thuật cổ điển khi n lớn. Giải thuật này là loại giải thuật Monte Carlo.

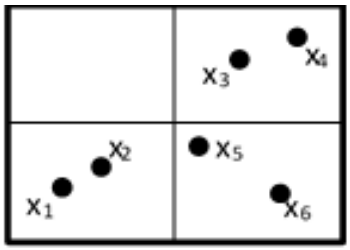
4.4. Tìm cặp điểm gần nhất

Bài toán tìm cặp điểm gần nhất trong một tập điểm của không gian hai chiều có thể giải quyết bằng giải thuật chia để trị cổ điển với độ phức tạp là $O(n\,log_2 n)$ [3]. Trong phần này, chúng tôi sẽ chỉ ra một giải thuật ngẫu nhiên với độ phức tạp là $O(n)$ để giải quyết bài toán tìm cặp điểm gần nhất [2].

Gọi x_1,x_2,….x_n là n điểm trên mặt phẳng. Yêu cầu của bài toán là tìm cặp điểm x_i và x_j mà khoảng cách giữa x_i và x_j là nhỏ nhất trong toàn bộ các cặp điểm. Với phương pháp tiếp cận trực tiếp dùng để giải, cần tính $n(n-1)/2$ khoảng cách sau đó tìm con số nhỏ nhất trong tất cả khoảng cách này, thời gian thực hiện là $O(n^2)$.

Ý tưởng chính của giải thuật ngẫu nhiên cho bài toán này là dựa trên sự quan sát sau: Nếu hai điểm x_i và x_j có khoảng cách giữa chúng được dự đoán không là ngắn nhất thì có thể bỏ qua, chỉ quan tâm đến các cặp gần nhau hơn. Với ý tưởng này, giải thuật ngẫu nhiên trước hết phân vùng các điểm thành một vài cụm theo một cách nào

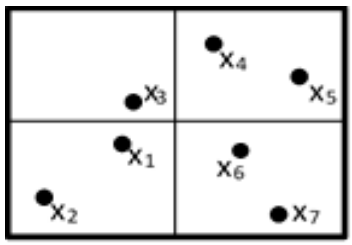
đó mà ở mỗi cụm có các điểm gần nhau. Sau đó chúng ta tính toán khoảng cách giữa các điểm trong cùng một cụm. Xét các điểm trong Hình 4.4.1:



Hình 4.4.1: Phân vùng các điểm

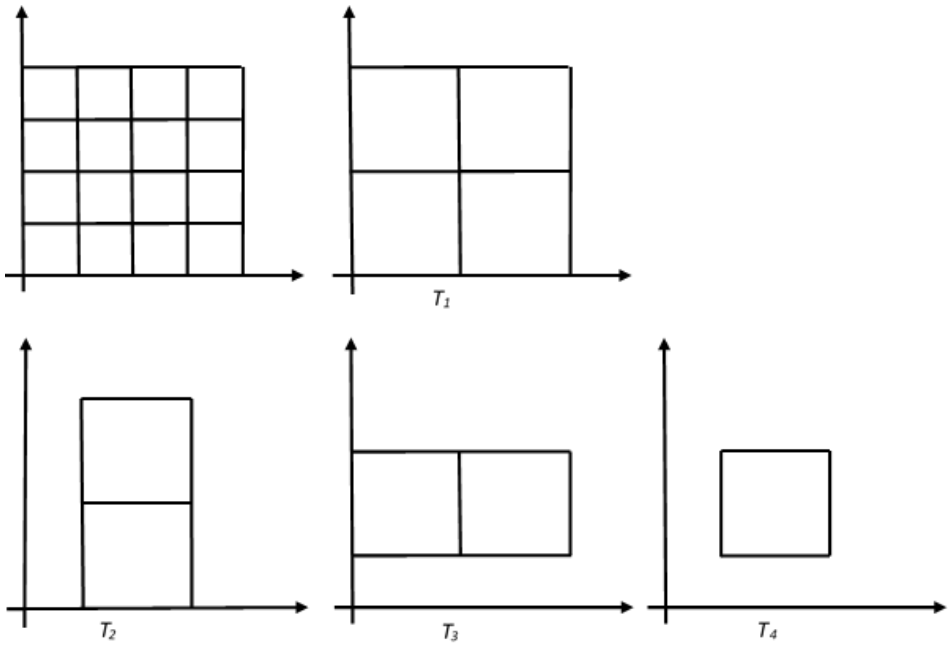
Nếu chúng ta chia vùng cho sáu điểm ở Hình 4.4.1 thành ba cụm $S_1=\{x_1,x_2\}$, $S_2=\{x_3,x_4\}$, $S_3=\{x_5,x_6\}$, sau đó chúng ta chỉ tính toán ba khoảng cách $d(x_1,x_2)$, $d(x_3,x_4)$, $d(x_5,x_6)$, cuối cùng chúng ta tìm giá trị nhỏ nhất trong ba khoảng cách đó. Ngược lại, nếu chúng ta không chia các điểm thành các cụm, số khoảng cách phải tính là $(6 \text{ x } 5)/2 = 15$ khoảng cách.

Dĩ nhiên, mô tả này có thể 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. Tuy nhiên, nó có một quá trình phân chia nhưng không có quá trình hợp nhất. Xét Hình 4.4.2



Hình 4.4.2: Phân vùng các điểm hình vuông với cạnh δ

Chúng ta có thể thấy cặp điểm gần nhất trong Hình 4.4.2 là $\{x_1,x_3\}$. 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 đặt ra là mạng lưới kích thước δ như thế nào thì 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 nhất trong tập đó. Khoảng cách ngắn nhất này sẽ trở thành δ.

Giải thuật

Input: Một tập S gồm n điểm $x_1, x_2, \ldots, x_n$ với $S \subseteq R^2$.

Output: Cặp điểm gần nhất trong tập S.

Bước 1: Chọn ngẫu nhiên một tập điểm $S_1=\{x_{i1}, x_{i2}, \ldots, x_{im}\}$ với $m = n^{\frac{2}{3}}$. Tìm cặp điểm gần nhất trong tập S_1 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 T_1, T_2, T_3 và T_4 bắt nguồn từ T bằng cách nhân đôi kích thước cạnh hình vuông thành 2δ.

Bước 4: Đối với mỗi T_i, xác định các tập điểm $S_j^{(i)}$ là giao giữa tập điểm S và các hình vuông của T_i, sao cho $S = S_1^{(i)} \cup S_2^{(i)} \cup \ldots \cup S_j^i, 1 \leq i < 4$

Bước 5: Đối với mỗi cặp $x_p, x_q \in S_j^{(i)}$, tính $d(x_p, x_q)$. Đặt x_a và x_b là cặp điểm có khoảng cách ngắn nhất trong tất cả các cặp điểm. Trả về x_a và x_b là cặp điểm gần nhất.

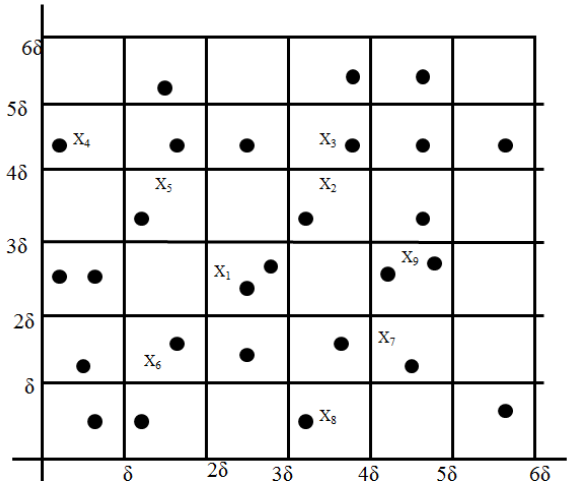
Ví dụ 4.4.1 Cho tập điểm S có 27 điểm, được mô tả trong Hình 4.4.4. Bước 1, chúng ta chọn ngẫu nhiên $\mathbf{27^{\frac{2}{3}} = 9}$ điểm, được đánh dấu là $x_1, x_2, \ldots, x_9$. Nhìn vào Hình 4.4.4, ta có thể thấy rằng cặp điểm gần nhau nhất là (x_1, x_2). Gán khoảng cách giữa x_1 và x_2 cho δ và 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 T_1, T_2, T_3, T_4 như yêu cầu ở bước 3:

$T_1 = \{[0: 2\delta, 0: 2\delta], [2\delta: 4\delta, 0: 2\delta], [4\delta: 6\delta, 0: 2\delta], \ldots, [4\delta: 6\delta, 4\delta: 6\delta]\}.$

$T_2 = \{[\delta: 3\delta, 0: 2\delta], [3\delta: 5\delta, 0: 2\delta], \ldots, [3\delta: 5\delta, 4\delta: 6\delta]\}.$

$T_3 = \{[0: 2\delta, \delta: 3\delta], [2\delta: 4\delta, \delta: 3\delta], [4\delta: 6\delta, \delta: 3\delta], \ldots, [4\delta: 6\delta, 3\delta: 5\delta]\}.$

$T_4 = \{[\delta: 3\delta, \delta : 3\delta], [3\delta: 5\delta, \delta: 3\delta], \ldots, [3\delta: 5\delta, 3\delta: 5\delta]\}.$



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:

$N(T_1): C_4^2 + C_3^2 + C_2^2 + C_3^2 + C_3^2 + C_3^2 + C_3^2 + C_3^2 = 28.$

$N(T_2): C_3^2 + C_3^2 + C_2^2 + C_5^2 + C_3^2 + C_4^2 = 26.$

$N(T_3): C_4^2 + C_3^2 + C_3^2 + C_3^2 + C_4^2 + C_3^2 = 24.$

$N(T_4): C_3^2 + C_4^2 + C_3^2 + C_5^2 = 22.$

Trong $28 + 26 + 24 + 22 = 100$ cặp điểm trên, cặp điểm gần nhất được tìm thấy tại ô vuông $[3\delta: 5\delta, 3\delta: 5\delta]$.

Phân tích độ phức tạp giải thuật

Bước 1 trong giải thuật là tìm cặp điểm gần nhất trong $n^{\frac{2}{3}}$ điểm. Khoảng các ngắn nhất này có thể được tìm bằng cách gọi đệ quy giải thuật. Nghĩa là, chọn ngẫu nhiên $(n^{\frac{2}{3}})^{\frac{2}{3}} = n^{\frac{4}{9}}$ từ $n^{\frac{2}{3}}$ điểm ban đầu, sau đó dùng chiến lược trực tiếp để tìm cặp điểm gần nhất trong $n^{\frac{4}{9}}$ điểm, với số phép tính khoảng cách cần thực hiện là $(n^{\frac{4}{9}})^2 = n^{\frac{8}{9}} < n$.

Bước 2 và Bước 3 có thể hoàn thành với O*(n)*. Trong Bước 4, có thể xác định nhanh hình vuông mà một điểm thuộc về nó bằng kỹ thuật băm (hashing). Từ đó Bước 4 có thể thực hiện với độ phức tạp O*(n)*.

Số phép tính khoảng cách ở Bước 5 là O(n) với xác suất $1 - 2e^{-cn^{\frac{1}{6}}}$, trong đó c là một hằng số, e là số Napier [2].

Do đó, chúng ta có thể kết luận rằng thời gian chạy của giải thuật ngẫu nhiên tìm cặp điểm gần nhất là O*(n)* với xác suất $1 - 2e^{-cn^{\frac{1}{6}}}$. Như vậy, giải thuật ngẫu nhiên tốt hơn nhiều so giải thuật cổ điển tốt nhất (chia để trị) với độ phức tạp $O(n \log_2 n)$. Giải thuật ngẫu nhiên tìm cặp điểm gần nhau nhất là loại giải thuật Las Vegas.

4.5. Kiểm tra phép nhân ma trận

Giải thuật Freivalds (đặt tên theo Rusins Freivalds) [9] là một giải thuật ngẫu nhiên được sử dụng để kiểm tra phép nhân ma trận. Cho ba ma trận *A*, *B* và *C* kích

thước n x n, kiểm tra liệu A x $B = C$. Một giải thuật thông thường có thể tính A x B sau đó so sánh kết quả với C. Tuy nhiên, ngay cả khi dùng một giải thuật nhân ma trận tốt nhất, thời gian chạy cũng không thể nhỏ hơn $O(n^{2.3727})$. Giải thuật Freivalds sử dụng sự ngẫu nhiên để giảm thời gian xuống $O(n^2)$ với một xác suất cao [9].

Giải thuật

Input: Matrix $A \in R^{m \times p}$, $B \in R^{p \times n}$, $C \in R^{m \times n}$ and $k \in N$

Output: True if $C = A \cdot B$; false if $C \neq A \cdot B$

MatrixEqualityTest(A, B, C, k)

1 **for** i=1 **to** k **do**

2 Choose $r=(r_1,...,r_n) \in \{0,1\}^n$ at random

3 Compute $C \cdot r$ and $A \cdot (B \cdot r)$

4 **if** $C \cdot r \neq A \cdot (B \cdot r)$

5 **then return** false

6 $i = i + 1$

7 **return** true

Định lý 4.5.1 Giải thuật chạy cho kết quả chính xác (correct) với xác suất $1-(1/2)^k$.

Thật vậy nếu $A.B \neq C$ thì $D = A.B - C \neq 0$. Không mất tính tổng quát, giả sử $d_{11} \neq 0$. Mặt khác, $\Pr(A.B.r = C.r) = \Pr((A.B-C).r = 0) = \Pr(D.r = 0)$. Nếu $D.r = 0$ thì phần tử đầu tiên của $D.r$ bằng 0. Nghĩa là $\Sigma_{j=1,n} d_{1j}r_j = 0$. Vì $d_{11} \neq 0$ nên ta có

$$r_1 = -\frac{\Sigma_{j=2}^{n} d_{1j} r_j}{d_{11}}$$

Nếu cố định tất cả r_j trừ r_1, thì đẳng thức trên thỏa nhiều nhất là hai lựa chọn $r_1 \in \{0, 1\}$. Vì vậy $\Pr(A.B.r = C.r) \leq 1/2$.

Vòng lặp thực hiện k lần. Nếu $C=A.B$ thì giải thuật luôn luôn cho kết quả đúng. Nếu $C \neq A.B$, giải thuật cho kết quả đúng với xác suất ít nhất là $1-(1/2)^k$. Vậy định lý được chứng minh.

Phân tích độ phức tạp giải thuật

Chi phí dòng 2 là O(n), dòng 3, 4 là O(n^2), từ đó thời gian chạy của giải thuật là O(kn^2).

Giải thuật ngẫu nhiên kiểm tra phép nhân của ma trận là giải thuật Monte Carlo.

Ví dụ 4.5.1 Giả sử có một yêu cầu xác định đẳng thức ma trận

$$AB = \begin{bmatrix}2 & 3\\ 3 & 4\end{bmatrix}\begin{bmatrix}1 & 0\\ 1 & 2\end{bmatrix} \stackrel{?}{=} \begin{bmatrix}6 & 5\\ 8 & 7\end{bmatrix} = C.$$

Chọn ngẫu nhiên một ma trận $r = \begin{bmatrix}1\\ 1\end{bmatrix}$, tính toán theo dòng 3 của giải thuật. Ta có

$$A\,.\,(Br) - C.r = \begin{bmatrix}2 & 3\\ 3 & 4\end{bmatrix}\left(\begin{bmatrix}1 & 0\\ 1 & 2\end{bmatrix}\begin{bmatrix}1\\ 1\end{bmatrix}\right) - \begin{bmatrix}6 & 5\\ 8 & 7\end{bmatrix}\begin{bmatrix}1\\ 1\end{bmatrix}$$

$$= \begin{bmatrix}2 & 3\\ 3 & 4\end{bmatrix}\begin{bmatrix}1\\ 3\end{bmatrix} - \begin{bmatrix}11\\ 15\end{bmatrix} = \begin{bmatrix}11\\ 15\end{bmatrix} - \begin{bmatrix}11\\ 15\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix}$$

Ta được kết quả là ma trận 0. Tuy nhiên, nếu trong lần thử thứ hai với $r = \begin{bmatrix}1\\ 0\end{bmatrix}$ kết quả được tính toán là

$$A\,.\,(B.r) - C.r = \begin{bmatrix}2 & 3\\ 3 & 4\end{bmatrix}\left(\begin{bmatrix}1 & 0\\ 1 & 2\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix}\right) - \begin{bmatrix}6 & 5\\ 8 & 7\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \begin{bmatrix}-1\\ -1\end{bmatrix}.$$

Kết quả khác 0, nên $AB \neq C$ (phù hợp với thực tế $AB \neq C$).

Có tất cả bốn ma trận 0-1, hai trong số đó cho kết quả vectơ $\vec{0}$ ($r = \begin{bmatrix}0\\ 0\end{bmatrix}$ và $r = \begin{bmatrix}1\\ 1\end{bmatrix}$), nên giải thuật trả lời $AB = C$ với xác suất là $1/2^2$. Nghĩa là, giải thuật trả lời kết quả sai với xác suất nhiều nhất là 1/4. Giải thuật này là loại giải thuật Monte Carlo.

4.6. 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 $O(n + m)$. 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 $O(n^2 \log_2 n)$ hay giải thuật Prim, độ phức tạp $O(n + m\alpha(m, n))$ [2].

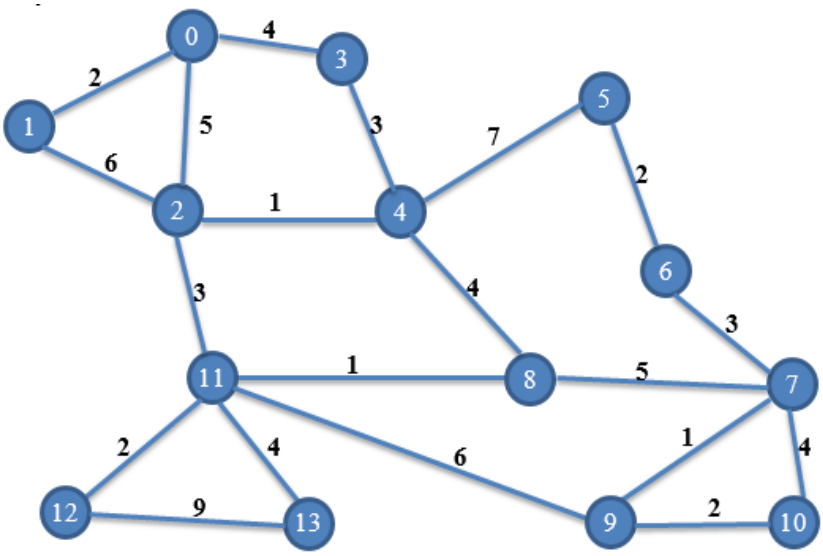
Giải thuật ngẫu nhiên này dựa trên khái niệm *bước Boruvka* (Boruvka step), được đề xuất bởi Boruvka vào năm 1926 [2], khi tìm cây bao trùm nhỏ nhất bằng giải thuật cổ điển. Bổ đề sau minh họa ý tưởng bên trong bước Boruvka.

Bổ đề 4.6.1 Giả sử $G = (V, E)$ là đồ thị vô hướng liên thông có trọng số, V_1 và V_2 là các tập con khác rỗng của V sao cho $V_1 \cup V_2 = V$ và $V_1 \cap V_2 = \varnothing$, cạnh (u,v) là cạnh có trọng số nhỏ nhất với một điểm kết thúc trong V_1 và một điểm kết thúc trong V_2. Thì, (u,v) là một trong các cạnh trong cây bao trùm nhỏ nhất của G.

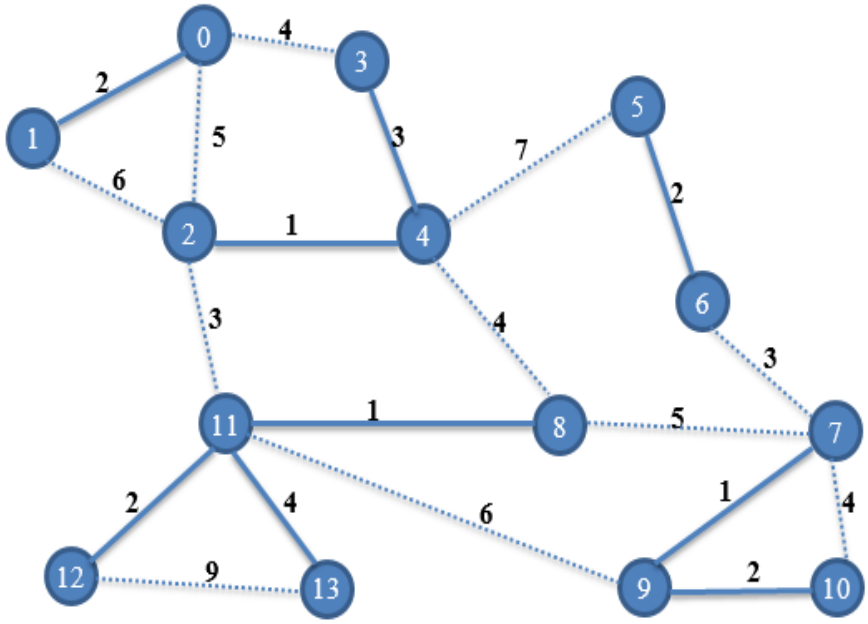
Bổ đề 4.6.1 có thể được phát biểu dưới dạng khác như sau: trong một đồ thị G vô hướng liên thông có trọng số, với mỗi đỉnh u, trong số các cạnh liên thuộc u, nếu cạnh (u,v) có trọng số nhỏ nhất thì (u,v) phải là cạnh nằm trong cây bao trùm nhỏ nhất của G.

Ví dụ 4.6.1 Xem đồ thị ở Hình 4.6.1, với đỉnh 2, trong số tất cả cạnh liên thuộc với đỉnh 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, cạnh (5,6) cũng phải nằm trong cây bao trùm nhỏ nhất.

Ví dụ 4.6.2 Áp dụng Bổ đề 4.6.1, các cạnh phải thuộc cây bao trùm nhỏ nhất của đồ thị trong Hình 4.6.1, được chỉ ra trong Hình 4.6.2. Các cạnh này gom thành các nhóm tạo nên các thành phần liên thông của đồ thị ban đầu.

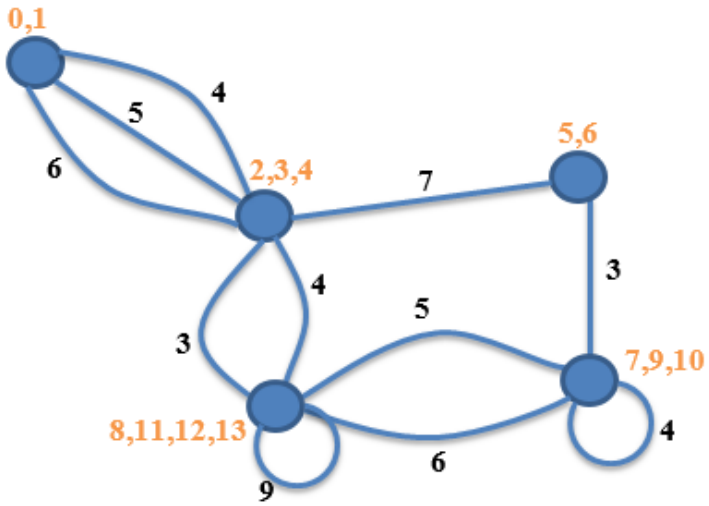


Hình 4.6.1: Một đồ thị vô hướng liên thông có trọng số



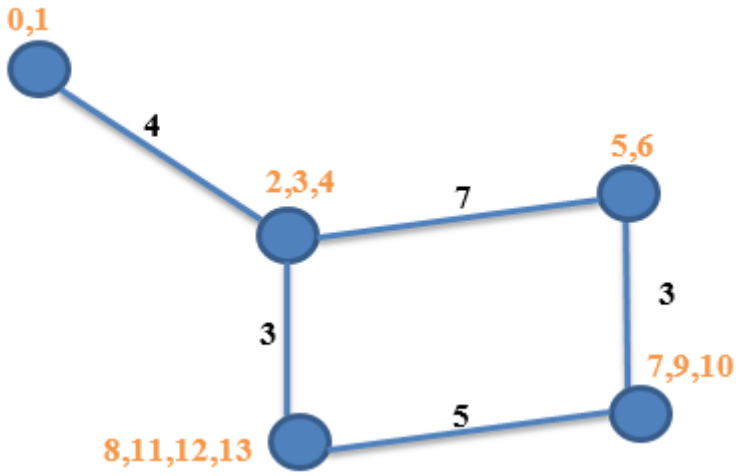
Hình 4.6.2: Các cạnh được chọn của đồ thị theo bổ đề Boruka

Do cây bao trùm nhỏ nhất của một đồ thị là liên thông chứa tất cả các đỉnh của đồ thị và không có chu trình, nên để tìm các cạnh còn lại của cây bao trùm nhỏ nhất ta thu tất cả các đỉnh của một thành phần liên thông về chỉ một đỉnh duy nhất như Hình 4.6.3.



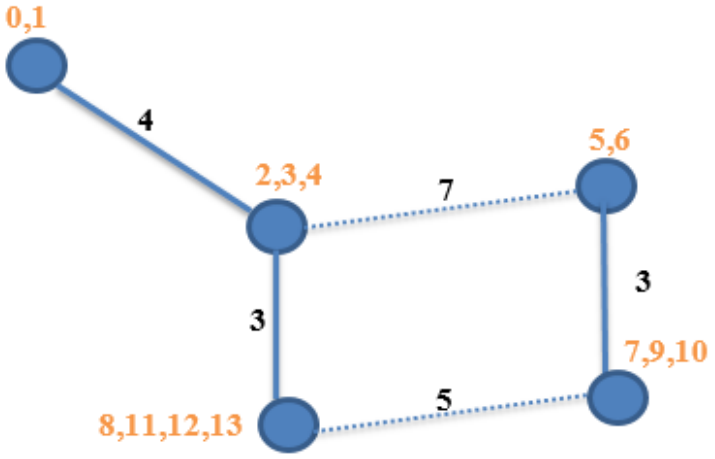
Hình 4.6.3: Đồ thị hợp nhất các thành phần liên thông về một nút

Sau khi loại bỏ các cạnh bội có trọng số lớn và cạnh khuyên chúng ta được kết quả như Hình 4.6.4.



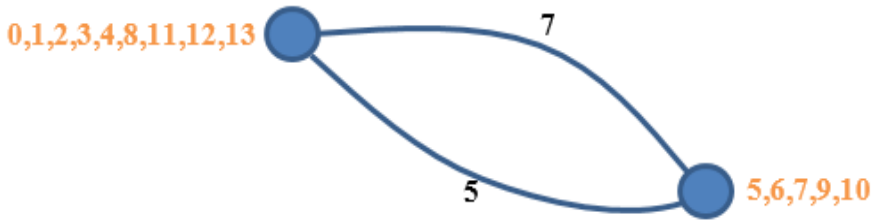
Hình 4.6.4: Kết quả sau khi áp dụng Boruvka lần thứ nhất

Từ đồ thị trong Hình 4.6.4, áp dụng chọn các cạnh theo bổ đề Boruvka lần thứ hai ta có kết quả như Hình 4.6.5.



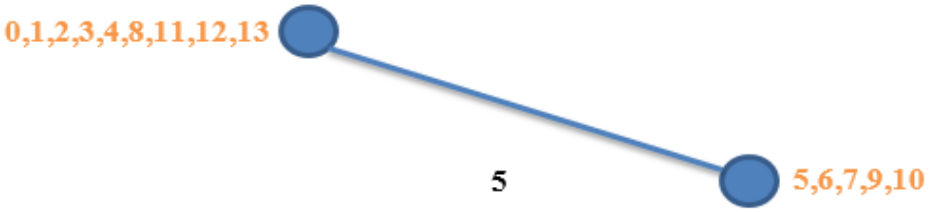
Hình 4.6.5: Các cạnh được chọn theo bổ đề Boruvka lần hai

Sau khi thu các nút ở mỗi thành phần liên thông về một nút đơn, chúng ta có hai đỉnh như Hình 4.6.6.



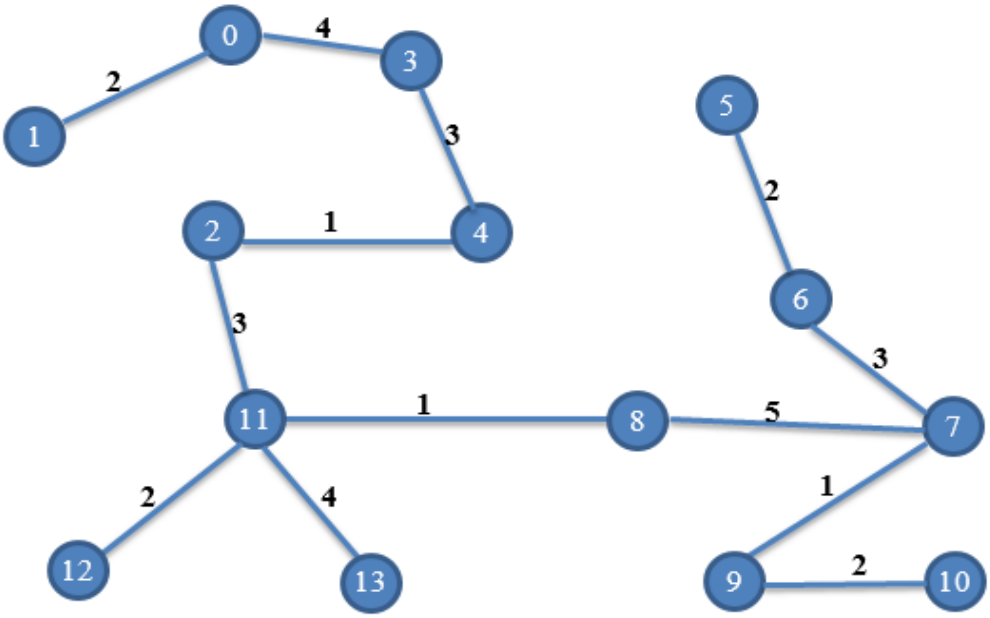
Hình 4.6.6: Đồ thị kết quả hợp nhất các thành phần liên thông lần thứ hai

Từ đồ thị trong Hình 4.6.6, loại bỏ cạnh bội có trọng số lớn (cây bao trùm không chứa chu trình) và cạnh khuyên, chúng ta có đồ thị chỉ còn hai đỉnh và một cạnh như Hình 4.6.7.



Hình 4.6.7: Kết quả sau khi áp dụng Boruvka lần thứ hai

Quá trình áp dụng bổ đề Boruvka hoàn thành, cây bao trùm nhỏ nhất được xây dựng (suy dẫn từ kết quả trong Hình 4.6.7) như Hình 4.6.8.



Hình 4.6.8: Cây bao trùm nhỏ nhất của đồ thị

Giải thuật Boruvka để tìm cây bao trùm nhỏ nhất áp dụng một số bước dựa trên bổ đề Boruvka 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 $G'=(V', E')$. Mỗi bước Boruvka (gồm hai giai đoạn) được phát biểu như sau:

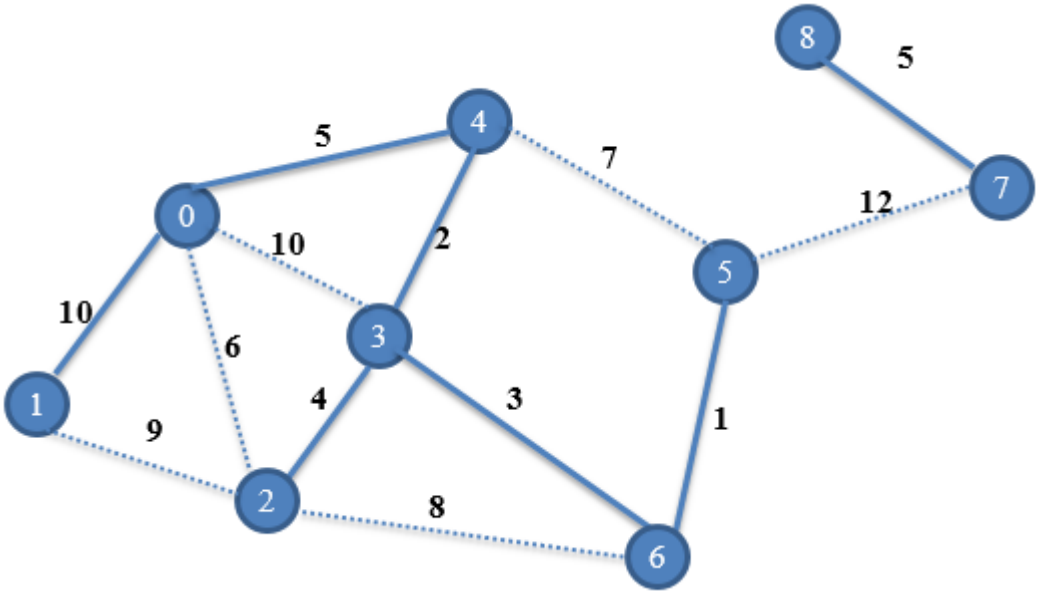
1. Với mỗi đỉnh u, tìm cạnh (u,v) với trọng số nhỏ nhất liên thuộc với nó. Tìm tất cả cách thành phần liên thông bằng cánh đánh dấu cạnh.
2. Thu nhỏ mỗi thành phần liên thông được xác định bởi cạnh đánh dấu thành một đỉnh duy nhất. Gọi đồ thị kết quả là $G'=(V', E')$. Loại bỏ các cạnh bội và cạnh khuyên.

Độ phức tạp của một bước Boruvka là $O(n+m)$ với n là số đỉnh và m là số cạnh. Vì G liên thông nên $m \geq n-1$. Do đó $O(n+m)=O(m)$. Vì mỗi thành phần liên thông được xác định bởi một cạnh đánh dấu chứa ít nhất 2 đỉnh, sau khi thi hành mỗi bước Boruvka số cạnh của đồ thị sẽ nhỏ hơn một nữa so với ban đầu. Do đó, tổng các bước Boruvka là $O(\log_2 n)$. Từ đó suy ra độ phức tạp của giải thuật Boruvka là $O(m \log_2 n)$.

Để sử dụng các bước Boruvka hiệu quả, chúng ta phải sử dụng khái niệm cạnh nặng, cạnh nhẹ của đồ thị theo một *rừng bao trùm nhỏ nhất* (minimum spanning forest) như trong định nghĩa sau.

Định nghĩa 4.6.1 Giả sử G_s là một đồ thị con của G, F là rừng bao trùm nhỏ nhất của G_s và $w_F(x, y)$ là trọng số cạnh lớn nhất trong các cạnh trên đường đi từ x đến y trong F ($w_F(x, y)=\infty$, nếu không có đường đi từ x đến y). Cạnh (x, y) được gọi là *cạnh nặng* (F-*heavy*) hay *cạnh nhẹ* (F-light) theo F nếu $w(x, y) > w_F(x, y)$ hay $w(x, y) \leq w_F(x, y)$.

Ví dụ 4.6.3 Xét rừng bao trùm nhỏ nhất F của đồ thị G trong Hình 4.6.9 (coi $G_s \equiv G$). Rừng F bao gồm các cạnh vẽ đậm liền nét. Trọng số lớn nhất của các cạnh trên đường đi $4 \rightarrow 3 \rightarrow 6 \rightarrow 5$ là 3. Theo Định nghĩa 4.6.1, suy ra cạnh (4, 5) có trọng số bằng $7>3$ là cạnh nặng theo F.



Hình 4.6.9: Cạnh nặng của đồ thị theo rừng bao trùm nhỏ nhất

Chúng ta dễ dàng nhận thấy rằng các cạnh nặng không thể thuộc bất kỳ cây bao trùm nhỏ nhất nào như bổ đề sau.

Bổ đề 4.6.2 Gọi G_s là một đồ thị con của G, F là một rừng bao trùm nhỏ nhất của G_s. cạnh nặng trong G theo F không thể là cạnh nằm trong cây bao trùm nhỏ nhất.

Cuối cùng, để có thể phát biểu được các bước của giải thuật ngẫu nhiên tìm cây bao trùm nhỏ nhất của đồ thị G, cần tìm các cạnh nhẹ của đồ thị dựa trên bổ đề sau.

Bổ đề 4.6.3 Gọi H là đồ thị con có được từ đồ thị G bằng cách chọn mỗi cạnh của G một cách độc lập với xác suất p, và gọi F là rừng bao trùm nhỏ nhất của H. Số trung bình của các cạnh nhẹ trong G nhiều nhất có thể đạt được là n/p, với n là số đỉnh của G.

Bây giờ giải thuật tìm cây bao trùm nhỏ nhất được thiết kế gồm ba bước như dưới đây [2].

Giải thuật

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

Output: Cây bao trùm nhỏ nhất của *G*.

Bước 1: Thực hiện các bước Boruvka ba lần. Kết quả đạt được là đồ thị $G_1=(V_1, E_1)$. Nếu G_1 chứa một đỉnh, trả về bộ các cạnh đánh dấu ở bước 1 và kết thúc.

Bước 2: Phát sinh một đồ thị con *H* của G_1 bằng cách chọn mỗi cạnh độc lập với xác suất ½. Áp dụng đệ quy giải thuật cho *H* để tạo một rừng bao trùm nhỏ nhất *F* của *H*. Đồ thị con $G_2=(V_2, E_2)$ đạt được bằng cách xóa toàn bộ cạnh nặng theo *F* trong G_1.

Bước 3: Áp dụng đệ quy giải thuật cho *G*2.

Độ phức tạp của giải thuật

Gọi $T(|V|,|E|)$ là thời gian chạy kỳ vọng của giải thuật ứng với đồ thị $G = (V, E)$. Mỗi lần thực thi Bước 1 chi phí là $O(|V| + |E|)$. Sau Bước 1, ta có $|V_1| \leq \frac{|V|}{8}$ và $|E_1| \leq |E|$. Với Bước 2, thời gian để tính *H* là $O(|V_1| + |E_1|) = O(|V| + |E|)$, thời gian để tính *F* là $\mathrm{T}\left(|V_1| + \frac{|E_1|}{2}\right) = T\left(\frac{|V|}{8} + \frac{|E|}{2}\right)$. Thời gian cần thiết để loại bỏ những cạnh nặng là $(|V_1| + |E_1|) = O(|V| + |E|)$. Sử dụng Bổ đề 4.6.3 ta có giá trị kỳ vọng cho $|E_2|$ lớn nhất là $2|V_2| \leq \frac{|V|}{4}$. Từ đó, ta có thời gian kỳ vọng cần để thực hiện Bước 3 là $\mathrm{T}(|V_2| + |E_2|) = T\left(\frac{|V|}{8}, \frac{|V|}{4}\right)$. Nếu $|V| = n$ và $|E| = m$. Chúng ta có hệ thức truy hồi sau.

$$T(n, m) \leq T\left(\frac{n}{8}, \frac{m}{2}\right) + T\left(\frac{n}{8}, \frac{n}{4}\right) + c(n + m), \text{ với } c \text{ là hằng số.}$$

Từ đây suy ra $T(n, m) \leq 2c.(n + m)$ [2].

Như vậy, thời gian chạy kỳ vọng của giải thuật là $O(n + m)$.

Mã giả

```
1.  Randomized-MST (G)
2.  G1 = BoruvkaStep(G); // các bước Boruvka lần thứ I
3.  if G1 has edges then
4.      if G1 has one edg  //nếu G1 chỉ có một cạnh thì thêm vào đồ thị kết quả (solvedGraph)
5.          then add this edg to solvedGraph
6.      else
7.          G2 = BoruvkaStep(G1); // các bước Boruvka lần thứ II
8.  if G2 has edges then
9.      if G2 has one edg //nếu G2 chỉ có một cạnh thì thêm vào đồ thị kết quả
10.         then add this edg to solvedGraph
11.     else
12.         G3 = BoruvkaStep(G2); // các bước Boruvka lần thứ III
13. if G3 has edges then
14.     if G3 has one edg //nếu G3 chỉ có một cạnh thì thêm vào đồ thị kết quả
15.         then 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 suất 1/2
17.         Obtain a subgraph H of G3 by selecting each edge with probability ½
18.         F=Randomized-MST(H)
19.         for each edg in G3 and not in F
20.             if this edg is not a F-heavy edge
21.                 then  add this edg to G4 // kết thúc bước 2
22.         Randomized-MST(G4); // bước 3
23. return  solvedGraph
```

4.7. Kết luận

Trong Chương này, chúng tôi đã giới thiệu và phân tích độ phức tạp của một số giải thuật ngẫu nhiên để giải một số bài toán thuộc các lĩnh vực khác nhau như lý thuyết số, lý thuyết đồ thị, hình học tính toán, lý thuyết giải thuật. Các giải thuật ngẫu nhiên được giới thiệu đều hiệu quả hơn các giải thuật truyền thống tương ứng. Đặc biệt giải thuật ngẫu nhiên giải bài toán tìm cặp điểm gần nhau nhất và tìm cây bao trùm nhỏ nhất đều có độ phức tạp tuyến tính tốt hơn nhiều so với các giải thuật truyền thống có độ phức tạp $O(n\log_2 n)$. Điều đó chứng tỏ, tính hiệu quả của các giải thuật ngẫu nhiên so với giải thuật truyền thống. Đó cũng là động lực để các giải thuật ngẫu nhiên được tiếp tục nghiên cứu để giải các bài toán thực tế. Trong Chương 5 tiếp theo, chúng tôi sẽ hiện thực các giải thuật đã được giới thiệu ở các phần trên thành một hệ thống để giải các bài toán đã nêu một cách nhanh chóng và hiệu quả.

Chương 5

HIỆN THỰC CÁC ỨNG DỤNG GIẢI THUẬT NGẪU NHIÊN

5.1. Giới thiệu

Chương này trình bày những vấn đề cơ bản nhất của quá trình hiện thực một số giải thuật ngẫu nhiên đã đề cập trong Chương 4. Đầu tiên, Phần 5.2 là phần hiện thực giải thuật ngẫu nhiên Quicksort để giải bài toán sắp xếp một dãy số thực bất kỳ. Kế đến, Phần 5.3 là giải thuật giải quyết bài toán xác định số nguyên tố. Tiếp theo, Phần 5.4 là giải thuật ngẫu nhiên để giải bài toán tìm cặp điểm gần nhất trong không gian hai chiều. Phần 5.5 trình bày phần hiện thực cho giải thuật ngẫu nhiên kiểm tra phép nhân ma trận. Phần 5.6 là giải thuật ngẫu nhiên tìm cây bao trùm nhỏ nhất của đồ thị liên thông có trọng số. Tất cả các giải thuật ngẫu nhiên trong chương này đều được hiện thực bằng ngôn ngữ C#. Việc hiện thực thành công các giải thuật ngẫu nhiên một cách hiệu quả (có so sánh với giải thuật cổ điển) chứng tỏ khả năng áp dụng của chúng vào thực tế. Phần 5.7 là phần hệ thống các chương trình thực hiện các giải thuật thành một chương trình thống nhất. Cuối cùng, Phần 5.8 là một số kết luận của chương này.

5.2. Chương trình sắp xếp dãy số

Cấu trúc dữ liệu và hàm chính hiện thực giải thuật ngẫu nhiên Quicksort lần lượt được giới thiệu như dưới đây.

Cấu trúc dữ liệu

Như trong giải thuật cổ điển, danh sách các số cần sắp xếp được tổ chức theo kiểu mảng một chiều. Vì giải thuật ngẫu nhiên được dùng để khắc phục hạn chế của giải thuật Quicksort truyền thống trong trường hợp xấu nhất bằng cách hoán vị ngẫu nhiên phần tử chốt nên chọn mảng là hợp lý nhất.

Hàm chính hiện thực giải thuật

Giải thuật Quicksort ngẫu nhiên chỉ khác với giải thuật Quicksort cổ điển ở chỗ có thêm hàm randomizedPartition(): chọn một phần tử ngẫu nhiên trong mảng và đưa nó về cuối mảng để làm phần tử chốt. Hàm randomizedQuicksort() sẽ gọi hàm randomizedPartition() để chọn chốt, sau đó thực hiện hàm Parttition() để tiến hành xếp các phần tử nhỏ hơn chốt về bên trái và các phần tử lớn hơn chốt về bên phải, cuối cùng tiếp tục gọi đệ qui trên hai mảnh đó cho đến khi toàn mảng được sắp xếp. Các hàm chính của chương trình được thể hiện như trong Hình 5.2.1.

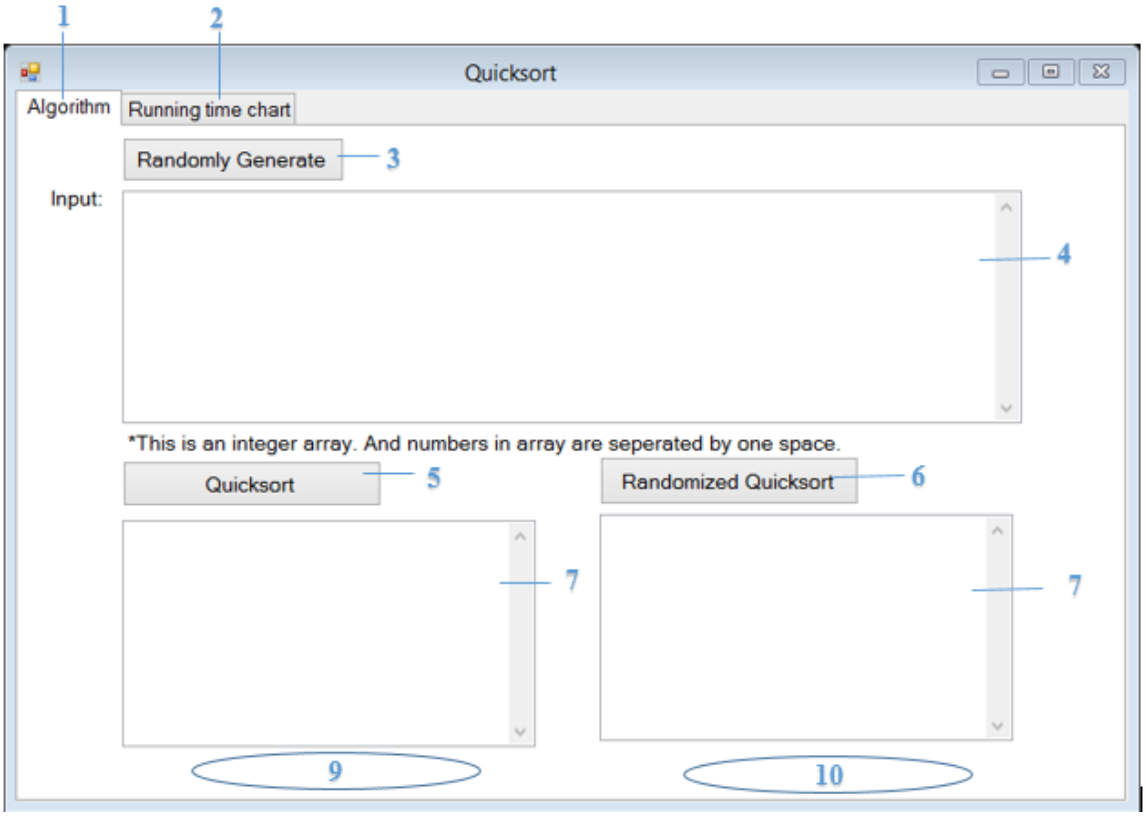
```
private void randomizedQuicksort(int[] a, int p, int r)
        {
            if (p < r)
            {
                int q=randomizedPartition(a,p,r);
                randomizedQuicksort(a, p, q - 1);
                randomizedQuicksort(a, q + 1, r);
            }
        }
        //Chọn chốt
        private int randomizedPartition(int[] a, int p, int r)
        {
            //Chọn chỉ mục i ngẫu nhiên trong mảng.
            int i = random.Next(p, r+1);
            //Hoán vị trí của phần tử thứ i với phần tử cuối mảng.
            permutation(ref a[i],ref a[r]);
            return Parttition(a, p, r);
        }
        // Sắp xếp mảng: phần tử nhỏ hơn bằng chốt về bên trái chốt và phần tử lớn
hơn chốt về bên phải chốt
```

```
        private int Parttition(int[] a, int p, int r)
        {
            int x = a[r];
            int i = p - 1;
            for (int j = p; j < r; j++)
            {
                if (a[j] <= x)
                {
                    i++;
                    permutation(ref a[i],ref a[j]);
                }
            }
            permutation(ref a[i + 1], ref a[r]);
            return i + 1;

        }
```

Hình 5.2.1: Hàm chính của giải thuật ngẫu nhiên Quicksort

Giao diện và cách sử dụng chương trình

Giao diện được chia thành 2 tab như trong Hình 5.2.2: Algorithm (1)-hiển thị chương trình sắp xếp sử dụng giải thuật Quicksort và Quicksort ngẫu nhiên và Running time chart (2)-hiển thị đồ thị thời gian chạy.

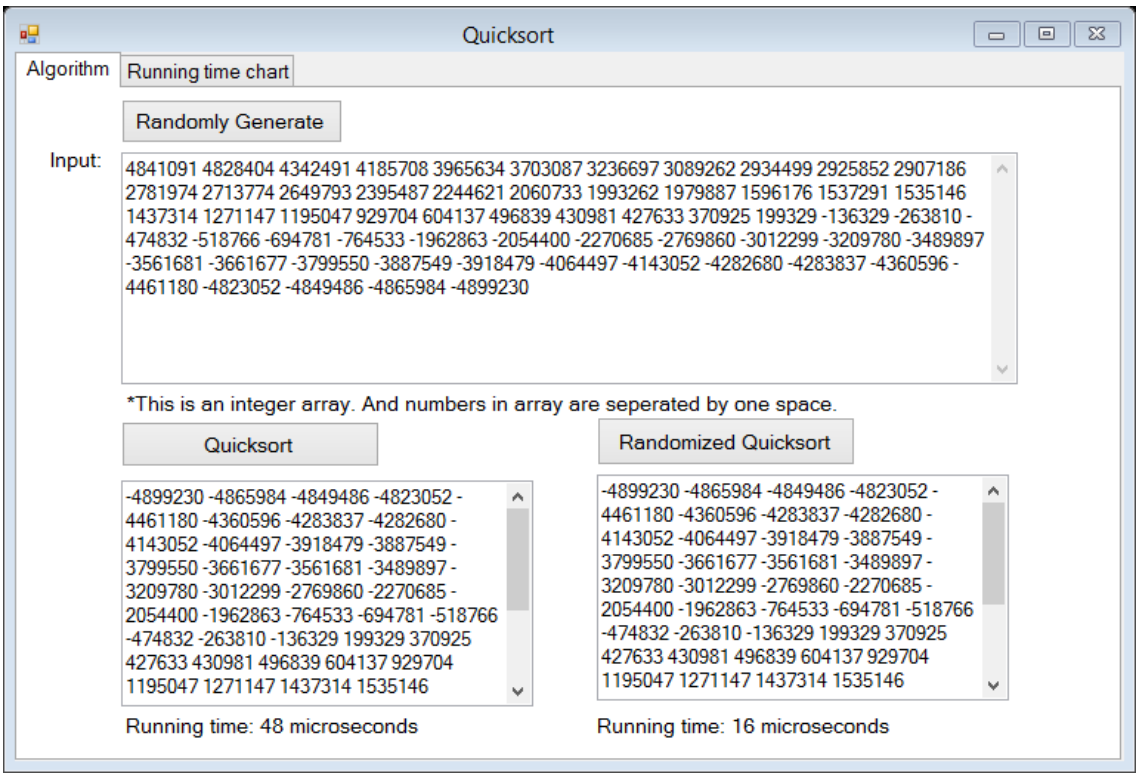


Hình 5.2.2: Giao diện chính của giải thuật Quicksort

Nút Randomly Generate (3) có chức năng phát sinh dãy số ngẫu nhiên và được hiển thị tại textbox (4) , kích thước của mảng sẽ do người dùng nhập tại hộp thoại xuất hiện sau khi bấm nút Randomly Generate.

Nút Quicksort (5) và nút Randomized Quicksort (6) có chức năng thực thi giải thuật Quicksort và giải thuật Quicksort ngẫu nhiên trên dãy số ở textbox (4) và kết quả quá trình sắp xếp được hiển thị lần lượt tại textbox (7) và textbox (8). Thời gian chạy của giải thuật cũng được hiển thị tại vùng (9) và (10).

Trình tự thao tác thực thi trên giao diện chương trình như sau: đầu tiên, người dùng phải nhập một dãy số vào textbox (4) hoặc sử dụng nút Randomly Generate để phát sinh dãy các số ngẫu nhiên (dãy số này đã được xếp giảm dần để có được đầu vào thuộc trường hợp xấu nhất của giải thuật Quicksort cổ điển). Kế đó, người dùng chọn lựa giải thuật để tiến hành sắp xếp bằng hai nút Quicksort và nút Randomized Quicksort. Kết quả trả về được hiển thị tương ứng tại textbox (7) và (8) cùng thời gian chạy tương ứng ở vùng (9) và (10).



Hình 5.2.3: Kết quả chạy chương trình sắp xếp 60 số có giá trị giảm dần

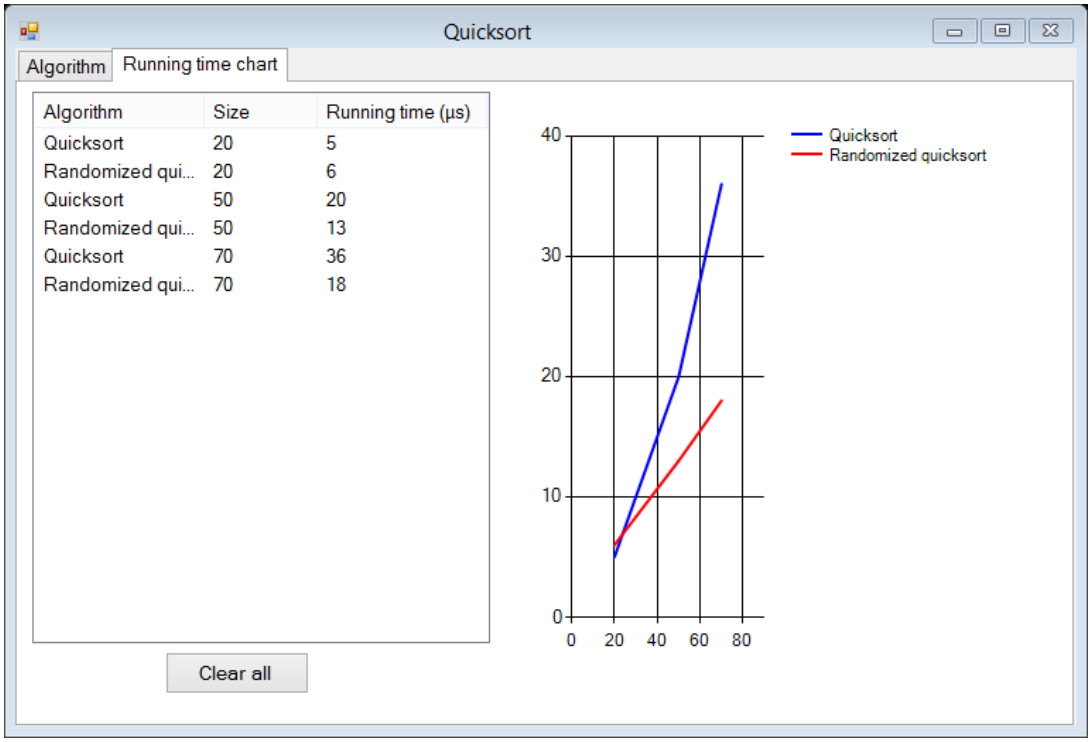
Thời gian chạy thực của RandomizedQuicksort

Như đã phân tích trong Phần 4.2, Độ phức tạp theo thời gian của giải thuật Quicksort ngẫu nhiên là O(n log n) và độ phức tạp của giải thuật Quicksort trong trường hợp xấu nhất là O(n^2). Sau đây, chúng tôi xin trình bày kết quả khảo sát thời gian chạy thực (đơn vị: microsecond) của hai giải thuật.

Bảng 5.2.1 cho biết kết quả thời gian thực thi tính bằng microsecond của hai giải thuật trên các dãy số với kích thước tương ứng là 20, 50, 70. Rõ ràng tốc độ thực thi của giải thuật RandomizedQuicksort nhanh hơn giải thuật Quicksort cổ điển.

Bảng 5.2.1: Thời gian chạy thực của Quicksort và RandomizedQuicksort

N	Quicksort	RandomizedQuicksort
20	6	6
50	14	8
60	17	10



Hình 5.2.4: Đồ thị thời gian chạy thực của Quicksort và RandomizedQuicksort

5.3. Chương trình xác định số nguyên tố

Cấu trúc dữ liệu và hàm chính hiện thực giải thuật ngẫu nhiên để phát triển chương trình xác định một số tự nhiên là số nguyên tố hay không lần lượt được giới thiệu như sau.

Cấu trúc dữ liệu

Như đã trình bày trong Phần 4.3 của Chương 4, mã giả của giải thuật ngẫu nhiên xác định số nguyên tố là khá đơn giản. Tuy nhiên, có một vấn đề lớn trong phần hiện thực của giải thuật này là hàm tính lũy thừa sẽ trả về một kết quả là một số rất lớn (khi *n* lớn), mà trong C# không có một kiểu dữ liệu nào có thể mô tả được. Vì vậy chúng tôi phải kế thừa một *lớp số nguyên lớn* (BigInteger class) [11], để làm việc với các số này.

Khi sử dụng lớp BigInteger chúng ta có thể thực hiện các phép toán +, -, *, /, %, >>, <<, ==, !=, >, <, >=, <=, &, |, ^, ++, -- và ~ trên số nguyên lớn.

Hàm chính hiện thực giải thuật

Hàm hiện thực giải thuật xác định số nguyên tố theo định lý Fermat được tổ chức như trong Hình 5.3.1.

```
public bool FermatLittleTest(int confidence)
    {
            BigInteger thisVal;

            if(thisVal.dataLength == 1)
            {
                    // đối với số nguyên tố nhỏ
                    if(thisVal.data[0] == 0 || thisVal.data[0] == 1)
                            return false;
                    else if(thisVal.data[0] == 2 || thisVal.data[0] == 3)
                            return true;
            }

            if((thisVal.data[0] & 0x1) == 0)     // số chẵn
                    return false;

          int bits = thisVal.bitCount();
          BigInteger a = new BigInteger();
          BigInteger p_sub1 = thisVal - (new BigInteger(1));
          Random rand = new Random();
```

```
                for(int round = 0; round < confidence; round++)
                {
                        bool done = false;

                        while(!done)        // phát sinh số a, a < n
                        {
                                int testBits = 0;

                                // đảm bảo a ít nhất 2 bit
                                while(testBits < 2)
                                        testBits = (int)(rand.NextDouble() * bits);

                                a.genRandomBits(testBits, rand);

                                int byteLen = a.dataLength;

                                // đảm bảo a khác 0
                                if(byteLen > 1 || (byteLen == 1 && a.data[0] != 1))
                                            done = true;
                        }


                        // tính a^(p-1) mod p
                        BigInteger expResult = a.modPow(p_sub1, thisVal);

                        int resultLen = expResult.dataLength;

                          // nếu a^(p-1) mod p != 1 thì n là hợp số

                       if(resultLen > 1 || (resultLen == 1 && expResult.data[0] !=
1))
                        {
                                return false;
                        }
                }

                return true;
        }
```

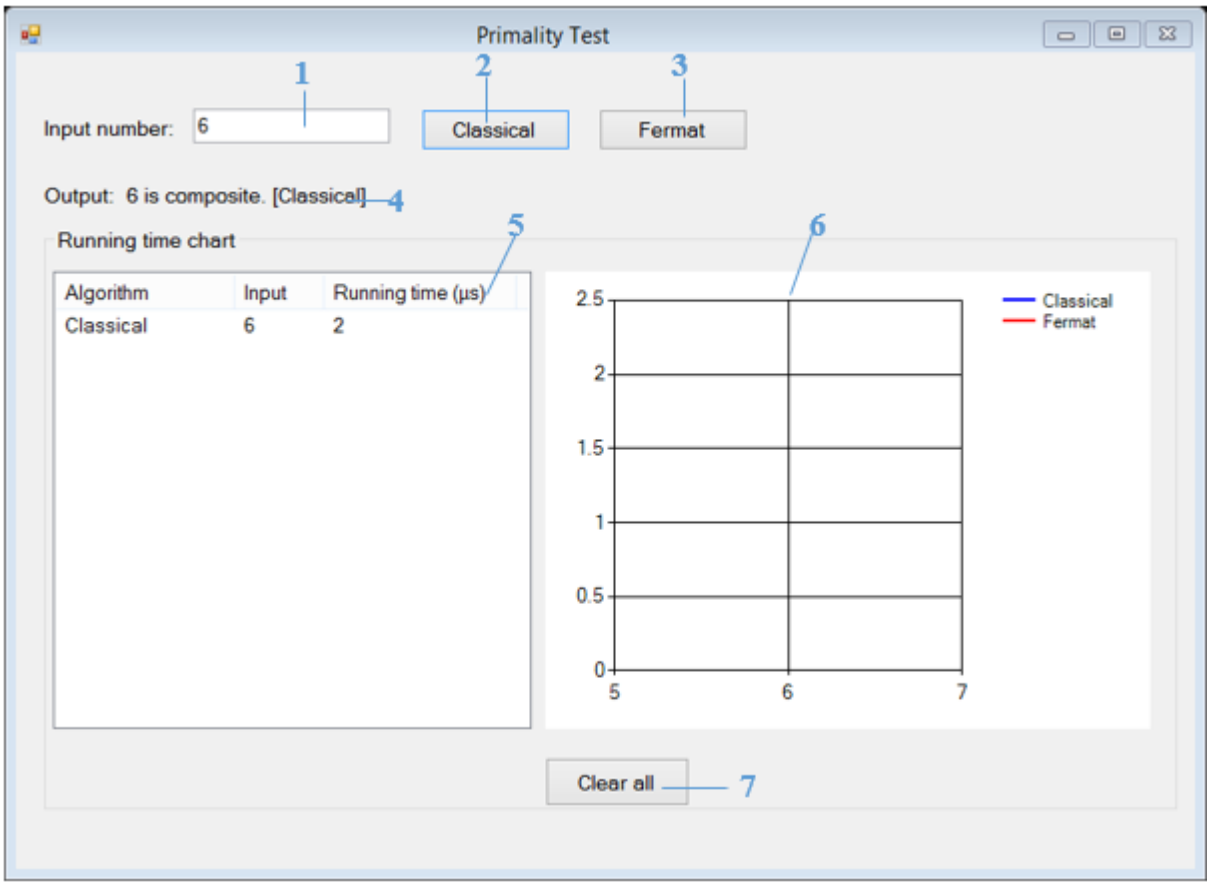
Hình 5.3.1: Hàm chính của giải thuật ngẫu nhiên kiểm tra số nguyên tố

Giao diện và cách sử dụng chương trình

Giao diện của chương trình hiện thực cho giải thuật ngẫu nhiên xác định số nguyên tố được thiết kế như Hình 5.3.2.

Đầu tiên chúng ta sẽ nhập một số tự nhiên vào vào hộp thoại Input number (1), sau đó click nút Classical (2) (để xác định số nguyên tố theo phương pháp cổ điển) hoặc Fermat (3) (để xác định số nguyên tố theo giải thuật ngẫu nhiên). Nếu kiểm tra theo giải thuật ngẫu nhiên, chương trình sẽ yêu cầu chúng ta nhập tham số *m* (*m* là số lần kiểm tra)

Tiếp theo, chương trình sẽ thực hiện việc kiểm tra và hiển thị kết quả (N là số nguyên tố hay hợp số - ở vị trí (4); thời gian chạy theo microsecond - vị trí (5) và vẽ đồ thị ở vị trí (6)). Khi chúng ta muốn xóa hết dữ liệu thời gian chạy và đồ thị để vẽ lại đồ thị khác thì click nút Clear all (7).



Hình 5.3.2: Giao diện chính của giải thuật xác định số nguyên tố

Thời gian chạy thực của giải thuật ngẫu nhiên Fermat

Như đã trình bày ở Phần 4.3 độ phức tạp của giải thuật Fermat là $O(k.\log_2 n)$, độ phức tạp của giải thuật cổ điển kiểm tra số nguyên tố là $O(\sqrt{n})$.

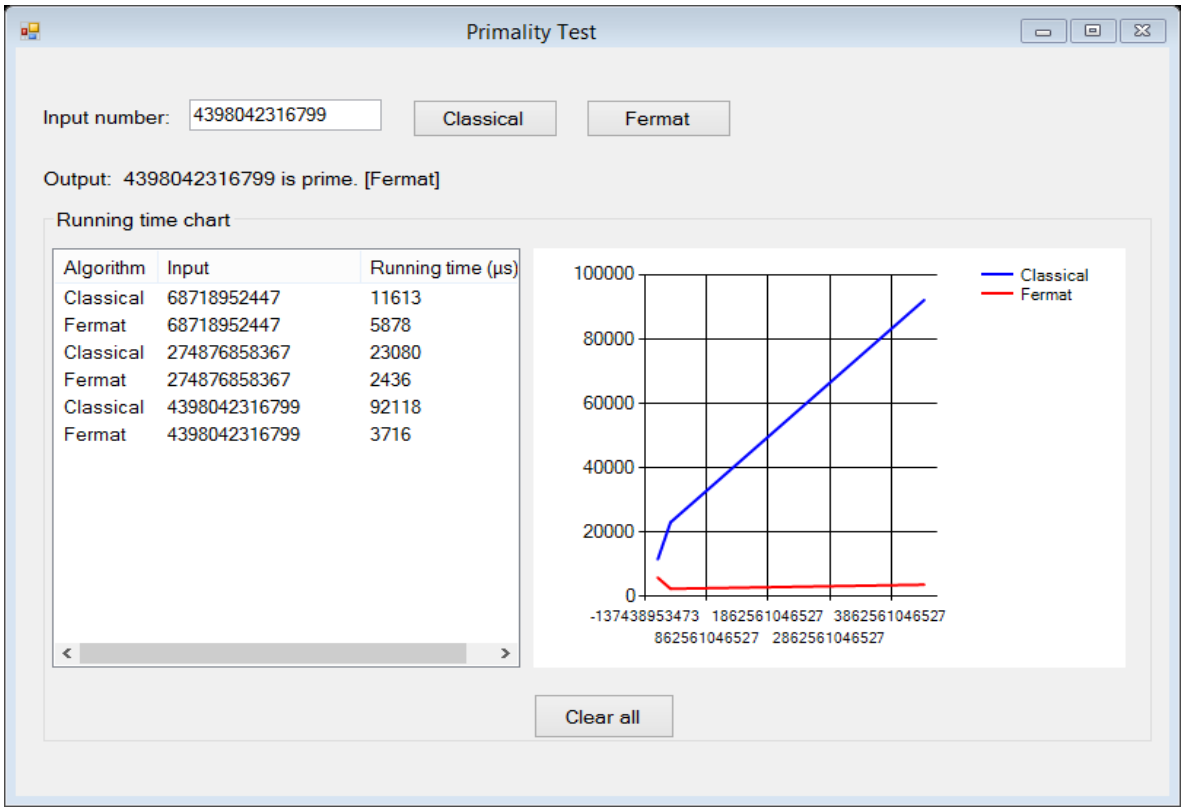
Bảng 5.3.1 phản ánh khá chân thực thời gian chạy thực tế dựa trên một số thí nghiệm so với đánh giá thời gian chạy của chúng theo lý thuyết. Rõ ràng, giải thuật ngẫu nhiên chạy nhanh hơn giải thuật cổ điển.

Bảng 5.3.1: Thời gian chạy của các giải thuật xác định số nguyên tố

Số N	Kết quả khi thực hiện bằng giải thuật cổ điển	Kết quả khi thực hiện bằng giải thuật ngẫu nhiên
68718952447	11613	5878
274876858367	23080	2436
4398042316799	92118	3716

Hình 5.3.3 là đồ thị biểu diễn thời gian chạy của hai giải thuật xác định số nguyên tố dựa trên

Bảng 5.3.1 và một số thí nghiệm bổ sung. Trong hình này, đường màu xanh là đường biểu diễn thời gian chạy của giải thuật cổ điển, đường màu đỏ là đường biểu diễn thời gian chạy của giải thuật ngẫu nhiên. Dễ thấy tốc độ của giải thuật ngẫu nhiên nhanh hơn giải thuật cổ điển.



Hình 5.3.3:Đồ thị so sánh thời gian chạy của hai giải thuật xác định số nguyên tố

5.4. Chương trình tìm cặp điểm gần nhất

Cấu trúc dữ liệu và những hàm thực thi chủ yếu tạo nên chương trình được lần lượt trình bày như sau.

Cấu trúc dữ liệu

Trong bài toán tìm cặp điểm gần nhất, tập các điểm của không gian hai chiều được định nghĩa như một mảng các cặp hai số thực. Các hình vuông chứa các điểm được định nghĩa là danh sách của các điểm.

Hàm chính hiện thực giải thuật

Hàm RandomizedAl(): có chức năng hiện thực Bước 1, Bước 2 và Bước 5 của giải thuật được đề cập ở Phần 4.4 được triển khai mã như Hình 5.4.1.

```
bool recursive = true,inrecurisve=false;
private IPoint[] RandomizedAl(IPoint[] P, out double distanceRA)
        {
            distanceRA = double.MaxValue;

            if (P.Length <= 3 && !inrecurisve)
                return bruteForce(P, ref distanceRA);

            //Chọn ngẫu nhiên n^2/3 điểm từ input
            //Tạo thành mảng S[]
            int rp = (int)Math.Round(Math.Pow(P.Length, (2 * 1.0 / 3)), 0);
            IPoint[] S = new IPoint[rp];

            List<int> indexP = new List<int>();
            for (int i = 0; i < P.Length; i++)
            {
                indexP.Add(i);
            }

            for (int i = 0; i < S.Length; i++)
            {
                int r = random.Next(0, indexP.Count);
                S[i] = P[indexP[r]];
                indexP.RemoveAt(r);
            }

            //Tính khoảng cách ngắn nhất trong mảng S[], bằng cách đệ qui gọi hàm
RandomizedAl 1 lần
            //Đặt làm delta
            double delta=double.MaxValue;
```

```
            //Trong lần đệ qui đó dùng chiến lược trực tiếp để tìm khoảng cách
delta

            if (inrecurisve)
            {
                inrecurisve = false;
                IPoint[] tmp = bruteForce(S, ref delta);
            }
            if (recursive)
            {
                recursive = false;
                inrecurisve = true;
                IPoint[] resultS = RandomizedAl(S, out delta);
            }

            //Xác định số lượng ô vuông có cạnh bằng delta:
            //Xác định hoành độ lớn nhất và tung độ lớn nhất trong các điểm
            //Chọn giá trị lớn và đem chia cho denta
            int maxX, maxY;
            FindMaxXY(P, out maxX, out maxY);
            int maxValue = maxX >= maxY ? maxX : maxY;
            int SquareInRow = (int)Math.Round(maxValue / delta);
            if (SquareInRow * delta <= maxValue)
                SquareInRow++;

            //Xây dựng tập T
            Dictionary<int, List<IPoint>> boT = new Dictionary<int,
List<IPoint>>();
            for (int i = 0; i < P.Length; i++)
            {
                double a = P[i]._X / delta;
                double b = P[i]._Y / delta;
                int index = ((int)a + SquareInRow * (int)b);
                if (!boT.ContainsKey(index))
                {
                    boT[index] = new List<IPoint>();
                }
                boT[index].Add(P[i]);
            }
            //Mở rộng các hình vuông có chứa điểm
            Dictionary<string, List<IPoint>> doubleSize = FindNeighbor(boT,
SquareInRow);

            //Tìm khoảng cách ngắn nhất từ các hình vuông mở rộng
            IPoint[] resultP = new IPoint[2];
            foreach (var pair in doubleSize)
            {
                IPoint[] resultTmp = bruteForce(pair.Value.ToArray(), ref
distanceRA);
                if (resultTmp != null)
                {
                    resultP = resultTmp;
                }
            }
            return resultP;
        }
```

Hình 5.4.1: Hàm thành viên RandomizedAl tính cặp điểm gần nhau nhất

Hàm FindNeighbor() là hàm hiện thực Bước 3 và Bước 4 của giải thuật được đề cập ở Phần 4.4. như trong Hình 5.4.2.

```
private Dictionary<string, List<IPoint>>
FindNeighbor(Dictionary<int,List<IPoint>> boT,int SquareInRow )
        {
            Dictionary<string, List<IPoint>> doubleSize = new Dictionary<string,
List<IPoint>>();
            //Mở rộng các hình vuông trong bộ T -Xây dựng T1,T2,T3,T4
            foreach (var s in boT)
            {
                int keyT = s.Key, rightN = keyT + 1, aboveN = keyT + SquareInRow,
arN = aboveN + 1, alN = aboveN - 1;
                string key = keyT.ToString();

                //Nếu s hiện tại có hai điểm hoặc tồn tại các tập điểm lân cận s-
không tính các s ở biên phải
                if (s.Value.Count > 1 || ((keyT % SquareInRow != SquareInRow - 1)
&& (boT.ContainsKey(rightN) || boT.ContainsKey(aboveN) || boT.ContainsKey(arN))))
                {
                    doubleSize[key] = new List<IPoint>();
                    doubleSize[key].AddRange(s.Value);

                    //Thêm neighbor
                    if (boT.ContainsKey(rightN))
                        doubleSize[key].AddRange(boT[rightN]);
                    if (boT.ContainsKey(aboveN))
                        doubleSize[key].AddRange(boT[aboveN]);
                    if (boT.ContainsKey(arN))
                        doubleSize[key].AddRange(boT[arN]);
                }

                //Trường hợp s có tập điểm lân cận nằm chéo trái trên, không tính
các s ở biên trái
                if (boT.ContainsKey(alN) && keyT % SquareInRow != 0)
                {
                    string specialKey = key + "_special";
                    doubleSize[specialKey] = new List<IPoint>();
                    doubleSize[specialKey].AddRange(boT[keyT]);
                    doubleSize[specialKey].AddRange(boT[alN]);
                }
            }
            return doubleSize;
        }
```

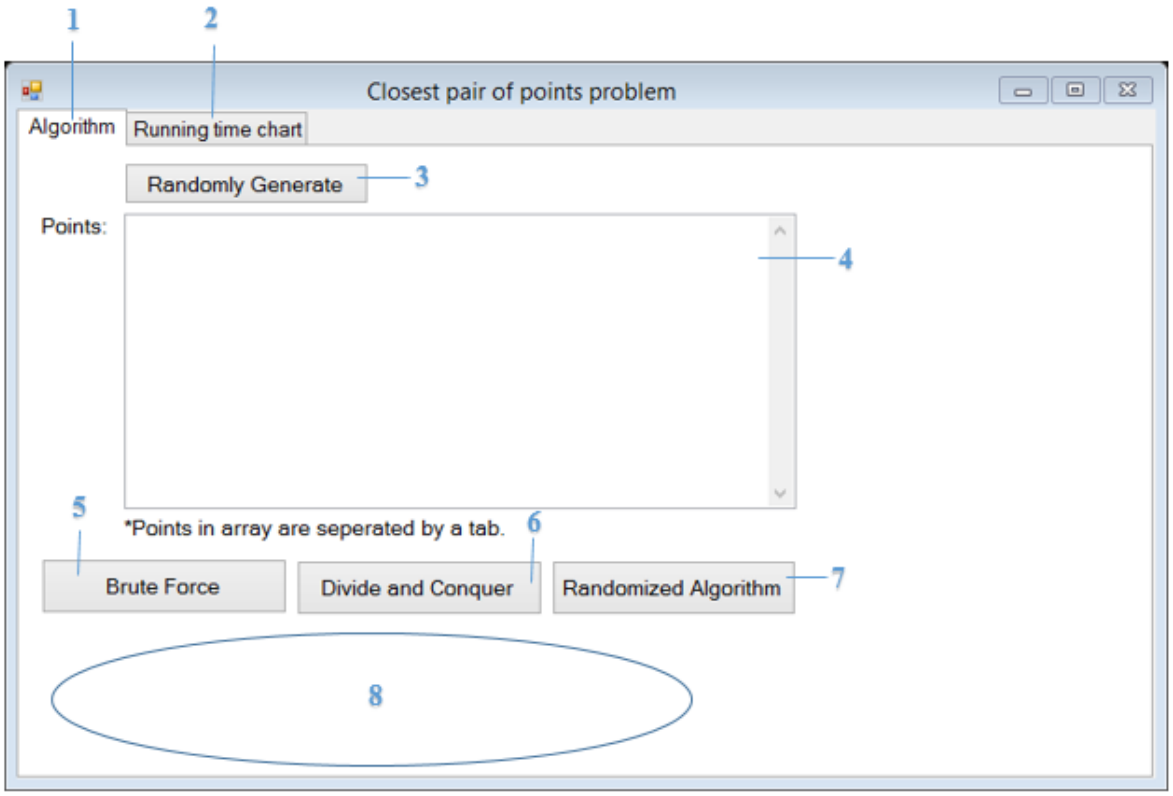
Hình 5.4.2: Hàm thành viên FindNeighbor tính cặp điểm gần nhau nhất

Giao diện và cách sử dụng chương trình

Giao diện được chia thành 2 tab như Hình 5.4.3: Algorithm (1)-hiển thị chương trình tìm cặp điểm gần nhất sử dụng giải thuật trực tiếp, giải thuật chia để trị và giải thuật ngẫu nhiên; Running time chart (2)-hiển thị đồ thị thời gian chạy.

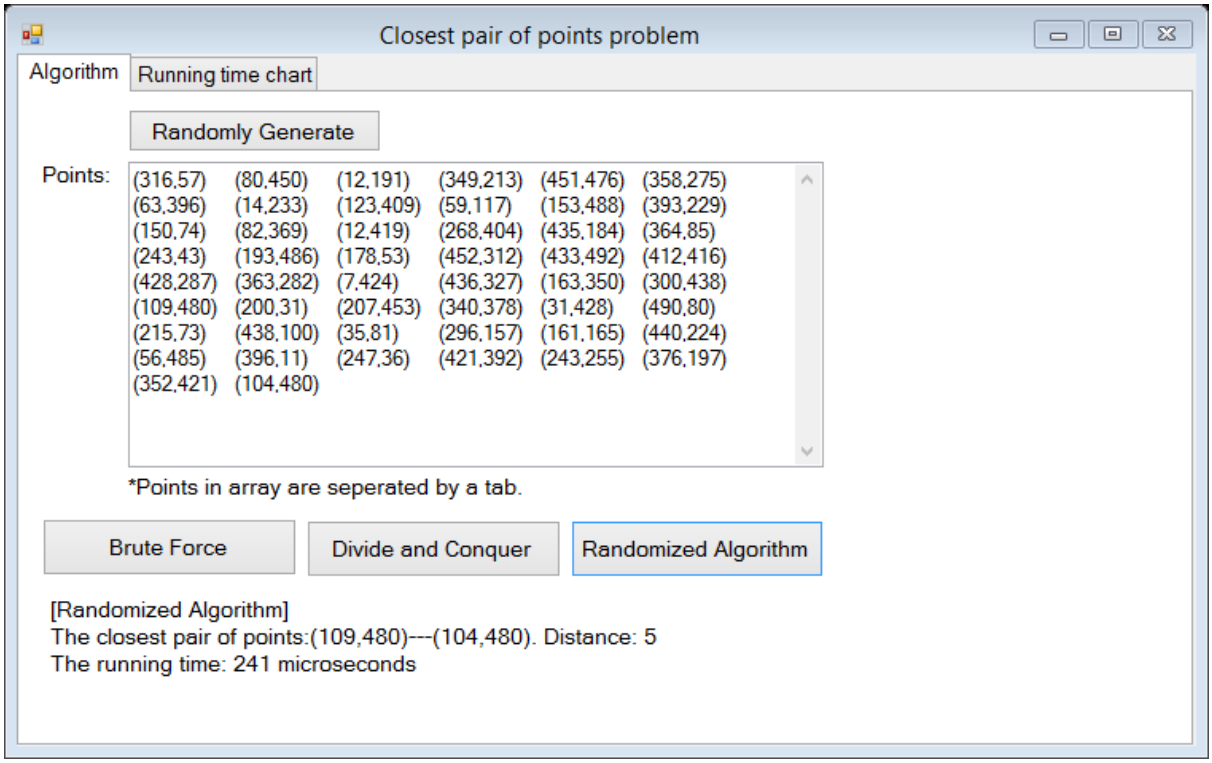
Nút Randomly Generate (3) có chức năng phát sinh mảng các điểm có tọa độ ngẫu nhiên và được hiển thị tại textbox (4), sau khi click chuột vào nút này sẽ có một hộp thoại yêu cầu chúng ta nhập kích thước của mảng.

Nút Brute Force (5), nút Divide and Conquer (6) và nút Randomized Algorithm (7) lần lượt có chức năng thực thi giải thuật trực tiếp, giải thuật chia để trị và giải thuật ngẫu nhiên tìm cặp điểm gần nhất trong mảng điểm ở textbox (4). Vùng (8) hiển thị thông tin kết quả trả về, sau khi thực hiện giải thuật gồm: cặp điểm gần nhất, khoảng cách giữa chúng và thời gian thực hiện giải thuật.



Hình 5.4.3: Giao diện chính của giải thuật tìm cặp điểm gần nhất

Trình tự thao tác thực thi của chương trình được chỉ ra như sau: đầu tiên, người dùng phải nhập một dãy điểm vào textbox (4) hoặc sử dụng nút Randomly Generate để phát sinh dãy các số ngẫu nhiên. Kế đó, người dùng chọn lựa giải thuật để tiến hành tìm kiếm bằng ba nút Brute Force, Divide and Conquer, Randomized Algorithm. Kết quả sẽ được trả về tại vùng (8).



Hình 5.4.4: Kết quả chương trình tìm cặp điểm gần nhất trong 50 điểm

Thời gian chạy thực của giải thuật tìm cặp điểm gần nhau nhất

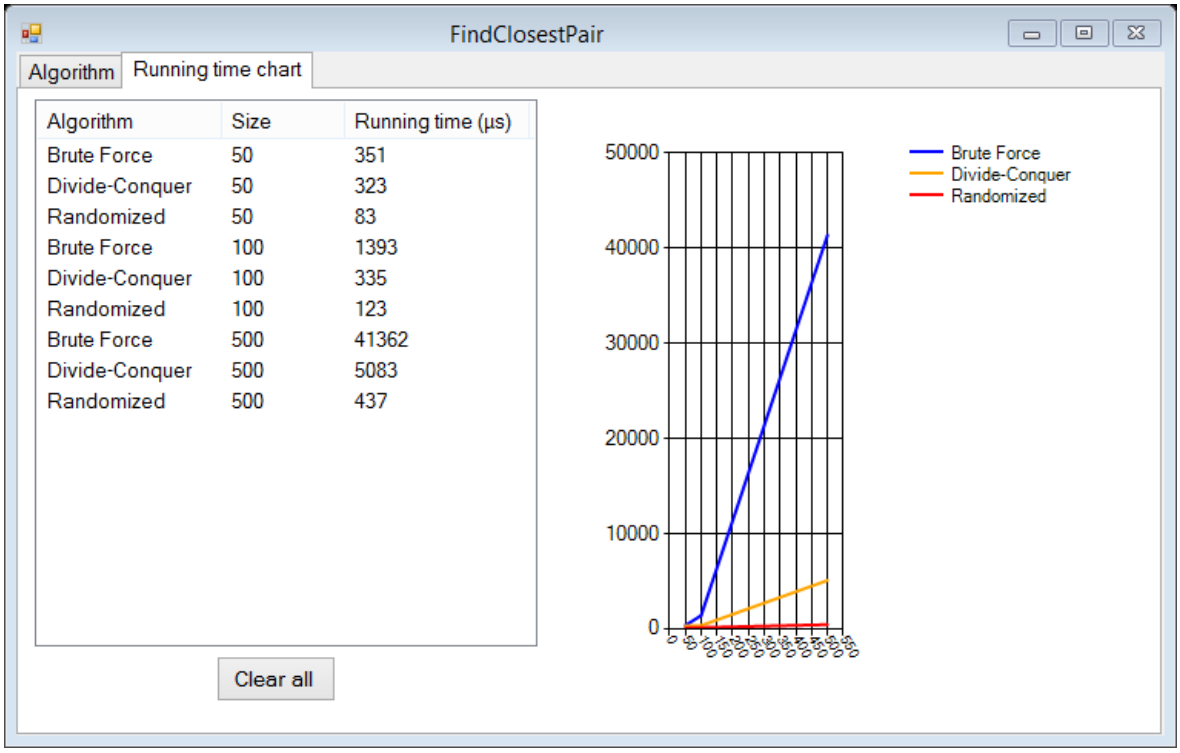
Như đã phân tích trong Phần 4.4, độ phức tạp của giải thuật ngẫu nhiên là O(n), trong khi độ phức tạp của giải thuật trực tiếp và chia để trị lần lượt là O(n^2) và O($n \log_2 n$).

Một số thí nghiệm lần lượt với cả ba giải thuật trên cùng một tập điểm, cho kết quả thời gian chạy tương ứng như trong Bảng 5.4.1.

Bảng 5.4.1: Thời gian chạy thực của các giải thuật tìm cặp điểm gần nhau nhất

N	Giải thuật trực tiếp	Giải thuật chia để trị	Giải thuật ngẫu nhiên
50	351	323	83
100	1393	335	123
500	41362	5083	437

Sử dụng số liệu từ Bảng 5.4.1 và bổ sung thêm một số thí nghiệm chúng ta vẽ các đường biểu diễn tốc độ tăng của thời gian chạy của các giải thuật trong Hình 5.4.5. Dễ thấy giải thuật ngẫu nhiên (đường màu đỏ) là hiệu quả nhất, khi kích thước bài toán tăng lên.



Hình 5.4.5: Đồ thị thời gian chạy thực của ba giải thuật tìm cặp điểm gần nhất

5.5. Chương trình kiểm tra phép nhân ma trận

Cấu trúc dữ liệu và hàm chính cho chương trình kiểm tra phép nhân các mà trận là khá đơn giản như dưới đây.

Cấu trúc dữ liệu

Trong chương trình kiểm tra phép nhân ma trận, các ma trận được lưu trữ trong các mảng hai chiều. Chúng tôi cũng xin lưu ý rằng các ma trận được sử dụng ở đầu vào là ma trận vuông.

Hàm chính hiện thực giải thuật

Hàm MatrixEqualityTest() như trong Hình 5.5.1 là hàm chính của giải thuật Freivalds với các tham số truyền vào là các ma trận *A,B,C* kích thước n x n và số lần thử là *k*.

```
private bool MatrixEqualityTest(int[,] MTA, int[,] MTB, int[,] MTC,ref int k)
        {
            strVector = "";
            int MTsize=MTA.GetLength(0);
            for (int j = 0; j < k; j++)
            {
                //Tạo vector ngẫu nhiên r
                int[,] vectorR = new int[MTsize, 1];
                for (int i = 0; i < MTsize; i++)
                {
                    vectorR[i, 0] = random.Next(0, 2);
                }
                //mỗi lần chạy in vector ngẫu nhiên
                if(!startBF)
                    strVector +="random vector in trial "+(j+1).ToString()+":\r\n"
+PrintMatrix(vectorR) + "\r\n";

                int[,] MTResultL, MTResultR;
                //A x (B x r)
                MTResultL = MultiplyMT(MTB, vectorR, MTsize, MTsize, 1);
                MTResultL = MultiplyMT(MTA, MTResultL, MTsize, MTsize, 1);

                //C x r
                MTResultR = MultiplyMT(MTC, vectorR, MTsize, MTsize, 1);

                //A x (B x r)!C x r
                if(!CompareMT(MTResultR, MTResultL))
                    return false;
            }
            return true;

        }
```

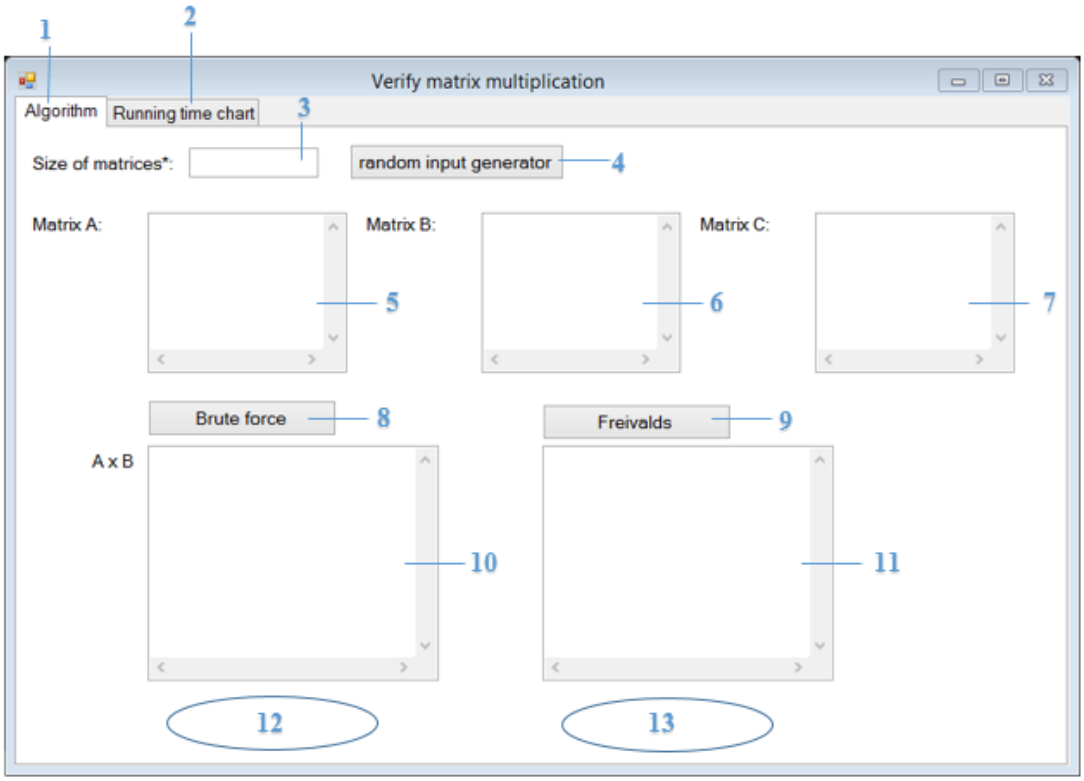
Hình 5.5.1: Hàm chính kiểm tra phép nhân ma trận

Giao diện và cách sử dụng chương trình

Giao diện được chia thành 2 tab như Hình 5.5.2: Algorithm (1)-hiển thị chương trình kiểm tra phép nhân ma trận bằng giải thuật trực tiếp và giải thuật Freivalds và Running time chart (2)-hiển thị đồ thị thời gian chạy.

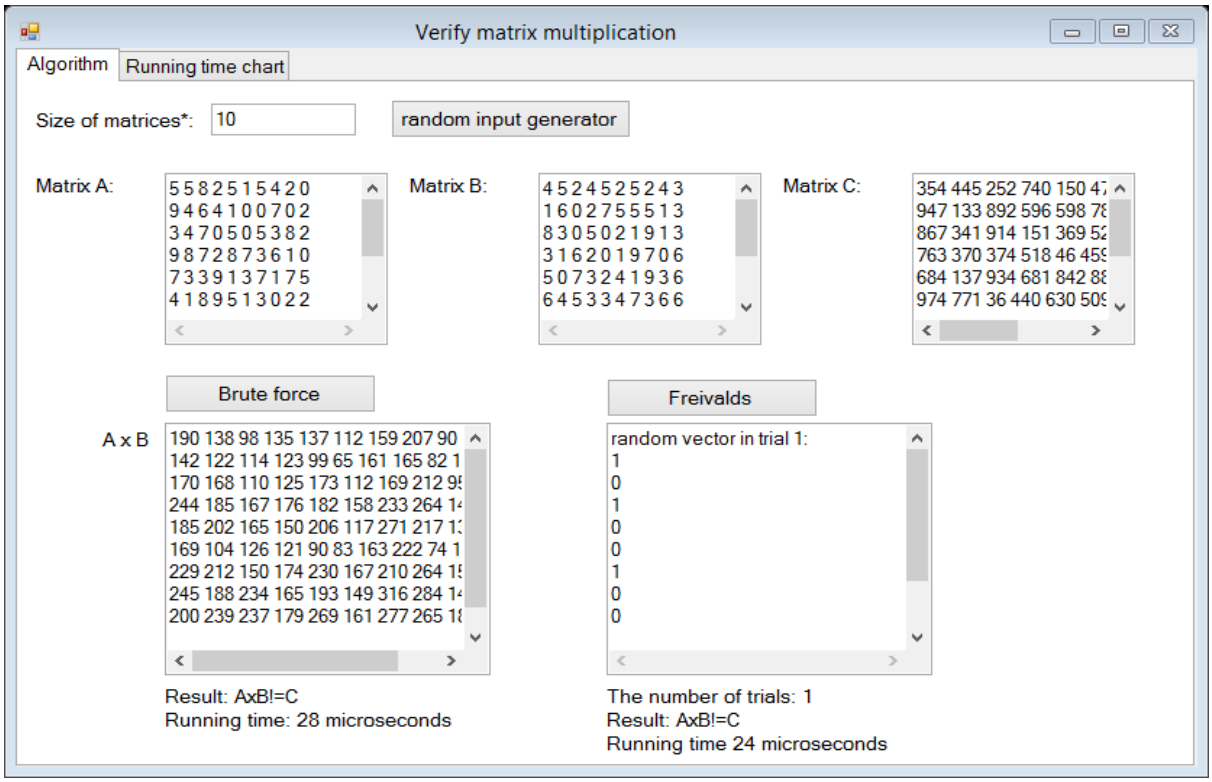
Nút Random input Generator (4) có chức năng phát sinh ba ma trận ngẫu nhiên *A, B* và *C* được hiển thị lần lượt tại textbox (5), (6) và (7). Kích thước của ba ma trận bắt buộc phải nhập vào textbox (3) trước khi bấm nút này.

Nút Brute force (8) và nút Freivalds (9) có chức năng thực thi giải thuật trực tiếp và giải thuật Freivalds kiểm tra phép nhân ma trận $AxB = C$. Kết quả trả về sau khi thực hiện giải thuật trực tiếp gồm kết quả phép nhân ma trận AxB tại textbox (10) và xác nhận liệu AxB có bằng C cùng thời gian chạy hiển thị tại vùng (12). Kết quả trả về sau khi thực hiện giải thuật Freivalds gồm: các vectơ ngẫu nhiên được dùng trong các lần thử tại textbox(11) và hiển thị số lần thử, xác nhận liệu AxB có bằng C và thời gian chạy được hiển thị tại vùng (13).



Hình 5.5.2: Giao diện chính của giải thuật kiểm tra phép nhân ma trận

Thao tác thực thi chương trình và kết quả chạy như sau: đầu tiên, người dùng phải nhập kích thước của ba ma trận vào textbox (3). Sau đó có thể dùng nút Random input Generator để sinh ma trận ngẫu nhiên hoặc nhập vào tại textbox (5), (6) và (7), lưu ý rằng kích thước của ba ma trận phải bằng với số được nhập tại textbox (3). Cuối cùng chọn giải thuật để thực hiện việc kiểm tra phép nhân ma trận bằng nút (8) hoặc (9).



Hình 5.5.3: Kết quả kiểm tra phép nhân ma trận với kích thước bằng 10

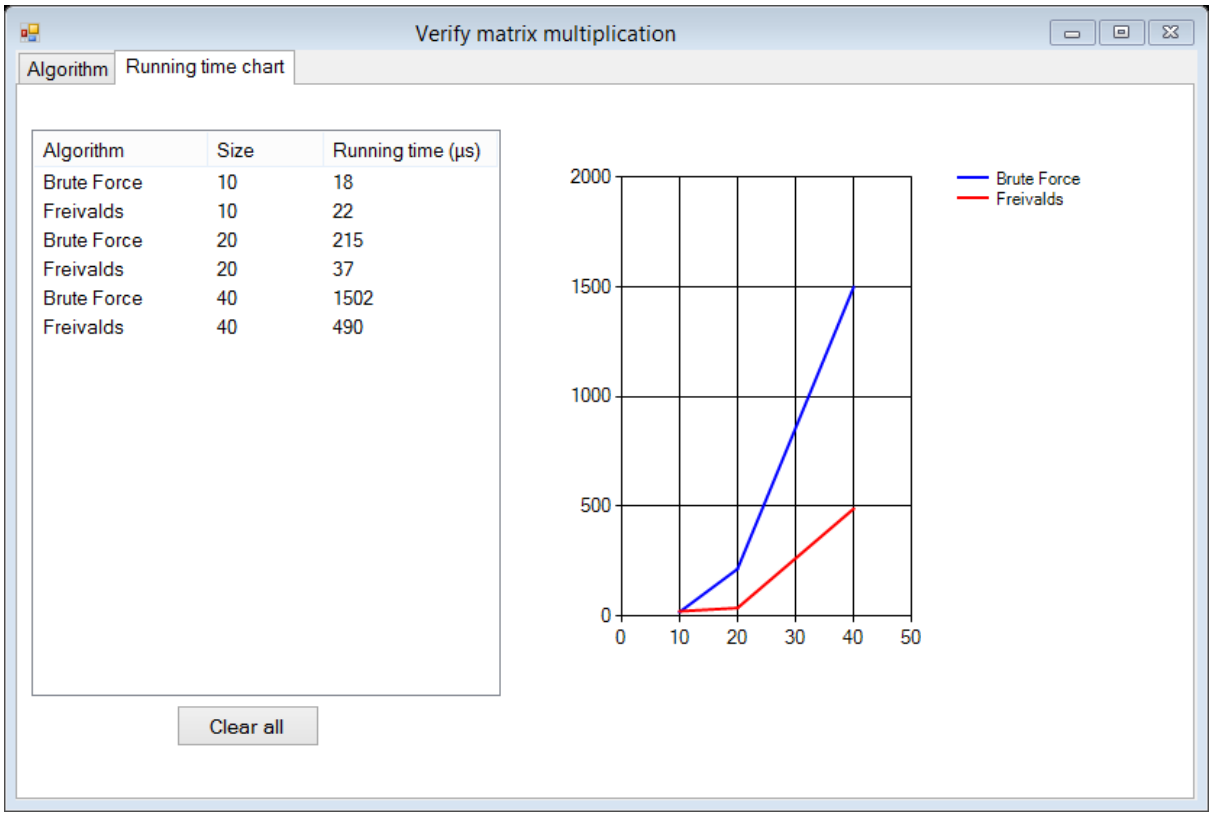
Thời gian chạy thực của giải thuật Freivalds

Bảng 5.5.1 cho biết thời gian chạy thực của giải thuật trực tiếp và giải thuật Freivalds ứng với một số thí nghiệm theo kích thước đầu vào tăng dần. Rõ ràng giải thuật ngẫu nhiên nhanh hơn.

Bảng 5.5.1: Thời gian chạy thực của giải thuật trực tiếp và giải thuật Freivalds

N	Giải thuật trực tiếp	Giải thuật Freivalds
10	18	22
20	215	37
40	1502	490

Sử dụng số liệu từ Bảng 5.5.1 và một số thí nghiệm bổ sung có thể vẽ đồ thị biểu diễn tốc độ tăng thời gian chạy của hai giải thuật như trong Hình 5.5.4. Một lần nữa chúng ta thấy giải thuật ngẫu nhiên hiệu quả hơn giải thuật cổ điển.



Hình 5.5.4: Đồ thị thời gian chạy thực của hai giải thuật kiểm tra ma trận

5.6. Chương trình tìm cây bao trùm nhỏ nhất của đồ thị

Cấu trúc dữ liệu và hàm chính của chương trình tìm cây bao trùm nhỏ nhất bằng giải thuật ngẫu nhiên lần lượt được giới thiệu như dưới đây.

Cấu trúc dữ liệu

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

Mỗi đỉnh bao gồm các thuộc tính *tên đỉnh* (name), *vùng* (rank), *nút gốc* (root), *vị trí hiển thị đỉnh trên form* (point). (Rank và root dùng để xác định các cạnh đang xét có cùng một cây hay không).

Hàm chính hiện thực giải thuật

Hàm chính của chương trình là hàm thực hiện các bước Boruvka như trong Hình 5.6.1.

```
        private IList<Edge> Boruvka(IList<Edge> graph, IList<Edge>
solvedGraph_Input, out IList<Edge> solvedGraph)
        {
            solvedGraph = new List<Edge>();//  solvedGraph  là những cạnh gạch
liền trong hình 11-9
            foreach (Edge ed in solvedGraph_Input) // với mỗi cạnh từ
solvedGraph_input thêm vào solvedGraph
                solvedGraph.Add(ed);
            IList<Edge> solvedGraphBoruvka = new List<Edge>();// kết quả
            // Tìm danh sách cạnh còn lại
            IList<Edge> remainListEdge = new List<Edge>();//danh sách cạnh còn lại
            #region Lấy danh sách tên đỉnh
            IList<int> vertexName=new List<int>(); // lưu tên đỉnh
            foreach (Edge ed in graph)
            {

                bool check1 = false;

                foreach (int i in vertexName)
                {
                    if ( ed.V1.Name2 == i)
                    {
                        check1 = true;
                        break;
                    }

                }
                if (check1 == false)
                    vertexName.Add(ed.V1.Name2);
                bool check2 = false;
                if (ed.V1.Name2 != ed.V2.Name2)
                {
                    foreach (int i in vertexName)
                    {
                        if (ed.V2.Name2 == i)
                        {
                            check2 = true;
                            break;
                        }

                    }
                    if (check2 == false)
```

```
                        vertexName.Add(ed.V2.Name2);
                    }
                }
                #endregion

                // Nếu số cạnh bằng số đỉnh -1 thì không phải giải tiếp
                if (vertexName.Count - 1 == graph.Count)
                    return graph;

                #region tính với mỗi đỉnh tìm cạnh có trọng số nhỏ nhất trả về
solvedGraph
                for (int i = 0; i < vertexName.Count; i++)// với mỗi đỉnh tìm cạnh có
trọng số nhỏ nhất
                {
                    int minCost=int.MaxValue;
                    Edge minCostEdge = graph[0]; // cạnh có trọng số nhỏ nhất liên
thuộc với đỉnh i
                    foreach (Edge ed in graph) // Tìm minCostEdge
                    {

                        if (ed.V1.Name2 == vertexName[i] || ed.V2.Name2 ==
vertexName[i]) // nếu cạnh này là liên thuộc với đỉnh i
                        {
                            if (minCost > ed.Cost)
                            {
                                minCost = ed.Cost;
                                minCostEdge = ed;
                            }
                        }
                    }

                    // Kết hợp (join) lại,

                    Vertex root1 = minCostEdge.V1.GetRoot();
                    Vertex root2 = minCostEdge.V2.GetRoot();
                    if (root1.Name != root2.Name)
                    {
                        Vertex.Join(root1, root2);
                        solvedGraph.Add(minCostEdge);
                    }


                }
                #endregion

                #region tim danh sach cac cạnh còn lại, trả về remainListEdge
                foreach (Edge ed in graph)
                {
                    bool kiemtra = false;
                    foreach (Edge ed2 in solvedGraph)
                    {
                        if (ed == ed2)
                        {
                            kiemtra = true;
                            break;
                        }
                    }
                    if (kiemtra == false) // nếu cạnh này không tồn tại trong
solvegraph
```

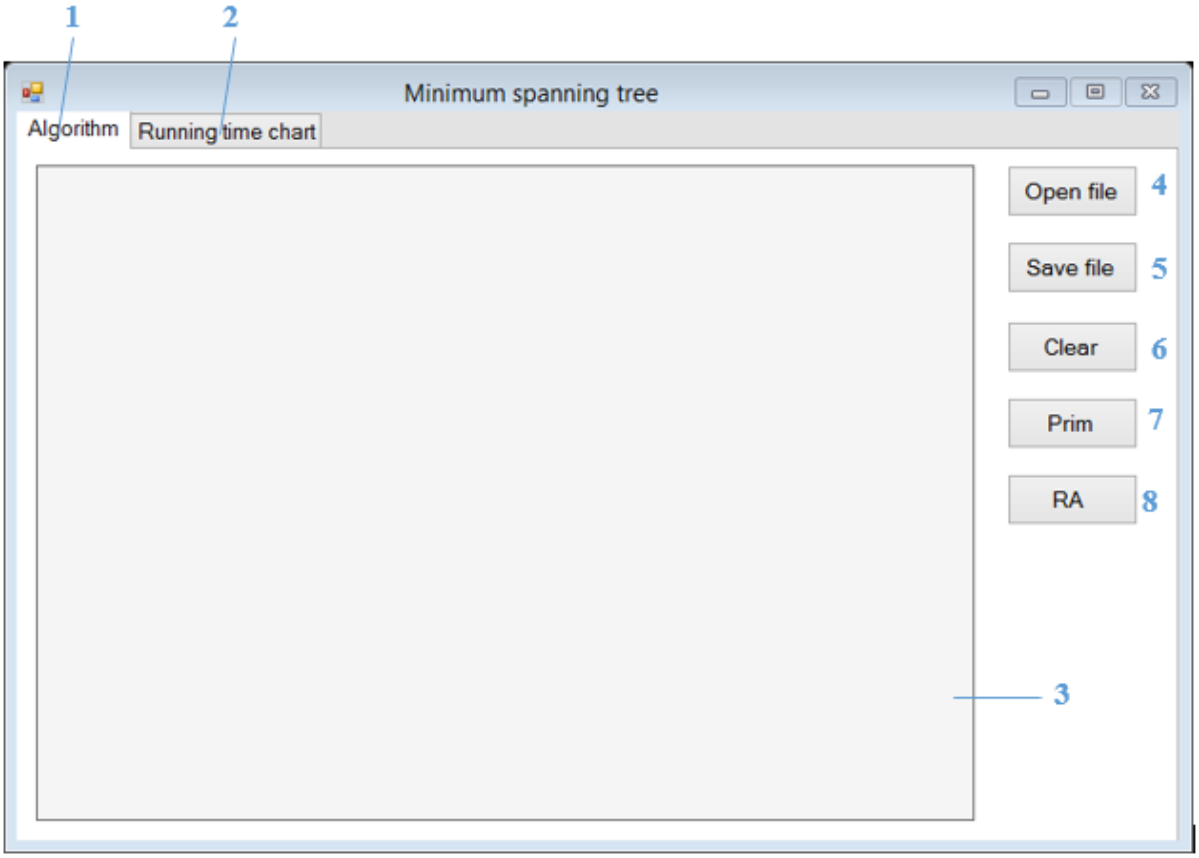
```
                {
                    remainListEdge.Add(ed);
                }

            }
            #endregion

            #region tính đồ thị hình thành sau khi bỏ cạnh trùng và khuyên, trả về
solvedGraphBoruvka
            // trong số các cạnh còn lại nếu một đỉnh thuộc root này, một thuộc
root kia. tìm cạnh có trọng số nhỏ nhất
            foreach (Edge ed in remainListEdge)
            {

                Vertex root1 = ed.V1.GetRoot();
                Vertex root2 = ed.V2.GetRoot();
                if (root1.Name != root2.Name)
                {
                    Edge minCostListEgdeRemain = ed; // cạnh có trọng số nhỏ nhất
trong các cạnh trùng
                    foreach (Edge ed2 in remainListEdge)
                    {
                        Vertex root3 = ed2.V1.GetRoot();
                        Vertex root4 = ed2.V2.GetRoot();
                        if (((root3.Name == root1.Name) && (root4.Name ==
root2.Name)) || ((root3.Name == root2.Name) && (root4.Name == root1.Name)))
                        {
                            if (minCostListEgdeRemain.Cost > ed2.Cost)
                                minCostListEgdeRemain = ed2;
                        }
                    }
                    // kiểm tra nếu solvedGraphBoruvka không có cạnh này mới add
cạnh này vô
                    bool testResult = false; //
                    foreach (Edge ed3 in solvedGraphBoruvka)
                    {

                        if (ed3 == minCostListEgdeRemain)
                        {
                            testResult = true;
                            break;
                        }


                    }
                    if(testResult==false)
                       solvedGraphBoruvka.Add(minCostListEgdeRemain);

                }

            }
            #endregion


            return solvedGraphBoruvka;
        }
```

Hình 5.6.1: Hàm chính tính cây bao trùm nhỏ nhất của đồ thị

Giao diện và cách sử dụng chương trình

Giao diện được chia thành 2 tab như trong Hình 5.6.2: Algorithm (1) - hiển thị giải thuật và Running time chart (2) - hiển thị đồ thị thời gian chạy.

Vùng (3) là khu vực vẽ đồ thị mới, hiển thị đồ thị đã vẽ trước đó, đồng thời cũng là nơi hiển thị kết quả (cây bao trùm nhỏ nhất cần tìm).



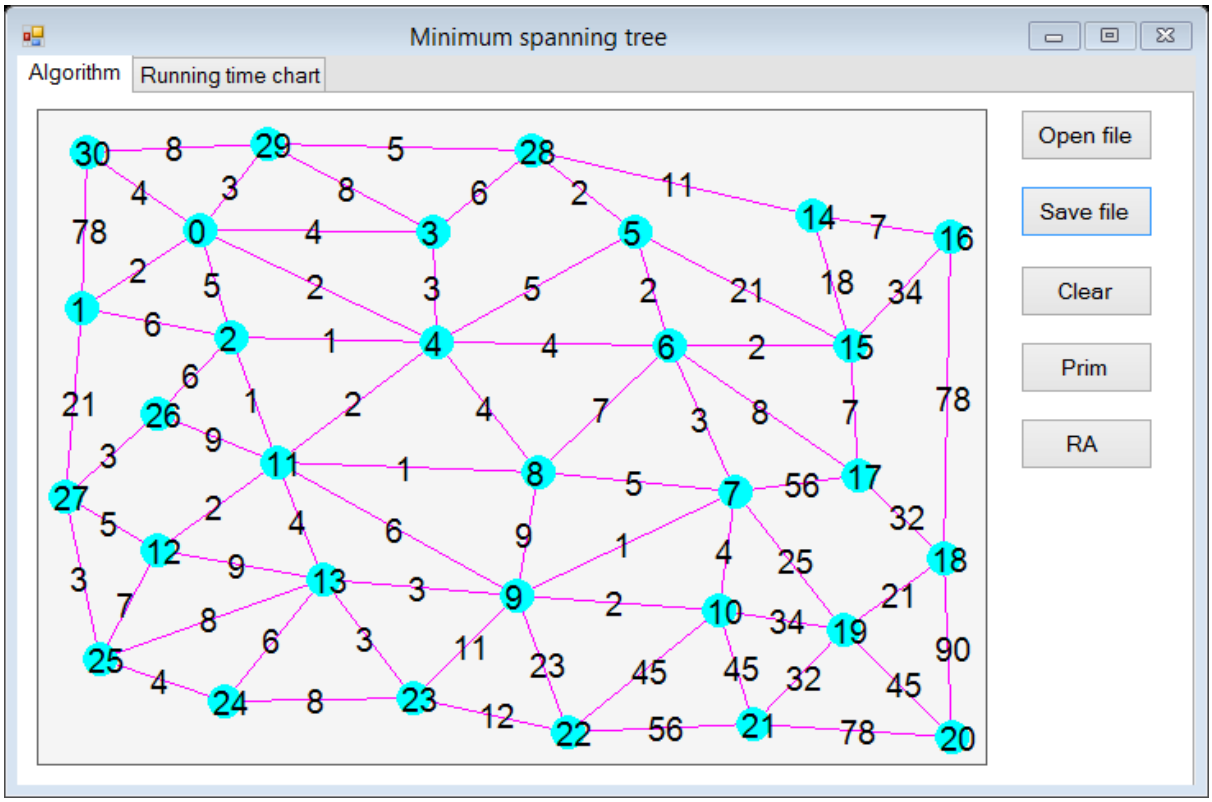
Hình 5.6.2: Giao diện chính của giải thuật tìm cây bao trùm nhỏ nhất

Muốn vẽ một đồ thị mới, chúng ta click chuột trên vùng (3), mỗi lần click ta sẽ nhận được một đỉnh của đồ thị tại vị trí click. Sau khi đã vẽ tất cả các đỉnh, chúng ta vẽ các cạnh bằng cách giữ phím Ctrl click chọn lần lượt 2 đỉnh thuộc cạnh cần vẽ. Một hộp thoại hiện lên ta nhập trọng số của cạnh này vào.

Sau khi vẽ xong đồ thị chúng ta có thể sử dụng nút Save file (5) để lưu lại đồ thị đang có, để tiện sử dụng cho lần sau. Khi click Save file, một cửa số hiện ra ta chọn

vị trí lưu, và đặt tên file, click Save. Một đồ thị đã được vẽ và lưu trữ trên giao diện chương trình như trong Hình 5.6.3.

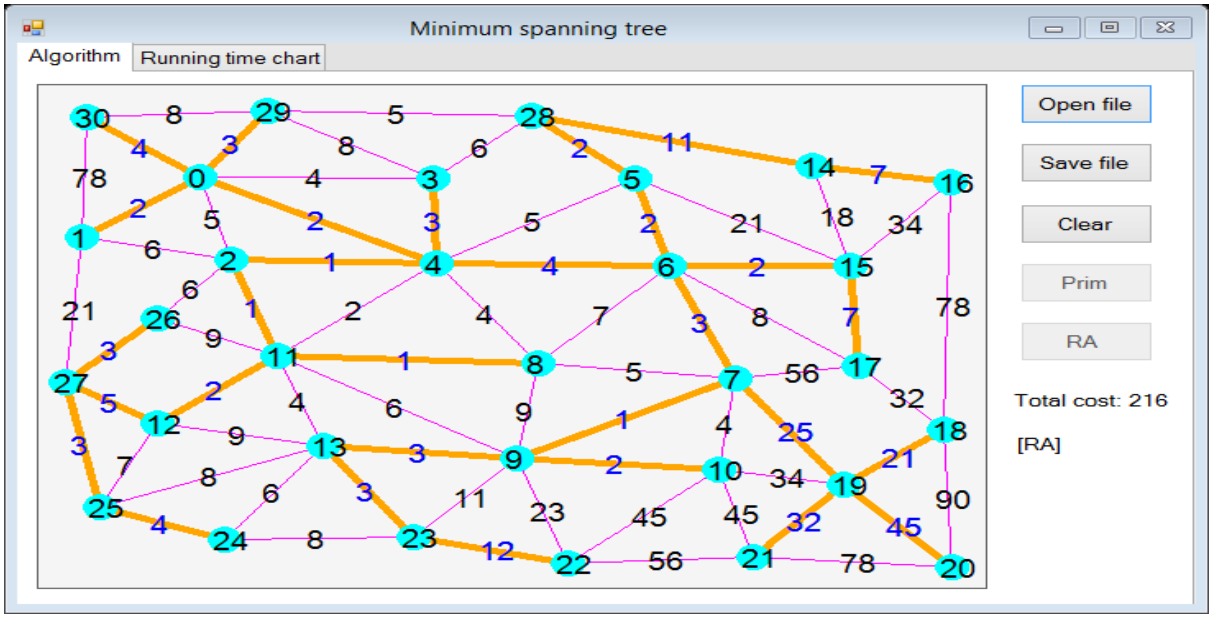
Chúng ta có thể mở đồ thị đã vẽ sẵn (đã được lưu trước đó) bằng cách click nút Open file (4). Một của sổ hiện ra, ta chọn file cần mở, rồi click Open. Đồ thị đã lưu sẽ hiện lại lên vùng (3), chúng ta có thể sử dụng để chạy giải thuật hay vẽ thêm cạnh vào đồ thị này. Nút Clear (6) cho phép ta xóa đồ thị đang có trên vùng (3).



Hình 5.6.3: Vẽ đồ thị đầu vào của chương trình tìm cây bao trùm nhỏ nhất

Nút Prim (7) và RA (8), được sử dụng (bằng cách click) tương ứng để tìm cây bao trùm nhỏ nhất theo giải thuật Prim và giải thuật ngẫu nhiên đã trình bày ở Chương 4. Sau khi thực hiện giải thuật chương trình sẽ hiển thị cây bao trùm nhỏ nhất tìm được, *tổng trọng số* (total cost) của cây bao trùm nhỏ nhất ở tab Algorithm và vẽ đồ thị thời gian chạy ở tab Running time chart.

Hình 5.6.4 biểu diễn cây bao trùm nhỏ nhất sau khi thực thi chương trình với giải thuật cổ điển và giải thuật ngẫu nhiên.



Hình 5.6.4: Cây bao trùm nhỏ nhất sau khi thực thi chương trình

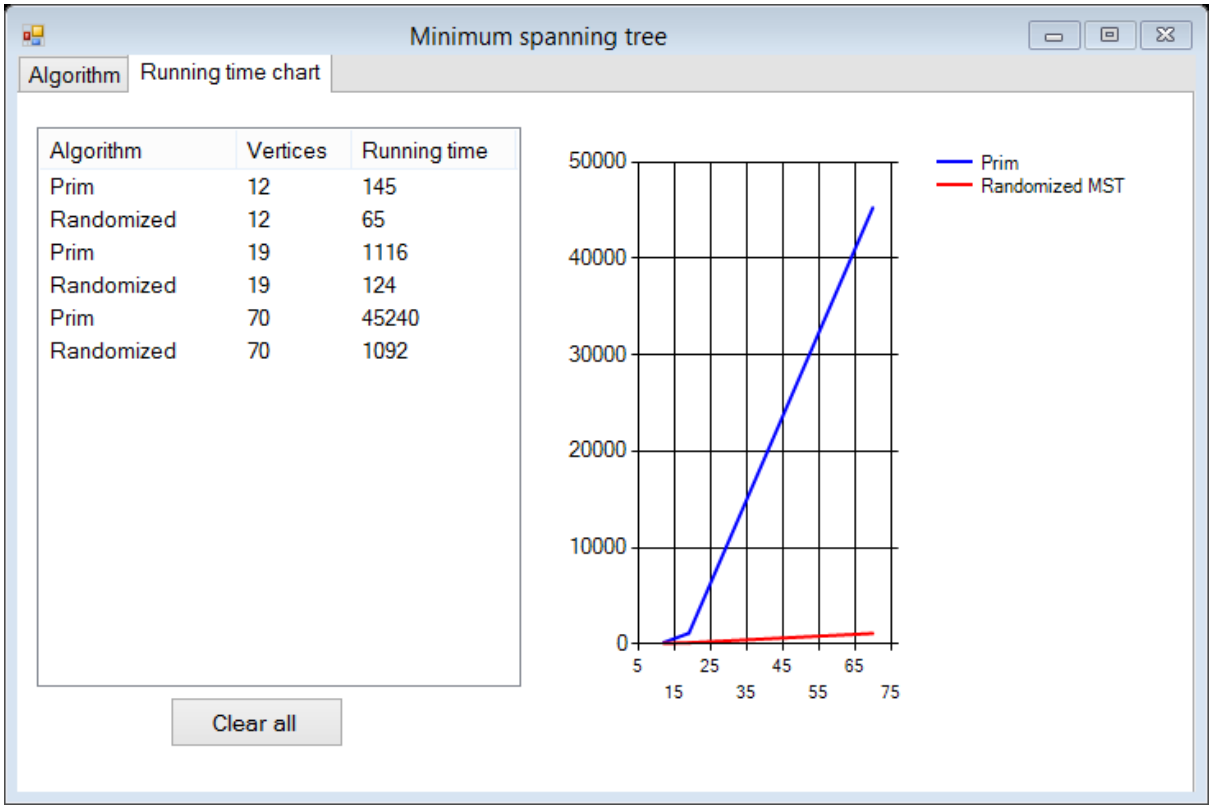
Thời gian chạy thực của giải thuật tìm cây bao trùm nhỏ nhất

Như chúng tôi đã trình bày ở Phần 4.6, giải thuật Prim có độ phức tạp $O(n + m\alpha(m,n))$, còn giải thuật ngẫu nhiên tìm cây bao trùm nhỏ nhất có độ phức tạp là $O(n + m)$. Bảng 5.6.1 dưới đây thể hiện thời gian chạy thực tế của 2 giải thuật với ba đồ thị có kích thước tăng dần.

Bảng 5.6.1: Thời gian chạy của giải thuật Prim và giải thuật ngẫu nhiên

Đồ thị	Thời gian chạy của giải thuật Prim	Thời gian chạy của giải thuật ngẫu nhiên
Đồ thị 1	116	124
Đồ thị 2	145	65
Đồ thị 3	45240	1092

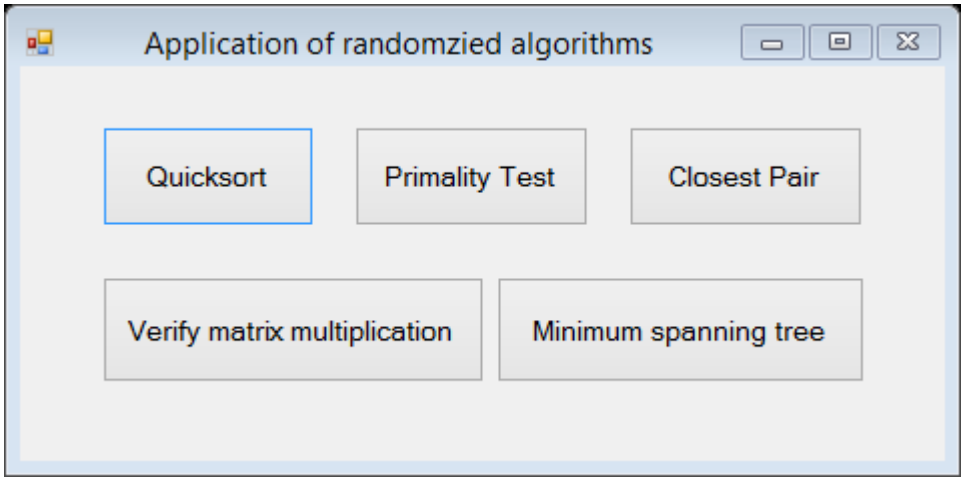
Từ dữ liệu của bảng 5.6.1 và bổ sung thêm một số thí nghiệm (chạy thử với một số đồ thị) chúng ta có thể vẽ đồ thị biểu diễn tốc độ tăng của thời gian chạy của hai giải thuật như Hình 5.6.5. Từ kết quả trong Bảng 5.6.1 cũng như Hình 5.6.5 chúng ta dễ dàng nhận thấy giải thuật ngẫu nhiên là hiệu quả hơn.



Hình 5.6.5: Đồ thị thời gian chạy thực của giải thuật tìm cây bao trùm nhỏ nhất

5.7. Giao diện chương trình ứng dụng

Cuối cùng, để hiện thực các giải thuật ngẫu nhiên áp dụng giải các bài toán như một hệ thống hoàn chỉnh, chúng tôi kết hợp các chương trình đã trình bày trong các phần trên vào một thực đơn như Hình 5.7.1. Khi muốn áp dụng giải bài toán nào chỉ cần click chọn nút tương ứng thì giao diện ứng dụng sẽ xuất hiện như đã trình bày ở các phần trên.



Hình 5.7.1: Giao diện của chương trình ứng dụng giải thuật ngẫu nhiên

5.8. Kết luận

Trong chương này, các giải thuật ngẫu nhiên để giải một số bài toán ứng dụng đề cập ở Chương 4 đã được hiện thực một cách hiệu quả. Kết quả so sánh thời gian chạy thực tế trên máy của các chương trình ứng dụng giải thuật ngẫu với các chương trình ứng dụng giải thuật cổ điển tương ứng đều cho thấy giải thuật ngẫu nhiên là tốt hơn. Điều này hoàn toàn phù hợp với kết quả đánh giá độ phức tạp thời gian lý thuyết của chúng. Đây là cơ sở và động lực cho các nghiên cứu và ứng dụng giải thuật ngẫu nhiên trong tương lai.

Chương 6

TỔNG KẾT VÀ ĐỀ NGHỊ

6.1. Tổng kết

Như đã trình bày trong Chương 1, nghiên cứu và xây dựng các giải thuật hiệu quả (độ phức tạp nhỏ, chi phí thời gian ít) là nhu cầu thường trực của các ứng dụng nói chung và các phần mềm máy tính nói riêng. Tuy nhiên do hạn chế về mặt kỹ thuật thiết kế và cách thức thực thi đơn định của giải thuật cổ điển, dẫn đến các giới hạn không thể vượt qua của độ phức tạp của chúng. Chẳng hạn, độ phức tạp thời gian của các giải thuật cổ điển tìm cặp điểm gần nhau nhất không thể nhỏ hơn $O(n\log_2 n)$. Điều này đã dẫn đến việc nghiên cứu và phát triển kỹ thuật thiết kế giải thuật ngẫu nhiên như là một kiểu giải thuật không đơn định nhằm giảm thiểu khối lượng tính toán của giải thuật và từ đó giảm độ phức tạp về thời gian của chúng. Có hai loại giải thuật ngẫu nhiên, Monte Carlo và Las Vegas. Một giải thuật ngẫu nhiên Monte Carlo là giải thuật có thời chạy luôn luôn cố định nhưng chỉ thành công với một xác suất nào đó. Một giải thuật ngẫu nhiên Las Vegas là giải thuật chắc chắn thành công nhưng thời gian chạy không xác định chắc chắn.

Đã có nhiều công trình khoa học trên thế giới nghiên cứu và ứng dụng các giải thuật ngẫu nhiên được công bố. Các giải thuật ngẫu nhiên được đề nghị không chỉ giúp giải các bài toán mới đòi hỏi từ thực tế mà còn giải cả các bài toán đang tồn tại trong khoa học một cách hiệu quả hơn so với các lời giải trước đây. Giải thuật ngẫu nhiên được ứng dụng rộng rãi trong các lĩnh vực như lý thuyết số, hình học tính toán, lý thuyết đồ thị, tối ưu tổ hợp v.v.

Phát triển các hệ thống tính toán lớn và hiệu quả luôn luôn là đòi hỏi cấp thiết của khoa học và thực tiễn, vì vậy việc nghiên cứu, xây dựng các giải thuật có chi phí thời gian nhỏ nói chung và giải thuật ngẫu nhiên nói riêng vẫn được tiếp tục nghiên cứu. Các vấn đề đã giải quyết và trình bày trong khóa luận này là một đóng góp trong lĩnh vực nghiên cứu và ứng dụng giải thuật ngẫu nhiên. Các kết quả đạt được trong khóa luận càng có ý nghĩa hơn khi việc nghiên cứu, phổ biến lý thuyết và ứng dụng giải thuật ngẫu nhiên tại Việt Nam còn rất hạn chế. Kết quả đạt được của khóa luận có thể được tóm lược như sau:

1. Hệ thống hóa các vấn đề liên quan đến giải thuật cổ điển như khái niệm, tính chất, ngôn ngữ biểu diễn, các phương pháp thiết kế và các kỹ thuật tính toán độ phức tạp của giải thuật.
2. Cung cấp một cách có chọn lọc, hệ thống các công cụ về toán học nói chung và lý thuyết xác suất nói riêng hỗ trợ cho việc tính toán độ phức tạp giải thuật nói chung và giải thuật ngẫu nhiên nói riêng.
3. Phát biểu và trình bày có hệ thống về khái niệm, tính chất, sự phân loại và khả năng ứng dụng của giải thuật ngẫu nhiên.
4. Đề xuất các giải thuật ngẫu nhiên hiệu quả để giải một số bài toán ứng dụng như sắp xếp dãy số, xác định số nguyên tố, kiểm tra phép nhân ma trận, tìm cặp điểm gần nhau nhất và tìm cây bao trùm nhỏ nhất.
5. Cuối cùng, hiện thực thành công các giải thuật ngẫu nhiên bằng một chương trình máy tính để giải các bài toán ứng dụng trong thực tế.

6.2. Đề nghị

Từ các nghiên cứu liên quan đã được đề cập và từ các kết quả của khóa luận này, chúng tôi đề nghị một số vấn đề và hướng nghiên cứu tiếp theo như sau:

1. Khóa luận là một nghiên cứu về giải thuật ngẫu nhiên, vì vậy việc trình bày các ứng dụng chưa đi sâu vào một lĩnh vực cụ thể. Do đó đề nghị đầu tiên là thực hiện một nghiên cứu để xây dựng các giải thuật ngẫu nhiên cho phép giải các bài toán trong một lĩnh vực chuyên sâu cụ thể như lý thuyết số, lý thuyết đồ thị, v.v.
2. Như đã trình bày ở trên, một giải thuật ngẫu nhiên Monte Carlo là giải thuật có thời chạy luôn luôn cố định nhưng chỉ thành công với một xác suất nào đó. Nghĩa là giải thuật ngẫu nhiên Monte Carlo có thể trả lời sai với một xác suất nhỏ nào đó. Vì vậy một đề nghị tiếp theo là nghiên cứu để chuyển một giải thuật ngẫu nhiên loại Monte Carlo thành giải thuật ngẫu nhiên Las Vegas (loại giải thuật ngẫu nhiên luôn cho kết quả đúng).
3. Hiện thực chương trình ứng dụng như một hệ thống tiện lợi, thân thiện và chuyên nghiệp hơn nhằm tích hợp thêm các ứng dụng để giải các bài toán khác.

TÀI LIỆU THAM KHẢO

[1] Karp Richard M. *An introduction to randomized algorithms*. Discrete Applied Mathematics, 34, 1991, 165-201.

[2] 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.

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

[4] Martin Dietzfelbinger. *A Reliable Randomized Algorithm for the Closest-Pair Problem*. Academic Press, 1997.

[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 and Its Applications*. 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] http://en.wikipedia.org/wiki/Freivalds%27_algorithm

[10] http://en.wikipedia.org/wiki/Randomized_algorithm

[11] http://www.codeproject.com/Articles/2728/C-BigInteger-Class

[12] http://www.wisdom.weizmann.ac.il/~robi/teaching/2013a-RandomizedAlgorithms