ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

_ꕥ_

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Dương Thị Phương Mai

TRAVELLING SALESMAN PROBLEM

Đồ Án Môn Học

Thuật Toán và Phương Pháp Giải Quyết Vấn Đề

TP HỒ CHÍ MINH – Năm 2014

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH

_ꕥ_

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Dương Thị Phương Mai

TRAVELLING SALESMAN PROBLEM

Ngành: Khoa Học Máy Tính

Đồ Án Môn Học

Thuật Toán và Phương Pháp Giải Quyết Vấn Đề

GIẢNG VIÊN HƯỚNG DẪN
PGS.TS. Đỗ Văn Nhơn

TP HỒ CHÍ MINH – Năm 2014

Mục Lục

MỞ ĐẦU

Bài toán Người du lịch, tìm đường đi ngắn nhất cho người thương nhân (salesman), hay còn gọi là người chào hàng xuất phát từ một thành phố, đi qua lần lượt tất cả các thành phố duy nhất một lần và quay về thành phố ban đầu với chi phí rẻ nhất, được phát biểu vào thế kỷ 17 bởi hai nhà toán học vương quốc Anh là Sir William Rowan Hamilton và Thomas Penyngton Kirkman, và được ghi trong cuốn giáo trình Lý thuyết đồ thị nổi tiếng của Oxford. Nó nhanh chóng trở thành bài toán khó thách thức toàn thế giới bởi độ phức tạp thuật toán tăng theo hàm số mũ (trong chuyên ngành thuật toán người ta còn gọi chúng là những bài toán NP-khó). Người ta bắt đầu thử và công bố các kết quả giải bài toán này trên máy tính từ năm 1954 (49 đỉnh), cho đến năm 2004 bài toán giải được với số đỉnh lên tới 24.978, và dự báo sẽ còn tiếp tục tăng cao nữa.

Bài tiểu luận này sẽ trình bày cách giải bài toán này theo hướng tiếp cận nhánh cận theo quy trình xây dựng giải pháp cho vấn đề. Với quy mô của một tiểu luận môn học thì ở đây chỉ trình bày thuật toán theo cách đơn giản nhất, cùng với một số cài đặt đi kèm.

Chương 1. GIỚI THIỆU TỔNG QUAN VỀ BÀI TOÁN VÀ ĐỘ PHỨC TẠP CỦA BÀI TOÁN

1.1 Vấn đề và bài toán

Vấn đề được xem như là một việc hoặc tình huống được coi là không mong muốn hoặc có hại và cần phải được xử lý và khắc phục.

- Một điều khó khăn để đạt được hoặc thực hiện.
- Một truy vấn bắt đầu từ những điều kiện được cho để điều tra hoặc chứng minh một thực tế, một kết quả, một luật

Trong cuộc sống của mình, con người hầu như lúc nào cũng phải giải quyết "vấn đề": từ giải một bài toán đố, trả lời các câu hỏi thi trắc nghiệm cho đến giải quyết các vấn đề liên quan đến kinh tế, xã hội... Dưới góc nhìn khoa học, vấn đề thường được thể hiện dưới dạng một bài toán. Bài toán là một loại vấn đề mà để giải quyết nó, chúng ta phải ít nhiều cần đến tính toán như các bài toán trong Vật lý, Hóa học, Xây dựng, Kinh tế...

Tuy số lượng vấn đề là vô hạn nhưng theo nhà toán học Pythagoras, mọi vấn đề con người cần giải quyết đều có thể chia làm 2 loại:

1. **Theorem**: Là vấn đề cần được khẳng định tính đúng-sai. Chúng ta thường quen với vấn đề này qua việc chứng minh các định lý trong toán học.

Ví dụ: Chứng minh gia tốc của chuyển động tròn đều là gia tốc hướng tâm

2. **Problem**: là vấn đề cần tìm giải pháp để đạt được một mục tiêu xác định từ những điều kiện ban đầu nào đó. Loại vấn đề này thường gặp trong các bài toán dựng hình, tổng hợp hóa học...

Ví dụ: Cho đồ thị G = (V,E), hãy tìm chu trình Euler của đồ thị này

Cả hai loại vấn đề nêu trên có thể biểu diễn bằng sơ đồ sau:

$$A \longrightarrow B$$

Trong đó:

A có thể là giả thiết, điều kiện ban đầu.

B có thể là kết luận, mục tiêu cần đạt được.

→ là suy luận, giải pháp cần xác định.

Như vậy, lập trình ở đây chính là →, tức là tìm cách giải quyết vấn đề - bài toán bằng một số hữu hạn các bước suy luận hoặc các thao tác hợp lý để xuất phát từ A, thông qua các bước thực hiện đó thì sẽ đạt được B.

Các thành phần của vấn đề:

(1) giả thiết,

(2) mục tiêu,

(3) các điều kiện ràng buộc hay phạm vi,

(4) cơ sở dữ liệu – thông tin – tri thức của vấn đề.

1.2 Độ phức tạp của bài toán - Các lớp bài toán

1.1.1. Khái niệm các loại thời gian tính

- Thời gian tính tốt nhất: là thời gian tính tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n.

- Thời gian tính tồi nhất: là thời gian tính tối đa cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào có kích thước n

- Thời gian tính trung bình: là thời gian tính cần thiết để thực hiện thuật toán trên một tập hữu hạn các bộ dữ liệu đầu vào có kích thước n. Thời gian tính trung bình được tính theo công thức sau:

- Thời gian tính trung bình = (Tổng thời gian tính tất cả các bộ dữ liệu có thể) / Số bộ dữ liệu.

1.1.2. Bài toán quyết định

Định nghĩa: *Bài toán quyết định là bài toán mà đầu ra chỉ có thể là 'yes' hoặc 'no'(đúng/sai, 0/1).*

Đối với một bài toán quyết định, có những bộ dữ liệu vào cho ra câu trả lời(đầu ra) là 'yes', chúng ta gọi đây là bộ dữ liệu 'yes', nhưng cũng có những bộ dữ liệu vào cho ra câu trả lời là 'no', chúng ta gọi những bộ dữ liệu này là bộ dữ liệu 'no'.

Rất nhiều các bài toán quyết định có một đặc điểm chung, đó là để xác nhận câu trả lời 'yes' đối với bộ dữ liệu vào 'yes' của chúng, chúng ta có thể đưa ra bằng chứng ngắn gọn dễ kiểm tra xác nhận câu trả lời 'yes' cho bộ dữ liệu vào 'yes' đó. Tính ngắn gọn dễ kiểm tra ám chỉ việc thời gian kiểm tra để đưa ra kết quả chỉ mất thời gian đa thức. Ví dụ về những bài toán quyết định kiểu này rất nhiều, sau đây là một số ví dụ:

- Bài toán kiểm tra tính hợp số: "Có phải số n là hợp số?", để xác nhận câu trả lời 'yes' cho đầu vào n, chúng ta có thể đưa ra một ước số b (1<b<n) của n. Để kiểm tra xem b có phải là ước số của n chúng ta có thể thực hiện phép chia n cho b sau thời gian đa thức. Trong ví dụ này , b là bằng chứng ngắn gọn (vì b<n) và dễ kiểm tra (có thuật toán đa thức để kiểm tra b đúng là ước số của n không).
- Đối với bài toán Hamilton, để xác nhận câu trả lời là 'yes' cho đồ thị đã cho G, chúng ta có thể đưa ra một chu trình Hamilton của đồ thị:

  $$v_1, v_2, v_3 \ldots\ldots v_n, v_1$$

  Việc kiểm tra dãy đỉnh nói trên có là chu trình Hamilton của đồ thị đã cho hay không có thể thực hiện sau thời gian đa thức. Khi đó chúng ta nói dãy đỉnh nói trên là bằng chứng ngắn gọn dễ kiểm tra để xác nhận câu trả lời 'yes' của bài toán Hamilton.

Một cách tương tự, có thể đưa ra khái niệm bằng chứng ngắn gọn dễ kiểm tra để xác nhận câu trả lời 'no'. Đối với một số bài toán, việc đưa ra bằng

chứng ngắn gọn dễ kiểm tra để xác nhận câu trả lời ‘no’ là dễ hơn so với việc đưa ra bằng chứng ngắn gọn xác định câu trả lời là ‘yes’.

1.1.3. Khái niệm quy dẫn

Định nghĩa: *Giả sử chúng ta có hai bài toán quyết định A và B. Chúng ta nói A có thể quy dẫn về B nếu như sau một thời gian tính đa thức nếu tồn tại một thuật toán thời gian đa thức R cho phép biến đổi bộ dữ liệu x của A thành bộ dữ liệu vào R(x) của B sao cho x là bộ dữ liệu yes của A khi và chỉ khi R(x) là dữ liệu vào yes của B.*

Nếu A quy dẫn về được B sau thời gian đa thức và B có thể giải được sau thời gian đa thức thì A cũng có thể giải được sau thời gian đa thức.

1.1.4. Lớp bài toán P

Định nghĩa: *Chúng ta gọi P là lớp các bài toán có thể giải được trong thời gian đa thức.*

Ví dụ: Bài toán cây khung nhỏ nhất giải được nhờ thuật toán Prim với thời gian $O(n^2)$ thuộc lớp bài toán P.

1.1.5. Lớp bài toán NP

Định nghĩa: *Ta gọi NP là lớp các bài toán quyết định mà để xác nhận câu trả lời ‘yes’ của nó ta có thể đưa ra bằng chứng ngắn gọn dễ kiểm tra.*

Ví dụ: Bài toán kiểm tra tính hợp số: “Có phải n là hợp số không?”, để xác nhận câu trả lời ‘yes’ cho đầu vào n ta có thể đưa ra một ước số b ($1< b < n$) của n. Để kiểm tra xem b có phải là ước số của n hay không ta có thể thực hiện phép chia n cho b sau thời gian đa thức. Trong ví dụ này dễ thấy b là bằng chứng ngắn gọn ($b<n$) và dễ kiểm tra (có thuật toán thời gian tính đa thức để kiểm tra xem b có là ước số của n).

1.1.6. Lớp bài toán Co-NP

Định nghĩa: *Ta gọi Co-NP là lớp các bài toán quyết định mà để xác nhận câu trả lời 'no' của nó ta có thể đưa ra bằng chứng ngắn gọn dễ kiểm tra.*

Ví dụ: Bài toán kiểm tra tính nguyên tố: "Có phải n là số nguyên tố không?", để đưa ra bằng chứng ngắn gọn dễ kiểm tra xác nhận câu trả lời 'no' cho đầu vào n ta có thể đưa ra một ước số b của n.

1.1.7. Lớp bài toán NP-Complete (NP-Đầy đủ)

Định nghĩa: *Một bài toán quyết định A được gọi là NP-Complete nếu như:*

- *A là một bài toán trong NP.*
- *Mọi bài toán trong NP đều có thể quy dẫn về A.*

Bổ đề: Giả sử bài toán A là NP-Complete, bài toán B thuộc NP, và bài toán A qui dẫn được về bài toán B. Khi đó bài toán B cũng là NP-Complete

1.1.8. Lớp bài toán NP-Hard (NP-Khó)

Một cách ngắn gọn có thể hiểu bài toán NP-Hard là bài toán mà không có thuật toán thời gian tính đa thức để giải nó trừ khi P = NP, mà chỉ có các thuật toán giải trong thời gian hàm mũ. Sau đây là định nghĩa chính thức của bài toán NP-Hard:

Định nghĩa: *Một bài toán A được gọi là NP-Hard nếu như sự tồn tại thuật toán đa thức để giải nó kéo theo sự tồn tại thuật toán đa thức để giải mọi bài toán trong NP.*

Một số bài toán NP-Hard tiêu biểu:

Bài toán bè cực đại (MaxClique): Cho một đồ thị vô hướng $G = (V, E)$. V là tập các đỉnh, E là tập các cạnh tương ứng các đỉnh trong V. Cần tìm bè lớn nhất của G. Bè là tập các đỉnh trong đồ thị mà đôi một có cạnh nốivới nhau (là một đồ thị con đầy đủ trong đồ thị G).

Bài toán tập độc lập (Independent set): Cho đồ thị vô hướng $G = (V, E)$ và số nguyên K, hỏi có thể tìm được tập độc lập S với $|S| \geq K$. Tập độc lập là tập các đỉnh trong đồ thị mà chúng đôi một không có cạnh nối với nhau.

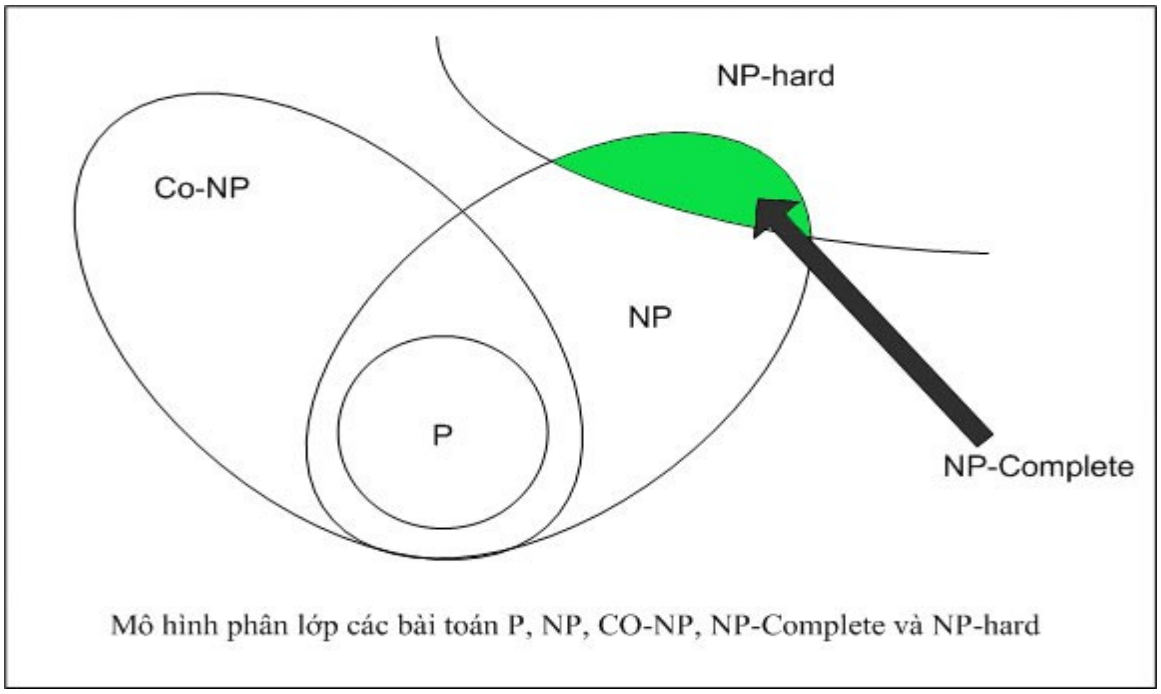
Bài toán phủ đỉnh (Vertex cover): Ta gọi một phủ đỉnh của đồ thị vô hướng $G = (V, E)$ là một tập con các đỉnh của đồ thị $S \subseteq V$ sao cho mỗi cạnh của đồ thị có ít nhất một đầu mút trong S. Bài toán đặt ra là: Cho đồ thị vô hướng $G = (V, E)$ và số nguyên k. Hỏi G có phủ đỉnh với kích thước k hay không?

Một cách không hình thức, có thể nói rằng nếu ta có thể giải được một cách hiệu quả một bài toán NP-Hard cụ thể, thì ta cũng có thể giải hiệu quả bất kỳ bài toán trong NP bằng cách sử dụng thuật toán giải bài toán NP-Hard như một chương trình con.

Từ định nghĩa bài toán NP-Hard có thể suy ra rằng mỗi bài toán NP-Complete đều là NP-Hard. Tuy nhiên một bài toán NP-Hard không nhất thiết phải là NP-Complete.

Cũng từ bổ đề nêu trên, ta có thể suy ra rằng để chứng minh một bài toán A nào đó là NP-Hard, ta chỉ cần chỉ ra phép qui dẫn một bài toán đã biết là NP-Hard về nó.

Sau đây là mô hình phân lớp các bài toán đã nêu trên.



Mô hình phân lớp các bài toán P, NP, CO-NP, NP-Complete và NP-hard

Từ phần trình bày trên, ta thấy có rất nhiều bài toán ứng dụng quan trọng thuộc vào lớp NP-Hard, và vì thế khó hy vọng xây dựng được thuật toán đúng hiệu quả để giải chúng. Do đó, một trong những hướng phát triển thuật toán giải các bài toán như vậy là xây dựng các thuật toán gần đúng.

Chương 2. GIỚI THIỆU HEURISTIC VÀ METAHEURISTIC

2.1 Heuristic

Heuristic: là những cách giải mang tính thông minh, hiệu quả và chấp nhận được.

- Thông minh: Cách giải hợp lý nhất có thể theo tư duy thông thường của con người
- Hiệu quả: độ phức tạp thấp hơn
- Chấp nhận được: đúng hoặc gần đúng nhưng chấp nhận được

Thuật giải Heuristic là một sự mở rộng khái niệm thuật toán. Nó thể hiện cách giải bài toán với các đặc tính sau :

- Thường tìm được lời giải tốt (nhưng không chắc là lời giải tốt nhất).
- Giải bài toán theo thuật giải Heuristic thường dễ dàng và nhanh chóng đưa ra kết quả hơn so với giải thuật tối ưu, vì vậy chi phí thấp hơn.
- Thuật giải Heuristic thường thể hiện khá tự nhiên, gần gũi với cách suy nghĩ và hành động của con người.

Có nhiều phương pháp để xây dựng một thuật giải Heuristic, trong đó người ta thường dựa vào một số nguyên lý cơ sở như sau:

- Nguyên lý vét cạn thông minh:
  Trong một bài toán tìm kiếm nào đó, khi không gian tìm kiếm lớn, ta thường tìm cách giới hạn lại không gian tìm kiếm hoặc thực hiện một kiểu dò tìm đặc biệt dựa vào đặc thù của bài toán để nhanh chóng tìm ra mục tiêu.
- Nguyên lý tham lam (Greedy):
  Lấy tiêu chuẩn tối ưu (trên phạm vi toàn cục) của bài toán để làm tiêu chuẩn chọn lựa hành động cho phạm vi cục bộ của từng bước (hay từng giai đoạn) trong quá trình tìm kiếm lời giải.
- Nguyên lý thứ tự :
  Thực hiện hành động dựa trên một cấu trúc thứ tự hợp lý của không gian khảo sát nhằm nhanh chóng đạt được một lời giải tốt.

- Hàm Heuristic:
  Trong việc xây dựng các thuật giải Heuristic, người ta thường dùng các hàm Heuristic. Đó là các hàm đánh giá thô, giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán tại mỗi bước giải. Nhờ giá trị này, ta có thể chọn được cách hành động tương đối hợp lý trong từng bước của thuật giải.

2.2 Metaheuristic

Metaheuristic là một cách gọi chung cho các giải thuật heuristic trong việc giải quyết các bài toán tổ hợp khó. Metaheuristic bao gồm những chiến lược khác nhau trong việc khám phá không gian tìm kiếm bằng cách sử dụng những phương thức khác nhau và phải đạt được sự cân bằng giữa tính đa dạng và chuyên sâu của không gian tìm kiếm

2.3 Thuật giải nhánh cận

Phương pháp (hay giải thuật) nhánh cận là một trong những phương pháp giải các bài toán liệt kê cấu hình có điều kiện tối ưu.

2.3.1. Bài toán tối ưu

- Bài toán yêu cầu tìm ra một phương án tốt nhất thỏa mãn một số yêu cầu ràng buộc nào đó – nghiệm của bài toán đạt giá trị max/min trong không gian nghiệm.
- Thuộc lĩnh vực Tối ưu toán học hoặc Quy hoạch toán học. Lời giải toán có thể khó => Sự vào cuộc của Tin học
- Hai hướng tiếp cận tìm lời giải tối ưu cho bài toán:

  + Tìm từng lời giải, khi hoàn tất một lời giải thì so sánh của nó với chi phí tốt nhất hiện có. Nếu tốt hơn thì cập nhật chi phí tốt nhất mới.

  + Với mỗi lời giải, khi xây dựng các thành phần nghiệm luôn kiểm tra điều kiện nếu đi tiếp theo hướng này thì có khả năng nhận được lời giải tốt hơn lời giải hiện có không? Nếu không thì thôi không đi theo hướng này nữa. => Nguyên lí nhánh cận (Branch and Bound)

2.3.2. Ý tưởng

- Nhánh cận (Branch and Bound): Thuật toán tìm lời giải cho các bài toán tối ưu dạng liệt kê cấu hình dựa trên nguyên lí đánh giá nhánh cận.
- Nguyên lí đánh giá nhánh cận: Sử dụng các thông tin đã tìm được trong lời giải của bài toán để loại bỏ sớm phương án không dẫn tới lời giải tối ưu
- Bản chất:

  + Sử dụng phương pháp quay lui nhưng tại mỗi bước đưa thêm thao tác đánh giá giá trị phương án hiện có.

  + Nếu đó là phương án tối ưu hoặc có hy vọng trở thành phương án tối ưu (tức là tốt hơn phương án hiện có) thì cập nhật lại phương án tối ưu hoặc đi tiếp theo hướng đó.

  + Trong trường hợp ngược lại thì bỏ qua hướng đang xét.

2.3.3. Mô hình

- Không gian của bài toán (tập khả năng) D={(x1,x2,…,xn)} gồm các cấu hình liệt kê có dạng (x1, x2, ……, xn).
- Mỗi cấu hình x sẽ xác định một giá trị hàm chi phí f(x)

  Nghiệm của bài toán: x= (x1,x2,…,xn) sao cho f(x)= giá trị tối ưu (max/min)
- Cho xi nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho xi xét khả năng chọn xi+1, xi+2…
- Tại mỗi bước i: Xây dựng thành phần xi
- Xác định xi theo khả năng v.

  + Tính chi phí lời giải nhận được. Nếu “tốt hơn” lời giải hiện thời thì chấp nhận xi theo khả năng v. Tiếp tục xác định xi+1,…đến khi gặp nghiệm

  + Nếu không có một khả năng nào chấp nhận được cho xi hoặc lời giải xấu hơn thì lùi lại bước trước để xác định lại thành phần xi-1.

Chương 3. BÀI TOÁN NGƯỜI ĐƯA THƯ

3.1 Các khái niệm cơ bản về đồ thị

4.1.1. Định nghĩa đồ thị

Đồ thị là một cấu trúc rời rạc gồm các đỉnh và các cạnh (vô hướng hoặc có hướng) nối các đỉnh đó. Người ta phân loại đồ thị tùy theo đặc tính và số các cạnh nối các cặp đỉnh của đồ thị. Nhiều bài toán thuộc những lĩnh vực rất khác nhau có thể giải được bằng mô hình đồ thị. Ta có thể dùng đồ thị để giải các bài toán như bài toán tính số các tổ hợp khác nhau của các chuyến bay giữa hai thành phố trong một mạng hàng không, hay để giải bài toán đi tham quan tất cả các đường phố của một thành phố sao cho mỗi đường phố đi qua đúng một lần, hoặc bài toán tìm số các màu cần thiết để tô các vùng khác nhau của một bản đồ.

3.1.1.1 Đồ thị vô hướng

Đồ thị vô hướng G là một cặp không có thứ tự G:=(V, E), trong đó:
V là tập các đỉnh hoặc nút
E là tập các cặp không thứ tự chứa các đỉnh phân biệt, được gọi là cạnh. Hai đỉnh thuộc một cạnh được gọi là các đỉnh đầu cuối của cạnh đó.

3.1.1.2 Đồ thị có hướng

Đồ thị có hướng G là một cặp có thứ tự G = (V, A), trong đó:
V là tập các đỉnh hoặc nút
A là tập các cặp có thứ tự chứa các đỉnh, được gọi là các cạnh có hướng hoặc cung. Một cạnh e = (x, y) được coi là có hướng từ x tới y; x được gọi là điểm đầu/gốc và y được gọi là điểm cuối/ngọn của cạnh.

3.1.1.3 Đơn đồ thị và đa đồ thị

- Đơn đồ thị (đồ thị đơn) là đồ thị mà không có khuyên và không có cạnh song song.

- Đa đồ thị là đồ thị mà không thỏa đồ thị đơn.
- Đa đồ thị có hướng là một đồ thị có hướng, trong đó, nếu x và y là hai đỉnh thì đồ thị được phép có cả hai cung (x, y) và (y, x).
- Đơn đồ thị có hướng (hoặc Đơn đồ thị có hướng) là một đồ thị có hướng, trong đó, nếu x và y là hai đỉnh thì đồ thị chỉ được phép có tối đa một trong hai cung (x, y) hoặc (y, x).

4.1.2. Bậc của đỉnh

Định nghĩa: Bậc của đỉnh v trong đồ thị G = (V,E), ký hiệu deg(v), là số các cạnh liên thuộc với nó, riêng khuyên tại một đỉnh được tính hai lần cho bậc của nó. Đỉnh v gọi là đỉnh treo nếu deg(v) = 1 và gọi là đỉnh cô lập nếu deg(v) = 0.
Mệnh đề: Cho đồ thị G = (V,E). Khi đó:
Mỗi cạnh e = (u,v) được tính một lần trong deg(u) và một lần trong deg(v). Từ đó suy ra tổng tất cả các bậc của các đỉnh bằng hai lần số cạnh.
Hệ quả: Số đỉnh bậc lẻ của một đồ thị là một số chẵn.

4.1.3. Bậc của đỉnh

Định nghĩa: Một đồ thị (vô hướng) được gọi là liên thông nếu có đường đi giữa mọi cặp đỉnh phân biệt của đồ thị. Một đồ thị không liên thông là hợp của hai hay nhiều đồ thị con liên thông, mỗi cặp các đồ thị con này không có đỉnh chung. Các đồ thị con liên thông rời nhau như vậy được gọi là các thành phần liên thông của đồ thị đang xét. Như vậy, một đồ thị là liên thông khi và chỉ khi nó chỉ có một thành phần liên thông.
Mệnh đề: Mọi đơn đồ thị n đỉnh ($n \geq 2$) có tổng bậc của hai đỉnh tuỳ ý không nhỏ hơn n đều là đồ thị liên thông. Nếu một đồ thị có đúng hai đỉnh bậc lẻ thì hai đỉnh này phải liên thông, tức là có một đường đi nối chúng.

4.1.4. Đồ thị Euler và đồ thị Hamilton

Đồ thị Euler

Cho đồ thị (có hướng) G = (V,E).

Đồ thị Hamilton

- Chu trình (có hướng) Hamilton là chu trình (có hướng) sơ cấp qua mọi đỉnh đồ thị.
- Đường đi (có hướng) Hamilton là đường đi (có hướng) sơ cấp qua mọi đỉnh đồ thị.

Như vậy mọi chu trình Hamilton có độ dài bằng số đỉnh, và mọi đường đi Hamilton có độ dài bằng số đỉnh trừ 1.

Đồ thị chứa chu trình (có hướng) Hamilton gọi là đồ thị Hamilton.

Định lí: Giả sử đồ thị G có chu trình Hamilton C. Khi đó

(i) Đồ thị G liên thông.

(ii) Mọi đỉnh của G có bậc lớn hơn hoặc bằng 2, và có đúng hai cạnh liên thuộc thuộc chu trình C.

(iii) Nếu xóa đi k đỉnh bất kỳ cùng các cạnh liên thuộc chúng, thì đồ thị còn lại sẽ có tối đa k thành phần liên thông.

Hệ quả: Giả sử đồ thị n đỉnh G có đường đi Hamilton P. Khi đó

(i) Đồ thị G liên thông.

(ii) Có ít nhất n–2 đỉnh bậc ≥ 2, và mỗi đỉnh có đúng hai cạnh liên thuộc thuộc đường đi P.

(iii) Nếu xóa đi k đỉnh bất kỳ cùng các cạnh liên thuộc chúng, thì đồ thị còn lại sẽ có tối đa k +1 thành phần liên thông.

3.2 Bài toán người đưa thư

4.2.1. Phát biểu bài toán và lịch sử bài toán người đưa thư

Bài toán người đưa thư (bài toán người đi giao hàng, bài toán người đi du lịch (TSP)): *Một người đưa thư xuất phát từ bưu điện là nơi làm việc*

của anh ta, anh ta muốn tìm một đường đi ngắn nhất đi qua tất cả các địa chỉ của khách hàng để giao thư mỗi địa điểm đúng một lần sau đó trở về địa điểm ban đầu là bưu điện khởi đầu.

Bài toán người du lịch (người đưa thư) (Travelling Salesman problem (TSP)) là một bài toán khá nổi tiếng trong lĩnh vực tối ưu tổ hợp được nghiên cứu trong lý thuyết khoa học máy tính. Nội dung của nó khá đơn giản, nó được phát biểu như sau: Cho một danh sách các thành phố và khoảng cách giữa chúng, nhiệm vụ là phải tìm đường đi ngắn nhất có thể mà chỉ thăm mỗi thành phố đúng 1 lần.

Nguồn gốc của bài toán người du lịch đến nay vẫn chưa rõ ràng. Một cuốn sách cho người du lịch từ năm 1832 đã đề cập tới vấn đề và bao gồm vài ví dụ về các đường đi từ đức qua Thụy Sỹ nhưng không chứa đựng ý nghĩa toán học nào.

Vấn đề toán học liên quan tới bài toán người du lịch đã được nhắc đến trong những năm 1800 bởi nhà toán học Ireland W. R. Hamilton và nhà toán học người Anh Thomas Kirkman. Trò chơi Icosian Game của Hamilton là một trò đố vui dựa trên cơ sở tìm chu trình Hamilton. Dạng tổng quát của bài toán TSP được nghiên cứu bởi các nhà toán học suốt những năm 1930 ở đại học Harvard, đáng chú ý là Karl Menger người đã định nghĩa bài toán, xem xét giải thuật brute-force và quan sát thấy tính không tối ưu của heuristic dựa trên láng giếng gần nhất.

Hassler Whitney ở đại học Princeton University là người đầu tiên đặt tên người du lịch cho bài toán không lâu sau đó.

Trong những năm 1950 và 1960 , bài toán trở nên ngày càng phổ biến trong khoa học ở châu Âu và Mỹ. Những đóng góp đáng chú ý được kể đến như George Dantzig, Delbert Ray Fulkerson và Selmer M. Johnson tại RAND Corporation ở Santa Monica, những người đã trình bày bài toán như bài toán số nguyên tuyến tính và phát triển phương thức cắt cho lời giải của nó.

Với những phương thức mới này họ đã giải được một thí dụ của bài toán với 49 thành phố để xây dựng một cách tối ưu và chứng minh rằng không còn đường đi nào ngắn hơn nữa. Trong những thập kỷ tiếp theo, bài toán được nghiên cứu bởi rất nhiều nhà nghiên cứu từ toán học, khoa học máy tính, hóa học, vật lý và những khoa học khác.

Nhiều thành tựu đã đạt được trong suốt những năm cuối thập kỷ 1970 và 1980, khi Grötschel, Padberg, Rinaldi và những người khác cố gắng giải một cách chính xác một thể hiện của bài toán với 2392 thành phố, sử dụng phương thức cắt và branch-and-bound.

Trong những năm 1990 Applegate, Bixby, Chvátal, và Cook đã phát triển chương trình Concorde mà đã được sử dụng nhiều trong việc giải các bài toán TSP cho đến nay . Gerhard Reinelt đã công bố thư viện TSPLIB vào năm 1991, đó là một tập các thể hiện của bài toán TSP với nhiều độ khó khác nhau, và đã được sử dụng bởi nhiều nhóm nghiên cứu khác nhau để so sánh kết quả. Năm 2005, Cook và những người khác đã tính được độ dài tối ưu cho chu trình với thể hiện của bài toán TSP lên tới 33,810 thành phố, được lấy ra từ bài toán xây dựng layout cho microchip, cho tới nay vẫn là thể hiện lớn nhất trong các thể hiện ở TSPLIB .Nhiều thể hiện khác với hàng triệu thành phố, lời giải tìm được có thể chứng minh nằm sai khác 1% so với lời giải tối ưu.

4.2.2. Mô hình hóa vấn đề

3.2.1.1 Vấn đề thực tế và vấn đề cần giải quyết

Khảo sát và thu thập dữ liệu, thông tin và tri thức (DIK)

- Thông tin về đường đi: một chiều, hai chiều...
- Thông tin về điểm đến (bưu điện)
- Thông tin về chi phí khi đi trên đường
- Hai bưu điện có thể nối với nhau bằng nhiều đường đi trực tiếp được không?
- Mật độ giao thông trên đường
- Có cầu vượt hay không?

Chọn lọc vấn đề và chuẩn hóa DIK, Xác định cơ sở DIK cho vấn đề

- Đường đi: 2 chiều
- Có thể có 1 đường đi hoặc không có đường đi nào giữa 2 bưu điện bất kì, và có thông tin chi phí cho đường đi này (>0)

Xác định phạm vi gây giới hạn của vấn đề

- Người đưa thư chỉ đi qua mỗi bưu điện đúng 1 lần và phải quay về điểm xuất phát
- Có thể không có đường nào đi qua tất cả các bưu điện

Thu thập mẫu vấn đề và phân lớp

- Danh sách bưu điện
- Danh sách đường đi nối giữa các bưu điện đó và trọng số tương ứng với đường đi

Mô tả giả thiết của vấn đề

- Đầu vào: danh sách bưu điện, danh sách đường đi nối giữa các bưu điện đó và trọng số tương ứng, bưu điện đầu tiên (bưu điện bắt đầu từ đó sẽ đi)
- Đầu ra: một đường đi bắt đầu từ bưu điện đầu tiên, đi qua tất cả các bưu điện và quay trở về lại điểm xuất phát kèm với tổng chi phí là thấp nhất, hoặc trả về kết quả là không tìm được đường đi này.

Mục tiêu của vấn đề

- Tìm được đường đi qua tất cả các bưu điện và trở về lại bưu điện đầu tiên sao cho tổng chi phí là thấp nhất

Mô tả các ràng buộc liên quan

- Phải có ít nhất một đường đi (có thể là gián tiếp) nối giữa 2 bưu điện bất kì
- Chi phí phải là số dương
- Số lượng bưu điện phải là hữu hạn

3.2.1.2 Xây dựng mô hình

Mô hình cho DIK

- Được mô hình hóa bằng đồ thị đơn vô hướng có trọng số dương. Trong đó:
    - Số đỉnh là số nguyên dương >= 2
    - Mỗi đỉnh thể hiện cho một bưu điện
    - Cạnh là một đường đi nối giữa 2 đỉnh bất kì trong đồ thị
    - Bậc của đỉnh x: Là tổng số cạnh liên thuộc với với đỉnh x, ký hiệu deg(x)
      G = (V, E)
      V = $\{x_1, \ldots, x_n\}$
      E = $\{e_1, \ldots, e_m\}$; e ∈ E liên kết x_i và x_j, w_i là trọng số cho cạnh e_i ($w_i > 0$)

Mô hình cho giả thiết

- Một đồ thị đơn vô hướng có trọng số dương
  G = (V, E)
  V = $\{x_1, \ldots, x_n\}$
  E = $\{e_1, \ldots, e_m\}$; e ∈ E liên kết x_i và x_j
- Điểm xuất phát là một đỉnh trong đồ thị x ∈ V

Mô hình cho mục tiêu

- Chu trình Hamilton ngắn nhất xuất phát từ x
    - Đường đi theo thứ tự các đỉnh như thế nào (x, x_3, x_5, ..., x)
    - Tổng trọng số

4.2.3. Thiết kế thuật toán / thuật giải

Chọn lựa phương pháp giải quyết vấn đề dựa trên những phương pháp đã biết

- Một trong những phương pháp đầu tiên người ta nghĩ đến để giải quyết bài toán này là thuật toán vét cạn. Thuật toán này tìm tất cả các chu trình Hamilton trong đồ thị, sau đó chọn một chu trình nhỏ nhất làm đáp án. Việc tìm chu trình Hamilton được làm theo phương pháp duyệt theo chiều sâu và kết hợp quay lui. Do quá trình duyệt rất sâu nên ta không dùng đệ quy mà dùng stack để khử đệ quy. Sử dụng một biến min để lưu lại tổng trọng số của chu trình Hamilton nhỏ nhất. Ban đầu min = ∞.

Chu trình Hamilton là chu trình đi qua hết tất cả các đỉnh và sau đó quay trở về đỉnh xuất phát do đó ta dùng một danh sách ChuaXet[] để lưu lại các đỉnh chưa xét, một biến Sum để lưu lại trọng số của chu trình hiện thời. Quá trình duyệt như sau:

- Ban đầu đưa đỉnh 1 vào stack, Sum = 0
- Lặp lại quá trình sau: Lấy (top) đỉnh từ stack ra là đỉnh i và trọng số wi, gán ChuaXet[i] là false và Sum = Sum + wi, nạp các đỉnh kề j và trọng số tương ứng với đỉnh đang xét mà ChuaXet[j] = true. Nếu i không có đỉnh kề thỏa yêu cầu, ta cộng trọng số của cạnh i->1 vào Sum và gán min bằng min(Sum, min).

Thuật giải vét cạn này cho ta một đáp án tối ưu, tuy nhiên độ phức tạp của nó là quá cao (n-1)! thuộc nhóm O(n!). Do đó, ta sẽ tìm một thuật giải khác tuy chỉ cho kết quả gần đúng nhưng độ phức tạp nhỏ hơn để có thể áp dụng được trong thực tế.

- Phương pháp nhánh cận: Thuật toán nhánh cận là một trong các phương pháp chủ yếu giải bài toán tối ưu tổ hợp. Tư tưởng cơ bản của nó là trong quá trình tìm kiếm ta phân hoạch các phương án của bài toán ra thành hai hay nhiều tập con như là các nút của cây tìm kiếm và cố gắng đánh giá cận cho các nút, loại bỏ những nhánh mà ta biết chắc chắn là không chứa phương án tối ưu.

Hình thành ý tưởng thiết kế thuật toán

- Trong bài toán người du lịch khi tiến hành tìm kiếm lời giải ta sẽ phân tập hành trình thành hai tập con: Một tập chứa cạnh (i,j) và tập không chứa cạnh này. Ta gọi việc đó là phân nhánh, mỗi tập con nói trên gọi là nhánh.
- Việc phân nhánh sẽ được dựa trên qui tắc hợp lý nào đó cho phép ta rút ngắn quá trình tìm kiếm phương án tối ưu. Sau khi phân nhánh ta sẽ tính cận dưới của hàm mục tiêu trong mỗi tập con nói trên. Việc tìm kiếm sẽ tìm trên tập con có cận dưới nhỏ hơn. Thủ tục sẽ tiếp tục cho đến khi thu được hành trình đầy đủ, tức là phương án của bài

toán người du lịch. Sau đó ta chỉ cần xét những tập con có cận dưới nhỏ hơn giá trị hàm mục tiêu tìm được.

Mô tả cụ thể: tác vụ, chiến lược...

Gọi C = {c_{ij}: i, j = 1, 2, ..., n} là ma trận chi phí.

Mỗi hành trình v = v(1)→v(2)→ ... →v(n−1)→v(n)→v(1) có thể viết dưới dạng v = (v(1), v(2)), (v(2), v(3)), ..., (v(n−1),v(n)), (v(n),v(1)) trong đó mỗi thành phần (v(i−1),v(i)) gọi là một cạnh của hành trình.

Rõ ràng tổng chi phí của một hành trình sẽ chứa đúng một phần tử trên mỗi dòng và mỗi cột của ma trận chi phí C = (cij). Do đó nếu ta trừ bớt mỗi phần tử của một dòng (hay một cột) đi cùng một giá trị thì chi phí của tất cả hành trình sẽ giảm đi một lượng, vì thế hành trình tối ưu sẽ không thay đổi. Vì vậy nếu tiến hành trừ bớt các phần tử của mỗi dòng và mỗi cột đi một hằng số sao cho thu được ma trận không âm và mỗi cột cũng như mỗi dòng chứa ít nhất một số 0 , thì tổng các hằng số trừ đi đó sẽ cho ta cận dưới của mọi hành trình. Thủ tục trừ bớt này gọi là thủ tục rút gọn, các hằng số trừ ở mỗi dòng (cột) gọi là hằng số rút gọn dòng (cột), ma trận thu được gọi là ma trận rút gọn.

Thủ tục rút gọn:

- Đầu vào : Ma trận chi phí C = (c_{ij})
- Đầu ra : Ma trận rút gọn và tổng hằng số rút gọn *Sum*
- Thuật toán:

  (i) Khởi tạo :

  Sum := 0 ; (chỉ áp dụng cho ma trận chi phí ban đầu)

  (ii) Rút gọn dòng:

  Với mỗi dòng r từ 1 đến n của ma trận C thực hiện:

  - Tìm phần tử $c_{rj} = \alpha$ nhỏ nhất trên dòng.
  - Trừ tất cả các phần tử trên dòng đi một lượng α.
  - Cộng dồn: *Sum := Sum* + α

  (iii) Rút gọn cột:

Với mỗi cột c từ 1 đến n của ma trận C thực hiện:

- Tìm phần tử $c_{ic} = \alpha$ nhỏ nhất trên cột.
- Trừ tất cả các phần tử trên cột đi một lượng α.
- Cộng dồn: *Sum* := *Sum* + α

Phân nhánh:

Giả sử ta chọn cạnh phân nhánh (r, s). Như vậy các hành trình sẽ được chia làm hai tập: P1 chứa các hành trình qua (r,s) và P2 chứa các hành trình không qua (r,s).

Nhánh tập P1: Cận dưới β với giá trị xuất phát có từ thủ tục rút gọn.

- Giảm cấp ma trận chi phí C bằng cách loại dòng r và cột s.
- Ngăn cấm tạo chu trình con :

Cấm cạnh (s,r) bằng cách đặt $c_{sr} = \infty$.

Nếu (r,s) là cạnh phân nhánh thứ hai trở đi thì phải xét các cạnh đã chọn nối trước và sau cạnh (r,s) thành dãy nối tiếp các cạnh và cấm tất cả các cạnh dạng (h,i) bằng cách đặt chi = ∞.

Rút gọn ma trận chi phí ta có cận dưới $\beta := \beta$ + (tổng hằng số rút gọn).

Ta có thể tiếp tục thủ tục phân nhánh theo nhánh này với ma trận chi phí đã được hiệu chỉnh và giảm 1 bậc. Việc chọn cạnh nào để phân nhánh ta sẽ bàn ở mục tiếp theo.

Nhánh tập P2:

- Cấm cạnh (r,s) bằng cách đặt $c_{rs} = \infty$.
- Thực hiện thủ tục rút gọn với ma trận chi phí tương ứng và tính cận dưới $\beta := \beta$ + (tổng hằng số rút gọn) cho nhánh.

Ta có thể tiếp tục thủ tục phân nhánh theo nhánh này với ma trận chi phí đã được hiệu chỉnh cùng cận dưới tương ứng.

Biểu diễn thuật toán dạng mã giả

```
program TravellingSalesman;
const  InputFile = 'TOURISM.INP';
OutputFile = 'TOURISM.OUT';
max = 20;
maxC = 20 * 100 + 1;{+∞}
var
C: array[1..max, 1..max] of Integer; {Ma trận chi phí}
X, BestWay: array[1..max + 1] of Integer; {X để thử các khả năng,
BestWay để ghi nhận nghiệm}
T: array[1..max + 1] of Integer; {Ti để lưu chi phí đi từ X1 đến
Xi}
Free: array[1..max] of Boolean; {Free để đánh dấu, Free[i]= True
nếu chưa đi qua tp i}
m, n: Integer;
MinSpending: Integer; {Chi phí hành trình tối ưu}
procedure Enter;
var    i, j, k: Integer; f: Text;
begin
Assign(f, InputFile); Reset(f);
ReadLn(f, n, m);
for i := 1 to n do {Khởi tạo bảng chi phí ban đầu}
for j := 1 to n do
if i = j then C[i, j] := 0 else C[i, j] := maxC;
for k := 1 to m do
begin
ReadLn(f, i, j, C[i, j]);
C[j, i] := C[i, j]; {Chi phí như nhau trên 2 chiều}
end;
Close(f);
end;
procedure Init; {Khởi tạo}
begin
FillChar(Free, n, True);
Free[1] := False; {Các thành phố là chưa đi qua ngoại trừ thành
phố 1}
X[1] := 1; {Xuất phát từ thành phố 1}
T[1] := 0; {Chi phí tại thành phố xuất phát là 0}
MinSpending := maxC;
end;
procedure Try(i: Integer); {Thử các cách chọn xi}
var j: Integer;
begin
for j := 2 to n do {Thử các thành phố từ 2 đến n}
if Free[j] then {Nếu gặp thành phố chưa đi qua}
begin
X[i] := j; {Thử đi}
T[i] := T[i - 1] + C[x[i - 1], j]; {Chi phí := Chi phí bước trước
+ chi phí đường đi trực tiếp}
if T[i] < MinSpending then {Hiển nhiên nếu có điều này thì C[x[i -
1], j] < +∞ rồi}
if i < n then {Nếu chưa đến được xn}
```

```
begin
Free[j] := False; {Đánh dấu thành phố vừa thử}
Try(i + 1); {Tìm các khả năng chọn xi+1}
Free[j] := True; {Bỏ đánh dấu}
end
else
if T[n] + C[x[n], 1] < MinSpending then {Từ xn quay lại 1 vẫn tốn
chi phí ít hơn trước}
begin {Cập nhật BestConfig}
BestWay := X;
MinSpending := T[n] + C[x[n], 1];
end;
end;
end;
procedure PrintResult;
var    i: Integer; f: Text;
begin
Assign(f, OutputFile); Rewrite(f);
if MinSpending = maxC then WriteLn(f, 'NO SOLUTION')
else
for i := 1 to n do Write(f, BestWay[i], '->');
WriteLn(f, 1);
WriteLn(f, 'Cost: ', MinSpending);
Close(f);
end;
begin
Enter;
Init;
Try(2);
PrintResult;
end.
```

4.2.4. Chứng minh tính đúng đắn

(Bỏ qua)

4.2.5. Phân tích thuật toán / thuật giải

Thuật giải nhánh cận tuy không tìm được lời giải tối ưu như thuật toán vét cạn, nhưng nó có một ưu điểm lớn hơn nhiều là đã giảm độ phức tạp của thuật toán vét cạn, và do đó ta có thể hiện thực được phương pháp này trong thực tế khi đồ thị có số đỉnh lớn.

TÀI LIỆU THAM KHẢO

[1] Bùi Đức Dương, *Bài Giảng Trí Tuệ Nhân Tạo,* Khoa CNTT Đại học Nha Trang.

[2] Nguyễn Cam, Chu Đức Khánh: *Lý thuyết đồ thị*. NXB TP.HCM 1999.

[3] Trần Đan Thư: *Lý thuyết đồ thị và ứng dụng*, Đại học Quốc gia TPHCM, 2007.

[4] Trần Quốc Chiến: *Toán rời rạc*, Đại học Đà Nẵng, 2007.