ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Toán – Cơ – Tin học

##### Hà Vũ Long

##### Đàm Thị Hồng Nhung

#### **BÀI TOÁN**

#### **TÌM ĐƯỜNG ĐI NGẮN NHẤT**

Ngành Máy tính và khoa học thông tin

(Chương trình đào tạo chuẩn)

**Hà Nội - 2021**

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Toán – Cơ – Tin học

##### Hà Vũ Long

##### Đàm Thị Hồng Nhung

#### **BÀI TOÁN**

#### **TÌM ĐƯỜNG ĐI NGẮN NHẤT**

Tiểu luận môn học Thiết kế và đánh giá thuật toán

Ngành Máy tính và khoa học thông tin

(Chương trình đào tạo chuẩn)

##### Cán bộ hướng dẫn: PGS.TS Nguyễn Thị Hồng Minh

**Hà Nội - 2021**

**LỜI CẢM ƠN**

*Chúng em xin gửi lời cảm ơn chân thành tới cô Nguyễn Thị Hồng Minh, giảng viên môn học Thiết kế và đánh giá thuật toán, đồng thời cô cũng là người đồng hành cùng chúng em trong hai môn học chuyên ngành khác…*

*Cảm ơn cô đã truyền đạt cho chúng em những kiến thức về chuyên ngành hữu ích và giá trị để giúp chúng em hoàn thiện được nghiên cứu này. Cảm ơn cô với những câu chuyện mà cô kể đã xóa tan bao mệt mỏi của những buổi học với thời tiết không dễ chịu lắm, cảm ơn cô vì những bài học cuộc sống, những trò chơi thú vị giữa giờ…*

*Chúng em xin chúc cô luôn luôn mạnh khỏe, công tác tốt.*

*Vì kiến thức của các thành viên trong nhóm còn hạn chế, hoàn thành báo cáo này nhóm chúng em không tránh khỏi những sai xót, kính mong nhận được những ý kiến nhận xét từ cô và các bạn.*

*Em xin trân thành cảm ơn!*

**LỜI MỞ ĐẦU**

Theo trang wikipedia, lý thuyết đồ thị (*graph theory*) nghiên cứu các tính chất của đồ thị, là cấu trúc toán học được sử dụng để mô hình hóa mối quan hệ giữa các cặp đối tượng. Đồ thị biểu diễn được rất nhiều cấu trúc, nhiều bài toán thực tế có thể được biểu diễn bằng đồ thị. Sự phát triển của các thuật toán để xử lý đồ thị là một trong những mối quan tâm của nghành khoa học máy tính.

Những vấn đề trong lý thuyết đồ thị được sinh ra từ các vấn đề thực tiễn như tâm đồ thị, đồ thị phẳng, luồng vận tải, đường đi, chu trình… Từ đó cũng có những bài toán mang tính chất “kinh điển” và ứng dụng cao, có thể kể đến như bài toán tìm đồ thị con, tô màu đồ thị hay các bài toán tìm đường đi. Với yêu cầu tìm kiếm một bài toán có ứng dụng cao, phân tích và tìm các thuật toán giải quyết bài toán, nhóm chúng em lựa chọn “*Bài toán tìm đường đi ngắn nhất*” thuộc nhóm các bài toán tìm đường đi của lý thuyết đồ thị, để nghiên cứu.

1. **Mục tiêu nghiên cứu.**

Tìm hiểu về “*Bài toán tìm đường đi ngắn nhất*” cùng các ứng dụng của nó, các thuật toán thuộc các phương pháp đã được học trong môn học Thiết kế và đánh giá thuật toán, từ đó đưa ra các nhận xét, đánh giá các thuật toán đó.

1. **Đối tượng nghiên cứu và phạm vi nghiên cứu**
   1. **Đối tượng nghiên cứu**

Đối tượng nghiên cứu của đề tài là “*Bài toán tìm đường đi ngắn nhất*” và các thuật toán giải quyết bài toán đó.

* 1. **Phạm vi nghiên cứu**

Các thuật toán giải quyết bài toán đường đi ngắn nhất và các lý thuyết liên quan.

1. **Phương pháp nghiên cứu**

Sử dụng phương pháp nghiên cứu tài liệu (video, slide bài giảng và các mục trên internet liên quan tới đề tài nghiên cứu) để thu thập thông tin nhằm phân tích, hệ thống lý thuyết, các thuật toán về đường đi ngắn nhất và đánh giá các thuật toán.

1. **Cấu trúc của luận văn**

Ngoài các phần mở đầu, kết luận, mục lục, các tài liệu tham khảo…, trong luận văn gồm:

Phần 1: Cơ sở lý thuyết giải quyết bài toán

Phần 2: Phân tích, thiết kế các thuật toán giải quyết bài toán

Phần 3: Kết luận và hướng phát triển

1. **Ý nghĩa khoa học của nghiên cứu**
   1. **Tìm đường đi giữa các vị trí xác định**

Google maps, MapQuest…

* 1. **Tìm giải pháp tối ưu cho các đối tượng trừu tượng**

Thời gian ngắn nhất để thực hiện công việc…

* 1. **Nghiên cứu về vấn đề vận chuyển**

Sinh học, Y học, Kinh doanh, Công nghệ…

* 1. **Mạng lưới giao thông**

**LỜI CAM ĐOAN**

*Chúng em xin cam đoan:*

1. *Những nội dung trong báo cáo này là do chúng tôi thực hiện dưới sự hướng dẫn trực tiếp từ cô PGS.TS Nguyễn Thị Hồng Minh.*
2. *Mọi tham khảo dùng trong báo cáo đều được trích dẫn rõ ràng tên tác giả tên công trình, thời gian, địa điểm công bố.*
3. *Mọi sao chép không hợp lệ, vi phạm quy chế đào tạo, hay gian trá, chúng em xin chịu hoàn toàn trách nhiệm.*

Sinh viên,

*Đàm Thị Hồng Nhung*

*Hà Vũ Long*

**MỤC LỤC**

[PHẦN 1: BÀI TOÁN VÀ CƠ SỞ LÝ THUYẾT GIẢI QUYẾT BÀI TOÁN 1](#_Toc74957362)

[I. Một số lý thuyết về đồ thị 1](#_Toc74957363)

[1. Khái niệm đồ thị 1](#_Toc74957364)

[2. Đường đi, chu trình và tính liên thông của đồ thị 1](#_Toc74957365)

[3. Biểu diễn đồ thị 2](#_Toc74957366)

[4. Bài toán đường đi ngắn nhất trên đồ thị 3](#_Toc74957367)

[II. Một số các phương pháp giải quyết bài toán 4](#_Toc74957368)

[1. Phương pháp vét cạn (Brute Force) 4](#_Toc74957369)

[2. Phương pháp tham lam(Greedy) 5](#_Toc74957370)

[3. Phương pháp quy hoạch động(Dynamic Programming) 6](#_Toc74957371)

[PHẦN 2: PHÂN TÍCH, THIẾT KẾ CÁC THUẬT TOÁN GIẢI QUYẾT BÀI TOÁN 8](#_Toc74957372)

[I. Bài toán đường đi ngắn nhất nguồn đơn 8](#_Toc74957373)

[1. Bài toán 8](#_Toc74957374)

[2. Phân tích bài toán 8](#_Toc74957375)

[3. Giải quyết bài toán 8](#_Toc74957376)

[4. So sánh, đánh giá thuật toán Dijkstra và thuật toán Bellman-Ford 15](#_Toc74957377)

[II. Bài toán đường đi ngắn nhất giữa một cặp đỉnh 16](#_Toc74957378)

[1. Bài toán 16](#_Toc74957379)

[2. Giải quyết bài toán 16](#_Toc74957380)

[III. Bài toán đường đi ngắn nhất giữa mọi cặp đỉnh 16](#_Toc74957381)

[1. Bài toán 16](#_Toc74957382)

[2. Phân tích bài toán 16](#_Toc74957383)

[3. Giải quyết bài toán 16](#_Toc74957384)

[4. So sánh các thuật toán giải quyết bài toán đường đi ngắn nhất cho mọi cặp đỉnh**.** 23](#_Toc74957385)

[PHẦN 3: KẾT LUẬN 24](#_Toc74957386)

[TÀI LIỆU THAM KHẢO 25](#_Toc74957387)

# PHẦN 1: BÀI TOÁN VÀ CƠ SỞ LÝ THUYẾT GIẢI QUYẾT BÀI TOÁN

## Một số lý thuyết về đồ thị

### Khái niệm đồ thị

* 1. **Đồ thị**

Đồ thị *G = (V,E)* bao gồm một tập hữu hạn *V* các đỉnh (hay nút) và một tập hữu hạn *E* các cặp đỉnh mà ta gọi là cung ( hay cạnh).

* 1. **Định nghĩa đồ thị vô hướng**

Đồ thị vô hướng *G = (V,E)* bao gồm *V* là tập các đỉnh và *E* là tập các cặp đỉnh không có thứ tự gọi là các cung.

* 1. **Định nghĩa đồ thị có hướng**

Đồ thị có hướng *G=(V,E)* bao gồm *V* là tập các đỉnh và *E* là tập các cặp đỉnh có thứ tự gọi là các cung.

* 1. **Đồ thị có trọng số**

Đồ thị mà mỗi cung gắn với một giá trị nào đó được gọi là đồ thị có trọng số, các giá trị đó được gọi là trọng số hoặc độ dài của cung ( hay chi phí đường đi ).

### Đường đi, chu trình và tính liên thông của đồ thị

* 1. **Đường đi**

Đường đi độ dài *n* từ đỉnh *u* đến đỉnh *v*, trong đó *n* là số nguyên dương, trên đồ thị vô hướng *G = (V, E)* là dãy

***x0 ,x1 ,x2,… xn-1 ,xn***

trong đó *u = x0 , v = xn , (xi, xi+1) E, i = 1, 2, …, n-1*

Đường đi nói trên còn có thể hiểu diễn dưới dạng dãy các cạnh:

***(x0 ,x1), ( x1, x2),… (xn-1 ,xn)***

Đỉnh *u* gọi là đỉnh đầu, còn đỉnh *v* gọi là đỉnh cuối của đường đi. Đường đi được gọi là đơn nếu như không có cạnh nào bị lặp lại.

Đường đi ngắn nhất từ đỉnh *u* đến đỉnh *v* là đường đi có độ dài *n* là nhỏ nhất so với các đường nối hay đỉnh *u* với *v.*

* 1. **Chu trình**

Đường đi có đỉnh đầu trùng với đỉnh cuối (tức là *u = v*) được gọi là chu

trình. Chu trình đơn là chu trình không có cạnh nào bị lặp lại.

Khái niệm đường đi và chu trình trên đồ thị có hướng được định nghĩa hoàn toàn tương tự như trường hợp đồ thị vô hướng, chỉ khác là ta có chú ý đến hướng trên các cung.

* 1. **Tính liên thông**

Đồ thị vô hướng *G = (V, E)* được gọi là liên thông nếu luôn tìm được

đường đi giữa hai đỉnh bất kỳ của nó.

Đồ thị có hướng *G = (V, E)* được gọi là liên thông mạnh nếu luôn tìm

được đường đi giữa hai đỉnh bất kỳ của nó.

Đồ thị có hướng *G = (V, E)* được gọi là liên thông yếu nếu đồ thị vô

hướng ứng với nó là đồ thị vô hướng liên thông.

### Biểu diễn đồ thị

* 1. **Ma trận kề**

Xét đơn đồ thị vô hướng *G = (V, E)*, với tập đỉnh *V = {1, 2,..., n}*, tập cạnh

*E = {e1, e2, … en}*. Ma trận kề A biểu diễn đồ thị G là một ma trận vuông kích thước *n\*n.* Các phần tử của ma trận A gồm các giá trị 0 và 1.

*A = {aij : i, j = 1, 2,…,n}*

Các phần tử được xác định theo qui tắc sau đây:

*aij = 0* nếu *(i, j) E*  và *aij = 1* nếu *(i, j) E* với  *i, j = 1, 2,…,n*

Ma trận kề của đồ thị có hướng được định nghĩa một cách hoàn toàn tương tự

Trong trường hợp đồ thị có trọng số, thay vì ma trận kề, để biểu diễn đồ thị ta sử dụng ma trận trọng số

*C[i,j] = c(i,j)* nếu *(i, j) E*

*C[i,j] =* nếu *(i, j) E*

trong đó *c(i,j)* là trọng số của cạnh *e = (i,j),* sốtùy từng trường hợp cụ thể, có thể dặt bằng một trong các giá trị sau: 0,

Ưu điểm lớn nhất của phương pháp biểu diễn đồ thị bằng ma trận kể (hoặc ma trận trọng số) là để trả lời câu hỏi: Hai đỉnh *u*, *v* có kề nhau trên đồ thị hay không, chúng ta chỉ phải thực hiện một phép so sánh.

Nhược điểm lớn nhất của phương pháp này là: không phụ thuộc vào số cạnh của đồ thị, ta luôn phải sử dụng n2 đơn vị bộ nhớ để lưu trữ ma trận kề của nó.

* 1. **Ma trận liên thuộc**

Xét đồ thị *G = (V, E)*, với tập đỉnh *V = {1, 2,..., n}*, tập cạnh *E = {e1, e2, … en}* là đơn đồ thị có hướng. Xây dựng ma trận *A = {aij : i = 1, 2,…,n; j = 1, 2,…,m },* trong đó

*1,* nếu đỉnh *i* là đỉnh đầu của cung *ej*

*aij = -1,* nếu đỉnh *i* là đỉnh cuối của cung *ej*

*0,* nếu đỉnh *i* không là đầu mút của cung *ej*

Ma trận *A* xây dựng theo qui tắc vừa nêu được gọi là ma trận liên thuộc đỉnh-cạnh.

Ma trận liên thuộc đỉnh-cạnh làm ột trong những cách biểu diễn rất hay được sử dụng trong các bài toán liên quan đến đồ thị có hướng mà trong đó phải xử lý các cung của đồ thị.

### Bài toán đường đi ngắn nhất trên đồ thị

Như đã đề cập ở phần mở đầu, bài toán đường đi ngắn nhất có những ứng dụng thực tế quan trọng.

Bài toán đường đi ngắn nhất trên đồ thị có hướng *G = (V,E)*, *|V| = n*,

*|E| = m*, trong đó *V* là tập đỉnh, *E* là tập các cung của đồ thị *G*, mỗi cung được gán một trọng số, nghĩa là mỗi cung *(u,v) E* tương ứng với một số thực *a(u,v)* gọi là trọng số của nó. Trong ứng dụng cụ thể cho từng bài toán thì trọng số của một cung có thể là độ dài cạnh *(u,v)*, có thể là chi phí đi từ *u* đến *v* cũng có thể là thời gian đi từ *u* đến *v*... Nếu *(u,v) E*, ta đặt *a(u,v) = .* Nếu dãy *v0, v1, …, vp* là một đường đi trên *G* thì độ dài của nó được định nghĩa là:

(*vi* -1 ,*vi*),

nghĩa là độ dài của một đường đi chính là tổng các trọng số của các cung trên đường đi đó. Với đồ thị vô hướng ta hoàn toàn có thể chuyển thành đồ thị có hướng bằng cách thay mỗi cạnh *(u, v)* bằng hai cung *(u, v)* và cung *(v, u)* có cùng trọng số là trọng số của cạnh *(u, v)*.

***Bài toán đường đi ngắn nhất dạng tổng quát*** là bài toán tìm đường đi giữa hai đỉnh sao cho tổng trọng sốcủa các cạnh tạo nên đường đi là ngắn nhất.

Cụ thể, đường đi từ đỉnh *u* đến đỉnh *v* là

*p là đường đi từ đỉnh u đến đỉnh v*

*nếu không có đường đi từ đỉnh u đến đỉnh v*

Bài toán có bốn biến thể phổ biến:

* **Bài toán đường đi ngắn nhất nguồn đơn**:
  + Tìm các đường đi ngắn nhất từ đỉnh nguồn *v* đến mọi đỉnh khác trong đồ thị.
  + Các thuật toán giải quyết: Thuật toán Dijkstra, thuật toán Bellman-Ford,…
* **Bài toán đường đi ngắn nhất một đích đến:** 
  + Tìm những đường đi ngắnnhất từ tất cả các đỉnh trong đồ thị đến một đỉnh

đích duy nhất *v.* Chúng ta có thể đưa bài toán về bài toán đường đi ngắn nhất nguồn đơn bằng cách đảo ngược hướng của các vòng cung của đồ thị có hướng.

* **Bài toán đường đi ngắn nhất giữa một cặp đỉnh:**
  + Tìm đường đi ngắn nhấtgiữa đỉnh *u* và đỉnh *v* trong đồ thị.
  + Các thuật toán giải quyết: Thuật toán Dijkstra, thuật toán A\*.
* **Bài toán đường đi ngắn nhất giữa mọi cặp đỉnh:** 
  + Tìm đường đi ngắn nhấtgiữa mỗicặp đỉnh *v*, *v’* trong đồ thị.
  + Các thuật toán giải quyết: Thuật toán [Floyd-Warshall](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Floyd-Warshall), thuật toán Johnson.

Ngoài ra, một bài toán có liên quan là *bài toán người bán hàng –* bài toán tìm đường đi ngắn nhất đi qua tất cả các đỉnh, mỗi đỉnh đúng một lần và trở về đỉnh xuất phát, thuộc lớp các bài toán NPC.

## Một số các phương pháp giải quyết bài toán

Sau đây sẽ là một số phương pháp có liên quan tới các thuật toán giải quyết bài toán, thuộc phạm vi môn học *Thiết kế và đánh giá thuật toán.*

### Phương pháp vét cạn (Brute Force)

* 1. **Ý tưởng**
* Vét cạn(Brute Force) là chiến lược tiếp cận xét qua tất cả các trường hợp có thể

xảy ra để tìm kết quả, xây dựng lời giải từ dưới lên

* Độ phức tạp thời gian của phương pháp này là lũy thừa
  1. **Mô hình**

Tập khả năng *D = {(x1, x2 , …, xn)}*

*xi Di*, trong đó  *Di* = *{ dij | j = 1..mi}* là tập hữu hạn

Các khả năng có trong *D* có thể đúng hoặc sai

*f : D {true, false}*

Nghiệm của bài toán là *x = (x1, x2 , …, xn)* nếu *f(x) = true*

* **Input:** Tập *D* chứa các đối tượng của bài toán(tập khả năng)
* **Output:** Tập nghiệm của bài toán *x* = *{x1, x2 , …, xn}, xi ϵ D* sao cho *f(x)* là giá trị tối ưu (min/max)
* **Diagram**

*BruteForce(D)≡*

***for*** *(x1 = d11 .. d1m1)*

***for*** *(x2 = d21 .. d2m2)*

*......*

***for*** *(xn = dn1 .. dnmn)*

***if*** *(f(x1,x2,...xn))*

*<Solution x=(x1,x2,...xn)>;*

***End****.*

### Phương pháp tham lam(Greedy)

* 1. **Ý tưởng**
* Thuật toán tham lam là thuật toán tối ưu hóa tổ hợp. Thuật toán tìm

kiếm, lựa chọn giải pháp tối ưu địa phương ở mỗi bước đi với hy vọng tìm được giải pháp tối ưu toàn cục và không quan tâm tới tương lai.

* Có hai tính chất mang lại hiệu quả cho phương pháp là:
  + Sự lựa chọn tham
  + Cấu trúc con tối ưu
  1. **Mô hình**
* **Input**: Tập *A* chứa các đối tượng của bài toán
* **Output**: Tập nghiệm của bài toán *S* = *{x1, x2 , …, xn}, xi ϵ A* sao cho *f(S)* là giá trị tối ưu (min/max)
* **Functions:**
  + Chọn giải pháp tốt nhất ở mỗi bước: *x = BestSelect(A),*
  + Kiểm tra giải pháp đó có tối ưu hay không: *Acceptable(S,x),*
  + Chấp nhận là nghiệm: *Integrate(S,x),*
  + Kiểm tra tập nghiệm đã xây dựng đủ hay chưa:*Is\_solution(S).*
* **Diagram:**

*Greedy (A) ≡*

*S = ∅ ;*

***while*** *( A ≠ ∅ and !Is\_solution(S) )*

*x = BestSelect(A);*

*A = A / {x};*

***if*** *(Acceptable(S, x))*

*Integrate(S, x);*

***endwhile****;*

***return*** *S;*

***End****.*

### Phương pháp quy hoạch động(Dynamic Programming)

* 1. **Ý tưởng**
* Qui hoạch động là phương pháp :
  + Giải quyết bài toán bằng cách chia bài toán lớn thành các bài toán con, nhỏ hơn ( chỉ giải một lần)
  + Lưu lại các giá trị có được từ bài toán con vào bảng
  + Kết hợp lời giải từ các bài toán con để đưa ra kết quả bài toán ban đầu
  + Sử dụng phương thức Bottom-Up, xây dựng lời giải từ dưới lên
* Ưu điểm của phương pháp là thời gian thực hiện nhanh do không phải tốn thời gian giải lại một số bài toán con đã được giải.
* Trong một số trường hợp, qui hoạch động không mang lại hiệu quả:
  + Không tìm được không thức truy hồi
  + Số lượng các bài toán con cần giải quyết và lưu trữ kết quả là rất lớn
  + Sự kết hợp lời giải của các bài toán con chưa chắc cho ta lời giải của bài toán ban đầu.
  1. **Các yếu tố cơ bản của phương án qui hoạch động :**
  + Công thức truy hồi
  + Cơ sở qui hoạch động
  + Bảng phương án
  + Kết quả tối ưu của bài toán
  + Truy vết tìm nghiệm
  1. **Các bước xây dựng thuật toán**

Để xây dựng thuật toán theo phương pháp qui hoạch động ta thực hiện:

* Thiết kế thuật toán
  + Nhận dạng bài toán giải bằng qui hoạch động
  + Xây dựng công thức truy hồi
  + Xây dựng cơ sở qui hoạch động
* Triển khai thuật toán
  + Xây dựng bảng phương án
  + Tìm kết quả tối ưu – nghiệm của bài toán
  + Truy vết, liệt kê thành phần nghiệm
  1. **Đánh giá độ phức tạp**
* Độ phức tạp thời gian
  + Thời gian tính toán:

*T(n) = O(nk)* , trong đó k là số chiều của bảng phương án

* + Thời gian truy vết tìm nghiệm: Độ phức tạp của thuật toán quay lui
* Độ phức tạp không gian**:** Kích thước bảng phương án.

# PHẦN 2: PHÂN TÍCH, THIẾT KẾ CÁC THUẬT TOÁN GIẢI QUYẾT BÀI TOÁN

## Bài toán đường đi ngắn nhất nguồn đơn

### Bài toán

Bài toán đường đi ngắn nhất nguồn đơn được mô tả như sau:

*Cho đồ thị G với tập đỉnh V và tập các cạnh E (đồ thị có hướng hoặc vô hướng). Cho trước một đỉnh xác định s, gọi là đỉnh nguồn. Tìm đường đi từ đỉnh s đến các đỉnh của G với chi phí thấp nhất có thể.*

### Phân tích bài toán

Dữ liệu đầu vào của bài toán là đồ thị *G = (V,E)*

* Có thể có hướng hoặc vô hướng, trên đó có ghi trong số là chi phí đường đi. Các cặp đỉnh không có cạnh nối thì trọng số là
* Đỉnh xuất phát là đỉnh *s V*

Đưa ra kết quả:

* + - Tất cả đường đi ngắn nhất từ đỉnh *s* đến mọi đỉnh của *G*

### Giải quyết bài toán

* 1. **Thuật toán Dijkstra**

Thuật toán *Dijkstra* là một thuật toán giải quyết bài toán tìm đường đi ngắn nhất trong đồ thị có cạnh mang trọng số không âm.

* + 1. **Phân tích bài toán hướng phương pháp tham lam**

Ý tưởng: Duy trì hai tập, tập *S* chứa các đỉnh mà đường đi nhỏ nhất từ nguồn *s* đến các đỉnh đó đã biết, tập *Q* chứa các đỉnh chưa có trong S. Tại mỗi bước, thêm vào *S* đỉnh

*u Q* nếu ước tính đường đi từ nguồn đến *u* là nhỏ nhất. Cập nhật lại khoảng cách đường đi ngắn nhất ứng với các đỉnh liền kề với *u*.

* *Bước 1 : Khởi tạo*
  + Tập *Q* chứa các đỉnh của *V*
  + Tập *S* chứa các đỉnh được chọn sau mỗi bước, ban đầu *S = ∅*
  + Độ dài hay chi phí đường đi từ đỉnh *i* đến đỉnh *j* là *a(i,j)* với *i* và *j* là hai đỉnh kề nhau.
  + *D[i]* chứa chi phí đường đi ngắn nhất từ đỉnh xuất phát *s* đến đỉnh *i*, khởi tạo khoảng cách từ nguồn đến tất cả các đỉnh là vô hạn và khoảng cách đến nguồn bằng 0.
* *Bước 2* *:* *Lặp cho tới khi Q không còn phần tử nào* :
  + Lấy *u* từ tập *Q* với *D[u]* là chi phí nhỏ nhất so với chi phí ứng với các đỉnh có trong *Q*
  + Thêm *u* vào tập S
  + Với mỗi cạnh (*u,v)*, *v* là đỉnh không thuộc tập *Q*, nếu *D[u] + a(u,v)* nhỏ hơn *D[v]* thì *D[v] = D[u] + a(u,v)*
* *Bước 3* *: Xuất kết quả:*
  + Kết thúc khi tập *S* chứa tất cả các đỉnh của tập *V* và đưa ra độ dàingắn nhất *D[z]* từ đỉnh xuất phát v đến đỉnh z
  + Truy vết tìm đường đi
    1. **Mô tả dữ liệu**
* Đánh chỉ số đỉnh từ *1..n*
* Biểu diễn đồ thị *G* bằng ma trận kề *M = (aij)* cỡ *n\*n, aij* = *Cost* nếu có cạnh nối đỉnh *i* với đỉnh *j* với chi phí *Cost* , *Cost* = + ∞ nếu không có đường đi
* Mảng *Before[u]* lưu trữ các đỉnh kề trước đỉnh *u*, thực hiện truy vết đường đi.
  + 1. **Lược đồ**

Try(C) ≡

**for** (v = 2..n): D[v] = ∞;

D[1] = 0;

S = ∅ ;

Q = V{ v0 , v1, …, vn }

**While**( Q ≠)

u = extractMin(Q) với D[u] là chi phí nhỏ nhất hiện tại

S = S Ս {u}

**for** each edge(v,u) E and v not in Q

**if** (D[u] + a(u,v) < D[v] )

D[v] = D[u] + a(u,v)

Before[v] = u

**endWhile;**

**End**.

Để truy vết tìm đường đi, thực hiện đoạn mã đệ qui:

printPath(Before[], i)

**if** i is source:

return;

**endif**;

printPath(Before, Before[j]);

<Enter result>;

* + 1. **Phân tích thuật toán**
       1. **Chứng minh tính đúng**

***Bổ đề 1*** : *Thiết lập bất đẳng thức d[v] (s,v) với mọi đỉnh v V luôn đúng trong bất kỳ trình tự nào của bước cập nhật lại d[v].*

***Chứng minh :***

* **Cơ sở:** Khởi tạo
  1. *d[s] = 0, (s,s) = 0* với *s* là đỉnh nguồn

*d[s] (s,s)*

* 1. *d[v] = , (s,v)* với *v s*

*d[v] (s,v)*

* **Phản chứng:** Giả sử, tồn tại đỉnh *v V* là đỉnh đầu tiên mà *d[v] (s,v)* và

đỉnh *u* là đỉnh mà tại đó *d[v]* được cập nhật lại giá trị : *d[v] = d[u] + a(u,v)*

* + Theo điều giả sử, ta có :

*d[v] < (s,v)*

* + Tính chất bất đẳng thức trong tam giác:

*(s,v) (s,u) + (u,v)*

* + Đường đi ngắn nhất luôn nhỏ hơn hoặc bằng trọng số cạnh *(u,v)*

*(s,u) + (u,v) (s,u) + a(u,v)*

* + *(s,u) + a(u,v) d[u] + a(u,v)* ( *v* là đỉnh đầu tiên vi phạm)

*d[v] + a(u,v) (s,v)* trái với giả sử.

Như vậy ***Bổ đề 1*** được chứng minh.

***Bổ đề 2****:* *Giả thiết s ...uv là đường đi ngắn nhất. Từ đó, nếu*

*d[u] (s,u) và đỉnh u liền kề với đỉnh v bởi cạnh (u,v) thì d[v] (s,v) sau bước cập nhật lại d[v].*

***Chứng minh:***

* Ta thấy rằng *(s,v) = (s,u) + a(u,v)*
* Giả sử *d[v] (s,v)* trước khi cập nhật lại *d[v]* (Điều ngược lại đã được chứng

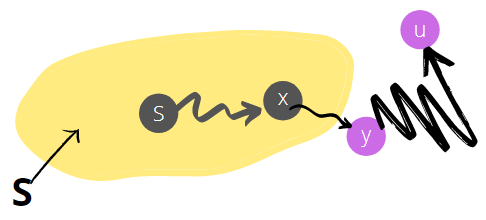
minh ở bổ đề 1). Sau đó bước kiểm tra *d[v] > d[u] + a(u,v)* thành công, do *(s,v) = (s,u) + a(u,v) = d[u] + a(u,v)* và thuật toán cập nhật lại :

*d[v] = d[u] + a(u,v) = (s,v)*

***Định lý****:* *Thuật toán Dijkstra kết thúc với d[v] = (s,v) với mọi đỉnh v V.*

***Chứng minh:*** Định lý luôn đúng với *d[v] = (s,v)* với mọi đỉnh *v V* khi đỉnh *v* được thêm vào tập *S.*

* Giả sử *u*  là đỉnh đầu tiên được thêm vào *S* và *d[u] > (s,u).* Đỉnh *y* là đỉnh đầu

tiên thuộc tập *V- S* và đỉnh *y* thuộc đường đi ngắn nhất từ *s* đến *u.* Đỉnh *x* là đỉnh kề trước của đỉnh *y*

*Hình 1: Minh họa tập S trước khi thêm đỉnh u*

* Từ đỉnh *u* là đỉnh đầu tiên vi phạm, ta có *d[x] = (s,x).* Khi đỉnh *x* được thêm

vào tập *S,* cạnh *(x,y)* đang được xét trong bước cập nhật lại, điều đó có nghĩa là

*d[y] = (s,y) (s,u) < d[u]*. Nhưng *d[u] d[y]* do cách chọn đỉnh *u*, điều này vô lý.

Vậy từ ***Bổ đề 1*** , ***Bổ đề 2*** và ***Định lý***, thuật toán được chứng minh.

* + - 1. **Độ phức tạp thuật toán**

Độ phức tạp thời gian của thuật toán khi cài đặt bằng

* + Array: *O(V2)*
    1. **Cải tiến thuật toán**

Để cải tiến thuật toán Dijkstra, đã có rất nhiều nghiên cứu và các bài báo khoa học trên kháp thế giới đưa ra, trong nghiên cứu này chỉ nói về cải tiến thuật toán bằng cài đặt Binary heap và Fibonacci heap với độ phức tạp thời gian được đưa ra ở phần trước.

* + - 1. **Thuật toán Dijkstra và Binary heap**

Binary heap: *O(ElogV)*

* + - 1. **Thuật toán Dijkstra và Fibonacci heap**

Fibonacci heap: *O(E+ VlogV)*

* 1. **Thuật toán Bellman Ford**

Sử dụng thuật toán Bellman-Ford để giải quyết bài toán tìm đường đi ngắn nhất trong đồ thị, tương tự với thuật toán Dijkstra nhưng có thể giải quyết bài toán có chi phí âm.

Điểm đặc biệt của đồ thị có trọng số âm là có thể có sự xuất hiện của chu trình âm, là chu trình mà tổng trọng số các cạnh nhỏ hơn 0. Sự xuất hiện của chu trình âm làm cho bài toán trở nên không xác định.

* + 1. **Phân tích bài toán hướng phương pháp qui hoạch động**

Ý tưởng : Giả sử rằng không có chu trình âm, nếu đã tiến hành tính toán các đường đi ngắn nhất với hầu hết các cạnh *i* thì một lần lặp lại trên các tất cả các cạnh đảm bảo đưa ra được đường đi ngắn nhất với các cạnh lớn hơn (*i + 1*). Điều đó có nghĩa là, tính toán theo phương thức bottom-up: Đầu tiên, tính toán đường đi ngắn nhất có nhiều nhất một cạnh; Sau đó tính toán với tối đa 2 cạnh….; Có thể tối đa |*V*| - 1 cạnh trong bất kỳ đường đi nào.

* *Bước 1 : Khởi tạo*
  + Độ dài hay chi phí đường đi từ đỉnh *i* đến đỉnh *j* là *a(i,j)*
  + *D[i]* chứa chi phí đường đi ngắn nhất từ đỉnh xuất phát *s* đến đỉnh *i*, khởi tạo khoảng cách từ nguồn đến tất cả các đỉnh là vô hạn và khoảng cách đến nguồn bằng 0.
* *Bước 2* *:* *Lặp |V|-1 lần, tính toán khoảng cách ngắn nhất* :
* Với mỗi cạnh (u, v), nếu *D[v] > D[u] + a(u,v)*, cập nhật lại giá trị *D[v] = D[u] + a(u,v)*
* Kết thúc khi ta không cập nhật hay làm tốt hơn *D[i]* nào,

với *i = 1..n.*

* *Bước 3* : *Kiểm tra đồ thị có chu trình âm hay không và in kết quả*
  + Với mỗi cạnh *(u,v)*, nếu *D[v] > D[u] + a(u,v)* thì đồ thị có chu trình âm, ngược lại *D[v] = (s,v)* với mọi đỉnh *v V.*
  + Truy vết tìm đường đi ngắn nhất

Ta có:

1. **Hàm tối ưu *D(i)*:** Chi phí đường đi nhỏ nhất từ đỉnh nguồn *s* đến đỉnh *i*

*( i ϵ V)*

1. **Công thức truy hồi**: *D(j) = D(i) + a(i,j) nếu D(j) > D(i) + a(i,j)*

*a(i,j)* là chi phí đường đi từ đỉnh *i* đến đỉnh *j* và *(i,j) E*

1. **Cơ sở qui hoạch động**: *D(s) = 0, D(i) =* ∞ ( *i ϵ V\{v}, s là đỉnh nguồn*)
   * 1. **Mô tả dữ liệu**
     + Đánh chỉ số đỉnh từ *1..n*
     + Biểu diễn đồ thị *G* bằng ma trận kề *M = (aij)* cỡ *n\*n, aij* = *Cost* nếu có

cạnh nối đỉnh *i* với đỉnh *j* với chi phí *Cost* , *Cost* = + ∞ nếu không có đường đi

* + - Mảng *Before[v]* lưu trữ các đỉnh kề trước đỉnh *v*
    1. **Lược đồ**

Try(C)

**for** (v = 2..n): D[v] = ∞;

D[1] = 0

**for** k = 1 to |V|-1

**for** each edge(u,v) E

**if** D[v] > D[u] + a(u,v)

D[v] = D[u] + a(u,v)

Before[v] = u

**endif**;

**endfor**;

**endfor**;

**for** each edge(u,v) E

**if** D[v] > D[u] + a(u,v)

Report that a negative-weight cycle exists

**endif**;

**endfor**

**End**.

At the end, d[v] = (s, v), if no negative-weight cycles.

Lược đồ truy viết đường đi

printPath(Before[], i)

**if** i is source:

return;

**endif**;

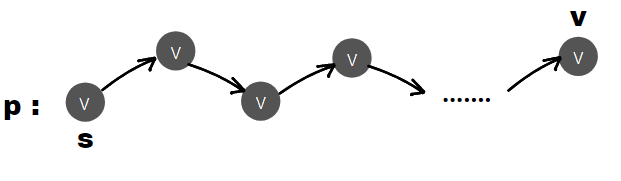
printPath(Before, Before[j]);

<Enter result>;

* + 1. **Phân tích thuật toán**
       1. **Chứng minh tính đúng**

***Định lý****: Nếu đồ thị G = (V,E) không chứa chu trình âm thì sau khi thực hiện thuật toán Bellman-Ford, d[v] = (s,v) với mọi đỉnh v V.*

***Chứng minh:*** Đỉnh bất kỳ *v V*, xét p là đường đi ngắn nhất từ đỉnh *s* đến đỉnh *v* với số cạnh tối thiểu.



k

0

1

2

3

*Hình 2: Biểu diễn đường đi ngắn nhất p*

* Từ *p* là đường đi ngắn nhất, ta có:

*(s,vi) = (s,vi-1) + a(vi-1, vi ).*

* Đầu tiên, *d[v0] =* 0 = *(s,v0)* và *d[v0]* không thay đổi trong các bước cập nhật lại,
  + Sau 1 lần qua *E*, ta có *d[v1] =* *(s,v1)*
  + Sau 2 lần qua *E*, ta có *d[v2] =* *(s,v2)*
  + Sau 3 lần qua *E*, ta có *d[v3] =* *(s,v3)*
  + **….**
  + Sau *k* lần qua *E*, ta có *d[vk] =* *(s,vk)*
* Từ đồ thị *G* khôngchứa chu kỳ âm, *p* là đường đi đơn.
* Đường đi đơn dài nhất có số cạnh nhỏ hơn |*V*| - 1

***Hệ quả:*** *Nếu có một giá trị d[v] không hội tụ sau khi qua* |*V*| - 1 *lần lặp thì tồn tại một chu trình âm trong đồ thị G*

* + - 1. **Độ phức tạp thuật toán**

Độ phức tạp thời gian của giải thuật là *O(VE)*

Độ phức tạp không gian của giải thuật là *O(V)*

* + 1. **Cải tiến thuật toán**

 Mặc dù cho đến nay chưa có thuật toán nào có thời gian tốt hơn O(VE) về mặt lý thuyết, ta vẫn có thể tăng tốc thuật toán Bellman-Ford trong thực tế. Nhận xét thấy trong giả mã Bellman-Ford ở trên, ta không quan tâm đến thứ tự các đỉnh hay cung mà ta sẽ nới lỏng.

Năm 1970, Yen đưa ra một thứ tự để nới lỏng các đỉnh, giảm số vòng lặp ngoài cùng xuống còn cỡ V/2. Năm 2012, Bannister và Eppstein chứng minh rằng nếu thứ tự các đinh nới lỏng được chọn ngẫu nhiên thì sau V/3 vòng lặp, với xác suất cao, chúng ta sẽ tìm được đường đi ngắn nhất.

**4. So sánh, đánh giá thuật toán Dijkstra và thuật toán Bellman-Ford**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **Dijkstra** | **Bellman-Ford** |
| **Tính chất đồ thị đưa vào** | **Đồ thị vô hướng** | Trọng số dương | Trọng số dương |
| **Đồ thị có hướng** | Trọng số dương | Trọng số có thể âm hoặc dương |
| **Độ phức tạp thời gian** | | O(E+ VlogV) | O(VE) |
| **Độ phức tạp không gian** | | O(V) | O(V) |
| **Hướng tiếp cận** | | Phương pháp tham lam | Phương pháp qui hoạch động |

## Bài toán đường đi ngắn nhất giữa một cặp đỉnh

### Bài toán

*Cho đồ thị có trọng số G với tập đỉnh V và tập các cạnh E (đồ thị có hướng hoặc vô hướng). Cho trước đỉnh xác định u và v, tìm đường đi từ đỉnh u đến đỉnh v của G với chi phí thấp nhất có thể.*

### Giải quyết bài toán

**Thuật toán A\***

**Thuật toán Dijkstra**

## Bài toán đường đi ngắn nhất giữa mọi cặp đỉnh

### Bài toán

*Cho đồ thị G với tập đỉnh V và tập các cạnh E (đồ thị có hướng hoặc vô hướng). Tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị.*

### Phân tích bài toán

**Input:**

Đồ thị *G = (V, E)*, trong đó *V = {1, 2, 3 …n}, E* là tập cạnh có trọng số

**Output:**

Ma trận *n\*n* chứa độ dài đường đi ngắn nhất *(i, j)* với *i, j V*

### Giải quyết bài toán

Bài toán đường đi ngắn nhất giữa mọi cặp đỉnh có thể áp dụng thuật toán Dijkstra hoặc thuật toán Bellman-Ford tùy vào trọng số của cạnh trong đồ thị.

Với ý tưởng giải bài toán đường đi ngắn nhất giữa mọi cặp đỉnh bằng phương pháp vét cạn, có thể áp dụng thuật Dijkstra hoặc thuật toán Bellman-Ford cho mỗi đỉnh của đồ thị (|*V*| lần)

* 1. **Thuật toán Dijkstra**
* Chỉ áp dụng với đồ thị không có trong số âm
* Độ phức tạp thời gian: *O(VE + V2log(V))*
* Đồ phức tạp không gian: *O(V2)*
  1. **Thuật toán Bellman-Ford**
* Độ phức tạp thời gian: *O(V2E)*
* Độ phức tạp không gian: *O(V2)*

Trong trường hợp đồ thị dày, độ phức tạp thời gian là *O(V4)*

* 1. **Phương pháp qui hoạch động – DP 1**

Nếu giải quyết bài toán đường đi ngắn nhất giữa mọi cặp đỉnh theo phương pháp vét cạn kết hợp thuật toán Bellman-Ford, độ phức tạp thời gian khá lớn, trong trường hợp xấu nhất *T = O(V4).*

Với ý tưởng giải bài toán bằng phương pháp quy hoạch động, quy trình xây dựng và thiết kế tương tự thuật toán Bellman-Ford.

* + 1. **Phân tích bài toán hướng phương pháp quy hoạch động**
  + **Hàm tối ưu *D(i,j)***: *D(i,j)(m)* là đường đi ngắn nhất từ đỉnh *i* đến đỉnh *j* đi qua tối đa *m* cạnh
  + **Công thức truy hồi:** *D(i,j)(m) =* *min k { D(i,k)(m-1) + a(k,j) }*

với *m = 1, 2, …, |V|-1*

* + **Cơ sở qui hoạch động:**

*0 nếu i = j*

*D(i,j)(0)=*

*nếu i j*

* + 1. **Lược đồ**

*Try(A)*

*initialize entries of D[1,2,…,V][1,2,…V] to +*

***for*** *each u V*

***for*** *each v V*

*D[u,v] = a(u,v)*

***endfor***

***for*** *m = 1 to n-1*

***for*** *i = 1 to n*

***for*** *j = 1 to n*

***for*** *k = 1 to n*

***if*** *D[i,j] > D[i,k] + a(k,j)*

*D[i,j] = D[i,k] + a(k,j)*

***endif;***

***endfor***

*Output D[u,v] as the shortest distance between u and v*

***End****.*

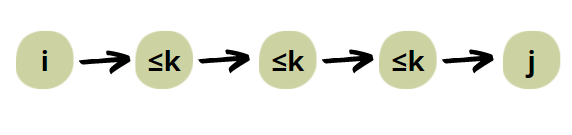
* + 1. **Đánh giá thuật toán**

Độ phức tạp thời gian của thuận toán qui hoạch động là *O(V2E)* gấp *V* lần thuật toán Bellman-Ford, trong trường hợp đồ thị dày, độ phức tạp thời gian có thể lên tới *O(V4)* bằng với độ phức tạp thời gian của thuật toán Bellman-Ford kết hợp với phương pháp vét cạn.

Độ phức tạp không gian là *O(V2)*

* 1. **Thuật toán Floyd-Warshall – DP 2**

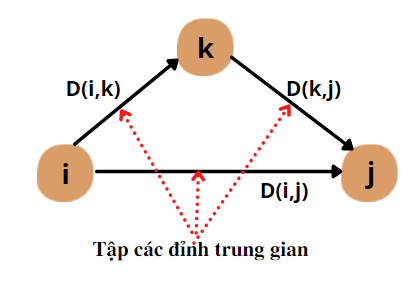
Floyd hay còn gọi là Floyd-Warshall là thuật toán để tìm đường đi ngắn nhất giữa mọi cặp đỉnh. Floyd hoạt động được trên đồ thị có hướng, có thể có trọng số âm, tuy nhiên không có chu trình âm. Ngoài ra, Floyd còn có thể được dùng để phát hiện chu trình âm. Thuật toán này cũng thuộc phương pháp qui hoạch động, tuy nhiên lại có hướng xây dựng khác, giảm độ phức tạp thời gian đáng kể so với thuật toán qui hoạch động thông thường.

* + 1. **Phân tích bài toán hướng phương pháp quy hoạch động**
* Ý tưởng: Xét tất cả các đỉnh là đỉnh trung gian. Duyệt mỗi đỉnh thuộc đồ thị và cập nhật đường đi ngắn nhất với đỉnh đó là đỉnh trung gian mà đường dẫn đi qua. Khi chọn đỉnh thứ *k* là đỉnh trung gian thì *{1,2,3..k-1}* cũng là đỉnh trung gian.
* *Bước 1 : Khởi tạo*
  + Độ dài hay chi phí đường đi từ đỉnh *i* đến đỉnh *j* là *a(i,j)*.
  + Cho *D(i,j)k* là chi phí đường đi ngắn nhất từ đỉnh *i* đến đỉnh *j* đi qua đỉnh trung gian thuộc tập *{1,2,…k}*

*Hình 3: Đường đi ngắn nhất từ đỉnh i tới đỉnh j qua các đỉnh trung gian*

* + *D(i,j)0* tương ứng là chi phí đường đi ngắn nhất từ đỉnh *i* đến đỉnh *j* qua 0 đỉnh trung gian, khi đó *D(i,j)0 = a(i,j)*
* *Bước 2 : Lặp qua từng đỉnh trong tập các đỉnh của đồ thị và đường chọn là đỉnh trung gian*
  + Với mỗi cặp *(i,j)* của đỉnh nguồn *i* và đỉnh đích *j* 
    - Nếu *k* không là đỉnh trung gian của đường đi ngắn nhất từ *i* đến *j* thì không thay đổi giá trị *D(i,j)*
    - Nếu *k* là đỉnh trung gian của đường đi ngắn nhất từ *i* đến *j*, cập nhật giá trị *D(i,j) = D(i,k) + D(k,j)*

Ta có:

* **Hàm tối ưu *D(i,j)*:** *D(i,j)n* là chi phí đường đi nhỏ nhất từ đỉnh *i*  đến đỉnh *j* đi qua tối đa *n-1* đỉnh trung gian.
* **Công thức truy hồi**: *D(i,j)k = mink {D(i,j)k-1, D(i,k)k-1+ D(k,j)k-1}*

**{1,2,…k-1}**

k-1

k-1

k-1

k-1

*Hình 4: Minh họa công thức truy hồi*

* **Cơ sở qui hoạch động**: *D(i,j)0 = a(i,j)*
  + 1. **Mô tả dữ liệu**
    - Đánh chỉ số đỉnh từ *1..n*
    - Biểu diễn đồ thị *G* bằng ma trận kề *M = (Cij)* cỡ *n\*n, Cij* = *Cost* nếu có cạnh nối đỉnh *i* với đỉnh *j* với chi phí *Cost* , *Cost* = + ∞ nếu không có đường đi
* Mảng *T[i][j]* lưu trữ các đỉnh mà đường đi ngắn nhất từ đỉnh *i* tới đỉnh *j* qua chúng.
  + 1. **Lược đồ**

*Try(A)*

*initialize entries of T[1,2,…,V][1,2,…V] to NULL*

***for*** *each u V*

***for*** *each v V*

*D[u,v] = a(u,v)*

*T[u,v] = 0*

***endfor***

***for*** *k = 1 to n*

***for*** *i = 1 to n*

***for*** *j = 1 to n*

***if*** *D[i,j] > D[i,k] + D[k,j]*

*D[i,j] = D[i,k] + D[k,j]*

*T[i,j] = k*

***endif;***

***endfor***

*Output D[i,j] as the shortest distance between i and j*

***End****.*

Truy vết tìm đường đi:

*FindPath(T,u,v)*

*i = T[u,v]*

***if*** *i == 0*

*print u*

***else***

*FindPath(T,i,v)*

*FindPath(T,u,i)*

***endif***

***End.***

* + 1. **Đánh giá thuật toán**

Độ phức tạp thuật toán Floyd-Warshall là *O(V3)*, trong trường hợp này, thời gian giải quyết bài toán con chỉ còn *O(1)*.

Độ phức tạp không gian của thuật toán là *O(V2)*

* + 1. **Cải tiến thuật toán**

Mặc dù sau rất nhiều năm nghiên cứu, cho đến nay vẫn chưa có thuật toán nào tốt hơn thời gian *O(V3)* một cách đáng kể và cho mọi trường hợp. Do đó, người ta giả thuyết rằng không tồn tại thuật toán như vậy (gọi là giả thuyết đường đi ngắn nhất giữa mọi cặp đỉnh). Thuật toán (ngẫu nhiên) tốt nhất cho đến nay được phát triển bởi [Ryan Williams](http://web.stanford.edu/~rrwill/) có thời gian *O(V3/)*

* 1. **Thuật toán Johnson**

Như đã biết thuật toán Floyd-Warshall tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong thời gian *O(V3)*. Thuật toán này áp dụng được cho cả trường hợp đồ thị có trọng số âm và không có chu trình âm. Cho đến nay vẫn chưa có thuật toán nào thực sự nhanh hơn *O(V3)* một cách đáng kể.

Với đồ thị không có trọng số âm, áp dụng thuật toán Dijkstra (với Fibonacci Heap) *V* lần, sẽ thu được thuật toán *O(VE+V2logV)*. Với đồ thị thưa, thuật toán này rõ ràng nhanh hơn *O(V3)* một cách đáng kể. Ở đây mình nhấn mạnh đồ thị không có trọng số âm thì ta mới áp dụng được Dijkstra. Câu hỏi đặt ra : Có tồn tại hay không thuật toán *O(VE+V2logV)* tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị có hướng, có trọng số âm và không có chu trình âm.

Năm 1977, Jonhson đã đưa ra câu trả lời cho vất đề này

* + 1. **Phân tích bài toán**
* Ý tưởng: Thay đổi trọng số của các cung của đồ thị *G* để thu được đồ thị *H* sao cho:
  + Trọng số cung của đồ thị *H* không âm; do đó ta có thể áp dụng Dijkstra cho *H* để tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong H
  + Đường đi ngắn nhất giữa hai đỉnh bất kì của *H* cũng chính là đường đi ngắn nhất giữa hai đỉnh tương ứng của đồ thị *G*.

Các bước thực hiện:

* *Bước 1 :* Thêm đỉnh *s* vào đồ thị *G* và với mỗi đỉnh *x ϵ V*, thêm cung *(s,x)*  có trọng số *a(s,x) = 0.*
* *Bước 2* : Áp dụng thuật toán Bellman-Ford để tìm khoảng cách ngắn nhất *(s,x)* từ đỉnh *s* tới mọi đỉnh *x ϵ V* và sử dụng khoảng cách đó làm thế năng của đỉnh *x*: *h(x) = (s,x)*
* *Bước 3* : Gán mỗi cung *(u,v)* *ϵ E* một trọng số mới tương ứng là trọng số mới trong đồ thị H.

*ah(u,v) = a(u,v) + h(u) - h(v)*, *ah(u,v) 0*

* *Bước 4* : Áp dụng thuật toán Dijkstra *V* lần cho đồ thị *H* để tím đường đi ngắn nhất giữa mọi cặp đỉnh trong *H*. Gọi *H(x,y)*, *(x,y)* lần lượt là khoảng cách ngắn nhất từ đỉnh *x* đến đỉnh *y* trong đồ thị *H* và đồ thị *G.*

*(x,y) = H(x,y) + h(y) - h(x)*

* + 1. **Lược đồ**

*Johnson(G,a)*

*Add s to V*

***for*** *each x ϵ V*

*add (s,x) to E*

*a(s,x) = 0*

*h[1,2,…,V] = BellmanFord(G,s,a)*

***for*** *each (u,v) ϵ E*

*aH (u,v) = a(u,v) + h[u] – h[v]*

***for*** *each x ϵ V*

*d[x][1,2,…V] = Dijkstra(G,x,aH)*

***for*** *each x V*

***for*** *y ϵ V*

*(x,y) = d[x,y] + h[y] – h[x]*

***endfor***

***End.***

* + 1. **Đánh giá thuật toán**

Độ phức tạp thời gian của thuật toán Johnson là *O(VE + V2logV)* trong đó: Tại bước 1, mất thời gian *O(V)*; Bước 2 thời gian tương ứng với Bellman-Ford là *O(VE)*; Bước 3 mất thời gian *O(E)*; Tại bước cuối cùng áp dụng *V* lần thuật toán Dijkstra cho đồ thị *H* mất *O(VE + V2logV).*

1. **So sánh các thuật toán giải quyết bài toán đường đi ngắn nhất cho mọi cặp đỉnh.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Tính chất đồ thị** | | **Độ phức tạp thời gian** | | **Độ phức tạp không gian** | **Hướng tiếp cận** |
| **Đồ thị vô hướng** | **Đồ thị có hướng** | **Time** | ***E = (V2)*** |
| **Dijkstra** | Trọng số dương | Trọng số dương | *O(VE+V2logV )* | *O(V3)* | *O(V2)* | Phương pháp vét cạn và tham lam |
| **Bellman-Ford** | Trọng số dương | Có trọng số, có thể có chu trình âm | *O(V2E)* | *O(V4)* | *O(V2)* | Phương pháp vét cạn và qui hoạch động |
| **Dynamic programming – DP 1** | Trọng số dương | Có trọng số, có thể có chu trình âm | *O(V2E)* | *O(V4)* | *O(V2)* | Phương pháp qui hoạch động |
| **Floyd - Warshall** | Trọng số dương | Có trọng số, không có chu trình âm | *O(V3)* | *O(V3)* | *O(V2)* | Phương pháp qui hoạch động |
| **Johnson** | Trọng số dương | Có trọng số, không có chu trình âm | *O(VE+V2logV )* | *O(V3)* | *O(V2)* | Kết hợp Dijkstra và BellmanFord |

# PHẦN 3: KẾT LUẬN

Bài toán đường đi ngắn nhất là bài toán khá nổi tiếng và có nhiều biến thể cũng như cách giải quyết từng biến thể đó. Nó có những ý nghĩa thực tế quan trọng trong cuộc sống và nghiên cứu. Nghiên cứu đã đi khái quát các vấn đề liên quan đến bài toán như lý thuyết về đồ thị và các thuật toán giải quyết bài toán.

Điểm lại các thuật toán quan trọng:

* Thuật toán Dijkstra giải quyết bài toán đường đi ngắn nhất nguồn đơn
* Thuật toán Bellman-Ford giải quyết đường đi ngắn nhất nguồn đơn có trọng số âm và chu trình âm.
* Thuật toán Floyd – Warshall giải quyết bài toán đường đi ngắn nhất giữa mọi cặp đỉnh
* Thuật toán Johnson giải quyết bài toán đường đi ngắn nhất giữa mọi cặp đỉnh, nhanh hơn Floyd – Warshall với đồ thị thưa
* Thuật toán tìm kiếm A\*  giải quyết cho đường dẫn ngắn nhất một cặp đỉnh bằng cách sử dụng heuristics để cố gắng tăng tốc tìm kiếm.

**TÀI LIỆU THAM KHẢO**

1. Lec 17: Shortest Paths I: Properties, Dijkstra's Algorithm, Breadth-first Search | MIT 6.046J / 18.410J Introduction to Algorithms (SMA 5503), Fall 2005 by Prof. Charles E. Leiserson and Prof. Erik Demaine

Link: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-17-shortest-paths-i-properties-dijkstras-algorithm-breadth-first-search/>

1. Lec 18: Shortest Paths II: Bellman-Ford, Linear Programming, Difference Constraints | MIT 6.046J / 18.410J Introduction to Algorithms (SMA 5503), Fall 2005 by Prof. Charles E. Leiserson and Prof. Erik Demaine

Link: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-18-shortest-paths-ii-bellman-ford-linear-programming-difference-constraints/>

1. Lec 19: Shortest Paths III: All-pairs Shortest Paths, Matrix Multiplication, Floyd-Warshall, Johnson | MIT 6.046J / 18.410J Introduction to Algorithms (SMA 5503), Fall 2005 by Prof. Charles E. Leiserson and Prof. Erik Demaine

Link: <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-19-shortest-paths-iii-all-pairs-shortest-paths-matrix-multiplication-floyd-warshall-johnson/>

1. Giáo trình Toán rời rạc – Nguyễn Đức Nghĩa – Nguyễn Tô Thành
2. Website: <https://www.geeksforgeeks.org/>