**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

A picture containing game

Description automatically generated-----🙞🙜🕮🙞🙜-----

**UIT**

**BÁO CÁO ĐỒ ÁN CUỐI KÌ**

**GIẢI TOÁN LÝ THUYẾT ĐỒ THỊ**

**Môn học: Trí Tuệ Nhân Tạo**

**Giảng viên: Nguyễn Đình Hiển**

**Sinh viên:**

**18520507 – Nguyễn Phước Bình**

**18520924 – Nguyễn Duy Khoa**

**18520898 – Hoàng Đức Khánh**

**Lời cảm ơn**

Lời đầu tiên nhóm xin chân thành cảm ơn thầy Nguyễn Đình Hiển - khoa Khoa Học Máy Tính – trường Đại học Công nghệ Thông tin, đã truyền đạt kiến thức bổ ích, hỗ trợ và giải đáp thắc mắc cho nhóm trong suốt quá trình học tập và thực hiện đồ án môn Trí Tuệ Nhân Tạo. Cùng với sự giảng dạy nhiệt tình của thầy và niềm đam mê học hỏi của các bạn sinh viên trong nhóm, chúng em đã ứng dụng những kiến thức đã học trong suốt học kỳ vừa qua để thực hiện đồ án “Giải toán Lý thuyết đồ thị”, đưa những kiến thức đã học vào thực tiễn.

Tuy đã hết sức cố gắng, song với quỹ thời gian và kiến thức hạn chế chắc chắn đồ án vẫn còn những vấn đề khiếm khuyết, chúng em rất mong nhận được sự đóng góp ý kiến quý báu của thầy và các bạn sinh viên để đồ án được hoàn thiện hơn.

*Mọi góp ý xin liên hệ:*

Nguyễn Phước Bình – 18520507@gm.uit.edu.vn

Nguyễn Duy Khoa – 18520924@gm.uit.edu.vn

Hoàng Đức Khánh – 18520898@gm.uit.edu.vn

**Xin chân thành cảm ơn!**

**Mở đầu**

Toán rời rạc là một lĩnh vực của toán học nghiên cứu về các đối tượng rời rạc. Mặc dù các đối tượng là rời rạc, không có ý nghĩa nhưng khi chúng ta liên kết các đối tượng rời rạc lại với nhau ta lại có được những thông tin rất lý thú và mang nhiều ý nghĩa. Chúng ta sẽ sử dụng công cụ của toán học rời rạc khi phải đếm các đối tượng, nghiên cứu mối quan hệ giữa các tập rời rạc, khi nghiên cứu các quá trình hữu hạn. Một trong những nguyên nhân chủ yếu làm tăng tầm quan trọng của toán rời rạc là việc lưu trữ và xử lý thông tin trên máy tính điện tử mà bản chất là các quá trình rời rạc. Ba lĩnh vực có nhiều ứng dụng của toán học rời rạc là lý thuyết tổ hợp, hàm đại số logic (đại số Boole) và lý thuyết đồ thị. Trong phạm vi đồ án này chúng em chỉ trình bày lĩnh vực có thể xem là quan trọng nhất và có nhiều ứng dụng nhất của toán học rời rạc là **Lý thuyết đồ thị**.

Nội dung đồ án này gồm các nội dung cơ bản nhất của lý thuyết đồ thị và được chia làm 3 chương:

**Chương 1:** Trình bày các thuật ngữ, định nghĩa và khái niệm cơ bản của đồ thị như đồ thị vô hướng, có hướng, đường đi, chu trình, tính liên thông….

**Chương 2:** Trình bày về thuật toán tìm đường đi ngắn nhất Dijkstra

**Chương 3:** Trình bày về thuật toán tìm cây khung nhỏ nhất Kruskal, Prim.

Qua các nội dung trên sẽ giúp ta hiểu rõ hơn về các thuật toán và ứng dụng vào đời sống thực tiễn.

Thành phố Hồ Chí Minh, ngày 02 tháng 07 năm 2020.

Mục lục

**Chương I. Đại cương về đồ thị**

**I. Các khái niệm cơ bản**

**1. Đồ thị**

**2. Biểu diễn đồ thị**

**II. Đồ thị có hướng (Directed Graph)**

**III. Tính liên thông của đồ thị**

**1. Đường đi**

**2. Chu trình**

**3. Tính liên thông trong đồ thị vô hướng**

**4. Tính liên thông trong đồ thị có hướng**

**4.1. Liên thông mạnh (Strongly Connected)**

**4.2. Liên thông yếu (Weakly Connected)**

**Chương II. Bài toán đường đi ngắn nhất**

**I. Mở đầu**

**II. Thuật toán Dijkstra tìm đường đi ngắn nhất**

**Chương III. Cây khung nhỏ nhất**

**I. Cây (Tree)**

**1. Định nghĩa**

**2. Các ví dụ**

**II. Cây khung**

**1. Định nghĩa**

**2. Ví dụ**

**3. Định lý**

**4. Cây khung bé nhất**

**4.1. Định nghĩa**

**4.2. Thuật toán Kruskal**

**4.3. Thuật toán Prim**

**Kết quả thử nghiệm**

**Kết Luận**

|  |  |
| --- | --- |
| **BẢNG PHÂN CÔNG** | |
| **Nguyễn Phước Bình** | Trình bày báo cáo khoa học, hỗ trợ xử lí giao diện, code thuật toán Prim, Dijkstra, làm github. |
| **Nguyễn Duy Khoa** | Làm giao diện chương trình, code thuật toán Kruskal, trình bày báo cáo kỹ thuật. |
| **Hoàng Đức Khánh** | Làm powerpoint báo cáo. |

**Chương I. Đại cương về đồ thị**

**I. Các khái niệm cơ bản**

**1. Đồ thị**

Đồ thị (graph) G = (V,E) là một bộ gồm 2 tập hợp V và E, trong đó V ≠ ∅ các phần tử của V được gọi là các đỉnh (vertices), các phần tử của E được gọi là các cạnh (edges), mỗi cạnh tương ứng với 2 đỉnh.

Nếu cạnh e tương ứng với 2 đỉnh v, w thì ta nói v và w là 2 đỉnh kề (hay 2 đỉnh liên kết) (adjacent) với nhau. Ta cũng nói cạnh e tới hay liên thuộc (incident) với các đỉnh v và w. Ký hiệu e = (hoặc e = vw; e = wv). Cạnh tương ứng với 2 đỉnh trùng nhau gọi là một vòng hay khuyên (loop) tại v.

Hai cạnh phân biệt cùng tương ứng với một cặp đỉnh được gọi là 2 cạnh song song (paralleledges) hay cạnh bội. Đồ thị không có cạnh song song và cũng không có vòng được gọi là đơn đồ thị (simple graph). Ngược lại là đa đồ thị (multigraph).

Đồ thị mà mọi cặp đỉnh của nó đều kề nhau được gọi là đồ thị đầy đủ (Complete graph). Đơn đồ thị đầy đủ bao gồm n đỉnh được ký hiệu: Kn.

Đồ thị G’ = (V’,E’) được gọi là một đồ thị con (subgraph) của đồ thị G = (V,E) nếu V’ ⊂ V; E’ ⊂ E.

Đồ thị có số đỉnh và số cạnh hữu hạn được gọi là đồ thị hữu hạn (finite graph), ngược lại được gọi là đồ thị vô hạn (Infinite graph).

Trong đồ án này, chúng ta chỉ khảo sát các đồ thị hữu hạn.

**2. Biểu diễn đồ thị**

Một đồ thị có thể được biểu diễn bằng hình học, một ma trận hay một bảng.

a) **Biểu diễn hình học**

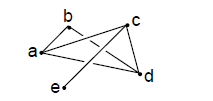
Người ta thường biểu diễn hình học của đồ thị như sau:

- Biểu diễn mỗi đỉnh của đồ thị bằng một điểm (vòng tròn nhỏ, ô vuông nhỏ).

- Một cạnh được biểu diễn bởi một đường (cong hay thẳng) nối 2 đỉnh liên thuộc với cạnh đó.

**Ví dụ:** Đồ thị G có: V = {a, b, c, d, e}

E = {ab, ac, ad, bd, cd, ce}

 Được biểu diễn hình học như sau:

**b) Biểu diễn đồ thị bằng ma trận**

Người ta có thể biểu diễn đồ thị bằng ma trận. Có 2 kiểu ma trận thường được dùng để biểu diễn đồ thị:

- Ma trận liên kết hay liền kề (adjacency matrix).

- Ma trận liên thuộc (incidence matrix).

* **Ma trận liền kề**

Cho G = (V,E) có n đỉnh v1, v2, …, vn. Ma trận liền kề của G tương ứng với thứ tự các đỉnh v1, v2, …, vn là một ma trận vuông cấp n.

A = (aij)n trong đó:

aij = 1 nếu vivj là một cạnh của G.

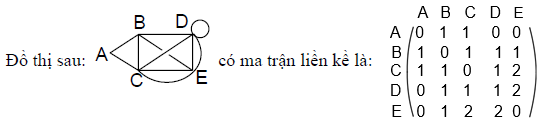
0 nếu vivj không là một cạnh của G.

**Chú ý:**

- Ma trận liền kề của một đồ thị khác nhau tùy thuộc vào thứ tự liệt kê các đỉnh. Do đó, có tới n! ma trận liền kề khác nhau của một đồ thị n đỉnh vì có n! cách sắp xếp n đỉnh.

- Ma trận liền kề của một đồ thị là một ma trận đối xứng vì nếu vi được nối với vj thì vj cũng được nối vi và ngược lại nếu vi không nối với vj thì vj cũng không nối với vi.

- Một vòng được tính là một cạnh từ đỉnh v vào v.



* **Ma trận liên thuộc**

Người ta còn dùng ma trận liên thuộc để biểu diễn đồ thị. Cho G = (V,E) là một đồ thị với v1, v2, …, vn là các đỉnh và e1, e2, …, em là các cạnh của G. Khi đó ma trận liên thuộc của G theo thứ tự trên của V và E là một ma trận M = (mij)n x m với:

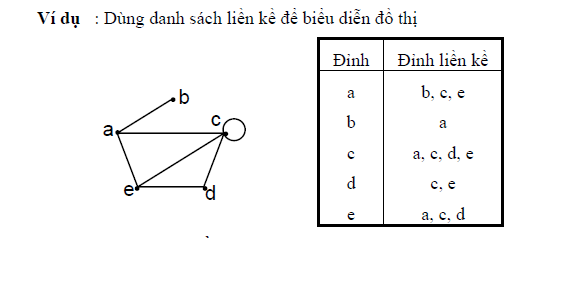
mij= 1 nếu cạnh ej nối với đỉnh vi.

0 nếu cạnh ej không nối với đỉnh vi.

**Chú ý**: Các ma trận liên thuộc cũng có thể được dùng để biểu diễn các cạnh bội và khuyên (vòng). Các cạnh bội (song song) được biểu diễn trong ma trận liên thuộc bằng cách dùng các cột có các phần tử giống hệt nhau vì các cạnh này được nối với cùng một cặp các đỉnh. Các vòng được biểu diễn bằng cách dùng một cột với đúng một phần tử bằng 1 tương ứng với đỉnh nối với khuyên đó.

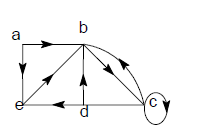
**c) Biểu diễn đồ thị bằng bảng**

Người ta có thể biểu diễn đồ thị không có cạnh bội bằng bảng hay còn gọi là danh sách liền kề. Danh sách này chỉ rõ các đỉnh nối với mỗi đỉnh của đồ thị.



**II. Đồ thị có hướng (Directed Graph)**

**Định nghĩa:** Một đồ thị có hướng G = (V,E) gồm tập hợp các đỉnh V và tập hợp các cạnh E bao gồm các cặp sắp thứ tự các phần tử của V. Cạnh e tương ứng với một cặp thứ tự (a,b) của 2 đỉnh a, b ∈ V được gọi là một cạnh có hướng từ a đến b. Ký hiệu: e = được gọi là đỉnh đầu, b được gọi là đỉnh cuối của cạnh e. Đỉnh đầu và đỉnh cuối của khuyên (vòng) là trùng nhau.

** Ví dụ:**

**III. Tính liên thông của đồ thị**

**1. Đường đi**

**Định nghĩa:** Đường đi (path) có độ dài n từ vo đến vn với n là một số nguyên dương, trong một đồ thị vô hướng là một dãy các cạnh liên tiếp vov1, v1v2, … , vn−1vn. Đỉnh vo được gọi là đỉnh đầu, đỉnh vn được gọi là đỉnh cuối. Đường đi này thường được viết gọn: vov1v2 … vn−1vn. Khi chỉ cần nêu ra đỉnh đầu vo và đỉnh cuối vn của đường đi, ta viết: đường đi vo – vn.

* Một đường đi không qua cạnh nào lần thứ hai được gọi là ***đường đi đơn giản (đường đi đơn)*.**
* Một đường đi không qua đỉnh nào lần thứ hai được gọi là ***đường đi sơ cấp*.**
* **Lưu ý:** Một đường đi sơ cấp là một đường đi đơn giản nhưng một đường đi đơn giản có thể không là đường đi sơ cấp).

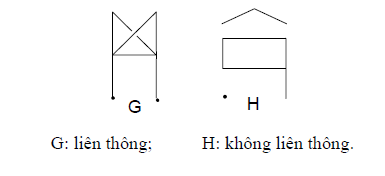
**2. Chu trình**

**Định nghĩa:** Một đường đi khép kín (đỉnh đầu ≡ đỉnh cuối) và có số lượng đỉnh n ≥ 2 được gọi là một chu trình (Cycle).

* Chu trình không đi qua cạnh nào lần thứ hai được gọi là ***chu trình đơn giản***.
* Chu trình không đi qua đỉnh nào lần thứ hai, trừ đỉnh đầu ≡ đỉnh cuối, được gọi là một ***chu trình sơ cấp*.**

**3. Tính liên thông trong đồ thị vô hướng**

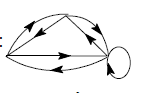
**Định nghĩa**: Một đồ thị vô hướng được gọi là liên thông nếu có đường đi giữa mọi cặp đỉnh phân biệt của đồ thị.

** Ví dụ:**

**4. Tính liên thông trong đồ thị có hướng**

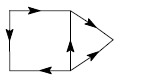
**4.1. Liên thông mạnh (Strongly connected)**

Đồ thị có hướng G được gọi là liên thông mạnh nếu có đường đi từ a đến b và từ b đến a; ∀ a, b ∈ đồ thị.

** Ví dụ:**

**4.2. Liên thông yếu (Weakly connected)**

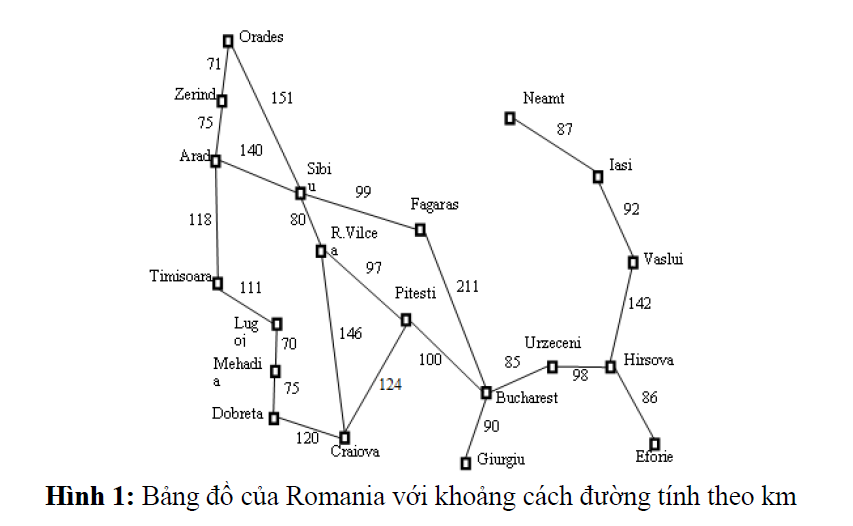
Đồ thị có hướng G được gọi là liên thông yếu nếu đồ thị vô hướng tương ứng của nó là liên thông.

** Ví dụ:**

**Chương II. Bài toán đường đi ngắn nhất**

**I. Mở đầu**

Trong thực tế, nhiều bài toán có thể mô hình bằng đồ thị có trọng số. Đó là đồ thị mà mỗi cạnh của nó được gán một con số (nguyên hoặc thực) gọi là trọng số ứng với cạnh đó. Ví dụ ta cần mô hình một hệ thống đường hàng không. Mỗi thành phố được biểu diễn bằng một đỉnh, mỗi chuyến bay là một cạnh nối 2 đỉnh tương ứng. Nếu trong bài toán đang xét ta cần tính đến khoảng cách giữa các thành phố thì ta cần gán cho mỗi cạnh của đồ thị cơ sở trên khoảng cách giữa các thành phố tương ứng. Nếu ta quan tâm đến thời gian của mỗi chuyến bay thì ta sẽ gán các thời lượng này cho mỗi cạnh của đồ thị cơ sở…



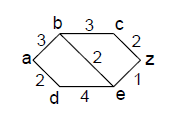
Bài toán đặt ra là *tìm đường đi ngắn nhất từ thành phố này đến thành phố khác*. Hay nói theo ngôn ngữ của lý thuyết đồ thị: *ta cần tìm đường đi có tổng trọng số (ngắn) nhỏ nhất từ đỉnh này đến một đỉnh khác của đồ thị.*

**II.** **Thuật toán Dijkstra tìm đường đi ngắn nhất**

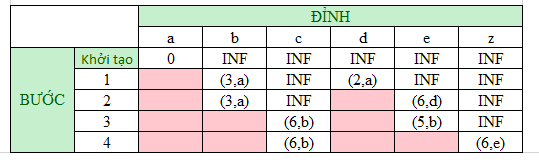
Có một số thuật toán tìm đường đi ngắn nhất giữa 2 đỉnh trên một đồ thị có trọng số. Ở đây ta sẽ sử dụng thuật toán Dijkstra, do nhà toán học người Hà Lan: E.Dijkstra đề xuất năm 1959.

Chúng ta sẽ áp dụng thuật toán Dijkstra đối với đồ thị vô hướng. Đối với đồ thị có hướng, ta chỉ cần thay đổi một chút.

Trước khi giới thiệu thuật toán, ta xét ví dụ:

Tính độ dài của đường đi ngắn nhất giữa 2 đỉnh **a** và **z** trong đồ thị có trọng số sau:

Đối với đồ thị này, ta dễ dàng tìm đượ c đường đi ngắn nhất từ **a** đến **z** bằng cách thử trực tiếp. Tuy nhiên, ta sẽ phát triển một số ý tưởng giúp ta hiểu thuật toán Dijkstra dễ dàng hơn. Ta sẽ tìm độ dài của đường đi ngắn nhất từ **a** đến các đỉnh kế tiếp cho đến khi đạt tới đỉnh **z**. Xuất phát từ đỉnh **a**, ta thấy chỉ có 2 đỉnh **b** và **d** liên thuộc với **a**. Nên chỉ có hai đường đi xuất phát từ **a** đến **b** và **d** là **ab** và **ad** với độ dài tương ứng là 3 và 2. Do đó, **d** là đỉnh gần **a** nhất.

 Bây giờ , ta tìm đỉnh tiếp theo gần **a** nhất trong tất cả các đường đi qua **a** và **d**. Đường đi ngắn nhất từ **a** tới **b** là **ab** với độ dài 3. Đường đi ngắn nhất từ **a** đến **e** là **a, b, e** với độ dài 5. Đường đi ngắn nhất từ **a** đến **c** là **a, b, c** với độ dài 6. Khi đó ta có 2 đường đi từ **a** đến **z** qua **c** và **e** là **a, b, c, z** với độ dài 8; **a, b, e, z** với độ dài 6. Vậy, đường đi ngắn nhất từ **a** đến **z** là: **a, b, e, z** với độ dài 6.

***Minh họa từng bước cho ví dụ trên***

Ví dụ trên đã minh họa những nguyên tắc chung dùng trong thuật toán Dijkstra. Đường đi ngắn nhất từ đỉnh **a** đến **z** có thể tìm được bằng cách thử trực tiếp. Nhưng phương pháp này không áp dụng được đối với đồ thị có nhiều cạnh.

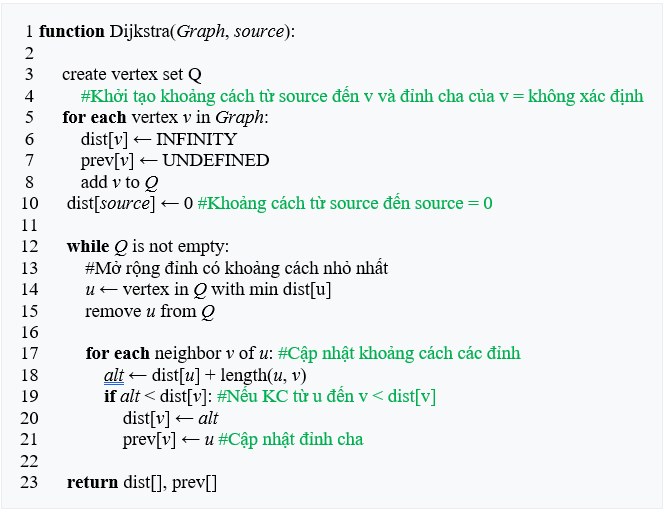
Bây giờ, ta nghiên cứu bài toán tìm độ dài của đường đi ngắn nhất giữa **a** và **z** trong đơn đồ thị liên thông, vô hướng và có trọng số.

Thuật toán Dijkstra được thực hiện bằng cách tìm độ dài của đường đi ngắn nhất từ **a** đến đỉnh đầu tiên, từ **a** đến đỉnh thứ hai,… cho đến khi tìm được độ dài ngắn nhất từ **a** đến **z**.

Thuật toán này dựa trên một dãy các bước lặp. Một tập đặc biệt các đỉnh được xây dựng bằng cách cộng thêm một đỉnh trong mỗi bước lặp. Thủ tục gán nhãn được thực hiện trong mỗi lần lặp đó.

Trong thủ tục gán nhãn này, đỉnh **w** được gán nhãn bằng độ dài đường đi ngắn nhất từ **a** đến **w** và chỉ đi qua các đỉnh thuộc tập đặc biệt. Một đỉnh được thêm vào tập này là đỉnh có nhãn nhỏ nhất so với các đỉnh chưa có trong tập đó.

**\* Mã giả thuật toán Dijkstra:**

****

Nguồn: Wikipedia.org

\***Ưu điểm:**

**+** Là thuật toán tìm kiếm mù. Điều này có nghĩa là không cần biết nút đích trước. Vì lý do này, nó tối ưu trong trường hợp ta không có bất kỳ kiến ​​thức nào về biểu đồ khi không thể ước tính khoảng cách giữa mỗi nút và mục tiêu.

+ Vì Dijkstra chọn các cạnh với chi phí nhỏ nhất ở mỗi bước nên nó thường chiếm một diện tích lớn của biểu đồ. Điều này đặc biệt hữu ích khi ta có nhiều nút mục tiêu nhưng không biết nút nào gần nhất.

\***Nhược điểm:**

**+** Thất bại với đồ thị có trọng số âm.

+ Vì là chiến lược tìm kiếm mù nên tốn rất nhiều thời gian trong khi xử lý.

**Chương III. Cây khung nhỏ nhất**

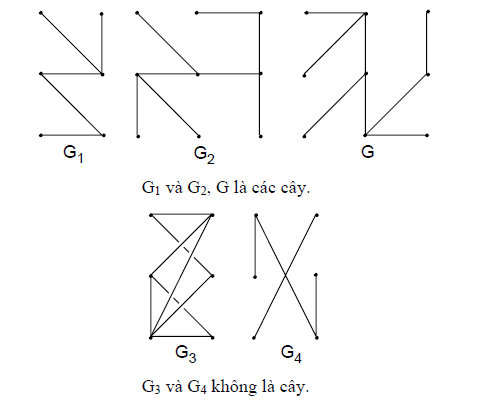
**I. Cây (Tree)**

**1. Định nghĩa**

Cây là một đồ thị vô hướng, liên thông và không có chu trình sơ cấp.

Do cây không có chu trình sơ cấp, nên cây không thể có cạnh bội và khuyên. Vậy mọi cây là đơn đồ thị.

**2. Các ví dụ**

****

+ G3 có chứa chu trình nên G3 không là cây.

+ G4 không liên thông nên G4 không là cây.

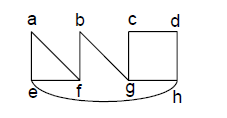
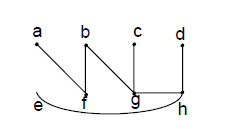
**II. Cây khung**

**1. Định nghĩa**

Cho G là một đơn đồ thị. Một cây được gọi là cây khung của G nếu nó là một đồ thị con của G và chứa tất cả các đỉnh của G.

**2. Ví dụ**

Cho đơn đồ thị G sau:

 Một cây khung tạo ra từ G bằng cách xóa đi các cạnh tạo ra chu trình đơn: ae, ef và cd là:

**3. Định lý**

Một đơn đồ thị là liên thông khi và chỉ khi nó có cây khung.

**4. Cây khung bé nhất**

**4.1. Định nghĩa**

Cây khung nhỏ nhất trong một đồ thị liên thông, có trọng số là một cây khung có tổng trọng số trên các cạnh của nó là nhỏ nhất.

**4.2. Thuật toán Kruskal**

Thuật toán này được xuất bản lần đầu tiên vào năm 1956, bởi Joseph Kruskal. Thuật toán Kruskal được dùng để tìm cây khung nhỏ nhất trong một đồ thị liên thông có trọng số.

Để thực hiện thuật toán Kruskal, ta xuất phát từ việc chọn cạnh có trọng số nhỏ nhất của đồ thị. Sau đó lần lượt ghép thêm vào các cạnh có trọng số tối thiểu và không tạo thành chu trình với các cạnh đã được chọn. Thuật toán dừng lại khi (n-1) cạnh được chọn.

Thuật toán Kruskal dưới dạng giả mã:

***Procedure Kruskal (G: đồ thị n đỉnh, liên thông có trọng số).***

*T: = đồ thị rỗng*

*for i: = 1 to n* – *1*

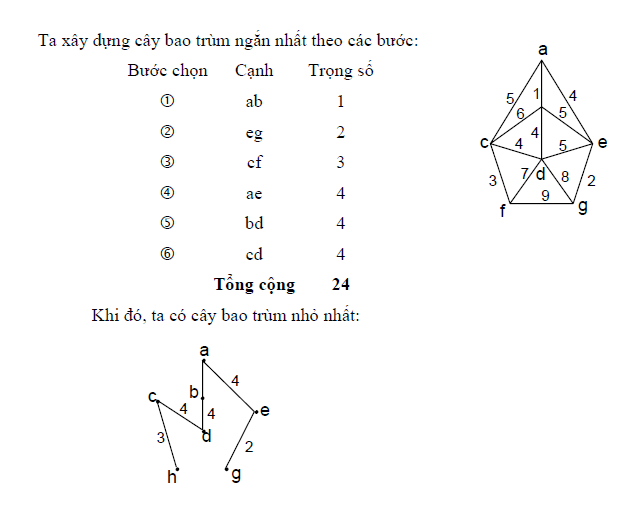
*begin*

*e: = một cạnh bất kỳ của G với trọng số nhỏ nhất và không tạo ra*

*chu trình trong T, khi ghép nó vào T.*

*T: = T với cạnh e đã được ghép vào.*

*end {T là cây khung nhỏ nhất}.*

**Ví dụ**: Dùng thuật toán Kruskal, tìm cây bao trùm ngắn nhất của đồ thị sau:

**4.3. Thuật toán Prim**

Thuật toán Prim được giới thiệu lần đầu tiên vào năm 1957 và một lần nữa độc lập bởi Edsger Dijkstra năm 1959. Do đó nó còn được gọi là thuật toán **DJP**, thuật toán **Jarník**, hay thuật toán **Prim–Jarník**.

Thuật toán Prim bắt đầu bằng việc chọn một đỉnh bất kỳ và lấy ra cạnh có trọng số nhỏ nhất nối với đỉnh này đặt nó vào cây khung. Sau đó, lần lượt ghép vào cây các cạnh có trọng số nhỏ nhất liên thuộc với một đỉnh của cây và không tạo ra chu trình trong cây. Thuật toán dừng lại khi (n – 1) cạnh được ghép vào cây.

* **Chú ý:** Có nhiều hơn một cây khung nhỏ nhất ứng với một đồ thị liên thông và có trọng số.

Thuật toán Prim dưới dạng giả mã:

***Procedure Prim (G: đồ thị liên thông có trọng số với n đỉnh)****.*

*T: = cạnh có trọng số nhỏ nhất nối với đỉnh đã chọn.*

*for i = 1 to n* – *2. # vì đã có 1 cạnh được thêm vào nên ta duyệt đến n-2 cạnh*

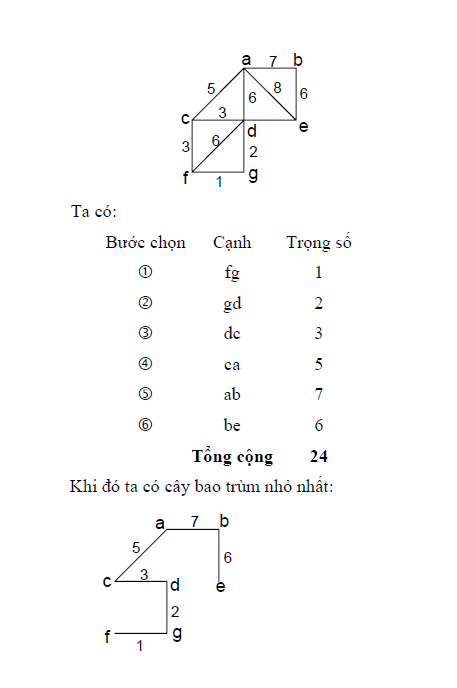
*begin*

*e:= cạnh có trọng số tối thiểu liên thuộc với một đỉnh trong T và*

*không tạo ra chu trình trong T nếu ghép nó vào T.*

*T: = T với e được ghép vào*

*end {T là cây khung nhỏ nhất của G}.*

 **Ví dụ**: Dùng thuật toán Prim để tìm cây khung nhỏ nhất trong đơn đồ thị có trọng số sau (bắt đầu từ **f** ):

* ***Sự khác nhau giữa thuật toán Prim và thuật toán Kruskal.***

- Trong thuật toán Prim, ta chọn các cạnh có trọng số nhỏ nhất, liên thông với các đỉnh đã thuộc cây và không tạo ra chu trình.

- Trong thuật toán Kruskal, ta chọn các cạnh có trọng số tối thiểu mà không nhất thiết phải liên thuộc với các đỉnh của cây và không tạo ra chu trình.

\* **Nhận xét:**

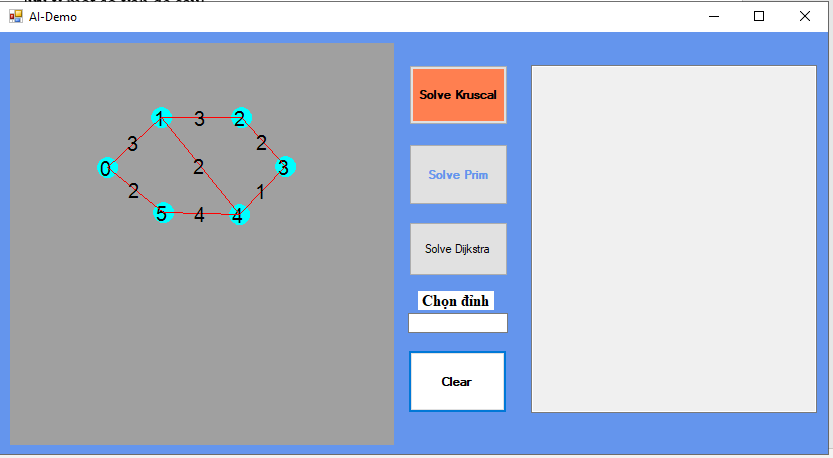
**+** Thuật toán Kruskal làm việc kém hiệu quả với những đồ thị dày (nhiều cạnh) vì phải sắp xếp tất cả các cạnh trong đồ thị theo trọng số tăng dần. Tuy nhiên, để xây dựng cây khung nhỏ nhất với n-1 cạnh, nói chung ta không cần phải sắp thứ tự toàn bộ các cạnh mà chỉ cần xét phần trên của dãy đó chứa r < m cạnh.

+ Do ra đời sau và được cải tiến hơn, thuật toán Prim tỏ ra hiệu quả hơn trong việc lựa chọn cạnh để đưa vào cây khung do chỉ xét các cạnh liên thuộc với đỉnh của cây. Thuật toán Prim còn được gọi là phương pháp lân cận gần nhất.

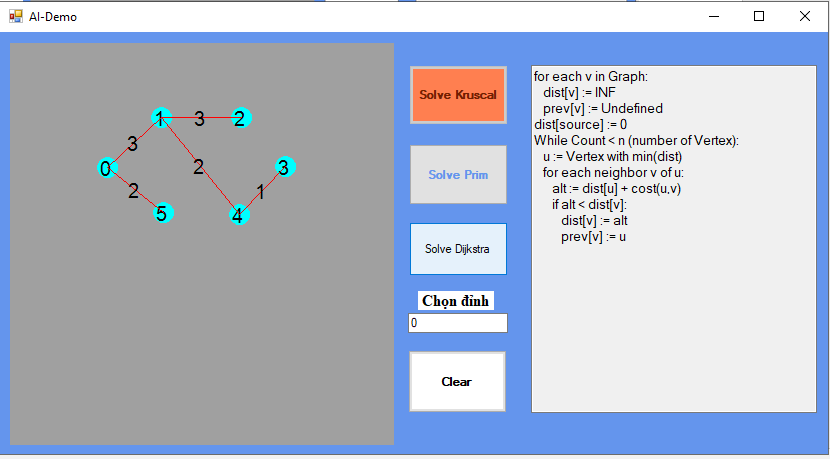
**Kết quả thử nghiệm**

**\* Thuật toán Dijkstra**

**+** Cho đồ thị :

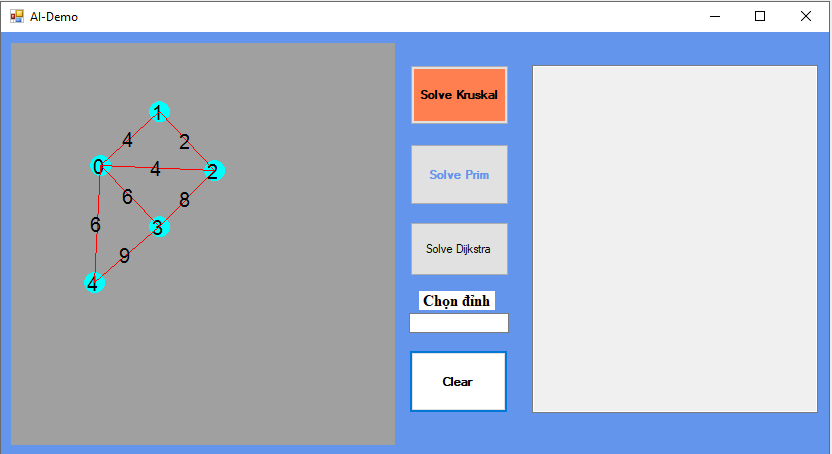


+ Kết quả chương trình với đỉnh bắt đầu là **0** :

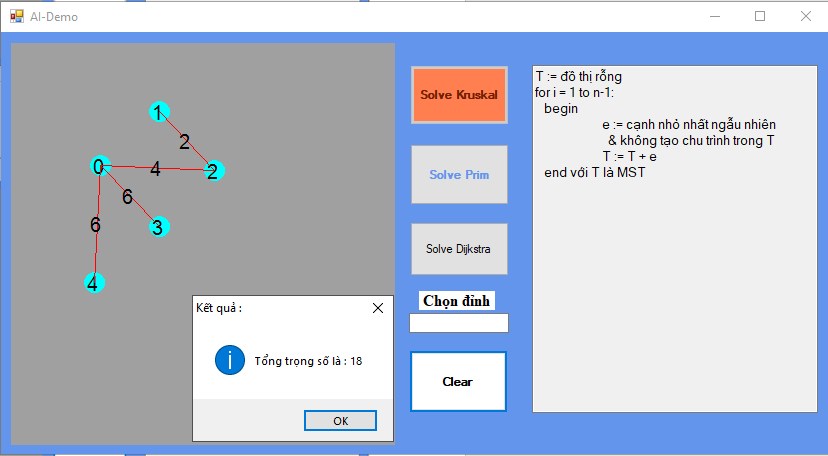


**\* Thuật toán Kruskal**

**+** Cho đồ thị:

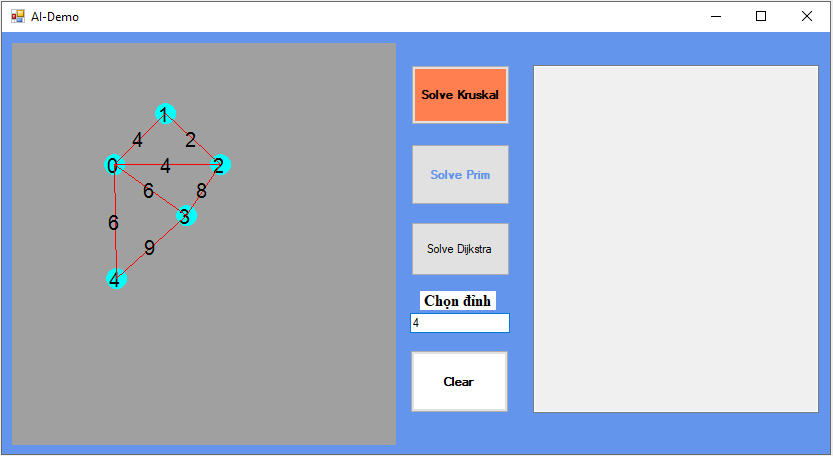


+ Kết quả chương trình:

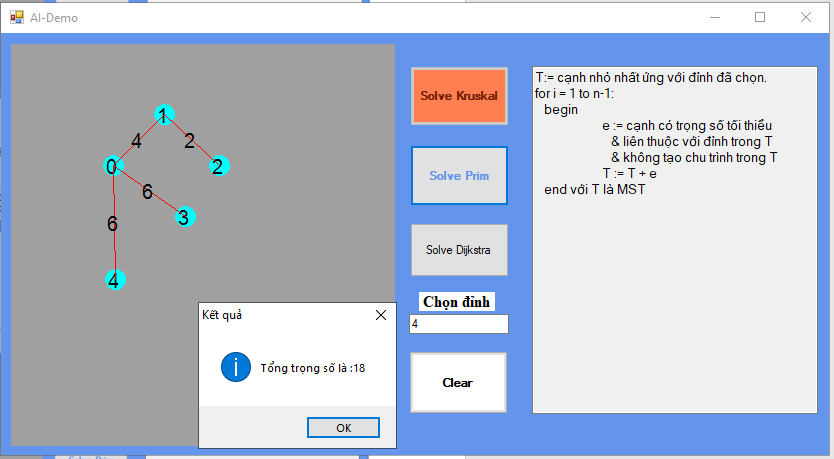


**\* Thuật toán Prim**

+ Cho đồ thị:



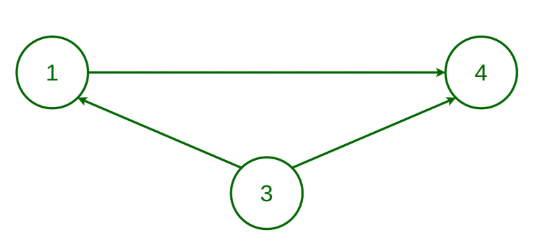
**+** Kết quả chương trình với đỉnh bắt đầu là 4:

****

**Kết luận**

Qua những nội dung đã được trình bày bên trên, ta đã nắm rõ được các khái niệm cơ bản về đồ thị, hiểu cách hoạt động của thuật toán tìm đường đi ngắn nhất Dijkstra, thuật toán tìm cây khung bé nhất Kruskal và Prim. Tuy nhiên ta cũng cần lưu ý một số vấn đề sau:

+ **Thuật toán Dijkstra thất bại khi tìm đường đi ngắn nhất trong đồ thị có chứa trọng số âm.** Ví dụ ta có 3 đỉnh A, B, C và các cạnh có trọng số AB = 3, AC = 4, BC=-2. Ta thấy rằng đường đi ngắn nhất từ A đến C là 1 (A🡪B🡪C) và A đến B là 2 (A🡪C🡪B). Tuy nhiên, nếu ta áp dụng thuật toán Dijsktra với đỉnh bắt đầu là A, ta sẽ kiểm tra đỉnh B trước vì là đỉnh gần nhất và gán nhãn cho đường đi ngắn nhất từ A🡪B = 3.

 + **Thuật toán tìm cây khung bé nhất không hoạt động với đồ thị có hướng.**

🡪 Ở đây không có đường đi nào có thể đến đỉnh 3 nên không thể tạo thành cây bao trùm được.

Ngày nay, các thuật toán nêu trên được ứng dụng rất nhiều vào đời sống thực tiễn như:

+ Dijkstra : bài toán định tuyến router (đã được đề cập trong môn Nhập môn mạng máy tính), tìm đường đi ngắn nhất giữa các thành phố như trong GG Maps,…

+ Thuật toán tìm cây khung bé nhất: bài toán xây dựng hệ thống đường sắt, bài toán nối mạng máy tính.

**Tài liệu tham khảo**

**1.** Giáo trình toán rời rạc, BỘ GIÁO DỤC VÀ ĐÀO TẠO TRƯỜNG ĐẠI HỌC CẦN THƠ KHOA SƯ PHẠM BỘ MÔN TOÁN HỌC, biên soạn: Ths Bùi Anh Kiệt, Ths Trương Quốc Bảo, năm 2003.

**2.** [**https://visualgo.net**](https://visualgo.net)

**3.** Chu Đức Khánh. Lý thuyết đồ thị. Nhà xuất bản đại học Quốc gia Thành phố Hồ

Chí Minh – 2002.

**4.** Giáo trình toán rời rạc, Học viện công nghệ bưu chính viễn thông, biên soạn: Ths Nguyễn Duy Phương, Hà Nội, 2006.