BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC SƯ PHẠM TP. HỒ CHÍ MINH**

Nhóm sinh viên thực hiện:

Trần Thanh Nhã – KHMT-17-005

Nguyễn Phương Nam – KHMT-17-004

Dương Xuân Huy - KHMT-17-002

**LÝ THUYẾT ĐỒ THỊ**

**(GRAPH THEORY)**

Chuyên ngành: **Khoa học máy tính**

Môn học: **Phương pháp toán trong tin học**

Người hướng dẫn: **TS. HUỲNH VĂN ĐỨC**

Tp. Hồ Chí Minh, tháng 8 năm 2018

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC SƯ PHẠM TP. HỒ CHÍ MINH**

**LÝ THUYẾT ĐỒ THỊ**

**(GRAPH THEORY)**

Chuyên ngành: **Khoa học máy tính**

Môn học: **Phương pháp toán trong tin học**

Tp. Hồ Chí Minh, tháng 8 năm 2018

**MU**

CHƯƠNG 1. 2

CHƯƠNG 2. 2

CHƯƠNG 3. 2

CHƯƠNG 4. CÂY KHUNG NHỎ NHẤT (MINIMUM SPANNING TREE) 2

4.1. Định nghĩa 2

4.2. Bài toán tìm cây khung nhỏ nhất 3

4.2.1 Bài toán xây dựng hệ thống đường sắt 3

4.2.2 Bài toán nối mạng máy tính 3

4.3. Thuật toán Kruskal 3

4.3.1 Mô tả thuật toán 4

4.3.2 Cài đặt thuật toán Kruskal 7

4.3.3 Chứng minh tính đúng đắn 7

4.3.4 Độ phức tạp của thuật toán Kruskal 8

4.4. Thuật toán Prim 8

4.4.1 Mô tả thuật toán Prim: 8

4.4.2 Thuật toán Prim 8

4.4.3 Cài đặt thuật toán Prim 12

4.4.4 Độ phức tạp thuật toán Prim 12

4.5. SO SÁNH HAI THUẬT TOÁN Prim và Kruskal 13

4.5.1 Xét các cạnh đưa vào cây 13

4.5.2 . Kiểm tra tính liên thông của đồ thị 13

4.5.3 . Chi phí cho việc kiểm tra chu trình 13

TÀI LIỆU THAM KHẢO 15

**DANH SÁCH HÌNH VẼ**

Hình 4.1 Sơ đồ hình cây 2

Hình 4.2 Cây khung nhỏ nhất của đồ thị 5

Hình 4.3 Cây khung nhỏ nhất 7

Hình 4.4 Cài đặt thuật toán bằng ngôn ngữ Python 7

Hình 4.5 Cây khung có trọng số 11

Hình 4.6 Cài đặt thuật toán Prim bằng ngôn ngữ Python 12

**DANH SÁCH BẢNG**

Bảng 4.1. Thuật toán Kruskal 5

Bảng 4.2. Kết quả chạy ví dụ 1 6

Bảng 4.3. Minh hoạ thuật toán Prim 11

Bảng 4.2. Kết quả chạy ví dụ 11

Bảng 4.2. Liệt kê độ phức tạp thuật toán 12

**LỜI MỞ ĐẦU**

# 

# 

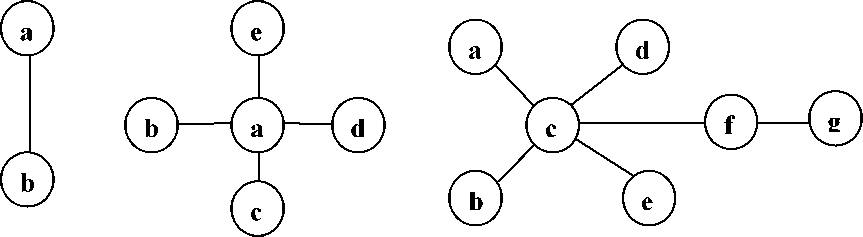
# 

# CÂY KHUNG NHỎ NHẤT (MINIMUM SPANNING TREE)

Một đồ thị liên thông và không có chu trình được gọi là cây. Cây được dùng từ năm 1857, khi nhà toán học Anh tên là Arthur Cayley dùng cây để xác định những dạng khác nhau của hợp chất hóa học. Từ đó cây đã được dùng để giải nhiều bài toán trong nhiều lĩnh vực khác nhau. Trong tin học cây được dùng để tìm kiếm các phần tử trong danh sách hoặc bài toán xây dựng các mạng máy tính với chi phí rẻ với các máy phân tán.

1. Định nghĩa và tính chất cơ bản của cây.

Cây là một đồ thị vô hướng liên thông, không chứa chu trình và có ít nhất hai đỉnh. Một đồ thị vô hướng không chứa chu trình và có ít nhất hai đỉnh gọi là một rừng. Trong một rừng, mỗi thành phần liên thông là một cây.



#### Sơ đồ hình cây

Định lý 1. Nếu T là một cây có n đỉnh thì T có ít nhất hai đỉnh treo.

Định lý 2. Cho đồ thị T=(V,E) có n >=2 đỉnh. Sáu mệnh đề là tương đương:

* T là một cây.
* T liên thông và có n-1 cạnh.
* T không chứa chu trình và có n-1 cạnh.
* T liên thông và mỗi cạnh là cầu.
* Giữa hai đỉnh phân biệt của T luôn có duy nhất một đường đi sơ cấp.
* T không chứa chu trình nhưng khi bổ sung vào một cạnh nối hai đỉnh không kề nhau thì xuất hiện một chu trình.

## Định nghĩa

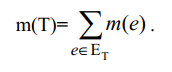
Trong đồ thị liên thông G, nếu ta loại bỏ cạnh nằm trên chu trình nào đó thì ta sẽ được đồ thị vẫn là liên thông. Nếu cứ loại bỏ các cạnh ở các chu trình khác cho đến khi nào đồ thị không còn chu trình (vẫn liên thông) thì ta thu được một cây nối các đỉnh của G. Cây đó gọi là cây khung hay cây bao trùm của đồ thị G.

Tổng quát, nếu G là đồ thị có n đỉnh, m cạnh và k thành phần liên thông thì áp dụng thủ tục vừa mô tả đối với mỗi thành phần liên thông của G, ta thu được đồ thị gọi là rừng khung của G. Số cạnh bị loại bỏ trong thủ tục này bằng m-n+k, số này ký hiệu là v(G) và gọi là chu số của đồ thị G.

## **Bài toán tìm cây khung nhỏ nhất**

Bài toán tìm cây khung nhỏ nhất của đồ thị là một trong số những bài toán tối ưu trên đồ thị tìm được ứng dụng trong nhiều lĩnh vực khác nhau của đời sống. Trong phần này ta sẽ có hai thuật toán cơ bản để giải bài toán này. Trước hết, nội dung của bài toán được phát biểu như sau.

Cho G=(V,E) là đồ thị vô hướng liên thông có trọng số, mỗi cạnh eeE có trọng số m(e)>0. Giả sử T=(Vt,Et) là cây khung của đồ thị G (Vt=V). Ta gọi độ dài m(T) của cây khung T là tổng trọng số của các cạnh của nó:



### Bài toán xây dựng hệ thống đường sắt

Giả sử ta muốn xây dựng một hệ thống đường sắt nối n thành phố sao cho hành khách có thể đi từ bất cứ một thành phố nào đến bất kỳ một trong số các thành phố còn lại. Mặt khác, trên quan điểm kinh tế đòi hỏi là chi phí về xây dựng hệ thống đường phải là nhỏ nhất. Rõ ràng là đồ thị mà đỉnh là các thành phố còn các cạnh là các tuyến đường sắt nối các thành phố tương ứng, với phương án xây dựng tối ưu phải là cây. Vì vậy, bài toán đặt ra dẫn về bài toán tìm cây khung nhỏ nhất trên đồ thị đầy đủ n đỉnh, mỗi đỉnh tương ứng với một thành phố với độ dài trên các cạnh chính là chi phí xây dựng hệ thống đường sắt nối hai thành phố.

### Bài toán nối mạng máy tính

Cần nối mạng một hệ thống gồm n máy tính đánh số từ 1 đến n. Biết chi phí nối máy i với máy j là m(i,j) (thông thường chi phí này phụ thuộc vào độ dài cáp nối cần sử dụng). Hãy tìm cách nối mạng sao cho tổng chi phí là nhỏ nhất. Bài toán này cũng dẫn về bài toán tìm cây khung nhỏ nhất.

Bài toán tìm cây khung nhỏ nhất đã có những thuật toán rất hiệu quả để giải chúng. Trong tài liệu này sẽ xét hai trong số những thuật toán như vậy: thuật toán Kruskal và thuật toán Prim.

## **Thuật toán** **Kruskal**

Thuật toán Kruskal dựa trên mô hình xây dựng cây khung nhỏ nhất bằng thuật toán hợp nhất.

* Thuật toán không xét các cạnh với thứ tự tuỳ ý.
* Thuật toán xét các cạnh theo thứ tự đã sắp xếp theo trọng số.

Để xây dựng tập n-1 cạnh của cây khung nhỏ nhất tạm gọi là tập K, Kruskal đề nghị cách kết nạp lần lượt các cạnh vào tập đó theo nguyên tắc như sau:

* Ưu tiên các cạnh có trọng số nhỏ hơn.
* Kết nạp cạnh khi nó không tạo chu trình với tập cạnh đã kết nạp trước đó.

Đó là một nguyên tắc chính xác và đúng đắn, đảm bảo tập K nếu thu đủ n - 1 cạnh sẽ là cây khung nhỏ nhất.

### **Mô** tả thuật toán

Giải thuật Kruskal không dựa trên tư tưởng của các thuật toán tìm kiếm theo chiều rộng hoặc chiều sâu. Trong các thuật toán này, tại từng bước của quá trình xây dựng T luôn là một cây, chỉ có điều kiện về số đỉnh của T phải đến bước cuối cùng mới thỏa mãn. Còn trong thuật toán Kruskal, trong quá trình xây dựng T có the chưa là cây, nó chỉ thỏa mãn điều kiện không có chu trình.

Thuật toán sẽ xây dựng tập cạnh E t của cây khung nhỏ nhất T=( Vt, E t) theo từng bước. Trước hết sắp xếp các cạnh của đồ thị G theo thứ tự không giảm của trọng số. Bắt đầu từ E t = 0, ở mỗi bước ta sẽ lần lượt duyệt trong danh sách cạnh đã sắp xếp, từ cạnh có độ dài nhỏ đến cạnh có độ dài lớn hơn, để tìm ra cạnh mà việc bo sung nó vào tập E t không tạo thành chu trình trong tập này. Thuật toán sẽ kết thúc khi ta thu được tập E t gồm n-1 cạnh. Cụ thể có thể mô tả như sau:

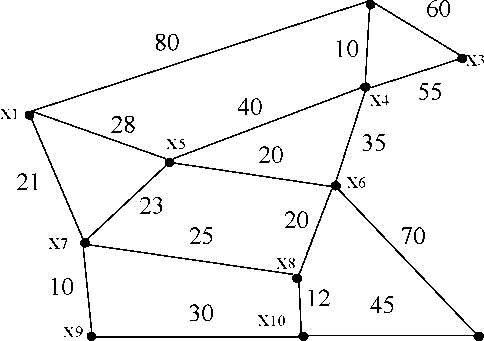
1. Bắt đầu từ đồ thị rỗng T có n đỉnh.
2. Sắp xếp các cạnh của G theo thứ tự không giảm của trọng số.
3. Bắt đầu từ cạnh đầu tiên của dãy này, ta cứ thêm dần các cạnh của dãy đã được xếp vào T theo nguyên tắc cạnh thêm vào không được tạo thành chu trình trong T.
4. Lặp lại Bước 3 cho đến khi nào số cạnh trong T bằng n-1, ta thu được cây khung nhỏ nhất cần tìm.

Từ ý tưởng của thuật toán Kruskal, ta có mô hình minh họa tìm cây khung nhỏ nhất cho đồ thị sau: G = (V, E) vô hướng, n = 5 đỉnh

|  |  |
| --- | --- |
|  | Khởi tạo T = 0 và sắp xếp các cạnh theo thứ tự trọng số tăng dần, ta có :  (A, C) = 1, (A, E) = 1 (C, D) = 2, (C E) = 2 (A, B) = 3 (B, E) = 4 (B, D) = 6 (D, E) = 7 |
|  | Bước 2: Lần lượt lấy từng cạnh trong danh sách đã sắp xếp vào T sao cho không tạo thành chu trình đơn.  🛆 Cạnh (A, C) bổ sung vào T (T = 0) không tạo chu trình nên: T = {(A, C)}. |
|  | Bước 2 (lần 1):  A Cạnh (A, E) bổ sung vào T không tạo chu trình nên: T = {(A, C); (A, E)}  Bước 3 (lần 1): Vì T chưa đủ n-1 phần tử 🡪 Tiếp tục Bước 2. |
|  | Bước 2 (lần 2):  A Cạnh (C, D) bổ sung vào T không tạo chu trình nên: T = {(A, C); (A, E); (C, D)}  Bước 3 (lần 2): Vì T chưa đủ n-1 phần tử 🡪 Tiếp tục Bước 2.  Bước 2 ( lần 3):  A Cạnh (C, E) bổ sung vào T tạo chu trình 🡪 Loại bỏ. |
|  | Bước 2 ( lần 4):  A Cạnh (A, B) bổ sung vào T không tạo chu trình nên: T = {(A, C); (A, E); (C, D); (A, B)} Bước 3 (lần 3): Lúc này ta thấy T đã có đúng n-1 cạnh. Quá trình lặp kết thúc. |
|  | Vậy **T = {(A, C); (A, E); (C, D); (A, B)}** chính là tập cạnh của cây khung nhỏ nhất cần tìm |

##### Thuật toán Kruskal

Ví dụ 1: Tìm cây khung nhỏ nhất của đồ thị cho trong hình bên dưới, đồ thị có 11 đỉnh, 17 cạnh như vậy cây khung có 10 cạnh.



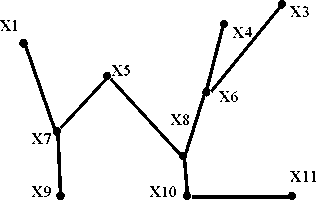
#### Cây khung nhỏ nhất của đồ thị

Bắt đầu từ đồ thị rỗng T có 11 đỉnh.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bước sắp thứ tự các cạnh | | | Bước chọn các cạnh cho cây k hung nhỏ nhất | | | |
| Thứ tự | Cạnh | Trọng số | Bước chọn | Cạnh | Trọng số | |
| 1 | x2 x4 | 10 | 1 | x2 x4 | 10 | |
| 2 | x7 x9 | 10 | 2 | x7 x9 | 10 | |
| 3 | x8 x10 | 12 | 3 | x8 | 12 | |
| 4 | x5 x6 | 20 | 4 | x5 x6 | 20 | |
| 5 | x6 x8 | 20 | 5 | x6 x8 | 20 | |
| 6 | x1 x7 | 21 | 6 | x1 x7 | 21 | |
| 7 | x5 x7 | 23 | 7 | x5 x7 | | 23 |
| 8 | x1 x5 | 25 | 8 | x4 x6 | | 35 |
| 9 | x7 x8 | 25 | 9 | x10 | | 45 |
| 10 | x9 x10 | 30 | 1 | x3 x4 | | 55 |
| 11 | x4 x6 | 35 | Tổng trọng số các cạnh | | | 251 |
| 12 | x4 x5 | 40 | Ghi chú  Sau bước chọn 7 ta không thể chọn các cạnh (x1,x5), (x7,x8) và (x9, x10) vì các cạnh này tạo thành chu trình với các cạnh đã chọn. Tình huống tương tự nếu chọn cạnh (x4,x5) | | | |
| 13 | x10 x11 | 45 |
| 14 | x3 x4 | 55 |
| 15 | x2 x3 | 60 |
| 16 | x6 x11 | 70 |
| 17 | x1 x2 | 80 |

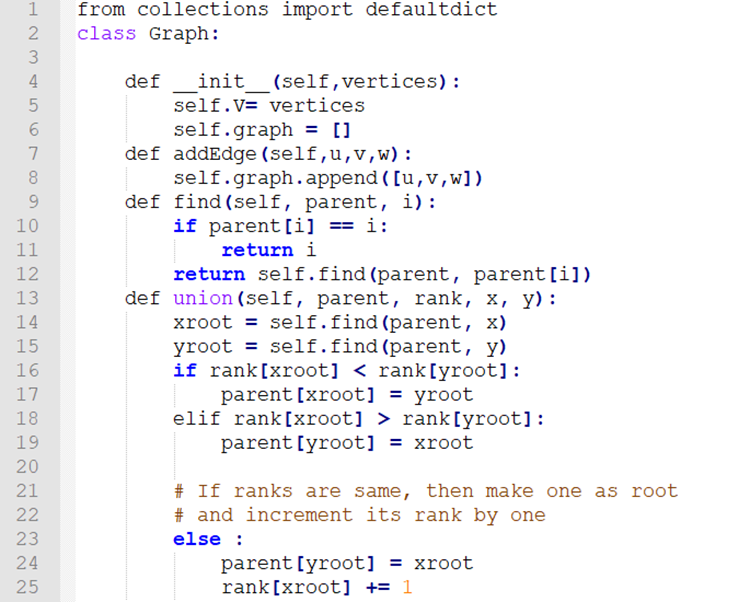
##### Kết quả chạy ví dụ 1

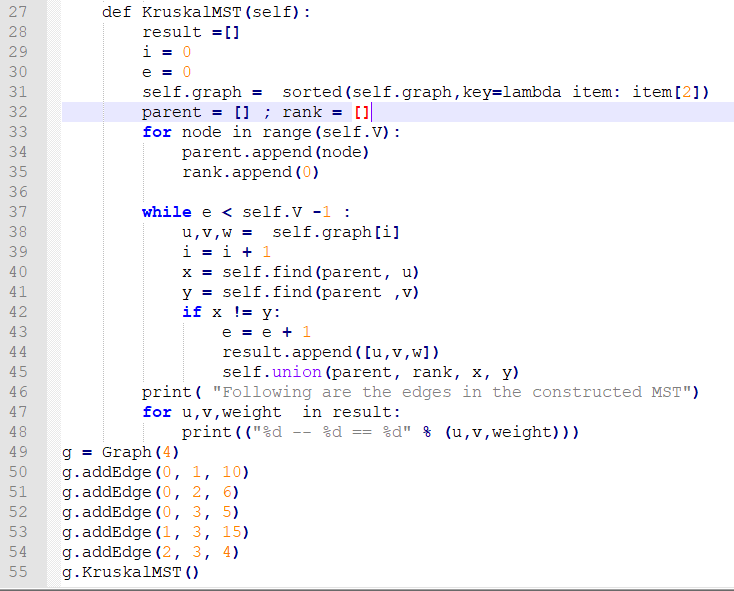
Kết thúc thuật toán được cây khung nhỏ nhất là cây. Trọng số của cây khung nhỏ nhất thu được là (T) = 251.



#### Cây khung nhỏ nhất

### Cài đặt thuật toán Kruskal





#### Cài đặt thuật toán bằng ngôn ngữ Python

### **Chứng** minh tính đúng đắn

Đồ thị thu được theo thuật toán có n-1 cạnh và không có chu trình, vì vậy theo định lý 2 nó là cây khung của đồ thị G. Như vậy, chỉ còn phải chỉ ra rằng T có độ dài nhỏ nhất. Giả sử tồn tại cây S của đồ thị mà c(S)<c(T). Ký hiệu là ek là cạnh đầu tiên trong dãy các cạnh của T không thuộc S. Khi đó đồ thị con của G sinh bởi cây S được bổ sung cạnh ek sẽ chứa một chu trình duy nhất C đị qua ek . Do chu trình C phải chứa cạnh e thuộc S nhưng không thuộc T nên đồ thị con thu được từ S bằng cách thay cạnh e của nó bởi ek (ký hiệu đồ thị này là S ') sẽ là cây khung. Theo cách xây dựng c(ek) < c(e) do đó c(S’) < c(S), đồng thời số cạnh chung của S’ và T đã tăng thêm một so với số cạnh chung của S và T. Lặp lại quá trình trên từng bước một ta có thể biến đổi S thành T và trong mỗi bước tổng độ dài không tăng, tức là c(T) < c(S). Ta thu được chứng tỏ T là cây khung nhỏ nhất.

### Độ phức tạp của thuật toán Kruskal

Trước tiên, việc sắp xếp các cạnh của G theo thứ tự trọng số tăng dần có độ phức tạp *O*(m2), với m là số cạnh của G. Người ta chứng minh được rằng việc chọn

cạnh uk +1 không tạo nên chu trình với k cạnh đã chọn trước đó có độ phức tạp *O*(n2). Do nên độ phức tạp thuật toán là *O*(n2).

## Thuật toán Prim

**Thuật toán Prim** là một thuật toán tham lam để tìm cây bao trùm nhỏ nhất của một đồ thị vô hướng có trọng số liên thông. Nghĩa là nó tìm một tập hợp các cạnh của đồ thị tạo thành một cây chứa tất cả các đỉnh, sao cho tổng trọng số các cạnh của cây là nhỏ nhất.

Thuật toán được tìm ra năm 1930 bởi nhà toán học người Séc Vojtěch Jarník và sau đó bởi nhà nghiên cứu khoa học máy tính Robert C. Prim năm 1957 và một lần nữa độc lập bởi Edsger Dijkstra năm 1959.

### Mô tả thuật toán Prim:

Thuật toán xuất phát từ một cây chỉ chứa đúng một đỉnh và mở rộng từng bước một, mỗi bước thêm một cạnh mới vào cây, cho tới khi bao trùm được tất cả các đỉnh của đồ thị.

* Dữ liệu vào: Một đồ thị có trọng số liên thông với tập hợp đỉnh *V* và tập hợp cạnh *E* (trọng số có thể âm). Đồng thời cũng dùng *V* và *E* để ký hiệu số đỉnh và số cạnh của đồ thị.
* Khởi tạo: *V*mới = {*x*}, trong đó *x* là một đỉnh bất kì (đỉnh bắt đầu) trong *V*, *E*mới = {}
* Lặp lại cho tới khi *V*mới = *V*:
  + Chọn cạnh (*u*, *v*) có trọng số nhỏ nhất thỏa mãn *u* thuộc *V*mới và *v* không thuộc *V*mới (nếu có nhiều cạnh như vậy thì chọn một cạnh bất kì trong chúng)
  + Thêm *v* vào *V*mới, và thêm cạnh (*u*, *v*) vào *E*mới
* Dữ liệu ra: *V*mới và *E*mới là tập hợp đỉnh và tập hợp cạnh của một cây bao trùm nhỏ nhất

### Thuật toán Prim

Cho G = (X, E) là một đồ thị liên thông có trọng số gồm n đỉnh. Thuật toán Prim được dùng để tìm ra cây khung nhỏ nhất của G.

• Bước 1: Chọn tùy ý x0 thuộc X và khởi tạo V:= { x0 }; T := Ø. Trong đó X là tập các đỉnh của đồ thị, V là tập các đỉnh được chọn vào cây khung nhỏ nhất và T là tập các cạnh của cây này.

• Bước 2: Trong số những cạnh nối đỉnh x với đỉnh y mà x ∈ V và y ∈X\V ta chọn cạnh e có trọng số nhỏ nhất. Nếu không có cạnh e thỏa yêu câu: DỪNG (1)

• Bước 3: Thêm đỉnh y vào tập V và thêm cạnh e vào tập T.

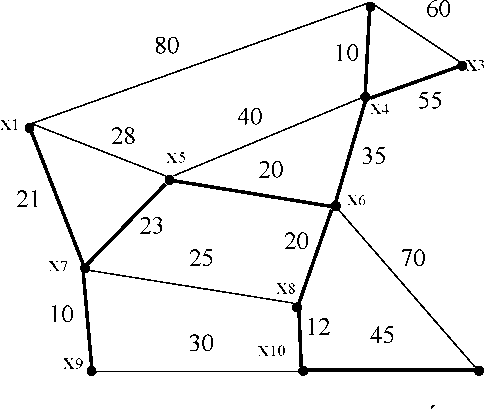
• Bước 4: Nếu T đủ n – 1 phần tử thì DỪNG (2), ngược lại làm tiếp tục bước 2.

**Ví dụ minh họa**: Tìm khung cây nhỏ nhất của bài toán sau:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Hình minh họa** | **U** | **Cạnh (u,v)** | **V \ U** | **Mô tả** |
| [Prim Algorithm 0.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_0.svg) | {} |  | {A,B,C,D,E,F,G} | Đây là đồ thị có trọng số ban đầu. Các số là các trọng số của các cạnh. |
| [Prim Algorithm 1.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_1.svg) | {D} | (D,A) = 5  (D,B) = 9 (D,E) = 15 (D,F) = 6 | {A,B,C,E,F,G} | Chọn một cách tùy ý đỉnh **D** là đỉnh bắt đầu. Các đỉnh **A**, **B**, **E** và **F** đều được nối trực tiếp tới **D** bằng cạnh của đồ thị. **A** là đỉnh gần **D** nhất nên ta chọn **A** là đỉnh thứ hai của cây và thêm cạnh **AD** vào cây. |
| [Prim Algorithm 2.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_2.svg) | {A,D} | (D,B) = 9 (D,E) = 15 (D,F) = 6  (A,B) = 7 | {B,C,E,F,G} | Đỉnh được chọn tiếp theo là đỉnh gần **D** hoặc **A** nhất. **B** có khoảng cách tới **D** bằng 9 và tới **A** bằng 7, **E** có khoảng cách tới cây hiện tại bằng 15, và **F** có khoảng cách bằng 6. **F** là đỉnh gần cây hiện tại nhất nên chọn đỉnh **F** và cạnh **DF**. |
| [Prim Algorithm 3.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_3.svg) | {A,D,F} | (D,B) = 9 (D,E) = 15 (A,B) = 7  (F,E) = 8 (F,G) = 11 | {B,C,E,G} | Thuật toán tiếp tục tương tự như bước trước. Chọn đỉnh **B** có khoảng cách tới **A** bằng 7. |
| [Prim Algorithm 4.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_4.svg) | {A,B,D,F} | (B,C) = 8 (B,E) = 7  (D,B) = 9  chu trình (D,E) = 15 (F,E) = 8 (F,G) = 11 | {C,E,G} | Ở bước này ta chọn giữa **C**, **E**, và **G**. **C** có khoảng cách tới **B** bằng 8, **E** có khoảng cách tới **B** bằng 7, và **G** có khoảng cách tới **F** bằng 11. **E** là đỉnh gần nhất, nên chọn đỉnh **E** và cạnh **BE**. |
| [Prim Algorithm 5.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_5.svg) | {A,B,D,E,F} | (B,C) = 8 (D,B) = 9 chu trình (D,E) = 15 chu trình (E,C) = 5 **V** (E,G) = 9 (F,E) = 8 chu trình (F,G) = 11 | {C,G} | Ở bước này ta chọn giữa **C** và **G**. **C** có khoảng cách tới **E** bằng 5, và **G** có khoảng cách tới **E** bằng 9. Chọn **C** và cạnh **EC**. |
| [Prim Algorithm 6.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_6.svg) | {A,B,C,D,E,F} | (B,C) = 8 chu trình (D,B) = 9 chu trình (D,E) = 15 chu trình (E,G) = 9 **V** (F,E) = 8 chu trình (F,G) = 11 | {G} | Đỉnh **G** là đỉnh còn lại duy nhất. Nó có khoảng cách tới **F** bằng 11, và khoảng cách tới **E** bằng 9. **E** ở gần hơn nên chọn đỉnh **G** và cạnh **EG**. |
| [Prim Algorithm 7.svg](https://vi.wikipedia.org/wiki/T%E1%BA%ADp_tin:Prim_Algorithm_7.svg) | {A,B,C,D,E,F,G} | (B,C) = 8 chu trình (D,B) = 9 chu trình (D,E) = 15 chu trình (F,E) = 8 chu trình (F,G) = 11 chu trình | {} | Hiện giờ tất cả các đỉnh đã nằm trong cây và [cây bao trùm nhỏ nhất](https://vi.wikipedia.org/wiki/C%C3%A2y_bao_tr%C3%B9m_nh%E1%BB%8F_nh%E1%BA%A5t) được tô màu xanh lá cây. Tổng trọng số của cây là 39. |

##### Minh hoạ thuật toán Prim

**Ví dụ 1:**



#### Cây khung có trọng số

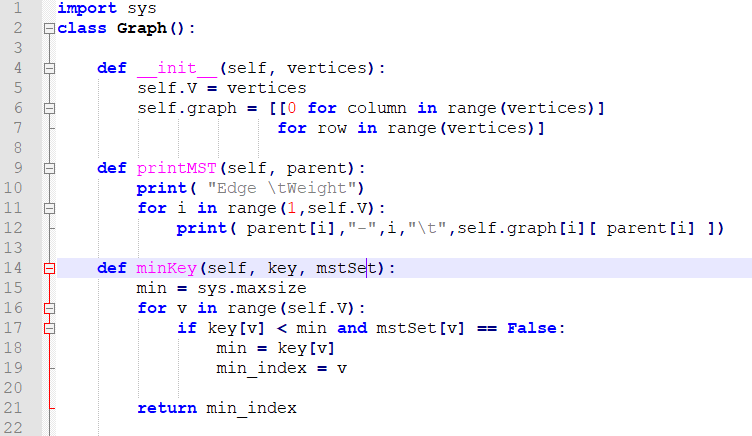
Hình 2. 4 Cây khung có trọng số

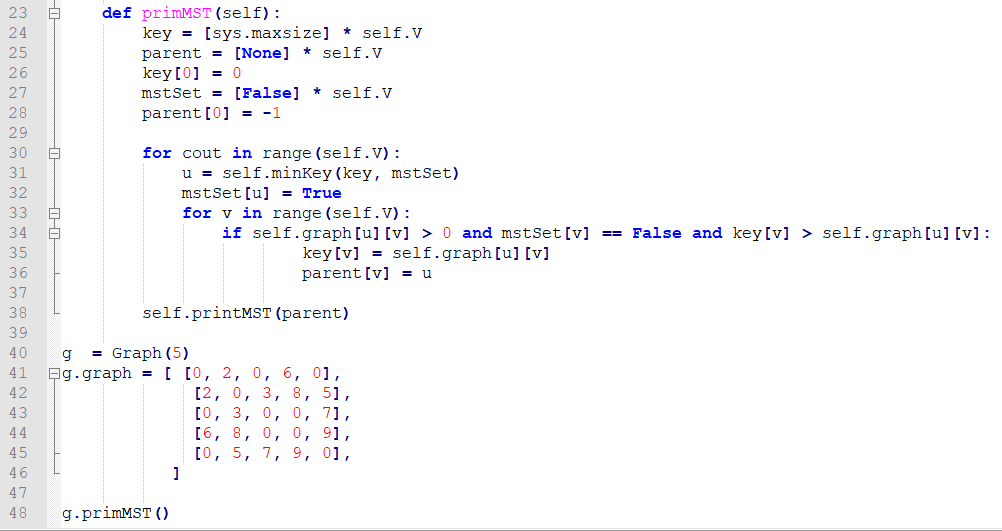
|  |  |  |
| --- | --- | --- |
| Bước | Cạnh | Trọng số |
| 1 | (x7,x9) | 10 |
| 2 | (x7,x1) | 21 |
| 3 | (x7,x5) | 23 |
| 4 | (x5,x6) | 20 |
| 5 | (x6,x8) | 20 |
| 6 | (x8,x10) | 12 |
| 7 | (x6,x4) | 35 |
| 8 | (x4,x2) | 10 |
| 9 | (x10,x11) | 45 |
| 10 | (x4,x3) | 55 |
| l(T) | | 251 |

##### Kết quả chạy ví dụ

Cây khung nhỏ nhất thu được thể hiện bằng nét đậm trên hình và trọng số của nó là l(T) = 251

### Cài đặt thuật toán Prim





#### Cài đặt thuật toán Prim bằng ngôn ngữ Python

### Độ phức tạp thuật toán Prim

|  |  |
| --- | --- |
| Cấu trúc dữ liệu tìm cạnh có trọng số nhỏ nhất | Độ phức tạp thời gian (tổng cộng) |
| Tìm kiếm trên ma trận kề | 0(|V|2) |
| Đống nhị phân và danh sách kề | 0((|V| + |E|) log |V|) = 0(|E| log |V|) |
| Đống Fibonacci và danh sách kề | 0(|E| + |V| log |V|) |

##### Liệt kê độ phức tạp thuật toán

## SO SÁNH HAI THUẬT TOÁN Prim và Kruskal

### Xét các cạnh đưa vào cây

* Prime: thực hiện việc mở rộng tập đã xét (ban đầu chỉ gồm một đỉnh, 0 cạnh thành n đỉnh, n-1 cạnh) dựa trên các cạnh ngắn nhất nối giữa tập đã xét và tập chưa xét. Tư tưởng chính là thêm vào các cạnh ngắn nhất sao cho không tạo ra chu trình. Như vậy, có trường hợp một cạnh sẽ phải xét đi xét nhiều lần rồi mới được chọn, thậm chí không hề được chọn.
* Kruskal: cũng thêm lần lượt các cạnh vào đồ thị, theo thứ tự từ trọng nhỏ nhất đến trọng lớn nhất (như vậy mỗi cạnh sẽ chỉ được duyệt một lần duy nhất). Ta chỉ bổ sung cạnh vào cây khung nếu việc thêm cạnh này không làm phát sinh ra chu trình.

### . Kiểm tra tính liên thông của đồ thị

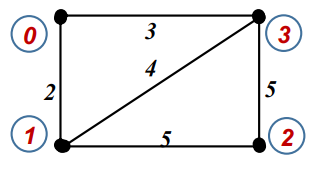
* Prim: cần kiểm tra đồ thị liên thông trước khi thi hành thuật toán.
* Kruskal: không cần kiểm tra đồ thị liên thông trước khi thi hành thuật toán. Nếu quá trình thi hành thuật toán tìm được cây khung/bao trùm thì đồ thị liên thông và ngược lại là đồ thị không liên thông.

### . Chi phí cho việc kiểm tra chu trình

* Prime: mở rộng tập đỉnh đã xét (mỗi lần 1 đỉnh nên không cần kiểm tra chu trình).

Kruskal: kiểm tra nếu khi thêm cạnh đang xét vào cây khung mà không làm phát sinh chu trình thì sẽ chọn cạnh này. Việc kiểm tra chu trình có khả năng gây tốn chi phíGợi ý cải tiến giảm chi nhí kiểm tra chu trình:

* Dùng một mảng Nhãn có kích thước bằng số đỉnh của đồ thị. Giá trị Nhãn[i] tại đỉnh i được khởi tạo bằng với chính chỉ số i của đỉnh.



Ví dụ: Với đồ thị G ở trên ta có

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 |
| Nhãn | 0 | 1 | 2 | 3 |

* Khi chọn một cạnh (u,v) được thêm vào cây khung/cây bao trùm, ta sửa nhãn của tất cả các đỉnh có cùng giá trị với nhãn của đỉnh v thành nhãn của đỉnh u.

Ví dụ:

* + Sau khi thêm cạnh (0,1) vào cây khung/cây bao trùm hay tập T, ta sửa nhãn của tất cả các đỉnh có cùng giá trị với nhãn của đỉnh 1 (là 1) thành nhãn của đỉnh 0 (là 0).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 |
| Nhãn | 0 | 0 | 2 | 3 |

* + Sau đó, khi chọn tiếp cạnh (0, 3), ta sửa nhãn của đỉnh 3 (đang có giá trị là 3) để có cùng giá trị với nhãn của đỉnh 0 (là 0).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 |
| Nhãn | 0 | 0 | 2 | 0 |

* Sau đó, khi chọn cạnh để thêm vào cây khung/cây bao trùm thì chỉ chọn cạnh có nhãn hai đỉnh là khác nhau sẽ không tạo thành chu trình.

Ví du:

* + Khi ta xét tới cạnh (1,3): ta không chọn cạnh này vì đỉnh 1 và đỉnh 3 có cùng nhãn là 0
  + Chọn tiếp cạnh (1,2): 2 đỉnh 1 và 2 có nhãn khác nhau nên cạnh (1,2) sẽ được chọn đưa vào cây khung/cây bao trùm.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Đỉnh | 0 | 1 | 2 | 3 |
| Nhãn | 0 | 0 | 0 | 0 |

Nhận xét: Sau khi ta đã chọn đủ n-1 cạnh, mảng nhãn chỉ chứa một giá trị duy nhất.

TÀI LIỆU THAM KHẢO

Tiếng Việt

Tiếng anh