Bài toán tìm cây khung nhỏ nhất trong đồ thị

I. Cây khung nhỏ nhất

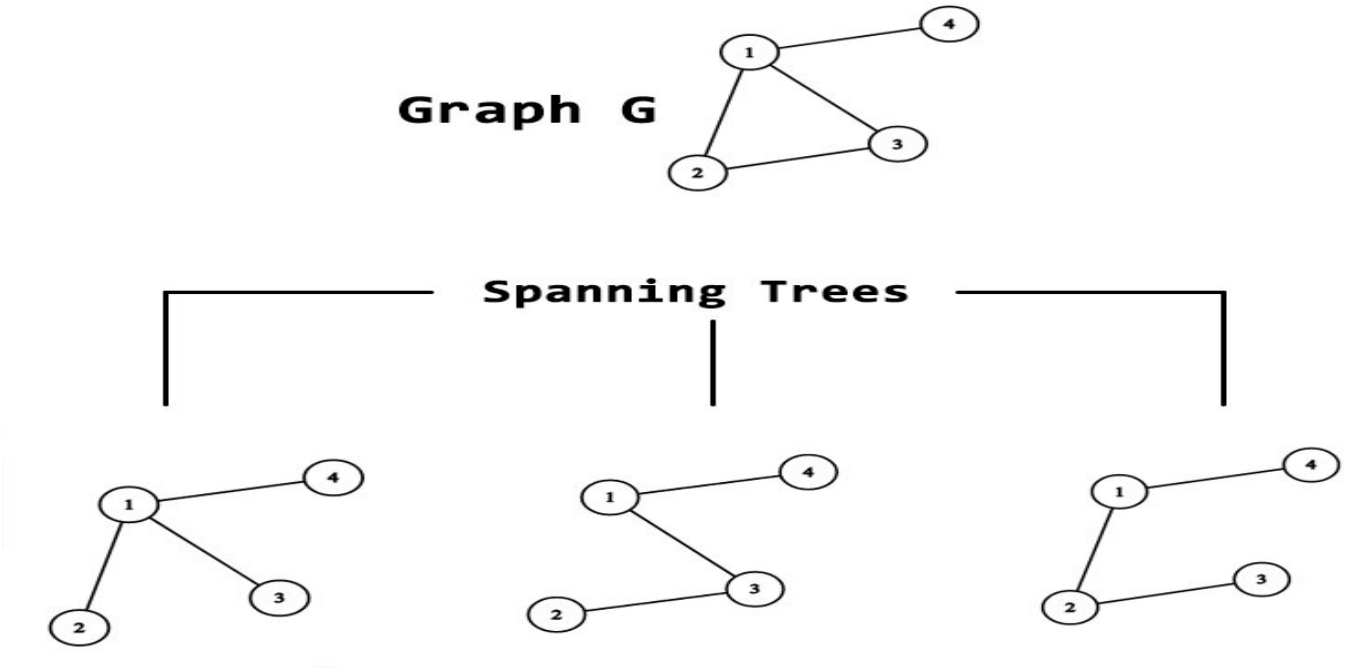
1. Định nghĩa

Theo lý thuyết đồ thị, chúng ta đều biết rằng 1 đồ thị được biểu diễn bằng công thức G = (V,E) , trong đó đồ thị G của chúng ta bao gồm tập các đỉnh V và tập các cạnh E .

Cây khung (*spanning tree*) của đồ thị là một tập hợp các cạnh của đồ thị thỏa mãn tập cạnh này *không chứa chu trình* và *liên thông* (từ một đỉnh bất kì có thể đi tới bất kỳ đỉnh nào khác theo mà chỉ dùng các cạnh trên cây khung)

Trong đồ thị có trọng số, cây khung nhỏ nhất (*minimum spanning tree*) là cây khung có tổng trọng số các cạnh trong cây nhỏ nhất .

Một ví dụ về cây khung trong đồ thị vô hướng không trọng số:



2. Tính chất

Một vài tính chất của cây khung nhỏ nhất trong đồ thị vô hướng có trọng số:

a. **Tính chất chu trình**: Trong một chu trình C bất kỳ, nếu e là cạnh có trọng số lớn nhất **tuyệt đối** (không có cạnh nào có trọng

số bằng e) thì e không thể nằm trên bất kỳ cây khung nhỏ nhất nào.

b. **Đường đi hẹp nhất**: Xét 2 đỉnh u,v bất kỳ trong đồ thị. Nếu w là trọng số của cạnh lớn nhất trên đường đi u từ đến v trên cây khung nhỏ nhất của đồ thị thì ta không thể tìm được đường đi nào từ u đến v trên đồ thị ban đầu chỉ đi qua những cạnh có trọng số nhỏ hơn w.
c. **Tính duy nhất**: Nếu tất cả các cạnh đều có trọng số khác nhau thì chỉ có duy một cây khung nhỏ nhất. Ngược lại, nếu một vài cạnh có trọng số giống nhau thì có thể có nhiều hơn một cây khung nhỏ nhất.
d. **Tính chất cạnh nhỏ nhất**: Nếu e là cạnh có trọng số nhỏ nhất của đồ thị, và không có cạnh nào có trọng số bằng e thì e nằm trong mọi cây khung nhỏ nhất của đồ thị .

II. Các Thuật Toán Tìm Cây Khung Nhỏ Nhất

1. Thuật toán Kruskal

Ý tưởng thuật toán: Ban đầu mỗi đỉnh là một cây riêng biệt, ta tìm cây khung nhỏ nhất bằng cách duyệt các cạnh theo trọng số từ nhỏ đến lớn, rồi hợp nhất các cây lại với nhau.

Cụ thể hơn, giả sử cạnh đang xét nối 2 đỉnh u và v , nếu 2 đỉnh này nằm ở 2 cây khác nhau thì ta thêm cạnh này vào cây khung, đồng thời hợp nhất 2 cây chứa u và v.

Giả sử ta cần tìm cây khung nhỏ nhất của đồ thị G . Thuật toán bao gồm các bước sau:

- Khởi tạo rừng F (tập hợp các cây), trong đó mỗi đỉnh của G tạo thành một cây riêng biệt.
- Khởi tạo tập S chứa tất cả các cạnh của G.
- Chừng nào S còn khác rỗng và F gồm hơn một cây
    - Xóa cạnh nhỏ nhất trong S
    - Nếu cạnh đó nối hai cây khác nhau trong F, thì thêm nó vào F và hợp hai cây kề với nó làm một
    - Nếu không thì loại bỏ cạnh đó.

Khi thuật toán kết thúc, rừng chỉ gồm đúng một cây và đó là một cây khung nhỏ nhất của đồ thị G.

Chứng minh tính đúng đắn của thuật toán :

- Mỗi cạnh (u,v) được xét đến, nó chỉ kết nạp vào câu khung nếu (u,v) thuộc 2 thành phần liên thông khác nhau ⇒ Do đó các cạnh được thêm không tạo thành chu trình
- Do T không có chu trình ⇒ số cạnh được thêm <= n-1 . Ta sẽ chứng minh T có đúng n-1cạnh
  - Giả sử số cạnh được thêm < n-1 ⇒ T gồm hai hay nhiều thành phần liên thông
  - Mặt khác, do G liên thông ⇒ tồn tại các cạnh thuộc nối các thành phần liên thông đó mà không thuộc T . Do đó cạnh đầu tiên nhỏ nhất trong số các cạnh này sẽ được đưa vào do nó không tạo thành chu trình, mâu thuẫn với giả thiết ở trên ⇒ Giả sử sai
  - Vậy số cạnh được thêm vào bằng đúng bằng n-1

Đánh giá độ phức tạp của thuật toán :

Gọi n là số đỉnh của đồ thi, m là số cạnh của đồ thị

Thuật toán gồm 2 phần:

- Sắp xếp mảng m cạnh theo trọng số tăng dần mất độ phức tạp O(mlogm).
- Ta duyệt m cạnh, mỗi cạnh dùng Disjoint Set mất độ phức tạp O(logn) , vậy tổng cộng mất độ phức tạp O(mlogn).

độ phức tạp của thuật toán Kruskal là : O(mlogm +mlogn)..

2. Thuật toán Prim

Ý tưởng thuật toán: Ý tưởng của thuật toán Prim rất giống với ý tưởng của thuật toán Dijkstra (tìm đường đi ngắn nhất trên đồ thị). Nếu như thuật toán Kruskal xây dựng cây khung nhỏ nhất bằng cách kết nạp từng cạnh vào đồ thị thì thuật toán Prim lại kết nạp từng đỉnh vào đồ thị theo tiêu chí: đỉnh được nạp vào tiếp theo phải chưa được nạp và gần nhất với các đỉnh đã được nạp vào đồ thị.
Thuật toán bao gồm các bước sau:

- Khởi tạo tập S là cây khung hiện tại, ban đầu **S** chưa có đỉnh nào.
- Khởi tạo mảng D trong đó Di là khoảng cách ngắn nhất từ đỉnh i đến 1 đỉnh đã được kết nạp vào tập S, ban đầu D[i]= +oo
- Lặp lại các thao tác sau n lần(n là số cạnh của đồ thị)
    - Tìm đỉnh u không thuộc S có Du nhỏ nhất, thêm vào tập S.
    - Xét tất cả các đỉnh v kề u, cập nhật Dv = min(Dv,Wu,v) với Wu,v là trọng số cạnh (u,v). Nếu Dv được cập nhật theo Wu,v thì đánh dấu tracev = u .
    - Thêm cạnh u-trace[u] vào tập cạnh thuộc cây khung nhỏ nhất.

Chứng minh tính đúng đắn :

- Đặt *G* là một đồ thi có trọng số liên thông. Trong mỗi bước, thuật toán Prim chọn một cạnh nối một đồ thị con với một đỉnh không thuộc đồ thị con đó. Vì *G* liên thông nên luôn tồn tại đường đi từ mỗi đồ thị con tới tất cả các đỉnh còn lại. Kết quả *T* của thuật toán Prim là một cây, vì các cạnh và đỉnh được thêm vào *T* là liên thông và cạnh mới thêm không bao giờ tạo thành chu trình với các cạnh cũ. Đặt T_1 là một cây bao trùm nhỏ nhất của *G*. Nếu $T_1=T$ thì *T* là cây bao trùm nhỏ nhất. Nếu không, đặt *e* là cạnh đầu tiên được thêm vào *T* mà không thuộc T_1, và *V* là tập hợp các đỉnh thuộc *T* trước khi thêm *e*. Một đầu của *e* thuộc *V* và đầu kia không thuộc *V*. Vì T_1 là một cây bao trùm của *G*, nên tồn tại đường đi trong T_1 nối hai đầu của *e*. Trên đường đi đó, nhất định tồn tại cạnh *f* nối một đỉnh trong *V* với một đỉnh ngoài *V*. Trong bước lặp khi *e* được thêm vào *Y*, thuật toán cũng có thể chọn *f* thay vì *e* nếu như trọng số của nó nhỏ hơn *e*. Vì *f* không được chọn nên

$$w(f) \geq w(e).$$

- Đặt T_2 là đồ thị thu được bằng cách xóa *f* và thêm *e* vào T_1.
- Dễ thấyT_2 liên thông, có cùng số cạnh như T_1, và tổng trọng số các cạnh không quá trọng số của T_1, nên nó cũng là một cây bao trùm nhỏ nhất của *G* và nó chứa *e* cũng như tất cả các cạnh được thuật

toán chọn trước nó. Lặp lại lập luận trên nhiều lần, cuối cùng ta thu được một cây bao trùm nhỏ nhất của *G* giống hệt như *T*. Vì vậy *T* là một cây bao trùm nhỏ nhất.

Đánh giá độ phức tạp thuật toán :

- Ta duyệt tổng cộng n lần, mỗi lần lấy 1 đỉnh ra khỏi heap mất O(logn) và cập nhật trọng số của tất cả các đỉnh kề với đỉnh đó, tổng số lần cập nhật là m lần, mỗi lần cập nhật ta mất độ phức tạp O(logn) .
- Như vậy, độ phức tạp của thuật toán Prim là O((m+n)logn) với n là số đỉnh và m là số cạnh của đồ thị.