Thuật toán trên đồ thị và Ứng dụng

Khoa Công Nghệ Thông Tin,
Trường Đại Học Thủy Lợi.

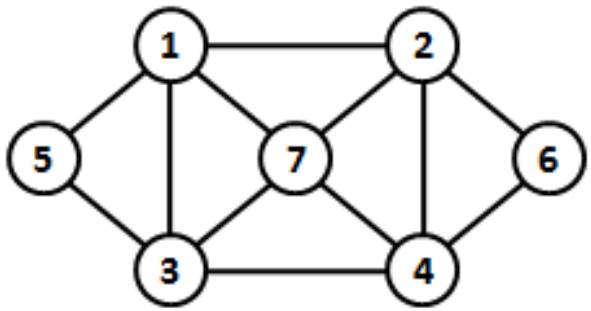
Ngày 28 tháng 12 năm 2017

Đồ thị và những khái niệm liên quan

- Nhiều đối tượng trong cuộc sống hàng ngày của chúng ta có thể được mô hình hóa bởi đồ thị
  - Internet, mạng xã hội (facebook), mạng giao thông, mạng sinh học, ...
- Một đồ thị G là một đối tượng toán học bao gồm 2 tập xác định, $G = (V, E)$
  - V là tập các đỉnh
  - E là tập các cạnh kết nối những đỉnh này
- Đồ thị có nhiều kiểu: có hướng, vô hướng, đa đồ thị, ...

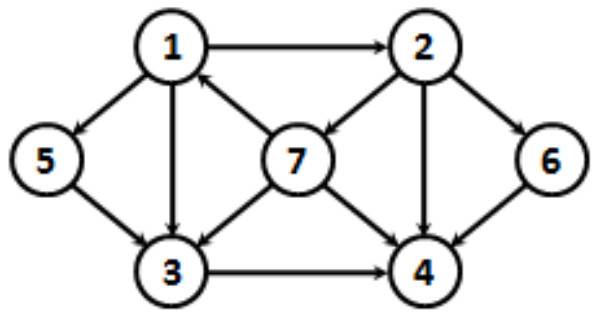
Những định nghĩa

- Một đồ thị vô hướng $G = (V, E)$
  - $V = (v_1, v_2, \ldots, v_n)$ là tập các đỉnh hoặc nút
  - $E \subseteq V \times V$ là tập các cạnh (cũng được gọi là các cạnh vô hướng). E là tập các cặp không có thứ tự sao cho $u \neq v \in V$
  - $(u, v) \in E$ khi và chỉ khi $(v, u) \in E$



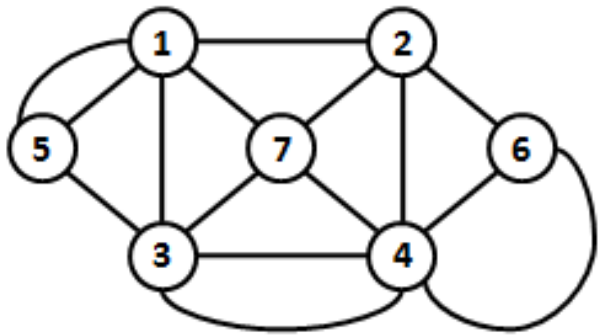
Những định nghĩa

- Một đồ thị có hướng $G = (V, E)$
  - $V = (v_1, v_2, \ldots, v_n)$ là tập các đỉnh hoặc nút
  - $E \subseteq V \times V$ là tập các cung (cũng được gọi là các cạnh có hướng). E là tập các cặp có thứ tự sao cho $u \neq v \in V$



Đa đồ thị

- Một đa đô thị vô hướng (có hướng) là một đồ thị có nhiều cạnh (cung), tức là những cạnh (cung) này có cùng hai đỉnh đầu cuối
- Hai đỉnh có thể được kết nối bởi nhiều hơn một cạnh (cung)



Những định nghĩa

- Cho một đồ thị $G = (V, E)$, với mỗi $(u, v) \in E$, chúng ta nói u và v là hai cạnh kề nhau
- Cho một đồ thị vô hướng $G = (V, E)$
  - bậc của một đỉnh v là số lượng cạnh nối với nó: $deg(v) = \sharp\{(u, v) \mid (u, v) \in E\}$
- Cho một đồ thị có hướng $G = (V, E)$
  - Một cung đến của một đỉnh là một cung đi vào nó
  - Một cung đi của một đỉnh là một cung đi ra từ nó
  - indegree (outdegree) của một đỉnh v là số lượng cung đến (đi) của nó

$$deg^+(v) = \sharp\{(v, u) \mid (v, u) \in E\}, deg^-(v) = \sharp\{(u, v) \mid (u, v) \in E\}$$

Những định nghĩa

Theorem

Cho một đồ thị vô hướng $G = (V, E)$, chúng ta có

$$2 \times |E| = \sum_{v \in V} deg(v)$$

Theorem

Cho một đồ thị có hướng $G = (V, E)$, chúng ta có

$$\sum_{v \in V} deg^{+}(v) = \sum_{v \in V} deg^{-}(v) = |E|$$

Định nghĩa - Đường đi, Chu trình

- Cho một đồ thị $G = (V, E)$, một đường đi từ đỉnh u tới đỉnh v trong G là một chuỗi $\langle u = x_0, x_1, \ldots, x_k = v \rangle$ trong đó $(x_i, x_{i+1}) \in E, \forall i = 0, 1, \ldots, k-1$
  - u: điểm (nút) bắt đầu
  - v: điểm kết thúc
  - k là chiều dài của đường đi (tức là, tổng số cạnh của đường đi)
- Một chu trình là một đường đi sao cho nút đầu và nút cuối là giống nhau
- Một đường đi (chu trình) được gọi là đơn giản nếu nó không chứa những cạnh (cung) lặp
- Một đường đi (chu trình) được gọi là căn bản nếu nó không chứa những nút lặp

Sự liên thông

- Cho một đồ thị vô hướng $G = (V, E)$. G được gọi là liên thông nếu với bất kỳ cặp (u, v) $(u, v \in V)$, luôn tồn tại một đường đi từ u đến v trong G
- Cho một đồ thị có hướng $G = (V, E)$, G được gọi là
  - **liên thông yếu** nếu đồ thị vô hướng tương ứng của G (bằng cách bỏ đi hướng trên các cung của nó) là liên thông
  - **liên thông mạnh** nếu với bất kỳ cặp (u, v) $(u, v \in V)$, luôn tồn tại một đường đi từ u đến v trong G
- Cho một đồ thị vô hướng $G = (V, E)$
  - một cạnh e được gọi là **cầu** nếu loại bỏ e khỏi G làm tăng số lượng thành phần liên thông của G
  - một đỉnh v được gọi là **điểm khớp** nếu loại bỏ nó khỏi G làm tăng số lượng thành phần liên thông của G

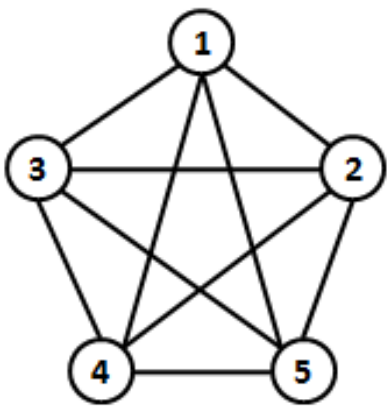
Sự liên thông

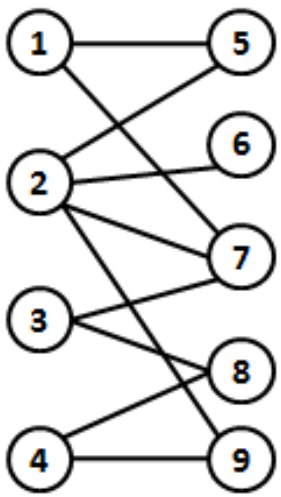
Theorem

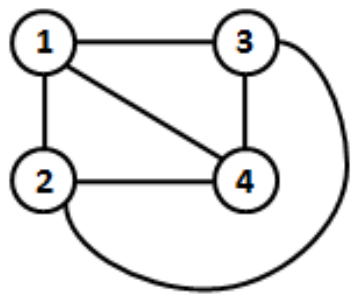
Một đồ thị vô hướng liên thông G có thể trở thành có hướng (mỗi cạnh của G có hướng) để đạt được một đồ thị liên thông mạnh khi và chỉ khi mỗi cạnh của G nằm trên ít nhất một chu trình

Những đồ thị đặc biệt

- Đồ thị hoàn thiện (Complete graphs) K_n: đồ thị vô hướng $G = (V, E)$ trong đó $|V| = n$ và $E = \{(u, v) \mid u, v \in V\}$
- Đồ thị hai phía (Bipartie graphs) $K_{n,m}$: là đồ thị vô hướng $G = (V, E)$ trong đó $V{=}X \cup Y$, $X \cap Y = \emptyset$, $|X| = n$, $|Y| = m$, $(u, v) \in E \Rightarrow u \in X \wedge v \in Y$
- Đồ thị phẳng (Planar graphs): là đồ thị có thể được vẽ trên một mặt phẳng sao cho các cạnh chỉ giao với nhau tại các đỉnh chung







Đồ thị phẳng - Công thức đa diện Euler

Theorem

Cho một đồ thị phẳng liên thông có n đỉnh và m cạnh. Số lượng vùng bị chia bởi G là $m - n + 2$.

Đồ thị phẳng - Định lý Kuratowski

Definition

Một **phép chia nhỏ** của đồ thị G là một đồ thị mới thu được bằng cách thay thế một vài cạnh bằng những đường đi sử dụng những đỉnh, cạnh mới (mỗi cạnh được thay thế bằng một đường đi)

Theorem

Kuratowski *Một đồ thị G là phẳng khi và chỉ khi nó không chứa một phân nhỏ của $K_{3,3}$ hoặc K_5*

Biểu diễn đồ thị

- 2 cách chuẩn để biểu diễn một đồ thị $G = (V, E)$
  - Danh sách kề
    - ★ Thích hợp cho những đồ thị thưa
    - ★ $Adj[u] = \{v \mid (u, v) \in E\}, \forall u \in V$
  - Ma trận kề
    - ★ Thích hợp cho những đồ thị dày
    - ★ $A = (a_{ij})_{n \times n}$ sao cho (giả sử $V = \{1, 2, \ldots, n\}$)

$$a_{ij} = \begin{cases} 1 & \text{nếu } (i, j) \in E, \\ 0 & \text{ngược lại} \end{cases}$$

Biểu diễn đồ thị

- Trong một vài trường hợp, chúng ta có thể sử dụng ma trận liên thuộc (incidence matrix) để biểu diễn một đồ thị có hướng $G = (V, E)$

$$b_{ij} = \begin{cases} -1 & \text{nếu cạnh } j \text{ đi ra khỏi đỉnh } i, \\ 1 & \text{nếu cạnh } j \text{ đi vào đỉnh } i, \\ 0 & \text{ngược lại} \end{cases}$$

Tìm kiếm theo chiều sâu (DFS)

- DFS ban đầu duyệt một đỉnh được lựa chọn (gọi là Nguồn)
- DFS duyệt những cạnh kề của một đỉnh được thăm gần nhất v mà đỉnh này vẫn còn những cạnh đi ra từ nó
- Khi tất cả cạnh của v đã được duyệt, thực hiện tìm kiếm **quay lui** để duyệt những cạnh đi ra khỏi đỉnh mà từ đó v được duyệt
- Tiến trình tiếp tục cho đến khi tất cả đỉnh có thể vươn tới từ nguồn ban đầu đều được duyệt
- Nếu tồn tại bất kỳ đỉnh nào không được duyệt, thì DFS chọn một trong số chúng làm nguồn mới và bắt đầu tìm kiếm từ đỉnh đó

Tìm kiếm theo chiều sâu (DFS)

- Thông tin quan trọng được lưu lại trong quá trình tìm kiếm DFS
  - $u.d$ là thời gian bắt đầu duyệt: thời điểm khi đỉnh u được duyệt lần đầu
  - $u.f$ là thời gian kết thúc duyệt: thời điểm khi việc tìm kiếm kết thúc kiểm tra danh sách kề của đỉnh u

Tìm kiếm theo chiều sâu (DFS)

Algorithm 1: DFS-VISIT(G, u)

$t \leftarrow t + 1$;
$u.d \leftarrow t$;
$u.color \leftarrow$ GRAY;
foreach $v \in G.Adj[u]$ **do**
    if $v.color$=*WHITE* **then**
        $v.p \leftarrow u$;
        DFS-VISIT(G, v);
$u.color \leftarrow$ BLACK;
$t \leftarrow t + 1$;
$u.f \leftarrow t$;

Tìm kiếm theo chiều sâu (DFS)

Algorithm 2: DFS(G)

foreach $u \in G.V$ **do**
    $u.color \leftarrow$ WHITE;
    $u.p \leftarrow$ NULL;
$t \leftarrow 0$;
foreach $u \in G.V$ **do**
    if $u.color = WHITE$ **then**
        DFS-VISIT(G, u);

Tìm kiếm theo chiều sâu (DFS)

Với bất kỳ hai đỉnh u và v, chính xác một trong những điều kiện sau thỏa mãn:

- $[u.d, u.f]$ và $[v.d, v.f]$ là hoàn toàn rời rạc, và u hoặc v không phải là hậu duệ của đỉnh kia trong rừng tìm kiếm DFS
- $[u.d, u.f]$ được chứa hoàn toàn bên trong $[v.d, v.f]$, và u là một hậu duệ của v trong rừng tìm kiếm DFS
- $[v.d, v.f]$ được chứa hoàn toàn bên trong $[u.d, u.f]$, và v là một hậu duệ của u trong rừng tìm kiếm DFS

Tìm kiếm theo chiều sâu (DFS)

Phân loại cạnh

- **Cạnh cây** (Tree edges): (u, v) là một cạnh cây nếu v được duyệt đầu tiên bằng cách duyệt cạnh (u, v)
- **Cạnh lùi** (Back edges): (u, v) là một cạnh lùi nếu v là một tổ tiên của u trong cây DFS
- **Cạnh tiến** (Forward edges): (u, v) là một cạnh tiến nếu u là một tổ tiên của v trong cây DFS
- **Cạnh xuyên** (Crossing edges): những cạnh còn lại của đồ thị cho trước

Tìm kiếm theo chiều rộng (BFS)

- Cho một đồ thị $G = (V, E)$ và một đỉnh nguồn s, khoảng cách của một đỉnh v được định nghĩa là chiều dài (số lượng cạnh) của đường đi ngắn nhất từ s đến v
- BFS duyệt một cách hệ thống các định có thể vươn tới được từ s
  - Duyệt các đỉnh có khoảng cách 1, sau đó
  - Duyệt các đỉnh có khoảng cách 2, sau đó
  - Duyệt các đỉnh có khoảng cách 3, sau đó
  - ...

Tìm kiếm theo chiều rộng (BFS)

Algorithm 3: BFS(G, s)

```
s.color ← GRAY;
s.d ← 0;
Q ← ∅;
Enqueue(Q, s);
while Q ≠ ∅ do
    u ← Dequeue(Q);
    foreach v ∈ G.Adj[u] do
        if v.color = WHITE then
            v.color ← GRAY;
            v.d ← u.d + 1;
            v.p ← u;
            Enqueue(Q, v);
    u.color ← BLACK;
```

Tìm kiếm theo chiều rộng (BFS)

Algorithm 4: BFS(G)

foreach $u \in G.V$ **do**
    $u.color \leftarrow$ WHITE;
    $u.d \leftarrow \infty$;
    $u.p \leftarrow$ NULL;

foreach $u \in G.V$ **do**
    if $u.color = WHITE$ **then**
        BFS(G, u);

Ứng dụng của DFS, BFS

- DFS và BFS: tính toán những thành phần liên thông của một đồ thị cho trước
- BFS: Tìm đường đi ngắn nhất (chiều dài của một đường đi được định nghĩa là số lượng cạnh của đường đi đó)
- BFS: Kiểm tra liệu một đồ thị cho trước có là đồ thị hai phía không
- BFS và DFS: Phát hiện chu trình trong một đồ thị vô hướng
- DFS: tính toán những thành phần liên thông mạnh của một đồ thị có hướng cho trước
- DFS: tính toán những cầu và điểm khớp của một đồ thị vô hướng liên thông
- DFS: sắp xếp topo trên một đồ thị có hướng không chu trình (DAG)
- BFS và DFS: Tìm đường đi dài nhất trên một cây

Tính thành phần liên thông

- Cho một đồ thị vô hướng $G = (V, E)$, chúng ta muốn tính tất cả những thành phần liên thông của G
- Áp dụng DFS (hoặc BFS) cho đỉnh nguồn cho trước u sẽ tìm tất cả những đỉnh cùng thành phần liên thông của u

Algorithm 5: COMPUTE-CC(G)

```
foreach u ∈ G.V do
    u.color ← WHITE;
foreach u ∈ G.V do
    if u.color = WHITE then
        C ← new set;
        DFS-CC(G, u, C);
        output(C);
```

Tính thành phần liên thông

Algorithm 6: DFS-CC(G, u, C)

```
Insert(C, u);
u.color ← GRAY;
foreach v ∈ G.Adj[u] do
    if v.color=WHITE then
        DFS-CC(G, v, C);
```

Tính thành phần liên thông mạnh

Cho một đồ thị có hướng $G = (V, E)$

1. Gọi DFS(G) để tính thời gian hoàn thành từ tất cả các đỉnh V
2. Tính đồ thị còn lại $G^T = (V, E^T)$ của G: $E^T = \{(u, v) \mid (v, u) \in E\}$
3. Gọi DFS(G^T), nhưng ở trong vòng lặp (LOOP) chính, xét các đỉnh của V theo thứ tự giảm của thời gian hoàn thành được tính trong dòng 1
4. Các đỉnh của mỗi cây trong rừng DFS của dòng 3 hình thành một thành phần liên thông mạnh của G

Kiểm tra liệu một đồ thị cho trước có là hai phía

SAMSUNG

- Gọi BFS từ một vài đỉnh
- Tô màu những đỉnh bậc-chẵn bởi "BLACK" và những đỉnh bậc-lẻ bởi "WHITE"
- Nếu tồn tại một đỉnh sao cho cả hai điểm đầu cuối có cùng màu, thì G không phải là đồ thị hai phía

Sắp xếp topo

- Cho một đồ thị có hướng không chu trình $G = (V, E)$
- Sắp xếp các đỉnh của G sao cho nếu (u, v) là một cung của G thì u xuất hiện trước v theo thứ tự

Sắp xếp topo: sử dụng DFS

- Gọi DFS(G) để tính thời gian hoàn thành cho tất cả các đỉnh
- Bất kỳ khi nào một đỉnh được hoàn thành, chèn nó vào trước của một danh sách liên kết L
- Trả lại danh sách liên kết L

Sắp xếp topo: sử dụng hàng đợi

Algorithm 7: TOPO-SORT(G)

Tính in-degree $d(v)$ với mỗi đỉnh v của G;
$Q \leftarrow \emptyset$;
foreach $v \in G.V$ **do**
    if $d(v) = 0$ **then**
        Enqueue(Q, v);
while $Q \neq \emptyset$ **do**
    $v \leftarrow$ Dequeue(Q);
    output(v);
    foreach $u \in G.Adj[v]$ **do**
        $d(u) \leftarrow d(u) - 1$;
        if $d(u) = 0$ **then**
            Enqueue(Q, u);

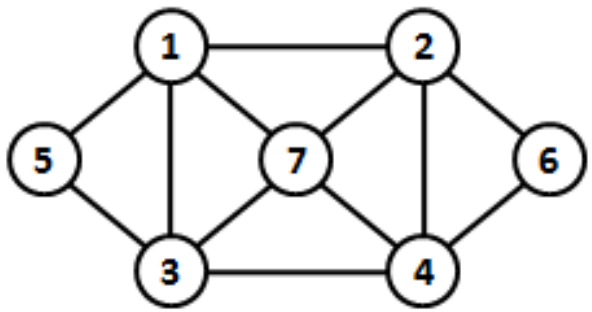
Tìm đường đi dài nhất trên một cây

1. Áp dụng BFS (hoặc DFS) từ một nút s để tìm nút xa nhất v từ s
2. Áp dụng BFS (hoặc DFS) từ nút v (tìm được ở bước 1) để tìm nút xa nhất u từ v
3. Đường đi duy nhất từ u đến v là đường đi dài nhất trên cây cho trước

Chu trình Euler và Hamilton

Definition

- Một chu trình (đường đi) đơn giản mà thăm mỗi cạnh của một đồ thị vô hướng $G = (V, E)$ chính xác 1 lần được gọi là **chu trình (đường đi) Euler** của G
- Những đồ thị chứa chu trình Euler được gọi là **đồ thị Euler**

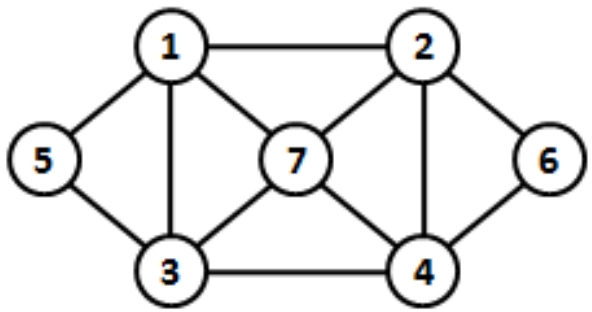


Chu trình Euler là $1, 5, 3, 1, 7, 3, 4, 7, 2, 4, 6, 2, 1$

Chu trình Euler và Hamilton

Definition

- Một chu trình (đường đi) đơn giản mà thăm mỗi nút của một đồ thị vô hướng $G = (V, E)$ chính xác 1 lần được gọi là **chu trình (đường đi) Hamilton** của G
- Những đồ thị chứa chu trình Hamilton được gọi là **đồ thị Hamilton**



Chu trình Hamilton là $1, 2, 6, 4, 7, 3, 5, 1$

Chu trình Euler và Hamilton

SAMSUNG

Theorem

Một đồ thị vô hướng liên thông $G = (V, E)$ là Euler khi và chỉ khi mỗi đỉnh của G có bậc chẵn

Chu trình Euler và Hamilton

- G là liên thông và bậc của mỗi nút là chẵn. Do đó, bậc của mỗi nút lớn hơn hoặc bằng 2
- $\Rightarrow$ tồn tại một chu trình $C = v_1, v_2, ..., v_k, v_1$ trên G
- Loại bỏ tất cả các cạnh của C, chúng ta thu được một đồ thị G' mà được chia ra thành các thành phần liên thông $G_1, ..., G_q$.
- Mỗi G_i là liên thông và bậc của mỗi nút của G_i là chẵn.
- $\Rightarrow$, tồn tại một chu trình Euler C_i trên G_i
- Chúng ta xây dựng một chu trình Euler của G như sau:
  - Bắt đầu từ v_1, chúng ta di chuyển dọc theo chu trình Euler của thành phần liên thông chứa v_1 và kết thúc tại v_1
  - Đi đến v_2. Nếu thành phần liên thông chứa v_2 mà chưa từng được thăm, thì chúng ta đi dọc chu trình Euler của thành phần liên thông này từ v_2 và kết thúc tại v_2
  - Đi đến v_3. Nếu thành phần liên thông chứa v_3 mà chưa từng được thăm, thì chúng ta đi dọc chu trình Euler của thành phần liên thông này từ v_3 và kết thúc tại v_3
  - ...
  - Quay trở lại v_1

Thuật toán để tìm chu trình Euler

Algorithm 8: EULER-CYCLE(G)

```
Stack S ← ⊘;
Stack CE ← ⊘;
u ← select a vertex of G.V;
Push(S, u);
while S ≠ ⊘ do
    x ← Top(S);
    if G.Adj[x] ≠ ⊘ then
        y ← select a vertex of G.Adj[x];
        Push(S, y);
        Remove (x, y) from G;
    else
        x ← Pop(S); Push(CE, x);
while CE ≠ ⊘ do
    v ← Pop(CE);
    output(v);
```

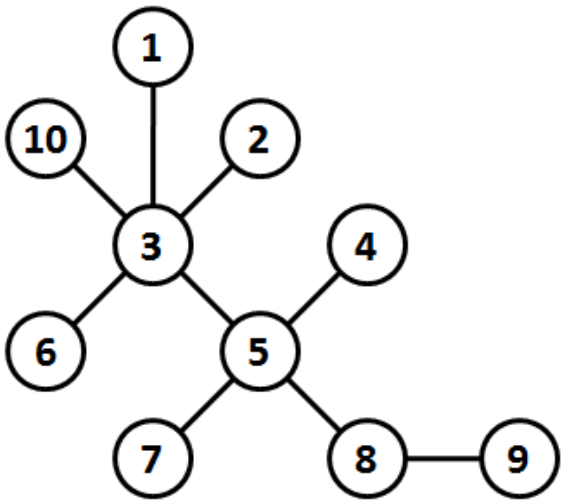
Định lý Dirak

Theorem

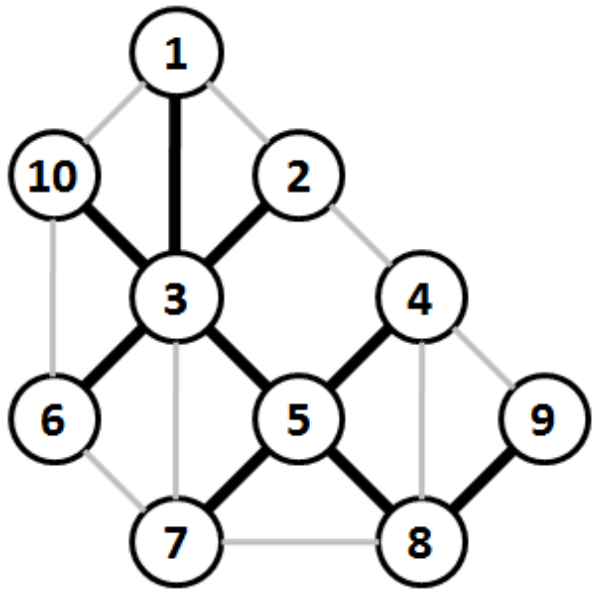
(Dirak 1952) *Một đồ thị vô hướng $G = (V, E)$ trong đó bậc của mỗi đỉnh lớn hơn hoặc bằng $\frac{|V|}{2}$ là đồ thị Hamilton*

Cây và Cây bao trùm

- Một cây là một đồ thị vô hướng liên thông không chứa chu trình
- Một cây bao trùm của một đồ thị vô hướng liên thông $G = (V, E)$ là một cây $T = (V, F)$ với $F \subseteq E$



a. Cây



b. Cây bao trùm (những cạnh in đậm)

Cây

Theorem

Cho một đồ thị vô hướng $T = (V, E)$. Chúng ta có:

- *Nếu T là một cây thì T không có chu trình và bao gồm $|V| - 1$ cạnh*
- *Nếu T không có chu trình nào và bao gồm $|V| - 1$ cạnh thì T liên thông*
- *Nếu T liên thông và bao gồm $|V| - 1$ cạnh thì mỗi cạnh của T là một cầu*
- *Nếu T liên thông và mỗi cạnh là một cầu thì với mỗi cặp $u, v \in V$, tồn tại một đường duy nhất trong T kết nối chúng*
- *Nếu với mỗi cặp $u, v \in V$ tồn tại một đường duy nhất trong T kết nối chúng, thì T không chứa chu trình và một chu trình sẽ được tạo ra nếu chúng ta thêm một cạnh kết nối bất kỳ cặp nút nào của nó*

Cây bao trùm nhỏ nhất (MST)

- Cho một đồ thị vô hướng có trọng số $G = (V, E)$, mỗi cạnh $e \in E$ được gắn với một trọng số $w(e)$
- Trọng số của cây bao trùm T được định nghĩa là

$$w(T) = \sum_{e \in E_T} w(e)$$

  với E_T là tập các cạnh của T
- Tìm một cây bao trùm của G sao cho tổng trọng số trên các cạnh là nhỏ nhất

Theorem

Với bất kỳ đồ thị G có trọng số phân biệt trên các cạnh, **MST** *$\mathcal{T}$ của G thỏa mãn những thuộc tính sau:*

- *Thuộc tính cut: với bất kỳ cut $(X, \overline{X})$ của G, $\mathcal{T}$ phải chứa những cạnh ngắn nhất xuyên qua cut*
- *Thuộc tính chu trình: đặt C là một chu trình trong G, $\mathcal{T}$ không chứa những cạnh dài nhất trong C*

Thuật toán Kruskal

Algorithm 9: KRUSKAL($G = (V, E)$)

$C \leftarrow$ tập các cạnh của G;
$E_T \leftarrow \emptyset$;
$V_T \leftarrow \emptyset$;
while $|V_T| < |V|$ **do**
    $(u, v) \leftarrow$ một cạnh ngắn nhất của C;
    $C \leftarrow C \setminus \{(u, v)\}$;
    if $E_T \cup \{(u, v)\}$ *không tạo bất kỳ chu trình* **then**
        $E_T \leftarrow E_T \cup \{(u, v)\}$;
        $V_T \leftarrow V_T \cup \{u, v\}$;
return (V_T, E_T);

Thuật toán Prim

Algorithm 10: PRIM($G = (V, E)$)

$s \leftarrow$ tập các cạnh của V;
$S \leftarrow V \setminus \{s\}$;
$V_T \leftarrow \{s\}$;
$E_T \leftarrow \emptyset$;
foreach $v \in V$ **do**
    $d(v) \leftarrow w(s, v)$;
    $near(v) \leftarrow s$;
while $|V_T| < |V|$ **do**
    $v \leftarrow \text{argMin}_{u \in S} d(u)$;
    $S \leftarrow S \setminus \{v\}$;
    $V_T \leftarrow V_T \cup \{v\}$;
    $E_T \leftarrow E_T \cup \{(v, near(v))\}$;
    foreach $u \in S$ **do**
        if $d(u) > w(u, v)$ **then**
            $d(u) \leftarrow w(u, v)$;
            $near(u) \leftarrow v$;
return (V_T, E_T);

Bài toán đường đi ngắn nhất

- Cho một đồ thị $G = (V, E)$, mỗi cạnh e được gắn với một trọng số $w(e)$.
  - **Bài toán đường đi ngắn nhất nguồn-đơn** Tìm những đường đi ngắn nhất từ một nút nguồn cho trước s tới tất cả các nút khác của G
  - **Bài toán đường đi ngắn nhất tất-cả-cặp-đỉnh** Tìm những đường đi ngắn nhất giữa mọi cặp đỉnh u, v trong G

Thuật toán Bellman-Ford

- Đồ thị không có chu trình âm

Algorithm 11: Bellman-Ford($G = (V, E), s$)

foreach $v \in V$ **do**
    $d(v) \leftarrow w(s, v)$;
    $p(v) \leftarrow s$;
$d(s) \leftarrow 0$;
foreach $k = 1, \ldots, n-2$ **do**
    foreach $v \in V \setminus \{s\}$ **do**
        foreach $u \in V$ **do**
            if $d(v) > d(u) + w(u, v)$ **then**
                $d(v) \leftarrow d(u) + w(u, v)$;
                $p(v) \leftarrow u$;

Bài toán đường đi ngắn nhất trên đồ thị có hướng không chu trình (DAG)

- Cho một đồ thị có hướng không chu trình (DAG) và một nút nguồn $s \in V$. Tìm những đường đi ngắn nhất từ s đến tất cả những nút khác của G

Algorithm 12: ShortestPathAlgoDAG($G = (V, E), s$)

$L \leftarrow$ Topological sort vertices of G;
foreach $v \in V$ **do**
    $d(v) \leftarrow w(s, v)$;
$d(s) \leftarrow 0$;
foreach $v \in L$ **do**
    foreach $u \in G.Adj[v]$ **do**
        $d(u) \leftarrow \min(d(u), d(v) + w(v, u))$;

Thuật toán Dijkstra

- Đồ thị không có cạnh trọng số âm

Algorithm 13: Dijkstra($G = (V, E), s$)

---

foreach $x \in V$ **do**
    $d(x) \leftarrow w(s, x)$;
    $pred(x) \leftarrow s$;
$NonFixed \leftarrow V \setminus \{s\}$;
$Fixed \leftarrow \{s\}$;
while $NonFixed \neq \emptyset$ **do**
    (*get the vertex v of $NonFixed$ such that $d(v)$ is minimal*);
    $v \leftarrow \text{argMin}_{u \in NonFixed} d(u)$;
    $NonFixed \leftarrow NonFixed \setminus \{v\}$;
    $Fixed \leftarrow Fixed \cup \{v\}$;
    foreach $x \in NonFixed$ **do**
        if $d(x) > d(v) + w(v, x)$ **then**
            $d(x) \leftarrow d(v) + w(v, x)$;
            $pred(x) \leftarrow v$;

Đường đi ngắn nhất Tất-cả-cặp-đỉnh - Thuật toán Floyd-Warshall

Algorithm 14: Floyd-Warshall($G = (V, E)$)

```
foreach u ∈ V do
    foreach v ∈ V do
        d(u, v) ← w(u, v);
        p(u, v) ← u;
foreach z ∈ V do
    foreach u ∈ V do
        foreach v ∈ V do
            if d(u, v) > d(u, z) + d(z, v) then
                d(u, v) ← d(u, z) + d(z, v);
                p(u, v) ← p(z, v);
```

Đường đi ngắn nhất giữa hai đỉnh rời nhau - Thuật toán Suurballe

Algorithm 15: Suurballe(G, s, t)

Input: $G = (V, E, w)$, $s, t \in V$

Output: Tập các cung của Đường đi ngắn nhất của Cặp rời nhau từ s đến t trong G

Áp dụng thuật toán Dijkstra trên G;

$P_1 \leftarrow$ đường đi ngắn nhất từ s đến t trong G;

foreach $v \in V$ **do**

  $d_G(s, v) \leftarrow$ khoảng cách ngắn nhất từ s đến v trong G;

foreach $(u, v) \in E$ **do**

  $w'(u, v) \leftarrow w(u, v) - d_G(s, v) + d_G(s, u)$;

$E' \leftarrow \{\}$;

foreach *arc* $(u, v) \in E$ **do**

  if $(u, v) \in P_1$ **then**

    if $(v, u) \notin E$ **then**

      $E' \leftarrow E' \cup \{(v, u)\}$;

    $w'(v, u) \leftarrow 0$;

  else

    $E' \leftarrow E' \cup \{(u, v)\}$;

$G' \leftarrow (V, E', w')$;

Áp dụng thuật toán Dijkstra trên G';

$P_2 \leftarrow$ đường đi ngắn nhất từ s đến t trong G';

$P \leftarrow \{(u, v) \mid (u, v) \in P_1 \wedge (v, u) \in P_2 \vee (u, v) \in P_2 \wedge (v, u) \in P_1\}$;

return tập các cung của P_1 và P_2 không bao gồm P;