TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

_____*_____

Báo cáo môn học

SEMINAR 2

THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG

Nguyễn Đức Thắng - Nguyễn Anh Tuấn - Phạm Thị Thu Phương
Lớp: KSTN Toán Tin K61

Giảng viên hướng dẫn: TS.Đoàn Duy Trung

Hà Nội - 10/2019

Mục lục

1 Giới thiệu thuật toán **3**

1.1 Đặt vấn đề 3

1.2 Tổng quan 4

1.2.1 Định nghĩa đường đi trên đồ thị 4

1.2.2 Định nghĩa tính liên thông của đồ thị 4

1.2.3 Các cách biểu diễn đồ thị 4

2 Thuật toán tìm kiếm theo chiều rộng **6**

2.1 Thuật toán 6

2.2 Chứng minh tính đúng đắn 10

2.3 Đánh giá độ phức tạp 13

3 Tính toán song song cho BFS **15**

3.1 Ý tưởng 16

3.2 Xây dựng thuật toán song song BFS 17

3.2.1 Đặt vấn đề 17

3.2.2 Cấu trúc dữ liệu bag 17

3.2.3 Thuật toán BFS song song 21

4 Phân công công việc của các thành viên **23**

Chương 1

Giới thiệu thuật toán

1.1 Đặt vấn đề

Lý thuyết đồ thị là một trong những hướng nghiên cứu hiện nay được rất nhiều người quan tâm và phát triển, bởi có quá nhiều ứng dụng thực tế cần đến. Một bài toán quan trọng trong lý thuyết đồ thị là bài toán duyệt tất cả các đỉnh có thể đến từ một đỉnh xuất phát nào đó. Vấn đề này đưa về một bài toán liệt kê mà yêu cầu của nó là không bỏ sót hay lặp lại bất cứ đỉnh nào. Chính vì vậy mà ta phải xây dựng những thuật toán cho phép duyệt 1 cách thống nhất các đỉnh, những thuật toán như vậy gọi là những thuật toán tìm kiếm trên đồ thị và ở đây ta quan tâm đến 2 thuật toán cơ bản nhất: thuật toán tìm kiếm theo chiều sâu (DFS) và thuật toán tìm kiếm theo chiều rộng (BFS).

Trong phạm vi bài báo cáo này, nhóm em xin được trình bày về thuật toán tìm kiếm theo chiều rộng BFS.

Một số bài toán sử dụng thuật toán BFS là:

- Tìm tất cả các đỉnh trong 1 thành phần liên thông của đồ thị.
- Xét tính liên thông của đồ thị hay tìm số thành phần liên thông của đồ thị.
- Tìm đường đi ngắn nhất giữa hai đỉnh u, v cho trước của **đồ thị không**

có trọng số.

- Kiểm tra xem một đồ thị có là đồ thị hai phía.
- Kiểm tra cạnh cầu, đếm số cạnh cầu của đồ thị.

1.2 Tổng quan

1.2.1 Định nghĩa đường đi trên đồ thị

Cho đồ thị $G = (V, E)$, một đường đi độ dài p từ đỉnh s đến đỉnh f là dãy $x[0..p]$ thỏa mãn $x[0] = s$, $x[p] = f$ và $(x[i], x[i+1]) \in E$ với mọi $i : p - 1 \geq i \geq 0$.

Đường đi nói trên còn có thể biểu diễn bởi dãy các cạnh:
$(s = x[0], x[1]), (x[1], x[2]), ..., (x[p-1], x[p] = f)$. Đỉnh s được gọi là *đỉnh đầu*, đỉnh f được gọi là *đỉnh cuối* của đường đi.

1.2.2 Định nghĩa tính liên thông của đồ thị

Cho đồ thị vô hướng $G = (V, E)$, G gọi là *liên thông* nếu luôn tồn tại đường đi giữa mọi cặp đỉnh phân biệt của đồ thị. Nếu G không liên thông thì chắc chắn nó sẽ là hợp của hai hay nhiều đồ thị con liên thông, các đồ thị con này đôi một không có đỉnh chung. Các đồ thị con liên thông rời nhau như vậy được gọi là các *thành phần liên thông* của đồ thị đang xét.

1.2.3 Các cách biểu diễn đồ thị

- Chúng ta có thể biểu diễn đồ thị bằng một **ma trận kề** A có kích thước n x n trong đó:

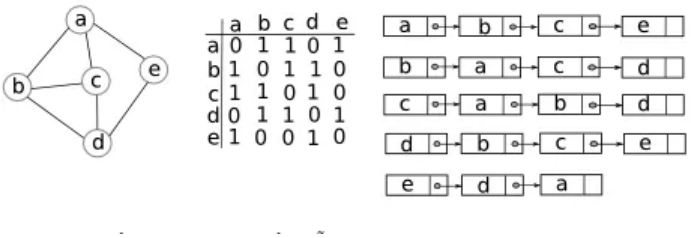
$$A[u, v] = \begin{cases} 1 & (uv \in E) \quad (1.1a) \\ 0 & (\text{otherwise}) \quad (1.1b) \end{cases}$$

- Sử dụng **danh sách kề**:

  Với mỗi đỉnh $u \in V$, ta lưu trữ một danh sách các đỉnh kề với nó. Như vậy, đỉnh u cần một danh sách có $d(u)$ phần tử. Do đó tổng số phần tử của các danh sách là:

$$\sum_{u \in V} d(u) = 2m$$

Ví dụ về hai cách biểu diễn đồ thị cho trong hình dưới đây:



Hình 1.1: Biểu diễn đồ thị

Ta còn có thể kết hợp cách biểu diễn danh sách kề với một vài cấu trúc dữ liệu khác. Cụ thể, thay vì dùng danh sách liên kết để biểu diễn các đỉnh kề với một đỉnh u, ta còn có thể dùng bảng băm hoặc cấu trúc cây để biểu diễn. Trong bài báo cáo này, ta không dùng các cấu trúc như vậy.

Chương 2

Thuật toán tìm kiếm theo chiều rộng

2.1 Thuật toán

- Ý tưởng:

  Sử dụng hàng đợi thay vì sử dụng ngăn xếp để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm. Về cơ bản BFS bao gồm 2 thao tác: cho trước 1 đỉnh của đồ thị và thêm các đỉnh kề với đỉnh đó vào danh sách có thể hướng tới tiếp theo.

- Mục đích:

  Tìm kiếm đường đi ngắn nhất và tìm khoảng cách ngắn nhất từ 1 đỉnh gốc cho trước tới 1 đỉnh đích hoặc tới tất cả các đỉnh khác trong đồ thị.

- Mô tả:

  Bước 1: Với mỗi đỉnh v thuộc đồ thị G, ta gán khoảng cách từ v đến s là d[v] bằng ∞.

  Bước 2: xây dựng 1 hàng đợi rỗng(tức là hàng đợi chưa có phần tử nào).

Bước 3: Chèn đỉnh gốc s vào hàng đợi. Đồng thời đánh dấu đỉnh s đã được thăm.

Bước 4: Lấy ra đỉnh u là đỉnh đầu tiên trong hàng đợi và quan sát nó:

- Nếu đỉnh này chính là đỉnh đích, dừng quá trình tìm kiếm và trả về kết quả.
- Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được quan sát trước đó vào hàng đợi.

Bước 5: Giả sử v là đỉnh kề với u và v chưa được thăm, sau khi thực hiện xong bước 4 ta đánh dấu v đã thăm và cập nhật lại d[v].

d[v] = d[u] +1

Bước 6: Xây dựng mảng Trace[v] để truy vết lại đường đi từ s đến v.

Bước 7: Nếu hàng đợi là rỗng thì tất cả các đỉnh có thể đến được đều đã được quan sát-dừng việc tìm kiếm.

Bước 8: Nếu hàng đợi không rỗng thì quay về Bước 4.

- Mã giả

BFS($G(V, E), s$):

for each $v \in V$

    d[v] $\leftarrow \infty$

$C \leftarrow$ an empty Queue

Enqueue(C, s)

mark s visited

d[s] $\leftarrow 0$

while $C \neq \emptyset$

    $u \leftarrow$ Dequeue(C)

    for all $uv \in E$ and v is unvisited

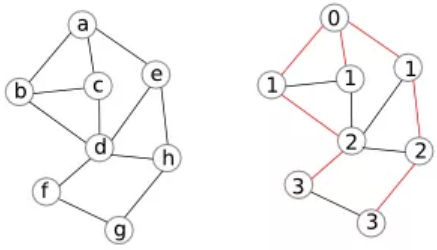
        Enqueue(C, v)

mark v visited

$\mathrm{d}[v] \leftarrow \mathrm{d}[u] + 1$

$\mathrm{Trace}[v] \leftarrow u$

- Ví dụ:



Hình 2.1: Thứ tự duyệt BFS

Các bước thực thi thuật toán BFS(đỉnh a là đỉnh bắt đầu) như sau:

Bước 1: Đặt $\mathrm{d}[v] = \infty$ với mỗi đỉnh v thuộc đồ thị G. Xây dựng hàng đợi C chưa có phần tử nào.

Bước 2: Đặt $\mathrm{d}[a] = 0$. Cho a vào hàng đợi và đánh dấu a đã thăm.

$C = \{a\}$

Bước 3: Lấy a ra khỏi hàng đợi. Nhận thấy, 3 đỉnh b, c, e là 3 đỉnh kề của a và đều chưa được thăm. Ta lần lượt cho b, c, e vào hàng đợi và đánh dấu chúng đã được thăm.

$C = \{b, c, e\}$

Bước 4: Cập nhật:

$\mathrm{d}[b] = \mathrm{d}[c] = \mathrm{d}[e] = \mathrm{d}[a] + 1 = 1$

$\mathrm{Trace}[b] = \mathrm{Trace}[c] = \mathrm{Trace}[e] = a$

Bước 5: Hàng đợi chưa rỗng. Ta tiếp lúc ra lấy đỉnh b(đỉnh đầu tiên trong hàng đợi) từ hàng đợi. Ta có d và c là 2 đỉnh kề của b nhưng chỉ có đỉnh d chưa được thăm nên ta cho đỉnh d vào cuối hàng đợi C.

$C = \{c, e, d\}$

Bước 6: Cập nhật:

d[d] = d[b] + 1 = 2

Trace[d] =b

Bước 7: Lấy đỉnh c ra khỏi hàng đợi, đỉnh c có đỉnh kề là d và b tuy nhiên 2 đỉnh này đã được thăm rồi. Ta chuyển sang bước 8.

$C = \{e, d\}$

Bước 8: Tương tự, ta lấy đỉnh e ra khỏi hàng đợi và thêm đỉnh h vào cuối hàng đợi.

$C = \{d, h\}$

Cập nhật:

d[h] = d[e] + 1 = 2

Trace[h] =e

Bước 9: Xóa đỉnh d và thêm đỉnh f vào cuối hàng đợi.

Cập nhật:

d[f] = d[d] + 1 = 3

Trace[f] =d

$C = \{h, f\}$

Bước 10: Xóa đỉnh h và thêm đỉnh g vào cuối hàng đợi.

Cập nhật:

d[g] = d[h] + 1 = 3

Trace[g] =h

$C = \{f, g\}$

Bước 11: Xóa đỉnh f khỏi hàng đợi.

Bước 12: Xóa đỉnh g khỏi hàng đợi.

Bước 13: Kiểm tra thấy hàng đợi đã rỗng, dừng và đưa ra kết quả.

$\mathrm{d}[b] = \mathrm{d}[c] = \mathrm{d}[e] = 1$

$\mathrm{d}[d] = \mathrm{d}[h] = 2$

$\mathrm{d}[f] = \mathrm{d}[g] = 3$

Input: đồ thị bên trái. Ta thực thi thuật toán BFS được:

Output: đồ thị bên phải.

Các số ứng với các đỉnh tương ứng là nhãn của các đỉnh đó. Nhãn của mỗi đỉnh thu được sau khi duyệt BFS chính là khoảng cách ngắn nhất từ đỉnh xuất phát a tới đỉnh đó. Những cạnh màu đỏ là những cạnh được ghé thăm khi duyệt đồ thị bằng thuật toán BFS.

- Nhận xét:

  Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể.

  Qúa trình tìm kiếm theo chiều rộng cho ta một cây BFS gốc s. Quan hệ cha con trên cây được định nghĩa là: nếu từ đỉnh u tới thăm đỉnh v thì u là nút cha của v.

2.2 Chứng minh tính đúng đắn

Trước hết, trong đồ thị có hướng hoặc vô hướng, ta định nghĩa khoảng cách ngắn nhất từ đỉnh u đến v là $\delta(u, v)$ được tính bằng số cạnh tối thiểu trong tất cả đường đi từ u đến v, hoặc bằng ∞ nếu không tồn tại đường đi từ u đến v. Ta có một số bổ đề sau:

Bổ đề 2.2.1. *Cho $G = (V, E)$ là đồ thị có hướng hoặc vố hướng và $s \in V$ tùy ý. Khi đó, với bất kì cạnh $(u, v) \in E$ thì: $\delta(s, v) \leq \delta(s, u) + 1$*

Chứng minh. Nếu tồn tại đường đi từ s đến u thì cũng tồn tại đường đi từ s đến v (Do $(u, v) \in E$). Khi đó, hiển nhiên đường đi ngắn nhất từ s đến v không thể dài hơn đường đi ngắn nhất từ s đến u rồi qua cạnh (u, v) và như vậy thì bất đẳng thức được thỏa mãn.

Nếu không tồn tại đường đi từ s đến u thì $\delta(s, u) = \infty$. Do đó $\delta(s, v) \leq \infty$ và bất đẳng thức được thỏa mãn.

Vậy bổ đề đã được chứng minh. □

Bổ đề 2.2.2. *Cho $G = (V, E)$ là đồ thị có hướng hoặc vô hướng và thuật toán BFS duyệt trên G bắt đầu từ đỉnh nguồn $s \in V$. Khi kết thúc thuật toán, với mỗi đỉnh $v \in V$, giá trị $d[v]$ thỏa mãn: $d[v] \geq \delta(s, v)$*

Chứng minh. Nếu bất kì đỉnh $u \in V$ không được đẩy vào hàng đợi trong thuật toán, khi đó $d[u]$ không được cập nhật nên $d[u] = \infty$. Hiển nhiên bất đẳng thức được chứng minh.

Ta xét các đỉnh $v \in V$ được đẩy vào hàng đợi. Sử dụng phương pháp quy nap để chứng minh.

Dễ thấy $d[s] = 0 = \delta(s, s)$. Giả sử bất đẳng thức đúng với đỉnh v, tức là: $d[v] \geq \delta(s, v)$. Ta cần chứng minh với mỗi đỉnh t được khám phá trong quá trình duyệt từ đỉnh v phải thỏa mãn bất đẳng thức, tức là: $d[t] \geq \delta(s, t)$.

Thật vậy, từ bổ đề 2.2.1 và thuật toán BFS ta thu được:

$d[t] = d[v] + 1 \geq \delta(s, v) + 1 \geq \delta(s, t)$

Do đó, bất đẳng thức thỏa mãn với mọi đỉnh $v \in V$ được đẩy vào hàng đợi. Vậy bổ đề đã được chứng minh. □

Bổ đề 2.2.3. *Giả sử trong khi thực hiện BFS trên đồ thị $G = (V, E)$, hàng đợi Q chứa các đỉnh $< v_1, v_2, ..., v_r >$ trong đó v_1 ở đầu hàng đợi và v_r ở cuối hàng đợi. Khi đó: $d[v_r] \leq d[v_1] + 1$ và $d[v_i] \leq d[v_{i+1}]$ với $i = 1, 2, ..., r - 1$*

Chứng minh. Ta thực hiện chứng minh bằng phương pháp quy nạp. Dễ thấy bổ đề đúng với trường hợp ban đầu khi hàng đợi chỉ có đỉnh nguồn s.

Với bước quy nạp, ta phải chứng minh bổ đề đúng với cả hai trường hợp lấy đỉnh ra khỏi hàng đợi lẫn đưa vào hàng đợi. Trong trường hợp v_1 được lấy ra khỏi hàng đợi, khi đó v_2 được đưa lên đầu hàng đợi. Theo giả thuyết

quy nạp ta có $d[v_1] \leq d[v_2]$ và ta có $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$ và các bất đẳng thức khác được giữ nguyên. Do đó bổ đề đúng trong trường hợp lấy đỉnh ra khỏi hàng đợi

Tiếp theo, ta xét trường hợp cho đỉnh vào hàng đợi. Khi đó, cuối của hàng đợi là v_{r+1}. Giả sử v_0 là đỉnh vừa lấy ra để xét. Khi đó ta có $d[v_{r+1}] = d[v_0]+1$. Mà $d[v_0] \leq d[v_1]$ và $d[v_r] \leq d[v_0] + 1$ nên ta có: $d[v_{r+1}] = d[v_0] + 1 \leq d[v_1] + 1$ và $d[v_r] \leq d[v_{r+1}]$. Do đó bổ đề đúng trong trường hợp đưa đỉnh vào hàng đợi.

Vậy bổ đề đã được chứng minh. □

Định lý 2.2.4. *Giả sử $G = (V, E)$ là đồ thị có hướng hoặc vô hướng và thuật toán BFS trên G xuất phát từ đỉnh nguồn $s \in V$. Khi đó thuật toán sẽ tìm kiếm được mọi đỉnh $v \in V$ cùng thành phần liên thông với s và $d[v]$ là khoảng cách ngắn nhất từ đỉnh s đến v.*

Chứng minh. Ta bắt đầu với trường hợp $v \in V$ và không tồn tại đường đi từ s đến v.Khi đó, $\delta(s, v) = \infty$. Theo bổ đề 2.2.2, có $d[v] \geq \delta(s, v)$ suy ra $d[v] = \infty$. Vì $d[v]$ không có giá trị hữu hạn mà theo thuật toán BFS sẽ thay đổi các giá trị $d[v]$ thành hữu hạn nếu được khám phá. Do đó, thuật toán không thể khám phá ra những đỉnh thuộc thành phần liên thông khác với đỉnh nguồn s.

Đối với những đỉnh u thuộc cùng thành phần liên thông với s, ta định nghĩa $V_k = \{u \in V : \delta(s, u) = k\}$. Ta thực hiện phương pháp quy nạp theo k. Ở đó với $\forall u \in V_k$:

- u được khám phá.
- $d[u] = k$

Với $k = 0$, ta có $V_0 = \{s\}$ do s là đỉnh nguồn.Khi đó s được khám phá và $d[s] = 0$ nên thỏa mãn giả thiết quy nạp.

Ta chứng minh định lý đúng với k+1. Thật vậy, dựa vào bổ đề 2.2.3 và giả thiết quy nạp ở trên, ta suy ra để thêm đỉnh thuộc V_{k+1} thì thuật toán đã phải thêm tất cả các đỉnh thuộc V_k. Với bất kì $t \in V_{k+1}$, gọi u là đỉnh

kề với t và thuộc đường đi ngắn nhất từ s đến t. Khi đó, $u \in V_k$ nên u đã được đưa vào hàng đợi và được khám phá. Khi thuật toán duyệt đến đỉnh u, thuật toán BFS sẽ thêm tất cả cạnh kề với u mà chưa được khám phá vào hàng đợi để chờ duyệt. Do đó, t sẽ được thuật toán BFS khám phá và $d[t] = d[u] + 1 = k + 1$. Vậy bổ đề đúng với trường hợp $k + 1$

Vậy bổ đề đã được chứng minh. □

Với định lý trên, ta đã chứng minh được tính đúng đắn của thuật toán. Hơn thế nữa, thuật toán BFS có thể tính được khoảng cách ngắn nhất từ một đỉnh đến đỉnh gốc.

2.3 Đánh giá độ phức tạp

Thuật toán BFS tìm kiếm trên đồ thị bắt đầu từ đỉnh gốc và thăm tất cả các đỉnh có đường đi đến đỉnh gốc. Do đó, cách biểu diễn trên đồ thị có ảnh hưởng lớn tới chi phí về thời gian thực hiện giải thuật. Cụ thể:

- Trường hợp ta biểu diễn bằng danh sách kề thuật toán BFS có độ phức tạp là $O(|V| + |E|) = O(max(|V|, |E|))$. Ta mất $O(V)$ thời gian để khởi tạo khoảng cách d cho các đỉnh. Mỗi đỉnh được đến thăm nhiều nhất là một lần và được lấy ra khỏi hàng đợi nhiều nhất một lần (các đỉnh lấy ra khỏi hàng đợi sẽ không được đưa lại vào) .Đối với mỗi đỉnh được lấy ra khỏi hàng đợi, ta tìm các đỉnh kề với đỉnh đó và kiểm tra xem đỉnh đó đã được đánh dấu chưa. Trong quá trình này thuật toán sẽ đi qua mỗi cạnh tối đa là một lần. Do đó, quá trình này có độ phức tạp là $O(|V| + |E|)$. Vậy độ phức tạp của thuật toán BFS là $O(|V| + |E|) = O(max(|V|, |E|))$.

- Trường hợp biểu diễn bằng ma trận kề, với mỗi lần tìm đỉnh kề ta cần duyệt một hàng.Do đó độ phức tạp của thuật toán trong trường hợp này là $O(|V|^2)$

- Trường hợp biểu diễn đồ thị bằng danh sách cạnh, thao tác tìm những

đỉnh kề với một đỉnh sẽ dẫn tới việc duyệt toàn bộ danh sách cạnh, đây là cài đặt tệ nhất, thuật toán có độ phức tạp tính toán là $O(|V||E|)$

Chương 3

Tính toán song song cho BFS

Xử lý song song hoàn toàn không xa lạ trong cuộc sống của chúng ta. Từ quầy siêu thị tính tiền, mua vé vào công viên hay đường cao tốc có nhiều làn... Tất cả đều là mô hình song song, xảy ra đồng thời. Xử lý song song giúp chúng ta tiết kiệm thời gian và tiền bạc, chia nhỏ để xử lý nhanh hơn. Từ đó, nâng cao năng suất. Có những bài toán phức tạp thực tế đòi hỏi cao về tốc độ, đưa ra quyết định dựa trên dữ liệu lớn, cần xây dựng mô hình và tính toán trên máy tính như:

- Dự báo thời tiết
- Chuẩn đoán y khoa
- Kinh tế - tài chính (mua bán chứng khoán)
- ...

Thuật toán tìm kiếm theo chiều rộng có nhiều ứng dụng và cũng cần xử lý trên dữ liệu lớn, ví dụ như bài toán phân cụm các nhóm người dùng mạng xã hội với hàng tỷ tài khoản, xác định và phân loại các thực thể quan trọng, phân tích các hệ thống phức tạp như mạng lưới điện và internet...Những bài toán này có tầm quan trọng ngày càng tăng khiến cho thuật toán duyệt đồ thị càng ngày càng quan trọng và đòi hỏi tốc độ xử lý cao.

Trong phần này, chúng ta sẽ cùng áp dụng lập trình song song cho thuật toán BFS.

3.1 Ý tưởng

Gọi v_0 là đỉnh bắt đầu duyệt BFS.

Thuật toán song song cho BFS phải thỏa mãn tính chất của BFS, đó là tất cả các đỉnh cách v_0 một khoảng d phải được duyệt trước các đỉnh cách v_0 một khoảng $d+1$.

Gọi mức (level) của 1 đỉnh là khoảng cách từ đỉnh ban đầu đến đỉnh v_0.

Ý tưởng căn bản nhất là ta sẽ duyệt tuần tự đồ thị theo từng mức, và sẽ dùng một cấu trúc dữ liệu để chứa tập các đỉnh ở một mức. Tại mỗi mức, ta chia ra tập của mức đó thành các tập con để thuật toán chạy đồng thời trên các tập con đó và thêm các đỉnh kề của các đỉnh trong tập con đó thành một tập mới (gọi là tập các đỉnh sẽ xét ở bước sau).

Từ đó, ta có thuật toán sơ bộ như sau:

Input: Đồ thị $G = (V, E)$, điểm bắt đầu s

Output: $d[v]$ là cho biết khoảng cách từ s đến đỉnh v

- Thuật toán:

```
ParallelBFS(G=(V,E),s):
    d[v] ← ∞
    L ← 0, d[s] ← L
    F1={s} //F1 là tập chứa các đỉnh ở mức đang xét
    While !F1.empty():
        F2 ← {} //F2 là tập chứa các đỉnh kề với mức hiện tại
        processLevel(G,F1,F2,d)
        L ← L+1
    return d
```

Xử lý song song sẽ được xử lý ở trong hàm processLevel.

3.2 Xây dựng thuật toán song song BFS

3.2.1 Đặt vấn đề

Trước hết, ta cần xây dựng cấu trúc dữ liệu để xử lý các layer (tập các đỉnh ở từng mức), cụ thể phải thỏa mãn:

- Sắp xếp không có thứ tự các phần tử
- Hỗ trợ chạy song song hiệu quả (BFS phải đi qua tất cả các đỉnh trong cấu trúc dữ liệu đó, cấu trúc dữ liệu đó phải split và union hiệu quả để chạy song song)
- Các luồng đồng thời thêm phần tử vào được mà không gây xung đột

3.2.2 Cấu trúc dữ liệu bag

Bag (túi) là một cấu trúc dữ liệu thỏa mãn các yêu cầu đặt ra, cụ thể bag sẽ bao gồm các phương thức sau:

- create: Tạo một bag
- insert: Chèn một phần tử mới vào bag
- split: Chia 1 bag thành 2 bag có kích thước bằng nhau (nếu số phần tử của bag là chẵn) hoặc chênh nhau 1 phần tử (nếu số phần tử của bag là lẻ)
- union: Ghép 2 bag thành 1 bag

Bag được tạo thành từ các pennant - là một cây có 2^i node với i là số nguyên không âm. Node gốc chỉ có 1 con (giả sử là con trái) và là một cây nhị phân perfect.

Hình 3.1: Cấu trúc dữ liệu bag (phải) và phép union 2 pennant (trái)

Bag và pennant phải thỏa mãn các đặc điểm sau.

*** Đặc điểm:**

- Pennant có thể union và split với độ phức tạp O(1) bằng cách thay đổi con trỏ.
- Pennant chỉ union 2 pennant cùng kích thước.
- Bag là một mảng các con trỏ trỏ tới pennant. Ô thứ i trong mảng là null hoặc trỏ tới pennant có kích thước 2^i.

*** Các phép toán trên pennant:**

- PennantUnion(x,y):

```
y.right = x.left
x.left = y
return x
```

- PennantSplit(x):

```
y = x.left
x.left = y.right
y.right = NULL
return y
```

*** Các phép toán trên bag:**

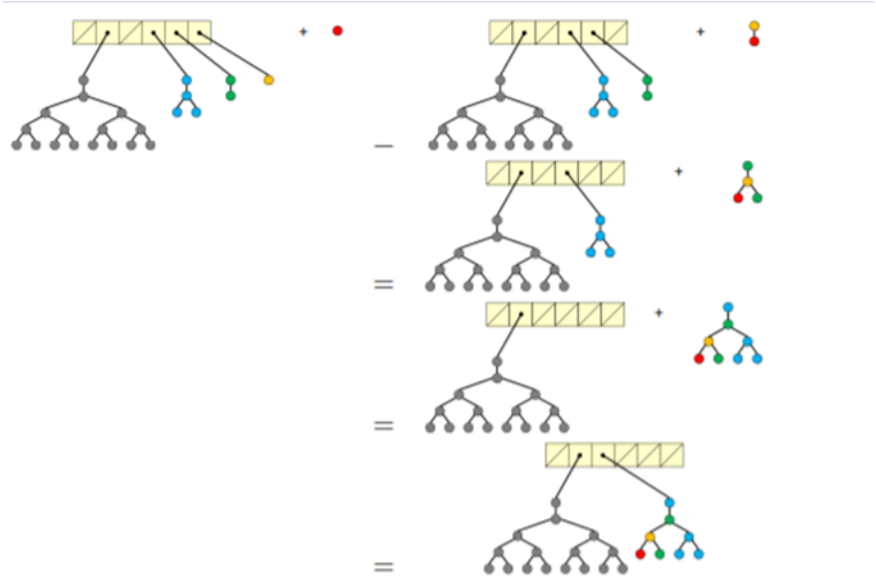
- Bag Insert:

Việc chèn một phần tử vào một bag giống như việc cộng một số nhị phân (ứng với bag hiện tại) với 1 (ứng với phần tử thêm vào). Bag insert mất thời gian O(logn) để thực hiện việc chèn một phần tử mới vào nó. (Một Bag cần tối đa [logn] vị trí để chứa n phần tử, giống như một số n khi biểu diễn sang nhị phân thì có độ dài tối đa là [logn]).

Thuật toán chèn phần tử x vào Bag S:

BagInsert(S,x):

```
k = 0
while S[k]!=null
    x = PennantUnion(S[k],x)
    S[k++] = null
S[k] = x
```



Hình 3.2: Phép chèn một phần tử vào Bag

- Bag Union:

Phép union hai bag sử dụng thuật toán giống như cộng hai số nhị phân. Để

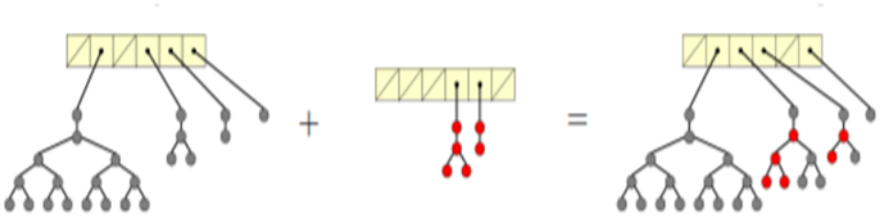
thực hiện bag union, đầu tiên chúng ta xem xét quá trình union 3 pennant thành 2 pennant. Cho 3 pennant x, y, z. Mỗi trong số chúng có kích thước 2^k hoặc rỗng, ta có thể union để tạo ra một cặp pennant (s, c), trong đó s có kích thước 2^k hoặc rỗng, c có kích thước 2^{k+1} hoặc rỗng. Gọi FA(x, y, z) là hàm ứng với bảng sau, trong đó 1 cho biết pennant có kích thước 2^k, 0 cho biết pennant đó là rỗng:

x	y	z	s	c
0	0	0	NULL	NULL
1	0	0	x	NULL
0	1	0	y	NULL
0	0	1	z	NULL
1	1	0	NULL	PENNANT-UNION(x,y)
1	0	1	NULL	PENNANT-UNION(x,z)
0	1	1	NULL	PENNANT-UNION(y,z)
1	1	1	x	PENNANT-UNION(y,z)

Hình 3.3: Bảng union 3 pennant thành 2 pennant

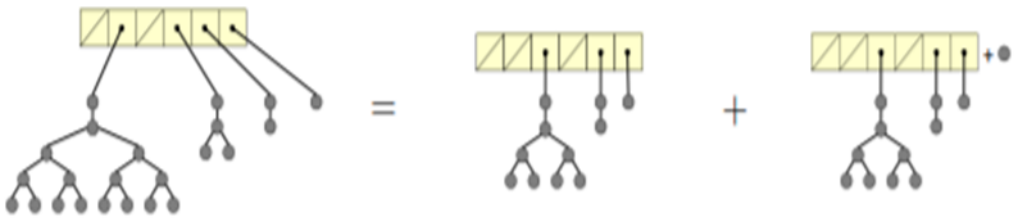
Sau khi có bảng trên tương ứng với hàm FA. Chúng ta có thuật toán union 2 bag như sau:
BagUnion(S1, S2):

```
y = NULL // biến đóng vai trò lưu trữ phần dư khi cộng bit
for k = 0 to r: // r là độ dài bag
    (S1[k], y) = FA(F1[k], F2[k], y)
```



Hình 3.4: Minh họa phép union 2 bag

- Bag split:
Phép Spit 1 bag thành 2 bag tương tự như phép dịch phải số nhị phân. Ta có minh họa như sau:



Hình 3.5: Minh họa phép Split bag

3.2.3 Thuật toán BFS song song

Sau khi đã thiết lập cấu trúc dữ liệu và các phương thức trên nó, ta có hàm processLevel sử dụng thuật toán song song như sau:

```
processLevel(G,F1,F2,d):
If size(F1) > C then:
    (A,B) <-- bagSplit(F1)
    spawn processLevel(G,A,F2,d)
    processLevel(G,B,F2,d)
    syns
Else:
    for v in F1 do
    parallel for (v,w) in E do:
        if d[w] = inf then:
            d[w] = l+1
            bagInsert(F2,w)
```

Trong thuật toán trên, nếu bag F1 đủ lớn (lớn hơn 1 số C nào đó) thì ta thực hiện phép chia bag đó thành 2 bag, sau đó xử lý song song gọi đệ quy tiếp hàm processLevel cho 2 bag nhỏ đó. Nếu bag đưa vào đủ nhỏ thì ta thực hiện phép duyệt tất cả các phần tử trong bag để tìm đỉnh kề với tập đó và thêm vào F2. [1]

[1]Souce code: *https://github.com/nducthang/BFSParallel*

Chương 4

Phân công công việc của các thành viên

Công việc đã thực hiện của các thành viên:

Nguyễn Đức Thắng:

- Code chương trình (tuần tự và song song) + viết báo cáo chương 3
- Làm slide và thuyết trình các phần tương ứng (trừ phần chứng minh tính đúng đắn và đánh giá độ phức tạp)

Nguyễn Anh Tuấn:

- Chứng minh tính đúng đắn và đánh giá độ phức tạp (Viết báo cáo + làm slide + thuyết trình)

Phạm Thị Thu Phương:

- Đưa ra bài toán, trình bày thuật toán, ứng dụng và đưa ra ví dụ. (Viết báo cáo các phần tương ứng)

KẾT LUẬN

Như vậy qua bài báo cáo này, chúng ta đã có một kiến thức tổng quan nhất về thuật toán BFS và một số ứng dụng của BFS. Thuật toán BFS và DFS là 2 thuật toán cơ bản của lý thuyết đồ thị, mỗi thuật toán có cách tổ chức dữ liệu khác nhau. Tùy vào từng bài toán cụ thể mà chúng ta hãy xem xét và lựa chọn thuật toán phù hợp nhất.

Nhóm sẽ cố gắng tiếp tục nghiên cứu, cải tiến giải thuât BFS, đặc biệt là cải tiến lập trình song song để tối ưu tốt hơn cho thuật toán. Chính vì vậy, nhóm rất mong nhận được những ý kiến đóng góp của TS.Đoàn Duy Trung và các độc giả để có thể cải tiến thuật toán tốt hơn nữa.

Nhóm xin chân thành cảm ơn!

Tài liệu tham khảo

[1] Charles E. Leiserson, Tao B. Schardl, *A Work-Efficient Parallel Breadth-First Search Algorithm*, Cambridge, MA 02139, 2010

[2] Lê Minh Hoàng, *Giải thuật và lập trình*, Đại học sư phạm Hà Nội, 1996-2006

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest *Introduction to Algorithms*, 3rd Edition (The MIT Press)